ORQUESTACIÓN AUTOMÁTICA DE SERVICIOS CONVERGENTES

JESÚS DAVID RAMÍREZ MEDINA

Tesis de Maestría en Ingeniería Telemática

Director:
Juan Carlos Corrales Muñoz
Doctor en Ciencias de la Computación

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Línea de Investigación en Aplicaciones y Servicios sobre Internet, y Servicios Avanzados de Telecomunicaciones

Popayán, Noviembre de 2015

JESÚS DAVID RAMÍREZ MEDINA

ORQUESTACIÓN AUTOMÁTICA DE SERVICIOS CONVERGENTES

Tesis presentada a la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad de Cauca para la obtención del Título de:

Magister en:
Ingeniería Telemática

Director
Juan Carlos Corrales Muñoz
Doctor en Ciencias de la Computación

Popayán
2015

Agradecimientos

Quiero expresar mis más sinceros agradecimientos al Dr. Juan Carlos Corrales, por su excelente dirección, por sus aportes tan importantes y por su apoyo durante todo este trabajo. A todos los integrantes del proyecto Telcomp2.0, los asesores Dr. Álvaro Rendón Gallón, Mg. Francisco Martínez, Dr. Armando Ordóñez y Mg. Cristhian Figueroa, por sus conocimientos y valiosos consejos. A mis compañeros de trabajo dentro del proyecto Mg. Julián Rojas, Mg. Julián Caicedo, Mg. Javier Suarez, Mg. Luis Rojas y Mg. Leandro Ordoñez, por todos sus aportes que complementaron mi trabajo. Finalmente agradezco los estudiantes de pregrado que trabajaron como soporte de este trabajo, los ingenieros Gustavo Enríquez y Andrés Benavides.

Quiero, además extender mis agradecimientos a COLCIENCIAS, la Universidad de Cauca y el Grupo de Ingeniería Telemática, por el apoyo financiero a través del programa Joven Investigador; y a la Universidad Politécnica de Madrid y el Dr. Juan Carlos Yelmo por permitirme realizar la pasantía de investigación.

Resumen Estructurado

Antecedentes: Una solución ampliamente adoptada para obtener un bajo Time to Market por parte de los operadores de Telecomunicaciones y empresas en Internet, es la utilización del concepto de composición de servicios ya que su filosofía es reutilizar componentes software previamente implementados. La composición tiene dos fases, la síntesis y la orquestación, siendo la segunda un reto para los servicios convergentes debido a que requiere amplio conocimiento técnico y experiencia. Actualmente existen propuestas que automatizan la orquestación adaptando tecnologías de la Web como BPEL, sin embargo lo hacen en tiempo de ejecución por lo tanto el rendimiento en la ejecución se ve afectado. Esta situación implica la búsqueda de nuevas técnicas de orquestación para servicios convergentes.

Objetivos: Proponer mecanismos que automaticen la orquestación de procesos abstractos en tiempo de diseño, para soportar la composición de servicios en entornos convergentes.

Métodos: Para cumplir el objetivo se plantearon cuatro fases, la primera correspondió a la selección de dos representaciones formales, una para procesos abstractos (la síntesis) y otra para servicios ejecutables (la orquestación); en la segunda fase fue utilizado un método para definir dos conjuntos de patrones específicos para componer servicios convergentes, uno del flujo de control y otro del flujo de datos; en la tercera fase se propusieron los algoritmos y el mecanismo que automatizan la orquestación a partir de un proceso abstracto y generan un servicio ejecutable en entornos JSLEE; finalmente en la quita fase fue realizada una evaluación del mecanismo definido, mediante la orquestación de un servicio convergente utilizando un prototipo implementado, y además se evaluó el desempeño del mecanismo.

Resultados: La presente propuesta entregó como resultados: la definición de los grafos y las redes de Petri coloreadas, para representar formalmente la síntesis y la orquestación de servicios convergentes respectivamente; un conjunto de 13 patrones de control y 16 patrones de datos, específicos para representar servicios convergentes; un mecanismo de orquestación automática basado en un algoritmo que utiliza los patrones y las representaciones formales definidas; una evaluación experimental que demostró el correcto funcionamiento del mecanismo según el criterio de 12 evaluadores expertos, y además valores de desempeño del mecanismo en tiempos de procesamiento, uso de memoria y porcentaje de CPU.

Conclusiones: la evaluación experimental realizada demostró un correcto funcionamiento del mecanismo de orquestación automática. Un servicio de prueba puede ser orquestado en alrededor de 150ms y simultáneamente puede orquestar hasta 4 servicios, este valor se puede mejorar implementando un conexión directa al repositorio de servicios, el cual limita su número de conexiones y por lo tanto afecta al mecanismo. No obstante el mecanismo brinda ventajas a los operadores de telecomunicaciones , ya que este permite generar servicio convergentes en muy poco tiempo y de una manera fácil a partir de un proceso abstracto, evitándoles la difícil tarea de orquestar manualmente servicios sobre entornos JSLEE, lo cual implica un amplio conocimiento técnico y experiencia, en instalación, configuración y desarrollo.

Palabras Clave: Composición de servicios, servicios convergentes, orquestación Automática, JAIN SLEE, patrones de control, patrones de datos.

Structured Abstract

Background: A widely adopted solution in order to obtain a low Time to Market by a segment of Telecommunication operators is the use of the concept of service composition because their philosophy is to reuse software components previously implemented. The composition has two phases, the synthesis and orchestration, the second being a challenge to converged services because it requires extensive technical knowledge and experience. Currently, there are proposals that automate the orchestration adapting Web technologies like BPEL, however they perform the orchestration at runtime so the execution performance is affected. This involves finding new orchestration techniques for converged services.

Aims: To propose mechanisms that automate the orchestration of abstract processes at design time, to support the services composition in converged environments.

Methods: To meet the objective were defined four phases, the first corresponded to the selection of two formal representations, one for abstract processes (synthesis) and the other for executable services (orchestration). In the second phase it was used a method for defining two sets of specific patterns to compose converged services, one for control flow and the other for data flow. In the third phase were proposed the mechanism and algorithms that automate the orchestration from an abstract process, and generate an executable service over JSLEE environment. Finally in the fifth phase was conducted an evaluation of the mechanism defined, through the orchestration of a converged service using the implemented prototype, besides the performance of the mechanism was evaluated.

Results: This proposal presents the following results: the definition of graphs and colored Petri nets, to formally represent the synthesis and the orchestration of converged services respectively. A set of 13 control patterns and 16 data patterns to represent specifically converged services. An automatic orchestration mechanism based on an algorithm that uses the patterns and formal representations defined. An experimental evaluation showed the proper functioning of the mechanism at the discretion of expert referees, and also performance values of the mechanism in processing times, memory usage and CPU load.

Conclusions: The experimental assessment proved correct functioning of the automatic orchestration mechanism. A service can be orchestrated at around 150ms and it is possible to orchestrate simultaneously up to 4 services, this value can be improved by implementing a direct connection to the service repository, which limits the number of connections and therefore affects the mechanism. However the mechanism provides benefits to telecom operators, allowing them to generate convergent services very quick and easily from an abstract process, avoiding the difficult task of manually orchestrating services on JSLEE environments, which implies extensive technical knowledge and experience about installation, configuration and development.

Keywords: Service composition, converged services, Automatic orchestration, JAIN SLEE, control-flow patterns, data-flow patterns.

Tabla de contenido

Tabla de figuras

Lista de Tablas

xx

Capítulo 1

1. Introducción

1.1. Contexto general

Actualmente los avances en las Tecnologías de la Información y la Comunicación (TIC), han generado en los usuarios una creciente demanda de nuevos servicios más personalizados y con mayor funcionalidad [1]. Para satisfacer esta demanda los operadores de telecomunicaciones han modificado sus modelos de negocio, siguiendo tendencias que incorporan las tecnologías de la Web. Una de estas tendencias es conocida como Telco 2.0 [2], la cual pretende integrar los conceptos, tecnologías y servicios del dominio de la Web, con los servicios tradicionales del dominio de las telecomunicaciones (servicios Telco). De esta manera, es posible que los operadores amplíen su portafolio de servicios, y pasen de ofrecer servicios tradicionales Web o Telco, a ofrecer servicios convergentes, que se definen como: la coordinación de un conjunto de servicios Web y Telco proporcionados mediante diferentes tipos de redes y protocolos, a través de distintos terminales [3][4].

No obstante, existen dos factores que los operadores de telecomunicaciones deben tener en cuenta para desarrollar servicios convergentes. Primero, el alto rendimiento que los servicios convergentes, al igual que los servicios Telco, requieren para ser ofrecidos en tiempo real y con un alto grado de disponibilidad [5]; y segundo, el bajo Time to market, es decir el tiempo que tarda en ser desarrollado un servicio, desde que es planeado hasta que es puesto en marcha y ofrecido a la venta [6].

Por una parte, una solución para mantener un alto rendimiento, es la adquisición de plataformas para despliegue de servicios (SDP), que permitan crear nuevos

servicios convergentes con la calidad que un operador de telecomunicaciones requiere. Estas plataformas se basan en tecnologías como: SIP Servlets, JAIN SLEE, SCIM (Gestor de interacción entre capacidades de servicios), entre otras [7], aunque actualmente la más utilizada para dicho fin es la especificación JAIN SLEE (*Java APIs for Integrated Networks Service Logic Execution Environment)* [8], ya que esta propone un entorno de alto rendimiento caracterizado por una alta disponibilidad, baja latencia, estar orientado a eventos, basado en componentes, iteraciones asíncronas, y además, permite la abstracción de la red sobre múltiples protocolos como HTTP, SIP, SS7, MEGACO, entre otros.

Por otra parte, una solución ampliamente adoptada para obtener un bajo Time to market, es utilizar la composición de servicios [9], la cual es definida como: la coordinación de múltiples servicios de manera que su interacción está encaminada a satisfacer un fin común [10]. La composición permite disminuir el Time to market, ya que su filosofía está basada en reutilizar servicios previamente implementados, de esta manera evita emplear largo tiempo implementando todo el servicio compuesto, y solo se enfoca en definir las interacciones entre los servicios [11].

Por lo tanto, para el desarrollo rápido de servicios convergentes con alto rendimiento y un bajo Time to market, se puede utilizar la composición de servicios sobre plataformas o entornos convergentes como JAIN SLEE.

Para realizar la composición de servicios, es necesario tener en cuenta su estructura y sus fases. La estructura contiene los siguientes elementos principales: servicios componentes[1], flujo de control y flujo de datos [12]. Las fases de la composición son dos, la Síntesis y la Orquestación [13][14]. En la síntesis es generado un plan que combina las funcionalidades de los múltiples servicios componentes definiendo de esta manera el comportamiento del servicio compuesto [13]. El resultado de la síntesis se puede definir como un proceso abstracto, ya que solo define el flujo de control parcial y no define el flujo de datos, por lo tanto no es un servicio ejecutable. Para realizar la síntesis, es posible utilizar técnicas automáticas empleadas en el dominio de la Web, como los planeadores de inteligencia artificial o la herramientas basadas en semántica, propuestas en

[1] En la presente monografía se utilizara el término Servicios Componentes para referirse a Recursos según [12], ya que representan el mismo concepto, pero en la composición es más utilizado Servicios Componentes.

[13][14][15], puesto que los procesos abstractos al no ser ejecutables, no dependen del entorno donde es ejecutado el servicio. Por otro lado, la orquestación se refiere al proceso de definir detalladamente el flujo de datos y de control entre los servicios, utilizando algún lenguaje estándar de tal manera que el resultado sea un servicio ejecutable [10][14]. Para llevar a cabo la orquestación de servicios convergentes, es importante tener en cuenta que esta depende del entorno donde es ejecutado el servicio, y por lo tanto tiene un alto nivel de complejidad, ya que para entornos como JAIN SLEE es requerido conocimiento a nivel técnico sobre diferentes protocolos, lenguaje y el formato de la lógica del servicio.

1.2. Escenarios de motivación

Uno de los escenarios de motivación se presenta en los operadores de telecomunicaciones, ya que en los últimos años han surgido nuevos operadores que entran a competir en el mercado. Esta situación obliga a cada operador buscar ventajas competitivas, las cuales generalmente están basadas en la oferta de servicios novedosos [2]. Dichos servicios requieren de un largo proceso de desarrollo, puesto que generalmente las plataformas tecnológicas de los operadores abarcan tecnologías basadas en protocolos complejos. Por consiguiente, algunas organizaciones como OpenCloud o JNetX han propuesto nuevas plataformas que facilitan el desarrollo de servicios, brindado la posibilidad utilizar la composición y agregar servicios de la Web. No obstante, dichas plataformas a pesar de contar con interfaces graficas que permiten la composición, implican conocimiento técnico amplio sobre instalación, configuración y desarrollo. En este sentido, los operadores desarrollan sus nuevos servicios realizando la orquestación de manera manual [16][17].

Otro escenario de motivación se puede observar en la Web, donde existe gran cantidad de organizaciones que ofrecen diferentes servicios de información, y además tecnologías que permiten componer dichos servicios, tales como Yahoo Pipes y Google Mashups, los cuales están basados en motores de Mashups [18]; otras más tradicionales utilizan tecnologías como BPEL (Business Process Execution Language) y XPDL (XML Process Definition Language) para la composición [19]. Dichas tecnologías son bastante amigables y robustas, además dado que actualmente existe la tendencia a utilizar servicios de comunicación, estas

tecnologías permiten invocar a algunos servicios o API de comunicación disponibles como llamadas o mensajes de texto SMS. Un inconveniente es que usualmente los servicios de comunicación no son gratuitos lo que disminuye su utilización en la Web, además un aspecto aún más importante es que al utilizarlos, la composición es ejecutada en algún entorno Web, lo cual no permite que la gestión de la calidad de estos servicios sea llevada a cabo con alta disponibilidad y en tiempo real [5], características que los servicios de comunicación o servicios Telco obtienen al ser prestados en un entorno específico para telecomunicaciones. De esta manera, en la Web la orquestación de servicios no se adapta a los requerimientos de las telecomunicaciones[20].

1.3. Definición del problema

Ante la complejidad que implica la orquestación de servicios convergentes, ya que para entornos como JAIN SLEE es requerido conocimiento a nivel técnico sobre diferentes protocolos, lenguaje y el formato de la lógica del servicio, algunos trabajos como [21][22] buscan facilitar la orquestación adaptando técnicas del dominio de la Web, en las cuales utilizan reconocidos lenguajes como BPEL para definir la orquestación, e integran motores BPEL al entorno convergente para ejecutarla. Sin embargo, esto implica desarrollar un módulo intermedio que adapte la orquestación de BPEL al lenguaje y formato del entorno convergente; además, dicha adaptación se realiza durante la ejecución del servicio, es decir cuando el servicio ya ha sido desplegado y es ejecutado por el usuario, por lo tanto en este tipo de trabajos no se considera el alto rendimiento en la ejecución [23]. Otros trabajos como [24] también definen la orquestación con BPEL, aunque por el contrario, el módulo de adaptación se utiliza en tiempo de diseño (previamente a ser desplegado el servicio), de manera que no afectan el rendimiento; no obstante, la orquestación debe llevarse a cabo mediante técnicas manuales, lo que requiere conocimiento avanzado sobre diferentes protocolos y eventos asíncronos del dominio de las telecomunicaciones, para lo que BPEL no es muy adecuado [7]. En conclusión, aún no está estandarizada la forma de realizar la orquestación sobre entornos convergentes, y las aproximaciones al respecto muestran algunas desventajas.

En este orden de ideas, la presente propuesta de investigación está enfocada en la fase de orquestación para servicios convergentes, y pretende resolver la siguiente

pregunta de investigación: *¿Cómo realizar la orquestación automática de un proceso abstracto en un entorno convergente, considerando el rendimiento en la ejecución?*

1.4. Alcance

El objetivo principal de este trabajo es soportar la composición de servicios en entornos convergentes mediante la definición de mecanismos que automaticen la orquestación de procesos abstractos sobre entornos convergentes en tiempo de diseño.

Para cumplir dicho objetivo fueron llevadas a cabo las cuatro fases que se observan en la Figura 1.1, donde la primera correspondió a la selección de dos representaciones formales, una para procesos abstractos (la síntesis) y otra para servicios ejecutables (la orquestación). En la segunda fase fueron definidos dos conjuntos de patrones específicos para componer servicios convergentes, uno del flujo de control y otro del flujo de datos. En la tercera fase se describen los algoritmos y el mecanismo que automatizan la orquestación a partir de un proceso abstracto y generan un servicio ejecutable en entornos JSLEE. Finalmente en la cuarta fase es presentada la evaluación del mecanismo definido, mediante la orquestación de un servicio convergente utilizando un prototipo implementado, y además se evaluó el desempeño del mecanismo.

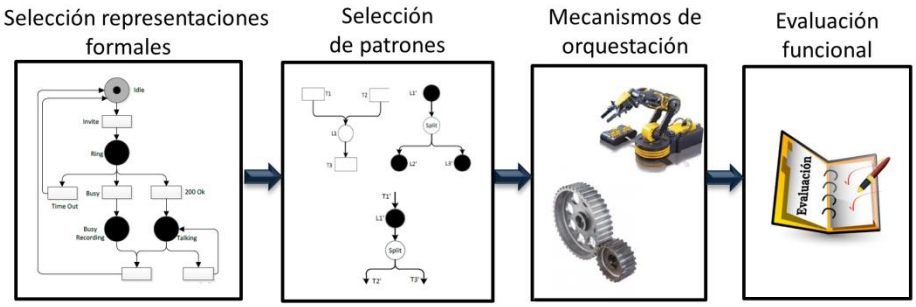


Figura 1.1. Fases para el desarrollo de este trabajo.

Por otra parte, es importante aclarar que el mecanismo de orquestación automática propuestos en este proyecto, se llevan a cabo en tiempo de diseño, es decir previamente a que el servicio sea desplegado y ejecutado por el usuario.

En cuanto a los servicios componentes utilizados durante la composición, y el entorno Web para generar procesos abstractos, se aclara que no están contemplados dentro del alcance, y por lo tanto se utilizaron desarrollos externos previamente implementados por terceros, en este caso proporcionados por el proyecto TelComp2.0 [25].

Adicionalmente, para la ejecución de los servicios convergentes, existen variedad de plataformas, algunas de ellas como Rhino, son propietarias y su uso requiere el pago de licencias. Sin embargo, Mobicents tiene una licencia de uso libre, por lo cual fue la única plataforma utilizada para desarrollar el prototipo del este proyecto.

Respecto a las representaciones formales, en este trabajo se realiza una selección de las representaciones adecuadas tanto para la síntesis como para la orquestación. Si bien la síntesis esta fuera del alcance de este trabajo ya que solo se centra en la orquestación, fue necesario evaluar la síntesis y definir su representación formal solo para establecer un puto de partida en la operación del mecanismo.

1.5. Contribuciones y resultados

El presente trabajo de investigación busca automatizar la orquestación de servicios convergentes en tiempo de diseño, generando de esta manera los siguientes aportes:

- Un análisis y selección de las representaciones formales para modelar tanto el proceso abstracto como la orquestación de los servicios convergentes.

- La selección de un conjunto de patrones del flujo de control y de datos, específicos para representar la estructura de los servicios convergentes.

- Un algoritmo para pasar de un procesos abstracto a una orquestación de servicios convergentes, basado en patrones del flujo de ejecución y representaciones formales.

- Un mecanismo para la orquestación automática en tiempo de diseño, de procesos abstractos sobre un entorno convergente JAIN SLEE.

- Un prototipo que implementa el mecanismo de orquestación automática, y un servicio de prueba.

- Resultados experimentales sobre la evaluación del mecanismo de orquestación automática propuesto.

- Soporte al proyecto TelComp2.0, el cual está enfocado en la recuperación y composición de componentes complejos para la creación de servicios Telco 2,0. Este proyecto es financiado por Colciencias y la universidad del Cauca

1.6. Estructura de la monografía

El presente trabajo de grado se estructura de la siguiente manera:

Capítulo 2: estudia los conceptos principales como la estructura y las fases de la composición, además las técnicas para llevarla a cabo y los trabajos relacionados con la temática de orquestación de servicios convergentes.

Capítulo 3: describe el proceso de selección de dos representaciones formales, una para procesos abstractos (la síntesis) y otra para servicios ejecutables (la orquestación).

Capítulo 4: presenta el método con el cual fueron definidos dos conjuntos de patrones específicos para componer servicios convergentes, uno del flujo de control y otro del flujo de datos.

Capítulo 5: describe el mecanismo y los algoritmos que automatizan la orquestación a partir de un proceso abstracto y generan un servicio ejecutable en entornos JSLEE.

Capítulo 6: presentan la evaluación del mecanismo definido, mediante la orquestación de un servicio convergente utilizando un prototipo implementado, y además presenta la evaluación de desempeño del mecanismo.

Capítulo 7: describe las conclusiones obtenidas con el desarrollo de este trabajo, los resultados y los trabajos a futuro.

Capítulo 2

2. Estado actual del conocimiento

La composición de servicios busca la coordinación de múltiples servicios, de manera que sus interacciones estén encaminadas a satisfacer una solicitud del usuario, la cual no se puede cumplir con un solo servicio [10]. En este sentido el desafío principal de la composición se centra en definir las interacciones entre servicios. Para comprender dichas interacciones, a continuación se describen los dos aspectos a tener en cuenta para realizar la composición: su estructura y sus fases. Además son descritas las técnicas utilizadas para llevar a cabo la composición. Finalmente son descritos los entornos convergentes, y cómo se realiza la composición de servicios en dichos entornos.

2.1. Base conceptual

2.1.1. Estructura de la Composición

Aunque actualmente no se tiene algún estándar que especifique claramente la estructura de la composición de servicios, la gran mayoría de trabajos al respecto han tomado como referencia [26], ya que propone que un servicio compuesto puede ser visto como un proceso de negocio o Workflow, y descrito mediante cinco perspectivas: servicios componentes, flujo de control, flujo de datos, excepciones, y presentación. Cada una de estas perspectivas es modelada mediante patrones, los cuales son estructuras genéricas recurrentes que representan una funcionalidad dentro de un proceso [27]. La principal ventaja de utilizar patrones es la reutilización, puesto que al implementar la funcionalidad de un patrón, este se puede reutilizar dentro de la composición, y así componer un nuevo servicio en menos tiempo.

En este trabajo de grado, solo se tendrán en cuenta las perspectivas para los servicios componentes, flujos de control y de datos, ya que son las más relevantes para describir el funcionamiento básico de un servicio convergente, y con las tres es suficiente para ejecutar el servicio. En este sentido, la estructura de un servicio compuesto tiene los siguientes elementos:

- **Servicios componentes**

Estos son más conocidos como servicios atómicos, los cuales dentro de su implementación no invocan o dependen de otros servicios. En el contexto de este trabajo corresponden a los servicios Web y Telco que se utilizan dentro de un servicio compuesto para implementar su funcionamiento.

- **Flujo de control**

Describe el orden en que se ejecutan los servicios atómicos, teniendo en cuenta divisiones, convergencia y condicionales en el flujo. En [28] se definen 43 patrones, los cuales están clasificados en las siguientes categorías:

- **Patrones básicos:** capturan los aspectos elementales del flujo de control (*ejemplo:* secuencia, división paralela).
- **Patrones de ramificación y sincronización avanzada:** utilizan conceptos de ramificación y sincronización para múltiples hilos de ejecución (*ejemplo:* selección y sincronización múltiple).
- **Patrones de múltiples instancias:** describen situaciones donde hay múltiples hilos de ejecución activos y múltiples instancias de un servicio (*ejemplo:* sincronización de múltiple instancia).
- **Patrones basados en estados:** muestran situaciones donde el flujo de ejecución depende de datos internos del proceso, o del estado de los servicios componentes (*ejemplo:* selección diferida).
- **Patrones de cancelación y de terminación forzada:** utilizan el concepto de cancelación de actividades, donde las instancias de los servicios son activadas o desactivadas (*ejemplo:* terminación implícita).
- **Patrones de iteración:** capturan un comportamiento repetitivo (*ejemplo:* ciclo estructurado).

- **Patrones de terminación:** abordan circunstancias donde un proceso se considera como completo (*ejemplo:* terminación explicita).
- **Patrones de activación:** se encargan de manejar las señales externas que pueden ser requeridas para iniciar ciertos servicios (*ejemplo:* disparador persistente).

Cada una de las categorías nombradas anteriormente tiene asociado un determinado número de patrones [28], los cuales son descritos y representados mediante redes de Petri, ya que estas permiten capturar diferentes comportamientos de los patrones como sincronismo, paralelismo, concurrencia entre otros.

➢ **Flujo de datos**

Describe la forma de transferir los datos entre los servicios atómicos, incluyendo transformaciones, variables persistentes y datos externos. En [29] se definen 40 patrones, los cuales están clasificados en las siguientes categorías:

- **Visibilidad de Datos**: describen la manera en que los datos pueden ser vistos por los diversos servicios de un proceso.
- **Interacción de Datos**: examinan las diversas formas en que los datos se transmiten entre los servicios de un proceso, y cómo las características de los servicios individuales pueden influir en la forma en que el tráfico de datos se produce. Hay distinción entre la comunicación de datos entre los servicios dentro de un proceso, como la interacción de datos de un servicio del proceso con el ambiente externo.
- **Transferencia de Datos:** consideran la manera en que la transferencia de datos se produce entre servicios de diferentes procesos. Estos patrones sirven como una extensión a los de interacción.
- **Enrutamiento de datos**: muestran el comportamiento de la transmisión de datos entre servicios que son disparados externamente. Relacionan lo datos con la perspectiva de control.

Por otra parte, es importante resaltar que en los últimos años han surgido lenguajes para describir la estructura de la composición de servicios, tales como BPEL, XPDL, WSFL (Web Services Flow Language) entre otros [30], los cuales implementan algunos de los patrones de control y datos mencionados anteriormente

[28][29]. Sin embargo, estos lenguajes han sido diseñados para la composición de servicios Web, pero no para la composición de servicios de Telecomunicaciones, por lo tanto aún no se ha definido claramente cuáles son los patrones de control y de datos que debería tener un lenguaje para la composición de servicios convergentes.

2.1.2. Fases de la composición

Para realizar la composición de servicios hay que definir por completo su estructura, lo cual por facilidad, en esta propuesta, es realizado en dos fases: la síntesis y la orquestación [13][14]. Estas dos fases, se diferencian por el nivel de detalle en la descripción de la estructura, ya que la síntesis la describe de forma abstracta y la orquestación de forma concreta. A continuación se define cada una de estas fases.

- **Síntesis de la composición**

La síntesis está enfocada en seleccionar los servicios componentes, y la forma de combinar sus funcionalidades con el fin de lograr un comportamiento del servicio compuesto que satisfaga la solicitud hecha por el usuario [13]. El resultado de la síntesis puede definirse como un proceso abstracto, el cual tiene las siguientes características [31]:

- Define los servicios componentes por su funcionalidad, pero sin un enlace al servicio exacto que lo implementa.
- Define un flujo de control parcial que indica el orden de ejecución de los servicios, aunque no contiene todos los elementos necesarios para controlar el servicio compuesto. El flujo de control es descrito mediante patrones o compuertas lógicas, pero solo descriptivas y sin implementación.
- Establece las dependencias de datos de entrada y salida de los servicios, pero no un flujo de datos a través de todas las interacciones del servicio compuesto.
- Esta descrito en algún lenguaje estándar como BPEL (Abstract Process[2]) [32] o alguna representación formal como grafos.

[2] El lenguaje BPEL permite describir servicios de dos maneras: Abstract y Concret, donde Abstract solo describe la composición como un proceso abstracto.

- Es un proceso centralizado, es decir que está visto desde la perspectiva de un solo actor del negocio.
- No es ejecutable.

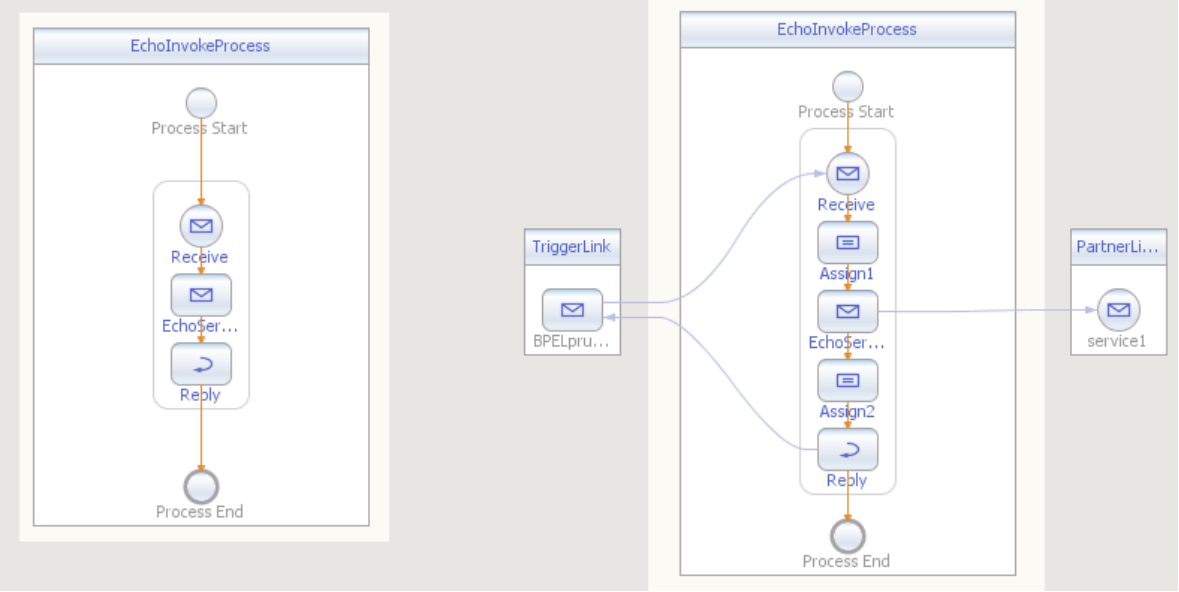
La Figura 2.1 a) muestra un ejemplo sencillo de un proceso abstracto descrito en BPEL Abstract, el cual recibe una información del usuario, la codifica y se la devuelve. Para implementarlo utiliza tres servicios componentes, el primero para recibir una petición externa del cliente con una información (*Receive*), el segundo para codificar dicha información (*EchoService*), y el tercero para enviar la información codificada al cliente (*Reply*) [33].

➢ **Orquestación de servicios**

La orquestación se refiere al proceso de definir detalladamente el flujo de datos y de control entre los servicios componentes utilizando algún lenguaje estándar, de tal manera que el resultado sea un servicio ejecutable por un entorno o motor de orquestación [10][14]. Es importante aclarar que la orquestación requiere una fase previa de síntesis. El servicio ejecutable producto de la orquestación, tiene las siguientes características [31]:

- Define los servicios componentes con un enlace al servicio exacto que lo implementa, bien sea a un repositorio local o externo.
- Específica por completo el flujo de control del servicio, para lo cual utiliza algunos de los patrones implementados previamente.
- Define mediante patrones un flujo de datos que describe el paso de datos entre servicios; además, en caso de presentar incompatibilidad entre salidas y entradas de servicios, describe las transformaciones de datos realizadas.
- Implementa la lógica adicional del servicio, como persistencia, temporizadores o comunicación con la aplicación cliente.
- Esta descrito en algún lenguaje estándar, como por ejemplo BPEL [32], que es entendible y ejecutable por algún motor de orquestación.
- Es un proceso centralizado, es decir que está visto desde la perspectiva de un solo actor del negocio. En otras palabras, existe un servicio en el que se define la orquestación, el cual invoca a los servicios componentes y procesa su respuesta.
- Es ejecutable.

La Figura 2.1 b) muestra la orquestación descrita en BPEL del ejemplo mostrado en la Figura 2.1 a). Como diferencia se observa que los tres servicios componentes tienen enlaces a los servicios Web que los implementan, además aparecen los bloques *Assing* que definen el flujo de datos [33].



a) b)

Figura 2.1. a) Ejemplo síntesis, b) Ejemplo orquestación [33].

Teniendo en cuenta los conceptos de síntesis y orquestación de servicios, es importante aclarar que este trabajo de grado solo está enfocada en la fase de orquestación, ya que en el contexto de la composición de servicios convergentes, es posible utilizar soluciones efectivas de la Web para la fase de síntesis, tales como la inteligencia artificial o algoritmos basados en semántica [13][14][15], debido a que un proceso abstracto es independiente del entorno de ejecución. Por el contrario, la orquestación depende del entorno de ejecución, de manera que aún no es clara la forma de realizarla para entornos convergentes.

2.1.3. Técnicas para composición de servicios

Para llevar a cabo la composición de servicios existen diferentes técnicas, las cuales se pueden clasificar como: composición estática, dinámica, manual y automática. A continuación es definida cada una de ellas.

- **Composición estática:** se realiza durante el tiempo de diseño, es decir que la estructura de la composición es definida cuando el software es planeado, y posteriormente será ejecutada [34].

- **Composición dinámica:** es caracterizada por llevarse a cabo en tiempo de ejecución, es decir cuando el servicio está siendo ejecutado, de manera que se adapta a cambios inesperados o fallos que afecten el flujo de ejecución normal de un servicio y generar el flujo alternativo para garantizar su funcionamiento [34].

- **Composición manual:** se presenta cuando la estructura de la composición es definida por alguna persona, quien decide cuales son los servicios componentes y el flujo de ejecución que permiten la funcionalidad requerida. Este tipo de composición puede presentarse tanto en una composición estática como en una dinámica, sin embargo no es muy común en la dinámica [15].

- **Composición automática** se presenta cuando la estructura de la composición es definida por algún sistema inteligente o mecanismo que utiliza algoritmos para esto. Dichos sistemas crean la estructura a partir de algunos requerimientos definidos. Por otra parte, es importante resaltar que es tipo de composición puede estar inmerso tanto en una composición estática como en una dinámica, dependiendo si es realizada en tiempo de diseño o ejecución [15].

Teniendo en cuenta que este trabajo de grado se enfoca en la orquestación de servicios, la cual es una fase de la composición, es importante resaltar que los anteriores tipos de composición también aplican para la orquestación. En este sentido, se aclara que esta investigación está orientada hacia la orquestación automática en tiempo de diseño o estática, ya que como se mencionó en el planteamiento del problema, trabajos como [35] realizan la orquestación manual de servicios convergentes la cual es bastante compleja, y por otro lado trabajos como [22] realizan la orquestación dinámica de servicios convergentes, teniendo inconvenientes de bajo rendimiento en la ejecución.

2.1.4. Entornos convergentes

En general, la estructura de un servicio compuesto esta descrita por medio de la síntesis y la orquestación, no obstante, un factor a tener en cuenta, es que según el tipo de servicios componentes (Web, Telco o la combinación de ambos) existen tres tipos de servicios compuestos. El primero son los servicios compuestos Web, los cuales son ejecutados por motores, como por ejemplo motores BPEL o motores de mashups[3], en donde el rendimiento no es crucial; el segundo son los servicios compuestos Telco, para los que se utilizan plataformas VoIP de alto rendimiento, como las evaluadas en [36]; y el tercero son los servicios convergentes, los cuales al contener servicios Telco, también requieren un entorno de alto rendimiento. En este sentido, un entorno convergente debe cumplir con tres características: alto rendimiento, orientación a eventos y componentes, e integración con múltiples protocolos.

Actualmente existe una tendencia a utilizar plataformas de composición de mashups tales como [37][38], las cuales si bien son diseñadas para componer servicios Web, también pueden invocar servicios Telco que hayan sido exportados por operadores de telecomunicaciones, como API o servicios Web. No obstante, el rendimiento de los motores de mashups presenta eficiencia alta para gran cantidad de peticiones y balanceo de carga, mas no para procesamiento interno, además la integración con múltiples protocolos es posible pero se debe utilizar como intermediario el HTTP/REST, lo cual disminuye aún más el rendimiento [20]. Otro inconveniente es que la composición ejecutada en este tipo de entornos no permite que el aprovisionamiento del servicio, sea llevado a cabo con alta disponibilidad, seguridad y en tiempo real [39][20].

Teniendo en cuenta las anteriores limitantes, en este trabajo de grado se optó por utilizar un entorno convergente más acorde con los requerimientos Telco. En este sentido, existe una variedad de entornos convergentes tales como: Ericsson Converged Service Studio [40], Alcatel-Lucent uReach CSF (Converged Services Framework) [41], IMS (IP Multimedia Subsystem), SIP Servlets y JAIN SLEE [7]. sin embargo en esta propuesta son considerados los entornos basados en JAIN SLEE [8] para ejecutar la orquestación, ya que presentan las mejores características en rendimiento e interacción con múltiples protocolos [42], por lo que son ampliamente utilizados en trabajos de investigación y en la industria.

[3] Los Mashup de definen como una aplicación de Web que contiene una composición de servicios Web [18].

➢ **Entorno JAIN SLEE**

Un entorno JAIN SLEE, está basado en componentes software reutilizables llamados SBB (Service Building Block), donde se implementa la lógica de funcionamiento de los servicios. Por lo tanto la composición de servicios es realizada definiendo las interacciones entre los SBB. Por otra parte, la ejecución de cada SBB está basada en el envío y recepción de eventos, ya que el SBB ejecuta una lógica de acuerdo al evento que recibe. Dichos eventos pueden porvenir de diferentes fuentes: internamente desde el SLEE, externamente desde los adaptadores de recursos, o desde otros SBB. Los adaptadores de recursos (RA) son módulos que actúan como interfaz entre el SLEE, y recursos externos como servidores Web, softphones, browsers, servidores multimedia, entre otros. De esta manera, los RA trasforman un evento de un recurso externo a un evento entendible por un SBB. Generalmente, los RA están asociados a protocolos específicos, como por ejemplo el SIP RA o el HTTP RA.

La Figura 2.2 presenta la arquitectura de un entorno JAIN SLEE [43], en los bloques inferiores muestra las fuentes externas de eventos y los adaptadores de recursos RA, en la parte de la derecha en amarillo el modelo de componentes donde se encuentran los SBB, en el bloque gris del centro están las facilidades para manejar alarmas, perfiles y trazas, en la parte de la izquierda en verde están los módulos de gestión para eventos, SBB, RA, despliegue, etc.

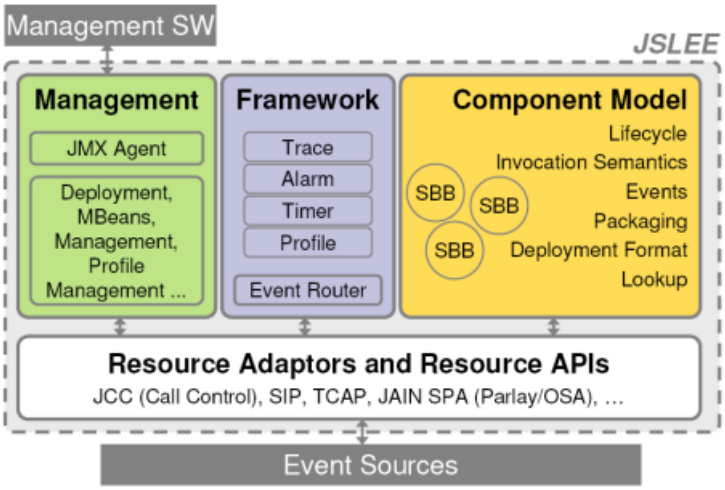


Figura 2.2. Arquitectura de entornos JAIN SLEE [43].

2.2. Trabajos relacionados

Los siguientes trabajos están relacionados con la composición de servicios, específicamente con la fase de orquestación de servicios convergentes para entornos JAIN SLEE. Para un análisis más claro, los trabajos principales están agrupados según el tiempo en que se lleva a cabo la Orquestación en el entorno convergente, es decir si es en tiempo de diseño o tiempo de ejecución. Adicionalmente, se tiene en cuenta si la orquestación es manual o automática; es decir, manual cuando el desarrollador debe definir la orquestación escribiendo algún código fuente o utilizando alguna herramienta grafica; y automática si es definida a partir de algún archivo que describa la síntesis de la composición, de tal manera que algún sistema software realiza la orquestación por sí solo. Posteriormente se analizarán otros trabajos que aunque no utilizan entornos JAIN SLEE, sus aportes son significativos dentro de este trabajo de grado.

2.2.1. Orquestación en tiempo de ejecución

La orquestación en tiempo de ejecución se caracteriza porque todas las tareas necesarias para definir la orquestación sobre el entorno convergente, son realizadas en el momento en que el servicio está siendo ejecutado por el entorno del operador y consumido por el usuario. A continuación son descritos los trabajos más relevantes.

En [44] proponen utilizar el lenguaje BPEL para describir la orquestación de servicios convergentes, la cual posteriormente, deberá ser adaptada al SLEE para su ejecución. Un factor a tener en cuenta, es que el SLEE no es capaz de ejecutar el lenguaje BPEL, por lo tanto para esta solución se requiere un motor BPEL que ejecute la orquestación, y un módulo que adapte dicho motor al SLEE, de tal manera que desde el motor BPEL se tenga control de todos los SBB, RA y demás componentes del SLEE. Para el módulo de adaptación mencionado, se propone incluir el motor BPEL dentro del SLEE, implementando un adaptador de recursos BPEL (BPEL RA), el cual aloja un contenedor Servlet que tiene desplegado el motor BPEL. De esta manera el BPEL RA puede comunicarse con todos los SBB activos, sin embargo la especificación JAIN SLEE no permite la comunicación entre dos RA, por lo tanto para comunicar el BPEL RA con otro RA, se implementan los SBB proxy como intermediarios, uno por cada RA.

Este trabajo logra fácilmente orquestación de servicios convergentes, sin embargo la comunicación con el BPEL RA disminuye el rendimiento en tiempo real del SLEE, ya que implica una constante conversión de XML del archivo BPEL a JAVA en tiempo de ejecución, además la utilización de varios SBB proxy duplica o triplica el número de eventos a procesar por el SLEE. Por otra parte, si bien el lenguaje BPEL es ampliamente utilizado en la industria para describir la orquestación de servicios web, en este trabajo no es clara la forma como modela la interacción con los servicios Telco y los descriptores utilizados para estos servicios. Otro aspecto a tener en cuenta, es que el diseño del BPEL RA no es genérico para cualquier motor BPEL, por lo tanto la implementación del adaptador está ligada fuertemente con el SLEE y el motor que se utiliza. Por otra parte, es importante resaltar que en esta propuesta el desarrollador tiene que definir la orquestación mediante una herramienta grafica BPEL, por lo tanto este trabajo presenta un tipo de orquestación manual de servicios convergentes.

Los autores en [45] toman como base la misma solución del trabajo anterior, sin embargo se enfocan en automatizar la orquestación del servicio. Para dicho fin utilizan nuevamente el lenguaje BPEL, no obstante, aquí solo es descrito el proceso abstracto y automáticamente es generada la orquestación. El mecanismo que permite dicha automatización está basado en semántica, para lo que emplean varias ontologías de dominio y un algoritmo de descubrimiento con el lenguaje OWL-S. Este trabajo aporta un mecanismo para orquestación automática de servicios convergentes, aunque tiene los mismos inconvenientes de rendimiento mencionados anteriormente.

En [46] es propuesta nuevamente la idea de utilizar el lenguaje BPEL para describir la orquestación de servicios convergentes, y posteriormente realizar una adaptación a un entorno JAIN SLEE para ser ejecutada. Por el contrario, en este trabajo la integración del motor BPEL al SLEE está basada en la especificación JBI (Java Business Integration) [47], la cual permite la integración de componentes que se comunican intercambiando mensajes por medio de un enrutador. En este sentido, los autores diseñan un entorno de ejecución JBI que tiene 2 componentes principales, el primero empaqueta el motor BPEL y el segundo empaqueta el SLEE, los cuales se comunican enviando mensajes por medio del enrutador de mensajes JBI. Algunos componentes adicionales son requeridos ya que el SLEE no es capaz

de procesar el mensaje JBI, por lo tanto implementan componentes para intercambio de mensajería que transforman un mensaje JBI a un evento del SLEE. Este trabajo provee una forma fácil para orquestación de servicios convergentes basada en una arquitectura estándar de integración en Java, no obstante la comunicación entre el motor BPEL y el SLEE implica una disminución del rendimiento, debido al tiempo adicional que toma el enrutador de mensajes JBI y el componente para la traducción de mensaje JBI a eventos del SLEE. Adicionalmente el desarrollador tiene que definir la orquestación mediante una herramienta grafica BPEL, por lo tanto este trabajo presenta un tipo de orquestación manual de servicios convergentes.

Otra aproximación como en [21] propone una plataforma completa para creación y ejecución de servicios convergentes (HSCEE) soportada en dos capas, la de creación y la de ejecución. En la capa de creación el desarrollador describe un proceso abstracto utilizando BPEL, y automáticamente es generada la orquestación utilizando algoritmos semánticos. Una ventaja que ofrece esta capa, es la reutilización de procesos BPEL existentes, de manera que al crear un nuevo proceso no es necesario partir desde cero. Por otro lado, la capa de ejecución implementa un ESB (Enterprise Service Bus) utilizando la especificación JBI, el cual está encargado de comunicar un motor BPEL con un SLEE. El motor BPEL está encargado de ejecutar la orquestación y realizar directamente las invocaciones a los servicios Web, y el SLEE solo ejecuta los servicios telco. Para realizar la comunicación entre el ESB y el SLEE, es implementado un adaptador de recursos ESB RA, que traduce mensajes basados en XML a eventos del SLEE. En este trabajo el rendimiento se ve afectado por la constante comunicación entre el motor BPEL, el ESB y el SLEE, la cual implica traducción XML a Java, además esta propuesta requiere de dos módulos intermediarios, el ESB y el ESB RA.

Por otra parte, en [48] se plantea una arquitectura para el desarrollo de servicios convergentes basada en la integración de un motor workflow JBPM y un entorno JAIN SLEE; la principal ventaja de esta integración es un rendimiento mejor que los trabajos anteriores, ya que evitan la constante conversión de XML a java. Dicha arquitectura contiene 3 modulos, el motor JBPM, el SLEE mobicents, y el servidor Jboss, los cuales son integrados fácilmente dado que el motor y el SLEE están implementados sobre Jboss; de esta manera, la comunicación entre los 3 módulos es realizada por medio de las API java que provee Jboss. Adicionalmente en esta

arquitectura es requerido un SBB de control, el cual actúa como intermediario entre todos los RA del SLEE y el motor JBPM. Este SBB crea una nueva instancia del proceso en el motor cuando un evento inicial llega al SLEE, y posteriormente ejecuta los métodos que el motor requiera en el proceso, creando y enrutando los eventos en el SLEE. Por otro lado, para describir la orquestación del servicio se utilizada una herramienta grafica basada en grafos, donde un usuario no experto podría diseñar el proceso; de esta manera el desarrollador es el encargado de implementar los métodos en java. El grafo del proceso generado es descrito con el lenguaje nativo del motor JPDL que está basado en java.

Este trabajo provee una forma fácil de crear servicios en entornos JAIN SLEE a partir de una herramienta basada en grafos, lo que da como resultado un proceso abstracto, sin embargo el desarrollador debe implementar gran parte de la lógica del servicio, por lo tanto este trabajo es considerado un tipo de orquestación manual. Por otra parte, a pesar de las mejoras en rendimiento debido a la utilización de JBPM, en [22] es realizado un análisis completo de la creación de un servicio con esta arquitectura, en el cual es posible observar una disminución de rendimiento por la integración del motor. Adicionalmente, es de resaltar que esta arquitectura está ligada al SLEE mobicents, y no funcionaría con otros entornos JAIN SLEE.

En [49] se expone un trabajo doctoral el cual integra varias características de los trabajos descritos anteriormente. La propuesta es utilizar un motor JBPM que ejecute la orquestación al igual que en [22], no obstante aquí el motor se ubica dentro de un contenedor de servlets que recibe peticiones via HTTP, similar a [44]. El principal aporte de este trabajo es que la orquestación no queda ligada al SLEE mobicents, y que el lenguaje de orquestación es java, lo cual agrega eficiencia, sin embargo tiene desventajas similares a [44] en cuanto al intercambio excesivo de mensajes entre el motor y el SLEE, y la traducción constante de cualquier evento al protocolo HTTP.

2.2.2. Orquestación en tiempo de diseño

La orquestación en tiempo de diseño se caracteriza porque todas las tareas necesarias para definir la orquestación sobre el entorno convergente, son realizadas previamente a la ejecución del servicio en el entorno del operador.

El trabajo más característico de este tipo de orquestación es el proyecto europeo TeamCom [24], el cual ofrece a los desarrolladores una forma fácil y rápida para crear servicios de telecomunicaciones, mediante el uso de una herramienta gráfica BPEL. Dicha herramienta utiliza un generador de código basado en templates[4], el cual traduce la descripción del servicio en BPEL a un servicio JAIN SLEE, para posteriormente desplegarlo. El generador de código analiza paso a paso cada uno de los elementos de la estructura del servicio descrita en BPEL, con el fin de generar los archivos necesarios en el servicio JAIN SLEE, tales como clases java y descriptores XML. El contenido de estos archivos es añadido mediante templates predefinidos y una tabla de correspondencias entre BPEL y JSLEE [24]. Por otro lado, para la orquestación del servicio están disponibles 8 bloques básicos llamados *bloques constructores de comunicación* (CBB), los cuales tienen la siguiente funcionalidad: *Entrada de datos, Salida de datos, Activación de datos, Archivos, Video, Texto, Conferencia y Audio*, de esta manera, cada servicio nuevo es una combinación de los 8 CBB. Posteriormente en [35] es analizado el modelo de SBB del servicio JSLEE generado, planteando las siguientes 4 posibilidades: un solo SBB que realiza todo, un SBB de control y varios SBB por cada actividad ejecutada en paralelo, un SBB con la orquestación y un SBB por cada CBB (modelo de orquestación), o un SBB por cada CBB y un SBB por cada actividad básica BPEL de manera que no existe ningún SBB de control (modelo de coreografía). Finalmente en [50] son evaluadas las 4 opciones y se concluye que lo mejor son las últimas dos (modelos de orquestación y de coreografía).

El proyecto TeamCom presenta un buen rendimiento en la ejecución debido a que su orquestación es en tiempo de diseño, sin embargo tiene inconvenientes respecto a la escalabilidad, puesto que solo permite utilizar 8 funciones básicas y no ofrece la posibilidad de integrar servicios de la Web. Otro inconveniente es la orquestación en BPEL, la cual debe ser realizada manualmente por el usuario.

Otro proyecto relevante es el entorno Umbra Designer [51], el cual provee una herramienta grafica para componer servicios de telecomunicaciones ejecutados sobre un entorno JSLEE. Umbra Designer se distribuye como un plugin del IDE Eclispe, y utiliza la tecnología EGL (Epsilon Generation Language) como generador de código basado en templates. El resultado del entorno es un proyecto Maven, el

[4] Los Templates son plantillas configurables que representan fragmentos de código fuente.

cual es cargado posteriormente en el entorno JSLEE. En este sentido, este proyecto contiene de forma implícita una tabla de equivalencias entre los elementos gráficos y los códigos JSLEE. Los servicios generados son representados con máquinas de estados que representan su orquestación. Para crear la máquina de estados dispone de transiciones (o eventos), estados y acciones, las transiciones son asociadas a funcionalidades de llamada, SMS, IVR[5], peticiones HTTP, Text to speech y reconocimiento de voz. Los estados son: inicial, final, simple, selección y compuesto, donde el compuesto permite invocar a una sub máquina de estados. Las acciones están asociadas a cada transición, vienen predefinidas y es posible añadir más mediante código java.

Entre las desventajas del entorno Umbra Designer está su disponibilidad, ya que este inicio como proyecto de investigación en [51], y actualmente es ofrecido bajo costo por ser propiedad de la organización Almira Labs [17]. Además, la orquestación a pesar de ser descrita gráficamente, debe ser realizada manualmente por el usuario lo que implica conocimiento técnico de máquinas de estados y de configuración del entorno Eclipse y Umbra Designer, adicionalmente si se requiere incluir servicios Web se debe codificar en java utilizando la transición HTTP. Otro inconveniente, es que el flujo de control es bastante limitado ya que solo provee patrones básicos de secuencia y selección múltiple.

Por otro lado, en [16] es presentado un entorno para orquestación y ejecución de servicios convergentes. Este entorno está compuesto por un servidor JSLEE llamado Rhino, y las herramientas graficas VSA (Visual Service Architect) y FSM Tools, las cuales permiten definir la orquestación mediante máquinas de estados, que posteriormente son codificadas como SBB. No obstante, dichas herramientas generan el esqueleto de la orquestación pero el desarrollador debe implementar la lógica de los eventos y métodos. Adicionalmente, todo este entorno es propietario y su uso solo está permitido bajo licencias académicas o adquiriendo el producto.

Otra aproximación es presentada en [52], donde implementan un entono JAIN SLEE llamado StarSLEE, el cual provee una herramienta grafica para la composición de servicios convergentes denominada StarSCE. Dicha herramienta permite generar

[5] IVR (Interactive Voice Response) permite en una llamada interactuar con el usuario mediante respuestas en DTMF por teclado.

procesos abstractos que automáticamente se traducen a servicios ejecutables, descritos con un lenguaje propio de este trabajo llamado StarSDL. Sin embargo el modelo de SBB generado no corresponde exactamente con una orquestación, ya que no hay un SBB de control y por el contrario los SBB se comunican entre sí, lo que requiere modificar por cada composición los SBB en función de sus eventos de entrada y salida. Adicionalmente, este entorno no está disponible para uso académico ni comercial.

En [53] es descrito un entorno para la ejecución de servicios convergentes llamado Mobi4D, en el cual fue escogido como elemento central el servidor JAIN SLEE mobicents debido a sus ventajas para las telecomunicaciones [54]. En este entorno, los RA están divididos en dos grupos, uno dedicado al acceso de los clientes y otro al acceso de los recursos o servicios básicos. Para cada uno de los RA es utilizado un SBB proxy, con el fin de tener independencia de protocolos en el SBB donde es definida la orquestación. Adicionalmente, en este trabajo implementan servicios con acceso móvil para validar el entorno, orientados al área de servicios de gobierno [55] y servicios de aprendizaje [56]. A pesar de las ventajas de este entorno, un inconveniente es que solo se enfocan en la ejecución, pero no en la composición, por lo tanto no indican como es llevada a cabo la orquestación de los servicios.

2.2.3. Orquestación sobre otros entonos convergentes

En esta sección son presentados algunos trabajos relacionados con la orquestación de servicios convergentes. Aunque dichos trabajos no están relacionados específicamente con entornos JAIN SLEE, son tenidos en cuenta debido a sus aportes.

Actualmente existe una tendencia por entornos que pretenden la orquestación de servicios convergentes basada en motores de mashups. Dicha tendencia surge como una evolución de tecnologías como BPEL, ya que también permite la orquestación de servicios web, aunque tiene mejoras en eficiencia y facilidad al utilizar API REST. Para lograr la convergencia con servicios Telco, es necesario que los operadores exporten sus servicios Telco como un API REST, con este propósito algunos operadores e intermediarios han publicado algunas API como Twilio y Tropo [4]. Este tipo de API generalmente tiene un costo por uso, puesto que la comunicación pasa por la red fija y móvil del operador. Teniendo en cuenta lo anterior, el proyecto

OMELETTE [57][37] plantea un entorno robusto para composición automática de mashups convergentes centrada en el usuario final, por lo cual provee una gran facilidad de manejo y además abarca la síntesis y la orquestación automática. Este proyecto tiene como núcleo de orquestación el motor de mashups llamado Node.js [20]. Otro proyecto importante que permite la orquestación de servicios convergentes basada en mashups es WSO2 [38], el cual provee un entorno completo para desarrollo de aplicaciones web en diferentes tecnologías, entre las cuales se destaca Jaggery.js [58] como su motor de mashups.

Como se había mencionado anteriormente, la orquestación de servicios convergentes basada en mashups tiene desventajas en el rendimiento y la calidad del servicio, con respecto a los requerimientos de los operadores [5], adicionalmente, es de resaltar que en los anteriores proyectos, las plataformas de ejecución de los servicios se encuentran externas a la infraestructura del operador, factor que no permite mantener una adecuada calidad el servicio.

Por otra parte, en [59] es presentado el proyecto europeo SPICE, en el cual implementan un entorno de creación de servicios enfocado a dos tipos de usuarios, el desarrollador y el usuario no experto. Para el desarrollador ofrece una herramienta gráfica que permite definir la orquestación mediante máquinas de estado, y para el usuario no experto brinda una herramienta basada en grafos que describe un proceso abstracto, el cual por medio de anotaciones semánticas y ontologías es orquestado automáticamente como una máquina de estados. El gran aporte de este trabajo es la definición de un lenguaje para describir específicamente la orquestación de servicios convergentes llamado SPATEL [60], modelado gráficamente como máquina de estados. Por otra parte, este trabajo propone que la orquestación generada se puede traducir hacia diferentes entornos, sin embargo las implementaciones mostradas solo utilizan motores BPEL que no satisfacen los requerimientos de las telecomunicaciones.

Otro proyecto europeo importante es OPUCE [61], el cual está enfocado en implementar un entorno para composición y ejecución de servicios centrado en el usuario final. Este entorno provee un editor gráfico amigable al usuario, que le permite crear procesos abstractos teniendo en cuenta información de su perfil [62]. Dichos procesos son orquestados automáticamente sobre BPEL utilizando

algoritmos semánticos basados en ontologías y el lenguaje OWL. Por otro parte, la ejecución está compuesta por un motor BPEL para ejecutar la orquestación, y varios servidores Web y JSLEE para los servicios básicos; estos servidores se comunican con el motor mediante una Gateway que enruta los eventos. A pesar de las ventajas de este trabajo, un inconveniente es el motor utilizado para ejecutar la orquestación, ya que no es apto para las telecomunicaciones.

En [63] es presentada una plataforma que permite la composición de servicios implementados con diferentes tecnologías como SIP, Web, AJAX, redes conmutadas, entre otras. Para interactuar con dichas tecnologías proponen los agentes para ejecución de composición (CEA), los cuales son capaces de ejecutar y controlar los protocolos subyacentes de cada tecnología. Por otro lado, la composición la define el usuario generando un proceso abstracto con una interfaz gráfica, dicho proceso es denominado esqueleto y solo indica las características de cada servicio componente, ya que en tiempo de ejecución son seleccionados los servicios indicados. Esta aproximación logra la orquestación automática de servicios convergentes, no obstante el motor de orquestación está basado en SIP Servlets y los servicios son escogidos en tiempo de ejecución, lo que no aporta un rendimiento óptimo.

Otra aproximación es [64], donde los autores definen un nuevo lenguaje específico para servicios de telecomunicaciones denominado TSDSL (Telecom Service Domain Language), el cual permite abstraer la complejidad de los protocolos telco. El diseño del servicio es realizado por medio de una interfaz gráfica que modela máquinas de estados, y posteriormente es utilizado un generador de código para orquestar automáticamente el servicio como SIP Servlets, siguiendo algunas reglas de mapeo. En este trabajo es posible identificar como aporte la definición de un lenguaje abstracto específico para servicios Telco, sin embargo la orquestación sobre SIP Servlets no presenta rendimiento óptimo [42].

2.3. Brechas del conocimiento

Teniendo en cuenta el planteamiento del problema y el anterior análisis de los trabajos relacionados, a continuación son descritas las brechas del conocimiento identificadas.

Enfoque	Brecha	Trabajos
Orquestación en tiempo de ejecución	Este tipo de orquestación siempre requiere algún módulo adicional de adaptación, ya que pretende que la orquestación sea ejecutada en un motor externo al entono convergente, y adaptada a este cuando el servicio es consumido por el usuario. Además, la adaptación implica mapeo de protocolos y mensajes al formato del motor. Dicho módulo intermedio genera una disminución del rendimiento en la ejecución.	[44][45] [46][21] [22]
	Desde la perspectiva del usuario que consume el servicio, esta orquestación adiciona latencia, por lo tanto el usuario podría percibir una baja calidad del servicio.	[44][45][46][21][22]
Orquestación en tiempo de diseño	Requiere un mecanismo para traducción del servicio hacia el formato específico del entorno convergente. No obstante aún no están definidos claramente los lenguajes o representaciones formales que se deben tomar como punto de partida en la traducción, generando diferentes propuestas que varían tanto en el lenguaje como en las reglas de correspondencia de la traducción.	[24][16] [52][53] [59][61] [63][64]
	Algunas organizaciones proveen entornos robustos que si bien permiten especificar por completo la orquestación de servicios JSLEE de manera gráfica, son enfocados a usuarios con conocimientos técnicos en protocolos, máquinas de estados, instalación y configuración de algún IDE. Además funcionan como aplicaciones en un solo equipo que no puede ser accedido remotamente.	[16][17] [51]
	A pesar de que esta orquestación soluciona los problemas de rendimiento en la ejecución, los trabajos de este tipo no han presentado ninguna evaluación del rendimiento de la fase de traducción. Esto es considerado una brecha, ya que sería un factor clave en	[24][16] [52][53] [59][61] [63][64]

	el caso que aplicar esta solución en arquitecturas que abarquen todas las fases de la composición de servicios.	
Orquestación sobre otros entornos convergentes	Si bien es cierto que existen diferentes entornos convergentes, los basados en JAIN SLEE presentan las mejores características para la ejecución de este tipo de servicios como la abstracción sobre múltiples protocolos, baja latencia y alto rendimiento [42]. Por lo tanto los motores basados en BPEL, JBPM, SIP Servlets o Mashups no son adecuados para ejecutar la orquestación de servicios convergentes con calidad de operador Telco [23][65].	[59][61] [63][64]
	El uso de motores de mashups implica la exportación de servicios Telco como servicios web o API, por parte del operador, lo cual es ideal si se quiere realizar la ejecución de los servicios de manera externa al operador, sin embargo esto disminuye la calidad del servicio, ya que la orquestación no es monitoreada ni respaldada por el operador, de manera que está sujeta a fallos, inseguridad o baja calidad [5][4][20].	[37][38] [58]
En general para la Orquestación de servicios convergentes	En la mayoría de trabajos relacionados han sido utilizados lenguajes como BPEL y JPDL para describir la estructura de la composición. No obstante, estos lenguajes han surgido para describir procesos de negocio y procesos Web, pero no para servicios de telecomunicaciones. Como desventajas de esto se observa que en la orquestación existen dificultades para: mantener la sesión en todo el proceso y entre los servicios componentes los cuales usualmente se comunican de forma asíncrona; y describir iteraciones con servicios componentes de protocolos diferentes a HTTP [7]. Por otra parte en la síntesis las desventajas de estos lenguajes son: el diseñador del proceso abstracto está ligado a un IDE lo que implica conocimientos de	[44][45][46][21][22][24][16][61]

	instalación y configuración; están basados en XML, el cual no es el lenguaje más eficiente para realizar procesamiento sobre este.	
	Ante la carencia de lenguajes específicos para describir servicios convergentes, algunos trabajos han propuesto nuevos lenguajes para dicha tarea. Un inconveniente radica en que estos lenguajes no son estandarizados por ninguna organización, por lo que podrían no ser utilizados. Por otra parte, la mayoría de lenguajes están basados en máquinas de estado, las cuales tiene falencias al modelar estados concurrentes, que son imprescindibles en las telecomunicaciones. Adicionalmente, estos nuevos lenguajes solo soportan patrones básicos del flujo de control y de datos, los cuales fueron escogidos sin criterios fundamentados.	[52][16] [59][63][64]
	Algunos trabajos permiten la orquestación de servicios sobre entornos convergentes, sin embargo este proceso es manual, de tal manera que el usuario debe tener conocimientos acerca de protocolos, complejas herramientas o algunos IDE's. Adicionalmente, la orquestación manual causa un aumento del tiempo total de la composición, lo cual es una gran desventaja para los operadores (incrementa el *time to market*).	[44][46] [22][24][16][53]

Tabla 2.1. Brechas del conocimiento.

2.4. Resumen

Este capítulo presentó una descripción del contexto tecnológico en el cual se desenvuelve el trabajo de grado, describiendo los principales conceptos como la composición de servicios, la estructura y las fases de la composición. Dentro de la estructura de la composición se definieron principalmente los servicios componentes, el flujo de control y el flujo de datos. Por otra parte fueron definidas la síntesis y la

orquestación como las fases de la composición, resaltando que en este trabajo se llevara a cabo la orquestación de servicios convergentes, utilizando específicamente los entornos de ejecución basados en JAIN SLEE.

Adicionalmente, en este capítulo fue realizada una revisión del estado actual de conocimiento sobre las técnicas para orquestación de servicios convergentes, clasificándolas en orquestación sobre entornos JAIN SLEE en tiempo de ejecución, y en tiempo de diseño, además se abordaron algunos trabajos sobre otro tipo de entornos que representan algún aporte a este trabajo.

Capítulo 3

3. Representaciones formales para composición de servicios

La composición de servicios es definida como la coordinación de múltiples servicios de manera que su interacción está encaminada a satisfacer un fin común [10]. En la actualidad es una poderosa técnica que permite la creación y el desarrollo de nuevos servicios complejos que satisfagan las necesidades de los usuarios, en tiempos muy cortos.

Generalmente, los trabajos que llevan a cabo procesos de composición de servicios, emplean mecanismos para describir de manera formal los servicios compuestos, esto facilita la aplicación de algoritmos y análisis matemáticos [34]. Existen diferentes modelos a través de los cuales es posible representar formalmente a los servicios compuestos, obteniendo una descripción de la estructura del servicio (flujos de datos y de control) con distintas características y niveles de abstracción.

En este capítulo se pretende definir dos tipos de representaciones formales para describir los servicios compuestos en cada una de sus fases, es decir una para los procesos abstractos (la síntesis) y otra para los servicios ejecutables (la orquestación). En este sentido, se realiza una descripción de los tres modelos más utilizados en la actualidad para la representación formal de servicios compuestos, según la revisión bibliográfica realizada, estos son: Grafos, Maquinas de Estados Finitos (FSM) y Redes de Petri (PN). Posteriormente, mediante una comparación teórica es seleccionada la representación más adecuada tanto para la síntesis como para la orquestación de servicios. Finalmente, en el Anexo A es presentado el modelado de algunos servicios convergentes realizado con el fin de verificar la selección de las representaciones formales.

3.1. Representaciones Formales

3.1.1. Grafos

Los grafos son una estructura de datos generalizada para la representación de objetos y conceptos. Formalmente un grafo en su forma básica es un par G = (N, E) donde N es un conjunto finito no vacío de elementos denominados nodos (también llamados vértices o puntos) tales que $N = \{n_1,\dots , n_m\}$; y E es un conjunto de pares (n_i, n_k) no ordenado de distintos elementos de N llamados aristas, tales que $E \subset N \times N$. Además N y E son distintos tal que $N \cap E = \varnothing$. Cuando todas las aristas tienen direcciones, y por tanto (n_i, n_k) y (n_k, n_i) pueden ser distinguidos, el grafo es dirigido. Así, como lo muestra la Figura 1 un grafo dirigido o dígrafo G = (N, E) consiste de un conjunto de N nodos y un conjunto de E aristas, las cuales son pares ordenados de elementos de N [66].

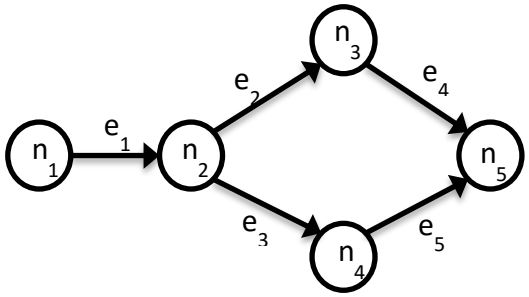


Figura 3.1. Ejemplos de grafos dirigidos

De la anterior definición, los nodos representan típicamente objetos, mientras que las aristas describen relaciones entre esos objetos. De esta manera, los grafos son muy apropiados para modelar objetos en términos de sus partes y relaciones entre estas, lo cual hace que sean muy atractivos en varios campos de aplicación como reconocimiento de patrones y visión computacional, similitud de objetos, etc. [67]. En ciencias de la computación, los grafos son usados para representar redes de comunicación, organización de datos, dispositivos de cómputo, flujos de ejecución, etc. Un ejemplo práctico es la estructura de enlaces de un sitio Web, el cual puede ser representado por un grafo dirigido. Los vértices hacen referencia a las páginas Web disponibles y las aristas dirigidas representan los diferentes enlaces entre esas páginas Web en el sitio. Otro ejemplo para el manejo de grafos se presenta específicamente en el desarrollo de algoritmos, lo cual es de gran interés en

informática y áreas afines, ya que a través de estos formalismos, es posible realizar la construcción, modificación y comparación de estructuras lógicas que representan procesos computacionales de forma dinámica.

Desde el punto de vista de la composición de servicios, los grafos son de gran importancia, dado que permiten modelar la estructura de un servicio compuesto con un alto nivel de abstracción, representando las relaciones entre sus servicios componentes, como el flujo de datos y el flujo de control. Aunque por sí sola, la expresividad provista por los grafos es limitada, es posible adicionar cierto poder expresivo mediante la anotación de sus nodos y aristas, asignando por ejemplo, atributos sobre los nodos del grafo, que permitan registrar el estado de ejecución del servicio compuesto. Cuando las etiquetas de las aristas pertenecen a un conjunto ordenado (por ejemplo, los números reales), se denomina *Grafo Ponderado*. Cuando es usado sin calificación, el término *Grafo Etiquetado* generalmente se refiere a un *Grafo con Vértices Etiquetados*, donde las etiquetas son distintas. Tal grafo puede ser equivalentemente etiquetado mediante enteros consecutivos {*1,..., n*}, donde *n* es el número de vértices en el grafo.

3.1.2. Redes de Petri

Una Red de Petri es un lenguaje formal y gráfico para modelar sistemas y procesos. Formalmente, una Red de Petri se define como una 5-tupla de la forma: $PN = (P, T, A, W, M_0)$, donde:

- $P = \{p_1, p_2, \ldots, p_m\}$ es un conjunto finito de lugares.
- $T = \{t_1, t_2, \ldots, t_n\}$ es un conjunto finito de transiciones.
- $A \subseteq (P \times T) \cup (T \times P)$ es el conjunto de arcos o aristas (relación de flujo).
- $W : A \rightarrow \{1,2,3,\ldots\}$ es una función de peso.
- $M_0 : P \rightarrow \{0,1,2,3,\ldots\}$ es el marcado inicial.
- Se cumple que: $P \cap T = \emptyset$ y $T \cap P = \emptyset$

Así, las Redes de Petri se componen de lugares, transiciones, arcos y marcadores o también denominados tokens. Los lugares representan los estados de la Red y estos a su vez pueden contener marcadores. Los arcos de entrada conectan lugares con transiciones, mientras que los arcos de salida inician en una transición y

terminan en un lugar. El estado en un momento dado, llamado marcado, está dado por el número de marcadores en cada lugar especificado. El marcado del sistema cambia cuando una transición es ejecutada, es decir cuando los marcadores son removidos de los lugares de entrada e insertados en los lugares de salida de una transición [67].

El comportamiento de muchos sistemas puede ser descrito en términos de sus estados y sus cambios. Para simular el comportamiento dinámico de un sistema, un estado o marcado en una Red de Petri cambia de acuerdo con las siguientes reglas de transición [68]:

- Se dice que una transición *t* está habilitada si cada lugar de entrada p_i de *t* está marcado con al menos *w(p, t)* marcadores, donde *w(p, t)* es el peso de la arista que va desde p_i hacia *t.*
- Una transición habilitada puede o no ser ejecutada dependiendo si el evento actual ocurre o no.
- La ejecución de una transición habilitada *t* remueve *w(p, t)* marcadores de cada uno de los lugares de entrada p_i de *t,* y añade *w(t, p)* marcadores a cada uno de los lugares de salida p_o de *t,* donde *w(t, p)* es el peso de la arista que va desde *t* hacia p_o.

Una transición sin ningún lugar de entrada es llamada *transición fuente* y una transición sin ningún lugar de salida es llamada *transición de disipación.* Se puede notar que una transición fuente está incondicionalmente habilitada y que ejecutar una transición de disipación consume marcadores pero no produce ninguno. Una pareja de un lugar *p* y una transición *t* es llamada auto-bucle si *p* es un lugar tanto de entrada como de salida de *t.* Se dice que una Red de Petri es pura si no contiene auto-bucles. Las anteriores reglas de transición se ilustran en la Figura 5, usando la conocida reacción química: $2H_2 + O_2 \rightarrow 2H_2O$. Dos marcadores en cada lugar de entrada en la Figura 2(a) indican que dos unidades de H_2 y O_2 están disponibles y la transición *t* está habilitada. Después de ejecutar *t,* el marcado cambiará al mostrado en la Figura 2(b), donde la transición deja de estar habilitada.

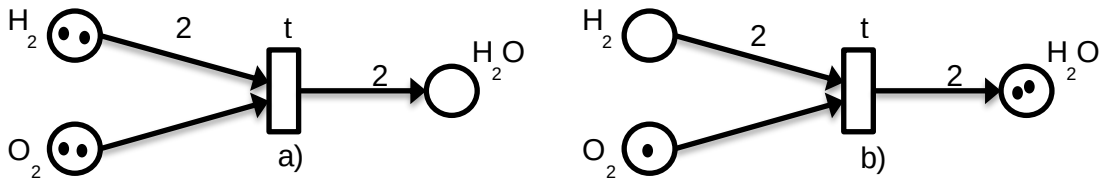


Figura 3.2. Ejemplos de Redes de Petri a) sin ejecutar b) ejecutada

Las Redes de Petri proporcionan una herramienta para describir sistemas caracterizados por ser concurrentes, asíncronos, distribuidos y no determinísticos. Desde un punto de vista gráfico, una Red de Petri puede ser utilizada como una ayuda de comunicación visual, de forma similar a las notaciones para diseño estructural del análisis de sistemas, y las metodologías de diseño tradicionales. El lenguaje de las Redes de Petri proporciona también una base matemática sólida para la descripción y análisis de ecuaciones de estado, modelos algebraicos y otros modelos matemáticos [67]. Sin embargo, una de las principales debilidades de las Redes de Petri es el problema de la complejidad, es decir que los modelos basados en Redes de Petri tienden a ser demasiado extensos para ser analizados, incluso para sistemas pequeños. En la aplicación de Redes de Petri, frecuentemente es necesario añadir modificaciones especiales o restricciones que se acomoden a la aplicación en particular. Dos áreas de aplicación exitosas son las evaluaciones de desempeño y los protocolos de comunicaciones. Otras áreas prometedoras incluyen el modelamiento y análisis de sistemas software distribuidos, sistemas de bases de datos distribuidos, programación concurrente, sistemas de eventos discretos, entre otros [68].

Desde el punto de vista de la composición de servicios, el lenguaje matemático de modelado que proveen las Redes de Petri define un formalismo que permite especificar y describir, tanto la estructura como el comportamiento de este tipo de servicios. La expresividad asociada a esta representación formal es mayor que la ofrecida por los formalismos de grafos y máquinas de estados finitos. No obstante, esta potencia expresiva compromete la ejecución de operaciones de análisis sobre un sistema modelado con esta representación formal, en términos de la complejidad computacional de los métodos empleados para implementar dichas operaciones. Este factor se convierte en un problema crítico dada la necesidad de una alta eficiencia en términos de tiempo dentro de los procesos que involucran el

descubrimiento, composición y reconfiguración de servicios compuestos en tiempo de ejecución.

3.1.3. Máquinas de Estados Finitos

Las máquinas de estados finitos (FSM por sus siglas en inglés) son un modelo matemático usado en el diseño de software y circuitos de lógica digital. Son concebidas como una máquina abstracta que puede estar en alguno de un número finito de estados, en un momento determinado, llamado estado actual, y puede cambiar de un estado a otro mediante la ocurrencia de un evento o cumplimiento de una condición específica, llamada transición. Las máquinas de estados finitos pueden ser usadas para modelar un gran número de problemas, entre los cuales se encuentran el diseño automático de circuitos electrónicos, diseño de protocolos de comunicaciones, flujo de ejecución de sistemas y otras aplicaciones de ingeniería. En el campo de la biología y la inteligencia artificial, las máquinas de estados son usadas para describir sistemas [69].

Formalmente, una FSM es una 5-tupla de la forma:(Σ, S, S_0, δ, F), donde:

- Σ es un conjunto de entradas
- S es un conjunto de estados
- S_0 es el estado inicial
- δ es la función de transición
- F es el conjunto de estados finales

Una FSM puede ser representada como un grafo con un único estado inicial, donde los nodos representan estados y las aristas representan transiciones entre dos estados. Las transiciones son anotadas con mensajes provenientes del conjunto de mensajes, especificando el tipo de transición [67]. En la Figura 3 se puede observar un ejemplo sencillo de una FSM, con eventos de entrada {a, b}, eventos de salida {u, v} y donde cada evento es considerado como una variable la cual está *presente* o *ausente.* Cada nodo elíptico representa un estado y cada arista representa una transición. La arista sin un estado de origen indica el estado inicial de la máquina de estados, es decir el estado α. Cada arista conecta un estado de origen con un estado de destino, representando una posible transición. La evaluación de un evento en un momento dado es verdadera o falsa cuando el evento está *presente* o *ausente*

respectivamente, y los operadores ¬, ∨ y ∧ corresponden a los operadores booleanos *not*, *or* y *and* respectivamente [69] .

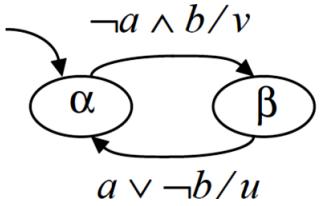


Figura 3.3. Ejemplo de una FSM

Desde el punto de vista de los servicios compuestos, las máquinas de estados finitos permiten especificar el flujo de ejecución de un servicio, sin embargo resulta difícil describir el flujo de control mediante su formalismo básico, ya que en una FSM el formalismo básico determina que en un momento dado una máquina solo puede presentar un único estado lo cual limita de forma considerable la representación de servicios compuestos, donde pueden existir flujos de ejecución concurrentes. Sin embargo, existen modelos que extienden el formalismo convencional de las FSM considerando aspectos adicionales, entre estos modelos se encuentra los Mapas de Estados Harel (Harel Statecharts) [70], este modelo presenta una extensión al formalismo convencional adicionándole las nociones de jerarquía, concurrencia y comunicación, dándole relevancia en la especificación y el diseño de sistemas complejos de eventos discretos, como sistemas computacionales de tiempo real múltiples, protocolos de comunicación y unidades de control digital.

Cabe aclarar que a pesar de las extensiones al modelo formal de las FSM, las cuales permiten modelar comportamientos y características que en principio no son consideradas en el modelo convencional, estas extensiones pueden representar una pérdida de formalidad del modelo debido a que están orientadas a aplicaciones específicas en contextos predeterminados [70].

3.2. Comparación y selección de representaciones formales

Teniendo en cuenta las características de las tres representaciones formales descritas anteriormente, en esta sección se plantea una comparación entre estas,

con el fin de seleccionar la representación más adecuada para los servicios convergentes, tanto para los procesos abstractos (la síntesis) como para los servicios ejecutables (la orquestación). En este sentido, se han elegido las características de cada fase de la composición (síntesis y orquestación) como los criterios de selección de la representación formal.

3.2.1. Selección de representación formal para la síntesis

A continuación son presentadas las características más relevantes de los procesos abstractos [31].

a) **Servicios abstractos:** cada servicio componente tiene una palabra que describe su funcionalidad, pero no un enlace al servicio que lo implementa.
b) **Flujo de control parcial:** define el orden de ejecución, pero abstrae algunos detalles como invocaciones o excepciones.
c) **Patrones de control abstractos:** cada patrón tiene solo un elemento que lo describe, pero no una estructura asociada.
d) **Dependencias de datos:** establece dependencias de datos entre las salidas de un servicio y entradas de otro servicio.
e) **No tiene flujo de datos:** no define variables, transformaciones de tipos de datos, o datos externos en el servicio.
f) **Centralizado:** el flujo de control y datos se implementa en un solo servidor central.
g) **No ejecutable:** no se puede ejecutar directamente en el servidor.

Características Proceso Abstracto	Grafos	FSM	PN
a) Servicios abstractos	2	2	2
b) Flujo de control parcial	2	1	1
c) Patrones de control abstractos	2	0	0
d) Dependencias de datos	1	1	1
e) No tiene flujo de datos	2	0	0
f) Centralizado	2	2	2
g) No ejecutable	2	0	0
TOTAL	**13**	**6**	**6**
Soportada (2), No soportada (0) y Condicionada (1)			

Tabla 3.1. Comparación de representaciones forales para la síntesis

En la Tabla 3.1, se puede observar la comparación entre las representaciones formales según las características de los procesos abstractos, y a continuación se discuten las características que presentan una marcada diferencia entre representaciones: b) está condicionada en FSM y PN ya que en estas se debe describir el flujo de control completo, y solo en algunos casos se pueden abstraer elementos; c) no está soportada en FSM y PN puesto que en estas los patrones requieren de cierta estructura para logran un comportamiento dado, y solo algunos patrones podrían modelarse con un único elemento abstracto; d) está condicionada en las representaciones, ya que en grafos las dependencias requerirían aristas adicionales y etiquetado sobre estas, en FSM y PN las dependencias se podrían marcar pero solo entre servicios contiguos; e) no está soportada en FSM y PN ya que estas deben describir el flujo de datos completo a través del servicio; finalmente g) no está soportada en FSM y PN puesto que existen herramientas que simulan y ejecutan lenguajes para este tipo de representaciones.

En conclusión, la representación formal que más se adecua a las características de un proceso abstracto son los grafos, ya que como muestran los resultados de la Tabla 2, los grafos obtienen un puntaje de 13 tomando todos los criterios con igual ponderación. La anterior evaluación es aplicada de manera general para los procesos abstractos, sin embargo en este trabajo será aplicada para los servicios convergentes. Es importante resaltar que las características propias de los servicios convergentes no son relevantes en la fase de síntesis, debido a la abstracción que se tiene en esta fase.

3.2.2. Selección de representación formal para la orquestación

A continuación son presentadas las características del servicio ejecutable (orquestación) [31].

a) **Servicios concretos:** cada servicio componente tiene un enlace al servicio que lo implementa.

b) **Flujo de control total:** define el orden de ejecución con todos los elementos necesarios para la ejecución.
c) **Patrones de control concretos:** cada patrón tiene una estructura asociada que describe su implementación.
d) **Flujo de datos:** establece el paso de datos entre cada servicio incluyendo transformaciones de datos y variables.
e) **Lógica adicional:** implementa lógica adicional del servicio, como persistencia, temporizadores o comunicación con la aplicación cliente.
f) **Lenguaje estándar:** es descrito en algún lenguaje estándar que es ejecutable por algún motor.
g) **Centralizado:** el flujo de control y datos se implementa en un solo servidor central.
h) **Ejecutable:** se ejecuta directamente en el servidor.

Un aspecto muy importante a tener en cuenta, es que las anteriores características son definidas de manera general para la orquestación de servicios, sin embargo este trabajo de grado está enmarcado en el contexto de los servicios convergentes, por lo tanto se deben evaluar algunas características adicionales que sean propias de los servicios convergentes. A continuación son descritas dichas características, tomando como referencia [71].

i) **Mantener la sesión:** los servicios Telco generalmente son invocados durante una sesión que guarda la información de los usuarios.
j) **Eventos:** en telecomunicaciones los servicios o funcionalidades son activadas mediante la recepción de eventos.
k) **Invocaciones implícitas:** un servicio componente puede invocar internamente a otros.
l) **Comunicación entre componentes:** los servicios Telco se pueden comunicar entre ellos sin pasar por el orquestador.
m) **Componibilidad:** cada servicio puede ser compuesto en otro más grade.
n) **Concurrencia:** varios servicios pueden ser ejecutados al mismo tiempo.
o) **Asincronia:** un evento puede invocar un servicio y seguir el flujo de control sin esperar respuesta.

Características Servicios Ejecutables en Entornos Convergentes	Grafos	FSM	PN
a) Servicios concretos	2	2	2
b) Flujo de control total	1	2	2
c) Patrones de control concretos	0	0	2
d) Flujo de datos	0	2	2
e) Lógica adicional	0	1	2
f) Lenguaje estándar	0	2	2
g) Centralizado	2	2	2
h) Ejecutable	0	2	2
i) Mantener la sesión	0	2	2
j) Eventos	1	2	2
k) Invocaciones implícitas	2	2	2
l) Comunicaciones entre componentes	2	2	2
m) Componibilidad	2	2	2
n) Concurrencia	2	0	2
o) Asincronia	1	0	2
TOTAL	**15**	**23**	**30**
Soportada (2), No soportada (0) y Condicionada (1)			

Tabla 3.2. Comparación de representaciones formales para la orquestación

En la Tabla 3.2, se presenta la comparación entre las tres representaciones según las características de los servicios ejecutables en entornos convergentes (la orquestación) [31]. Entre las características en que se marca una diferencia en las representaciones tenemos lo siguiente: b) está condicionada en grafos ya que patrones como disparadores externos o estados de ejecución del servicio serian difíciles de modelar, y tendría que utilizarse una gran cantidad de etiquetas para toda esta información; c) no está soportada en FSM ya que al tener un único estado no soportaría patrones de flujo paralelo, tampoco en grafos puesto que solo se podría etiquetar nodos pero la estructura no representaría el comportamiento de los patrones; d) no está soportada en grafos ya que no se podrán modelar las variables globales y las transformaciones de datos a través del servicio; e) no está soportada en grafos ya que en estos solo son descritas las relaciones entre servicios; en FSM está condicionada ya que se podrían soportan solo algunas de esas funciones; f) no está soportada en grafos debido a que no hay un lenguaje estándar, por el contrario para FSM existe ECharts y para PN existe PNML; finalmente h) no está soportada en grafos puesto que no existen herramientas estándar que simulan y ejecutan este tipo de representaciones. Por otra parte las en las características propias de los servicios

convergentes se tiene: i) no está soportada para los grafos pero si para FSM y PN, ya que se guarda información del estado y la respuesta a los eventos depende del estado; j) esta soportada directamente por FSM y PN, mas no en grafos aunque se podría definir las aristas como eventos de entrada y salida; k) y l) están soportados en todas puesto que los nodos, transiciones y eventos, pueden contener más acciones internamente y comunicarse con otros elementos; n) no está soportada en FSM ya la red se debe mantener en un único estado a la vez; o) está condicionada en grafos ya que no hay una notación específica pero se podría definir alguna, y en FSM no se soporta ya que los eventos asíncronos llevan a estados diferentes.

En conclusión, la representación formal que más se adecua a las características de los servicios ejecutables en entornos convergentes, son las redes de Petri, ya que como muestran los resultados de la Tabla 3, las Redes de Petri obtienen un puntaje de 30 tomando todos los criterios con igual ponderación.

3.2.3. Selección de tipo de Red de Petri para la orquestación

Actualmente, existe una gran cantidad de variantes y tipos de redes de Petri, por lo tanto es necesario seleccionar alguna en específico. Dentro los tipos de Redes de Petri, existe una división en dos grandes grupos, el primero, relacionado con las PN básicas y elementales; el segundo con PN de Alto Nivel (HPN, High PN) [72]. Las redes básicas corresponden a las primeras PN concebidas una vez fue definida la teoría de PN, estas fueron usadas para modelar sistemas simples, donde los datos y estados manipulados eran de fácil control y detección. Un ejemplo de estas son las conocidas redes de Lugar/Transición, donde el cambio de lugar se daba por algún evento o transición en particular. Sin embargo, los sistemas actuales contienen un conjunto de datos complejo para manipular y modelar con sistemas simples. En ese sentido, surgen las HPN, las cuales permiten adicionar, de manera adecuada y precisa, información importante del sistema dinámico. Este tipo de redes corresponden a aquellas que añaden información detallada de un sistema en particular. A diferencia de las redes elementales, las cuales son más adecuadas para modelos teóricos, las HPN son lenguajes con un mayor poder de modelamiento debido a la construcción de estructuras compactas y parametrizadas, que a través de la adición de tipos de datos y módulos estructurales, representan de mejor manera los sistemas dinámicos [72].

Las HPN son extensiones de las PN básicas, en consecuencia, es posible en teoría transformarlas a un comportamiento equivalente de una PN básica [73]. En este sentido, a continuación son descritos brevemente los principales tipos de HPN para representar sistemas concurrentes.

a) Redes de Petri Temporizadas

Este es un tipo de PN que consideran restricciones de tiempo asociadas a la ejecución, duración o retraso en una transición o arco de la PN. Cuando se considera un intervalo de tiempo para una transición se tiene una PN de tiempo (TPN, Time PN); cuando el tiempo está asociado a un arco, se tiene un PN temporizada (TdPN) [74].

b) Redes de Petri Jerárquicas

Este tipo de PN sigue el mecanismo de estructuración por módulos, similar al uso dado por los programadores. Esto se hace con el objetivo de reducir la complejidad de los sistemas. En la práctica es inviable modelar un sistema complejo en una sola PN, debido a la dificultad en el análisis de todo el sistema y sus detalles. Para resolver este inconveniente, se plantea la abstracción de módulos o cajas negras que componen el sistema total, de manera tal, que los diseñadores pueden desentenderse del detalle dentro del módulo y preocuparse por la interacción con otros módulos del sistema, mejorando considerablemente el análisis, compresión y síntesis de éste.

c) Redes de Petri de Objetos

Este tipo de PN tiene la finalidad de integrar el formalismo de las PN con los conceptos del paradigma de orientación a objetos [75]. De manera general, se puede destacar las siguientes categorías [76].

Objetos embebidos en PN: este enfoque se ve a toda la PN como una estructura de control, donde las marcas o tokens de la red son objetos, las transiciones como invocaciones a los métodos de los objetos, los arcos describen el flujo de los objetos por los lugares de la red, y los lugares representan las clases.

PN embebidas en objetos: en este enfoque el sistema es dividido en varios objetos, los cuales pueden comunicarse y trabajar entre sí, donde el comportamiento interno de cada uno de ellos está definido por una PN específica.

La gran ventaja de este tipo de redes es el uso del formalismo para la descripción de sistemas concurrentes y el reúso de componentes para la composición de sistemas más complejos.

d) Redes de Petri Coloreadas

Este es un tipo especial de PN, el cual añade información a la red definiendo un conjunto de tokens o marcadores especiales. Cada uno de estos tokens puede contener información diferente, implicando que sean procesados de acuerdo al dato contenido. Gráficamente, los tokens se representan con puntos de colores para identificar la diferencia en los datos que contienen. De manera general, las CPN son un lenguaje gráfico para modelado de eventos discretos que combinan las capacidades de las PN con las capacidades de lenguajes de programación de alto nivel [72].

A continuación se presenta una comparación entre los diferentes tipos de redes de Petri descritas anteriormente. Los criterios de selección a tener en cuenta para dicha comparación son:

a) **Manejo de tiempo:** permite realizar la sincronización de actividades o eventos para la comunicación entre los servicios de la orquestación.
b) **Flujo de datos:** cada uno de los servicios de la orquestación procesa una serie de datos para la obtención de alguna respuesta en particular. En este flujo de datos es posible añadir, eliminar o modificar algún dato específico, en consecuencia, se debe diferenciar los tipos de datos que maneja cada servicio.
c) **Facilidad de composición:** permite el acople entre los servicios de la orquestación. Para definir la estructura de ejecución a través de interacción entre servicios, se debe tener interfaces bien definidas y normalizadas.

d) ***Herramientas y* algoritmos:** permite la validación de la estructura del servicio convergente. Este punto hace posible que se realice el análisis de las propiedades dinámicas de las PN y de desempeño del sistema.
e) **Estandarización:** el modelado de la orquestación debe estar soportado por lenguajes estandarizados y ampliamente usados en la industria. Esto permitirá que la estructura de ejecución definida guarde coherencia con las tendencias internacionales.

Teniendo en cuenta lo anterior, la Tabla 3.3 ilustra la comparación entre las HPN consideradas. Para la evaluación general del cumplimiento de los criterios se hace uso de un sistema de ponderación, otorgándole un nivel de relevancia a cada criterio evaluado, en este caso se le da mayor ponderación al flujo de datos ya que dentro del servicio convergente se manipula una gran cantidad de datos diferentes, y a la facilidad de composición puesto que el trabajo de grado pretende componer servicios Telco y web; las herramientas y algoritmos también son importantes ya que se utilizaran para generar el servicio ejecutable. Posteriormente la calificación de cada criterio se multiplica por el ponderado y se suman las calificaciones para cada tipo de red.

Criterios	Ponderación	Temporizada	Jerárquicas	Orientada a objetos	Coloreada
a) Manejo de tiempo	2	2	0	0	0
b) Flujo de datos	5	1	1	1	2
c) Facilidad de composición	5	1	2	1	1
d) Herramientas y algoritmos	3	1	1	1	1
e)Estandarización	2	1	1	1	2
TOTAL		**19**	**21**	**15**	**22**

Tabla 3.3. Comparación entre diferentes redes de Petri

Como conclusión, las redes de Petri coloreadas son seleccionadas para la representación formal de servicios ejecutables (orquestación) en entornos convergentes, ya que brindan un mayor poder de modelamiento para el análisis y

diseño de este tipo se servicios, principalmente, por la capacidad para definir tokens con tipos de datos diferentes, y su relativa facilidad en la integración de PN independientes, lo cual facilita la composición de servicios.

3.3. Resumen

Este capítulo presentó una descripción de las principales representaciones formales utilizadas para la composición de servicios, los grafos, las máquinas de estados (FSM) y las redes de Petri (PN). Posteriormente fueron realizadas comparaciones entre las tres representaciones, con el fin de seleccionar la más adecuada tanto para describir los procesos abstractos (la síntesis), como para describir los servicios ejecutables (la orquestación). Como resultado de dichas comparaciones, fueron seleccionados los grafos para describir la síntesis y las redes de Petri para describir la orquestación. Adicionalmente, se describieron los diferentes tipos de redes de Petri, las cuales se clasifican en PN básicas y PN de alto nivel. Dentro de las de alto nivel se encuentran las PN temporizadas, jerárquicas, orientadas a objetos, y coloreadas, por lo tanto también fue realizada una comparación entre dichas PN, y como resultado se definió a las coloreadas para representar la orquestación de servicios convergentes.

Capítulo 4

4. Patrones del flujo de ejecución

La composición de servicios esta descrita por su estructura, en la cual se definen los flujos de control y de datos mediante patrones. De esta manera, en el mundo de la Web han surgido varios lenguajes que implementan algunos de estos patrones, como BPEL o JPDL, permitiendo describir la estructura de la composición de servicios Web. Sin embargo, en el contexto de los servicios convergentes aún no se han definido claramente cuáles son los patrones que deberían ser utilizados para describir los flujos de control y de datos.

En este capítulo se pretende definir un conjunto de patrones del flujo de control y flujo de datos, específicos para componer servicios convergentes. La Figura 4.1 muestra un esquema de cuatro fases, las cuales representan el proceso a seguir para definir los patrones. En la primera fase, se explica el proceso de elaboración de un catálogo de servicios presentes en un entorno convergente, con el fin de establecer un banco de servicios sobre los cuales puedan detectarse patrones del flujo de ejecución. La segunda fase, muestra el modelado con representaciones formales de los servicios del catálogo, lo cual permite homogenizar su descripción y aplicación algoritmos de detección. En la tercera fase, se utiliza un algoritmo para la detección de patrones del flujo de ejecución presentes en los servicios modelados. Finalmente, la cuarta fase, corresponde al análisis de los patrones detectados, y selección de la lista de patrones específicos para componer servicios convergentes.

A continuación son descritos los resultados obtenidos en cada una de las fases definidas anteriormente:

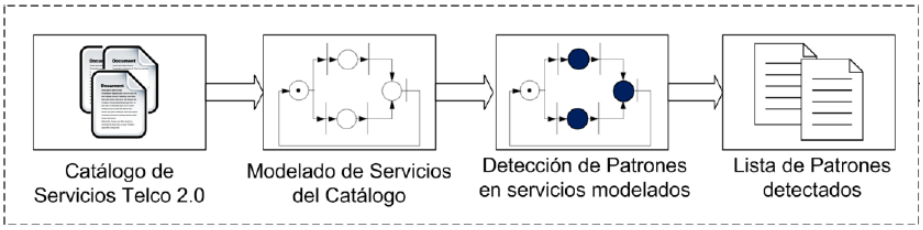


Figura 4.1. Fases para la definición de patrones.

4.1. Catálogo de servicios

Con el objetivo de establecer un banco de servicios presentes en un entorno convergentes, sobre los cuales puedan detectarse patrones del flujo de ejecución, en esta sección fue realizada una revisión bibliográfica de servicios, lo cual permitió conocer cuantos y cuales servicios están presentes en los entornos actuales, y además de que formas se pueden componer para crear nuevos servicios. Dicha revisión fue realizada sobre tres tipos de organizaciones, estándares como: ITU, 3GPP, ETSI-TISPAN, OneApi; algunas plataformas para el despliegue de servicios como: Mobicents, Rhino Slee, Ericsson SDP; y los operadores de telecomunicaciones como: Tigo, Claro Vodafone.

Posteriormente a la revisión bibliográfica, la cual se encuentra en el Anexo B, fue propuesta una clasificación para los servicios recopilados, con el fin de organizarlos en un catálogo de servicios, según algunas características en común.

4.1.1. Clasificación de servicios

Como se explicó anteriormente, un entorno convergente permite la integración de servicios Telco y servicios Web, de manera que se pueden obtener diferentes formas de integración, generando así servicios compuestos Web, servicios compuestos Telco o servicios convergentes. Por esta razón, en la revisión bibliográfica también se tuvo en cuenta la forma en que las organizaciones clasifican sus servicios, luego con base en estas clasificaciones, se propone un modelo para clasificar los servicios de un entorno convergente, presentado en la Figura 4.2.

A continuación son descritas cada uno de los tipos de servicios que componen el modelo propuesto.

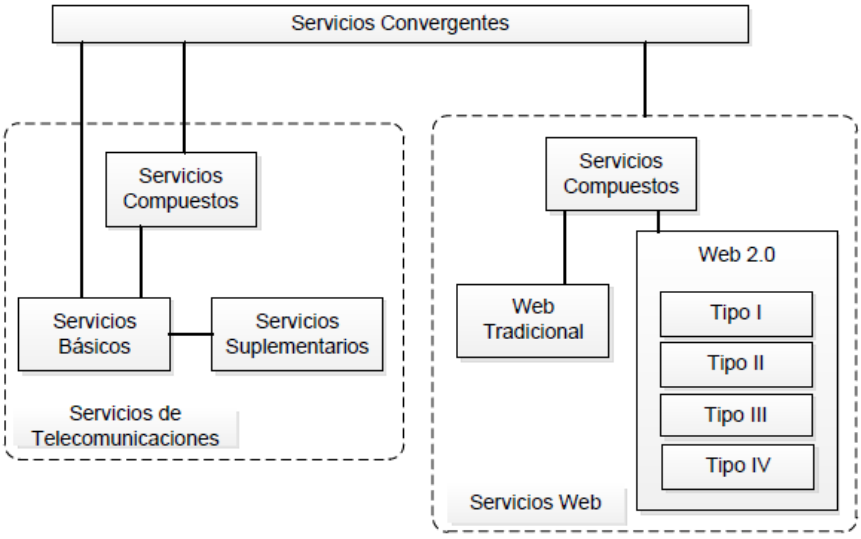


Figura 4.2. Modelo para clasificación de servicios en un entorno convergente.

- **Servicio Básico:** hacen referencia a los servicios tradicionales, es decir, aquellos que comúnmente son suministrados al usuario final por parte de un proveedor de telecomunicaciones [77]. Este tipo de servicio es también conocido como servicio final, ya que proporciona en sí mismo la funcionalidad completa (incluida la del equipo terminal) para la comunicación entre usuarios a través de una red fija o móvil de acuerdo a los protocolos establecidos entre los operadores. Un servicio básico constituye la base para los servicios suplementarios [78][79].

- **Servicio Suplementario:** tipo de servicio que modifica o complementa un servicio básico. Se encuentra soportado sobre un servicio básico, razón por la cual no puede ser ofrecido como un servicio único. Un mismo servicio suplementario puede ser aplicado a varios servicios básicos de telecomunicaciones [77]. Comprenden funciones como llamada abreviada, identificación de llamada entrante, conferencia entre varios usuarios, etc.

- **Servicio Web tradicional:** sistema software con interfaces y conexiones públicas definidas y descritas con lenguajes estándar como XML. Pueden ser utilizados por otros servicios a través de un Identificador de Recurso Uniforme (URI, Uniform Resource Identifier) e interactuar entre ellos por medio de mensajes transmitidos por protocolos de Internet [80].

- **Servicio Web 2.0:** estos servicios se basan en plataformas de red que permiten a los usuarios contribuir al desarrollo de contenidos, herramientas y comunidades en Internet. Un Servicio Web 2.0 se distingue de las páginas y servicios Web tradicionales que son estáticos y pasivos, y sigue paradigmas de creación de conocimiento dinámicos e interactivos en Internet [81]. Los servicios Web 2.0 están clasificados en cuatro tipos [81]: intercambiador (Tipo I), donde los usuarios comparten información de manera instantánea, por ejemplo: llamadas, VoIP y chat; Agregador (Tipo II), donde los pueden compartir y retener información, por ejemplo: blog, música y fotos; Colaborador (Tipo III), donde los usuarios comparten, retienen y asimilan conocimiento bajo estándares, por ejemplo: Wiki, ofimática y juegos de aprendizaje. Por último, liberador (Tipo IV), donde los usuarios pueden gestionar el conocimiento bajo estándares libres.

Es importante resaltar que los tipos de servicios descritos anteriormente se pueden componer, generando así servicios compuestos Telco (integran servicios básicos y suplementarios), servicios compuestos Web (integran servicios web tradicionales y Web 2.0), y servicios convergentes (integran servicios Telco y Web).

4.1.2. Servicios del catalogo

Teniendo en cuenta la revisión bibliográfica realizada, a continuación se describe el catálogo de servicios propuesto. Cada servicio es organizado según la clasificación propuesta en la sección anterior.

- **Servicios Básicos de telecomunicaciones**

Nombre	Descripción
Llamada	Comunicación mediante transmisión de voz.
SMS	Servicio de envío de mensajes cortos.

Video Llamada	Comunicación mediante transmisión de voz y video.
Transferencia de Datos	Permite la conexión en tiempo real entre dos puntos extremos con el fin de transferir diferentes tipos de contenidos.

Tabla 4.1. Servicios básicos de telecomunicaciones.

- **Servicios suplementarios de telecomunicaciones**

Nombre	**Descripción**
Identificación de llamada	Permite conocer el número de la persona que realiza una llamada.
Numero privado	Permite ocultar el número del cual se realiza la llamada.
Desvío de llamada	Permite a un usuario realizar la configuración de redirigir las llamadas entrantes hacia otro número previamente definido.
Presentación de los datos de usuario	Muestra la identificación (establecido en 80 caracteres), del usuario de la red.
Llamada en espera	Este servicio se divide en: *Llamada en espera:* cuando un usuario A se encuentra en comunicación con el usuario B, y un usuario llama a A, este puede colocar en llamada en espera al usuario B, para atender la llamada de C. *Llamada retenida:* permite interrumpir una llamada, y si lo desea volver a activarla o terminar definitivamente con esta.
Llamadas múltiples	Permite establecer múltiples llamadas simultáneamente.
Aviso de carga	Brinda información del costo de la llamada para el usuario.
Señalización usuario a usuario	Permite transmitir información del usuario, al momento de iniciar, terminar o durante la llamada.
Bloqueo de todas las llamadas entrantes/salientes	Brinda al subscriptor la capacidad de restringir las comunicaciones salientes, de acuerdo a una lista de restricción de llamadas.
Transferencia de llamada explicita	El usuario A recibe o realiza las llamadas a B y C, donde A tiene la capacidad de invocar este servicio, para establecer la comunicación entre los Usuarios B y C.
Prioridad de llamada	Permite al usuario establecer el nivel de prioridad de una llamada. El nivel de prioridad más alto, es utilizado por ejemplo, para llamadas de emergencia, y el segundo nivel puede ser reservado para la red. Ofrece 5 niveles de prioridad (0-4).

Tabla 4.2. Servicios suplementarios de telecomunicaciones.

- **Servicios compuestos de telecomunicaciones**

Nombre	Descripción
Mensajería instantánea	Permite al usuario crear un conversatorio basado en texto (chat room) con otro individuo en tiempo real.
Pulsar Para Hablar (PTT, Push To Talk)	Permite a una persona establecer una sesión multimedia con un grupo en particular pulsando sólo un botón. Cuando se ofrece a través de una red de telefonía celular es llamado *PoC* (Push To Talk over Cellular).
Conversión de voz a texto:	Convierte los mensajes de voz a texto. Una vez convertido, se envía el contenido al destinatario a través de un SMS.
Escucha Directa	Es un número corto que se incluye en el SMS y permite oír directamente el mensaje de voz que un usuario acaba dejar. Desde el teléfono móvil se puede realizar una llamada a este número para escuchar el mensaje.
Dicta SMS:	Cuando un usuario llama a un teléfono pero el destinatario no contesta o rechaza la llamada, se ofrece la posibilidad de grabar un mensaje de voz, que llegará como SMS al destinatario.
Desvío de Llamadas:	Permite desviar las llamadas que lleguen a un móvil para que pueda recibirlas en un fijo o en otro móvil, o para que vayan directamente a un contestador.
Restricción de llamada por el usuario o el operador	Restringe la recepción o emisión de determinadas llamadas, tanto las que un usuario realiza, como las que recibe.
Servicios personalizados por el usuario	Servicios de telecomunicaciones creados por un usuario determinado por medio de la unión de servicios básicos y servicios suplementarios

Tabla 4.3. Servicios compuestos de telecomunicaciones.

- **Servicios Web tradicionales**

Nombre	Descripción
Pagos en línea	Permite el pago a alguna entidad. Se puede realizar antes de obtener el servicio o después de consumirlo.
Llamada a un tercero	Creación y manejo de una llamada iniciada por una aplicación Web
Notificación de llamada	Proporciona funciones simples a desarrolladores de aplicaciones para determinar cómo debe ser tratada una llamada, por ejemplo, si es necesario finalizar continuar, o

	redirigir una llamada.
Manejo de cuenta	Consulta de Cuenta, Recarga Directa y Recarga a través de comprobantes o “vouchers”.
Estado del terminal	Obtener el estado de un terminal (Alcanzable, Inalcanzable u Ocupado).
Localización de terminal	Obtener la información sobre la ubicación de un Terminal.
Manejo de lista de direcciones	Manejo de grupos (alias) de subscriptores.
Presencia	Información de presencia (en línea, ocupado, no disponible).
Geocoding	Obtener la dirección de ubicación de un suscriptor, EJ: país, estado, distrito, ciudad, calle, número de casa, información adicional, código postal.
Capacidades del dispositivo y configuración	Obtener información relacionada con las capacidades de un dispositivo y enviar la configuración de este a otro.
Multimedia streaming control	Control de streaming (flujo de datos) de multimedia a un subscriptor. EJ: Transferencia entre terminales de usuarios.
Manejo de contenido	Permite subir y consumir contenido de la red (o de un proveedor de contenido).
Políticas	Ofrece aprovisionamiento y las funciones de evaluación para políticas.
Consulta de información	Acceso a un sitio Web para obtener información, por ejemplo: búsqueda en diccionarios, consulta de saldo, recarga de celular, compra boletos, traducción de textos, etc.

Tabla 4.4. Servicios web tradicionales.

- **Servicios Web 2.0**

Tipo	**Nombre**	**Descripción**
Intercambiador (Tipo I)	MSN	Abreviación de MicroSoft Network, conjunto de servicios que incluyen mensajería instantánea, correo electrónico, VoIP, etc.
	Skype	Software que permite la comunicación mediante texto, voz y video sobre internet (VoIP), empleando el modelo punto a punto (P2P).
	Google Talk	Cliente de mensajería instantánea y VoIP que hace uso del protocolo extensible de mensajería y comunicación de presencia (XMPP).
Agregador (Tipo II)	MySpace	Sitio web donde los usuarios comparten sus fotos, actividades, entre otros. Cuenta con servicios de noticias, clasificados, videos, aplicaciones para móviles, etc, además de APIs

		para desarrolladores.
	Youtube	Sitio web donde los usuarios pueden subir y compartir videos. Utiliza la tecnología Flash para reproducir los videos en línea.
	Facebook	Sitio web de red social, permite acceder y compartir sus pensamientos, fotos, videos y comunicación por chat.
	Twitter	Red social que permite compartir a sus usuarios información mediante: texto de máximo 140 caracteres y fotos; cuenta con una API abierta para desarrolladores.
	43things	Red social basada en los principios del etiquetado (tagging). Los usuarios ingresan una serie de gustos y expectativas, y dependiendo de estas, los usuarios son conectados a otros, que compartan características similares.
	Linkedin	Red social orientada a negocios y a profesionales. Permite a los usuarios, encontrar ofertas laborales, realizar negocios, conocer gente que comparte sus mismos gustos profesionales, entre otros.
Colaborador (Tipo lll)	Answers.com	Sitio Web basado en el intercambio de conocimiento sobre internet. Permite a los usuarios escribir preguntas o consultar preguntas previamente respondidas sobre diferentes temas.
	Saleforces.com	Sitio Web enfocado a desarrolladores donde estos pueden desarrollar sus aplicaciones y programas, en línea, gracias a una plataforma que está en un servidor.
	Yahoo Widget	Sitio Web que permite crear widgets en línea basados en la tecnología de JavaScript, AdobeFlash entre otros.
	Wikipedia	Proyecto para escribir comunitariamente una enciclopedia libre. Los usuarios pueden editar el contenido de los artículos de manera muy simple.
Liberador (Tipo IV)	Openoffice	Plataforma ofimática de libre distribución que incluye: Procesador de textos, hojas de cálculo, presentaciones, etc. Ésta disponible en varias plataformas como: Microsoft Windows, GNU/Linux, BSD, Solaris y Mac OS
	Linux	Linux es el Núcleo del sistema operativo GNU/Linux. Es el ejemplo más representativo de software libre; todo el código fuente puede ser

		utilizado, modificado y redistribuido libremente por cualquiera bajo los términos de la GPL.

Tabla 4.5. Servicios web 2.0.

- **Servicios Web compuestos**

Nombre	**Descripción**
Planeación de viajes:	Permite al usuario: reservar su vuelo, el hotel donde se desea hospedar, los recorridos por la ciudad, el alquiler de vehículos, etc.
Compras	El usuario escoge una variedad de artículos y los agrega a un carrito de compras, realiza el pago con tarjeta de crédito y adicionalmente se hace una llamada a un número telefónico mediante VoIP, para reconfirmar su compra.
Ubicación de mercancía	Ubica en qué lugar se encuentra la mercancía de un cliente y permite saber si esta ya fue entregada o cuantos días faltan para su entrega.
Servicios Bancarios	Notificación de transferencias, retiros, consignaciones, extractos.
Comunicación interactiva colaborativa:	Todo el conjunto de servicios para el soporte de conferencia multimedia con disponibilidad de compartir archivos y aplicaciones, como e-learning y juegos.
Servicios de despliegue de contenido:	Aplicaciones para la entrega de video y otros flujos de medios a los usuarios. Entre ellos se encuentran la radio, música, difusión de video, video bajo demanda, distribución de canal de televisión digital, distribución de información financiera, y distribución de contendido médico y profesional
Servicios de información	Por ejemplo, servicios de disponibilidad de boletos, de estado de tráfico, servicios avanzados de notificación, etc.
Servicios basados en localización	Entre ellos, servicios de guía turística, servicios de asistencia médica y hospitalaria, etc.
Servicios personalizados por el usuario	Servicios web creados por un usuario determinado por medio de la unión de servicios de la web tradicional o web 2.0 y cualquier combinación compatible entre estos dos.

Tabla 4.6. Servicios web compuestos.

- **Servicios convergentes**

Nombre	**Descripción**
Servicio convergente de	Este servicio notifica al servicio de presencia que el dispositivo del usuario está registrado y disponible para mensajería

voz y presencia	instantánea, de esta forma los usuarios en la lista de contactos son notificados que el usuario está disponible. Cuando el usuario recibe una llamada el servicio de presencia cambia su estado a "no molestar durante la llamada" el cual puede ser observado por todos los contactos.
Servicio convergente reenvío de llamada y videoconferencia	Cuando el usuario inicia el servicio de videoconferencia, su terminal cambia el estado a ocupado, con lo cual la red cambia su comportamiento en respuesta a futuras llamadas, de manera que si entran otras llamadas, deben ser reenviadas automáticamente a otro terminal
Movimiento de llamada conferencia de un cliente fijo a un cliente móvil	Servicio en el cual un usuario "A" mantiene una videoconferencia con "B". "A" decide cambiar de ubicación pero no desea interrumpir su conversación por el dispositivo móvil. Para ello, puede apretar un botón en su teléfono móvil y la parte audio de la llamada pasará automáticamente a su teléfono móvil.
Compras (Shopping)	Consiste en una tienda online capaz de realizar llamadas VoIP para informar a los usuarios o administradores acerca del estado de sus cuentas. Los usuarios llamados pueden afectar las decisiones de compra (aceptar, negar, establecer fecha de envío).
Facebook Event Reminder	Es un servicio que utiliza los servicios Telco de llamada y SMS, para recordarle a un usuario que se encuentra registrado en el servicio, los eventos próximos de Facebook a los que este va a asistir.
Confirmación de compras en Ebay	Complementa las funcionalidades del servicio Web 2.0 de Ebay. Al momento de finalizar el proceso de compra, el usuario recibe una llamada de confirmación, con el fin de evitar fraudes por suplantación de identidad.
CallMessage	Realiza una llamada a un usuario; en caso de que este no conteste, el servicio envía automáticamente un mensaje directo a Twitter, la información del usuario que lo llamo, la fecha y la hora.
TwitterSMS	El usuario ingresa a una aplicación Web, la cual ofrece dos servicios: enviar un SMS o enviar un mensaje directo a Twitter.
TwitterCalls	Servicio que permite llamar, enviar SMS, o dar un tweet a las personas que "te siguen", o las personas que "sigues", con la condición de estar registradas en un portal Web.

Tabla 4.7. Servicios convergentes.

4.2. Modelado se servicios

En esta sección es realizado el modelado de los servicios del catálogo utilizando la representación formal de redes de Petri, la cual fue seleccionada en el capítulo 3.

Debido a la gran cantidad de servicios presentes en el catálogo (73 servicios), solo fueron modelados 30 ya que algunos servicios tenían la misma estructura. A continuación se presenta un ejemplo de un servicio convergente del catálogo modelado con redes de Petri.

La Figura 4.2 muestra el modelado del servicio Shopping en redes de Petri, el cual ofrece al usuario la posibilidad de comprar diferentes productos en un portal Web, y una vez realizada la compra, este recibe una llamada para confirmar la compra del producto y la fecha estimada de envío. En el modelado de este servicio las transiciones representan que ha ocurrido un evento el cual ha sido generado por un usuario o por el mismo servicio dependiendo de su funcionamiento, y los lugares representan estados donde el usuario interactúa, o estados donde el servicio se encuentra esperando a que ocurra un evento.

Como lo indica la Figura 4.3, este servicio comienza en el lugar Idle, donde espera por la solicitud del usuario, luego con una transición que envía una petición Http pasa al lugar donde es invocado el servicio Web que permite escoger los productos a comprar, luego si el usuario cancela la orden vuelve al lugar Idle, y si confirma la orden y realiza el pago pasa por una transición al lugar donde se realiza una llamada automáticamente al usuario para confirmar su compra, si contesta la llamada pasa por una transición al lugar donde mediante un lector de pulsos de teléfono (DTMF Dual Tone Multi-Frecuency) puede aprobar (marca 1) o rechazar el pedido (marca 2), cuando aprueba pasa por una transición que analiza si el monto es superior a 100 dolares o no, si no es superior pasa a un lugar donde se hace una llamada al usuario y si contesta pasa a otro lugar para que establezca la hora y fecha de envió del producto; si el monto es superior pasa a un lugar donde es llamado un administrador, cuando contesta pasa a un lugar donde puede rechazar o aprobar la orden.

4.3. Detección de patrones

Ésta sección describe el método utilizado para la detección de patrones del flujo de control (CFP) y de flujo de datos (DP). Dicho método está basado en el análisis de los servicios modelados en la sección anterior y en un conjunto de posibles patrones presentes en dichos servicios. Al final de esta sección se muestra una lista con los patrones del flujo de ejecución detectados.

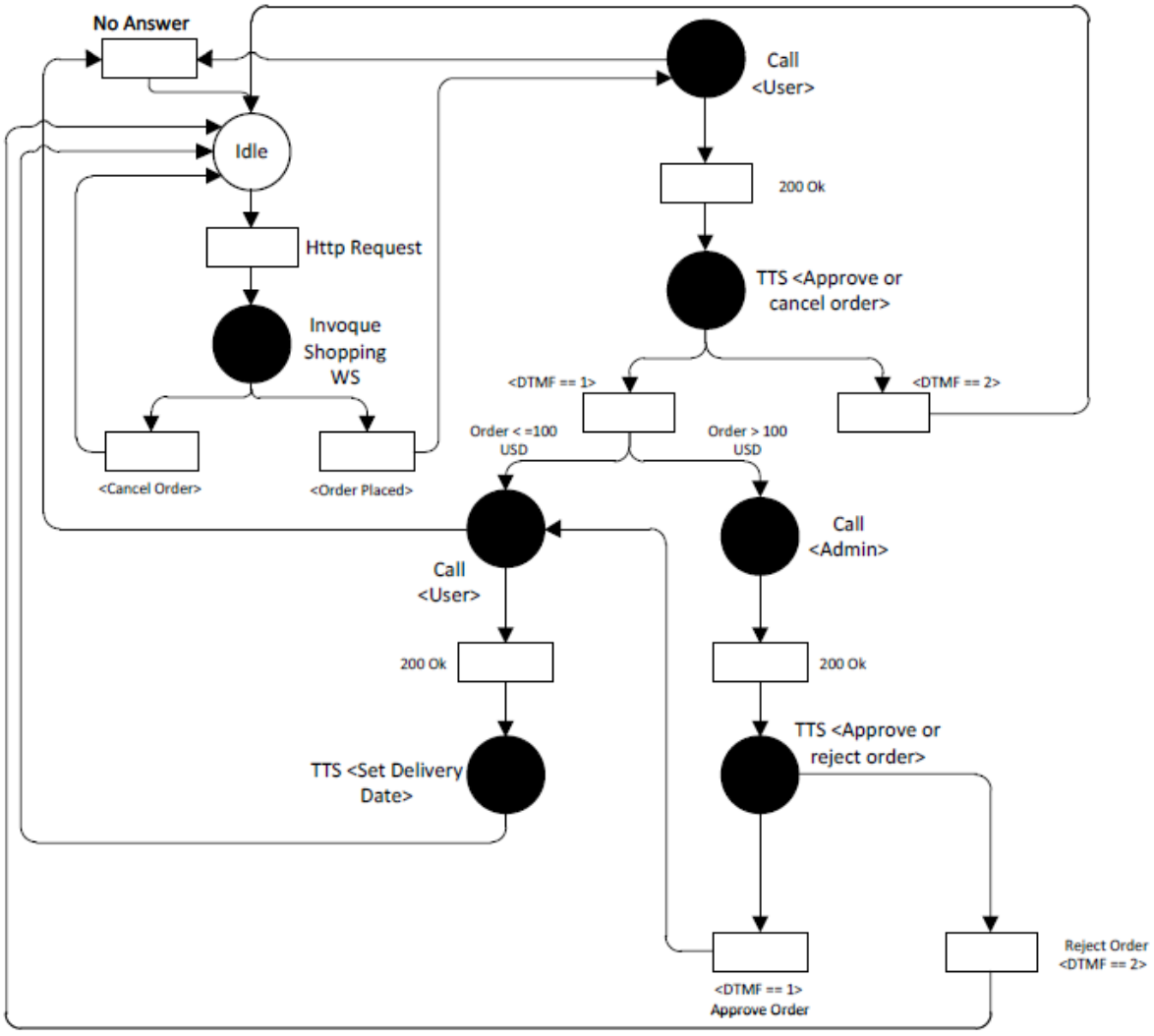


Figura 4.3. Ejemplo de servicio convergente con redes de Petri.

4.3.1. Detección patrones del flujo de control

Para la detección de los CFP se propone un método, cuyo principal componente es un algoritmo de detección, el cual permite hacer comparaciones estructurales para determinar similitudes entre subestructuras, sin embargo este algoritmo solo funciona con grafos, por lo que es necesaria una transformación de PN a grafos. Para realizar dicha transformación se tomó como referencia [82].

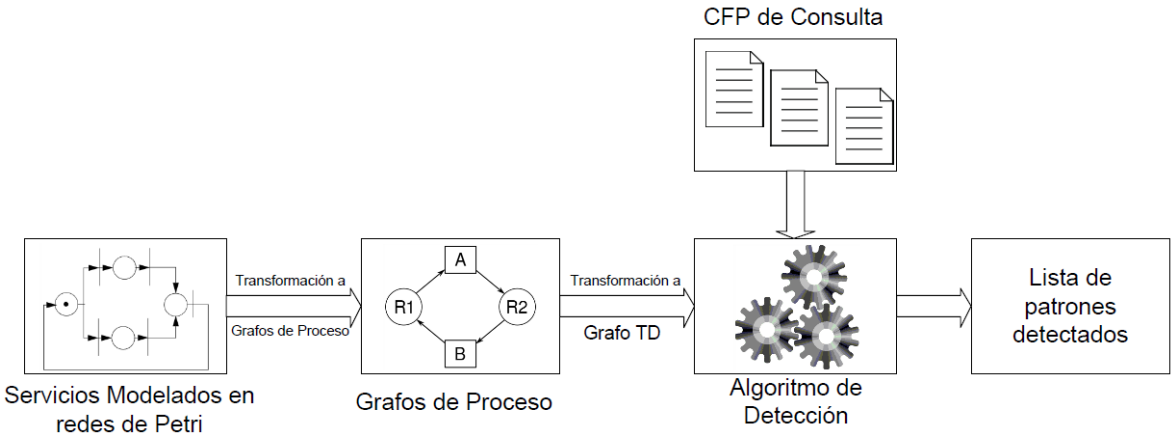


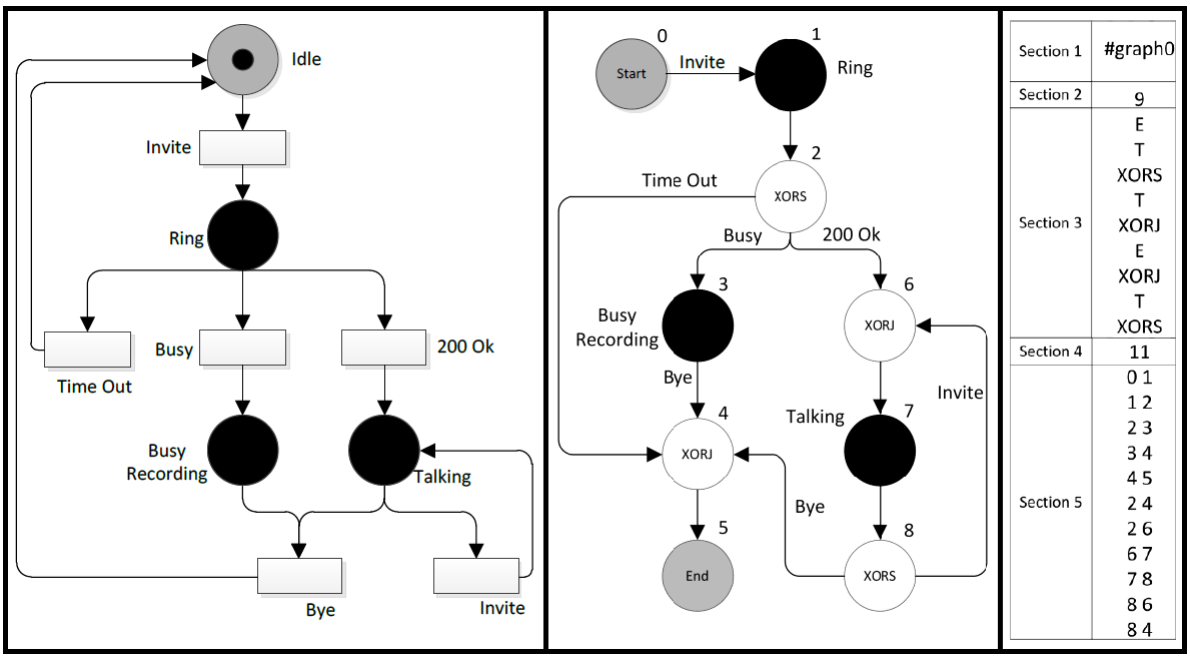
Figura 4.4. Método para detección de patrones

El método utilizado para la detección de CFP se ilustra en la Figura 4.4, el primer paso consiste en transformar todos los servicios modelados en redes de Petri, a grafos de proceso, los cuales permiten la representación del flujo de control mediante compuertas lógicas como AND, OR, XOR, de tipo JOIN o SPLIT; la Figura 4.5 b) muestra un ejemplo del grafo de proceso obtenido a partir de la PN descrita en la Figura 4.5 a).

El segundo paso consiste en expresar el grafo de proceso en un formato llamado “Grafo TD”, el cual es un archivo de texto con una sintaxis específica que representa el grafo de proceso, esto se debe a que el algoritmo de detección solo es capaz de procesar este formato; la Figura 4.5 c) muestra el un ejemplo del grafo TD correspondiente a la Figura 4.5 b), este se divide en tres secciones, la primera es el nombre, la segunda son los nodos y la tercera son las aristas.

El tercer paso es definir un conjunto de CFP de consulta, los cuales serán detectados en el grafo TD, estas CFP están definidos en [26] mediante PN, por lo tanto también es necesario transfórmalos a grafos de proceso. Un aspecto importante a resaltar es que del conjunto de 43 CPN que define [26], no fueron tenidos en cuenta los patrones de múltiples instancias, ya que dentro del catálogo de servicios no se encontró algún servicios que creara múltiple instancia de un servicio componente sin estar dentro de un ciclo repetitivo.

Finalmente, es aplicado el algoritmo de detección basado en grafos, el cual permite verificar correspondencias entre la estructura de los CFP de consulta y la estructura de los CFP presentes en el grafo TD del servicio, dando como resultado una lista de patrones detectados. Para utilizar este algoritmo y llevar a cabo la detección de patrones, fue utilizado el prototipo descrito en [83].



a) b) c)

Figura 4.5. a) PN del servicio b) Grafo de proceso del servicio c) GrafoTD del servicio.

La Figura 4.6 presenta un ejemplo de la detección de patrones realizada sobre el grafo TD de un servicio, en este ejemplo son detectadas dos subestructuras, la primera con una compuerta OR-SPLIT, la cual corresponde con un patrón *Multi-Choice* y la segunda con una compuerta OR-JOIN que corresponde a un patrón *Multi-Merge*. Las reglas de correspondencia entre subestructuras y patrones se encuentran en el Anexo C.

Un aspecto importante a resaltar, es que en algunos casos una subestructura detectada puede corresponder a varios patrones, por ejemplo los patrones *Generalized-AND-Join, Critical-Section* y *Interleaved-Routing* están asociados a una

subestructura con compuerta AND-Split + AND-Join, por lo tanto es necesario revisar manualmente la PN para determinar cuál es el patrón correcto.

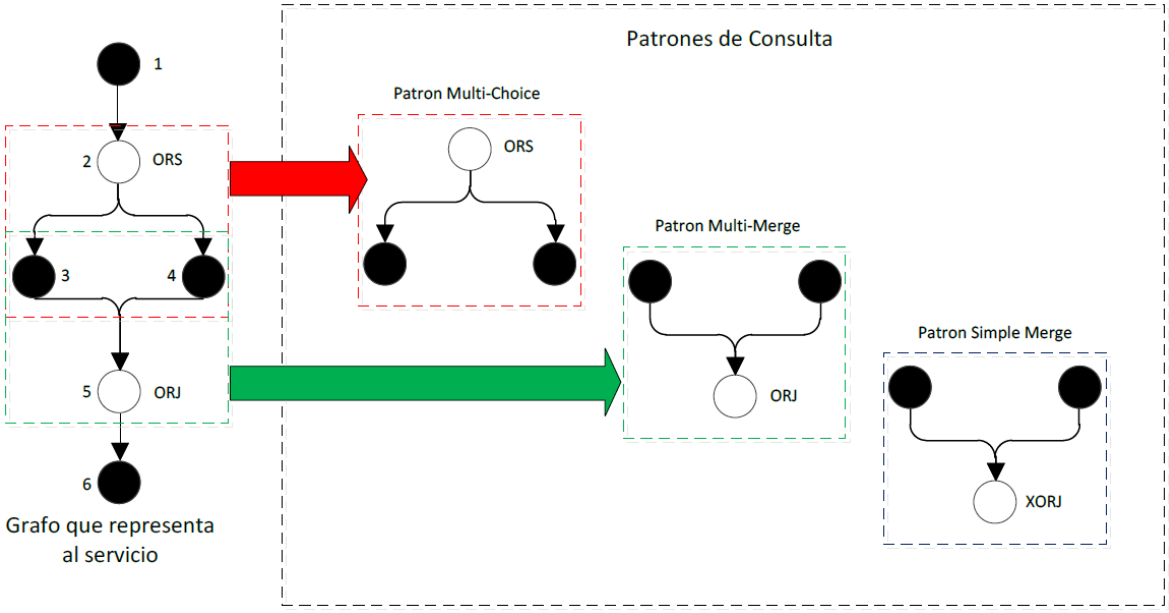


Figura 4.6. Ejemplo de detección de CFP.

4.3.2. Detección patrones del flujo de datos

Para la detección de los DP no se encontró algún algoritmo que facilitara dicha tarea, por lo tanto el proceso de detección tuvo que realizarse de forma manual, es decir tomando cada uno de los servicios del catálogo modelados en PN e identificando cuales DP estaban presentes. No obstante en [26] se definen 40 DP, lo cual implicó un proceso muy extenso, por lo tanto fue aplicado un criterio para reducir el conjunto de patrones a detectar. Dicho criterio consistió en analizar la especificación JAIN SLEE [43] con el fin de evaluar la viabilidad de implementación de cada DP sobre este tipo de entornos, de esta manera, el conjunto de patrones de estudio se redujo de 40 a 16 DP, los cuales están en la Tabla 4.8 con su respectiva descripción y forma como JAIN SLEE los soporta.

Una explicación detallada del funcionamiento y comportamiento de los 16 patrones, además de su representación mediante PN se encuentra en el Anexo D.

Tipo	Patrón	Soporte JAIN SLEE
Visibilidad de Datos	Scope Data	• *SLEE Profiles*: Define las variables de un servicio con relación al perfil de cada usuario [84].
	Case Data	
	Environment Data	
Interacción Interna de Datos	Data Interaction between Tasks	• *Firing Events:* Disparo de eventos dentro del SLEE [43]. • *Activity Context*: objetos que almacenan los datos dentro de los eventos.
Interacción Externa de Datos	Data Interaction - Task to Environment - Push-Oriented	• *Firing Events*: Disparo de eventos hacia y desde procesos y aplicaciones externas. • *Adaptadores de Recursos*
	Data Interaction - Environment to Task - Pull-Oriented	
	Data Interaction - Task to Environment - Pull-Oriented	
Transferencia de Datos	Data Passing by Value - Incoming	• *Adaptadores de Recursos* • *Integración con bases de datos (JPA, DAO).* • *Activity Context y Activity objects*.
	Data Passing by Value - Outgoing	
	Data Passing - Copy In/Copy Out	
	Data Passing by Reference - Unlocked	
	Data Transformation - Input	
	Data Transformation - Output	
Enrutamiento de Datos	Task Precondition - Data Value	• *Events Routing*: Eventos disparados desde aplicaciones externas. • *Firing Events*
	Event-based Task Trigger	
	Data-based Routing	

Tabla 4.8. Correspondencia entre DP y JAIN SLEE

4.4. Selección de patrones

4.4.1. Patrones del flujo de control seleccionados

Una vez fue aplicado el método para la detección de CFP explicado anteriormente a cada uno de los servicios del catálogo, se obtuvieron los CFP detectados en cada servicio, posteriormente fue determinada la recurrencia de cada CFP teniendo en cuenta el total detectado, como lo indica la ecuación (4.1):

$$Recurrencia\ patron\ n = \frac{Total\ p_n}{P}\ x\ 100 \tag{4.1}$$

$$\therefore\ n = patron\ \{1,2,..21\}$$
$$p_n = cantidad\ de\ patrones\ identificados\ de\ n$$
$$P = Total\ de\ Patrones\ identificados$$

Utilizando la ecuación (1) fueron obtenidos los resultados que muestra la Figura 4.7. Como se puede observar los patrones más recurrentes son: *Simple Merge* (P5) el cual converge de manera asíncrona o exclusiva dos ramas del flujo de control, este patrón se comporta de forma similar a una compuerta XOR u OR-exclusivo; *Deferred Choice* (P11) el cual diverge el flujo en dos ramas dependiendo del resultado de una tarea previa, este patrón se comporta de forma similar a una sentencia de programación If-Else; y *Cancel Case* (P16) el cual representa la terminación de actividades dentro del flujo.

Con los resultados obtenidos, se concluye que de los 43 CFP propuestos en [26] para implementar procesos de negocio y servicios en la Web, un conjunto de 13 CFP pueden ser utilizados para implementar servicios convergentes. Además, es importante resaltar que los patrones P1 a P5 son patrones básicos los cuales son la base para la formación de otros patrones definidos en [26]. Finalmente los 13 patrones identificados se resumen en la Tabla 4.9.

4.4.2. Patrones del flujo de datos seleccionados

Una vez realizada la detección de DP sobre cada uno de los servicios del catálogo, se obtuvieron los DP detectados en cada servicio, posteriormente fue determinada la recurrencia de cada DP teniendo en cuenta el total detectado, como lo indica la ecuación (1):

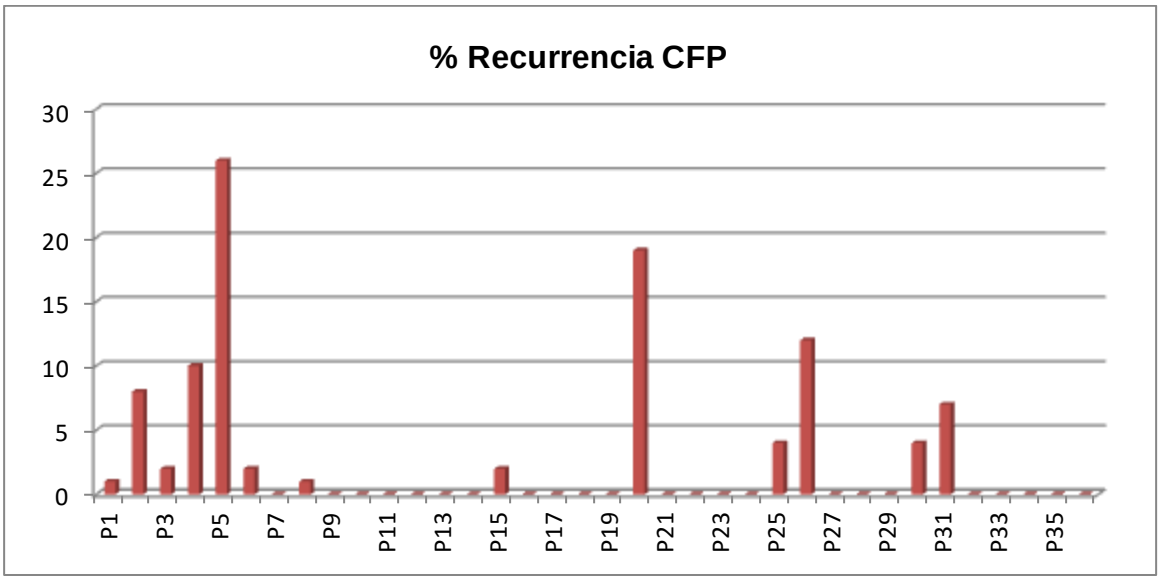


Figura 4.7. Recurrencia en la detección de CFP.

ID	Patrón
P1	Secuence
P2	Parallel Split
P3	Synchronization
P4	Exclusive Choice
P5	Simple Merge
P6	Multi-Choice
P8	Multi-Merge
P9	Generalized And-Join
P11	Deferred Choice
P15	Cancel Task
P16	Cancel Case
P17	Arbitrary Cycles
P18	Structured Loop

Tabla 4.9. Patrones del flujo de control seleccionados

Mediante la ecuación (1) fueron obtenidos los resultados que muestra la Figura 4.8, en los cuales se puede observar que los patrones más recurrentes son: *Case Data* (P2) el cual representa la definición y el uso de variables comunes para todo el servicio compuesto, es decir que todos los lugares y transiciones emplean estos datos establecidos previamente. *Task Precondition – Data Value* (P14) este patrón

representa le existencia de una condición previa que tiene que cumplirse para que el flujo pueda continuar. *Event-based Task Trigger* (P15) el cual representa el disparo de las actividades del flujo mediante actividades o ambientes externas, como por ejemplo: servicios web, aplicaciones de terceros, etc.

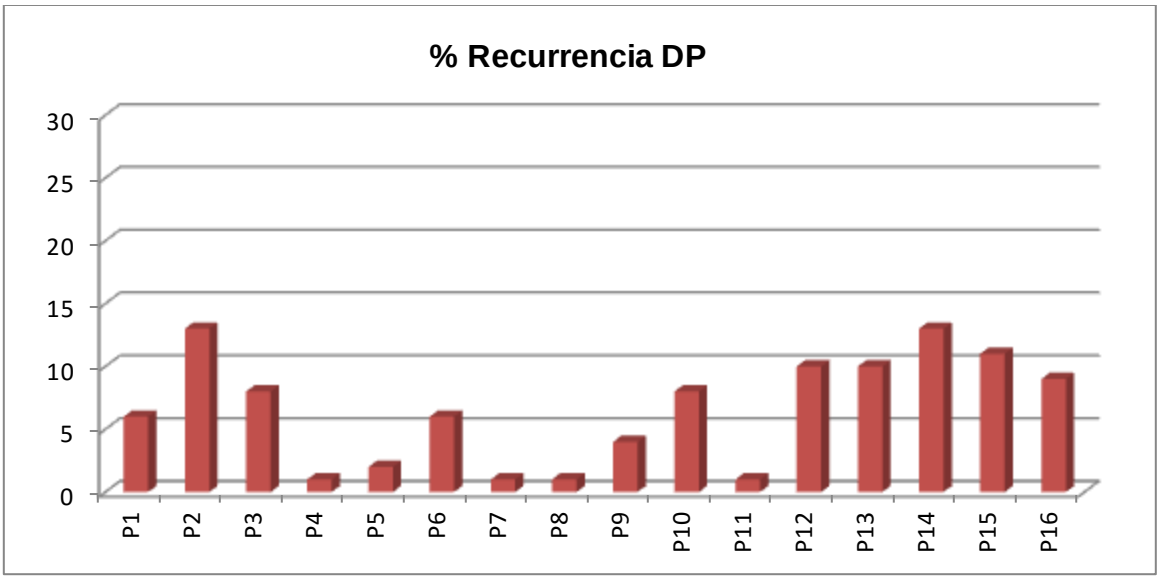


Figura 4.8. Recurrencia en la detección de DP

Con los resultados obtenidos, se concluye que de los 40 DP propuestos en [26] para implementar procesos de negocio y servicios en la Web, fueron seleccionados como patrones de consulta solo un conjunto de 16 DP tomando como criterio la viabilidad de implementación sobre entornos JAIN SLEE, posteriormente el análisis de recurrencia demuestra que todos los 16 DP son detectados en algún porcentaje, por lo tanto los 16 DP pueden ser utilizados para implementar servicios convergentes. Finalmente los 16 patrones identificados se resumen en la Tabla 4.10.

ID	Patrón
1	Scope Data
2	Case Data
3	Environment Data
4	Data Interaction between Tasks
5	Data Interaction - Task to Environment - Push-Oriented
6	Data Interaction - Environment to Task - Pull-

	Oriented
7	Data Interaction - Task to Environment - Pull-Oriented
8	Data Passing by Value - Incoming
9	Data Passing by Value - Outgoing
10	Data Passing - Copy In/Copy Out
11	Data Passing by Reference - Unlocked
12	Data Transformation - Input
13	Data Transformation - Output
14	Task Precondition - Data Value
15	Event-based Task Trigger
16	Data-based Routing

Tabla 4.10. Patrones del flujo de datos seleccionados.

Teniendo en cuenta la lista de patrones seleccionados tanto del flujo de control como del flujo de datos, los cuales son específicos para servicios convergentes, en el capítulo posterior será definido el mecanismo de orquestación automática, el cual utilizara dichos patrones como elementos claves que permiten la orquestación al interconectar los servicios web y Telco, por lo tanto estos patrones serán codificados e incluidos en un repositorio con el fin implementar un prototipo del mecanismo de orquestación automática.

4.5. Resumen

Este Capítulo presentó un método para la selección de patrones de flujo de control y de datos, específicos para componer servicios convergentes. La selección de patrones se realizó mediante cuatro fases, en la primera fase fue elaborado un catálogo de 73 servicios presentes en un entorno convergente, según estándares, operadores y plataformas para despliegue de servicios conocidas; en la segunda fase fueron modelados los servicios del catálogo mediante redes de Petri; en la tercera fase se realizó la detección de CFP aplicando un algoritmo basado en similitud de subestructuras, además la detección de DP por medio de una revisión manual de los servicios modelados. Finalmente, en la cuarta fase fue realizado un análisis de recurrencia sobre los patrones detectados, el cual entrego una lista de 13 CFP y 16 DP específicos para componer servicios convergentes.

Capítulo 5

5. Mecanismo de Orquestación

En este capítulo, se propone un mecanismo para automatizar la fase de orquestación de servicios convergentes en entornos JAIN SLEE, este mecanismo está basado en las representaciones formales seleccionadas anteriormente (Grafos y CPN) y utiliza los patrones del flujo de control y de datos identificados en el capítulo anterior.

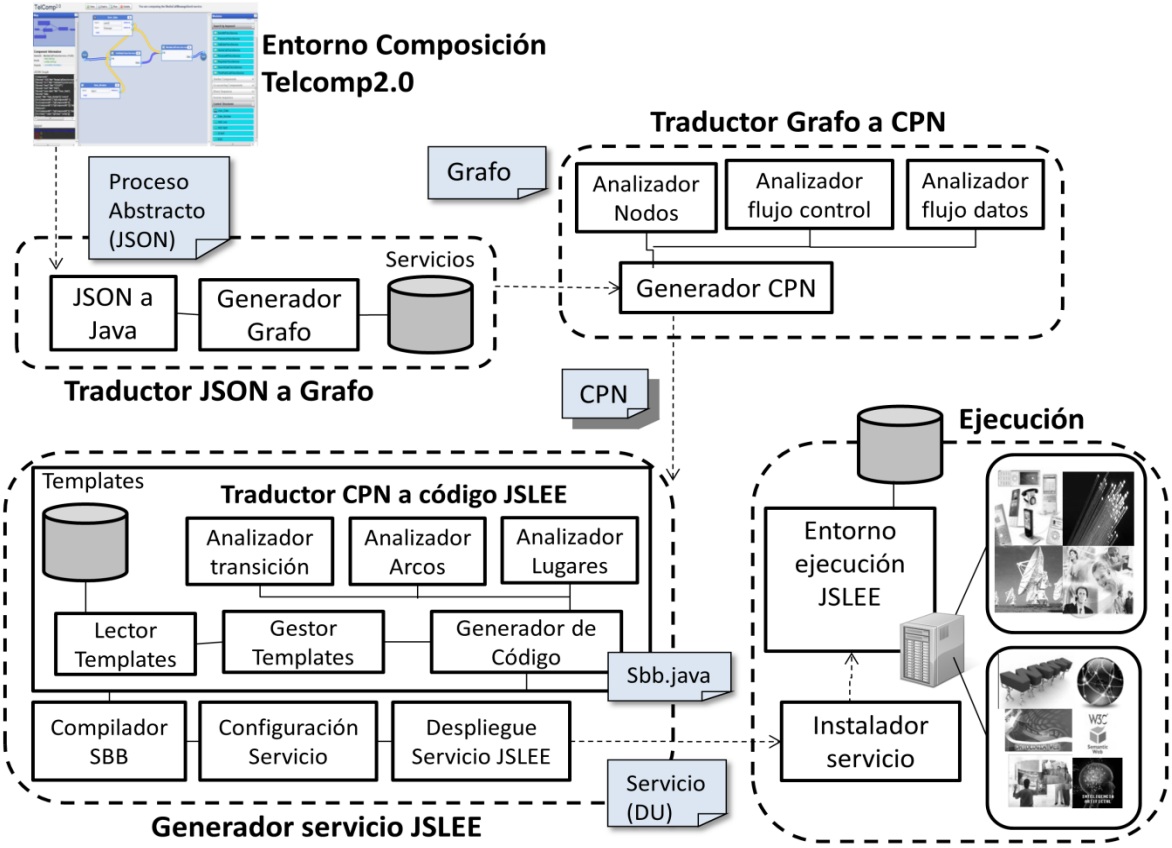


Figura15.1. Diagrama del mecanismo propuesto.

La Figura 5.1 presenta el diagrama general del mecanismo propuesto con todos sus módulos funcionales. A continuación será descrito cada uno de estos.

5.1. Entorno de composición

En esta sección, se define un entorno que permita componer servicios convergentes, y generar el proceso abstracto de un servicio compuesto, el cual es tomado como punto de partida en el mecanismo de orquestación.

Como se aclaró anteriormente, es posible utilizar técnicas automáticas empleadas en el dominio de la Web, como los planeadores de inteligencia artificial o la herramientas basadas en semántica, propuestas en [13][14][15], las cuales permiten generar procesos abstractos, a partir de simples solicitudes de los usuarios finales. Sin embargo, estas plataformas no se encuentran disponibles libremente, y además solo están enfocadas en servicios Web y no en servicios Telco, los cuales generalmente residen en servidores propios de los operadores de telecomunicaciones. Por lo tanto para el desarrollo de este proyecto, se utilizó el entorno de composición de servicios convergentes desarrollado en el proyecto TelComp2.0 [25], ya que tiene las siguientes ventajas: es una aplicación Web, genera procesos abstractos representados mediante grafos, y utiliza un repositorio disponible de servicios Web y Telco.

La Figura 5.2 presenta el entorno de composición del proyecto TelComp2.0, en el centro es posible observar el diseño de un servicio convergente mediante grafos. En este grafo se evidencian tres bloques que representan los servicios componentes, también aristas de diferentes colores, los cuales indican si la arista es de control (azul), de datos (amarilla), o de ambos tipos (verde); En la parte de la derecha se encuentran: arriba los buscadores de servicios dentro del repositorio, en el medio los servicios disponibles, y abajo los patrones que soporta la herramienta. En la parte de la izquierda se observa: un cuadro de texto (JSON Graph) donde es entregado el grafo resultado de la composición en notación JSON (JavaScript Object Notation), más arriba (Component Information) son descritos el nombre y los parámetros de entrada y salida de cada servicio.

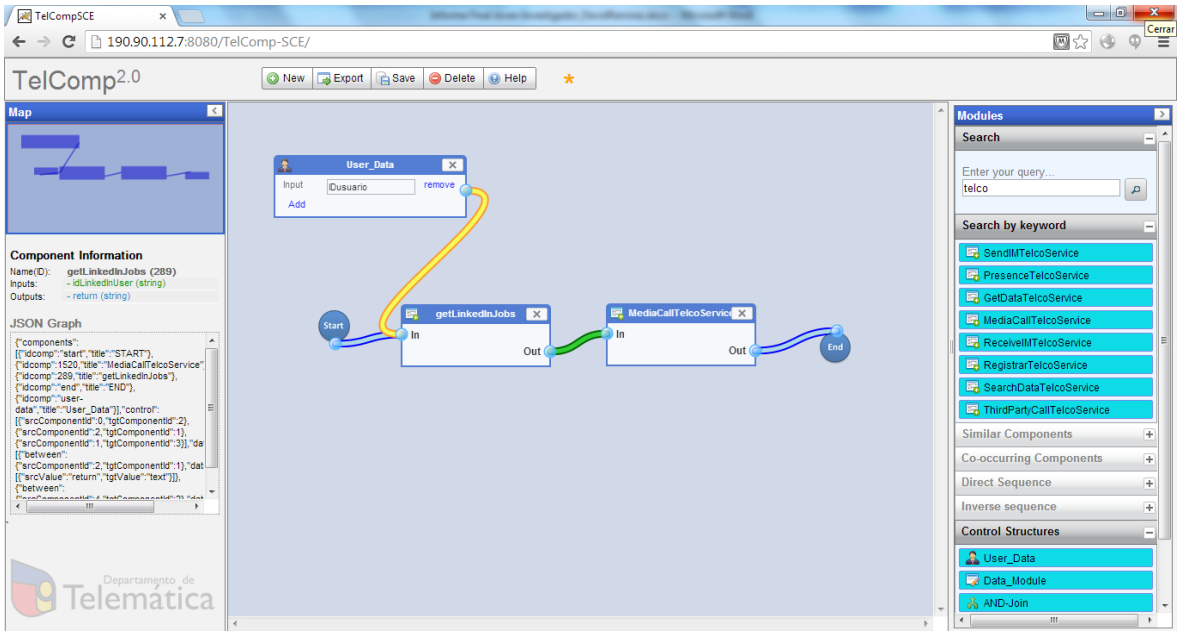


Figura 5.2. Entorno de composición del proyecto Telcomp2.0

La Figura 5.3 muestra en detalle el grafo en JSON resultante de la composición con este entorno, cuya estructura está constituida por tres partes (en rojo columnas A, B y C), en la columna A está una lista de los cinco componentes incluyendo servicios y patrones, descritos simplemente con un nombre (en rojo e) y un ID que los identifica dentro del repositorio de servicios (en rojo d). En la columna B están listadas las aristas del flujo de control entre componentes, definiendo desde (en rojo f) y hacia (en rojo g) que componentes van, por medio del índice en que son listadas en la columna A. Finalmente en la columna C están las dependencias de datos entre componentes, definiendo desde y hacia que componentes van al igual que las aristas de control, y además el nombre del parámetro de salida de un servicio (en rojo h) y el nombre del parámetro de entrada de otro servicio (en rojo i) al cual va conectada.

Por ejemplo para la Figura 5.3, en la columna A los componentes tendrían los nombres: Start, MediaCallTelcoService, getLinkedInJobs, End y User_Data, con los índices en la lista 0, 1, 2, 3 y 4 respectivamente. En la columna B la primer arista de control que va desde el componente 0 (Start) al 2 (getLinkedInJobs). En la columna C hay dos aristas de datos, la primera del 2 al 1, y la segunda del 4 al 2, la cual sería una arista de control y de datos al mismo tiempo; por ejemplo la segunda arista relaciona las variables IDusuario (salida de 4) e idLinkedInUser (entrada de 2).

Figura 5.3. Formato JSON del proceso abstracto

Como se evidencio en las gráficas anteriores, el entorno de composición del proyecto Telcomp2.0 permite la generación procesos abstractos correspondientes a servicios convergentes, por lo tanto se utilizó para probar el mecanismo propuesto en este trabajo. Dicho entorno está disponible en el siguiente enlace http://190.90.112.7:8080/TelComp-SCE/.

5.2. Traductor JSON a Grafo

Como se mencionó anteriormente, el proceso abstracto que genera el entorno de composición corresponde a un grafo que esta descrito en la notación JSON, sin embargo este grafo no contiene toda la información necesaria para realizar la orquestación. Por lo tanto, este módulo accede al repositorio de servicios con el fin de completar la información de cada servicio con las operaciones, parámetros de entrada y salida, tipos de datos, y ubicación física donde se implementa.

Para representar toda la información del proceso abstracto, es propuesto el modelo de grafos de la Figura 5.4, el cual define en la parte de la izquierda las clases: Conector que representan todos los patrones, Servicio que puede ser Telco o Web con su respectiva ubicación física, Operación que indica las tareas o acciones del servicio; y por la parte de la derecha las clases: Arista Control que une dos

conectores, Dato que define los datos y su tipo, Asociación Datos que asocia los datos de dos servicios, y Arista Datos que agrupa varias asociaciones de datos. Por otra parte, el grafo define relaciones entre las clases, ya que los servicios heredan de los conectores y contienen las operaciones donde se encuentra la información de los servicios Web y Telco, las aristas de datos heredan de las de control y contienen asociaciones entre los datos de entra y salida de las operaciones.

Es importante resaltar que en este módulo hay dos sub-módulos, el primero transforma el proceso abstracto descrito en JSON a un proceso abstracto similar descrito en el lenguaje Java, para este proceso es utilizada una librería de Google llamada *gson-2.2.1.jar*. El segundo sub-modulo transforma el proceso abstracto descrito en Java a un grafo que utiliza el modelo propuesto de la Figura 5.4, igualmente implementado en lenguaje Java.

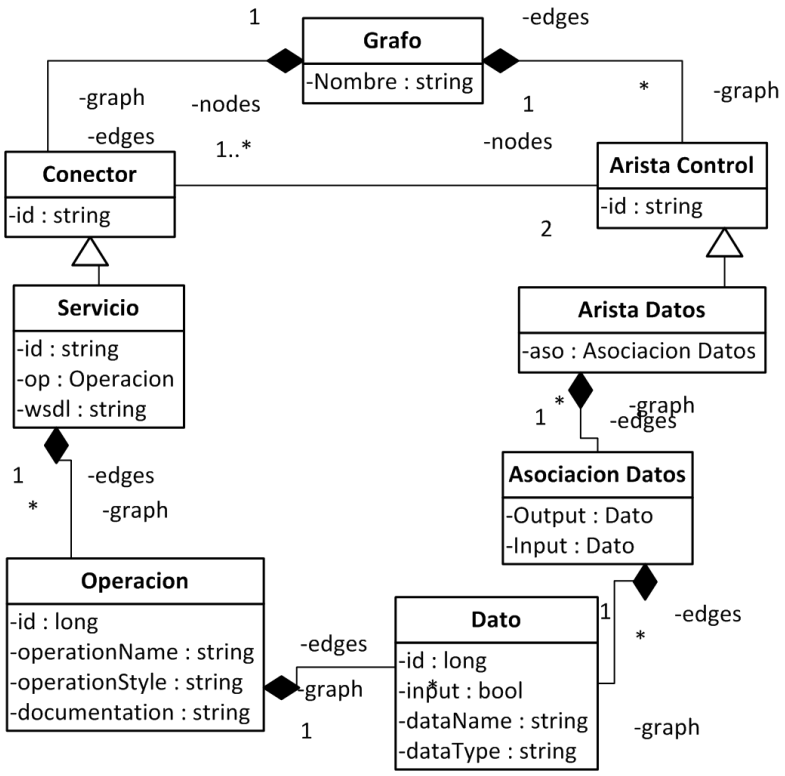


Figura 5.4. Modelo de grafos propuesto para procesos abstractos

5.3. Traductor grafo a CPN

Este módulo se encarga de convertir el proceso abstracto descrito en el modelo de grafos propuesto, en una CPN que represente la orquestación del servicio convergente. A continuación son analizadas las técnicas existentes para dicha conversión, y posteriormente es descrita la técnica utilizada en este trabajo.

5.3.1. Revisión de técnicas de transformación de Grafos a PN

Actualmente varios estudios académicos presentan enfoques y aproximaciones para realizar la traducción de grafos a PN. En ese sentido, a continuación es presentada una descripción de los trabajos de mayor relevancia.

En [85] los autores realizan la adaptación de una estructura de la composición representada en BPEL a una representada en PN. La finalidad de la adaptación es efectuar la verificación de la estructura de composición en BPEL, específicamente, la propiedad estructural de solvencia[6]. Para llevar a cabo dicha adaptación, los autores desarrollaron la herramienta WolfBPEL para el análisis de procesos BPEL. Este elemento está acompañado de la herramienta BEPL2PNML encargado de trasladar la definición del proceso en BPEL a una representación de PN en su respectivo lenguaje, PNML.

En [86] los autores realizan un estudio de validez de un servicio de videoconferencia orquestado con BPEL. Para ello, parten de la estructura de composición del servicio a la cual se le aplican reglas de conversión con base en patrones de comportamiento, dando lugar a una estructura compuesta en lenguaje de PN. La transformación es llevada a cabo con el fin de analizar la estructura del servicio compuesto, tomando como base algoritmos y técnicas de evaluación como el árbol de cobertura, la matriz de incidencia, ecuación de estado y la matriz transitiva.

En [82] proponen diferentes métodos de transformación de estructuras PN a grafos modelados en BPEL. Para ello, los autores presentan un conjunto de reglas de reducción estructural, obteniendo modelos PN menos complejos para el análisis. Entre las técnicas empleadas se pueden resumir las siguientes: reducción de clase, donde se pasa de una HPN a una red básica de PN equivalente; reducción estructural, donde el uso de reglas permiten transformar actividades complejas en

[6] Propiedad estructural para el análisis de puntos muertos (deadlocks) o falta de sincronización (lack of synchronization)

tareas más sencillas de procesar. De igual manera, se presentan otros métodos de transformación de PN a grafos. En el primero, las transiciones son tomadas como nodos, mientras los lugares y sus respectivos arcos de salida y entrada son reemplazados por arcos en el grafo. En el segundo, las transiciones y sus arcos son trasladados a un simple arco en el grafo, los lugares tomados como nodos del grafo. En el tercer método, los lugares y transiciones son reemplazados por nodos del grafo, los arcos de la PN trasladados a arcos del grafo. Es importante resaltar que el proceso de conversión de PN a grafos empleados en este trabajo, es recíproco. Esto indica que es posible usar las mismas técnicas para realizar la transformación del grafo a una estructura PN equivalente.

En [87] proponen el modelado de servicios convergentes mediante PN, utilizando el concepto de Snet (Service Net), el cual propone que cada servicio componente de la orquestación se puede expresar como una PN que tiene un lugar de entrada y una de salida, una transición, y un token de entrada y de salida. El objetivo de este trabajo es modelar la orquestación en BPEL con PN para verificar que la estructura de la composición esta correcta.

En conclusión, los anteriores cuatro trabajos presentan técnicas de transformación de grafos a estructuras en PN partiendo del análisis de patrones en la estructura de la composición, utilizando algoritmos propios y especializados, lo que dificulta su reproducción en otros ambientes distintos a los abarcados por las herramientas que los soportan. En este sentido los primeros dos trabajos son adecuados para representar BPEL y verificar si su estructura es correcta, pero no enfatizan en el paso de grafos a PN. El tercer trabajo propone métodos analíticos que sirven como base para formular nuevos esquemas y técnicas de transformación, no obstante estos métodos no tienen en cuenta el contexto de los servicios convergentes. Finalmente el cuarto trabajo plantea una técnica sencilla enfocada en los servicios convergentes, sin embargo no propone algoritmos para el paso de grafos o PN.

5.3.2. Técnica de transformación de Grafos a CPN

Para definir la transformación de grafos a CPN fueron tomados como base [87][88] en la descripción de la orquestación de servicios convergentes y como técnica de conversión de grafos a CPN; y [26] como guía de modelamiento en CPN de los patrones seleccionados.

Teniendo en cuenta lo anterior, en este trabajo de grado se define una conversión de grafos a CPN la cual establece que para cada nodo que represente un servicio componente en el grafo, la traducción en CPN está definida como lo indica la Figura 5.5.

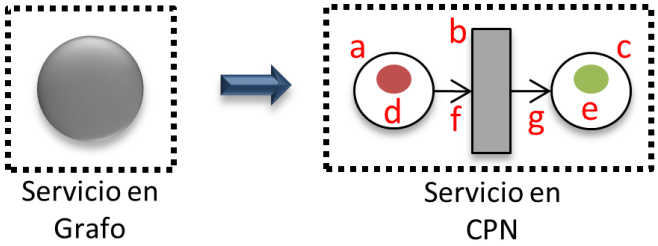


Figura 5.5. Equivalencia entre servicios en grafos y CPN.

En la figura 5.5 se puede observar en letras rojas los elementos de la CPN equivalente, los cuales son descritos a continuación:

- **a:** Un lugar de entrada el cual indica que el servicio está esperando su invocación
- **b:** Una transición intermedia que indica que el servicio está ejecutando su lógica
- **c:** Un lugar de salida el cual indica que el servicio está enviando su respuesta
- **d:** Un token en el lugar de entrada que indica los datos de entrada al servicio
- **e:** Un token en el lugar de salida el cual indica los datos de salida al servicio
- **f:** Un arco que conecta el lugar de entrada con la transición indicando el evento de invocación del servicio
- **g:** Un arco que conecta la transición con el lugar de salida indicando el evento de respuesta del servicio.

Esta representación en CPN tiene un nivel alto de abstracción para el funcionamiento de los servicios componentes, ya que el comportamiento interno del mismo podría estar descrito por una cantidad numerosa de estados y transiciones, lo cual haría complejo el análisis de la estructura del servicio convergente. En este sentido, se toma un modelo de petición respuesta para la ejecución del servicio componente, dejando la lógica de interacción asíncrona al interior del orquestador (el SLEE).

De igual manera, los patrones del flujo de control son empleados en el modelo de grafos como un solo nodo con su respectivo nombre (ParallelSplit SimpleMerge,

etc.), pero en CPN son modelados siguiendo la estructura definida en [26]. En este sentido, la Figura 5.6 expone algunos ejemplos de conversión para los patrones de flujo de control.

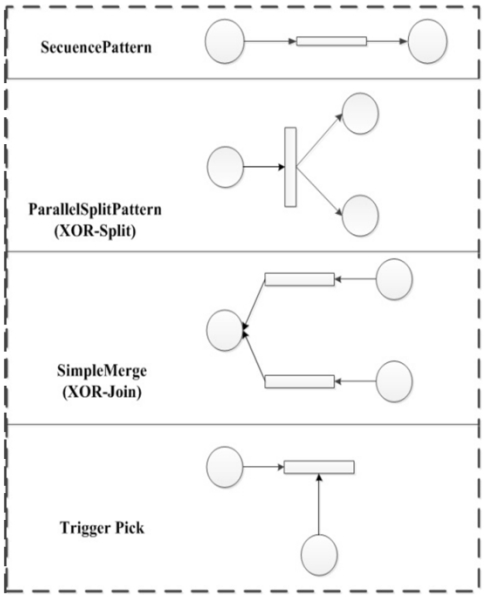


Figura 5.6. Ejemplos de CPN asociadas a cada patrón

Teniendo en cuenta las equivalencias en CPN de los servicios componentes y los patrones de control, para la traducción de grafos a CPN existen cuatro sub módulos como lo muestra la Figura 5.1; el sub módulo Analizador de nodos revisa cada nodo del grafo y detecta el tipo de nodo, si es servicio o patrón, en el caso de ser patrón accede al repositorio y carga la CPN asociada al patrón, en caso de ser servicio es creada una CPN lugar entrada – transición – lugar salida, como se especificó en la Figura 5.5. Posteriormente el sub modulo Analizador del flujo de control determina el orden de ejecución de la CPN, analizando todas las aristas de control y creando arcos que conectan las CPN de servicios y patrones. A continuación, el sub módulo Analizador del flujo de datos define los datos de entrada y salida de cada servicio, creando los tokens de la CPN; en este sentido, crea los token de entrada y salida de cada CPN, tomando cada nodo y analizando todas las aristas de datos, con el propósito de buscar las que lo tengan como nodo destino y crear el token de entrada, además para crear el token de salida toma cada nodo y analiza todas las aristas de datos buscando las que lo tengan como nodo fuente. Finalmente, el sub módulo Generador de CPN es el módulo encargado de organizar la información adquirida y

crear la CPN del servicio convergente. El procedimiento llevado a cabo en los sub módulos descritos está definido en el Algoritmo 1, el cual tiene como entrada el servicio convergente descrito con el modelo de grafos propuesto, y como salida el servicio convergente descrito en CPN.

Algoritmo 1. Traducción de grafos a CPN

Inputs: $Grafo$ {Servicio convergente en grafos}
Outputs: CPN {Servicio convergente en CPN}.

Begin

1: **for** each $Nodo$ in $Grafo$ **do**
2: **if** $Nodo \in Patron$ **then**
3: Get $CPN\ patron$
4: **else**
5: ADD $new\ Transicion$
6: ADD $new\ Lugar\ Entrada$
7: ADD $new\ Lugar\ Salida$
8: ADD $new\ Arco\ Entrada\ (Lugar\ Entrada\ to\ Transicion)$
9: ADD $new\ Arco\ Salida\ (Transicion\ to\ Lugar\ Salida)$
10: **end if**
11: **end for**
12: **for** $each\ Arista\ Control$ **do**
13: **if** $Arista\ Control\ .getNodoFuente \in$ *Patron* **then**
14: ADD $new\ Arco\ (Transicion\ patron\ to\ Lugar\ Entrada)$
15: **else**
16: ADD *new Arco (Lugar Salida to Transición Patrón)*
17: **end if**
18: **end for**
19: **for** $each\ Nodo\ in\ Grafo$ **do**
20: ADD $new\ Token\ Entrada$
21: **if** $Nodo \in Servicio$ **then**
22: **for** $each\ Arista\ Datos$ **do**
23: **if** $Arista\ Datos\ .getNodoDestino = Nodo$ **then**
24: ADD $Data\ Output\ to\ Token\ Entrada$
25: **else if** $Arista\ Datos\ .getNodoFuente = Nodo$ **then**
26: ADD $Data\ Input\ to\ Token\ Salida$
27: **end if**
28: **end for**
29: **end if**
30: **end for**

31: crearCPN
32: añadirElementos to CPN
33: **return** CPN
End

La Figura 5.7 presenta un ejemplo de la traducción de grafos (parte de arriba) a CPN (parte de abajo), utilizando el algoritmo 1. En este ejemplo es posible observar que los servicios componentes 1 y 2, que son Web y Telco respectivamente, se traducen a sus CPN equivalentes, independientemente si es Web o Telco. Además, se agregan los patrones inicio, fin, trigger y secuencia, con sus respectivas CPN, también fue definido el flujo de control como lo especifican las aristas del grafo. Adicionalmente, se crean los token para los lugares de entrada y salida de cada servicio, y el lugar de interacción con el usuario que define el patrón trigger. Es de resaltar que los token tienen diferentes colores ya que representan diferentes tipos de datos.

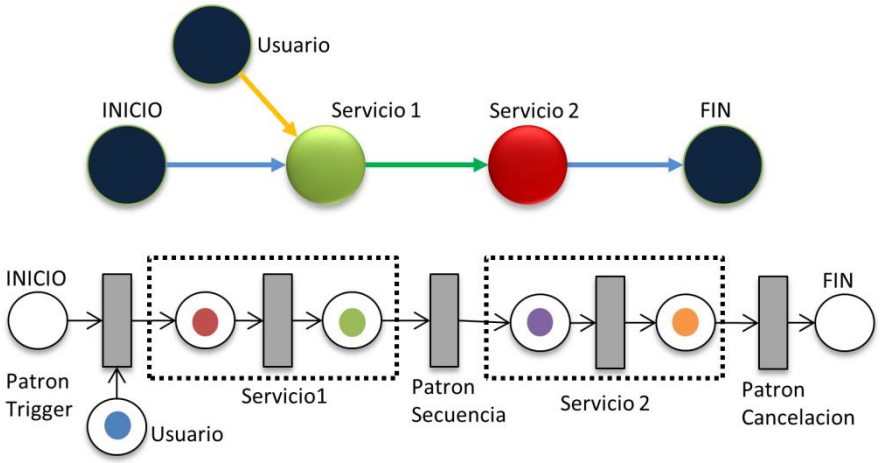


Figura 5.7. Equivalencia entre servicios en grafos

5.4. Traductor CPN a código JSLEE

Este módulo genera un archivo Java ejecutable (SBB) correspondiente a la orquestación del servicio, a partir de la CPN creada. La técnica utilizada para esta tarea son los Templates o Snippets, los cuales son fragmentos de código genéricos donde se configuran algunas variables, dicha técnica es utilizada en la mayoría de

entornos gráficos que generan algún tipo de código fuente. Actualmente existe una variedad de herramientas que permiten gestionar templates, sin embargo gran parte de estas herramientas están ligadas a entornos de desarrollo como Netbeans o Eclipse, lo cual dificulta la automatización y disminuiría el rendimiento del mecanismo, además los tipos de archivos que generan pueden contener código fuente genérico, pero ninguna de las herramientas genera el formato especifico de servicios JSLEE. Por lo tanto en este módulo se optó por proponer un generador de código basado en templates específico para JSLEE teniendo en cuenta las herramientas antes mencionadas.

En este sentido, el sub modulo principal que permite crear el SBB.java es el generador de código, el cual por una parte utiliza los sub módulos que analizan y extraen información de la CPN, y por otra parte los sub módulos que gestionan los templates. A continuación es escrita la función de cada sub modulo.

- **Analizador transiciones:** por cada transición analiza: si es servicio para determinar su ubicación física con el fin de crear el evento de invocación, ó si es patrón para obtener el código alojado en el repositorio.
- **Analizador lugares:** por cada lugar determina un estado de ejecución de los servicios componentes o patrones. Además a cada lugar está asociado un token, el cual brinda información de los datos que se deben agregar a los eventos de invocación y respuesta.
- **Analizador arcos:** por cada arco analiza su fuente y destino, creando el flujo de control del servicio, el cual define el paso de un estado de ejecución a otro estado, cuando se realizan las operaciones respectivas.
- **Lector templates:** accede al repositorio y carga en memoria todos los templates disponibles, ordenándolos mediante etiquetas para facilitar el acceso a otros sub módulos.
- **Gestor templates:** gestiona la carga y configuración de las variables en los templates.
- **Generador de código:** utiliza todos los sub módulos anteriores para configurar toda la información de la orquestación en los templates, generando así un archivo SBB.java y su descriptor SBB.xml.

Como se mencionó anteriormente el traductor de CPN a JSLEE genera un archivo SBB.java utilizando templates, por lo tanto fue propuesta una plantilla principal para el SBB.java en la cual el generador de código ensambla los templates configurados. Dicha plantilla principal contiene una lista de ítems, cada ítem tiene asociado un template configurable. Todos los templates utilizados se encuentran descritos en el Anexo E. A continuación se presenta el contenido de la platilla principal.

- **Import:** todas las librerías externas utilizadas.
- **Importevent:** las librerías que definen los servicios Telco utilizados.
- **Classstart:** el código básico que siempre se ejecuta para iniciar el SBB.
- **Activities:** se crea una actividad por cada rama del flujo de control.
- **Variables:** todos los datos de entradas y salidas constituyen una variable.
- **Fireevent:** son los disparadores abstractos que invocan los servicios componentes.
- **Estates:** cada lugar de entrada y salida corresponde a un estado de ejecución de los servicios componente y los patrones.
- **Mapping:** corresponde a las relaciones entre variables de los servicios componentes.
- **Receiveevent:** son los manejadores de los eventos que recibe el SBB como respuesta desde los servicios componentes.
- **Trigger:** aquí se configuran los eventos de invocación a los servicios componentes. Antes de disparar los eventos, estos se configuran con sus datos de entrada.
- **Webservice:** se configuran todas las invocaciones de los servicios web.
- **Controlflow:** se define el flujo de control teniendo en cuenta las transiciones habilitadas.
- **Classend:** termina y cierra en código.

La Figura 5.8 presenta un ejemplo del código correspondiente al template Receiveevent, el cual permite recibir en el SBB los eventos de respuesta de los servicios componentes. En rojo están las variables que deben ser configuradas al momento de utilizar el template, por ejemplo _name_ es sustituida por el nombre del servicio componente, //_data_ es sustituida por la asignación de variables con datos recibidos, y //_receiveState_ es sustituida por la activación del estado de ejecución del siguiente servicio componente.

```
//template
//receiveevent
public void onEnd_name_Event(End_name_Event event, ActivityContextInterface aci) {
        System.out.println("<<<---- Recibe evento _name_ !!!!! ");
    //_data_
    //_receiveState_
    this.dataMapping();
    this.controlFlow();
    this.triggerEvent();
}
//end
```

Figura 5.8. Ejemplo de Template utilizado en el generador de código

La secuencia de pasos que se llevan a cabo dentro de este módulo se encuentra descrita en el algoritmo 2, y en el Anexo F se encuentra un ejemplo del código fuente generado para un servicio, utilizando este módulo y dicho algoritmo.

Algoritmo 2. Traducción de CPN a código JSLEE

Inputs: CPN {Servicio convergente en CPN}, templateList {Plantilla principal de templates para el SBB}, templates {Templates disponibles}
Outputs: $SBBcode$ {Código fuente del servicio convergente}.

Begin
1: new SBBcode
2: **for** each $Template$ in TemplateList **do**
3: **if** (Template = Import OR Template = Classstart OR Template = Classend) **then**
4: ADD Template. setParameters(CPN.nombre)
5: **else if** (Template = Importevent OR Template = Fireevent OR Template = Trigger) **then**
6: **for** each $Transicion$ in CPN **do**
7: **if** Transicion ∈ Servicio Telco **then**
8: ADD Template.setParameters(Transicion.nombre) to SBBcode
9: **end if**
10: **end for**
11: **else if** Template = Actividades **then**
12: **for** each Rama in CPN **do**
13: ADD new Actividad to SBBcode
14: **end for**
15: **else if** Template = Variables **then**
16: **for** each Lugar in CPN **do**

```
17:              GET Lugar.Token
18:             ADD Template.setParameters(Token.Datos) to SBBcode
19:             ADD new Arco (Lugar Salida to Transición Patrón)
20:         end for
21:     else if Template = Estados then
22:         for each Lugar in CPN do
23:             ADD Template.setParameters(Lugar) to SBBcode
24:         end for
25:     else if Template = Mapping then
26:         for each Arco in CPN do
27:             var1 = Arco.getlugarDestino.getToken.Datos
28:             var2 = nextArco.getLugarFuente.getToken.variable
29:             var2 = var1
30:             ADD Template.setParameters(var2) to SBBcode
31:         end for
32:     else if Template = Receiveevent then
33:         for each Lugar in CPN do
34:             if Lugar ∈ Lugar Salida
35:                 ADD Template.setParameters(Lugar.Token.Datos) to SBBcode
36:             end if
37:         end for
38:     else if Template = Webservice then
39:         for each Transicion in CPN do
40:             if Transicion ∈ Servicio Web then
41:                 ADD Template.setParameters(Transicion.nombre) to SBBcode
42:             end if
43:         end for
44:     else if Template = Controlflow then
45:         for each Arco in CPN do
46:             state1 = Arco.getlugarDestino.nombre
47:             state2 = nextArco.getLugarFuente.nombre
48:             state2 = state1
49:             ADD Template.setParameters(state2) to SBBcode
50:         end for
51:     end if
52: return SBBcode
End
```

Por otra parte, es importante resaltar que la función principal de este módulo es generar el SBB.java, sin embargo aquí también son generados otros archivos complementarios como los descriptores xml, utilizando la técnica de templates pero

con un bajo nivel de complejidad puesto que los templates de estos archivos solo requieren configuración del nombre del SBB. En el siguiente modulo serán explicados dichos archivos complementarios.

5.5. Generador servicio JSLEE

Este módulo es el encargado de convertir el código del servicio generado en el módulo anterior (SBB.java), en una unidad desplegable (DU), la cual es el formato definido por JSLEE para el empaquetado de sus servicios, es decir que este módulo debe generar un servicio (DU) que pueda ser desplegado e instalado sobre un entorno JSLEE.

La Figura 5.9 presenta la estructura del empaquetado de la DU con todos sus componentes, los cuales son descritos a continuación.

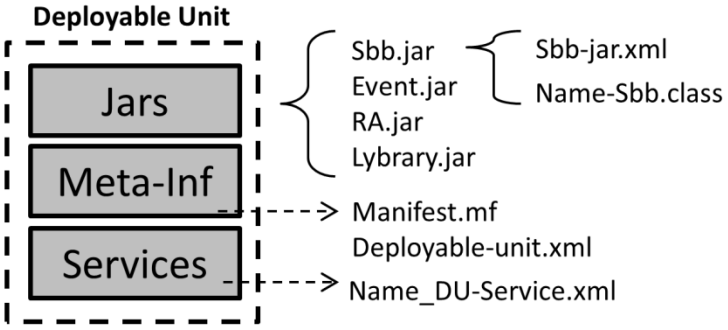


Figura 5.9. Estructura de empaquetado de servicios JSLEE

- **Jars:** contiene los elementos que interactúan en el funcionamiento del servicio. JSLEE ha definido un conjunto de componentes básicos, estos jar son: SBB, Event, Resource Adaptor y Lybrary. El elemento principal es el SBB, puesto que en este *jar* se encuentran alojados el código compilado del servicio SBB.*class*, y el descriptor del servicio SBB.xml. Los Event.jar también son importante ya que permiten la comunicación y el paso de datos con los servicios componentes. Es de resaltar que dependiendo de la complejidad del servicio, varios elementos pueden ser introducidos con el fin de implementar la funcionalidad de éste, como por ejemplo diferentes adaptadores de recursos o librerías específicas.

- **Meta-Inf:** contiene el descriptor xml de la DU (deployable-unit.xml) y el archivo meta-data (MANIFEST.MF), los cuales describen la versión y contenido de la DU.

- **Services:** contiene el descriptor xml del servicio, que permite gestionar su ciclo de vida una vez ha sido instalado.

Una vez generado el código del servicio (SBB.java) y los descriptores xml, en el módulo anterior, este módulo contiene tres sub módulos que permiten generar la DU. El primer sub modulo es el de Compilación del SBB, el cual se encarga de llamar al compilador de la máquina virtual de java, donde este alojado el mecanismo, generando así el SBB.class. El segundo sub módulo de configuración del servicio organiza los descriptores xml y demás archivos necesarios en la DU, según lo indica la estructura de la Figura 5.9. El tercer sub módulo está encargado de desplegar la DU del servicio convergente JSLEE, para esto envía la DU al servidor mediante HTTP. Finalmente, el sub módulo instalador del servicio que se encuentra del lado del entorno de ejecución JSLEE, instala y activa el servicio convergente, quedando así a la espera de su ejecución por parte del usuario.

5.6. Resumen

Este Capítulo presentó el mecanismo propuesto para la orquestación automática de servicios convergentes. Dicho mecanismo consta de cinco módulos que permiten generar una unidad desplegable JSLEE a partir de un proceso abstracto del servicio convergente. El primer módulo consta del entorno grafico de composición provisto por el proyecto Telcomp2.0, el cual genera procesos abstractos descritos en JSON; el segundo módulo traduce el proceso abstracto de JSON a un modelo propuesto de grafos; el tercer módulo convierte el grafo en una CPN utilizando el algoritmo 1, cuya base es transformar un nodo servicio en una terna lugar entrada – transición – lugar salida; el cuarto modulo convierte la CPN en un SBB.java que contiene el código fuente, para esta tarea utiliza la técnica de templates, los cuales son fragmentos genéricos de código que permiten la configuración de variables; finalmente, el quinto modulo compila el código y genera el SBB.class, además crea la DU que es enviada e instalada en el entorno de ejecución JSLEE.

Capítulo 6

6. Evaluación del Mecanismo

Este capítulo presenta la evaluación experimental realizada al mecanismo de orquestación automática propuesto. Para la metodología de evaluación fue tomada como referencia [89] la cual propone varias pruebas como la prueba de unidad, prueba funcional, prueba de integración y prueba de desempeño, no obstante para evaluar el mecanismo solo se utilizaron las prueba funcional y la prueba de desempeño, ya que con estas es suficiente para evaluar el cumplimiento del objetivo principal de este trabajo. La prueba funcional está encargada de verificar que el objetivo del mecanismo sea alcanzado por la ejecución del mismo, por lo cual se verifica la coherencia entre el servicio convergente diseñado y la ejecución del servicio. Por otra parte la prueba de desempeño tiene como objetivo medir la cantidad de recursos computacionales empleados por el mecanismo

Teniendo en cuenta las anteriores pruebas, fue necesario desarrollar un prototipo que permitiera realizarlas, por lo tanto la primera sesión de este capítulo describe el prototipo desarrollado, y posteriormente los resultados de dichas pruebas.

6.1. Descripción del prototipo

Esta sección describe los aspectos técnicos y funcionales del prototipo que implementa el mecanismo propuesto en este trabajo, denominado *JSLEE Orchestrator*, para la descripción emplea la metodología propuesta en [90] para la documentación de arquitecturas de software. Dicha metodología permite la descripción de sistemas software, teniendo en cuenta sus vistas más relevantes, vistas de Módulos, vistas de Componentes y Conectores, y vistas de Asignación, es

importante resaltar que dentro de cada vista solo serán presentados los diagramas más significativos para este trabajo de grado. Las vistas son descritas a continuación.

6.1.1. Vistas de módulos

Las primeras vistas corresponden a las unidades de implementación de software que proveen un conjunto relevante de funciones.

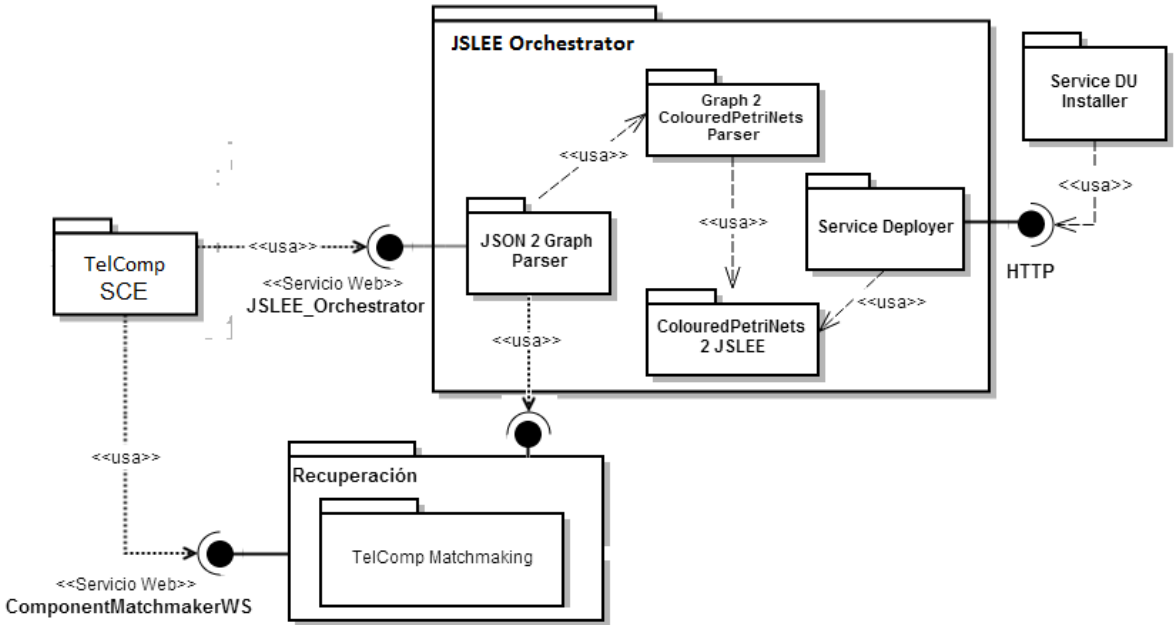


Figura 6.1 Vista de Descomposición del mecanismo

La Figura 6.1 presenta la vista de descomposición, la cual describe la organización del código en módulos, detallando cómo las funciones del sistema se distribuyen entre ellos. El proceso funcional de orquestación automática se encuentra en el módulo principal llamado JSLEE Orchestrator, el cual tiene internamente los componentes lógicos *JSON to GraphParser*, *Graph to CPN Parser* y *CPN to JSLEE*, y el proceso funcional de despliegue está representado por el componente lógico *Service Deployer*, que está organizando internamente en los componentes *SBB Compiler*, *DU Configuration* y *Service Loader*. A continuación son descritas las características de los modulos de la Figura 6.1

- ***JSON to Graph Parser*:** paquete encargado transformar un archivo JSON al modelo de grafos en lenguaje java. Para ello, dos subcomponentes lógicos intercambian funciones, el primero, *JSON To Java*, realiza la conversión de JSON a objetos JAVA, y el segundo, *Graph Generator*, transforma dichos objetos java al modelo de grafos propuesto en la sección 5.2.

- ***Graph to CPN*:** paquete encargado de transformar el grafo a un modelo de CPN en lenguaje java. Para ello, el subcomponente *CPN Generator* hace uso de los elementos lógicos *Nodes Analizer*, *Control Flow Analizer* y *Data Flow Analizer*, según lo indica el algoritmo 1.

- ***CPN to JSLEE*:** paquete encargado de transformar el modelo CPN en código ejecutable JAIN SLEE. Para ello, el subcomponente *SBB Generator* hace uso de los elementos lógicos *Transitions Analizer*, *Arcs Analizer* y *Places Analizer*, según lo indica el algoritmo 2.

- ***Service Deployer*:** paquete encargado de crear la DU.jar para el entorno de ejecución JSLEE. Para ello, tres sub módulos lógicos intercambian funciones, El *SBB Compiler* compila el código y genera el SBB.class, el *Service DU Configuration* crea la DU con todos sus archivos, y el *Service Loader*, envía el servicio al módulo *Service DU Installer* vía HTTP.

- ***Service DU Installer*:** paquete encargado de instalar la DU en el entorno JSLEE.

- ***JSLEE_Orquestrator*:** expone las funcionalidades del paquete de *Orquestación automática* como un servicio web *SOAP*, este módulo contiene la implementación del mecanismo.

- ***TelComp SCE*:** aplicación Web de creación del proceso abstracto JSON, que utiliza el módulo de orquestación invocando su servicio web *SOAP*.

- ***Recuperacion*:** Repositorio de servicios Web y Telco, que expone su información por medio de un servicio web *SOAP*.

6.1.2. Vistas de componentes y conectores

Estas vistas describen el mecanismo construido a partir de la especificación de elementos que presentan características de ejecución, tales como procesos, objetos, clientes o servidores, y registros de datos, los cuales incluyen elementos de interacción, tales como protocolos y flujos de información. Los elementos que hacen parte de este componente ejecutan actividades conjuntamente para que el servicio convergente esté disponible al usuario. La Figura 6.2 presenta el diagrama de actividades que orientan el proceso de este componente, en este diagrama se observa que el proceso de orquestación es llevado a acabo como se describió en el capítulo 5, además dicho proceso esta descrito a través de capas, las cuales corresponden a los módulos presentados en la figura 6.1. En las primeras tres capas es realizada la orquestación, donde se obtienen secuencialmente: un grafo en JSON, un grafo en java, una CPN y por ultimo un SBB.java. En la cuarta capa se realiza el despliegue, en el cual es generada la DU que es instalada en la capa sexta de ejecución. Es de resaltar que la capa cinco es el repositorio accedido por las capas de orquestación.

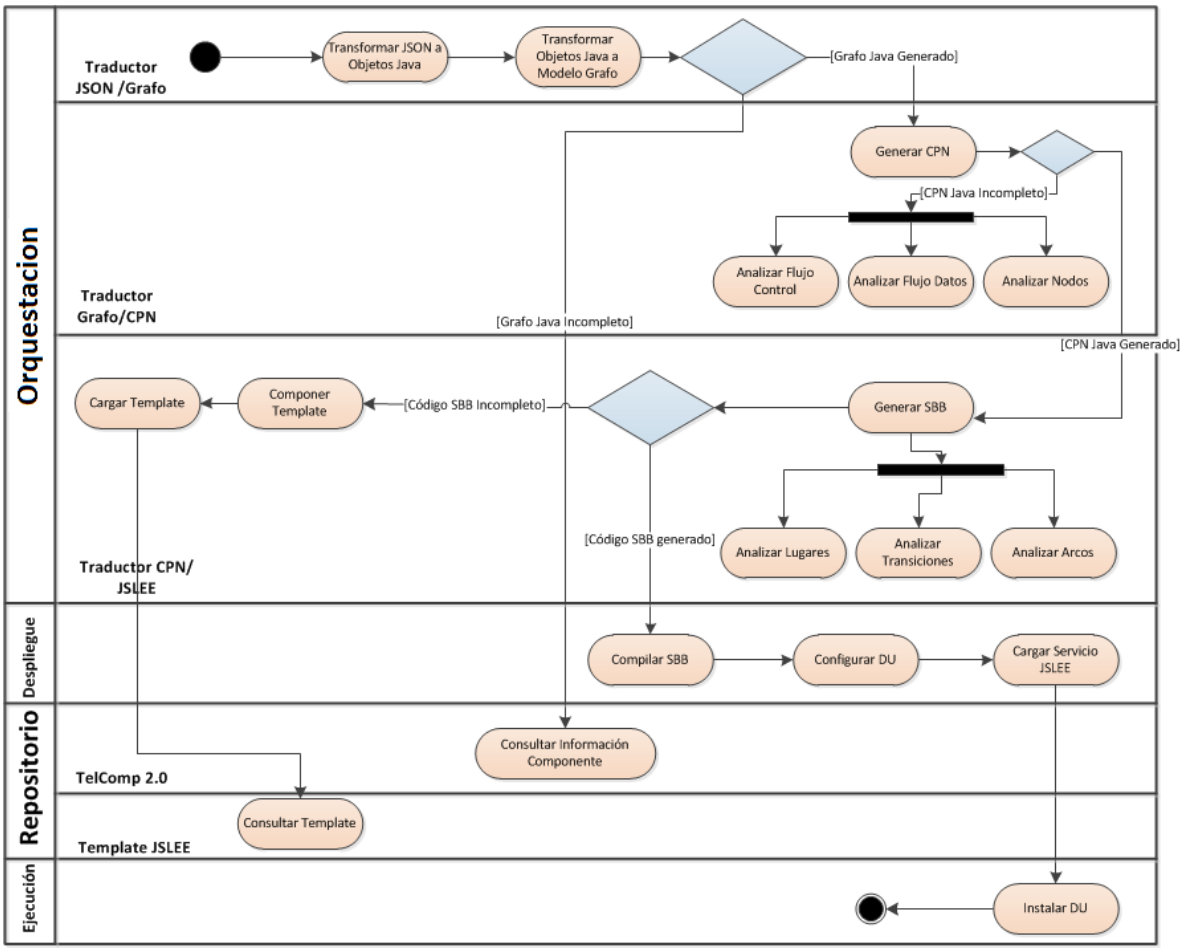


Figura 6.2 Diagrama de actividades del mecanismo

6.1.3. Vistas de asignación

Esta vista describe la relación entre la arquitectura software y el entorno sobre el cual se despliega, implanta y ejecuta la plataforma. La Figura 6.3 presenta el diagrama de despliegue del mecanismo. El equipo de la derecha (Contenedor Web, Equipo 2) contiene: el servidor Glassfish donde está instalado la aplicación Web de orquestación *JSLEE_Orchestrator*, el Repositorio JSLEE que corresponde a una carpeta del equipo que contiene los templates, librerías y eventos de los servicios Telco; y el servidor JSLEE con la aplicación Web de instalación de las DU. Por otra parte en el equipo del centro (Contenedor Web, Equipo 1) está la aplicación Web que contiene el entorno de creación de procesos abstractos *TelComp_SCE* y el repositorio de servicios que está compuesto por una base de datos en MySQL. En la izquierda (Cliente, Equipo externo) está el navegador Web que accede a la interfaz de creacion de procesos abstractos.

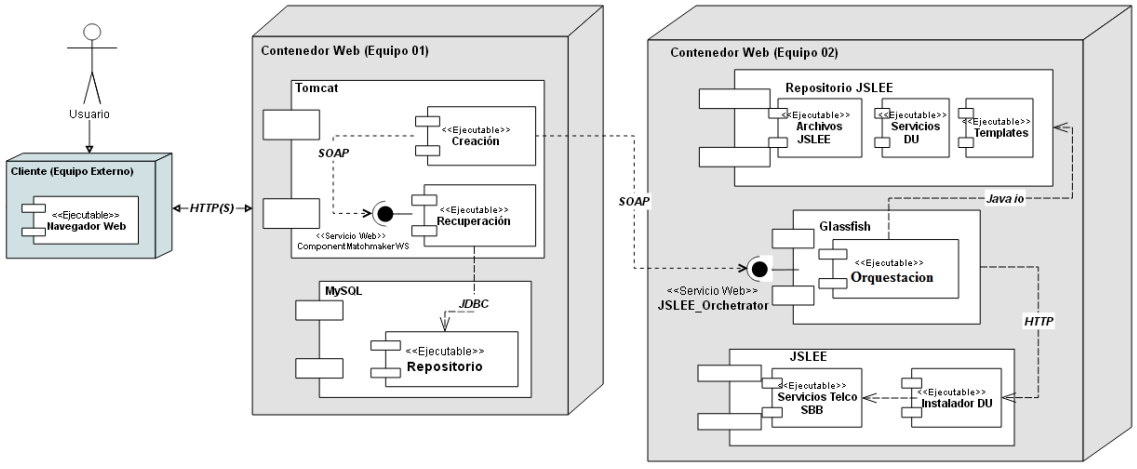


Figura 6.3 Diagrama de actividades del mecanismo

6.2. Evaluación funcional

Como se mencionó anteriormente la prueba funcional está encargada de verificar la coherencia entre el servicio convergente diseñado y la ejecución del servicio, realizando una comparación entre la traza generada por el servidor JSLEE en la

ejecución y los requerimientos establecidos por el diseñador a través del proceso abstracto. Por lo tanto fue necesario definir un servicio de prueba que pudiera ser orquestado automáticamente utilizando el prototipo.

En este sentido, para verificar la coherencia del módulo de orquestación automática se realizó una evaluación tipo encuesta cualitativa con base en la metodología propuesta en [89], donde los evaluadores participantes no representan a usuarios potenciales sino aquellos con experiencia en el manejo de métodos y procesos de orquestación de servicios en JSLEE.

6.2.1. Servicio de prueba

Para lleva a cabo la prueba funcional, se planteó el diseño de un servicio convergente denominado *LinkedInJobNotificator*, el cual envía una oferta de trabajo, actualmente registrada en el servicio de LinkedIn del usuario, a su red social twitter, al correo personal, y al celular a través de una llamada de voz, transformando el texto de la oferta de trabajo a un archivo de audio. Como presenta la Figura 6.4, el servicio inicia cuando el usuario activa el servicio desde su dispositivo móvil, el servicio accede a la base de datos de operador *(servicio DataAccess)* recupera el id de LinkedIn (*paso 1*); luego ejecuta el servicio web de LinkedIn que extrae información de una oferta laboral para el usuario (*paso 2*); esta información la envía por flujos paralelos (paso 3) mediante servicios de e-mail y Twitter, por lo que accede previamente a la base de datos para recuperar los id (*paso 4 y 5*), posteriormente, las respuestas de los servicios *Email Telco Service* y *Twitter-WS* se sincronizan para iniciar el servicio de TextToSpeech (*paso 6*), que transforma el texto en audio y realiza una llamada SIP al usuario, para lo que previamente debe obtener de la base de datos la SipUri (*pasos 7 y 8*). Es de resaltar que en este servicio de prueba permitió validar el funcionamiento del mecanismo cuando utiliza dos servicios Web (*LinkedIn, Twitter*), tres servicios Telco (*Email, Media Call*, *DataAccess*), y patrones de control (Trigger, Sequence, Parallel Split, Synchronization).

Por otra parte la Figura 6.5 muestra las dependencias de datos que requiere el servicio para obtener el comportamiento descrito, definiendo los parámetros de entrada y de salida para cada uno de los servicios que componen. En la figura cada arista está identificada con la notación [*in/out*];[*NombreVariable*]:[*TipoDato*]. *In/Out* indica si el dato es de entrada o salida; *NombreVariable*, es el nombre de la variable

que representa el dato de un servicio; *TipoDato*, es el tipo de dato enviado (String, entero, etc.). Por ejemplo entre los servicios LinkedInWs y TwitterWS existe una dependencia de datos, de manera que de LinkedInWs sale: [*out*];[*return*]:[*String*], es decir una variable de salida de tipo String que se llama *return*. Y a TwitterWS entra [*in*];[*text*]:[*String*], es decir una variable de entrada de tipo String que se llama *text*. Es de resaltar que tanto *return* como *text* corresponden a la oferta laboral.

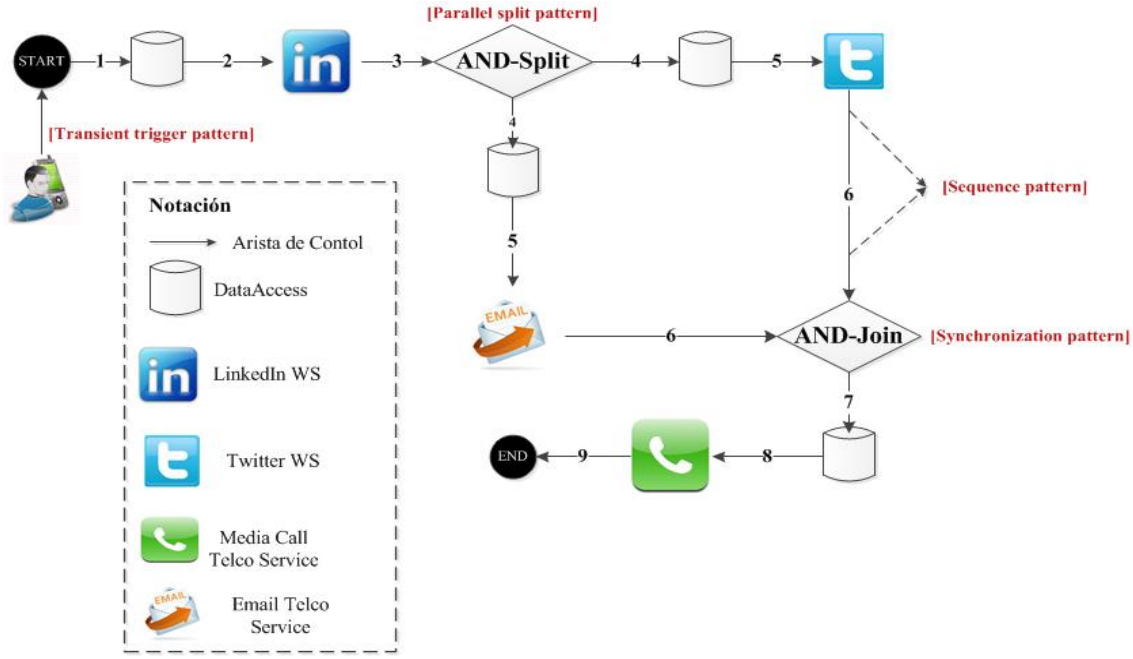


Figura 6.4 Flujo de control servicio LinkedInNotificator

6.2.2. Experimento prueba funcional

Como se mencionó anteriormente, los evaluadores tienen capacidades de un desarrollador con conocimientos en orquestación de servicios Web y de telecomunicaciones para entornos JSLEE. En ese sentido, se cuenta con 12 evaluadores pertenecientes al programa de Ingeniería en Electrónica y Telecomunicaciones, y la Maestría en Ingeniería Telemática de la Universidad del Cauca.

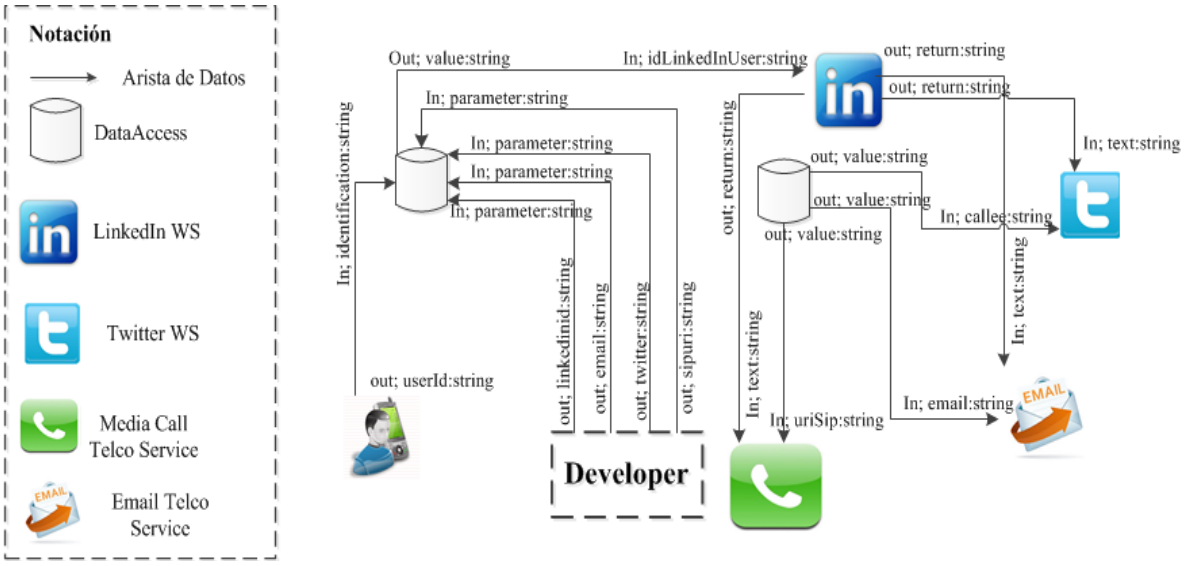


Figura 6.5. Dependencias de datos servicio LinkedInNotificator

La prueba fue realizada en tres partes, la primera consistió en una capacitación a los evaluadores en el manejo de los archivos JSON del proceso abstracto y el archivo Log de ejecución del servidor JSLEE Mobicents; en la segunda fue utilizada la herramienta TelComp SCE para crear y ejecutar el servicio LinkedInJobNotificator, ayudándose de un video tutorial que describe el procedimiento; la tercera consistió en realizar la encuesta a los evaluadores con base en los archivos JSON del proceso abstracto creado y el archivo Log generado en la ejecución del servicio.

La Figura 6.6 presenta el contenido del proceso abstracto en JSON, el cual fue obtenido con la herramienta TelComp SCE, después de obtener el JSON, fue utilizado el Tester para servicios Web que provee el IDE NetBeans con el fin de ejecutar el prototipo de orquestación automática. Posteriormente, en la Figura 6.7 es posible observar el Log del servidor JSLEE Mobicents, donde inicialmente se activa el servicio compuesto LinkedInNotificator, en seguida se observa como dispara y recibe los eventos de los servicios compuestos. Un factor importarte es que después de disparar los eventos, aparece el Log generado por los servicios componentes, indicando los parámetros de entrada que se le han pasado, por ejemplo al final el servicio MediaCall, recibe los parámetros: Oferta de trabajo, y david@127.0.0.1, los cuales provienen del servicio de LinkedIn y de la BD.

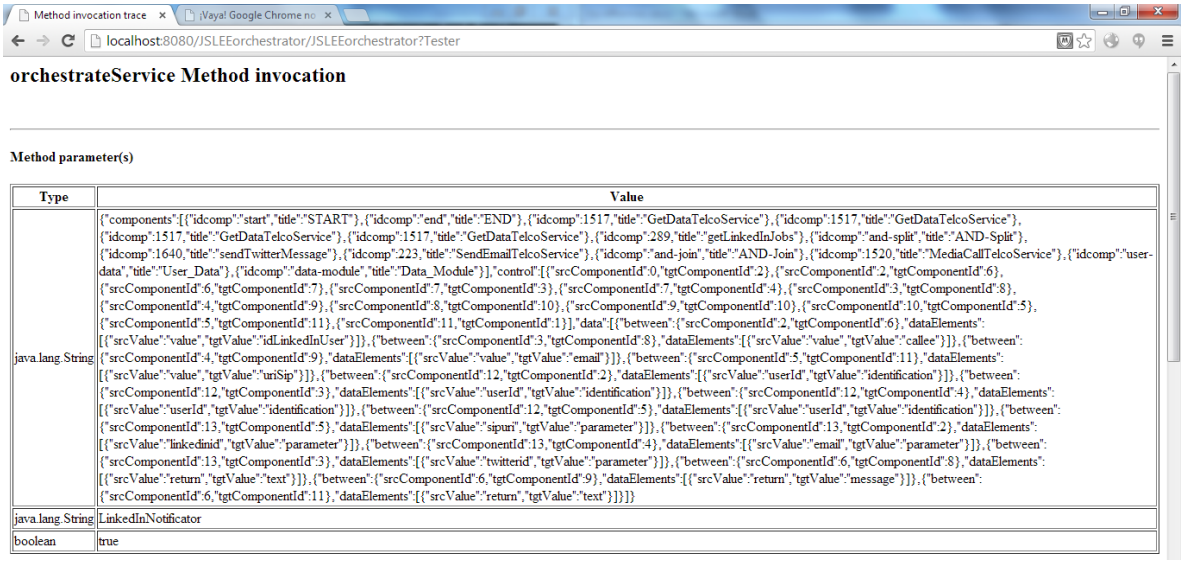


Figura 6.6 Solicitud de orquestación enviada al mecanismo propuesto

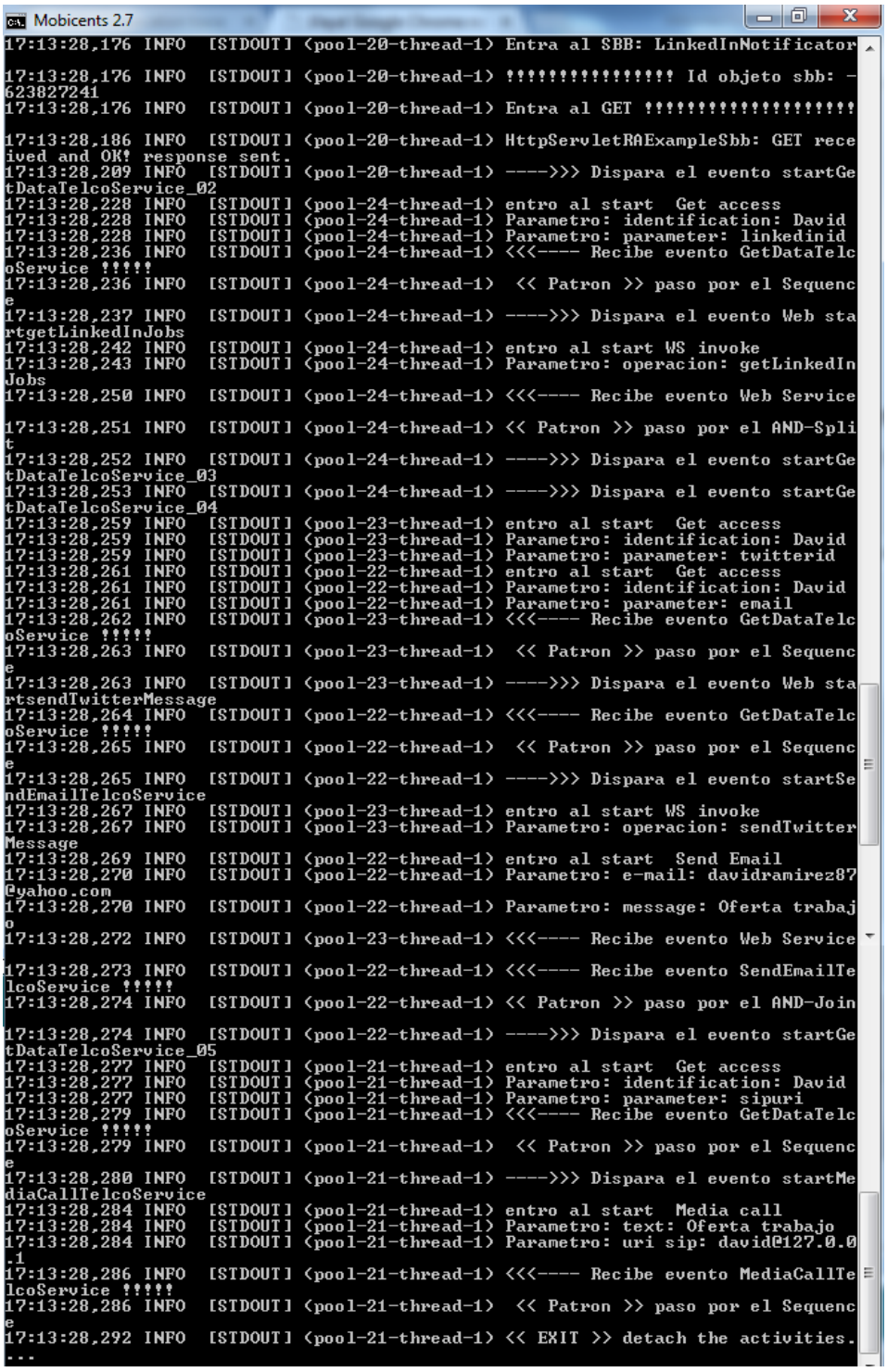


Figura 6.7. Log del entorno JSLEE Mobicents

Una vez efectuado el proceso de orquestación, los evaluadores realizaron la encuesta de coherencia, la cual fue diseñada según la metodología propuesta en [89] y es presentada a continuación.

1. Coherencia de los servicios componentes en la orquestación	
Descripción: Esta característica mide el grado de coherencia que tienen los servicios componentes, es decir si los servicios seleccionados en el proceso abstracto corresponden a los ejecutados en el servidor.	
Pregunta: ¿Qué grado de coherencia tienen los servicios componentes en la ejecución del servicio?	
Grado de Importancia: Altamente deseable	
Escala	
0 – 20	Nada
21 – 40	Muy Poco
41 – 60	Parcialmente
61 – 80	En alto Grado
81 – 100	Completamente
Nivel de soporte aceptable: Completamente.	

2. Coherencia del flujo de datos en la orquestación	
Descripción: Esta característica mide el grado de coherencia que tiene el flujo de datos de la orquestación, es decir si las entradas y salidas de los servicios en la ejecución, corresponden con las dependencias de datos definidas en el proceso abstracto.	
Pregunta: ¿Qué grado de coherencia tiene el flujo de datos en la ejecución del servicio?	
Grado de Importancia: Altamente deseable	
Escala	
0 – 20	Nada
21 – 40	Muy Poco
41 – 60	Parcialmente
61 – 80	En alto Grado
81 – 100	Completamente

Nivel de soporte aceptable: Completamente.

3. Coherencia del flujo de control en la orquestación	
Descripción: Esta característica mide el grado de coherencia que tiene el flujo de control de la orquestación, es decir si el orden de ejecución de los servicios componentes, corresponden con el flujo definido en el proceso abstracto.	
Pregunta: ¿Qué grado de coherencia tiene el flujo de control en la ejecución del servicio?	
Grado de Importancia: Altamente deseable	
Escala	
0 – 20	Nada
21 – 40	Muy Poco
41 – 60	Parcialmente
61 – 80	En alto Grado
81 – 100	Completamente
Nivel de soporte aceptable: Completamente.	

4. Coherencia entre el servicio diseñado y el servicio ejecutado	
Descripción: Esta característica mide el grado de coherencia de todo la orquestación, con el diseño definido en el proceso abstracto.	
Pregunta: ¿En general el resultado de la ejecución del servicio es coherente con el proceso abstracto?	
Grado de Importancia: Altamente deseable	
Escala	
0 – 20	Nada
21 – 40	Muy Poco
41 – 60	Parcialmente
61 – 80	En alto Grado
81 – 100	Completamente
Nivel de soporte aceptable: Completamente.	

Tabla 6.1. Encuesta evaluación funcional.

6.2.3. Resultados prueba funcional

Una vez los evaluadores realizaron la encuesta, fue efectuado un análisis de resultados consignado en la Tabla 6.2, para cada una de las preguntas el número que aparece en las columnas de calificación indica el valor asignado a dicha pregunta, la columna aceptación indica el porcentaje de calificaciones que superaron el nivel de aceptación definido en relación con cada uno de los criterios de aceptación. Los resultados presentados en la evaluación son altamente satisfactorios debido a que se obtuvo un 100% para cada una de las preguntas establecidas. En consecuencia, el mecanismo de orquestación automática cumple con los requerimientos establecidos por el proceso abstracto diseñado, logrando un nivel de confianza óptimo.

Pregunta	Importancia	Calificación					Aceptación (%)
		81 - 100	61 - 80	41 - 60	21 - 40	0 - 20	
1	AD	12					100
2	AD	12					100
3	AD	12					100
4	AD	12					100

Tabla 6.2. Resultados evaluación funcional.

Finalmente es posible concluir que el mecanismo de orquestación automático definido e implementado en este trabajo, soporta satisfactoriamente la composición de servicios en entornos convergentes, como se planteó en el objetivo general de este trabajo.

6.3. Evaluación de desempeño

La evaluación de desempeño tiene como objetivo medir la cantidad de recursos computaciones empleados por el mecanismo propuesto. Para esto se tomó como referente los tiempos de procesamiento en cada módulo del mecanismo, porcentaje de CPU y memoria Dinámica[7] como métricas de desempeño. De esta manera, fueron realizadas las pruebas de escalabilidad donde se aumenta la complejidad de la

[7] La memoria Dinámica es la memoria que se va ocupando a medida que se crear objetos durante la ejecución de un programa.

petición enviada [91], la cual está dada por el incremento del número de servicios componentes en el proceso abstracto; y pruebas de estrés donde el objetivo es encontrar el punto de quiebre del mecanismo.

6.3.1. Escenario de prueba.

Teniendo en cuenta las ventajas de la Arquitectura Orientada a Servicios (SOA) el mecanismo de orquestación se empaqueto como un servicio web. En ese sentido, fue utilizada la herramienta SOAPUI [92] para la configuración de los parámetros de la prueba. Esta herramienta es de código abierto y distribuida bajo licencia tipo GNU LGPL, además, facilita la rápida creación de test avanzados de desempeño y evaluación de servicios web [93].

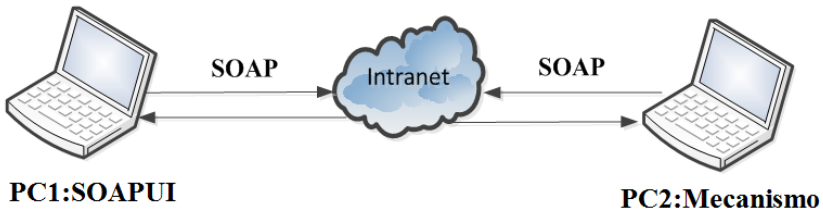


Figura 6.8. Escenario de prueba de desempeño.

La Figura 6.8 ilustra el escenario de la prueba, el cual consta de dos PC, de la herramienta SOAPUI y de una conexión de red entre los equipos contendedores del mecanismo y de la herramienta de simulación.

El PC1 que contiene la herramienta SOAPUI corre sobre un sistema Windows core i5 de 4GB de RAM; el PC2 contiene el mecanismo de orquestación corre sobre un sistema Linux Ubuntu 12.04, core i5 y 4GB de RAM.

6.3.2. Experimento prueba desempeño

- **Prueba de escalabilidad**

La escalabilidad hace referencia a incrementar la complejidad de la solicitud [91]. En ese sentido, la estructura del proceso abstracto se compone de ***n*** servicios componentes, donde ***n*** varía de dos (2) en dos (2) hasta un límite de 20 servicios. Los parámetros de configuración son los siguientes:

- Número de hilos: 1
- Total de ejecuciones: 8
- Tiempo entre ejecuciones: 3 segundos
- Número de servicios componentes: *n*, donde *n* = {2, 4, 6,…,20}

El número de hilos hace referencia a un sólo proceso abstracto con *n* nodos; los nodos van aumentando de dos en dos hasta un límite de 20 nodos dentro de la estructura del servicio. El Total de ejecuciones hace referencia al número de veces que se realiza la misma prueba. Esto se hace con el objetivo de encontrar una medida más precisa del tiempo empleado. Para ello, se calcula la media geométrica del conjunto de medidas obtenidas, dando como resultado un valor medio más significativo. El tiempo entre ejecuciones hace referencia al tiempo de espera entre cada una de las veces que se realiza la prueba, para este caso se ha dispuesto un tiempo de 3 segundos, suficiente para evitar traslapes entre solicitudes. El número de servicios componentes hace referencia a la cantidad de servicios que componen el proceso abstracto.

La prueba de desempeño cuenta con 10 pruebas de escalabilidad. Para cada una de ellas el total de ejecuciones, el número de hilos y el tiempo entre ejecuciones se mantiene; el número de servicios atómicos varía con base en el intervalo de incremento definido.

➢ **Prueba de estrés**

El objetivo de esta prueba es encontrar el punto de quiebre del mecanismo, el cual es el punto límite de carga donde el sistema colapsa [91]. En este sentido, se configura un proceso abstracto con ***n*** servicios atómicos, donde ***n*** varía de cuatro (4) en cuatro (4) hasta un límite de 12 servicios, y se aumenta el número procesos abstractos que se envían de manera simultánea al mecanismo de orquestación. Los parámetros de configuración son los siguientes:

- Número de hilos: *p*, donde *p* = {1,2,3,4,…,k}
- Número de servicios atómicos: *n*, donde *n* = {4, 8,12}

El número de hilos hace referencia a la cantidad de procesos abstractos que se envían simultáneamente para ser procesados. La cantidad de CC varía de uno (1) en

uno (1) hasta un límite de *k*, donde *k* es el punto de ruptura del mecanismo. Para cada *n* servicios atómicos se determina el punto de ruptura *k*. El límite de 12 servicios se seleccionó teniendo en cuenta la cantidad de servicios usados en el proceso de composición de servicios de aproximaciones como [10][50][22].

6.3.3. Resultados prueba de desempeño

- **Resultados prueba de escalabilidad**

La Figura 6.9 muestra los resultados de este experimento en el que se calculó la métrica de tiempo con relación al número de servicios componentes.

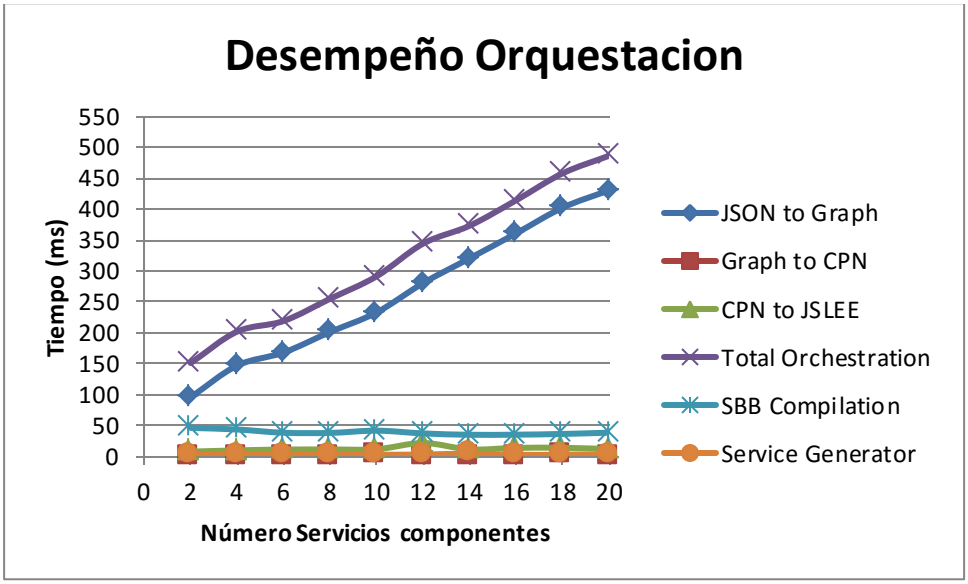


Figura 6.9. Resultados prueba de escalabilidad.

Como muestra la Figura 6.9, el tiempo total que toma el mecanismo de orquestación automática no supera los 500 ms para procesos abstractos de menos de 20 servicios, lo que demuestra un uso eficiente de recursos computacionales del mecanismo. Este tiempo de procesamiento está representado principalmente por el módulo de traducción del JSON a Grafos, esto se presenta debido a que en este módulo se accede a un repositorio mediante servicios web, lo que exige una mayor cantidad de recursos computacionales y de red. No obstante esto se podría mejorar creando una conexión directa con la base de datos del repositorio sin utilizar servicios web. Por otra parte los módulos de grafos a CPN y CPN a JSLEE,

demuestran un excelente rendimiento teniendo en cuenta que el PC de prueba no tiene características hardware de un servidor.

Teniendo en cuenta enfoques como [10][50][22] donde el número de servicios componentes de un servicio compuesto no superan los 10 servicios dentro del dominio Telco, el mecanismo de orquestación automática tiene un comportamiento óptimo y eficiente para este intervalo, además, diseñar un servicio compuesto con gran cantidad de servicios atómicos, es en la práctica, una tarea compleja de realizar, principalmente, por la dificultad en el control de las interacciones de las entidades que forman parte del modelo de negocio del servicio. Finalmente, es posible concluir que el mecanismo automático de orquestación hace uso eficiente de los recursos computacionales, específicamente, en el tiempo empleado en los módulos de grafos a CPN y CPN a JSLEE.

- **Resultados prueba de estrés**

Para este experimento se tomaron como métricas de desempeño: el porcentaje de CPU, memoria dinámica y cantidad de procesos abstractos que son desplegados con éxito. En consecuencia, se obtuvieron resultados mostrados en las Figuras 6.10, 6.11 y 6.12.

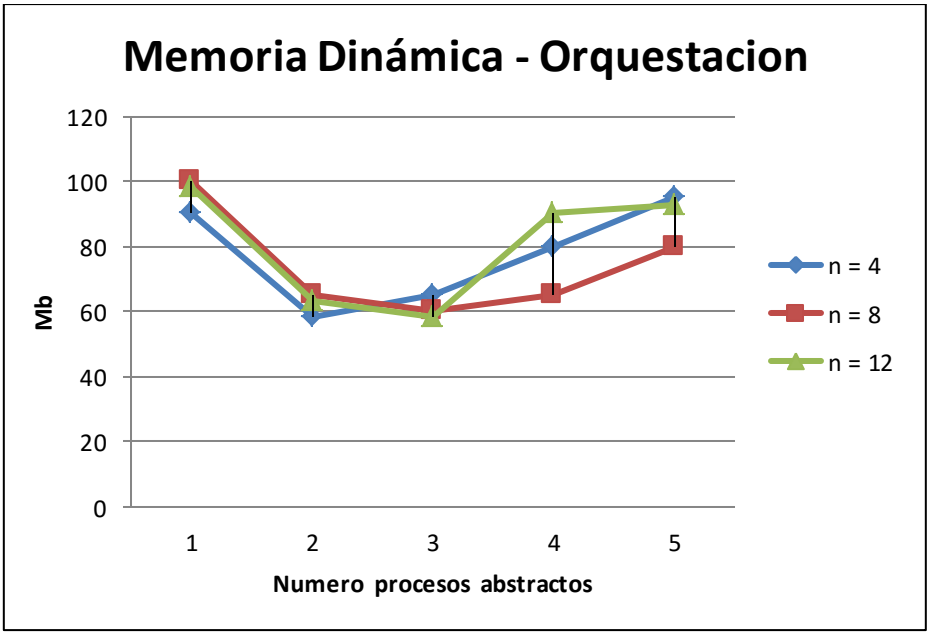


Figura 6.10. Consumo memoria orquestación.

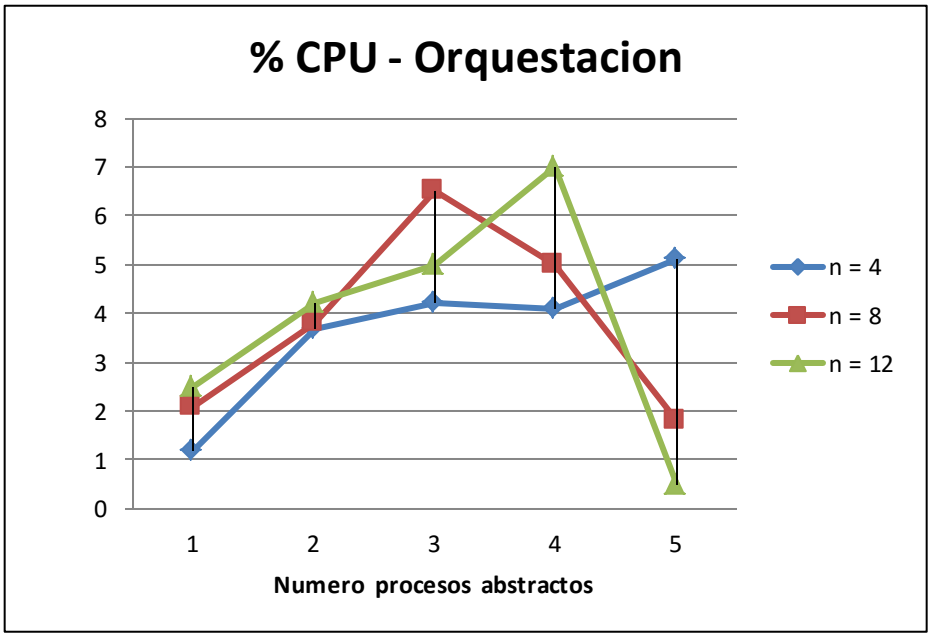


Figura 6.11. % de CPU orquestación.

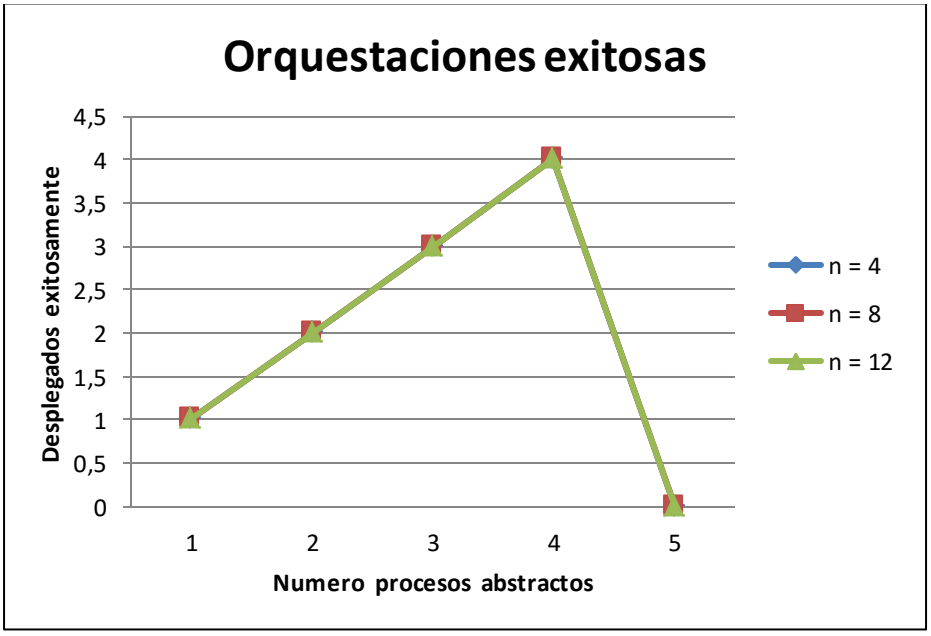


Figura 6.12. Tasa de orquestaciones exitosas.

De la Figura 6.12 se obtiene que el punto de quiebre del mecanismo es de 5 procesos abstractos independientemente de la complejidad de la estructura del servicio (4, 8 o 12 servicios componentes). Esto se debe a la saturación que produce el algoritmo de orquestación sobre el envío de peticiones simultáneas al repositorio de la plataforma TelComp2.0. Sin embargo, en la práctica dos o más procesos difícilmente serán enviados de manera simultánea para ser orquestados, ya que el diseño y creación de un servicio, a través de un SCE, requiere de un análisis de los requerimientos de negocio, donde el tiempo consumido por éste difícilmente coincidirá con el tiempo de diseño y creación de otro servicio.

Por otra parte, en el punto de ruptura se presenta un consumo elevado en la memoria dinámica (Figura 6.10) y un bajo uso de CPU del sistema (Figura 6.11), este último se debe a que la Máquina Virtual Java queda en estado inactivo producto del colapso del algoritmo de adaptación/despliegue.

Es importante resaltar que para modelos de composición automática de servicios centrados en lenguaje natural, la lógica de conexión al repositorio deberá implementarse al interior del módulo de adaptación/despliegue. Esto mejorará el desempeño del sistema, reaccionando de manera óptima a peticiones de composición simultánea.

Finalmente, es posible concluir que el comportamiento del algoritmo en el intervalo de 1 a 4 CC procesados simultáneamente es óptimo y eficiente; el consumo de memoria dinámica y de CPU muestra un uso adecuado de los recursos computacionales del sistema, sin embargo es recomendable que la conexión al repositorio sea implementada al interior del módulo de orquestación, lo cual mejorará el desempeño del sistema, reaccionando de manera óptima a peticiones de orquestación simultánea.

6.4. Resumen

Este capítulo presento la evaluación experimental del mecanismo para orquestación automática de servicios convergentes. La evaluación fue dividida en tres secciones, la primera describió el prototipo que implemento el mecanismo propuesto, por medio de las vistas de Módulos, vistas de Componentes y

Conectores, y vistas de Asignación. La segunda sección presento la evaluación funcional, la cual verifico la coherencia entre el servicio convergente diseñado y la ejecución del servicio, comparando la traza de ejecución en el servidor JSLEE y el comportamiento del proceso abstracto. Para la evaluación funcional fue obtenida una coherencia del 100% y se concluyó que el mecanismo cumple satisfactoriamente con su objetivo. Finalmente la tercera sección presento la evaluación de desempeño que tenía como objetivo medir la cantidad de recursos computacionales empleados por el mecanismo, tomando como métricas la memoria dinámica y el porcentaje de CPU utilizados, y el número orquestaciones exitosas cuando el mecanismo era utilizado con varios procesos abstractos simultáneamente. Los resultados de la prueba de desempeño fueron satisfactorios hasta el punto de orquestar 5 procesos simultáneamente, este valor límite se debió a la saturación de la conexión con el repositorio.

Capítulo 7

7. Conclusiones

A continuación se exponen las principales conclusiones del desarrollo del presente trabajo de grado.

- En este trabajo de grado fue propuesto un mecanismo para automatizar la orquestación de servicios sobre entornos convergentes JSLEE. Este mecanismo está basado en grafos y CPN para representar formalmente la síntesis y orquestación respectivamente, esto facilito en gran medida el modelado de cada fase y la aplicación de los algoritmos de análisis y traducción de una fase a otra. Además, las CPN permitieron plantear fácilmente un modelo de orquestación de servicios utilizando la especificación JSLEE.

- El método propuesto para definir el conjunto de patrones del flujo de control y de datos permitió analizar un conjunto grande de servicios con modelos complejos de manera fácil y rápida. Este método dió como resultado final 13 patrones del flujo de control comunes presentes en servicios convergentes. Por otra parte, con el análisis realizado a los patrones de datos se obtuvo como resultado un conjunto de 16 patrones de datos, relacionados con las capacidades del entorno de ejecución de servicios JSLEE. La aplicación de estos patrones del flujo de ejecución dentro del mecanismo agiliza en gran medida el proceso de orquestación, puesto que estos se representan como CPN predefinidas con su respectivo templates de código, evitando así el uso de nuevos algoritmos para lograr el comportamiento deseado.

- Una de las características importantes del mecanismo propuesto es que funciona en tiempo de diseño para no afectar el rendimiento en la ejecución del

servicio, esta característica aporta mejoras respecto a los trabajos relacionados. Sin embargo la evaluación de desempeño demostró que aunque el tiempo de procesamiento aumenta en forma lineal con el número de servicios componentes, para un valor de 20 servicios no supera los 500 ms, en este sentido el mecanismo aun en tiempo de diseño es bastante eficiente, por lo tanto podría ser utilizado en ejecución bajo modelos de composición dinámica de servicios.

- Este mecanismo brinda ventajas a los operadores de telecomunicaciones , ya que es permite generar servicio convergentes en muy poco tiempo y de una manera fácil a partir de un proceso abstracto, evitándoles la difícil tarea de orquestar manualmente servicios sobre entornos JSLEE, lo cual implica un amplio conocimiento técnico y experiencia, en instalación, configuración y desarrollo.

7.1. Resultados

A continuación se presenta un resumen de los principales resultados obtenidos con la ejecución del presente trabajo de grado:

- **Representaciones formales para servicios convergentes:** con base en un proceso de análisis y selección de las representaciones formales para modelar tanto el proceso abstracto como la orquestación de los servicios convergentes. Se determinó que los procesos abstractos deben ser modelados utilizando grafos, y los servicios ejecutables (orquestación) deben ser modelados por medio de redes de Petri coloreadas.

- **Patrones del flujo de ejecución:** La selección de un conjunto de patrones del flujo de control y de datos, específicos para representar la estructura de los servicios convergentes. El conjunto consta de 13 patrones de control y 16 patrones de datos. Como resultado también se incluye el método propuesto para la detección y selección de los patrones.

- **Catálogo de servicios:** Un catálogo de servicios presentes en un entorno Telco 2.0, el cual comprende un total de 73 servicios (23 Telco, 38 Web, 12 convergentes y servicios de seguridad y acceso a datos) obtenidos mediante el

estudio de organizaciones de estandarización (3GPP, ITU, ETSI-TISPAN, OneApi), plataformas para despliegue de servicios (Mobicents, Rhino) y algunos operadores de telecomunicaciones (Vodafone, Tigo, Claro).

- **Algoritmos de orquestación:** Un algoritmo para pasar de un proceso abstracto descrito en JSON al modelo de grafos propuesto, y un algoritmo para pasar el proceso abstracto descrito en el modelo de grafos a una CPN que representa la orquestación de servicios convergentes, dichos algoritmos están basados en patrones del flujo de ejecución y representaciones formales.

- **Mecanismo de orquestación automática:** Un mecanismo para la orquestación automática en tiempo de diseño, de procesos abstractos sobre un entorno convergente JAIN SLEE.

- **Prototipo de orquestación:** Un prototipo que implementa el mecanismo de orquestación automática, el cual exporta su funcionalidad mediante servicios Web. este prototipo funciona utilizando el repositorio y la interfaz e composición provista por el proyecto TelComp2.0. Además fueron desarrollados tres servicios de prueba utilizando el prototipo. El proceso de creación de dichos servicios fue documentado en video tutoriales.

- **Resultados de la evaluación:** Resultados experimentales sobre la evaluación del mecanismo de orquestación automática propuesto. Consta de la evaluación funcional, y la evaluación de desempeño, la cual ofrece valores de referencia sobre tiempos de ejecución, memoria y CPU utilizada, y tasa de orquestaciones exitosas.

Adicionalmente, fueron obtenidos los siguientes resultados complementarios:

- **Soporte a proyectos:** soporte al proyecto TelComp2.0 [25], el cual está enfocado en la recuperación y composición de componentes complejos para la creación de servicios Telco 2,0. Este proyecto es financiado por Colciencias y la universidad del Cauca. El presente trabajo de grado contribuyo directamente en los paquetes de trabajo WP4 y WP5, aportando su referente teórico, mecanismo, prototipo y evaluación experimental.

- **Beca joven investigador Colciencias:** en el marco de este trabajo de grado fue desarrollada la beca de Joven Investigador e Innovador Colciencias 2011, otorgada en la convocatoria 525, y financiada por Colciencias y la Universidad del Cauca durante mayo del 2012 a mayo del 2013 (código proyecto 3458).

- **Publicaciones:** hasta la fecha se han realizado las siguientes publicaciones.

  - "Control-Flow Patterns in Converged Services" presentado en la conferencia: "Service Computation 2012: The Fourth International Conferences on Advanced Service Computing" realizada en la ciudad de Niza, Francia. Artículo publicado en Agosto de 2012 con Copyright (c) IARIA, 2012. ISBN: 978-1-61208-215-8.

  - "Automatic Orchestration of Converged Services in JSLEE environtments", publicado en la revista Tecnura de la Universidad Distrital Francisco José de Caldas, en el Vol 19, No 46, en diciembre del 2015, con ISSN: 0123-921X. Indexada en Publindex como categoría A2.

  - "Recuperación y Composición de Servicios Convergentes en el proyecto TelComp2.0", presentado en el VI Seminario Nacional de Tecnologías Emergentes en Telecomunicaciones y Telemática TET 2013. Teatro Guillermo León Valencia. Popayán – Cauca. Octubre de 2013.

- **Asesoría a trabajo de grado:** durante el desarrollo de este trabajo, se realizada la asesoría en modalidad de co-director del trabajo de pregrado titulado: *Composición de servicios convergentes mediante el uso de patrones del flujo de ejecución en un entorno Telco 2.0,* el cual fue desarrollado por los Ingeniero Gustavo Enríquez y Andrés Benavides. Este trabajo fue merecedor de la mención de Honor por parte de la FIET.

- **Pasantía de investigación:** durante el desarrollo de este trabajo, fue realizada una pasantía corta de investigación en la Universidad Politécnica de Madrid UPM, ejecutando la propuesta titulada: *Comparación de rendimiento entre entornos de ejecución del TelComp2.0 y Motores BPEL,* a cargo del Dr. Juan

Carlos Yelmo. La estancia se llevó a cabo entre el 12 de noviembre y el 20 de diciembre del 2013.

- **Pasantía empresarial:** con el fin de validar el mecanismo propuesto de orquestación automática, por parte de un operador de telecomunicaciones, fue realizada una pasantía en la empresa EMTEL S.A E.S.P de Popayán, la cual se llevó a cabo entre el 25 de junio y el 24 de julio de 2013. Esta pasantía estuvo enmarcada dentro del proyecto TelComp2.0 mencionado anteriormente. Durante la pasantía se implanto y probó funcionalmente dentro de la red de telecomunicaciones del operador, el prototipo desarrollado en este trabajo de grado, con el cual se construyeron tres servicios convergentes.

7.2. Trabajos futuros

A continuación se listan los trabajos futuros que se plantean como continuación del presente proyecto de investigación.

- Integración del este mecanismo con propuestas que automaticen la fase de síntesis, de manera que las dos fases de la composición serian automática, permitiendo por ejemplo que con una petición en lenguaje natural se genere automáticamente un servicio convergente en entornos JSLEE.

- Integrar del mecanismo, con sistemas de soporte de operaciones (*Operation Support Systems*, OSS) y sistemas de soporte de negocio (*Business Support Systems*, BSS). De esta manera el mecanismo seria desplegado al interior de un operador de telecomunicaciones, para ofrecer a sus usuarios la creación de servicios convergentes. Por lo tanto, se abordarían aspectos como la gestión sobre entrega y aprovisionamiento de estos servicios, así como la gestión de su ciclo de vida y aspectos no funcionales como la provisión de calidad de servicio.

- Extensión del prototipo implementado para el soporte de otros tipos de servicios y recursos de la Web, ya que el prototipo solo utilizo servicios Web SOAP), pero actualmente hay una tendencia a utilizar los servicios RESTful o API REST, los mashups y contenidos multimedia. Lo anterior implicaría modificaciones

pertinentes en los templates que definen las interfaces y el repositorio de servicios.

- Extensión de los patrones del flujo de ejecución, puesto que en este trabajo solo se consideró las perspectivas de control y datos por ser las más básicas. Se podría utilizar patrones de excepciones, los cuales podrían ser la base de nuevas propuestas de investigación alrededor de la reconfiguración de servicios convergentes en tiempo de diseño.

Bibliografía

[1] A. Martinez, C. B. Zorita, A. M. L. Martin, C. G. Morchon, L. Calavia, J. A. Perez, and J. Caetano, "TelecomI+D03: New Business Models: User Generated Services," *IEEE Lat. Am. Trans.*, vol. 7, no. 3, pp. 395–399, 2009.

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

[3] ITU-T, "ITU-T Recommendation Y.2013 - Converged services framework funtional requirements and architecture," 2006.

[4] O. Chudnovskyy, F. Weinhold, H. Gebhardt, and M. Gaedke, "Integration of Telco Services into Enterprise Mashup Applications," in *Current Trends in Web Engineering*, 2011, pp. 37–48.

[5] C. R. Johnson, Y. Kogan, Y. Levy, F. Saheban, and P. Tarapore, "VoIP reliability: a service provider's perspective," *IEEE Communications Magazine*, vol. 42, no. 7, pp. 48–54, 2004.

[6] H. Haran, "Time to Market Research: Highlights and Key Findings," no. April. AMDOCS-Customer Experience Systems Innovation, pp. 1–25, 2011.

[7] G. Bond, E. Cheung, and R. Levenshteyn, "Unified Telecom and Web Services Composition : Problem Definition and Future Directions," in *IPTCOMM'09*, 2009, p. 13.

[8] JCP, "JAIN SLEE v 1.1," *JSR 240*, 2008. [Online]. Available: http://jcp.org/en/jsr/detail?id=240.

[9] T. Dinsing, G. A. P. Eriksson, I. Fikouras, K. Gronowski, and R. Levenshteyn, "Service composition in IMS using Java EE SIP servlet containers," no. 3. Ericsson, pp. 92–96, 2007.

[10] M. E. Goncalves da Silva, "User-centric Service Composition - Towards Personalised Service Composition and Delivery," PhD thesis, University of Twente, Enschede, The Netherlands, 2011.

[11] A. Dan, R. D. Johnson, and T. Carrato, "SOA service reuse by design," *Proc. 2nd Int. Work. Syst. Dev. SOA Environ. SDSOA 08*, pp. 25–28, 2008.

[12] N. Trcka, W. van der Aalst, and N. Sidorova, "Analyzing Control-Flow and Data-Flow in Workflow Processes in a Unified Way," Technische Universiteit Eindhoven, Eindhoven, Computer Science Report No. 08-31, 2008.

[13] U. Küster, M. Stern, and B. König-Ries, "A classification of issues and approaches in automatic service composition," *Intl Work. WESC*, vol. 5, pp. 25–35, 2005

[14] D. Berardi, G. De Giacomo, L. Sapienza, and B. Bozen, "Automatic Composition of Process-based Web Services : a Challenge," *Analysis*, vol. 531, pp. 1–3, 2005.

[15] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," *Semant. Web Serv. Web Process Compos.*, vol. 3387, no. 2, pp. 43–54, 2005.

[16] OpenCloud, "Opencloud Dev Portal," 2011. [Online]. Available: https://developer.opencloud.com/devportal/display/OCDEV/Home.

[17] Almira Labs, "Umbra Designer," 2015. [Online]. Available: http://www.almiralabs.com/products/umbra/designer.html. [Accessed: 01-Sep-2015].

[18] O. F. Díaz, C. P. Ortiz, V. E. Sentí, and J. P. F. Rodríguez, "Los Mashups: aplicaciones compuestas de la Web 2.0, exposición de caso. (Spanish)," *Ciencias la Inf.*, vol. 43, no. 3, pp. 43–48, 2012.

[19] T. Hornung, A. Koschmider, and J. Mendling, "Integration of heterogeneous BPM Schemas : The Case of XPDL and BPEL," *Transition*, vol. 6, pp. 23–26, 2006.

[20] J. Í. Martínez and V. P. Ferragud, "NODE.JS Do's and Don'ts," Universitat Politècnica de València, 2015.

[21] D. Zhu, Y. Zhang, B. Cheng, B. Wu, and J. Chen, "HSCEE : A Highly Flexible Environment for Hybrid Service Creation and Execution in Converged Networks," *J. Converg. Inf. Technol.*, vol. 6, no. 3, pp. 264–276, 2011.

[22] M. Femminella, E. Maccherani, and G. Reali, "Workflow Engine Integration in JSLEE AS," *IEEE Commun. Lett.*, vol. 15, no. 12, pp. 1405–1407, Dec. 2011.

[23] M. Drewniok, M. Maresca, S. Rego, J. Sienel, and M. Stecca, "Experiments and performance evaluation of Event Driven Mashups," in *IEEE Symposium on Computers and Communications, Sousse*, 2009, pp. 19–22.

[24] A. Lehmann, T. Eichelmann, U. Trick, R. Lasch, B. Ricks, and R. Toenjes, "TeamCom: A Service Creation Platform for Next Generation Networks," in *2009 Fourth International Conference on Internet and Web Applications and Services*, 2009, pp. 12–17.

[25] J. C. Corrales and E. Al, "Proyecto Telcomp2.0 (Financiado mediante contrato RC 458-2011 celebrado entre la Fiduciaria Bogotá, la Universidad del Cauca y COLCIENCIAS)," 2010.

[26] W. van der Aalst and A. ter Hofstede, "Workflow Patterns," *Workflow Patterns Initiative*. [Online]. Available: http://www.workflowpatterns.com/documentation/.

[27] N. Russell, A. H. M. Ter Hofstede, and W. M. Van Der Aalst, "newYAWL: Specifying a Workflow Reference Language using Coloured Petri Nets," *Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*. Department of Computer Science, University of Aarhus, Denmark, pp. 107–126, 2007.

[28] N. Russell and A. H. M. Hofstede, "WORKFLOW CONTROL-FLOW PATTERNS A Revised View," BPM Center Report BPM-06-22, 2006.

[29] N. Russell, A. H. M. Hofstede, D. Edmond, and W. M. P. Van Der Aalst, "Workflow Data Patterns: Identification, Representation and Tool Support," *Russell J. Bertrand Russell Arch.*, vol. 3716, pp. 353–368, 2005.

[30] B. Benatallah, R. M. Dijkman, M. Dumas, and Z. Maamar, "Service Composition: Concepts, Techniques, Tools and Trends," in *Service-oriented Software System Engineering: Challenges and Practices*, Hershey: Idea Group Publishing, 2005, pp. 48–65.

[31] A. Martens, "Consistency between executable and abstract processes," in *2005 IEEE International Conference on eTechnology eCommerce and eService*, 2005, pp. 60–67.

[32] Oasis, "Web Services Business Process Execution Language Version 2.0," *oasisopenorg*, vol. 11, no. April. OASIS (Organization for Advancement of Structured Information Standards), pp. 1–264, 2007.

[33] OpenESB, "Using the WSIT Mutual Certificates Security Mechanism with the HTTP BC," 2008. [Online]. Available: http://wiki.open-esb.java.net/Wiki.jsp?page=HTTPBCWSITMutualCerts.

[34] M. Espinoza-mejia and P. Alvarez, "El ciclo de vida de un servicio Web compuesto : virtudes y carencias de las soluciones actuales," pp. 1–19, 2007.

[35] T. Eichelmann, W. Fuhrmann, U. Trick, and B. Ghita, "Enhanced concept of the TeamCom SCE for automated generated services based on JSLEE," in *Eighth International Network Conference (INC)*, 2010.

[36] L. Burgy, L. Caillot, C. Consel, F. Latry, and L. Réveillère, "A Comparative Study of SIP Programming Interfaces," in *International Conference on Intelligence in service delivery Networks*, 2004, p. 5.

[37] I. OMELETTE, "D2.3 Final specification of Mashup description language and Telco mashup architecture," 2013.

[38] WSO2, "WSO2 Developer Studio," 2015. [Online]. Available: http://wso2.com/products/developer-studio/. [Accessed: 02-Oct-2015].

[39] J. A. Caicedo and J. C. Corrales, "Implementación de un mecanismo de despliegue automático de servicios convergentes en entornos JSLEE," *Rev. S&T*, vol. 12, no. 28, pp. 91–111, 2014.

[40] Ericsson, "Ericsson converged service studio," 2015. .

[41] Ur. Technologies, "Converged services framework," 2015..

[42] C. Chrighton, D. T. Long, and D. C. Page, "JAIN SLEE vs SIP Servlet Which is the best choice for an IMS application server?," in *2007 Australasian Telecommunication Networks and Applications Conference*, 2007, pp. 448–453.

[43] Oracle- Community Development of java Technology Specifications, "JSR 240: Jain Slee v 1.1," 2012..

[44] S. Bessler, J. Zeiss, R. Gabner, and J. Gross, "An Orchestrated Execution Environment for Hybrid Services," in *Kommunikation in Verteilten Systemen (KiVS)*, 2007, pp. 77–88.

[45] G. J. G. Jie, C. B. C. Bo, C. J. C. Junliang, and Z. L. Z. Lei, "A Template-Based Orchestration Framework for Hybrid Services," in *2008 Fourth Advanced International Conference on Telecommunications*, 2008, pp. 315–320.

[46] C. Bo, Z. Yang, Y. Bo, Z. Peng, and C. Junliang, "Design and implementation for integrated services orchestration bus using SCA over heterogeneous networks," *Ann. Telecommun. - Ann. Des Télécommunications*, vol. 67, no. 1–2, pp. 91–107, May 2011.

[47] R. Ten-hove and P. Walker, "JSR 208: Java Business Integration (JBI) 1.0." Sun Microsystems Inc, p. 176, 2005.

[48] M. Femminella, E. Maccherani, and G. Reali, "A software architecture for simplifying the JSLEE service design and creation," in *International Conference on Software Telecommunications and Computer Networks SoftCOM 2010*, 2010, vol. 22, pp. 235–239.

[49] T. Ambra, "Description and Composition of Services towards the Web-Telecom Convergence," in *Service-Oriented Computing–ICSOC 2013 Workshops*, Lecture No., Switzerland: Springer International Publishing, 2014, pp. 578–584.

[50] T. Eichelmann, W. Fuhrmann, U. Trick, B. Ghita, and U. Kingdom, "Discussion on a framework and its service structures for generating JSLEE based value added services," in *Fourth International Conferences on Internet Technologies and Applications (ITA 11)*, 2011, pp. 169–176.

[51] N. . Buezas, E. . Guerra, J. . De Lara, J. . Martín, M. . Monforte, F. . Mori, E. . Ogallar, O. . Pérez, and J. . Sánchez Cuadrado, "Umbra designer: Graphical modelling for telephony services," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7949 LNCS, pp. 179–191, 2013.

[52] P. Falcarin and C. Venezia, "Communication Web Services and JAIN-SLEE Integration Challenges," *Int. J. Web Serv. Res.*, vol. 5, no. 4, pp. 59–78, 2008.

[53] P. A. Van Brakel, "Exploring hybrid communications with a Mobi4d platform–the use of web services and the role of SOA in telecommunications," *Proc. 13th Annu. Conf. WORLD WIDE WEB Appl.*, no. September, 2011.

[54] A. Botha, I. Makitla, J. Tolmay, M. Ford, D. Seetharam, L. Butgereit, O. Ogunleye, and C. Abouchabki, "Mobile4D platform," in *ISTAfrica 2010*, 2010, pp. 1–7.

[55] O. S. Ogunleye, I. Makitla, A. Botha, J. P. Tomay, T. Fogwill, D. Seetharam, and P. Geldenhuys, "Mobi4D: A next generation service delivery platform for mobile government services: An African perspective," in *3rd IEEE International Conference on Adaptive Science and Technology ICAST 2011*, 2011, pp. 15–20.

[56] O. S. Ogunleye and I. Makitla, "Mobil4D Platform : A Mobile Learning Opportunity and Support for Nursing Education," in *mLife 2010 Conferences*, 2010, no. 2005, pp. 2–7.

[57] O. Chudnovskyy, T. Nestler, M. Gaedke, F. Daniel, and J. Ignacio, "End-User-Oriented Telco Mashups: The OMELETTE Approach," in *The 21st international conference companion on World Wide Web*, 2012, pp. 235–238.

[58] WSO2 Inc, "Jaggery.js," 2015. [Online]. Available: http://jaggeryjs.org/. [Accessed: 02-Oct-2015].

[59] M. Belaunde, P. Falcarin, and J. P. A. Almeida, "Model-Driven Service Creation for a Telecom Service Platform," in *Handbook of Research on Mobile Software Engineering: Design, Implementation, and Emergent Applications*, 2012, pp. 124–137.

[60] J. M. López, "Dynamic Service Composition in an Innovative Communication Environment," no. October. MSc thesis, University of Twente, Enschede, The Netherlands, p. 101, 2008.

[61] J. Yu, P. Falcarin, D. Alamo, J. Maria, J. Sienel, Q. Z. Sheng, and J. F. Mejia, "A User-Centric Mobile Service Creation Approach Converging Telco and IT Services," *2009 Eighth Int. Conf. Mob. Bus.*, pp. 238–242, 2009.

[62] J. Y. J. Yu, Q. Z. Sheng, and P. Falcarin, "A visual semantic service browser supporting user-centric service composition," *Adv. Inf. Netw. Appl. AINA 2010 24th IEEE Int. Conf.*, pp. 244–251, 2010.

[63] J. Niemöller, E. Freiter, K. Vandikas, R. Quinet, R. Levenshteyn, and I. Fikouras, "Composition in Converged Service Networks : Requirements and Solutions," in *International Workshop on Business System Management and Engineering (BSME 2010)*, 2010.

[64] A. Hartman, M. Keren, and D. Pikus, "Model-based Design and Generation of Telecom Services Services," *IBM Journals*, no. Section 2, 2009.

[65] Y. Yuan, J. J. Wen, W. Li, and B. B. Zhang, "A Comparison of Three Programming Models for Telecom Service Composition," *Third Adv. Int. Conf. Telecommun. AICT07*, pp. 1–1, 2007.

[66] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*. New Delhi, India: Prentice-Hall, 1974.

[67] D. Grigori, J. C. Corrales, and M. Bouzeghoub, "Behavioral matchmaking for service retrieval: Application

to conversation protocols," *Inf. Syst. J.*, vol. 33, no. 7–8, pp. 681–698, 2008.

[68] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541 – 580, 1989.

[69] B. L. B. Lee and E. A. Lee, *Interaction of finite state machines and concurrency models*, vol. 2, no. November. IEEE, 1998, pp. 1715–1719.

[70] D. Harel, "Statecharts: A visual formalism for complex systems.," *Sci. Comput. Program.*, vol. 8, no. 3, pp. 231–274, 1987.

[71] TelComp2.0, "Entregable 1.0: Representación de componentes complejos," Popayan, 2012.

[72] K. JENSEN and L. L. KRISTENSEN, Lars Michael KRISTENSEN, "Coloured Petri nets: modelling and validation of concurrent systems." Springer, 2009.

[73] H. J. Genrich and K. Lautenbach, "System modelling with high-level Petri nets," *Theor. Comput. Sci.*, vol. 13, no. 1, pp. 109–135, 1981.

[74] P. Bouyer, S. Haddad, and P.-A. Reynier, "Timed Petri nets and timed automata: On the discriminating power of zeno sequences," *Inf. Comput.*, vol. 206, no. 1, pp. 73–107, 2008.

[75] G. A. Agha, F. De Cindio, and G. Rozenberg, "Concurrent object-oriented programming and petri nets: advances in petri nets." Springer, 2001.

[76] Y. Liu, Y. Gu, and J. Chen, "A New Control Structure Model Based on Object-oriented Petri Nets," *J. Networks*, vol. 7, no. 14, pp. 746–753, 2012.

[77] ITU-T, "Vocabulario de términos relativos a la conmutación y la señalización," 1988. [Online]. Available: http://www.itu.int/rec/T-REC-Q.9-198811-I.

[78] W. Reichl and E. O. Ruhle, "Competition and Services in Next Generation Networks," in *17th Biennial International Telecommunications Society*, 2008.

[79] R. Taylor, B. Zhang, and S. Chen, "Value Added Services Policy Reform in China: Lessons for--and from--the US In Managing an Evolving Market," in *Telecommunications Policy Research Conference*, 2007.

[80] I. G. Ben Yahia, E. Bertin, J. P. Deschrevel, and N. Crespi, "Definition for Next Generation Networks," in *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*, 2006, pp. 22–29.

[81] S. S. C. Shang, E. Y. Li, Y.-L. Wu, and O. C. L. Hou, "Understanding Web 2.0 service models: A knowledge-creating perspective," *Inf. Manag.*, vol. 48, no. 4–5, pp. 178–184, 2011.

[82] A. S. Staines, "Rewriting Petri Nets as Directed Graphs," *Int. J. Comput.*, vol. 5, no. 2, pp. 289–297, 2011.

[83] D. Rivas, D. Corchuelo, C. Figueroa, J. Corrales, and R. Giugno, "Business Process Model Retrieval Based on Graph Indexing Method," in *Business Process Management Workshops*, 2011, vol. 66, pp. 238–250.

[84] Jboss Comunity, "Mobicents JAIN SLEE Datasources." .

[85] V. Der Aalst, M. P. Wil, and W. Stephan, "WofBPEL: A Tool for Automated Analysis of BPEL Processes," *Computing*, pp. 484–489, 2005.

[86] N. Lohmann, "A Feature-Complete Petri Net Semantics for WS-BPEL 2 . 0," in *Web Services and Formal Methods*, no. September, Springer Berlin Heidelberg, 2007, pp. 77–91.

[87] C. Bo, C. Junliang, and D. Min, "Petri net based formal analysis for multimedia conferencing services orchestration," *Expert Syst. Appl.*, vol. 39, no. 1, pp. 696–705, 2012.

[88] W. M. P. van der Aalst and K. B. Lassen, "Translating unstructured workflow processes to readable BPEL: Theory and implementation," *Inf. Softw. Technol.*, vol. 50, no. 3, pp. 131–159, 2008.

[89] B. Kitchenham, S. Linkman, and D. Law, "DESMET: a methodology for evaluating software engineering methods and tools," *Comput. Control Eng. J.*, vol. 8, no. 3, p. 120, 1997.

[90] F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, and J. Stafford, "Documenting Software Architectures: Views and Beyond." Addison-Wesley Professional, 2011.

[91] C. Kankanamge, "Web Services Testing with SoapUI." Packt Publishing Ltd., 2012.

[92] SmartBEAR, "SOAPUI," 2012. [Online]. Available: http://www.soapui.org.

[93] S. Hussain, Z. Wang, I. K. Toure, and A. Diop, "Web Service Testing Tools: A Comparative Study." arXiv preprint arXiv:1306.4063, 2013.