Composición Dinámica de Servicios Web RESTful en un Entorno Móvil

Tesis de grado

Andrés Felipe Dorado Dorado
Henrry Harvey Vallejo Cuero

Director: Ing. Fulvio Yesid Vivas Cantero

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telemática
Línea de Investigación de Servicios Avanzados de Telecomunicaciones
Popayán, Febrero de 2012

Resumen

La Web 2.0 ha establecido sus bases desde hace un tiempo en el entorno estático computacional, con el fin de ubicar al usuario como eje fundamental de los servicios Web, permitiéndole interactuar con los mismos, aprovechando la notable evolución que han tenido las tecnologías, tanto para las redes como para los dispositivos y elementos que interactúan en ellas.

Con la evolución de las redes inalámbricas y la gran aceptación por parte de los usuarios de las tecnologías que participan en ellas, es necesario integrar la Web 2.0 al entorno móvil, con el fin de acceder a la Web desde dispositivos que se desplazan junto al usuario. Es por ello que nace el concepto de la Mobile 2.0 o Mobile Web 2.0, el cual adapta los conceptos definidos por la Web 2.0 al entorno móvil. En este contexto diariamente se genera abundante contenido, tales como servicios, aplicaciones, mashups, widgets, etc. por parte de empresas desarrolladoras de servicios Web, desarrolladores independientes y otro tipo de individuos, que orientan sus procesos a obtener el mayor provecho a todas las ventajas que ofrece el medio, al mismo tiempo que se combaten las desventajas que presenta el entorno móvil.

Además de las redes y las tecnologías para la comunicación de datos que cada vez tiene mayores tasas de transferencia y disponibilidad para los usuarios, se tienen los dispositivos que acceden a ellas, que hasta hace 10 años, solo servían para hacer y recibir llamadas de voz, enviar mensajes de texto, o acceder a Internet de una manera muy poco practica, en el caso de los teléfonos móviles. Hoy en día son dispositivos que cuentan con diversos núcleos de procesamiento; sistemas operativos multifuncionales y multitarea; espacios de memoria que antes eran un privilegio de las computadoras de escritorio; interfaces de entrada y salida que posibilitan la interconexión con diferentes dispositivos; e incluso diversos sensores que permiten extraer medidas del entorno, para generar servicios y aplicaciones para interactuar con el mundo real.

En ese sentido, el presente trabajo de grado enfoca sus esfuerzos en tomar servicios que ya han sido creados en la Web, más específicamente los servicios Web RESTful y enlazarlos para conformar o componer nuevos servicios, que se adapten a los conceptos definidos por la Mobile 2.0 dando mayor relevancia a los requerimientos de usuario. Para esto, es necesario definir los mecanismos y herramientas pertinentes, para componer dinámicamente servicios Web RESTful, que se adecuen a requerimientos específicos de un usuario en el contexto móvil, tomando como punto de partida diversos tipos de adelantos e investigaciones realizadas hasta el momento por parte de empresas, grupos de investigación y personas del medio.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Definición del Problema

El desarrollo de las tecnologías software y hardware involucradas en la infraestructura de Internet, y los principios y prácticas utilizados para definir la Web 2.0, como por ejemplo, el enriquecimiento de la experiencia del usuario, han ocasionado un aumento en la base de servicios ofertados en la Web. Esto se aprecia, por citar un caso, en el rápido crecimiento de los sitios de redes sociales y de diferentes servicios multimedia, cuyo acceso no se restringe a los equipos conectados a redes fijas, sino que se extiende a los dispositivos móviles conectados a redes inalámbricas a través de Wi-Fi o de tecnologías celulares [1-3].

Adicionalmente, la Web 2.0 ha propiciado la creación y actualización constante de servicios Web, con el fin de satisfacer las necesidades específicas de mercados reducidos y permitir una interacción más directa del usuario con la Internet. Como resultado, la cantidad y diversidad de los servicios Web disponibles es notable, siendo posible clasificarlos en categorías como: correo electrónico, acceso a bases de datos, juegos, mapas, mensajería, música, noticias, telefonía, entre otras [4]. Debido a esta gran oferta, es necesario crear mecanismos que ayuden a componer nuevos servicios a partir de los existentes, fomentando la reutilización, reduciendo los costos de producción, los tiempos de creación y la necesidad de talento humano [5].

El fortalecimiento del entorno móvil, gracias a la convergencia de tecnologías de red y a la ubicuidad que ofrecen los dispositivos involucrados, ha generado la necesidad de adaptar el marco teórico proporcionado por la Web 2.0 a dicho entorno; es ahí donde surge el concepto de la Mobile 2.0, la cual es una tendencia en el sector de las redes de comunicaciones inalámbricas que cubre dicha necesidad, teniendo en cuenta, tanto las capacidades brindadas, como las restricciones impuestas por las limitantes del hardware y la dinámica propia de las redes inalámbricas [6].

En este sentido, la Mobile 2.0 es un entorno centrado en el usuario, que se caracteriza, entre otros aspectos, por la creación de aplicaciones personalizadas que responden a requerimientos específicos, y por el aprovechamiento de la portabilidad y ubicuidad de los dispositivos móviles. Estas características requieren la aplicación de mecanismos de composición dinámicos que permitan definir servicios a la medida, los cuales se actualicen constantemente de acuerdo con las peticiones del usuario, las respuestas del sistema y las condiciones de los servicios que lo constituyen. Un posible ejemplo son las aplicaciones que generan o consumen contenidos sensibles a la localización y al tiempo, en donde se utilizan las particularidades del contexto para definir los servicios con la mínima intervención del usuario [7, 8].

La composición dinámica en la Mobile 2.0 trae consigo la necesidad de ejecutar los servicios Web en los dispositivos móviles. Para lo cual, una de las alternativas de solución son los Servicios Web Móviles (SWM), los cuales se definen como una extensión de las tecnologías de la arquitectura de servicios Web del W3C (World Wide Web Consortium)

[9]. No obstante, existen problemas en cuanto al rendimiento alcanzado, debido principalmente a que los servicios Web tradicionales, que constituyen la base de los SWM, fueron desarrollados para ser ejecutados en entornos fijos, los cuales, a diferencia de los móviles, poseen plataformas potentes y capacidades de transmisión de datos elevadas [10]. Por esta razón, es necesaria la exploración y utilización de mecanismos de mayor eficiencia que se adapten a la Mobile 2.0. Como alternativa, en [11] se propone el uso de los servicios Web RESTful, los cuales son más ligeros que los tradicionales basados en SOAP (Simple Object Access Protocol) y están soportados en recursos referenciados por identificadores URI (Uniform Resource Identifier), que pueden ser manipulados a través del protocolo HTTP (Hypertext Transfer Protocol) [12].

Actualmente los servicios Web RESTful están recibiendo mayor atención en el sector de las telecomunicaciones, pues constituyen un método de programación ligero y atractivo para los desarrolladores [13]. Por lo tanto, un mecanismo que integre estos servicios en uno solo en el contexto Mobile 2.0, se convierte en un elemento de gran importancia, que trae beneficios como la reutilización de servicios existentes, aplicaciones más acordes a los requerimientos de cada usuario, reducción en costos finales, etc. Sin embargo, su utilización en el contexto SOA (Service Oriented Architecture), ha sido poco investigada, razón por la cual no se han explorado, en gran medida, conceptos como su descubrimiento, composición y ejecución, recayendo actualmente en mecanismos de composición estáticos, que requieren la selección manual de los servicios y no se adecuan al funcionamiento dinámico de la Mobile 2.0 [14, 15].

El escenario descrito y las tendencias a las que apunta el campo de las telecomunicaciones en el ambiente móvil, permiten plantear la siguiente pregunta de investigación: ¿Cómo se pueden componer dinámicamente los servicios Web RESTful en un entorno de la Mobile 2.0?

1.2. Objetivos

1.2.1. Objetivo general

- Proporcionar mecanismos para la composición dinámica de servicios Web RESTful en un entorno Mobile 2.0.

1.2.2. Objetivos específicos

- Adaptar un algoritmo para la composición dinámica de servicios Web RESTful en un entorno Mobile 2.0.
- Adaptar una arquitectura para la ejecución de servicios Web RESTful compuestos en un entorno Mobile 2.0.
- Implementar y evaluar un prototipo de la arquitectura y del algoritmo de composición de servicios Web RESTful propuestos.

1.3. Alcance del Trabajo

El alcance del presente trabajo de grado está limitado a las adaptaciones de una arquitectura para la ejecución de servicios Web RESTful compuestos y un algoritmo para la composición dinámica de servicios Web RESTful, teniendo como base, estudios e

investigaciones ya existentes sobre el tema, tanto para el entorno móvil, como para el estático.

El algoritmo permitie la composición dinámica de servicios Web RESTful presentes en la Web, que previamente han sido ordenados en un repositorio mediante su respectiva descripción (proceso realizado por el modulo de Descubrimiento de Servicios, que no corresponde a este proyecto), para ser ejecutado en un entorno móvil; tal servicio compuesto debe cumplir con los requerimientos iniciales de un usuario experto, que es enviada como una petición en un lenguaje formal comprensible por maquina. En cuanto al proceso de composición y reconfiguración del servicio, es ejecutado por el servidor, no por el dispositivo móvil, este solo consume la composición a través de un cliente navegador Web móvil.

La arquitectura corresponde a la adaptación de algunos bloques y la combinación de diversas arquitecturas, que utilizan servicios Web tradicionales, tales como SOAP, para que opere bajo servicios Web RESTful, haciendo énfasis en la ejecución del servicio, mediante las siguientes acciones:

- Actualización de los formatos de representación teniendo en cuenta únicamente la tasa de transferencia de la red inalámbrica.
- Reconfiguración del servicio compuesto cuando ocurren fallas en los servicios simples en tiempo de ejecución.
- Interactividad por parte del usuario, permitiéndole añadir, eliminar o cambiar servicios simples pertenecientes al compuesto en tiempo de ejecución.

Para la comprobación de los objetivos, se implementa un prototipo que hace uso del algoritmo adaptado, basado en los lineamientos propuestos en la arquitectura para la composición de servicios Web RESTful, que a su vez toman como fundamento los conceptos definidos para la Mobile 2.0.

El resultado de hacer la composición de servicios Web RESTful para el entorno móvil, está dirigido a usuarios que tengan acceso a dispositivos móviles que posean un navegador que soporte tecnología Java Script, tales como Smartphone, PDA's, consolas de video juego portables, e-Readers, entre otros.

Aunque la Mobile 2.0 define tanto conceptos técnicos (tipos de tecnologías usadas) como sociales (experiencias de usuario), en este trabajo de grado se tomaran de manera generalizada, para poder abordar la mayoría de ellos, con el fin de cumplir tanto con los requisitos de usuario como los técnicos, además de obtener el mejor nivel de acoplamiento a los estándares actuales.

Conviene aclarar que el presente trabajo de grado no es una referencia rigurosa que define el uso de herramientas de programación, desarrollo, diseño y ejecución como única opción para la Composición Dinámica de Servicios Web RESTful; en su lugar propone un conjunto de mecanismos y técnicas que se adaptan en mayor grado a estándares y conceptos del medio.

1.4. Estructura del Trabajo

El documento se encuentra organizado de tal forma, que primero se abordan los conceptos correspondientes a la temática necesaria para este trabajo de grado, seguido por los respectivos procedimientos para alcanzar los objetivos propuestos, los cuales serán puestos a prueba en un prototipo, cuyos resultados serán anotados y analizados al final de este documento; para terminar se encuentran conclusiones, recomendaciones y propuestas acordes a los efectos obtenidos. Partiendo de lo anterior, la estructura en detalle del presente documento de trabajo de grado para la Composición Dinámica de Servicios Web RESTful en un Entorno Móvil es:

- Capítulo 2: Construcción del marco teórico sobre el ambiente móvil y conceptos de la Mobile 2.0, que son las bases a tener en cuenta para el algoritmo y la arquitectura que se adaptan.
- Capítulo 3: Definición y características de los servicios Web RESTful y los conceptos más importantes para su composición. En este capítulo se fijan las bases para adaptar el algoritmo y la arquitectura.
- Capítulo 4: Se exponen algunas arquitecturas base, realizadas por otros grupos de trabajo, las cuales se convierten en referencias para realizar una adaptación, que cumpla con los requerimientos considerados para un entorno de la Mobile 2.0 y los servicios Web RESTful.
- Capítulo 5: Se presentan las referencias para adaptar el algoritmo para la composición dinámica de servicios Web RESTful. También se muestra el algoritmo resultante y sus detalles.
- Capítulo 6: Contiene todos los detalles acerca del prototipo, su diseño, implementación, pruebas y análisis de resultados, con el fin de cumplir y comprobar los objetivos del proyecto.
- Capítulo 7: Presenta las conclusiones y recomendaciones acerca de los resultados obtenidos a lo largo del trabajo, además se proponen acciones futuras que pueden ser llevadas a cabo como complemento a este proyecto.

Para complementar y hacer énfasis en algunos temas importantes del trabajo de grado, se presentan los siguientes anexos:

- Anexo A. Métodos HTTP para el sistema
- Anexo B. Herramientas para la detección de dispositivos
- Anexo C. Composición STE
- Anexo D. Diseño e implementación del prototipo para la composición dinámica de servicios Web RESTful
- Anexo E. Características de WURFL y Clasificación de Formatos
- Anexo F. Artículo

Capítulo 2

Entorno Mobile 2.0

2.1. Ambiente Móvil

El ambiente móvil tiene cada vez más aceptación por parte de los usuarios, a pesar de ciertas desventajas que presenta con relación al entorno estático, pero que en la actualidad se han visto aplacados por la creación y mejoras de tecnologías hardware y software (Figura 1), además de la principal y más importante ventaja que proporciona, la movilidad, característica que ha servido para la generación de diferentes tipos de servicios y aplicaciones que se adaptan a los requerimientos particulares de cada usuario [16].

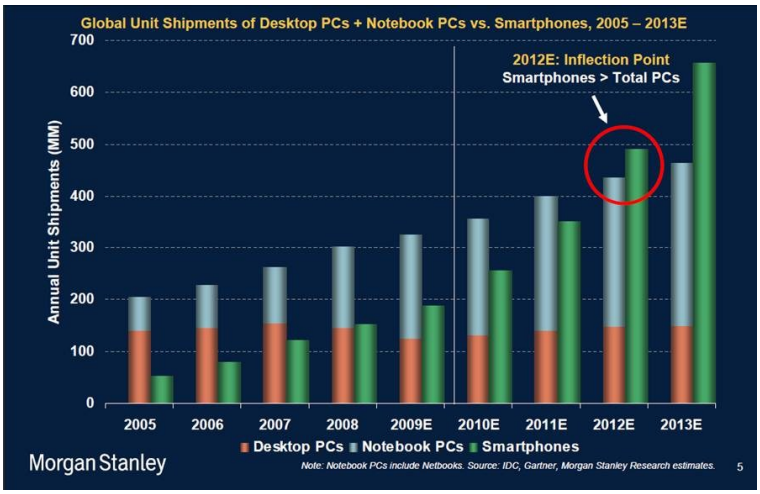


Figura 1. Estadísticas Dispositivos Entorno Móvil vs Entorno Estático [16].

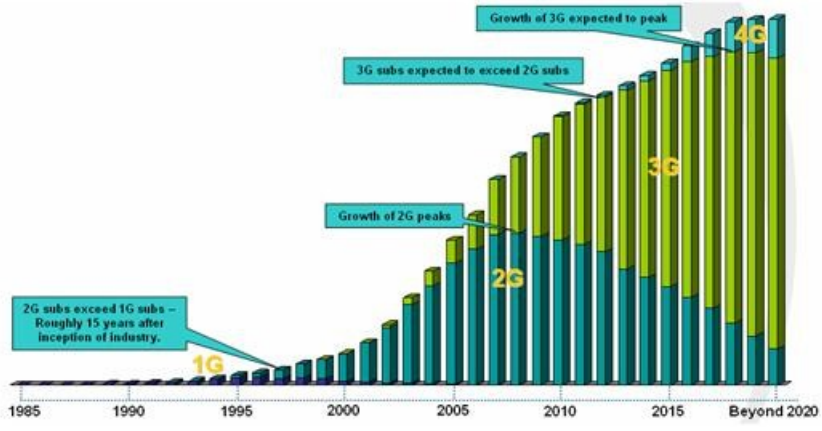


Figura 2. Aceptación de Tecnologías Inalámbricas a Nivel Mundial [17]

Junto con los dispositivos móviles, las redes también han sufrido diferentes tipos de cambios que han mejorado los aspectos del entorno móvil, creando con ello una mayor

cantidad de usuarios que interactúan con dicho entorno, generando un crecimiento de subscriptores por año y su evolución a la par con las tecnologías (Figura 2).

Algunas de las características y aspectos más importantes, que se constituyen en ventajas o desventajas y caracterizan al entorno móvil, son los siguientes [18]:

- La localización de los usuarios es variable, debido a que este se traslada de un sitio a otro dentro de una o muchas redes.
- Los dispositivos móviles sufren desconexiones o caídas de señal cuando salen de los rangos de cobertura, esto se debe a la inestabilidad del medio y problemas en los dispositivos, por lo cual no es posible el intercambio de información a través de la red en esos periodos de desconexión.
- La mayor ventaja de los dispositivos móviles recae en la libertad de transportarlos de un lado a otro por un usuario, lo que permite la creación de contenido muy particular en cualquier momento y lugar. Además de ello la posibilidad de acceder a diferentes elementos hardware y software que interactúan con el medio en el que se encuentra el usuario.
- Las características del medio no son constantes, esto se debe tanto a la diversificación de las redes y tecnologías, como a características inherentes en ellas (ruido, refracción, desvanecimiento, atenuación, interferencia, materiales de construcción, etc.).
- Las fallas en el entorno móvil para un usuario, se pueden clasificar en dos tipos: i) fallas irreparables, las cuales imposibilitan permanentemente o por un tiempo muy largo el acceso a la red, por ejemplo daños en el dispositivo móvil o falta del mismo y daños en la red de larga duración, ii) fallas sin daño permanente, las cuales imposibilitan el acceso a la red por un periodo corto de tiempo, como por ejemplo descarga de la batería, perdidas de información , caídas de red temporales, etc.

2.2. Mobile 2.0

El concepto de Mobile 2.0 o Mobile Web centra sus bases en la ya establecida Web 2.0, adaptando sus principios a las características del ambiente inalámbrico [6, 19], donde la mejor definición, es considerarla como la integración de la Web al entorno móvil, aunque dicha tesis aborda más que una simple traslación o traspaso de la existente Web a tal entorno, la cual encierra gran cantidad de conceptos que se investigan y evalúan a diario, debido a características tales como, la evolución de la tecnología inalámbrica, el constante cambio que experimenta el escenario móvil y que posee limitaciones muy particulares que lo diferencian de un entorno fijo, que también son limitantes para la Mobile 2.0, entre otras. Dentro de esos límites, los más importantes a tener en cuenta y que se constituyen en retos para la Mobile 2.0 son: el ancho de banda y las limitaciones de los dispositivos (operativas y físicas), compatibilidad de plataformas y modelos de negocio [19].

El concepto de Mobile 2.0 no es algo nuevo, desde hace años las empresas han intentado conectar la Web y lo móvil, pero sin conseguir resultados exitosos, esto se debió al grado de desarrollo tecnológico insuficiente y a procesos de división de soluciones que debían esperar a fases posteriores, lo que ocasionó pérdidas económicas y temporales, que rechazaban procesos de estandarización firmes [20]. Esto cambió con la evolución de las

redes (con mayores tasas de datos, fácil acceso) y los terminales que procesaban datos de manera más eficiente, basándose, en el caso de la telefonía móvil, en tecnologías como 3G, 3.5G [21].

Uno de las principales premisas de la Mobile 2.0 es que está dirigida al usuario de manera directa, por lo cual las aplicaciones y servicios creados, cuentan con características como: mejoras en aspecto grafico (con diseños simplificados), fácil y rápido acceso a redes inalámbricas (Wi-Fi, WiMAX, HSPA, etc.), rápido procesamiento, entre otras, todo esto, dejando atrás los métodos utilizados en el entorno estático [21].

Para la obtención de todos los aspectos que conforman la Mobile 2.0 se han fijado tres fases importantes como se muestra en la Figura 3.

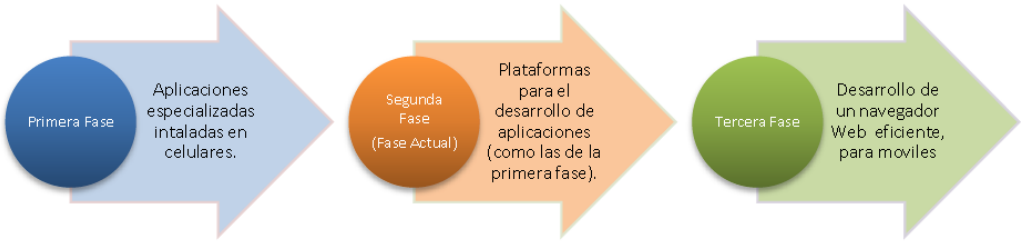


Figura 3. Fases hacia la Mobile 2.0 [22]

- La primera fase, está constituida por la creación de aplicaciones especializadas para ser instaladas en los dispositivos móviles, soportados en plataformas de desarrollo ya existentes (Java, Symbian, Windows Mobile, Android, etc.) que revolucionen el aspecto y la forma de utilizar los servicios al integrar la Web.
- La segunda fase está más enfocada en los desarrolladores y diseñadores de las aplicaciones móviles Web, ya que comprende la creación de plataformas específicas para desempeñar un trabajo más simple pero eficiente.
- Como tercera y última fase, es la consecución de un navegador Web para móviles con una efectividad similar a la de los computadores de escritorio, aunque ya existen desde hace tiempo, estos no cubren las necesidades particulares de un entorno móvil.

Teniendo en cuenta las partes importantes que pueden interactuar en la Mobile 2.0, se presentan por un lado los desarrolladores de servicios y aplicaciones, y por el otro, el usuario. Haciendo énfasis en la telefonía móvil y en la posición adquirida actualmente, los operadores de telecomunicaciones móviles son un elemento importante, ya que controlan las redes que permiten el acceso, la autenticación, la interconexión de dispositivos, llamadas, mensajes, roaming y otro tipos de servicios desplegados las 24 horas, 7 días a la semana, realizando la gestión o administración de dichos servicios como modelo de negocio [23].

El hecho de que los operadores móviles sean los encargados de brindar los servicios y los medios para acceder a ellos, no es suficiente para considerarlos como parte importante para el desarrollo de la Mobile 2.0. Este complemento se puede encontrar en la decisión que toman los operadores de abrir o liberar APIs y servicios, con el fin de descubrir y generar nuevo contenido mediante el trabajo colectivo que ofrece la Web 2.0 y la misma Mobile 2.0. [24].

2.2.1. Principios de la Mobile 2.0

La asociación entre la Web y el entorno móvil, trae consigo diversas ventajas que se comparan a las obtenidas con la Web 2.0 en el ambiente estático, esto se debe a la adaptación de los principios descritos en la Tabla 1.

Principios de la Mobile 2.0	Adaptación a la Web móvil 2.0
La Web como plataforma	Ofrece soporte a los SWM, de tal manera que compensa las características limitadas de los dispositivos [10].
Aprovechamiento de la inteligencia colectiva	El intercambio de información y la facilidad de acceso a los datos, son unas de las principales funcionalidades en la Web 2.0. Este principio se conserva en la Mobile 2.0.
Base de datos basada en la red	Las características limitadas de los dispositivos móviles, hacen que se requieran lugares de almacenamientos ubicados de manera distribuida en la red.
El fin del ciclo del lanzamiento de software	Facilita la actualización constante de los servicios almacenados remotamente, sin influir negativamente en el rendimiento de los dispositivos móviles.
Modelos de programación ligeros	Este principio busca la interacción ligera de las aplicaciones con los dispositivos móviles y la simplicidad en cuanto a procesamiento de datos, búsqueda, interfaces, enlaces, entre otras [25].
Software no limitado a un dispositivo	Con este principio la Mobile 2.0 pretende que los SWM sean soportados en múltiples plataformas.
Enriquecimiento de la experiencia del usuario	La ventaja principal del entorno móvil radica en que el dispositivo viaja junto al usuario, lo que permite la aplicación de LBS(Location Based Services) [26], búsqueda Web y creación de contenidos. La Mobile 2.0 busca aprovechar dichas ventajas para hacer del dispositivo móvil el equipo ubicuo por excelencia [27].

Tabla 1. Principios Mobile 2.0

Tales principios de la Mobile 2.0 dan soporte al mecanismo de composcion en este trabajo de grado, aunque dicho concepto de composición de servicios no es nuevo, si es el momento idóneo para abordarlo como un mecanismo eficiente.

Capítulo 3

Servicios Web RESTful

La Mobile 2.0, como se mencionó en el capítulo anterior, encuentra sus fundamentos en un proceso de adaptación de los principios de la Web 2.0, el cual tiene en cuenta, tanto las ventajas, como las limitantes propias del ambiente móvil. Dicho proceso, ha permitido que se plantee una participación más activa por parte de los usuarios móviles en los servicios de la Web, y especialmente en el consumo y producción de contenidos específicos para los dispositivos móviles [28]. También ha favorecido la utilización de modelos de programación ligeros, que reducen la cantidad de esfuerzo necesaria para producir nuevos servicios, como es el caso de los Mashup y de las técnicas de composición, que basan su funcionamiento en la agregación de información, obtenida mediante servicios Web u otras fuentes de datos en la red.

En este punto, el estilo arquitectónico REST (Representational State Transfer), definido como el conjunto de principios que rigen el comportamiento de la arquitectura de la Web, juega un papel importante, pues se convierte en el punto de partida sobre el cual se construyen los servicios Web RESTful. Este tipo de servicios se consideran más ligeros que los tradicionales basados en SOAP y por lo tanto, más idóneos en un entorno limitado como el móvil [11].

Este capítulo profundiza en los conceptos del estilo REST, los servicios Web derivados de dicho estilo, entre otros temas relacionados.

3.1. La composición de servicios Web RESTful y la Mobile 2.0

La composición es un proceso que busca generar nuevos servicios a partir de servicios existentes, promoviendo la reutilización y la agilidad en la creación de nuevas aplicaciones. Este proceso es de gran importancia en la Mobile 2.0, pues facilita desde un punto de vista práctico, el acogimiento de los conceptos definidos en los siete principios adaptados de la Web 2.0.

- **La Web como plataforma:** Los servicios Web RESTful usan la Web como su plataforma, transformándola en un ecosistema de servicios reutilizables que pueden servir como base en la creación de nuevas aplicaciones. Este ecosistema se beneficia de la sencillez de la arquitectura de la Web, del uso de un protocolo común (HTTP) y de la ubicuidad de algunos formatos de representación como XML (Extensible Markup Language [29]) y JSON (JavaScript Object Notation [30]), los cuales están basados en cadenas de texto y son utilizados para el intercambio de datos entre servicios [31].
- **Aprovechamiento de la inteligencia colectiva:** Los sitios encontrados comúnmente en un entorno Web 2.0 permiten a un usuario compartir todo tipo de información. Muchos de estos sitios proveen servicios Web RESTful como un medio para facilitar el acceso a la información a terceras partes, las cuales desarrollan nuevas aplicaciones que le agregan valor a los datos de los usuarios.
- **La Web como un medio para el almacenamiento remoto de información (Importancia de los datos):** Los servicios Web RESTful se encuentran basados

en recursos, los cuales son un mecanismo para acceder a datos e información mediante un identificador único, facilitando su distribución en diferentes servidores remotos. Este principio pone de manifiesto la importancia de los datos en el contexto de REST.

- **Fin del ciclo de lanzamiento de software:** Uno de los objetivos de REST es el de lograr un bajo acoplamiento entre los componentes de la arquitectura, permitiendo su evolución independiente. Esta característica facilita la adición de nuevas funcionalidades sin utilizar los clientes existentes, promoviendo la actualización y mejora constante de los servicios.
- **Modelos de programación ligeros:** Los servicios Web RESTful usan tecnologías ampliamente desplegadas como HTTP y XML, las cuales son fácilmente soportadas en marcos de desarrollo de diferentes lenguajes de programación, lo cual facilita el consumo y publicación de este tipo de servicios.
- **Software independiente del dispositivo:** Los servicios Web RESTful son independientes de la plataforma específica de un dispositivo, pues se basan en tecnologías estándar ampliamente utilizadas. Esto se observa por ejemplo en las aplicaciones soportadas en navegadores Web, que pueden ser fácilmente accedidas desde computadores de escritorio como dispositivos móviles con soporte de HTML.
- **Enriquecimiento de la experiencia del usuario:** La composición tradicional facilita la creación de servicios, pero no ofrece gran flexibilidad en la generación de experiencias personalizadas y adaptables al contexto de usuario de forma automática, para ello se introduce la composición dinámica de servicios, la cual provee mecanismos para la creación de servicios Web que se adecuen a las necesidades cambiantes de un usuario, teniendo en cuenta el contexto en el que se encuentra.

En las siguientes secciones correspondientes a este capítulo, se aborda con mayor profundidad los conceptos referentes a los servicios Web RESTful.

3.2. Fundamentos de los servicios Web RESTful

3.2.1. Transferencia de Estado Representacional

REST es definida en el trabajo de Fielding [32], como un estilo arquitectónico para sistemas de hipermedia distribuidos. REST comprende un conjunto de principios derivados de arquitecturas particulares que exhiben propiedades de interés en el dominio de aplicación. Cada uno de estos principios define reglas o criterios para dirigir el comportamiento de los elementos de una arquitectura, influyendo en los componentes e interacciones de las aplicaciones implementadas. De acuerdo con lo anterior, la utilización de un estilo arquitectónico induce en el sistema desarrollado, diferentes propiedades.

Sin embargo, el uso de un estilo específico no garantiza que la aplicación desplegada posea las propiedades deseadas, ya que un principio arquitectónico se encuentra condicionado por la implementación realizada. En el caso de REST, este se encuentra enmarcado en el dominio de los sistemas de hipermedia distribuidos, las propiedades que se consideran relevantes son aquellas pertenecientes a las aplicaciones de software soportadas en redes informáticas, las cuales son detalladas a continuación.

3.2.1.1. Propiedades de interés en una aplicación soportada en una red informática

En el campo de la Ingeniería de Software, el diseño de una arquitectura busca dotar una aplicación de un conjunto de características que garantice el funcionamiento óptimo del sistema. En el ámbito de las aplicaciones soportadas en redes informáticas, en el que se encuentra REST, se identifican las propiedades o características en la Tabla 2 .

Propiedad	Descripción
Rendimiento de la red	Tiene en cuenta medidas relativas al proceso de comunicación, como es el consumo de ancho de banda y su cantidad disponible. También se consideran el tipo de interacción entre los componentes de una arquitectura, la cantidad de sobrecarga o información de control y la granularidad de los mensajes.
Rendimiento percibido por el usuario	Las medidas más importantes sobre este aspecto son la latencia (tiempo entre el inicio de una interacción y la primera indicación de respuesta) y el tiempo de finalización (tiempo entre el inicio de una interacción y la recepción completa de una respuesta). Estas medidas pueden verse afectadas por las reglas de procesamiento de los datos.
Eficiencia de la red	Indica el nivel de utilización de la red por parte de un sistema. Se considera que la aplicación distribuida más eficiente es aquella que minimiza el uso de la red cuando es posible, como por ejemplo aprovechando interacciones previas o realizando procesamiento local de datos.
Escalabilidad	Es la capacidad de un sistema de soportar un gran número de componentes y las interacciones entre ellos, sin afectar notablemente el rendimiento y la eficiencia de la aplicación.
Sencillez	Se refiere a la cantidad y complejidad en el procesamiento de datos que debe realizar un componente. La forma más directa de lograr sencillez es por medio de la adecuada delegación de responsabilidades a otros componentes.
Evolución independiente	Mide que tan factible es el cambio en la implementación de un componente sin afectar negativamente a los demás. Esta propiedad puede mejorarse por medio de una división adecuada de responsabilidades entre los elementos del sistema y la definición de una interfaz estable.
Extensibilidad	Mide la capacidad de adicionar funcionalidades a una aplicación en ejecución sin afectar negativamente el resto del sistema. La extensibilidad es inducida en una arquitectura mediante la reducción del acoplamiento entre los componentes.
Configuración	Mide el grado de flexibilidad de un sistema desplegado para modificar el comportamiento de los componentes.
Reutilización	Es la capacidad de utilizar los elementos de una arquitectura en otra aplicación sin la necesidad de modificarlos. Un débil acoplamiento entre componentes y la generalidad de sus interfaces son aspectos importantes para la reutilización de un sistema.
Visibilidad	Hace referencia a la capacidad de un componente de seguir o mediar el proceso de comunicación entre otros dos, con el fin de optimizar o adicionar características de seguridad. La visibilidad de un sistema puede mejorarse por medio de la generalidad de las interfaces, usando mensajes auto-descriptivos entre los componentes del sistema.
Portabilidad	Se refiere a la capacidad de un sistema de ser desplegado en diferentes entornos. Los estilos basados en código móvil permiten desarrollar aplicaciones altamente portables, como aquellas desarrolladas para máquinas virtuales.

Confiabilidad	Indica que tan propenso es un sistema a las fallas. La confiabilidad puede mejorarse evitando puntos únicos de falla y usando redundancia de componentes, entre otros aspectos. La confiabilidad también hace referencia a la integridad y validez de los datos.

Tabla 2. Propiedades soportadas en redes informáticas [32]

3.2.2. Principios arquitectónicos de REST

El estilo arquitectónico REST se encuentra estrechamente relacionado con la WWW (World Wide Web) y las características que son presentadas en esta sección, son seleccionadas debido a que afectan favorablemente propiedades del sistema, que se consideran necesarias para el cumplimiento de los requerimientos de la arquitectura de la Web. Estos requerimientos se basan en [32]:

Uso simplificado: El esfuerzo tanto para utilizar los servicios por parte del usuario, como para publicarlos por parte de los desarrolladores debe ser mínimo, facilitando la adopción y crecimiento de la Web, para ello se destacan las siguientes características:

- Interfaz de usuario basada en la hipermedia
- Protocolos simples para la creación y transferencia de datos
- Extensibilidad

Dominios administrativos múltiples: La Web debe funcionar a través de diferentes dominios administrativos. Por lo que se requiere:

- Escalabilidad anárquica
- Soporte de plataformas heterogéneas
- Soporte para cambios graduales y fragmentados

Sistema de hipermedia distribuido: La Web es un sistema de hipermedia distribuido por lo que se requiere:

- Eficiencia en la transferencia de grandes volúmenes de datos
- Sensibilidad a la latencia percibida por el usuario
- Capacidad para operar en condiciones de desconexión

A continuación se detallan cada uno de los principios arquitectónicos que definen el estilo REST.

A. Cliente-Servidor

Modelo que define la estructura básica de una arquitectura REST. En él, un servidor provee los servicios y escucha las peticiones de diferentes clientes. En el flujo normal de eventos, el servidor procesa las peticiones, ejecuta uno o más procesos y envía una respuesta. En el otro extremo, un cliente consume los servicios y realiza peticiones a los servidores. Esta separación de responsabilidades entre el cliente y el servidor simplifica la arquitectura, reduce la carga en los servidores, facilita la escalabilidad y permite la evolución independiente de los componentes.

B. No Orientado a Sesión

El modelo Cliente-Servidor no orientado a sesión mejora la escalabilidad, la visibilidad y la confiabilidad del sistema. La escalabilidad es mejorada, pues el servidor simplifica su estructura al no tener que almacenar ninguna información de sesión; la visibilidad, debido a que cualquier intermediario puede entender una petición sin necesidad de analizar peticiones previas; la confiabilidad, pues facilita la recuperación de fallos. Por otro lado, el rendimiento de la red se ve afectado debido a que en ciertas situaciones es necesario repetir información en cada una de las peticiones, como por ejemplo información de autenticación.

C. Caché

Un caché permite reutilizar el resultado de peticiones previas equivalentes marcadas como almacenables explícita o implícitamente. Su aplicación al modelo Cliente-Servidor puede realizarse en ambos extremos de la comunicación. Un caché en el lado del cliente reduce el número de peticiones enviadas hacia el servidor, mejorando así la eficiencia de la red; por el lado del servidor reduce el procesamiento de peticiones, reduciendo su carga y aumentado así la escalabilidad. Por otro lado, la confiabilidad se reduce en algunas circunstancias, en donde los datos almacenados en caché son inválidos.

D. Interfaz Uniforme

El énfasis en una interfaz uniforme es una de las características principales del estilo REST. Una interfaz uniforme precisa que cada uno de los componentes de la arquitectura soporte la misma interfaz, lo cual simplifica el sistema. La propiedad de evolución independiente es mejorada, pues es posible modificar la implementación de los componentes mientras se mantenga la interfaz generalizada. La visibilidad es mejorada pues la semántica de los mensajes es conocida por todos los componentes de la arquitectura, permitiendo la optimización por parte de intermediarios o terceras partes. Por otro lado, la eficiencia es reducida pues el formato de los mensajes es genérico y no aprovecha las características particulares del dominio de una aplicación.

El principio de interfaz uniforme requiere la aplicación de restricciones adicionales, con el fin de guiar el comportamiento de los componentes:

- Identificación de recursos.
- Manipulación de recursos a través de representaciones.
- Mensajes auto-descriptivos.
- La hipermedia como el motor del estado de las aplicaciones (HATEOAS) [33].

E. Sistema en Capas

Este sistema opera de tal forma que las capas inferiores provean servicios a las superiores, restringiendo el conocimiento de los componentes de una capa a los componentes de la subsiguiente. La introducción de este principio en REST permite adicionar elementos intermedios, como servidores proxy y pasarelas, con el fin de añadir cachés, funcionalidades de seguridad, aplicación de políticas, entre otras.

La propiedad de evolución independiente es mejorada pues permite cambiar la implementación de los componentes de una capa de forma independiente a las demás,

excepto para la capa adyacente. También se mejora la reutilización de los componentes, pues es posible encapsular sistemas legados para que puedan interactuar con las demás aplicaciones en la red, aprovechando las funcionalidades ya implementadas. Por otra parte, un sistema en capas aumenta la latencia del sistema pues aumentan los componentes y el procesamiento requerido para llegar hasta el servidor, por lo que se reduce el rendimiento percibido por el usuario.

F. Código bajo Demanda

En el código bajo demanda el servidor envía junto con la respuesta a una petición, un conjunto de instrucciones para procesar o manipular los datos. El código bajo demanda reduce la complejidad de los servidores, delegando parte del procesamiento de un servicio al consumidor. Esto mejora la escalabilidad del sistema, además de la eficiencia de la red y el rendimiento percibido por el usuario, pues el procesamiento se realiza localmente en el cliente y no requiere el envío de peticiones al servidor. La extensibilidad y capacidad de configuración son mejoradas pues el servidor puede realizar cualquier modificación al código enviado al cliente sin perjudicar su ejecución. El código bajo demanda es definido en REST como un principio opcional, debido a que reduce significativamente la visibilidad del sistema, pues el control de las interacciones pasa a ser manejado por el servidor solamente, además puede traer problemas de seguridad.

G. Resumen

Las reglas o principios definidos en REST inducen efectos tanto positivos como negativos en diferentes propiedades de un sistema. La siguiente tabla, derivada de [32], resume las propiedades afectadas por cada uno de los principios que conforman el estilo REST según lo descrito en las secciones previas, además del efecto acumulativo que cada uno de ellos tiene sobre la arquitectura completa. En la tabla, un efecto positivo sobre una propiedad es marcado con un signo más (+), mientras que un efecto negativo con un signo ménos (-). El resultado final para el estilo REST tiene en cuenta los efectos acumulativos positivos de los diferentes principios, de forma que pueda verse con claridad el énfasis del estilo arquitectónico, el cual como puede observarse es la escalabilidad del sistema (la propiedad con mayor número de incidencias positivas).

	Rendimiento de la Red	Rendimiento Percibido	Eficiencia de la Red	Escalabilidad	Sencillez	Evolución Independiente	Extensibilidad	Configuración	Reutilización	Visibilidad	Portabilidad	Confiabilidad
Cliente-Servidor				+	+	+						
No Orientado a Sesión	–			+						+		+
Caché		+	+	+								–
Interfaz Uniforme			–		+	+				+		
Sistema en Capas		–		+		+			+		+	
Código bajo Demanda		+	+	+	–		+	+		–		
REST	–	2+	2+	5+	2+	3+	+	+	+	2+	+	+

Tabla 3. Resumen de propiedades inducidas por REST

3.2.2.1. Elementos arquitectónicos de REST

Los elementos arquitectónicos de REST están divididos en tres categorías: datos, conectores y componentes.

A. Datos

El estilo arquitectónico REST se encuentra enmarcado en el dominio de los sistemas de hipermedia distribuidos. La hipermedia es definida, tradicionalmente, como una forma de estructurar información por medio de nodos multimedia relacionados mediante enlaces. En el contexto de REST, sin embargo, toma un significado menos estricto: se considera como hipermedia la combinación de información de control de una aplicación con la información de presentación[1]. Derivado de lo anterior, la hipermedia distribuida permite almacenar la información de control y presentación en sitios remotos.

Debido a la relación estrecha entre REST y los sistemas de hipermedia distribuidos, los datos constituyen un elemento esencial de este estilo arquitectónico. De ahí que la abstracción principal en REST sea un recurso. Un recurso encapsula un estado y lo oculta de los demás elementos por medio de una interfaz genérica. Cada recurso de interés en una aplicación, es referenciado por medio de un identificador, de forma que el estado de un recurso pueda transferirse de un componente a otro para su procesamiento. REST no permite el envío directo del estado de un recurso, sino que requiere hacerlo por medio de representaciones. Sus elementos se definen en la Tabla 4.

Elemento	Descripción
Recurso	Un recurso es la abstracción de información más importante en REST y se define como cualquier cosa que pueda ser nombrada o identificada. Por ejemplo, una persona, un libro, un documento digital o incluso, una colección de otros recursos. Un recurso es un concepto que hace referencia a un conjunto de entidades particulares en un periodo de tiempo determinado. Los recursos pueden ser estáticos o dinámicos. Un recurso permite asociar un identificador con un grupo de representaciones por medio de un concepto, ocultando las fuentes de información e implementaciones específicas.
Identificador de Recurso	Un recurso es referenciado por los componentes de la arquitectura por medio de un identificador y puede tener más de uno, los cuales son conocidos como alías.
Metadatos de Recurso	Información sobre un recurso independientemente del tipo de representaciones que posea.
Representación	Es la serialización del estado de un recurso. Un recurso puede tener una o más representaciones en diferentes formatos o tipos de medio. REST sólo permite manipular un recurso por medio de la transferencia de representaciones.
Metadatos de Representación	Información específica a una representación. Permite al receptor de un mensaje, procesar de forma correcta la representación.
Datos de Control	Los datos de control son información acerca del tratamiento que debe darse a una representación por el último receptor de un mensaje como también por componentes intermedios.

[1] La información de presentación no se refiere únicamente a la información que se muestra a un usuario, sino que también incluye cualquier tipo de información destinada para el consumo de un cliente.

Tabla 4. Elementos arquitectónicos de REST - Datos

B. Conectores

Un conector se encarga del proceso de comunicación entre dos componentes. Estos elementos no realizan ningún tipo de transformación o procesamiento de información, solo se enfocan en la transferencia de los datos manteniendo su integridad. En REST la interfaz de los conectores es genérica y el proceso de comunicación no es orientado a sesión.

Elemento	Descripción
Cliente	El inicia un proceso de comunicación generando peticiones y dirigiéndolas a un servidor.
Servidor	Escucha peticiones de los clientes y envía las respuestas respectivas.
Caché	En REST puede ser utilizado en elementos intermedios de forma que puedan reutilizarse algunas respuestas por diferentes clientes.
Resolución de nombres	Requerido por los clientes para determinar una dirección física de un recurso a partir de un identificador.
Túnel	Permite transmitir a través de una frontera de conexión como la impuesta por los cortafuegos.

Tabla 5. Elementos arquitectónicos de REST - Conectores

C. Componentes

Los componentes son aquellos elementos encargados del procesamiento de la información. La Tabla 6 muestra los componentes de REST.

Elemento	Descripción
Servidor de Origen	Mantiene el estado de los recursos y es el fin último de una petición que requiera modificar dicho estado. Este componente debe asignar identificadores a los recursos y mantener su validez en mayor tiempo posible.
Pasarela	Elemento intermedio controlado por un servidor o una tercera parte. Permite la implementación de arquitecturas en capa jerarquizadas, redirigiendo peticiones a diferentes servidores. También, facilita la aplicación de políticas en las fronteras de una organización con el fin de mantener la seguridad y la transformación de datos.
Proxy	Elemento intermedio escogido por un cliente para la transformación de datos, protección contra amenazas a la seguridad o aumento en el rendimiento del sistema.
Agente de Usuario	Es el consumidor final de las respuestas de un servidor y el origen de las peticiones. Debe representar la voluntad de un usuario, por lo que los elementos intermedios y conectores deben proveer la suficiente información para la toma de decisiones de un usuario.

Tabla 6. Elementos arquitectónicos de REST - Componentes

3.3. Definición de los servicios Web RESTful

Desde un punto de vista práctico, las aplicaciones basadas en REST que se observan más frecuentemente, son aquellas que utilizan la Web como su plataforma. Un ejemplo de ello, son los servicios Web RESTful. Este tipo de servicios siguen los principios planteados por REST y son implementados utilizando la infraestructura y tecnologías proporcionadas por la Web.

La Tabla 7 muestra las tecnologías típicas utilizadas para implementar los servicios Web RESTful de acuerdo a cada principio REST.

Principio	Implementación
Cliente-Servidor	HTTP
No Orientado a Sesión	HTTP
Caché	HTTP
Interfaz Uniforme	
Identificación de Recursos	URI
Representaciones	Tipos de Medio de Internet
Mensajes Auto-descriptivos	HTTP
Hipermedia	Modelo de enlaces e información de control adicional embebida en formatos de hipermedia y cabeceras HTTP
Sistema en Capas	HTTP
Código bajo Demanda	Java Script, Java applets

Tabla 7. Implementación de los principios de REST en los servicios Web RESTful

Es necesario aclarar que la utilización de una tecnología específica de la Web no garantiza el cumplimiento de los principios REST, pues también depende del modo en que sean utilizadas y cómo se diseñen las interacciones particulares de cada aplicación.

Un servicios Web basado en REST está compuesto, generalmente, por un conjunto de recursos referenciados mediante identificadores URI, cuyo estado es accedido y modificado usando el protocolo HTTP. Los recursos son manipulados a través de representaciones basadas en hipermedia (típicamente derivados del modelo de enlaces del formato Atom estructurados usando XML o JSON) usando un conjunto de métodos reducidos.

3.3.1. Identificadores URI

Los identificadores URI (Identificadores de Recursos Uniforme) son un estándar de la W3C usado como un mecanismo uniforme para la designación de nombres y obtención de recursos. Estos constituyen unas de las convenciones más comunes en las aplicaciones de Internet y en especial de la Web.

En el contexto de los servicios Web RESTful, proveen un mecanismo para referenciar los puntos de entrada de un servicio. Estos puntos de entrada son identificadores de recursos estables, que un cliente debe utilizar para obtener la información inicial que lo guíe en las siguientes interacciones necesarias para cumplir con su objetivo. En este proceso, los identificadores URI son embebidos en las representaciones de los recursos, convirtiéndose en una red de hipermedia: fundamento básico de la navegación en Internet. Un cliente debe procesar las representaciones en busca de enlaces (identificadores embebidos) para continuar el consumo del servicio, siendo de gran importancia en el contexto de REST, pues le permite al servidor evolucionar de forma independiente sin temor a romper las implementaciones de los clientes[2].

[2] El tratamiento de los identificadores URI como información opaca en el lado del cliente, no implica que el servidor no pueda instruir al cliente en cómo crear un identificador. Este caso es común y es aceptado por el estilo REST, siendo ejemplo de ello, los formularios HTML que especifican el método HTTP GET.

3.3.2. Protocolo HTTP

El protocolo HTTP está basado en la arquitectura cliente-servidor; no es orientado a la sesión por lo que las peticiones actuales son independientes de las peticiones anteriores; permite caché mediante intermediarios o directamente en los conectores cliente-servidor; su estructura cuenta con una cabecera, donde se envía información de configuración como metadatos y un cuerpo, en el que se envía el mensaje o representación de un recurso. Una de las partes más importantes de su definición son los diferentes métodos u operaciones dirigidas a manipular los recursos almacenados en un servidor remoto.

Los métodos HTTP presentan una semántica bien definida, lo cual facilita la utilización de mensajes auto-descriptivos, permitiendo así la optimización por parte de componentes intermediarios. Por ejemplo, HTTP define dos conceptos cuyo significado es la base del sistema de caché y de los mecanismos de recuperación de fallas parciales, a saber, la seguridad y la ídem-potencia (enviar una o más peticiones, generan el mismo resultado). Los métodos más usados en la implementación de servicios basados en REST, son descritos a continuación (Tabla 8, Tabla 9, Tabla 10, Tabla 11). En el Anexo A se describen más detalladamente.

Método:	**GET**
Operación común:	Leer
Operación CRUD:	READ
Seguro:	Si
Ídem-potente:	Si

Tabla 8. Descripción del método HTTP GET

Método:	**POST**
Operación común:	Crear
Operación CRUD:	CREATE
Seguro:	No
Ídem-potente:	No

Tabla 9. Descripción del método HTTP POST

Método:	**PUT**
Operación común:	Reemplazar o Crear
Operación CRUD:	UPDATE
Seguro:	No
Ídem-potente:	Si

Tabla 10. Descripción del método HTTP PUT

Método:	**DELETE**
Operación común:	Eliminar
Operación CRUD:	DELETE
Seguro:	No
Ídem-potente:	Si

Tabla 11. Descripción del método HTTP DELETE

Las respuestas HTTP obtenidas a partir de la aplicación de peticiones con los métodos ya descritos, contienen un código de estado que le permite a un cliente identificar el tipo de

resultado obtenido. La combinación de métodos seguros e ídem-potentes junto con el manejo de códigos de error con semántica definida, permite la construcción de aplicaciones distribuidas robustas. La siguiente tabla muestra los códigos más comunes.

Familia	Código	Frase descriptiva
Información	100	Continue
Éxito	200	OK
	201	Created
	202	Accepted
	204	No Content
Redirección	301	Moved Permanently
	302	Found
	303	See Other
	304	Not Modified
Error del Cliente	400	Bad Request
	401	Unauthorized
	403	Forbidden
	404	Not Found
	405	Method not Allowed
	409	Conflict
	410	Gone
	412	Precondition Failed
	415	Unsupported Media Type
	417	Expectation Failed
Error del Servidor	500	Internal Server Error
	501	Not Implemented
	503	Service Unavailable

Tabla 12. Códigos de respuesta HTTP más comunes

3.3.3. Formatos de hipermedia y tipos de medio

Las representaciones son secuencias de bytes de un recurso, siguiendo cierto formato específico. Algunos de los formatos más comunes en los servicios Web RESTful son XML y JSON. Este tipo de formato por sí sólo no es de gran ayuda en el entorno REST, pues no consideran construcciones nativas para la codificación de enlaces o hipermedia. Sin embargo, mediante la estandarización o extensión de ellos se pueden agregar instrucciones que permiten la adición de los enlaces u otra información de control, convirtiendo el formato en uno de hipermedia. Ejemplo de lo anterior es el modelo de enlaces definido para el formato Atom, el cual está basado en XML.

Un tipo de medio es la herramienta para la codificación del estado de un recurso, por lo que en él se define la semántica y el procesamiento esperado que un cliente puede realizar con la información recibida.

Un servicio Web RESTful debe especificar un tipo de medio para la codificación de sus recursos, ya sea por medio de la reutilización de tipos estandarizados (escenario ideal, pero no siempre posible) o la creación de un tipo de uso específico. Sin embargo, debe entenderse que el uso de un medio específico reducirá el número de clientes potenciales del servicio.

3.3.4. La hipermedia como el motor del estado de las aplicaciones

Un servicio Web basado en REST presenta un grupo de identificadores URI como puntos de entrada para los servicios. Las representaciones basadas en formatos de hipermedia, que son obtenidas mediante estas URI, contienen las transiciones a estados de la aplicación que se consideran válidos. Un cliente debe obtener la información de control, o más específicamente los enlaces de hipermedia incluidos en las representaciones, de forma que pueda escoger el enlace adecuado para modificar el estado de la aplicación. El proceso de acceso a los recursos y selección de enlaces, se repite la cantidad de veces necesarias de acuerdo con un protocolo especificado por el servidor de origen y se prolonga hasta que el agente de usuario cumpla con su objetivo.

3.3.5. Recursos como máquinas de estado

De acuerdo con [31] y [34], los recursos son en realidad máquinas de estado complejas, cuyo estado interno puede ser inspeccionado (por ejemplo, mediante HTTP GET) y modificado por medio de la interfaz uniforme (por ejemplo, mediante HTTP POST, PUT, DELETE). La Figura 4 ilustra lo anterior, con la máquina de estado interna que rige el ciclo de vida de una orden en el ámbito de un servicio de e-comercio. Los estados de la orden son: “vacía”, “lista”, “confirmada”, “enviada” y “cancelada”. El estado inicial es “vacía” y la interacción que inicia su ciclo de vida es un mensaje POST a la URI /order. A continuación se detallan las transiciones entre los estados.

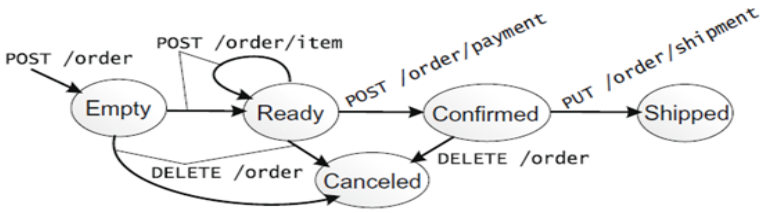


Figura 4. Estado de una orden gestionada por una tienda [35]

Estado “vacía”: En este estado la orden ha sido recién creada y no posee ningún producto.

- La orden pasa al estado “lista” al adicionar un producto mediante el envío de un mensaje POST a la URI /order/[ordeId]/ítem.
- La orden pasa al estado “cancelada” al enviar un mensaje DELETE a la URI /order/[orderId].

Estado “lista”: En este estado la orden posee productos y puede ser confirmada.

- La orden pasa al estado “confirmada” realizando el pago correspondiente mediante el envío del mensaje POST a la URI /order/[orderId]/payment.
- La orden pasa al estado “cancelada” al enviar un mensaje DELETE a la URI /order/[orderId].
- Aunque esta transición no se muestra en el diagrama, la orden pasa al estado “cancelada” si transcurre un tiempo de inactividad especificado en la lógica interna del recurso.

Estado "confirmada": En este estado la orden ha sido pagada y puede ser enviada.

- La orden pasa al estado "enviada" confirmando los detalles de envío enviando un mensaje PUT a la URI /order/[orderId]/shipment.
- La orden pasa al estado "cancelada" al enviar un mensaje DELETE a la URI /order/[orderId]

Estado "enviada": Es el estado final del recurso y representa el fin de su ciclo de vida. En este estado no hay más posibles transiciones.

Cabe aclarar que en cualquier momento es posible inspeccionar el estado de la orden mediante un mensaje GET a la URI /order/[orderId]. La URI concreta es obtenida como respuesta al primer mensaje que crea la orden, es decir, a la petición POST inicial a la URI /order. En REST al inspeccionar un recurso, aparte de obtener una representación del estado del recurso, se identifican también, las relaciones entre los recursos y las posibles transiciones de acuerdo a dicho estado. En este caso, las transiciones disponibles en las representaciones de la orden son dinámicas, es decir, dependen del estado interno del recurso. Por ejemplo, en el estado "lista", la representación obtenida tendrá transiciones (controles embebidos de hipermedia), para adicionar un producto a la orden, confirmar la orden mediante un pago o cancelar la orden, mientras que en el estado "confirmada", los controles disponibles serán para confirmar el envío o para cancelar la orden.

3.4. Descripción de los servicios Web RESTful

La descripción de los servicios Web RESTful, deben ser lo más estables y flexibles, de forma que no se impida el consumo de los servicios desplegados, pero se permita la evolución independiente del servicio. En [31] se establecen los elementos que se deben tener en cuenta para la descripción.

- **Identificadores URI de entrada:** Un servicio REST presenta un conjunto de reducido de identificadores URI bien definido con los cuales se espera que un cliente inicie una interacción con un servicio.
- **Tipos de medio:** Específica el formato de los mensajes y representaciones esperados para que sean intercambiados entre el servicio y el consumidor.
- **Idiomas HTTP:** Los idiomas HTTP establecen que métodos, cabeceras y códigos de estado serán manejados en una interacción con una representación específica.

3.4.1. Lenguajes de descripción para los servicios Web RESTful

Los servicios Web basados en REST son descritos en su mayoría usando lenguaje natural, este tipo de descripciones están orientadas a los desarrolladores, por lo que no pueden ser procesadas automáticamente. Aunque existen formatos preparados para la interacción entre máquinas, aunque no han tomado mucha fuerza en la Web debido a la ausencia de un estándar, pero son necesarios si se desea automatizar algunas de las tareas con los servicios basados en REST, como su invocación, descubrimiento y composición. Por esta razón se presentan las descripciones formales.

3.4.1.1. WSDL 2.0[3] [36]

WSDL (Web Services Description Language) es el lenguaje estándar para la descripción de servicios Web tradicionales. Permite especificar las capacidades funcionales de un servicio por medio de documentos en formato XML con un modelo particular, los cuales contienen la información necesaria para el consumo de un servicio por parte de un cliente.

3.4.1.2. WADL [37]

WADL (Web Applications Description Language) es uno de los lenguajes más extendidos para la descripción de servicios basados en REST y más específicamente en las peticiones y respuestas HTTP. Este lenguaje permite describir:

- **Conjunto de recursos:** Recursos ofertados por un servicio Web basado en REST.
- **Relaciones entre recursos:** Los enlaces presentes en las representaciones de los recursos.
- **Métodos que pueden ser aplicados a cada recurso:** Métodos HTTP permitidos por cada recurso, sus entradas y salidas.
- **Formato de las representaciones de los recursos:** Especificación del formato del tipo de medio.

WADL permite la creación automática de código en el cliente para la generación de peticiones (stubs). Pero su enfoque en un estilo RPC (métodos permitidos por cada recurso, con información específica de las entradas y salidas), no está muy acorde con los principios de REST. Por ejemplo, el uso de WADL para la creación de "*stubs*" genera mayor acoplamiento entre el cliente y el servidor, aumentando la probabilidad de inutilizar los clientes en el proceso de actualización de un servicio.

Además WADL se enfoca más en los métodos soportados por cada recurso que en la descripción de los enlaces y controles de hipermedia encontrados en las representaciones. Esto va en contra del principio de la hipermedia (HATEOAS), pues permite obtener información fuera de banda (en tiempo de diseño) sobre los recursos y no a través de la hipermedia (en tiempo de ejecución). Asimismo no tiene en cuenta que los mensajes en REST deben ser auto-descriptivos, por lo que la información debe ser descubierta en las representaciones de los recursos y no en un documento estático.

3.4.1.3. hRESTS [38]

La adaptación de la semántica XHTML hace más fácil publicar, indexar y extraer información semi-estructurada en la Web, esto es conocido como micro formatos [9], lo cual permite crear una descripción que puede ser leída por máquinas y así automatizar diferentes mecanismos, de una manera más ligera.

HTML para REST (hRESTS), es un micro formato basado en un modelo de servicio simple en RDF (Resource Description Framework), que captura descripciones de los servicios procesable por máquina, basándose en la documentación HTML del servicio dirigido a los desarrolladores. hRESTS cuenta con una extensión para la automatización

[3] Abreviado como WSDL

de servicios Web semánticos, la cual es MicroWSMO (Figura 5), aunque puede soportar otra extensión, tal como SA-REST (Semantic Annotation of Web Resources) [39].

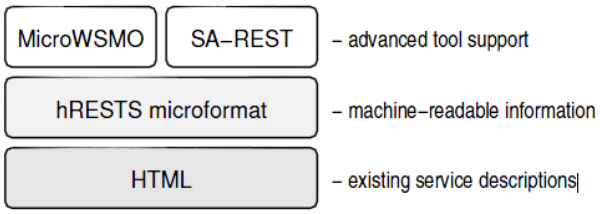


Figura 5. Interacción entra hRESTS y MicroWSMO [38]

3.4.1.4. ReLL [40]

ReLL describe un servicio RESTful por medio de recursos, representaciones y tipos de enlace, su esquema se puede observar en la Figura 6. Los recursos pueden tener o no un identificador URI, teniendo en cuenta si son puntos de entrada, Cada uno de ellos puede tener diferentes representaciones, las cuales permiten describir el formato de un tipo de medio, un esquema y diferentes enlaces; tales enlaces poseen reglas que permiten extraer una URI de una representación usando tecnologías como XPath para representaciones basadas en XML, JSONPath para representaciones basadas en JSON, expresiones regulares, o cualquier otro tipo de tecnología dirigida a extraer información de documentos. Dentro de la especificación de un enlace, es posible identificar aspectos relacionados con el protocolo específico que debe utilizarse en la interacción con el recurso al cual apunta el enlace. Aunque la descripción no está dirigida a algún protocolo en específico, el más utilizado en la Web es HTTP.

Las representaciones que no se encuentren definidas en el ámbito de un recurso, son consideradas representaciones abstractas y se utilizan como un mecanismo para extender los tipos de enlace soportados por una representación. Por ejemplo, se puede definir una representación abstracta donde se especifique un formato como XML (application/xml), posteriormente mediante otra representación abstracta se puede añadir un modelo de enlaces como el especificado en el formato Atom (application/atom+xml), igualmente este último tipo de representación puede extenderse para soportar otros enlaces o relaciones definidas en AtomPub.

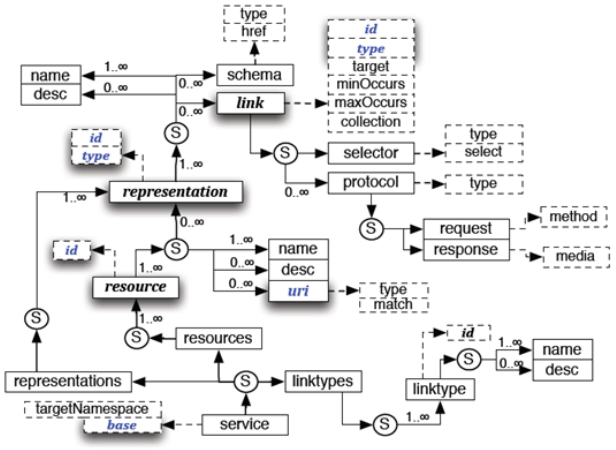


Figura 6. ReLL esquema [40]

3.4.1.5. Resumen y análisis

La Tabla 13 resume el análisis de los diferentes lenguajes de descripción para los servicios Web RESTful, teniendo en cuenta las construcciones de lenguaje soportadas (recursos, representaciones, enlaces, detalles de una transacción petición/respuesta) y el modelo tomado como base, ya sea RPC (Remote Procedure Call) o REST. Se realiza una valoración de cada lenguaje de acuerdo a la siguiente escala: i) Construcción soportada (+), ii) Construcción soportada parcialmente (+/-) y iii) Construcción no soportada específicamente (-). La metodología utilizada para desarrollar esta tabla se basa en el análisis de los diferentes esquemas de datos de los diferentes lenguajes de descripción presentados.

	WSDL 2.0	WADL	hRESTS	ReLL
Recursos	-	+	-	+
Representaciones	+	+	+/-	+
Enlaces	-	-	-	+
Petición /Respuesta	+	+	+	+/-
Modelo	RPC	RPC, REST	RPC	REST

Tabla 13. Valoración lenguajes de descripción para servicios Web RESTful

3.5. Composición de servicios Web RESTful

La composición de servicios Web basados en REST es el proceso de combinar diferentes servicios en uno único, con el fin de dar respuesta a peticiones complejas que no puedan ser satisfechas por los servicios individualmente. El proceso de composición en otros tipos de servicios Web como los basados en SOAP se encuentra bien definido, mediante estándares como WS-CDL (Coreografía) y WS-BPEL (Orquestación).

La coreografía en el contexto de los servicios Web tradicionales, hace referencia a la colaboración entre diferentes servicios Web, donde ninguno de los participantes tiene un control sobre el proceso en conjunto y donde cada uno de los servicios cumple un rol en especificico, con el fin de satisfacer una necesidad o proveer una funcionalidad. Por otro lado, la orquestación, aunque igual que en la coreografia, integra diferentes servicios Web, se diferencia en que existe un servicio central que dirige la interacción de los demás servicios.

Estos dos principios son de gran importancia en el contexto SOA, sin embargo, en el campo de REST han sido poco explorados, ocasionando la ausencia de un lenguaje de composición estándar.

3.5.1. Requisitos para la composición de servicios Web RESTful.

Un servicio compuesto debe poseer las mismas características externas que los servicios que lo componen, es decir debe ser también un servicio Web basado en REST, por lo que no puede romper ninguna de las reglas ya establecidas para estos.

En el trabajo de Pautasso [41] se proponen los siguientes requisitos para un lenguaje de composición de servicios Web RESTful basado en BPEL de acuerdo con los principios de REST (Tabla 14).

Principio	Requisitos
Recursos identificados por URI	• Enlace dinámico a identificadores URI • La especificación de la lógica que rige las transiciones de estado de un recurso compuesto
Interfaz uniforme	• La invocación síncrona de servicios (interacción petición-respuesta) usando los métodos HTTP
Mensajes auto-descriptivos	• El acceso a las cabeceras HTTP para la manipulación de metadatos
Principio de la hipermedia	• La generación y posterior envío de nuevos identificadores URI a los clientes RESTful • Enlace tardío de las peticiones con identificadores URI dinámicos extraídos de las representaciones recibidas

Tabla 14. Requisitos para la composición de servicios Web RESTful

En las siguientes secciones se describen un conjunto de lenguajes para la composición de servicios RESTful.

3.5.2. BPEL para REST

En el trabajo de Pautasso [41] se plantea una extensión de WS-BPEL para la composición de servicios Web tradicionales y servicios Web basados en REST. La extensión desarrollada permite invocar métodos HTTP con el fin de manipular los recursos publicados por los proveedores de servicios. Además, permite exponer los procesos BPEL como servicios Web consumibles por clientes RESTful. Estas dos posibilidades tienen su origen, en el hecho de que la composición de servicios Web es recursiva, por lo que un servicio RESTful compuesto es igual en naturaleza a los servicios que lo constituyen.

El trabajo propone a través de la extensión presentada, adicionar las actividades <get>, <post>, <put> y <delete> al conjunto de actividades de WS-BPEL. La ejecución de cada una de estas actividades requiere la especificación de una URI estática (en tiempo de compilación) o dinámica (en tiempo de ejecución a través de una variable). Además, la extensión define el elemento <header>, usado para manipular los metadatos en las actividades de invocación HTTP y el atributo response_headers, usado para obtener los metadatos de las respuestas recibidas al invocar un servicio RESTful. Por otra parte, la exposición de los procesos como recursos RESTful es soportada por la extensión con un elemento de tipo contenedor, denominado <resource> y diferentes manejadores, <onGet>, <onPut>, <onPost> y <onDelete>, que corresponden a los posibles métodos HTTP que un cliente RESTful puede generar para consumir un servicio.

El método de composición presentado es manual y estático, por lo que requiere la selección manual de cada uno de los servicios que constituirán el proceso a ejecutar y no puede ser aplicado directamente en un entorno móvil.

3.5.3. Bite

En el trabajo de Curbera [42] se presenta un lenguaje de composición ligero y extensible, denominado Bite, que permite la creación de flujos de trabajo en la Web y utiliza los servicios RESTful como entidades de composición principales. Bite se caracteriza por

soportar la integración entre la Web y la interacción humana, siendo capaz de construir flujos de trabajo colaborativo.

EL núcleo básico de actividades que identifican el modelo del lenguaje se encuentran divididas en tres grupos: i) actividades de comunicación primitiva HTTP para recibir y responder peticiones (<receiveGET/POST>, <replyGET/POST>) y también para generar peticiones a servicios externos (<GET>, <POST>, <PUT>, <DELETE>); ii) actividades utilitarias, para esperar e invocar código local o terminar un flujo de ejecución; y iii) actividades para el control del flujo, como los ciclos y condicionales.

El método de composición presentado es manual y estático, por lo que requiere la selección directa por el usuario, de cada uno de los servicios que constituirán el proceso a ejecutar y no puede ser aplicado directamente en un entorno móvil cambiante.

3.5.4. Composición basada en LPML y hRESTS

LPML (Lightweight Process Modeling Language) desarrollado en el marco del proyecto SOA4ALL [43] permite modelar un proceso de negocio basados en un flujo de control y un flujo de datos, usando un subconjunto de notaciones BPMN (Business Process Modeling Notation) [44]. Las construcciones de flujo de control, permiten dirigir la ejecución de las actividades, permitiendo patrones como ejecución en secuencia, ejecución en paralelo, sincronización, entre otras. Las construcciones de flujo de datos como unión, separación, contador, filtro, reducción, entre otras, permiten realizar composiciones basadas en mashups [45].

Una vez se haya especificado el proceso de negocio usando LPML, este es transformado a un lenguaje de composición concreto como BPEL 2.0, extendido con las construcciones necesarias para soportar tanto los servicios basados en WSDL, como los servicios REST descritos usando hRESTS.

El enfoque en hRESTS y en el soporte de servicios basados en WSDL hace que la composición basada en LPML no tenga en cuenta gran parte de los requerimientos planteados para un lenguaje de composición para servicios Web basados en REST.

3.5.5. BPEL Orientado a Recursos (RBPEL)

En la investigación de Overdick [34] se presenta una extensión al estándar WS-BPEL con el fin de incluir los conceptos de ROA (Resource Oriented Architecture) basada en REST y en especial, modelar el ciclo de vida interno de un recurso.

RBPEL permite expresar o modelar el comportamiento interno de un recurso usando una versión ligera de WS-BPEL desligada de WSDL. El concepto de estados es manejado usando la construcción <scope> (ámbito). Las peticiones realizadas a un determinado recurso son recibidas por manejadores de eventos, que para el caso de los métodos seguros e ídem-potentes, en RBPEL no ocasionan cambios de estado. Por otro lado, la recepción de un mensaje POST genera cambios de estado, es decir, cambios de ámbito.

Para la creación y obtención de identificadores URI, RBPEL usa funciones XPath basadas en plantillas URI. Las URI creadas dinámicamente son asignadas a variables internas mediante la construcción <assign>, las cuales pueden utilizarse posteriormente en las respuestas enviadas a los clientes.

3.5.6. Resumen

La Tabla 15 muestra un resumen de los lenguajes de composición de servicios Web RESTful previamente presentados, valorados de acuerdo a los requisitos presentados al iniciar la presente sección. La escala se define como: i) Requisito soportado (+), ii) Requisito parcialmente soportado (+/-) y iii) Requisito no soportado (-).

	BPEL4REST	BITE	LPML/hRESTS	RBPEL
Enlace tardío	+	-	-	-
Estado de los recursos	+	-	-	+
Manejo del protocolo	+	+	+/-	+
Generación de URI	+	-	-	+
Flujo de control y flujo de datos	+/-	+	+	+

Tabla 15. Valoración lenguajes de composición de servicios Web RESTful

3.6. Composición dinámica de servicios Web RESTful en un entorno Mobile 2.0

La composición dinámica se caracteriza por la flexibilidad que ofrece en entornos cambiantes. A diferencia de la composición estática, la cual se realiza en tiempo de diseño: donde los servicios son seleccionados, interconectados, compilados y desplegados [46]; la composición dinámica se realiza en tiempo de ejecución, permitiendo la adaptación de los servicios compuestos ante fallas eventuales en el entorno.

Los mecanismos de composición presentados en la sección anterior son considerados estáticos. Este tipo de composición es viable en entornos poco cambiantes, donde la evolución, adición y eliminación de los servicios por parte de los proveedores sea mínima, permitiéndole al servicio compuesto mantener su funcionalidad sin caer en inconsistencias. Sin embargo, en entornos abiertos como la Mobile 2.0, en los que diferentes actores (proveedores y usuarios) pueden contribuir con contenido o servicios y en los que la movilidad de los usuarios puede ocasionar un cambio constante en sus necesidades de acuerdo a su contexto, es necesario proveer mecanismos de composición dinámicos.

La composición dinámica facilita la adaptación de los servicios compuestos de acuerdo con las necesidades del usuario, su entorno y los servicios individuales disponibles. Dicha adaptación, de acuerdo con [47, 48] , se produce como una consecuencia de dos tipos de cambios: estructurales y de comportamiento.

Los cambios estructurales son aquellos que no ocasionan una modificación de la funcionalidad del servicio compuesto. Éstos se presentan, por ejemplo, cuando un servicio individual que hace parte de un servicio compuesto deja de funcionar, por lo que debe ser reemplazado por otro, que cumpla funcionalidad semejante. Este tipo de adaptaciones busca mantener o mejorar la calidad (QoS) prestada al cliente, por lo que requiere de un monitoreo de las propiedades no funcionales de los servicios individuales.

Los cambios de comportamiento son aquellos que requieren la modificación de la funcionalidad del servicio compuesto para realizar la adaptación. Este tipo de cambios permiten adicionar o eliminar funcionalidades sin la necesidad de realizar el proceso de

composición completo. Los procesos de adaptación originados por cambios de comportamiento están relacionados con el usuario.

Por otro lado, la composición dinámica en un entorno Mobile 2.0, debe tener en cuenta los servicios específicos de un dispositivo móvil, como por ejemplo, ubicación mediante GPS, capacidades de comunicaciones, entre otras. Estos servicios no están basados en REST y dependen de la plataforma específica de cada dispositivo. Por lo que para obtener acceso a estas capacidades en un proceso de composición se requieren desarrollar mecanismos estándar mediante la Web.

Una de las opciones presentadas en la literatura para el acceso a las capacidades e información de un dispositivo móvil, es la provisión directa de servicios Web basados en REST [49]. Este mecanismo requiere implementar toda la pila de un servidor HTTP en el dispositivo, con el fin de procesar peticiones e invocar los servicios específicos requeridos, retornando la información deseada por un cliente, lo cual, le permite a un servicio compuesto aprovechar la información de diferentes usuarios móviles, pero puede ocasionar problemas de rendimiento de la red y seguridad. Este mecanismo permite tratar todos los servicios, ya sean proveídos por dispositivos móviles o por servidores tradicionales en la Web, como servicios RESTful, simplificando el desarrollo del algoritmo de composición.

Otra alternativa menos explorada es el uso de CoD (Code-on-Demand), el cual es un estilo de código móvil que hace parte de los principios del estilo arquitectónico REST. CoD permite extender la funcionalidad de un cliente mediante el envío de código. En una arquitectura cliente/servidor tradicional, la lógica de los servicios es poseída por el servidor, quien recibe las peticiones del cliente y responde de acuerdo con su lógica interna. Adicionando el estilo de código móvil, es posible transmitir al cliente parte de dicha lógica, extendiendo las funcionalidades de forma dinámica [32]. En la Web la forma más extendida de CoD son las instrucciones Java Script, las cuales se encuentran junto a las representaciones de los recursos enviadas por un servidor y son ejecutadas internamente por el cliente, permitiendo su fácil extensión y configuración.

Este mecanismo, al estar basado en tecnologías de la Web no requiere la creación de código específico para cada plataforma, por lo que se facilita el desarrollo. Esto está siendo aprovechado para el desarrollo de marcos que le permiten a los creadores de aplicaciones ignorar la fragmentación de las plataformas móviles y la diversidad de lenguajes (Android-Java, iPhone/iPad-Objective-C, Symbian-Qt-Java ME-Flash, entre otras). También, esta alternativa en el contexto de composición, es más ligera, pues no requiere implementar la pila completa de un servidor HTTP.

Capitulo 4

Arquitectura para la Ejecución de Servicios Web RESTful Compuestos

En este capítulo se plantea una arquitectura para la ejecución de un servicio Web RESTful compuesto y la interacción directa con el usuario experto, que realiza la petición inicial.

Para cumplir con uno de los objetivos planteados en este trabajo de grado, es preciso realizar un estudio previo sobre algunas de las arquitecturas descritas en investigaciones realizadas por diferentes organizaciones o individuos, las cuales se convierten en la base para obtener la mejor adaptación ante la ejecución de servicios Web RESTful compuestos.

4.1. Arquitecturas Base

En el trabajo de F. Lécué*, et al* [50] se propone una arquitectura para el desarrollo de un framework, en el cual se define un ACE (*Automatic Composition Engine*) que contiene cuatro componentes básicos, Analizador Semántico, Fabrica de Composición, Modulo para agregar Propiedades y Enlazador Figura 7.

Esta arquitectura está integrada por diversos bloques que permiten tomar la solicitud del usuario en lenguaje natural y conseguir un resultado como servicio compuesto que se adapte a dicha solicitud. Este Framework tiene la siguiente secuencia de funcionamiento:

- Toma la solicitud en lenguaje natural y la procesa para obtener una solicitud formal en donde se especifiquen las entradas y/o precondiciones, las salidas y/o efectos, metas, propiedades no funcionales y ontologías del servicio Web que el usuario desea componer. Este paso tiene como fin convertir los requisitos de usuario de lenguaje natural, en lenguaje comprensible por maquina.
- Con base a la solicitud formal, realiza la composición, utilizando los servicios que previamente se han descrito y que se encuentran en un repositorio. Este bloque de composición permite la creación automática de servicios compuestos mediante el uso de un mecanismo basado en una Matriz de Enlace Causal (CLM, *Causal Link Matriz*).
- Los pasos siguientes, son los de Enlace y el de añadir Propiedades No Funcionales, los cuales se encargan de unir los servicios simples para conformar el compuesto y agregar una descripción para dicho servicio respectivamente, con el objetivo de entregar el servicio solicitado por el usuario para su ejecución y al mismo tiempo, al desarrollador de servicios la descripción para tenerla en cuenta al momento de realizar una nueva solicitud formal.

En el trabajo de K. Boumhamdi and Z. Jarir [51] se define una arquitectura para la composición dinámica, cuyo objetivo principal es hacer la composición de servicios Web más flexible y adaptable de acuerdo al perfil de usuario, con lo cual se aborda la calidad del servicio compuesto y los requerimientos del cliente en tiempo de ejecución. Como lo muestra la Figura 8.

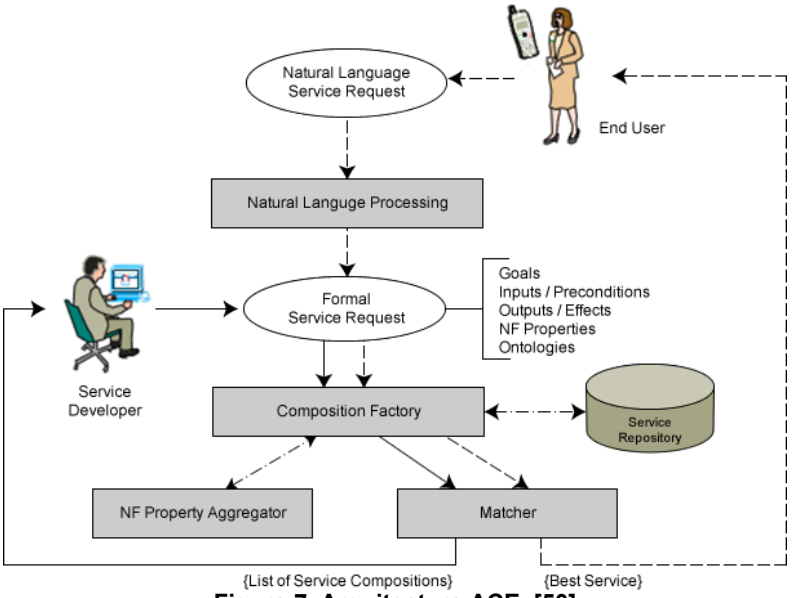


Figura 7. Arquitectura ACE [50]

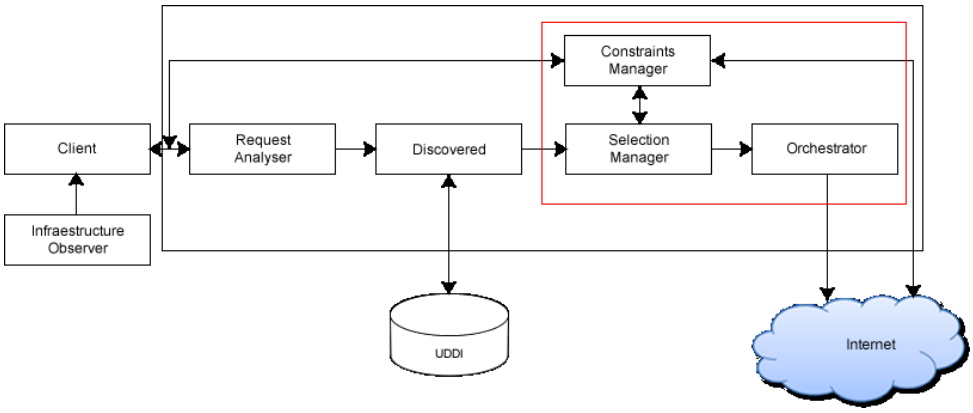


Figura 8. Arquitectura Propuesta en [51]

La arquitectura presenta cinco bloques importantes 1) Analizador de Solicitudes, 2) Descubrimiento, 3) Gestión de Límites, 4) Gestión de Selección y 5) Orquestador, las cuales inter-operan de la siguiente manera:

El objetivo del analizador de solicitudes recae en estudiar las necesidades ingresadas por los usuarios e identificar sus requerimientos, este proceso se realiza mediante un analizador léxico y semántico, al igual que lo hace el bloque de lenguaje natural de la arquitectura descrita en [50].

El bloque siguiente corresponde a Descubrimiento, el cual permite buscar y localizar servicios publicados en el UDDI, teniendo en cuenta los requerimientos obtenidos del bloque anterior. Los tres bloques siguientes abordan los procesos de composición y ejecución del servicio compuesto, el primero de ellos es el de Gestión de Limites, que es el responsable de verificar la disponibilidad de los recursos críticos (recursos importantes

para la ejecución del servicio) y detectar fallas del servicio compuesto en tiempo de ejecución. Este componente también es responsable de detectar cambios dinámicos en las preferencias de usuario, las cuales son notificadas al bloque de Gestión de Selección que se encarga de procesar dicha notificación y realizar los cambios pertinentes en el servicio compuesto; este bloque de selección también tiene como función, realizar el plan inicial del Servicio Compuesto.

Por último se encuentra el bloque encargado integrar las ejecuciones de los servicios individuales, siguiendo los lineamientos del grafo compuesto y colocarlo a dispocion del usuario; este bloque se denomina Orquestador.

4.2. Caracterización del Escenario

Teniendo en cuenta lo descrito en el Capítulo 2 y Capítulo 3 del presente documento y las arquitecturas base ya planteadas, es necesario definir el escenario en el cual se desplegara la arquitectura para la ejecución de servicios Web RESTful compuestos.

El uso de tecnología móvil inalmabrica, actualmente implica enfrentarse a multiples plataformas y sistemas operativos que se presentan; problema que se mitiga, por ejemplo, con el uso de códigos en Java Script con los cuales es posible acceder a las características particulares de los móviles, como por ejemplo, los sensores de manera generalizada.

Existen diferentes tipos de navegadores para los dispositivos móviles, como lo muestra la Tabla 16. En donde se aprecia, que desde la clase A hasta la clase C se adaptan a los requerimientos de ejecución, siendo el primero el más idóneo para los propósitos de este proyecto.

Clase	Lenguaje de Marcas	CSS	Soporte Java Script
A	XHTML, XHTML-MP, HTML5	CSS2, CSS3	Completo, incluye DHTML, Ajax
B	XHTML, XHTML-MP	CSS2(Bueno)	Limitado, algo de DHTML
C	XHTML, XHTML-MP	CSS2(Limitado)	Limitado
D	XHTML-MP	CSS2(Básico)	Ninguno
F	XHTML-MP, WML	Ninguno	Ninguno

Tabla 16. Tipos de navegadores Web móviles [19]

Además de estas características técnicas internas al dispositivo para la ejecución del servicio, es necesario tener en cuenta el contexto en el que se encuentra, e identificar las características que presenta, tales como tipo de red inalámbrica para el acceso a la Web, características no funcionales del dispositivo móvil, requerimientos particulares de usuario, etc.

El servicio compuesto RESTful estará orientado a satisfacer las necesidades del usuario de la manera más aproximada posible a sus requerimientos, con el fin de cumplir con una de las premisas más importantes de la Mobile 2.0, cuyos requerimientos son de tres tipos como se muestra en la Figura 9, los cuales son: i) Requerimientos personales de usuario: Están limitados a los requerimientos especificados en la petición inicial de usuario experto, la cual difiere entre individuos, ii) Requerimientos de dispositivo: Están dados por las características no funcionales del dispositivo móvil, tales como formatos multimedia, método de acceso a la red, resoluciones de pantalla, etc. y iii) Requerimientos del medio:

Debido a la inestabilidad y no uniformidad del medio, los requerimientos para el servicio compuesto varían, con el fin de que este se adapte a características, tales como tecnologías de las redes o métodos de acceso a Internet. Como la composición es dinámica, el propósito es adaptar el servicio compuesto a los requerimientos del usuario en tiempo de ejecución, los cuales se pueden considerar como la adición, la eliminación y la sustitución de servicios Web RESTful simples, o la reconfiguración automática de los mismos debido a dos tipos de fallas: i) La primera cuando uno o más servicios simples pertenecientes al compuesto fallan, lo cual se controla mediante su reemplazo y ii) La segunda debido al medio inalámbrico, teniendo en cuenta tan solo el tipo de red y su velocidad, ya que el servicio compuesto podría sufrir ralentizaciones en ejecución.

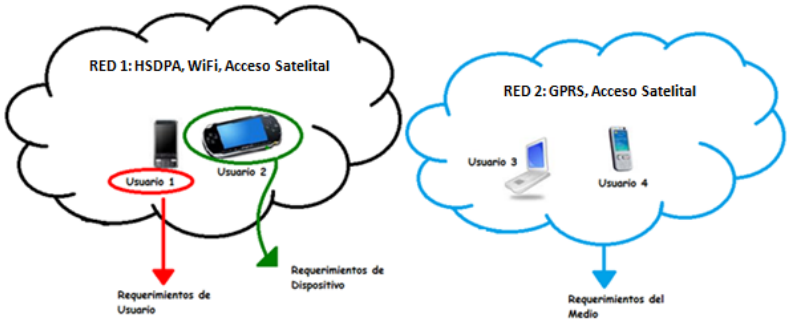


Figura 9. Tipos de requerimientos del sistema

Para reconocer el medio en el que se encuentra el dispositivo y generar representaciones de los servicios acordes a ese medio, se realiza un control de fallas ligado a las características físicas del dispositivo y el contexto, mediante la detección de ciertas capacidades del equipo y la medida de la velocidad de transporte de datos a través del medio inalámbrico.

4.2.1. Detección de Dispositivos

La detección de los dispositivos es una parte muy importante en este sistema, debido a la gran cantidad de dispositivos móviles presentes en el mercado mundial, que van desde GPS instalados en vehículos automotores, teléfonos celulares con funcionalidades sorprendentes y hasta dispositivos que reúnen todo en un mismo aparato. Por lo tanto es seguro encontrar una diversificación muy extensa en cuanto a características tanto funcionales como no funcionales dentro de esta gama de dispositivos, con lo cual, se hace necesario adaptar el contenido generado por el algoritmo de composición, a dichas características particulares, teniendo en cuenta al mismo tiempo, la red inalámbrica en la que se encuentre el usuario, minimizando con esto los problemas de interoperabilidad y usabilidad del contenido creado que siempre ha presentado el mundo móvil.

Existen diversas herramientas que permiten la detección de las características del dispositivo de un usuario por parte del servidor, entre ellas las más importantes son:

- UAProf 2.0
- CC/PP 2.0
- WURFL

A continuación se describe la herramienta seleccionada por sus características favorables para el sistema, en el Anexo B se encuentra detalladamente la especificación de las restantes.

4.2.1.1. WURFL

WURFL (*Wireless Universal Resource Files*) es un repositorio para la descripción de servicios o DDR (*Device Description Repository*), más específicamente es un archivo de configuración XML que contiene capacidades para miles de dispositivos móviles, que funciona mediante una serie de APIS que se soportan en una variedad de lenguajes de programación tales como Java, .Net, PHP, etc., para acceder a entornos de ejecución en tiempo real.

Una de las principales ventajas de WURFL con respecto a las herramientas propuestas anteriormente, es que cualquier persona o fabricante puede contribuir con el enriquecimiento del DDR, con lo cual se crea una base de datos de manera estandarizada y única. Además utiliza un agente de usuario para reconocer el dispositivo móvil teniendo en cuenta la información del DDR en tiempo de ejecución, mediante un archivo XML [52], con esto se soluciona la heterogeneidad entre dispositivos, aunque se pierde el manejo semántico. El documento XML en donde se encuentran las capacidades de cada dispositivo, está disponible en [52], el cual es sometido a constantes actualizaciones y a estandarización.

Para el presente trabajo de grado se implementa la API de java de WURFL basada en Spring [53], permitiendo la interacción con el fichero XML de WURFL, haciendo uso de ciertas características presentadas en el Anexo .

4.3. Arquitectura Propuesta

El entorno móvil sobre el cual se desarrolla este proyecto, permite la interacción de cualquier tipo de dispositivo que se adapte a ciertos requerimientos mínimos, tales como la posibilidad de acceder a la red de forma inalámbrica o la movilidad junto al usuario y navegar a través de la Web con un navegador móvil., lo que permite plantear un contexto general para definir una arquitectura del ambiente para la composición de servicios Web de manera dinámica, como lo muestra la Figura 10.

Figura 10. Esquema general para la composición de servicios Web RESTful dinámica

El trabajo principal de este proyecto de grado, se centra en el bloque correspondiente al servidor para la composición, cuyo componente principal es el algoritmo que desempeña la función de componer servicios Web RESTful; este bloque interactúa de manera directa con el usuario experto para realizar su proceso de forma dinámica, intercambiando peticiones y respuestas basadas en un modelo cliente servidor. Los componentes que integran el servidor (Figura 11) surgen de la adaptación y combinación de otras

arquitecturas analizadas, las cuales se basan en SOAP o servicios Web tradicionales, que mantienen un concepto de composición dinámica similar al planteado en este trabajo de grado; cuya diferencia principal radica en el tratamiento que se le da a los servicios. Estos servicios que se basan en recursos, permiten establecer peticiones muy bien definidas, pero representaciones no preestablecidas, que indican la falta de un camino o ruta que precise de antemano una plantilla para lograr una composición que se adapte a los requerimientos del usuario, por lo tanto las semejanzas llegan tan solo a un nivel conceptual.

Independientemente de las diferencias que puedan existir entre los servicios Web RESTful y los servicios Web tradicionales, la solución al problema de la composición dinámica debe considerar los siguientes aspectos:

- **Descubrimiento de servicios.** Permite encontrar y ordenar los servicios que cumplen las metas especificadas en las peticiones de usuario. Este aspecto no se aborda con mayor profundidad, pues se encuentra fuera del alcance del presente trabajo.
- **Composición de servicios.** Los servicios deben organizarse y combinarse, de forma que el resultado obtenido, cumpla con los objetivos especificados por el usuario en la petición inicial. En ambientes dinámicos, donde ocurren diferentes fallas que impiden la ejecución adecuada de los servicios, es necesario que el proceso de composición utilizado, genere diferentes alternativas de reemplazo.
- **Adaptación.** Una vez se esté ejecutando un servicio compuesto, los cambios que pueden presentarse en un ambiente dinámico, requieren la aplicación de procesos de adaptación, los cuales deben usar las diferentes alternativas dadas por el proceso de composición. Al igual que en el caso anterior, este aspecto es tratado con mayor profundidad en las siguientes secciones.

Por lo tanto, la arquitectura general del sistema para la composición dinámica de servicios Web RESTful (Figura 11), considera tales aspectos, aunque es este trabajo de grado el bloque de descubrimiento se encuentra por fuera de sus alcances, pero se asume su participación contando con servicios ya descubiertos.

El sistema inicia con una petición de usuario, quien conoce técnicas y mecanismos necesarios para realizar una solicitud de composición que sea comprensible por el servidor, las cuales son enviadas en conjunto con los datos relevantes para la identificación del dispositivo y el contexto en el que se encuentra (Mediante un mecanismo de detección utilizando un agente de usuario y WURFL y la medida del nivel transferencia de datos en la red respectivamente), toda esta información se envía a través de un cliente navegador Web móvil, que es una herramienta muy común en dispositivos como, Smartphones, Tablets, consolas de videojuego inalámbricas, etc., aunque es posible acceder a través de un navegador Web del entorno estático, que cumpla con los requerimientos de la Tabla 16.

Cuando la petición formal del servicio se encuentra en el servidor, es preciso un bloque para el descubrimiento que procese dicha solicitud, con el fin de encontrar servicios Web RESTful simples; este bloque para el descubrimiento de servicios Web RESTful, no es desarrollado en este trabajo de grado, debido a que es un concepto robusto que exige esfuerzos por otro equipo de trabajo, por lo tanto como se menciono anteriormente los servicios simples ya identificados y organizados de acuerdo a su nivel de importancia; otro

rumbo que toma parte de la solicitud formal del servicio(agente de usuario del navegador Web móvil), es hacia el bloque de identificación de características de dispositivo, en el cual se procesa dicha información para encontrar las características no funcionales del dispositivo hardware que accede a la Web, en la base de datos suministrada por WURFL

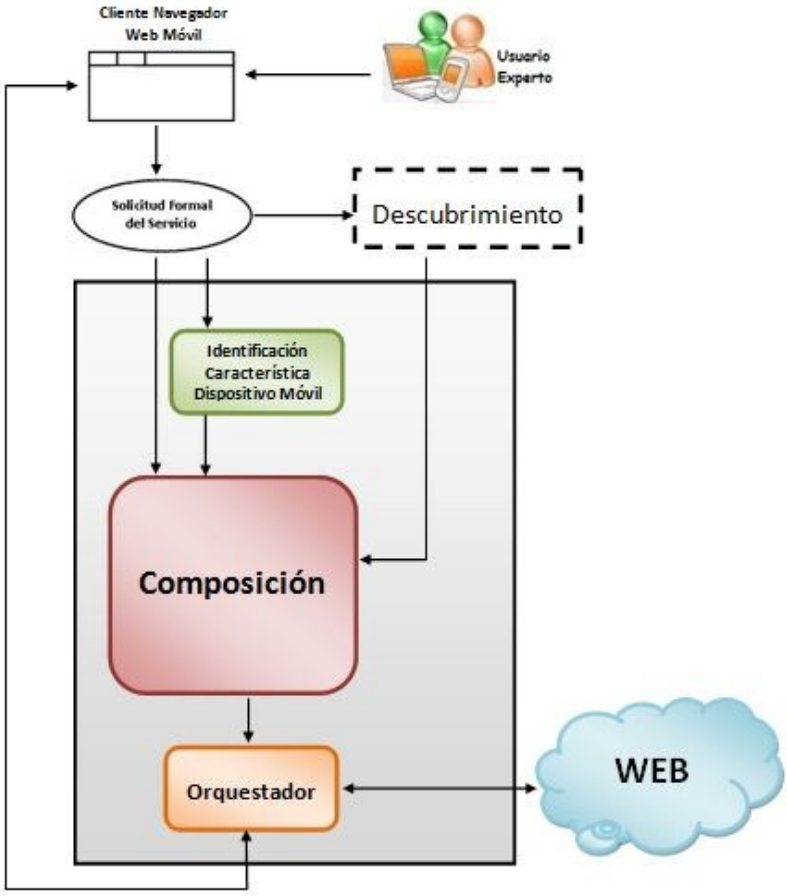


Figura 11. Arquitectura para la composición de servicios Web RESTful dinámica

Los servicios se clasifican en una matriz de enlace causal [50], interna al bloque de composición, con el fin de obtener de una manera más organizada un repositorio temporal, que se constituye en una elemento importante para ser utilizado por el bloque siguiendo los lineamientos de la solicitud formal del servicio, junto con las características halladas en la base de datos de WURFL.

El bloque de composición es el encargado de llevar a cabo la ejecución del algoritmo planteado, y crear las diversas posibilidades de composición basada en el requerimiento del usuario experto y las características del medio, además de escoger el mejor de ellos para ser entregado al dispositivo móvil. Los componentes de este bloque de manera detallada y su interacción con los bloques externos se observan en la Figura 12.

La composición STE (Sistema de Transición de Estados) es el primer módulo que realiza un procesamiento formal para la composición, siendo el bloque más representativo del sistema, ya que sus procesos se soportan en las características de los servicios basados en REST. Como se menciono en el Capítulo 3, los servicios Web RESTful poseen en su

interior toda una estructura conformada por diversos recursos que no tienen un comportamiento predefinido, sino hasta que se especifique la función que debe realizar, es por esto que la composición STE describe un camino interno para obtener la representación que coincide con los requerimientos dados por el usuario experto.

El Filtro de Formatos, toma las características del dispositivo suministradas por el bloque de identificación y compara los formatos de las representaciones de cada uno de los servicios Web RESTful, con el fin de descartar desde el comienzo, servicios que no cumplen con las capacidades soportadas por el dispositivo móvil. El bloque mas importarte para el sistema de composición, es el generador MEC (Matriz de Enlace Causal), ya que realiza los enlaces entre servicios, dada la solicitud del usuario experto, las características del dispositivo móvil y los servicios Web RESTful simples (ya definidos por la composición STE); estos enlaces se establecen mediante puntajes que son generados por el Analizador Sintáctico, que correlaciona los parámetros entre los servicios que componen la matriz. Tales bloques son descritos a profundidad en el Capitulo 4, donde se describe la adaptación del algoritmo.

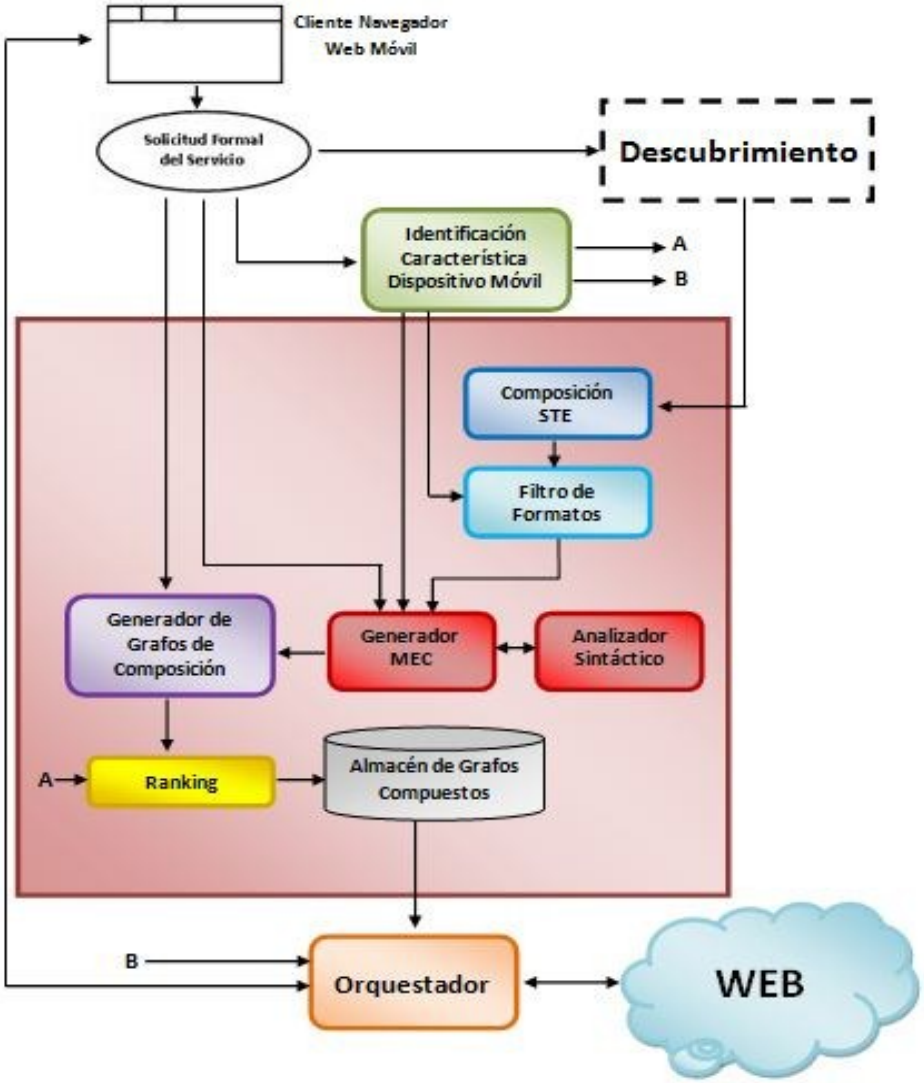


Figura 12. Bloque de Composición

El analizador sintáctico, presenta una clara ventaja con respecto a uno semántico, al igual que una falencia importante, la primera es la eficiencia que adquiere el sistema en tiempos de respuesta a una petición, y la segunda, es la baja calidad entre comparaciones, debido a que no se utilizan ontologías u otro tipo de mecanismo semántico, para establecer la relación entre servicios. Por lo tanto se establecen cuatro procesos importantes y secuenciales, que mitiguen la desventaja del uso de mecanismos sintácticos sin alterar considerablemente los tiempos de respuesta (Figura 13). El Tokenizador, se encarga de separar las palabras que componen el nombre de un parámetro (por ejemplo localizaciónLatitud), el bloque de diccionario de abreviaciones es una recopilación de abreviaciones conocidas el cual combate los problemas que surgen al comparar parámetros, cuyos nombres se componen por abreviaciones (por ejemplo lat, img), el siguiente mecanismo de N-grams compara dos nombres con el fin de encontrar similitud sintáctica entre grupos de N caracteres siendo en este caso N=2 y por ultimo un analizador con ciertas características semánticas ofrecido por la herramienta WordNet, que encuentra la correspondencia entre parámetros relacionados mas allá de la sintaxis. Estos procesos se convierten en una serie de filtros que no requieren ser ejecutados en su totalidad, ya que si en un bloque intermedio, por ejemplo N-grams o el diccionario de abreviaciones, arroja resultados muy positivos, no es necesario procesar el(los) siguiente(s) bloque(s).

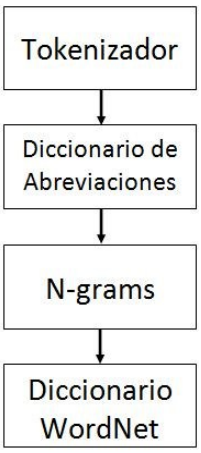


Figura 13. Proceso Secuencial del Analizador Sintáctico

Después de crear los enlaces, el generador de grafos de composición los utiliza para realizar los esquemas que cumplen con los requerimientos planteados en la solicitud formal del servicio. Los grafos resultantes son enviados al ranking para que sean ordenados jerárquicamente, siendo el primero el más apropiado para el usuario, teniendo en cuenta los siguientes criterios de clasificación:

1. Calidad de enlaces entre servicios.
2. Características acordes con el dispositivo.
3. Servicios preferentes por el usuario.
4. Popularidad de los servicios.

El almacén de grafos compuestos, alberga todos los grafos de forma ordenada generados anteriormente. Con el fin de acceder a ellos en tiempo de ejecución y ofrecer funciones dinámicas de manera más eficiente.

Hasta este punto se ha generado la composición automática de servicios Web RESTful, que aporta los procesos más importantes para realizar la composición dinámica en el bloque de cación, debido a que el concepto tomado para dicho fin, es mediante la aplicación de varias composiciones automáticas dependientes o no del usuario, entiempo de ejecución.

El orquestador es el bloque multifuncional por excelencia de la presente arquitectura; aunque no está compuesto por bloques internos, este es el encargado de ejecutar el servicio compuesto y capturar sus fallas, cada que se realice una petición. Las fallas contempladas para la reconfiguración del servicio compuesto se exponen a continuación en dos tipos:

Fallas tipo I

Este tipo de fallas son detectadas en el Orquestador y corresponden a errores en los servicios simples, los cuales son iguales a los códigos de estado en HTTP. El control de este tipo de fallas se realiza de tal forma que el usuario no detecte cambios considerables en tiempo real, por ejemplo, si hay un servicio que presenta un estado HTTP 500 desde su servidor de origen, el Orquestador selecciona del Almacén de grafos compuestos, uno que se asemeje al ejecutado, reemplazando el servicio anómalo.

El Orquestador mantiene la integridad del servicio compuesto, ya que considera un servicio fallido, siempre y cuando se compruebe que realmente existe, mediante la respuesta negativa a varias peticiones.

Fallas tipo II

Las fallas de este tipo, están relacionadas al comportamiento de la red, más específicamente a la capacidad que esta le brinda al dispositivo móvil de usuario. Estas fallas son detectadas cuando ocurren cambios en los niveles de velocidad de red en el entorno**.** Cuando se detectan medidas importantes, que signifiquen el cambio abrupto en los niveles de velocidad de transferencia de datos, e indiquen transiciones entre diversos tipos de tecnologías o niveles de señal dentro de una red, se controla mediante el cambio de los formatos de representación de los recursos que interactúan directamente con el usuario, como por ejemplo, cambiar de un formato de imagen png a gif cuando los niveles de red decaen (Tablas de clasificación de formatos en el Anexo).

Existe otro tipo de reconfiguración del servicio compuesto, que son generadas a causa de una petición directa del usuario experto, las cuales son: i) Adición de servicios simples: Cuando el usuario añade uno o más servicios al compuesto, ii) Eliminación de servicios simples: Se da cuando el usuario elimina uno o más servicios del compuesto, teniendo en cuenta que sea posible la reconfiguración de los servicios restantes y iii) Sustitución de servicios simples: el usuario reemplaza un servicio simple por otro que cumple con el mismo objetivo, vale la pena aclarar que la sustitución se debe dar uno a uno.

Cada uno de los tres eventos, utiliza la estructura detallada en este capítulo, es por eso que la mayor parte del trabajo se enfoca en la composición automática inicial, para brindar altos niveles de eficiencia y funcionalidad en las reconfiguraciones posibles de la parte dinámica.

Capítulo 5

Algoritmo para la Composición Dinámica de Servicios Web RESTful

El presente capítulo detalla los procesos y algoritmos que dan soporte a la arquitectura presentada anteriormente. Más específicamente, de aquellos módulos involucrados en la generación de servicios compuestos y en la adaptación de éstos una vez se encuentren en ejecución. Sin embargo, antes de describir dichos procesos, se presenta un modelo informal de los servicios Web RESTful, el cual establece un vocabulario común para los algoritmos presentados.

5.1. Modelo de los servicios Web RESTful

La Figura 14 muestra el modelo de los servicios Web RESTful, el cual se fundamenta en las definiciones y conceptos presentados en el Capítulo 3. Este modelo tiene como objetivo principal, eliminar cualquier tipo de dependencia que pueda existir entre un lenguaje de descripción particular y el algoritmo de composición. Lo cual se hace necesario, debido al soporte incompleto de los principios de REST en los lenguajes de descripción analizados y a la ausencia de un proceso de estandarización en este aspecto.

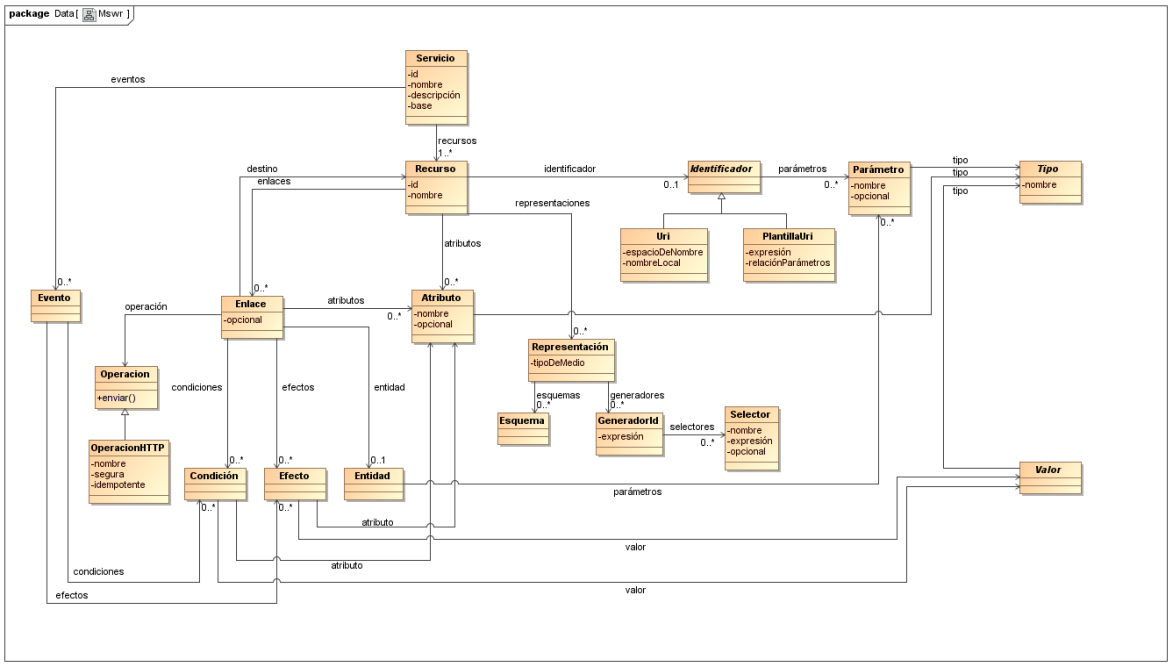


Figura 14. Modelo de los servicio Web RESTful[4]

En el modelo presentado, un servicio está formado por al menos un recurso y un enlace de entrada, además de los posibles eventos que puedan internamente cambiar el estado

[4] Las multiplicidades omitidas en el diagrama corresponden exactamente a 1.

de la aplicación. Dichos enlaces de entrada son definidos en este contexto, como aquellos identificadores de recurso (URI) que son conocidos por un cliente. Estos enlaces se suponen lo suficientemente estables, como para ser usados explícitamente por un cliente y no a través de representaciones.

Los eventos pueden modificar el estado de la aplicación sin la participación de un cliente RESTful y tienen una correspondencia directa con los cambios que puedan presentarse en un servicio, producto de la lógica interna del mismo. Este elemento permite modelar servicios complejos, en los que generalmente es necesario esperar un tiempo para que se termine un proceso y el cliente pueda continuar consumiendo el servicio.

Un recurso posee múltiples atributos y enlaces, los cuales son accedidos mediante representaciones. De acuerdo con este modelo, un recurso puede presentar diferentes atributos según la representación pedida. Un ejemplo de ello es el recurso de una fotografía, el cual en una representación incluye datos como el tamaño, la fecha de la fotografía, el formato, etc. y en otra, puede incluir únicamente la imagen de la fotografía.

Las representaciones son el medio utilizado por los clientes RESTful para seguir interactuando con un servicio, pues estas poseen diferentes enlaces que permiten modificar el estado actual de la aplicación. Un enlace está determinado por un recurso destino, un formulario o entidad de datos opcional y un conjunto de condiciones y efectos. El destino de un enlace es un recurso con un identificador opcional, lo cual permite modelar la transferencia de estado de un servidor a un cliente, de acuerdo con los principios de REST.

En algunas ocasiones, es necesario enviar información adicional para interactuar con un servicio, por lo que se requiere codificar en una representación con un formato particular. En el contexto de los servicios Web RESTful, la creación de la representación a partir de la información adicional que se desea enviar, es realizada mediante formularios codificados en diferentes tecnologías como HTML y XForms (Formularios basados en XML). En el modelo presentado, los formularios están asociados a enlaces y poseen diferentes parámetros que permiten producir entidades, los cuales corresponden a la información que puede ser introducida por un usuario o un servicio.

Por último, los enlaces poseen condiciones y efectos, los cuales imponen una restricción en los atributos de los recursos, la cual debe cumplirse antes de que el enlace pueda ser seguido o ejecutado. Estos dos elementos permiten, en los servicios donde sea necesario, modelar la máquina de estados interna dentro de un recurso.

En el caso de los enlaces basados en HTTP, las condiciones y efectos de un enlace, permiten modelar la semántica de los métodos de este protocolo. En este caso, es necesario determinar si un enlace es seguro o ídem-potente, para ello se aplican las siguientes reglas:

- Si el enlace no posee ningún efecto, entonces es considerado seguro.
- Si el enlace es seguro, entonces es también ídem-potente.
- En cualquier otro caso en el que se presenten efectos, el enlace debe ser marcado como ídem-potente o no ídem-potente, de acuerdo con la semántica del método del protocolo HTTP.

El enfoque establecido con el modelo presentado requiere la utilización de módulos de adaptación, encargados de transformar las descripciones de los servicios desde un lenguaje o tecnología externa a una representación del modelo utilizado, como se observa en la Figura 15.

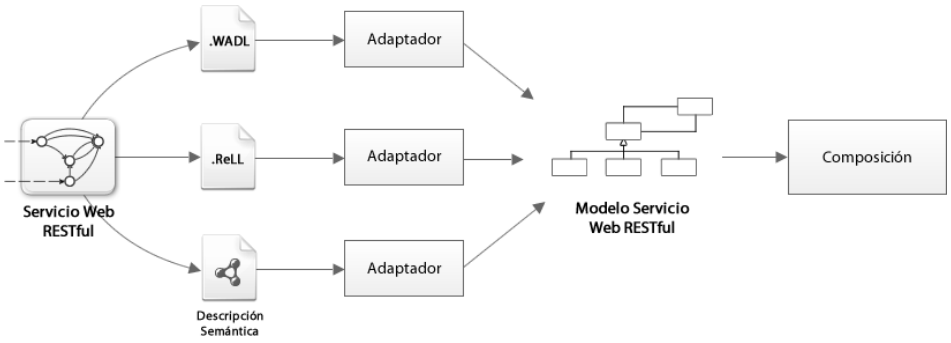


Figura 15. Transformación lenguajes de descripción

5.2. Composición de servicios

De acuerdo con el estudio de Fujii [54] existen tres tipos de sistemas o mecanismos para la composición dinámica de servicio Web tradicionales.

5.2.1. Composición basada en plantillas

La composición basada en plantillas permite la creación de un servicio, mediante la selección de servicios compuestos preestablecidos, conocidos como plantillas. Estos están formados por servicios abstractos que realizan una funcionalidad definida. Una vez se realice el proceso de composición, los servicios abstractos son remplazados por implementaciones reales, encontradas en el proceso previo de descubrimiento.

5.2.2. Composición basada en interfaces

La composición basada en interfaces toma las entradas y salidas del servicio compuesto de la petición del usuario, y en base a ellas, determina los servicios intermedios, que interconectados, permiten tomar las entradas establecidas y generar las salidas esperadas. La composición basada en interfaces es más flexible y adaptable que la basada en plantillas, pero presenta varios problemas, entre los cuales resalta, que no todos los servicios pueden ser modelados únicamente usando entradas y salidas.

5.2.3. Composición basada en lógica

Esta composición agrega expresiones en Lógica de Primer Orden (LPO) a la información dada por las interfaces, de forma que puedan modelarse una mayor cantidad de servicios. Este tipo de composición es más adaptable que las dos anteriores, pero es menos extensible, pues requiere que todos los proveedores de servicios manejen las mismas definiciones lógicas. La composición basada en lógica aumenta la complejidad de las descripciones de los servicios, pues requiere la utilización de una lógica formal.

El tipo de composición seleccionado es el basado en plantillas, pues el flujo de control del servicio es establecido por un usuario manejador del dominio de los servicios a utilizar

(usuario experto). Las alternativas restantes, requieren de procesamientos avanzados basados en lógica matemática o en el análisis de lenguaje natural, lo cual está fuera del alcance del presente trabajo. Teniendo en cuenta esto, en la siguiente sección se presenta el esquema general del algoritmo de composición de los servicios Web RESTful basado en plantillas.

5.3. Esquema general del algoritmo de composición

La Figura 16 presenta el esquema general del proceso de composición. En él, se observan las dos primeras etapas de la composición dinámica, tal como se describió anteriormente. El proceso inicia con el envío de la petición formal del servicio compuesto por parte de un usuario experto. Dicha petición contiene las metas esperadas en forma ordenada, donde se especifica el tipo de servicio, su posición y los enlaces que deben establecerse. El módulo de descubrimiento, ajeno al sistema de este trabajo de grado, toma la petición formal del usuario y encuentra los servicios que puedan cumplir con las metas finales.

En este punto inicia el proceso de composición, el cual se encuentra dividido en tres fases:

- En la primera, la secuencia de enlaces entre los recursos que debe ejecutarse para cumplir con la meta del servicio, es determinada, mediante un mecanismo basado en un Sistema de Transición de Estados (STE) o composición STE; proceso imprescindible para obtener la secuencia interna en un servicio Web RESTful y definir su función con respecto a la meta especificada.
- La segunda fase, realiza un proceso de establecimiento de enlaces entre los servicios. Dichos enlaces son codificados en una Matriz de Enlace Causal (MEC), la cual facilita la generación de los servicios compuestos y su elección para ejecución.
- En la última etapa, se consideran los procesos dinámicos del algoritmo, los cuales se encargan de la ejecución del servicio y de su mantenimiento mediante el control de fallas o mediante cambios generados por el usuario experto (reconfiguración del servicio).

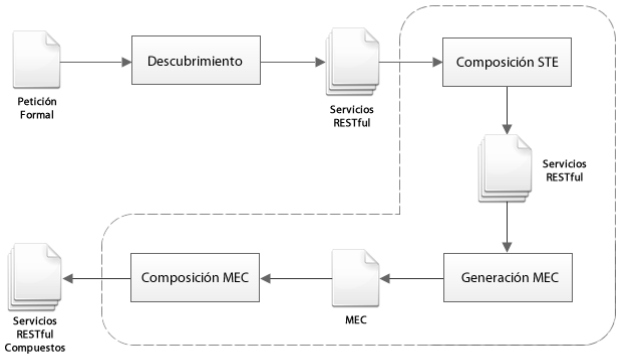


Figura 16. Esquema general del proceso de composición

5.4. Composición STE

A diferencia de los servicios Web tradicionales, que proveen funcionalidades concretas que pueden componerse directamente, los servicios Web RESTful están formados por un conjunto de recursos enlazados entre sí, que en determinados casos, pueden presentar comportamientos similares a los de una máquina de estado. Esta diferencia hace que no se puedan aplicar las técnicas de composición tradicionales, sin antes realizar un proceso previo, donde se seleccione la secuencia correcta en la que deben seguirse los enlaces del servicio. Dicha secuencia es guiada por la hipermedia y es influenciada dinámicamente por el estado de los recursos.

Para abordar el problema en el contexto REST, los servicios son modelados como un Sistema de Transición de Estados (STE).

5.4.1. Modelo STE para los servicios Web RESTful

Un STE es un modelo conceptual usado para describir sistemas dinámicos [55]. Formalmente definido como una 4-tupla $\Sigma = (S, A, E, \gamma)$, donde:

- $S = \{s_1, s_2, \ldots$, es un conjunto finito de estados;
- $A = \{a_1, a_2, \ldots$, es un conjunto finito de acciones;
- $E = \{e_1, e_2, \ldots$, es un conjunto finito de eventos y
- $\gamma : S \times A \times E \to 2^S$, es la función de transición de estados[5].

El Anexo se amplía las definiciones presentadas en esta sección.

Para modelar los servicios Web RESTful como sistemas STE y encontrar la secuencia adecuada de enlaces que cumplan con los requerimientos del cliente, son necesarios los componentes que se definen a continuación.

5.4.1.1. Conjunto de estados

El conjunto de estados S, está determinado por todos los posibles estados de la aplicación RESTful, los cuales dependen de la representación actual presente en el cliente y del estado interno de los recursos. Con el fin de evitar un número de estados infinitos, los recursos son agrupados en clases o tipos de recursos de acuerdo con el concepto que representen.

5.4.1.2. Conjunto de acciones

El conjunto de acciones A, está compuesto por cada uno de los enlaces que puedan aparecer en las representaciones de todos los recursos del servicio. Los enlaces que hacen parte de este conjunto, deben estar completamente especificados, es decir, deben indicar una URI o un mecanismo para obtenerla, además de los detalles del protocolo HTTP requeridos para su invocación. Este el caso, por ejemplo, de los enlaces de edición del formato ATOM, con los cuales es posible realizar tres tipos de acciones diferentes:

[5] 2^S es el conjunto potencia de S formado por todos los posibles subconjuntos de S.

obtención del recurso mediante una petición GET, actualización del recurso mediante una petición PUT y eliminación del recurso mediante una petición DELETE.

5.4.1.3. Conjunto de eventos

El conjunto de eventos E, está formado por los eventos internos al servicio RESTful que puedan afectar el estado de los recursos y por consiguiente los enlaces disponibles en un determinado estado. Los eventos corresponden a procesos ejecutados por la lógica interna de los servicios y pueden presentarse, al igual que las acciones en un grupo de estados particulares.

5.4.1.4. Función de transición

La función de transición γ, está determinada por las respuestas obtenidas al ejecutar los enlaces presentes en el recurso actual y por los eventos que puedan presentarse internamente en el servicio.

5.4.2. Características y restricciones del modelo STE de los servicios RESTful

Un modelo STE para los servicios RESTful, posee un conjunto de características que condicionan el tipo de algoritmo de planeación requerido. Estas características son descritas a continuación.

5.4.2.1. Finito

Un STE es finito si el conjunto de estados es finito. En el contexto actual, el STE es finito, pues la cantidad de tipos de recurso en un servicio RESTful, es limitada. Además se asume que la cantidad de estados internos en un recurso complejo es finita. Esto se realiza porque, los servicios RESTful basan el estado interno de sus recursos en valores restringidos de algunos de sus atributos. Por ejemplo, en un servicio de comercio electrónico en el que se ordenan productos, el estado interno de una orden está determinado por el valor de un atributo especial denominado “estado”, cuyo rango es la lista finita de valores: “no-pagado”, “preparando”, “lista”, “tomada” y “cancelada”.

5.4.2.2. Totalmente observable

Un STE es totalmente observable si es posible determinar el estado en el que se encuentra el sistema sin incertidumbre. En el contexto actual, el STE es totalmente observable, pues en cualquier instante es posible realizar peticiones GET, las cuales retornen una representación de un recurso. Dicha representación posee los atributos y enlaces que permiten determinar el estado actual del servicio.

5.4.2.3. Dinámico

Un STE es estático si el sistema permanece en el mismo estado a menos que se aplique una acción, lo cual implica que el conjunto de eventos es vacío. En el contexto actual, el STE es dinámico, pues la lógica interna de los servicios puede ocasionar cambios en el estado interno de los recursos.

5.4.2.4. No determinista

Un STE es no determinista si la aplicación de una acción puede llevar el sistema a más de un estado diferente. En el contexto actual, esta característica se deriva del hecho de que el STE es dinámico.

Las características descritas requieren algunas propiedades en los algoritmos de planeación. Dichas propiedades son:

- **Planes no secuenciales.** Debido al no determinismo, los planes producidos por un algoritmo, deben incluir instrucciones complejas como condicionales y ciclos, los cuales permitan a un controlador llevar el sistema al estado esperado.
- **Planeación en línea.** La planeación en línea requiere la observación del sistema modelado con el fin de determinar el estado actual y posteriormente actuar en concordancia. Esta característica es necesaria, pues la ejecución de planes no secuenciales, requiere del conocimiento del estado como se verá más adelante.
- **Metas restringidas.** Las metas u objetivos de un problema de planeación se consideran restringidas, en cuanto a que sólo consideran el estado final o estados finales a los que se desea llegar, pero no se tienen en cuenta otras directivas que restringen los estados intermedios que son incluidos en los planes resultantes.

5.4.3. Representación de un STE

El tamaño de un STE depende del nivel de abstracción utilizado en el proceso de modelado, pudiendo producirse STE reducidos, con pocos estados, acciones y eventos, que pueden manejarse fácilmente, pero que poseen poco valor práctico. Pero también, pueden producirse STE de gran tamaño, con una cantidad enorme de estados, acciones y eventos, que serian incontrolables.

Con el fin de no especificar completamente el STE de un sistema y evitar la lista de un gran número de estados, se utilizan diferentes representaciones, caracterizadas por notaciones genéricas que pueden codificar un STE de forma más compacta. Las representaciones básicas usadas en planeación clásica son:

- **Representación en teoría de conjuntos:** En esta representación los estados son codificados mediante conjuntos de proposiciones, que se consideran ciertas en una situación específica del sistema. Las acciones (o eventos) usan expresiones en las que se define que proposiciones deben pertenecer a un estado para ser aplicables y que proposiciones deben agregarse o eliminarse para generar un nuevo estado.
- **Representación clásica:** La representación clásica es similar a la de teoría de conjuntos, pero utiliza expresiones en lógica de primer orden más expresivas que las usadas con proposiciones.
- **Representación en variables de estado:** En esta representación los estados son codificados mediante un conjunto de variables de estado con un valor específico. Las acciones son expresiones que evalúan la igualdad de una variable de estado con un valor dado para ser aplicables y asignan nuevos valores a las variables para generar un nuevo estado.

De acuerdo con [55], la representación en variables de estado es la más adecuada cuando los estados son un conjunto de atributos, cuyo dominio de variación es finito o puede reducirse a un número finito de valores. Por esta razón se utilizará como base para representar los servicios RESTful, pues su estado está determinado por el valor de sus atributos. En el Anexo C se presenta una descripción general de este tipo de representaciones.

5.4.4. Modelo STE de un servicio Web RESTful usando la representación en variables de estado

5.4.4.1. Términos

Se define la clase de los recursos ($recurso$), la cual agrupa los símbolos de constantes que identifican los tipos de recurso de un servicio RESTful. Estos se obtienen, agrupando los recursos que posean atributos similares, como por ejemplo, las ordenes en un servicio de comercio electrónico, los mensajes en un servicio de redes sociales, o los lugares en un servicio de detalles geográficos.

En la definición de otros elementos de un STE, por ejemplo, los operadores, es necesario indicar el tipo de una variable por medio de una relación unaria, cuyo identificador es el nombre de una clase. En el caso de la clase de los recursos, por ejemplo, la expresión $recurso(r)$ indica que la variable r tiene rango en la clase de los recursos.

Además de las clases ya definidas, por cada atributo de un recurso que pueda aparecer en las expresiones dadas en las precondiciones o efectos de un operador, se define una clase a la cual pertenecen las constantes definidas por los valores de dichos atributos. El nombre de dicha clase está dado por el símbolo del tipo de recurso seguido por el nombre del atributo.

5.4.4.2. Variables de estado

Se define la función de variable de estado:

$$recursoActual : S \rightarrow recurso ,$$

La cual indica el recurso actual en el cliente de una aplicación RESTful.

Además, se define una variable de estado por cada atributo que aparezca en las expresiones de precondiciones y efectos de los operadores. La función de variable de estado en este caso, tiene la forma:

$$\langle \text{recurso} \rangle . \langle \text{atributo} \rangle : S \rightarrow \{\text{valores del atributo}\} ,$$

Donde $\langle \text{recurso} \rangle$ es el símbolo de constante del recurso al que pertenece el atributo, $\langle \text{atributo} \rangle$ es el nombre del atributo y el rango de la función es un conjunto finito de los posibles valores de dicho atributo.

5.4.4.3. Relaciones rígidas

Se define la relación $enlazado$ como:

$$enlazado \subseteq recurso \times recurso ,$$

La cual establece que existe un enlace dirigido desde el recurso inicial al recurso final.

Por cada enlace existente entre un par de recursos, debe definirse una relación rígida a partir de la relación establecida anteriormente.

5.4.4.4. Operadores, acciones y eventos

Se define el operador básico de transferencia como:

```
Operador de transferencia
1.  acción transferir(f, d)
2.      parámetros:     recurso(f), recurso(d)
3.      precondición:   recursoActual = f, enlazado(f, d)
4.      efecto:         recursoActual ← d
```

El cual indica la transferencia de estado ocurrida en el cliente RESTful al ejecutar un enlace. En la lista anterior:

- La línea 1 define un operador de nombre $transferir$ y lista de parámetros f y d. El operador definido pertenece al conjunto O_A, de acuerdo con la definición dada previamente. En el caso de un operador perteneciente a O_E, se usaría la palabra $evento$ en lugar de $acción$.
- La línea 2 indica que el tipo de las variables f y d, es la clase $recurso$.
- La línea 3 indica que la variable de estado $recursoActual$ debe ser igual al valor de la variable f y que f debe estar $enlazado$ a d, para que sea posible ejecutar el operador.
- La línea 4 indica que el efecto de ejecutar el operador es la actualización de la variable de estado $recursoActual$ al valor de la variable d.

Este operador corresponde al método GET del protocolo HTTP, el cual no tiene otro efecto, sino el de transferir el estado de un recurso, desde el servidor de origen al cliente RESTful. En teoría debería definirse un operador por cada uno de los métodos restantes del protocolo HTTP, sin embargo, aunque dichos métodos poseen una semántica definida, sus precondiciones y efectos están condicionados por las situaciones específicas de los servicios en los que son utilizados. Por ejemplo, aunque el método DELETE del protocolo HTTP esté definido para eliminar un recurso, en la práctica, este método puede usarse con una semántica distinta, como cancelar o terminar una operación en curso. Por lo cual se requiere definir, por cada servicio, acciones que no sigan la semántica del método GET del protocolo HTTP.

Además de lo presentado, se deben definir operadores específicos para la generación de eventos, según lo requiera la lógica interna del servicio.

5.4.4.5. Lenguaje, dominio y problemas de planeación

El lenguaje de planeación usado para modelar los servicios Web RESTful se define como $\mathcal{L}_{\mathcal{R}} = (D, R, X)$, donde:

- D, es la unión del conjunto de los tipos de recurso y el conjunto de los valores de los atributos de cualquier recurso que aparezca en una variable de estado.
- R, es la unión del conjunto de las relaciones rígidas, derivadas de la relación $enlazado$ y el conjunto de las relaciones rígidas particulares a cada servicio.
- X, es la unión del conjunto de todas las variables de estado substituidas derivadas de la función de variable de estado $recursoActual$ y el conjunto de todas las variables de estado substituidas derivadas de los atributos de los recursos que aparecen en las precondiciones o efectos de un operador, acción o evento.

Un Servicio Web RESTful es definido entonces, como un dominio de planeación $\Sigma_{\mathcal{R}} = (S, S_0, A, E, \gamma)$, en el lenguaje $\mathcal{L}_R$, donde:

- $S \subseteq \prod_{x \in X} D_x$, donde D_x es el rango de la variable de estado x. Un estado tiene la forma $s = \{(x = c) \mid x \in X\}$, donde $c \in D_x$.
- S_0, es el conjunto de los estados iniciales del servicio RESTful, formado por los estados resultantes de aplicar las acciones que no requieran ningún tipo de precondición. Dichas acciones son conocidas como acciones de entrada.
- A, es el conjunto de todas las acciones derivadas de los enlaces definidos en el Servicio Web RESTful, las cuales tienen como base los operadores en el conjunto O_A.
- E, es el conjunto de todos los eventos obtenidos como substituciones en los operadores pertenecientes al conjunto O_E.
- La función de transición γ, es definida de la misma forma que en la Sección 5.3.1.

Un problema de planeación en el contexto de un servicio RESTful, está dado por $\mathcal{P}_{\mathcal{R}} = (\Sigma_{\mathcal{R}}, S_0, S_g)$, donde:

- $\Sigma_{\mathcal{R}}$, es el dominio de planeación de un Servicio Web RESTful;
- S_0, es el conjunto de estados iniciales de un Servicio Web RESTful;
- S_g, es el conjunto de estados finales.

El conjunto de estados finales puede especificarse directamente o pude calcularse a partir de una expresión. En el último de los casos, la expresión utilizada debe contener los valores esperados de una o más variables de estado, además de requerimientos de información adicional. Estos requerimientos de información permiten determinar el recurso o los recursos en los que debe terminar la ejecución de un plan.

5.4.4.6. Sentencia de un problema de planeación

La sentencia de un problema de planeación en el dominio de un servicio RESTful, se define como $P_{\mathcal{R}} = \left(A, E, S_0, S_g\right)$, la cual incluye el conjunto de acciones y eventos del servicio RESTful.

5.4.5. Algoritmos de planeación

Debido a que el modelo realizado para los servicios Web RESTful es no determinista, es necesario encontrar planes no secuenciales, que permitan llevar al sistema a un estado final deseado y resolver un problema de planeación. Dichos planes no secuenciales son denominados políticas y se caracterizan por codificar comportamientos no lineales, como condicionales e iteraciones.

Una política π para un servicio Web RESTful $\Sigma_{\mathcal{R}} = \left(S, S_0, A, E, \gamma\right)$, es un conjunto de tripletas $\left(s, a, e\right)$, tal que $s \in S$, $a \in A\left(s\right)$ y $e \in E\left(s\right)$, donde $A\left(s\right)$ es el conjunto de acciones aplicables en s y $E\left(s\right)$ es el conjunto de eventos que pueden ocurrir en s. El conjunto de estados de una política es $S_\pi = \left\{s \mid \left(s, a, e\right) \in \pi\right\}$. Una política le indica a un cliente que enlace debe seleccionar de una representación de un recurso particular, de acuerdo con el estado en el que se encuentre el servicio y el objetivo final de un usuario.

Debido al no determinismo del modelo de los servicios Web RESTful, existen diferentes estrategias para solucionar un problema de planeación, las cuales producen diversos tipos de políticas. De acuerdo con [55], pueden producirse tres tipos de soluciones, cuya definición se presenta a continuación:

- **Soluciones débiles.** Las soluciones débiles se caracterizan por no garantizar la llegada del sistema a los estados finales. Las soluciones débiles buscan siempre el camino más corto. Esta solución sólo es confiable en aquellos casos en los que el servicio RESTful no presente eventos internos que cambien el estado de la aplicación.
- **Soluciones fuertes.** Las soluciones fuertes minimizan las desventajas de las débiles y se caracterizan por ser seguras, ya que proporcionan políticas que consideran todos los posibles estados en los que pueda terminar un cliente. La desventaja en este caso, es que en determinados casos, en los que sí existen los otros tipos de soluciones, no es posible obtenerlas. Lo cual se debe, a que las soluciones fuertes no tienen en cuenta los ciclos infinitos[6] que puedan presentarse en un servicio.
- **Soluciones cíclicas-fuertes.** Las soluciones c-fuertes ofrecen un punto intermedio entre las soluciones fuertes y débiles, y se caracterizan por asumir que el evento se producirá en algún periodo de tiempo futuro, en el caso de que puedan presentarse ciclos infinitos en el servicio.

La Figura 17 muestra la relación jerárquica entre los diferentes tipos de soluciones en donde, las soluciones fuertes son un subconjunto de las soluciones cíclicas-fuertes y

[6] Un ciclo infinito en un servicio se da cuando la transición a un próximo estado sólo puede darse en la presencia de un evento en la lógica interna.

éstas a su vez, lo son de las soluciones débiles. Esta relación permite seleccionar el algoritmo de planeación más adecuado, de acuerdo con el servicio RESTful del problema de planeación. La siguiente lista muestra las reglas usadas para dicha selección.

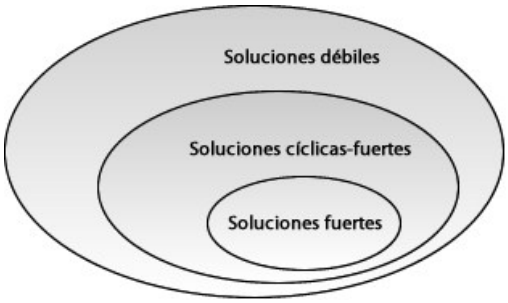


Figura 17. Relación jerárquica de los diferentes tipos de soluciones a un problema de planeación de un Servicio Web RESTful

Reglas para seleccionar algoritmo de planeación	
1.	**si** el servicio Web RESTful no presenta ningún evento **entonces**
2.	usar un algoritmo que produzca soluciones débiles
3.	**Sino**
4.	**si** existen soluciones fuertes **entonces**
5.	usar un algoritmo que produzca soluciones fuertes
6.	**sino**
7.	usar un algoritmo que produzca soluciones cíclicas-fuertes
8.	**fin si**
9.	**fin si**

Las reglas presentadas expresan una preferencia por las soluciones débiles en el caso de que no existan eventos, pues en este contexto son suficientes. Pero en el caso de que puedan presentarse eventos, las soluciones débiles no son adecuadas, por lo que se prefieren las soluciones fuertes, si estas llegasen a existir, y las soluciones cíclicas-fuertes en cualquier otro caso. Se prefieren las soluciones fuertes a las cíclicas-fuertes, pues éstas evitan los posibles ciclos infinitos que puedan presentarse en un servicio, en los cuales es necesario esperar la ocurrencia de un evento en la lógica interna del servicio.

En el Anexo C se detalla a profundidad el marco teórico del modulo STE generado para este trabajo de grado, además de la especificación de los algoritmos usados para producir los diferentes tipos de soluciones presentados en ésta sección y un ejemplo detallado para comprender su comportamiento en situaciones practicas.

5.5. Generación y Composición MEC

Una vez la secuencia de recursos y enlaces internos en un servicio Web RESTful ha sido establecida, se inicia el proceso de composición entre servicios. Para ello, se calcula una Matriz de Enlaces Causales (MEC), la cual codifica los diferentes enlaces que pueden establecerse entre los servicios. Esta matriz se convierte en el punto de partida de los diferentes algoritmos usados en los procesos de composición y adaptación.

5.5.1. Enlace causal

Un enlace causal es una relación de tipo secuencial establecida entre dos servicios Web RESTful, a la cual se le asocia una calificación numérica usando una función de similitud específica. Complementariamente, un enlace causal establece el flujo de datos entre los servicios asociados, permitiendo su ejecución en secuencia. Formalmente, se define como una tripleta:

$$\left\langle s_x, Sim\left(Atr\left(s_x\right), Param\left(s_y\right)\right), s_y \right\rangle,$$

donde s_x y s_y son dos servicios Web RESTful, $Atr\left(s_x\right)$ es el conjunto de atributos del servicio s_x, $Param\left(s_y\right)$ es el conjunto de parámetros del servicio s_y y Sim es la función de similitud entre servicios.

5.5.1.1. Función de similitud entre servicios

La función de similitud entre servicios mide el grado de similitud entre los parámetros de un servicio destino y los atributos de un servicio de origen, teniendo en cuenta la secuencia especificada en la petición inicial del usuario. Esta función facilita el establecimiento del flujo de datos del servicio compuesto y por tanto su posterior ejecución.

La función de similitud se define como:

$$Sim\left(Atr\left(s_x\right), Param\left(s_y\right)\right) = \frac{\sum_{param\_s_y \in Param\left(s_y\right)} \max\left(Sim_{AP}\left(atr1\_s_x, param\_s_y\right), ..., Sim_{AP}\left(atrN\_s_x, param\_s_y\right)\right)}{\left|Param\left(s_y\right)\right|}$$

donde $atr1\_s_x, ..., atrN\_s_x \in Atr\left(s_x\right)$ y Sim_{AP} es la función de similitud entre atributos y parámetros.

5.5.1.2. Función de similitud entre atributos y parámetros

Esta función de similitud tiene en cuenta tanto el nombre del parámetro o atributo del servicio, como también su tipo de dato (numérico, cadena de texto, etc.). El resultado se basa en el procesamiento de tipo sintáctico, relacionado con la estructura de las palabras, y de tipo semántico, relacionado con los significados tomados de un diccionario común.

La comparación entre los nombres o etiquetas, asociadas a los datos de un servicio, se inicia con un proceso de subdivisión, el cual reduce un nombre compuesto en un conjunto de palabras simples. Posteriormente, se realiza una comparación individual entre dichas palabras simples. Esta comparación está dada por la siguiente función:

$$Sim_N\left(P, Q\right) = \frac{\left|P\right| + \left|Q\right| - Dif\left(P, Q\right)}{\left|P\right| + \left|Q\right|}$$

donde $P = tokenize(nombre_1)$ es el conjunto de palabras que componen el $nombre_1$, $Q = tokenize(nombre_2)$ es el conjunto de palabras que componen el $nombre_2$. La función $tokenize$ subdivide un nombre compuesto en palabras simples reconocibles, tomando como referencia aquellos caracteres que no pueden pertenecer a una palabra (por ejemplo: -,_,&,etc.) y los diferentes cambios que puedan presentarse entre el tipo de letra, minúscula o mayúscula, entre caracteres sucesivos. Por ejemplo: el nombre *imageSize* es subdivido en las palabras *image* y *size*.

La función Dif está dada por:

$$Dif(P,Q) = \sum_{t \in (P \cup Q)} \left| \max\left(Sim_W(P,t)\right) - \max\left(Sim_W(Q,t)\right) \right|$$

donde:

$$\max\left(Sim_W(P,t)\right) = \max\left(Sim_W(p_1,t),...,Sim_W(p_i,t)\right), \text{ y}$$

$$\max\left(Sim_W(Q,t)\right) = \max\left(Sim_W(q_1,t),...,Sim_W(q_i,t)\right)$$

con $p_i \in P$ y $q_i \in Q$.

La función Sim_W es la función de similitud entre palabras definida según la siguiente expresión:

$$Sim_W(w_1,w_2) = \begin{cases} 1, \text{ if } m_1 = 1 \vee m_2 = 1 \\ m_1, \text{ if } 0.75 \le \mathrm{m}_1 < 1 \\ m_3, \text{ en cualquier otro caso} \end{cases}$$

donde $m_1 = Sim_{Lex}(w_1,w_2)$ es la similitud dada por un algoritmo de comparación sintáctica como NGrams, $m_2 = Sim_{Abbr}(w_1,w_2)$ es la similitud dada en base a un diccionario de abreviaciones y $m_3 = Sim_{LexSem}(w_1,w_2)$ es la similitud dada por una base de datos de relaciones semánticas como WordNet.

Por otro lado la función para comparar los tipos de datos requiere del establecimiento de una estructura jerárquica, en la que los elementos superiores corresponden a tipos de datos generales y a medida que se desciende se observan tipos de datos más específicos que representan un rango de valores más restringido. La Figura 18 muestra dicha estructura.

La función de similitud para los tipos de datos, $Sim_{TD}(t_1,t_2)$, está basada en una función de similitud semántica descrita en [56] para relacionar conceptos que se encuentren en una misma ontología. La Tabla 17 muestra la definición de esta función. En ésta función, la expresión $\text{distancia}(t_1,t_2)$ calcula el número de aristas o enlaces entre los dos tipos de datos y es utilizada para discriminar entre dos comparaciones con un mismo valor.

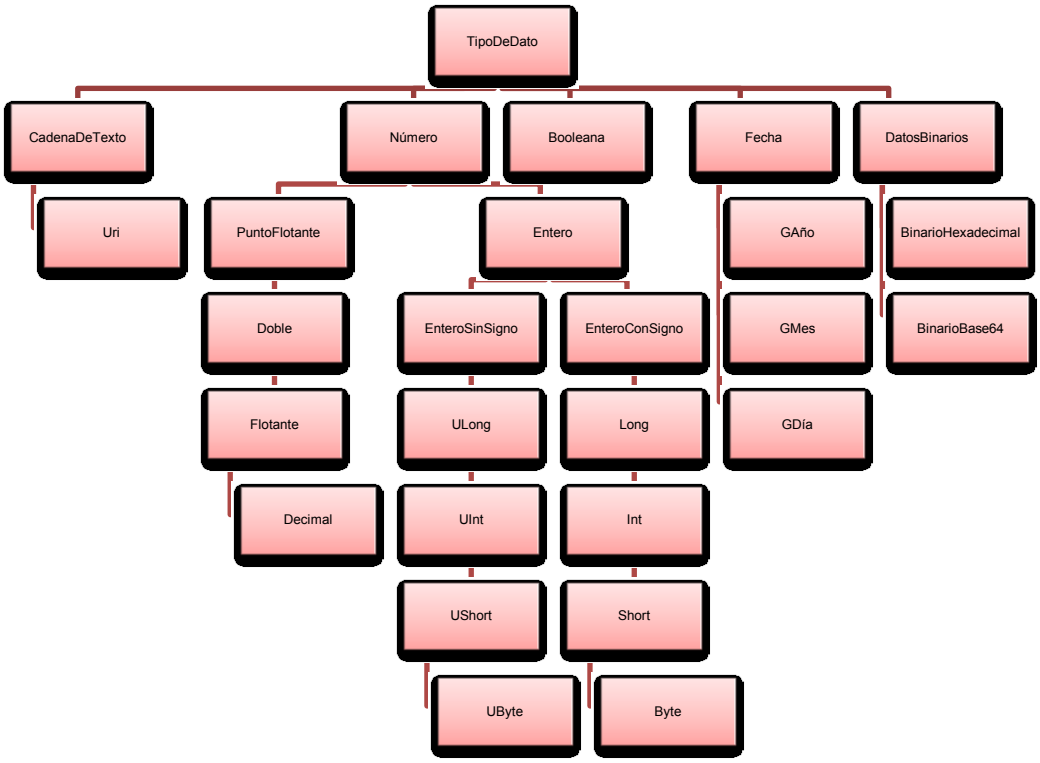


Figura 18. Jerarquía tipos de datos

Similitud	Significado	$Sim_{TD}(t_1,t_2)$
Exacto($\equiv$)	t_1 y t_2 corresponden al mismo tipo de dato en la estructura o tienen el mismo nombre.	$(1,0)$
Parte($\subset$)	t_1 y t_2 pertenecen a la misma rama, pero t_1 se encuentra en un nivel inferior al de t_2.	$\left(\frac{3}{4},\text{distancia}(t_1,t_2)\right)$
Subsunción($\supset$)	t_1 y t_2 pertenecen a la misma rama, pero t_1 se encuentra en un nivel superior al de t_2.	$\left(\frac{2}{4},\text{distancia}(t_1,t_2)\right)$
Intersección($\cap$)	t_1 y t_2 no pertenecen a la misma rama, pero comparten el tipo de dato raíz. Por ejemplo: Punto Flotante y Entero.	$\left(\frac{1}{4},\text{distancia}(t_1,t_2)\right)$
Falla($\varnothing$)	t_1 y t_2 pertenecen a distintos grupos de tipo de dato. Por ejemplo: Booleano y Cadena de Texto.	$(0,\infty)$

Tabla 17. Función de similitud para tipos de datos

Por último la similitud conjunta, Sim_{AP}, de un atributo y un parámetro está dada por:

$$Sim_{AP}\left(Atr\_s_x, Param\_s_y\right)=\begin{cases}0, \text{ si } Sim_{TD}<1/4\\ 0.8Sim_N+0.2Sim_{TD}, \text{ en cualquier otro caso}\end{cases}$$

donde

$$Sim_{TD}=Sim_{TD}\left(Tipo\left(Atr\_s_x\right),Tipo\left(Param\_s_y\right)\right),$$

$$Sim_N = Sim_N\left(tokenize\left(Nombre\left(Atr\_s_x\right)\right), tokenize\left(Nombre\left(Param\_s_y\right)\right)\right).$$

y $Tipo(\ )$ es el tipo de dato del atributo o parámetro y $Nombre(\ )$ es nombre del atributo o parámetro.

5.5.1.3. Enlace causal válido

Un enlace causal $\left\langle s_x, Sim\left(Atr\left(s_x\right), Param\left(s_y\right)\right), s_y\right\rangle$ es válido, si y sólo si la función de similitud $Sim\left(Atr\left(s_x\right), Param\left(s_y\right)\right)$ es mayor que cero.

5.5.2. Matriz de Enlace Causal

Una Matriz de Enlace Causal (MEC) está definida por una matriz $M_{p,q}$, cuyos elementos $m_{i,j}$ son enlaces causales:

$$\left\langle s_i, Sim\left(Atr\left(s_i\right), Param\left(s_j\right)\right), s_j\right\rangle$$

En la anterior definición:

Las metas de s_i y s_j se encuentran enlazadas en la petición de usuario, ya sea directamente o a través de la conexión de otros servicios. La matriz almacena todas las posibles relaciones que puedan establecerse entre los servicios RESTful siguiendo la secuencia dada por el usuario.

Ejemplo. Dada una petición de usuario:

```
Metas:            [geocoding, music events, maps, notifications]
Enlaces:          [(geocoding, music events), (music events, maps), (music
                  events, notifications)]
```

Donde los enlaces se organizan en parejas, siendo el primer elemento el origen y el segundo el destino de un enlace. Asumiendo los siguientes servicios por cada meta:

```
Geocoding:        Google Geocoding, MapQuest Geocoding, Sensor GPS
Music Events:     Lastfm Events, Songkick Events
Maps:             Google Maps, Bing Maps
Notifications:    Twitter Search
```

La MEC calculada es la siguiente:

	Destino	Geo.			Events		Maps		Not.
Origen		**Google**	**MapQ**	**GPS**	**Lfm**	**Snk**	**Google**	**Bing**	**Twitter**
Geo.	**Google**				0.470	1.000	0.355	0.401	0.220
	MapQ				0.465	0.986	0.353	0.398	0.220
	GPS				0.489	1.000	0.365	0.414	0.226
Events	**Lfm**						0.368	0.426	0.394
	Snk						0.395	0.465	0.417
Maps	**Google**								
	Bing								
Not.	**Twitter**								

Tabla 18. Ejemplo MEC

5.5.3. Algoritmo de composición

Al igual que el proceso de composición realizado internamente en cada uno de los Servicios Web RESTful ya presentado, el algoritmo para la composición de los servicios, está basado en técnicas de planeación soportadas en modelos de Sistemas de Transición de Estados.

En el contexto actual un problema de planeación P, está definido por la tripleta

$$P = (SWR, \mathcal{KB}, \beta),$$

donde SWR, el conjunto de los servicios RESTful, es el conjunto de acciones de un STE; $\mathcal{KB}$, el conjunto de servicios iniciales de la petición, es una base de conocimiento cuyos parámetros serán ingresados por el usuario; y β, el conjunto de servicios finales de la petición, son el conjunto de metas.

5.5.4. Planes

Los planes producidos por el algoritmo de composición, representan un servicio Web RESTful compuesto, cuyos recursos corresponden a los recursos de los servicios individuales y los enlaces están dados entre grupos de recursos.

De forma más precisa, un plan es una expresión en los servicios RESTful pertenecientes al conjunto SWR usando los siguientes operadores binarios:

- **Operador de enlace** ($\leftarrow$). La expresión $s_y \leftarrow s_x$, indica que el servicio s_x debe ser ejecutado antes que el servicio s_y. Es decir, una de las representaciones del servicio s_x contiene un enlace al servicio s_y, para lo cual, debe existir uno o más enlaces causales $\langle s_i, Sim(Atr(s_i), Param(s_j)), s_j \rangle$ codificado en la MEC. El operador de enlace es similar al operador de secuencia usado en la composición de Servicios Web tradicionales.
- **Operador de conjunción** ($\wedge$). La expresión $s_x \wedge s_y$, indica que tanto las representaciones de s_x como las de s_y son requeridas simultáneamente para

cumplir con la meta del usuario. La expresión $s_z \leftarrow (s_x \wedge s_y)$, indica que en las representaciones conjuntas de s_x y s_y existe un enlace hacia el servicio s_z.

- **Operador de disyunción** ($\vee$). La expresión $s_x \vee s_y$, indica que las representaciones de uno solo de los servicios es requerida para cumplir con la meta del usuario. La expresión $s_z \leftarrow (s_x \vee s_y)$, indica que existe un enlace entre s_x y s_z, pero también existe un enlace, igualmente válido, entre s_y y s_z. Este operador divide un único plan, en diferentes planes disjuntos, de los cuales debe seleccionarse uno para la ejecución final del servicio compuesto. Dicho proceso de selección es presentado más adelante.

La precedencia de los operadores, de mayor a menor, está dada por $(\wedge) > (\vee) > (\leftarrow)$.

Los controles de flujo de la composición se limitan a los operadores descritos anteriormente, los cuales se relacionan directamente con la estructura de las peticiones de usuario.

5.5.5. Técnica de planeación

La técnica de planeación usada está basada en un algoritmo de búsqueda en profundidad en el espacio de estados, hacia atrás y recursiva. El algoritmo empieza con una de las metas o atributos requeridos por el usuario para el servicio compuesto. Posteriormente, mediante el uso de la MEC, encuentra cuales son los servicios que pueden enlazarse con dicho servicio y los agrega al plan. A continuación, los servicios encontrados, son establecidos como las metas del servicio compuesto. El proceso es repetido desde el inicio, hasta que las metas requeridas se encuentren en la base de conocimiento o servicios iniciales.

Con el fin de facilitar la definición del algoritmo de composición entre servicios, se introduce un servicio abstracto al final de la petición de usuario enlazado a los servicios finales reales. Este servicio no es ejecutable y su único propósito es el de facilitar la definición del algoritmo.

El algoritmo de planeación se presenta a continuación. En dicho algoritmo R es la petición de usuario.

Algoritmo general de planeación MEC

1. $\text{planeacion}(R, SWR, \mathcal{KB}, \beta)$
2. $\quad \pi \leftarrow \text{planeacionRecursiva}(R, SWR, \mathcal{KB}, \beta, \varnothing)$
3. $\quad \Pi \leftarrow formaDisjunctiva(\pi)$
4. **fin**

Algoritmo de planeación recursiva MEC

1. $\text{planeaciónRecursiva}(R, SWR, \mathcal{KB}, \beta, \beta_s)$
2. $\quad \pi \leftarrow \beta$
3. **si** $\beta \in \mathcal{KB}$ **entonces**
4. **retornar** π

5. **fin si**
6. $S_c \leftarrow \varnothing$
7. **para cada** $s_i \in \text{servicios}(R) \mid (s_i, \text{meta}(\beta)) \in \text{enlaces}(R)$ **hacer**
8. $S_{ci} \leftarrow \text{fuentesValidas}(s_i, \beta, M_{p,q})$
9. $\text{adicionar}(S_{ci}, S_c)$
10. **fin para**
11. $\pi' \leftarrow \varnothing$
12. **para cada** $S \in S_{c1} \times ... \times S_{cN}, S_{ci} \in S_c$ **hacer**
13. $\pi'' \leftarrow \varnothing$
14. **para cada** $s \in S$ **hacer**
15. **si** $s \in \beta_s$ **entonces**
16. $\pi'' \leftarrow \pi'' \wedge \varnothing$
17. **sino**
18. $\text{adicionar}(\beta, \beta_s)$
19. $\pi'' \leftarrow \pi'' \wedge \text{planeaciónRecursiva}(R, SWR, \mathcal{KB}, s, \beta_s)$
20. $\text{remover}(\beta, \beta_s)$
21. **fin si**
22. **fin para**
23. $\pi' \leftarrow \pi' \vee \pi''$
24. **fin para**
25. $\pi \leftarrow (\pi \leftarrow \pi')$
26. **retornar** π
27. **fin**

El algoritmo de planeación se basa en las siguientes rutinas:

- $\text{adicionar}(e, E)$. Adiciona el elemento e a la lista o conjunto E.
- $\text{eliminar}(e, E)$. Elimina el elemento e de la lista o conjunto E.
- $\text{servicios}(R)$. El conjunto de servicios de la petición usuario R.
- $\text{enlaces}(R)$. El conjunto de enlaces de la petición de usuario R.
- $\text{meta}(\beta)$. La meta del servicio β.
- $\text{fuentesValidas}(s_i, \beta, M_{p,q})$. Los servicios de origen encontrados en los enlaces causales válidos entre los servicios s_i y β almacenados en la matriz $M_{p,q}$.
- $S_{c1} \times ... \times S_{cN}$. El producto cartesiano de conjuntos de servicios.

En general el algoritmo busca formar un servicio RESTful compuesto que satisfaga la petición inicial de usuario, usando los servicios individuales o simples obtenidos mediante descubrimiento. Para ello, se utilizan los operadores de planeación previamente descritos. La rutina de planeación recursiva tiene en cuenta los posible ciclos que puedan presentarse en la petición de usuario, evitando la ejecución continua del algoritmo.

El algoritmo produce una única expresión que representa un servicio RESTful compuesto, la cual es convertida a una forma normal disyuntiva, similar a la encontrada en las expresiones booleanas. Cabe aclarar que en este proceso el operador de enlace se comporta como una conjunción. Por ejemplo, la expresión:

$$A \leftarrow \left[(B \vee C) \wedge (D \vee E) \right]$$

tiene una forma disyuntiva normal dada por:

$$\left[A \leftarrow (B \wedge D) \right] \vee \left[A \leftarrow (B \wedge E) \right] \vee \left[A \leftarrow (C \wedge D) \right] \vee \left[A \leftarrow (C \wedge E) \right].$$

En este punto cada elemento no disyuntivo es una solución al problema de planeación. En el caso del ejemplo anterior, los planes resultantes del algoritmo serían:

$$\left[A \leftarrow (B \wedge D) \right], \left[A \leftarrow (B \wedge E) \right], \left[A \leftarrow (C \wedge D) \right] \text{ y } \left[A \leftarrow (C \wedge E) \right].$$

Este conjunto de planes es clasificado posteriormente, de forma que pueda seleccionarse el que provee mejor calidad. El proceso de clasificación y selección es explicado más adelante.

Posteriormente, los diferentes servicios compuestos encontrados son relacionados con la información del contexto de usuario obtenida del dispositivo del usuario. Esta relación se establece comparando los diferentes parámetros de servicio que no pudieron ser enlazados en la MEC, con la descripción de los datos del dispositivo móvil del usuario.

La comparación mencionada usa las mismas funciones de similitud utilizadas en la generación de la MEC.

5.5.6. Clasificación de planes

La clasificación de los planes generados previamente usando el algoritmo de planeación, se basa en una rutina de ordenamiento de listas, cuyo resultado está influido por una función de comparación entre pares de elementos de la lista.

La función de comparación tiene en cuenta en orden de relevancia los siguientes aspectos:

1. La calificación promedio de la preferencia de cada uno de los servicios que constituyen el servicio compuesto:
$$\frac{\sum_{s \in \text{servicios}(\pi)} \text{preferencia}(s)}{\left| \text{servicios}(\pi) \right|}.$$

2. La calificación promedio de la similitud entre los servicios encontrados en los enlaces del servicio compuesto:
$$\frac{\sum_{(s_x, s_y) \in \text{enlaces}(\pi)} Sim(s_x, s_y)}{\left| \text{enlaces}(\pi) \right|}.$$

3. La calificación promedio del orden de descubrimiento de cada uno de los servicios que constituyen el servicio compuesto:
$$\frac{\sum_{s \in \text{servicios}(\pi)} \text{descubrimiento}(s)}{\left| \text{servicios}(\pi) \right|}.$$

4. La calificación promedio de la popularidad de cada uno de los servicios que constituyen el servicio compuesto:

$$\frac{\sum_{s \in \text{servicios}(\pi)} \text{popularidad}(s)}{|\text{servicios}(\pi)|}.$$

Una vez los planes hayan sido ordenados de acuerdo con los aspectos anteriores, el plan mejor ubicado es el seleccionado para ser ejecutado, mientras los servicios restantes se almacenan, junto con la MEC asociada, en un repositorio de servicios compuestos.

5.6. Ejecución de Servicios Web RESTful

Conforme al proceso general seguido para realizar la composición dinámica de servicios, la etapa posterior a la de composición es la de adaptación. En esta etapa, un servicio compuesto en ejecución es adaptado a nuevas condiciones, derivadas de posibles fallas o de cambios requeridos por el usuario, de forma tal que pueda seguir en ejecución. No obstante, es necesario en primera instancia, considerar aspectos relacionados exclusivamente con la ejecución de los servicios, tanto de aquellos que proveen funcionalidades únicas, los cuales son resultado de la composición STE previamente vista, como de los formados por funcionalidades compuestas, obtenidos de la composición MEC.

5.6.1. Ejecución de las políticas obtenidas de la composición STE

La ejecución automática de un Servicio Web RESTful, desde el punto de vista de un cliente, requiere del conocimiento del estado actual de la aplicación, de forma tal que pueda establecerse la siguiente acción o enlace a ejecutar, según la información obtenida de la política de ejecución del servicio, previamente calculada.

La ejecución automática de un Servicio Web RESTful usando una política de ejecución, se basa en el concepto de contexto de ejecución. Este contexto, mantiene la información de los recursos del servicio que ya han sido recuperados u obtenidos en el cliente, además de la información provista por el usuario (parámetros iniciales del servicio e información del dispositivo del usuario).

A partir de la información almacenada en el contexto, el proceso de ejecución determina el estado actual de la aplicación e identifica y ejecuta el enlace correspondiente a dicho estado en la política correspondiente. La lista siguiente muestra el algoritmo de ejecución con mayor detalle.

Ejecución de un Servicio Web RESTful

1. ejecutarPolítica$(\pi, contexto)$
2. $s \leftarrow \text{estadoActual}(contexto)$
3. $a' \leftarrow \varnothing$
4. **mientras** $s \neq s_g$ **hacer**
5. $a \leftarrow A_\pi(s)$
6. **si** $a = a'$ **hacer**
7. esperar
8. **fin si**

9. $contexto \leftarrow \text{ejecutarAcción}(a, contexto)$
10. $s \leftarrow \text{estadoActual}(contexto)$
11. $a' \leftarrow a$
12. **fin mientras**
13. **fin**

La ejecución de una política de un Servicio Web RESTful individual se realiza hasta que se encuentre el estado final del servicio o hasta que se presente un error de ejecución de una acción o enlace. De acuerdo con la lista anterior, el contexto de ejecución es actualizado en cada una de las iteraciones en las que se pueda ejecutar un enlace. También el proceso tiene en cuenta los posibles ciclos que puedan presentarse al ejecutar una política determinada, estableciendo un periodo de espera entre la ejecución sucesiva del mismo enlace. Esta ejecución sucesiva del mismo enlace puede presentarse, por ejemplo, en aquellas políticas en las que se requiera cumplir una condición en el estado del servicio, antes de continuar con la ejecución.

Para la lista anterior, la rutina estadoActual determina el valor de cada una de las variables de estado, que puedan estar involucradas en las expresiones de condiciones y efectos de los diferentes enlaces de un servicio RESTful. Además el valor de la variable de estado recursoActual lo determina según información del contexto de ejecución, donde se almacena el recurso destino del último enlace ejecutado o el recurso raíz o punto de entrada de todo servicio.

$A_\pi(s)$ es la acción que puede ejecutarse en el estado s. La función ejecutarAcción, ejecuta un enlace en un contexto dado. El resultado de esta función es una representación de un recurso, cuyos datos son almacenados en el contexto de ejecución, permitiendo el establecimiento o determinación del estado actual y por tanto de la nueva acción a ejecutar.

A un nivel más práctico la ejecución de un Servicio Web RESTful inicia con un identificador URI bien definido como punto de entrada del servicio. Esta URI inicial puede requerir del establecimiento de parámetros por parte del usuario. También es posible enviar representaciones junto con la petición inicial. La petición enviada usa el protocolo HTTP con un método adecuado según la especificación del tipo de medio de las representaciones consumidas o descritas por el servicio.

Las respuestas obtenidas del servidor que posean una representación, son procesadas según su formato (XML, JSON, entre otros) y almacenadas en el contexto de ejecución, de donde se construye la URI del siguiente recurso que debe ser invocado para terminar la ejecución completa del servicio.

El Anexo C contiene en detalle el proceso de ejecución de una política que permite consumir el servicio de ejemplo que se presentó en las secciones relacionadas con la composición STE.

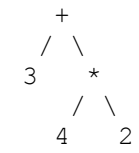
5.6.2. Ejecución de un Servicio Web RESTful compuesto

De acuerdo con lo presentado anteriormente en la composición basada en una MEC, el plan a ejecutar contiene únicamente los operadores de conjunción y de enlace, pues el

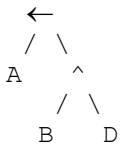
operador de disyunción es utilizado únicamente para generar diferentes planes o servicios compuestos alternativos. Por esta razón el cliente sólo tiene en cuenta los dos operadores mencionados.

La ejecución de una expresión que representa un servicio compuesto RESTful, se realiza en dos etapas. En la primera la expresión es transformada en una estructura en árbol, según los diferentes operadores que se presenten. En la segunda etapa, se recorre este árbol siguiendo unas reglas específicas que se explicarán más adelante.

La representación en árbol de una expresión de un servicio compuesto tiene la forma de un árbol de una expresión matemática. Por ejemplo la expresión aritmética 3 + (4 * 2), puede verse como:



De forma similar, una expresión de un servicio compuesto como $A \leftarrow (B \wedge D)$ puede verse de la siguiente forma:



A partir de la estructura en árbol mencionada, el procesamiento de la expresión es realizado de una forma más sencilla. La ejecución del servicio se realiza recorriendo cada uno de los nodos que integran el árbol, desde la raíz hasta la última de las hojas. Las reglas aplicadas en este procesamiento según el tipo de nodo encontrado en el árbol son las siguientes:

- Conjunción. Si el nodo encontrado es una conjunción, los nodos hijos son ejecutados en paralelo.
- Enlace. Si el nodo encontrado es un enlace, el nodo derecho del enlace (la fuente) es ejecutado en primera instancia y posteriormente se ejecuta el nodo izquierdo del enlace (el destino).
- Servicio. Si el nodo encontrado es un servicio, el nodo o servicio es ejecutado según las definiciones presentadas para la ejecución de un servicio individual.

Según lo anterior, la expresión $A \leftarrow (B \wedge D)$ se ejecutará en el siguiente orden: Primero los servicios B y D en paralelo y por último el servicio A.

En el caso de que la composición presente servicios móviles internamente, la ejecución del servicio compuesto no puede realizarse únicamente en el lado del servidor, sino también, es necesario incluir el cliente móvil en el proceso. Para realizar el consumo de un servicio compuesto de este tipo se plantean los siguientes pasos o etapas generales:

1. El estado de la ejecución de la composición es almacenado. Dicho estado, incluye los servicios que ya hayan sido ejecutados y la información obtenida de las respuestas de dichos servicios. En este punto un recurso RESTful es establecido para continuar con la ejecución del servicio compuesto en pasos posteriores.
2. Se envía al cliente una respuesta con un enlace al recurso creado en el paso anterior, de forma que pueda continuar con la ejecución, además de ello se incluye el código Java Script requerido para consumir el servicio del dispositivo móvil y los posibles datos o valores de entrada requeridos.
3. El cliente móvil ejecuta el código Java Script recibido y envía las respuestas de ejecución al enlace recibido de acuerdo con el paso anterior.
4. Las respuestas obtenidas del móvil se utilizan para continuar con la ejecución del servicio compuesto.

5.7. Adaptación del Servicio Web RESTful Compuesto

En un entorno dinámico como la Web y en especial el establecido por la Mobile 2.0, pueden presentarse diferentes cambios en los elementos que constituyen un servicio compuesto o en diferentes aspectos externos que puedan alterar y detener la ejecución de un servicio.

Bajo estas circunstancias, una composición estática que no pueda cambiar puede reducir la funcionalidad del sistema. Para evitar estos aspectos, se plantean mecanismos que permiten mantener en ejecución un servicio compuesto aún cuando se hayan presentado fallas en los servicios individuales o las condiciones del entorno que rodean el usuario cambien.

5.7.1. Adaptación del servicio compuesto debido a errores de servicio

Un error de servicio se considera como una condición irrecuperable de un servicio Web RESTful individual. Debido a que este tipo de servicios se basan en el protocolo HTTP, la forma de reconocer un error del servicio es por medio de los códigos de estado HTTP, encontrados en la respuesta de una petición HTTP.

Los códigos de estado están clasificados en grupos según el tipo de respuesta que representen. Los códigos 1XX, se refieren a códigos de información general; los códigos 2XX, referencian peticiones exitosas; los códigos 3XX, hacen referencia a códigos de redirección; los códigos 4XX, a errores del cliente y los 5XX a errores del servidor.

Para la identificación efectiva de condiciones de error del servicio, los códigos más representativos son los 4XX y los 5XX que notifican errores. En el caso de los 4XX que son del lado del cliente se seleccionan: 404 “not found”, 410 “gone”, que aunque en la teoría son considerados como errores del cliente, los dos códigos mencionados no pueden solucionarse fácilmente por el, por lo que se consideran como fallos de servicio. En el otro caso, los códigos de estado pertenecientes al grupo de los 5XX son considerados como fallas de servicio totalmente.

Ahora, cada vez que se presente una falla de servicio identificada a partir de los códigos de error HTTP mencionados, una excepción es lanzada por el sistema de ejecución, la cual contiene información de los servicios que hasta el momento antes de la falla han podido ejecutarse satisfactoriamente y el servicio que causó la falla. Es en este punto que

las diferentes expresiones de servicios compuestos generadas a partir de la MEC encuentran su utilidad.

El algoritmo para la adaptación de un Servicio Web RESTful Compuesto en ejecución cuando se presenta una falla debido a un error en un servicio individual, en general, utiliza los diferentes servicios compuestos previamente generados que no fueron usados, pero que en mayor o en menor medida representan la misma funcionalidad comparada con la provista por el servicio original que se ha intentado ejecutar. El algoritmo realiza una búsqueda en los servicios compuestos mencionados, tal que no contengan el servicio que generó el error, pero que también, incluyan la mayor cantidad de servicios que ya hayan sido ejecutados.

La siguiente lista detalla el proceso planteado, en ella: Π es el conjunto de planes de composición obtenidos previamente, los cuales ya han sido clasificados según los criterios ya presentados en una escala de mayor a menor calidad de composición, s_f es el servicio que presentó la falla y S_e es el conjunto de servicios que ya fueron ejecutados en el servicio compuesto antes de que se presentara la falla.

Adaptación debido a errores de servicio

1. adaptaciónPorErroresDeServicio(Π, s_f, S_e)
2. $\pi \leftarrow \text{primer}(\Pi)$
3. $\Pi' \leftarrow \text{remover}(\pi, \Pi)$
4. $alternativas \leftarrow \varnothing$
5. **para cada** $\pi' \in \Pi'$ **hacer**
6. **si** $s_f \notin \text{servicios}(\pi')$ **entonces**
7. adicionar$(\pi', alternativas)$
8. **fin si**
9. **fin para**
10. $\Pi_a \leftarrow \text{ordenar}(alternativas, S_e)$
11. $\pi_a = \text{primer}(\Pi_a)$
12. **si** compatibles(π_a, π) **entonces**
13. ejecutar(π_a)
14. **fin si**
15. **fin**

En la anterior definición, las rutinas:

- primer(A) obtiene el primer elemento de la lista A.
- ordenar$(alternativas, S_e)$ ordena el conjunto de servicios compuestos denominado $alternativas$ según la composición que mayor cantidad de servicios pertenezcan al conjunto de servicios ejecutados S_e.
- determina si la información actual enviada por el usuario es compatible o puede ser adaptada al nuevo servicio, de forma que la ejecución pueda realizarse sin interrupciones. Esta función se realiza mediante la comparación de los parámetros

de los servicios que han sido remplazados con los parámetros de los servicios que entran a sustituirlos.

- ejecuta la composición π_a, sin ejecutar los servicios ya ejecutados y usando los parámetros y datos de la composición que se ha estado procesando, de forma que se pierda la menor cantidad de información de estado.

5.7.2. Adaptación del servicio debido a cambios en la tasa de transferencia de la conexión del dispositivo

De acuerdo con la tasa de transferencia del dispositivo es posible seleccionar el tipo de formato o representación más adecuado para el usuario. Este proceso se realiza en base a las tablas de clasificación de los formatos presentada en el Anexo E. En cada petición realizada al servidor, cuya respuesta deba ser enviada al cliente, se establece la representación más adecuada para el cliente según las escalas presentadas en el mismo anexo para la clasificación de formatos con respecto a la velocidad de la red.

5.7.3. Adaptación del servicio debido a cambios en la petición del usuario

La adaptación de un servicio compuesto ya definido de acuerdo con las necesidades del usuario, permite adicionar, eliminar o sustituir uno de los servicios individuales, de forma que el usuario tenga control sobre la composición realizada. En las siguientes secciones se detallan cada uno de los casos de adaptación debido a cambios en la petición del usuario.

5.7.3.1. Debido a adiciones en la petición de usuario

La adición de servicios o metas a la petición de usuario es manejada, en términos generales, a partir del siguiente proceso: En primera instancia, los servicios asociados a la meta adicionada son pasados por la composición STE. Posteriormente los servicios son agregados a la MEC, según los nuevos enlaces adicionados por el usuario que conectan la composición original con el nuevo servicio. Por último, con la MEC editada, se generan nuevamente los servicios compuestos siguiendo la petición de usuario, se realiza la clasificación según los criterios ya descritos y se ejecuta el mejor servicio obtenido.

La siguiente lista detalla el proceso mencionado anteriormente. En ella R es la petición original de usuario, R' es la petición de usuario modificada, $M_{p,q}$ es la MEC asociada a la composición y Π es el conjunto de servicios compuestos.

Adición de metas a la petición de usuario

1. $\text{adicionarMeta}\left(M_{p,q}, \Pi, R, R'\right)$
2. **para cada** $s \in \left(\text{servicios}\left(R'\right) - \text{servicios}\left(R\right)\right)$ **hacer**
3. **para cada** $s_o \in \text{servicios}\left(R\right) \mid \left(s_o, s\right) \in \text{enlaces}\left(R'\right)$ **hacer**
4. $\text{adicionar}\left(\left\langle s_o, Sim\left(s_o, s\right), s\right\rangle, M_{p,q}\right)$
5. **fin para**
6. **para cada** $s_f \in \text{servicios}\left(R\right) \mid \left(s, s_f\right) \in \text{enlaces}\left(R'\right)$ **hacer**
7. $\text{adicionar}\left(\left\langle s, Sim\left(s, s_f\right), s_f\right\rangle, M_{p,q}\right)$

8. **fin para**
9. **fin para**
10. $\Pi' \leftarrow \text{planeacion}\left(R', M_{p,q}, \mathcal{KB}, \beta\right)$
11. $\Pi'' \leftarrow \text{clasificar}\left(\Pi'\right)$
12. $\pi' \leftarrow \text{primer}\left(\Pi''\right)$
13. $\pi \leftarrow \text{primer}\left(\Pi\right)$
14. **si** $\text{compatibles}\left(\pi', \pi\right)$ **entonces**
15. $\text{ejecutar}\left(\pi'\right)$
16. **fin si**
17. **fin**

En la anterior lista la matriz es actualizada con los nuevos servicios adicionados, además de los nuevos enlaces que pueden formarse, tanto desde servicios no modificados a nuevos servicios, como también en el sentido contrario. Una vez en la matriz ha sido actualizada, se realiza nuevamente un proceso de planeación, clasificación y ejecución. El nuevo servicio compuesto es relacionado con el servicio compuesto actual, con el fin de asociar los datos actuales con los parámetros ya enviados por el usuario.

5.7.3.2. Debido a eliminaciones en la petición de usuario

A diferencia del caso anterior, la eliminación de metas o servicios en la petición de usuario, no causa una modificación en la MEC asociada al servicio compuesto ejecutado. Por ésta razón, el algoritmo encargado de la adaptación se basa en la generación de nuevos servicios compuestos a partir de una petición de usuario modificada.

La siguiente lista detalla el proceso. En ella, Π corresponde al conjunto de servicios compuestos originales, R es la petición inicial de usuario y R' es la petición de usuario modificada con un conjunto de metas eliminadas.

Eliminación de metas en la petición de usuario

1. $\text{eliminarMeta}\left(\Pi, R, R'\right)$
2. $\Pi' \leftarrow \text{planeacion}\left(R', M_{p,q}, \mathcal{KB}, \beta\right)$
3. $\Pi'' \leftarrow \text{clasificar}\left(\Pi'\right)$
4. $\pi' \leftarrow \text{primer}\left(\Pi''\right)$
5. $\pi \leftarrow \text{primer}\left(\Pi\right)$
6. **si** $\text{compatibles}\left(\pi', \pi\right)$ **entonces**
7. $\text{ejecutar}\left(\pi'\right)$
8. **fin si**
9. **fin**

5.7.3.3. Debido a sustituciones de los servicios individuales en la petición de usuario

Debido a la existencia de múltiples servicios usados como alternativas para cumplir una meta específica en una composición, es posible que el servicio que ha sido seleccionado

inicialmente por el algoritmo no corresponda con la composición esperada por el usuario. Por tal razón, es la sustitución de un servicio en la composición es realizada.

Sustitución de servicios

```
1.  sustituirServicio(Π, s, s′)
2.      Π′ ← ∅
3.      para cada π ∈ Π hacer
4.          si s ∉ π ∧ s′ ∈ π entonces
5.              adicionar(π, Π′)
6.          fin si
7.      fin para
8.      π′ ← primer(Π′)
9.      π ← primer(Π)
10.     si compatibles(π′, π) entonces
11.         ejecutar(π′)
12.     fin si
13. fin
```

De acuerdo con la lista anterior, el algoritmo filtra aquellas composiciones pertenecientes a la composición original, que no poseen el servicio sustituido en lugar del servicio a sustituir. Después del filtrado, se obtiene la composición específica a ejecutar, se relacionan sus parámetros con los de la composición original y por último se ejecuta el servicio.En este caso no es necesaria la clasificación de los nuevos servicios compuestos, pues ésta se mantiene de la composición original.

Por último cabe resaltar, que los servicios preferidos o seleccionados directamente por el usuario experto, presentan la facilidad de establecer los enlaces de forma manual, en caso de que la composición falle, ya que al ser un servicio preferencial y por lo tanto el más importante en su categoría, es necesario establecer sus enlaces con los demás servicios. Aunque si el servicio presenta fallas ajenas al sistema, tales como errores desde el servidor remoto, o problemas de acceso debido a la red, se hace imposible integrarlo a un servicio compuesto y se utiliza otro de la misma categoría si existe.

Capitulo 6

Implementación y Evaluación de Prototipo

En este capítulo se plantea un prototipo y se realiza su implementación, utilizando el algoritmo desarrollado en la sección anterior, con el fin de analizar su funcionamiento en las etapas de composición y de ejecución, teniendo en cuenta la eficiencia en los tiempos de respuesta frente a eventos realizados por el usuario o por el sistema.

6.1. Prototipo

El prototipo para la composición dinámica de servicios Web RESTful fue desarrollado en el lenguaje Java edición empresarial versión 6 (Java EE 6) en el lado del servidor y el marco de desarrollo JQuery Mobile 1.0.1 basado en Java Script en el lado del cliente. La estructura establecida para la comunicación entre el cliente y el servidor fue la de un servicio Web RESTful usando representaciones en formato JSON.

En el cliente, se desarrolla una aplicación, con la cual un usuario puede crear una nueva composición estableciendo las metas, las preferencias de los servicios y los enlaces entre estas metas. Además, puede consumir el servicio compuesto y realizarle modificaciones como la adición, eliminación o sustitución de servicios ya establecidos.

El envío de las peticiones desde el cliente es realizado mediante la creación de un documento en formato JSON con la estructura requerida. En el caso de la petición de composición se envían los datos de los servicios y los enlaces entre ellos. Mientras que en el caso del consumo de un servicio compuesto, se envían los datos introducidos por un usuario o los datos producidos por la ejecución de un servicio móvil. Para el prototipo, el servicio móvil considerado, es dado por el sensor GPS del dispositivo, el cual fue accedido mediante el API estándar planteado por el W3C.

En el lado del servidor, el prototipo fue desarrollado usando el estándar JAX-RS de Java, estableciendo el método HTTP adecuado para cada tipo de operación que pueda realizarse con la aplicación. Los resultados entre peticiones de usuario son almacenados en una base de datos en memoria, asignándole identificadores únicos que permitan referenciarlos posteriormente. Estos identificadores constituyen la parte más importante en la construcción de identificadores URI enviadas al cliente para el flujo de aplicación.

Para la implementación del prototipo se consideran los servicios descritos y clasificados según metas simples de usuario como: geolocalización, mapas, eventos musicales y notificaciones. Teniendo en total 10 servicios disponibles. Para la generación de la MEC y la comparación de parámetros y atributos de servicios, se usaron los paquetes o librerías con métricas de similitud implementadas: Simmetrics (versión 1.6.2), JWI (Java WordNet Interface 2.2.1) y JWS (Java WordNet Similarity update 04).

La ejecución de los servicios Web RESTful compuestos se realiza a partir de peticiones HTTP, generadas por el cliente disponible en la implementación de referencia del estándar JAX-RS (Jersey en su versión 1.9). Los formatos de representación recibidas como respuesta, son XML y JSON, para lo cual se desarrollaron módulos específicos para la extracción de la información requerida en el proceso de ejecución. Las tecnologías

tomadas en este aspecto son JAXP (Java API for XML Processing) y Jackson (JSON Processor) para el proceso de parsing, además de XPath y JSONPath para la extracción de información precisa.

Y por ultimo en tiempo de ejecución, se utilizan los códigos de estado HTTP para identificar los tipos de error presentados en la ejecución de un servicio y para el lanzamiento de excepciones específicas, que permitan identificar las fallas de servicio y realizar los procesos de reconfiguración. Para detectar los cambios en los niveles de transferencia de datos, se utiliza un código en Java Script que mide en el lado del cliente, la velocidad al momento de realizar peticiones.

Las características técnicas del equipo utilizado y los elementos que trabajaron en conjunto para desarrollar los procesos mencionados son:

- Procesador Intel Core 2 Duo 2GHz.
- Memoria RAM de 2Gb.
- Maquina virtual de Java versión 1.6.
- SDK de android rev. 15.
- Dispositivo móvil Samsung Galaxy™ SII con plataforma android 2.3.
- Navegador Web móvil Opera Mobile 11.5

6.2. Metodología de Experimentación

El comportamiento del algoritmo se observa mediante pruebas y evaluaciones realizadas a sus diferentes módulos, es por esto que se plantea una metodología con el fin de obtener resultados oportunos y ordenados, que permitan realizar un análisis que arroje datos concluyentes sobre el comportamiento del sistema y sus posibles opciones de optimización.

Los módulos que integran el sistema para la composición dinámica de servicios Web RESTful, se clasifican en dos importantes: Modulo de composición automática y Modulo de composición dinámica, los cuales a su vez se dividen en otros, que interactúan en conjunto para lograr los objetivos del sistema, por lo tanto los criterios de evaluación adoptados como punto de comparación para el algoritmo, miden el comportamiento integro de cada modulo principal, aunque se realizan pruebas dentro de los mismos, que detallan el comportamiento al interior y el aporte que brinda cada sub-modulo al sistema.

En el estado del arte referente al campo de la composición dinámica de servicios Web RESTful, no existen claras evidencias sobre pruebas que evalúen el desempeño de módulos similares a los expuestos en este trabajo de grado, aunque hay casos como SOA4ALL [57], que realiza pruebas en servicios compuestos automáticamente basados en SOA, mecanismos dinámicos basados en conceptos semánticos para servicios tradicionales en [58], la composición automática de servicios Web tradicionales basado en la planeación de inteligencia artificial en [59], que se exponen más adelante. Debido a que las tecnologías de los servicios Web no coinciden con el planteado en este trabajo de grado (basados en REST) y la estructura tanto del algoritmo como de la arquitectura surge de la adaptación y la mezcla de conceptos de otros trabajos, es importante subrayar que los criterios de evaluación no establecen un punto de comparación cercano a los procesos del mecanismo aquí tratado, pero son tenidos en cuenta con el fin de ubicar el sistema en el contexto actual.

Por lo tanto, el método utilizado para obtener resultados que clasifiquen la eficiencia, está basado en la evaluación y experimentación con pruebas hechas a la medida para cada modulo y sus elementos internos, (que siguen ciertos patrones encontrados en las referencias consultadas para esta sección) cuyos resultados se tabulan y se grafican con el fin de explicar el comportamiento del modulo respectivo, en función del tiempo que gasta para generar una respuesta. Debido a que ningún trabajo previo presenta sub-módulos semejantes en su totalidad a los generados en este proyecto, no es posible establecer puntos de comparación detallados, pero si, obtener un balance general concluyente sobre el comportamiento integro del sistema.

Los experimentos realizados para evaluar el rendimiento del algoritmo, se basan en la variación de algunos parámetros importantes para cada modulo respectivamente, además de imponer puntos de partida para cada prueba como se detalla a continuación:

- Para el sub-modulo STE perteneciente al modulo automático se considera solamente un servicio Web RESTful, al cual se le varia internamente la cantidad de recursos y se define una estructura de manera aleatoria. Estos recursos son generados mediante un programa realizado por el equipo de trabajo en el lenguaje de programación java.
- Para el sub-modulo MEC que es un componente del modulo automático se tienen como variables, la cantidad de servicios Web RESTful que son tomados de [60] (repositorio de descripciones hREST), tanto de la petición inicial de usuario experto, como de alternativas por cada uno de ellos o el numero de parámetros de cada servicio, los resultados obtenidos por este proceso son los más importantes, ya que se establecen los enlaces entre servicios simples, que son la base principal del proceso de composición. Para este sub-modulo es preciso profundizar en un aspecto relevante para su funcionamiento, el cual es el analizador sintáctico, donde se consideran como variables la cantidad de parámetros que se comparan entre sí.
- Al igual que en el MEC, en el generador de grafos se tiene como variable la cantidad de servicios integrantes de la petición inicial de usuario experto (estos servicios al igual que para la composición MEC son obtenidos del repositorio encontrado en [60]), a diferencia de las alternativas por cada servicio, que no varían su cantidad. En el sub-modulo siguiente, correspondiente al Ranking, se varían los grafos generados previamente, manteniendo la cantidad de servicios por cada uno, con el fin de evaluar el proceso de clasificación de grafos, que guardan relación con la petición inicial de usuario.
- Para realizar las medidas de eficiencia al modulo dinámico, se plantean dos aspectos importantes: el primero se presenta cuando se tienen servicios compuestos bajo ejecución, lo que permite eventos ocasionados directamente por el usuario (adición, eliminación o sustitución de servicios simples); el segundo se da entre el envió de la petición de usuario en tiempo de ejecución y la respuesta por parte del servidor, que es aquí donde se detectan las fallas de los servicios simples o los cambios en el ancho de banda de la red inalámbrica; en ambos casos se mide el tiempo que tarda el sistema en reconfigurar el servicio compuesto. Para el desarrollo de estos procesos, se varían la cantidad de opciones para reconfigurar el servicio y la cantidad de servicios que integran al compuesto.

- Todos los resultados obtenidos, son organizados y graficados para obtener una clara perspectiva sobre el comportamiento del algoritmo adaptado, además de establecer un paralelo con los criterios de evaluación impuestos.

6.3. Criterios de Evaluación

Los criterios de evaluación son una herramienta clave para determinar el nivel de calidad y eficiencia del algoritmo para la composición dinámica de servicios Web RESTful, ya que son el punto de referencia sobre el cual se verifican los objetivos planteados y se adquieren datos para generar posibles soluciones o formas de optimización.

Los puntos de comparación posibles están soportados en investigaciones y trabajos previos por parte de terceros, los cuales no son acogidos totalmente, debido a que difieren en ciertos aspectos a los planteados en el presente trabajo de grado, pero que se constituyen en un enfoque externo al sistema que es preciso tener en cuenta, con el fin de observar el posicionamiento que tiene en el contexto actual.

En [57] se tienen los resultados experimentales para la composición automática de servicios Web SOA, teniendo como variables, la cantidad de usuarios y servicios en cada escenario de pruebas. El modulo de composición en este trabajo implementa el framework de Spring, el cual le brinda las funciones de configuración y gestión de los servicios Web; la composición se realiza en base a plantillas generadas previamente, que fijan el camino de la composición. Para la experimentación se tienen 5 casos diferentes: Caso 1: (10 peticiones de usuario con un máximo de 5 servicios), Caso 2: (50 usuarios; 10 servicios), Caso 3: (100 usuarios; 15 servicios), Caso 4: (150 usuarios; 20 servicios) y Caso 5: (200 usuarios; 25 servicios), los cuales se observan en la Figura 19, en donde los usuarios seleccionan las plantillas (generadas previamente, cuyo tiempo corresponde a la barra más oscura) en grupos o alternativas de 2, 4, 7, 10 y 12 que son utilizadas para realizar los enlaces semánticos entre servicios. El tiempo que tarda el bloque de composición, depende de diversos aspectos, tales como la dimensión de las ontologías, el número de reglas de dominio del lenguaje específico, las descripciones semánticas disponibles de servicios Web, los agentes que trabajan, etc. El último proceso, correspondiente a la optimización, se encarga de mejorar la composición generada mediante el reemplazo de servicios por otros que brinden mejores prestaciones.

Todas estas pruebas se realizaron en una maquina con un procesador de 2.4GHz, 2Gb de RAM y en un sistema operativo Linux.

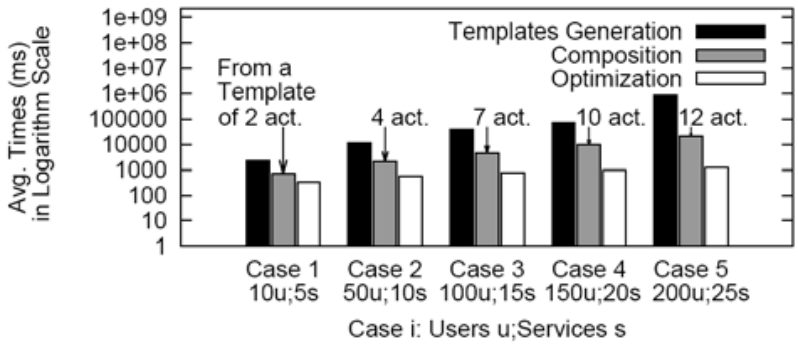


Figura 19. Escalabilidad de la Arquitectura propuesta por F. Lécué*, et al* [57].

En la Figura 20 se exponen los resultados de evaluaciones empíricas realizadas al mecanismo de composición de servicios dinámica basado en grafos semánticos, SeGSeC (Semantic Graph-Based Service Composition) propuesto en el estudio de K. Fujii and T. Suda [58]. Este sistema está compuesto por cuatro bloques importantes: analizador de petición de usuario (realiza procesamiento de lenguaje natural), modulo compositor de servicios (establece los enlaces entre servicios), analizador semántico (evalúa si el flujo de datos es correcto con los enlaces dados por el modulo anterior) y por último el modulo de ejecución (ejecuta el esquema resultante de los módulos anteriores).El desempeño de la composición, es afectado por el número de componentes, por lo tanto, los tiempos empleados para obtener un grafo, presentan un comportamiento que depende de dicha cantidad, es decir, entre más elementos de composición se requieran, mayor será el tiempo para componerlos[7].

Todas estas pruebas se realizaron en una maquina con un procesador Pentium 4 de 1.7GHz, 384Mb de RAM y J2SE versión 1.5.

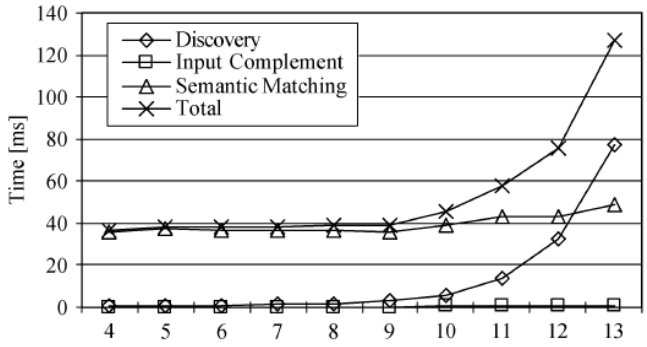


Figura 20. Comportamiento del sistema planteado por K. Fujii and T. Suda [58]

En el trabajo de F. H. Khan*, et al* [59] se plantea el algoritmo para la composición automática de servicios Web tradicionales, con mayor grado de similitud al tratado en este trabajo de grado (en cuanto a módulos y conceptos para la composición). En él, se presenta un mecanismo de composición basado en la planeación de inteligencia artificial, cuyo proceso inicia con la petición de usuario en un lenguaje formal, el cual es traducido por el sistema con el fin de comprender los requisitos deseados y así obtener los servicios Web de un UDDI o de una base de datos local (WSDBs optimiza los tiempos de búsqueda). Tales servicios son enlazados en base a un plan generado por inteligencia artificial, generando diversas posibilidades que cumplen con los requisitos iniciales. Al final un evaluador selecciona el más óptimo y retorna la respuesta al usuario. El framework propuesto en [59] cuenta con módulos similares a los desarrollados y adaptados en el trabajo de grado, aunque solo en el modulo automático, por lo cual no se plantea como una alternativa de peso en el Capitulo .

Los resultados que se observan en la Figura 21, reflejan el tiempo que tarda el algoritmo en componer servicios Web, con una cantidad de servicios integrantes, que va desde 2 a 8, variando el tamaño del repositorio del cual se toman dichos servicios (eje horizontal).

[7] Para el caso particular del presente trabajo de grado, el tiempo empleado para el descubrimiento de servicios, no es tenido en cuenta, debido a que no se implementa.

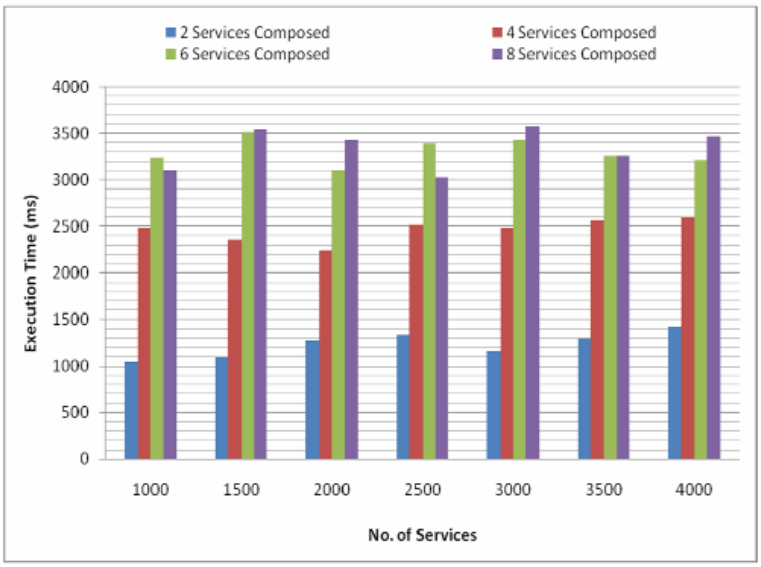


Figura 21. Eficiencia del algoritmo propuesto por F. H. Khan*, et al* [59].

6.4. Plan de Pruebas

Para el plan de pruebas, se consideran los sub-módulos y funcionalidades que integran cada modulo principal.

En el modulo automático.

- STE (Sistema de Transición de Estados): encargado de componer la secuencia correcta de mensajes al interior de un servicio Web RESTful.
- MEC (Matriz de Enlace Causal): su función es generar los enlaces y establecer relaciones entre los servicios Web RESTful.
- Generador de Grafos: toma los resultados del proceso anterior (MEC) y genera los grafos o estructuras que coinciden con la petición inicial de usuario experto.
- Ranking: ordena todos los grafos resultantes, teniendo en cuenta los aspectos de clasificación descritos en la sección 5.3 del presente documento.

En el modulo dinámico.

- Se presentan la reconfiguración a causa de eventos independientes del usuario, tales como fallas en los servicios o variaciones en los niveles de velocidad de la red (dos primeras pruebas en la Tabla 20).
- La reconfiguración debido a eventos dependientes del usuario, tales como adición, eliminación y sustitución de servicios simples (tres pruebas finales en la tabla más adelante).

MODULO AUTOMATICO	Las pruebas para este módulo tienen como objetivo, medir la eficiencia para generar grafos y elegir el más apropiado con respecto a la petición de usuario y las características del dispositivo. Para ello se realizan peticiones de usuario que varíen su cantidad de elementos integrantes, es decir, peticiones que vayan desde 2 hasta 16 servicios simples, al mismo tiempo que la cantidad de alternativas para cada uno.	
Sub-modulo	**Prueba**	**Actividad**
STE	Eficiencia del sub-modulo de composición STE.	Variar la cantidad de recursos que conforman un servicio Web RESTful, cuantificando el tiempo que toma en encontrar la secuencia de enlaces correcta, entre recursos de un mismo servicio.
MEC	Eficiencia para el enlace entre servicios Web RESTful disponibles.	Modificar la cantidad de servicios Web RESTful y el número de alternativas por cada uno de ellos en la matriz.
	Tiempo de establecimiento de enlaces entre parámetros.	Variar el número de parámetros, sin tener en cuenta la cantidad de servicios Web RESTful.
Generador de Grafos	Tiempo para la generación de grafos posibles.	Variar la cantidad de servicios de la petición inicial de usuario experto.
Ranking	Eficiencia del Sub-modulo Ranking.	Medir el tiempo utilizado por el modulo en clasificar los grafos, variando la cantidad de los mismos.

Tabla 19. Plan de Pruebas Modulo Automático

MODULO DINAMICO	Las pruebas para este modulo tiene como objetivo, medir la eficiencia en los tiempos de reconfiguración del servicio compuesto, considerando eventos dependientes e independientes del usuario. Para esto se desarrollan las actividades descritas a continuación.
Prueba	**Actividad**
Reconfiguración debido a errores de servicio	Medir el tiempo que tarda el sistema en su parte dinámica, para reconfigurar el servicio compuesto, frente a una falla en un servicio simple, variando la cantidad de opciones posibles para la solución y la cantidad de servicios que fallan al mismo tiempo.
Reconfiguración debido a errores en niveles de señal de red.	Medir el tiempo que tarda el modulo en reconfigurar la respuesta final del sistema, variando la cantidad de representaciones y la velocidad de la red.
Adición de servicios simples	Cuantificar el tiempo que le toma al modulo dinámico agregar un servicio, variando la cantidad de opciones disponibles.
Eliminación de servicios simples	Medir el tiempo que tarda la reconfiguración del servicio, al eliminar servicios simples, teniendo en cuenta la integridad del compuesto.
Sustitución entre servicios simples	Medir el tiempo que tarda el modulo en sustituir un servicio por otro, a voluntad del usuario, variando la cantidad de sustituciones simples (servicios uno a uno).

Tabla 20. Plan de Pruebas Modulo Dinámico

6.5. Resultados y Análisis de los Resultados

Para cada uno de los módulos se obtuvieron resultados en base al plan de pruebas de la sección anterior, los cuales miden la eficiencia del sistema. Estos datos son clasificados en dos partes, una para cada modulo respectivamente, en las cuales el análisis de la

información adquirida, integra los objetivos del proyecto, los criterios de evaluación y las medidas obtenidas.

Los resultados para el modulo automático se estructuran de manera jerárquica en dos niveles, el primero siendo el modulo mismo como un conjunto de bloques que interactúan entre ellos para obtener un resultado final y el segundo nivel, como cada uno de los sub-módulos de manera individual y aislada al resto del sistema.

Para el modulo dinámico se obtuvieron resultados referentes a la reconfiguración del servicio, haciendo hincapié en la diferencia entre la acción ejecutada por voluntad del usuario experto, o a causa comportamientos de contingencia del algoritmo.

6.5.1. Modulo automático

Los resultados obtenidos para este modulo se reflejan en la Figura 22, en la cual se observa un patrón creciente en los valores medidos, aunque se obtienen picos abruptos de forma aleatoria, esto se debe al uso de la herramienta WordNet y la diferencia en la cantidad de parámetros por servicio en el algoritmo, cuyos tiempos de ejecución varían en función de la complejidad de los términos comparados al realizar los enlaces.

Por lo tanto es oportuno mostrar cómo se comporta el algoritmo para componer servicios Web RESTful, cuando se consideran parámetros con sintaxis semejantes, con el fin de aislar el proceso de composición cuando se realiza netamente de manera sintáctica. Las medidas obtenidas se muestran en la Figura 23.

Los criterios para evaluar el comportamiento del algoritmo, son abstraídos de los siguientes trabajos, expuestos anteriormente:

- SOA4All: An innovative integrated approach to services composition [57] (Algoritmo 1).
- Semantics-based dynamic service composition [58] (Algoritmo 2).
- QoS Based Dynamic Web Services Composition & Execution [59] (Algoritmo 3).

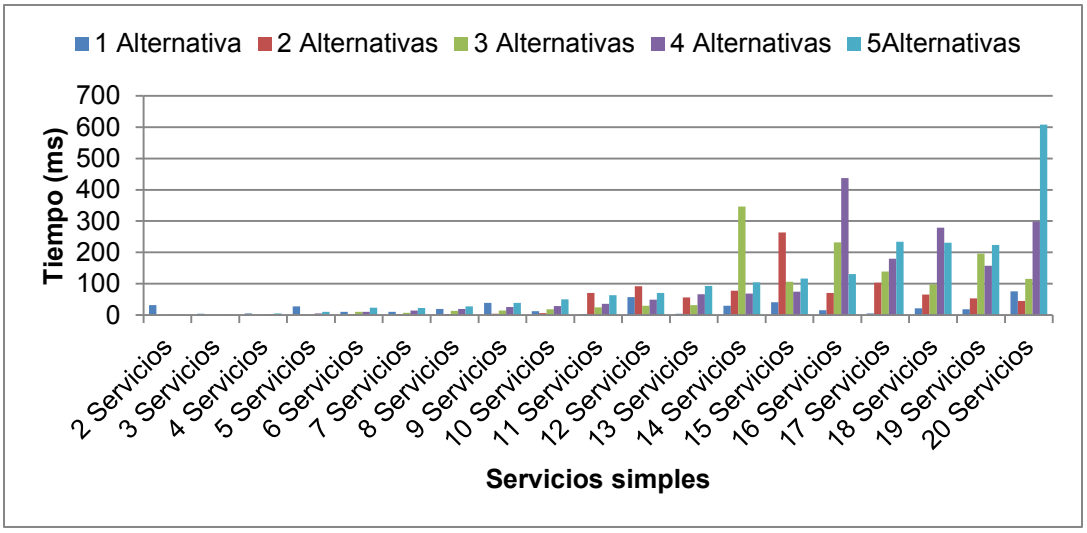


Figura 22. Eficiencia del módulo automático

No. de Servicios de Petición	Tiempo (ms)			
	1 Alternativa	2 Alternativas	3 Alternativas	4 Alternativas
2	31.6	0	0	0.4
4	5.2	1	2.4	3
6	10.8	2.2	10.8	10.2
8	19.8	3.4	13.4	19.6
10	12	6.4	18.6	29
12	41.4	92	30.2	49.2
14	30.2	77.4	346.6	68.2
16	16	70.8	231.8	437.4
18	21.4	65.4	98.6	278.6

Tabla 21. Eficiencia del módulo automático

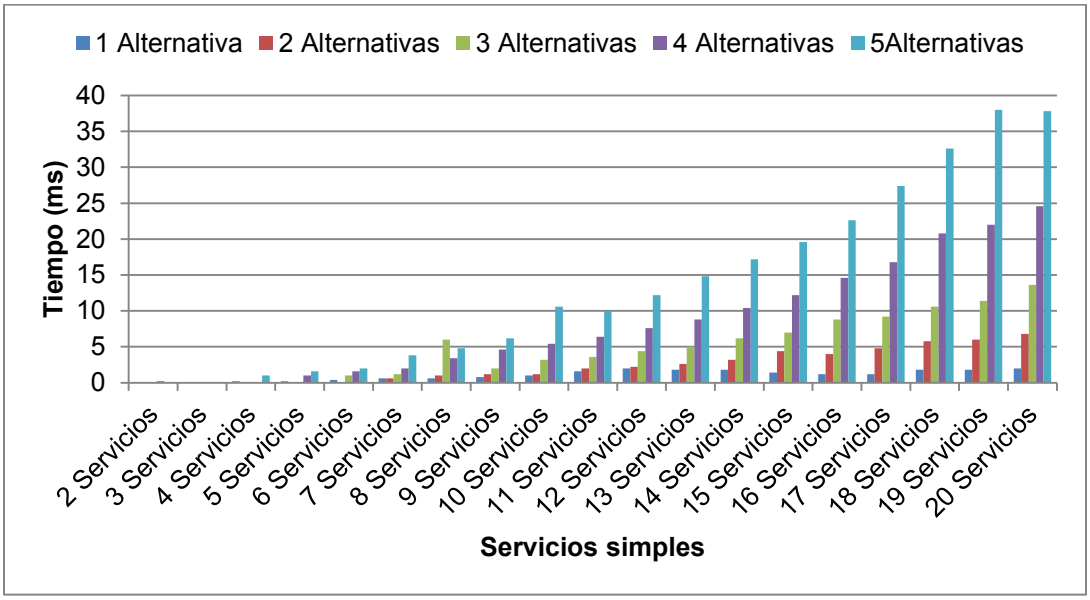


Figura 23. Eficiencia del módulo automático con (servicios con parámetros ideales)

Las comparaciones entre los diversos mecanismos y el expuesto en este trabajo de grado siguen a continuación:

Algoritmo 1 vs Algoritmo para la composición dinámica de servicios Web RESTful.

La medida de eficiencia del algoritmo 1, comprende todo el proceso desde la generación de plantillas, hasta la composición y su respectiva optimización (proceso adicional a petición del usuario), cuyos tiempos se encuentran entre 3.3 segundos y 877 segundos para los casos 1 (5 servicios y 2 plantillas) y 5 (25 servicios y 12 plantillas) respectivamente.

Para el algoritmo del presente trabajo de grado, se obtuvieron 1.6 milisegundos y 33.4 segundos en los mismos casos, considerando aleatorios los parámetros de los servicios simples (cantidad y sintaxis). Para el caso en donde los parámetros están rigurosamente controlados se obtienen valores menores a 1 milisegundos y 373 milisegundos respectivamente.

Las plantillas del algoritmo 1 corresponden a la cantidad de servicios alternativos en el algoritmo RESTful[8] y los servicios la cantidad que integra la petición inicial. Esta comparación ratifica en este caso, que la eficiencia aumenta cuando se utilizan métodos sintácticos en lugar de semánticos.

Algoritmo 2 vs Algoritmo para la composición dinámica de servicios Web RESTful.

Para el algoritmo 2 se obtienen los tiempos de composición desde que un usuario solicita en lenguaje natural un servicio Web de complejidad variable, hasta que se obtiene como resultado final un enlace entre servicios acorde con dicha petición, estos enlaces se realizan de forman semántica en base a ontologías. Los resultados que arroja este algoritmo se basan en una opción ya predefinida por cada servicio y composiciones que van desde 4 a 13 integrantes.

Los tiempos se mantienen por debajo de los puntos de referencia, tanto, en el caso donde se consideran parámetros aleatorios, como si son controlados. Existe sin embargo una medida que se encuentra por encima de dichos valores de referencia (12 servicios), debido a que en ese caso, el algoritmo RESTful empleó más tiempo para establecer enlaces que relacionaban parámetros muy diferentes entre sí. En la Figura 20[a] se demuestra que el algoritmo 2 también basa sus tiempos de composición, en función de la complejidad de los enlaces (aunque semánticamente), ya que para la primer y última prueba, que emplean 4 servicios cada una, los tiempos de composición son 18.1ms y 76.3ms respectivamente.

Algoritmo 3 vs Algoritmo para la composición dinámica de servicios Web RESTful.

Los resultados que brindan las pruebas realizadas al algoritmo 3, varían en relación al tamaño de un repositorio donde se encuentran alojados los servicios Web, y a la cantidad de elementos que pertenecen al servicio Web compuesto. Tales resultados varían desde 1.05 segundos hasta 3.23 segundos para un repositorio de 1000 servicios y llega hasta un máximo de 3.58 segundos aproximadamente cuando existe un repositorio de 3000 servicios, aunque dicho repositorio es global, donde para ciertas solicitudes puede darse el caso de existir solo una alternativa, como N posibles para un mismo servicio Web.

Para este caso, se tiene que el algoritmo RESTful obtiene medidas semejantes para el mínimo y el máximo (1.05 s y 3.58 s), cuando se componen servicios de 2 elementos y 114 alternativas cada uno y de 8 elementos y 21 alternativas respectivamente. Lo cual significa que es posible obtener 2^{114} y 8^{21} composiciones, cuando se obtiene un porcentaje de enlaces igual al 100% de éxito; cifras que son notablemente muy grandes para ofrecer a un usuario de composición de servicios Web RESTful.

A continuación se exponen y analizan los resultados obtenidos para cada sub-modulo, tal como lo especifica el plan de pruebas.

6.5.1.1. Resultados STE

La Figura 24 muestra la eficiencia del sub-modulo STE, el cual es el encargado de componer la estructura interna de cada servicio. Para la obtención de estos resultados, se cambiaron la cantidad de recursos y la organización al interior de un servicio Web

[8] Es el algoritmo presentado en este trabajo de grado, se denomina de esta manera, para mayor comodidad en las tablas y comparaciones.

RESTful, comenzando desde 1 hasta 100 recursos, con interconexiones posibles de forma aleatoria.

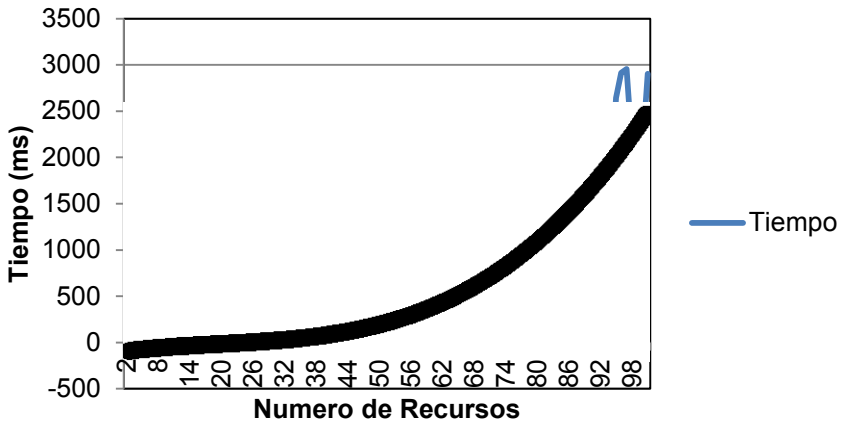


Figura 24. Tiempo de composición STE

Es difícil obtener a simple vista los tiempos cuando el número de recursos es pequeño, por lo cual en la Tabla 22 se especifican los primeros resultados obtenidos para este sub-modulo.

No. de Recursos	Tiempo (ms)
2	0.4
4	0.2
6	0.6
8	1.4
10	1.8
12	4.2
14	7

Tabla 22. Tiempos de composición STE

6.5.1.2. Resultados MEC

El sub-modulo MEC luego de ser sometido a prueba, arrojó los datos mostrados en la Figura 25 en donde se obtienen los tiempos que toma enlazar los servicios Web RESTful. Para comprender la grafica, hay que partir del método utilizado por el modulo MEC para componer los servicios, los cuales han sido enviados previamente por el bloque de descubrimiento (el cual no pertenece a este sistema) y están organizados por categorías según el objetivo común de cada uno de ellos, por lo cual se convierten en alternativas para ocupar una determinada posición dentro del grafo de composición final, es por esto que el tiempo que tarda el sub-modulo en completar el proceso, nace de la variación de la cantidad de servicios disponibles por cada categoría y categorías, generando un crecimiento, tanto horizontal como vertical de la matriz de enlace causal. Al igual que el sub-modulo anterior el crecimiento es exponencial, debido que al establecer los enlaces, se consideran las relaciones entre todos los servicios pertenecientes a la matriz, excluyendo los que pertenecen a la misma categoría. Este modulo, considera el orden de la petición inicial de usuario antes de establecer los enlaces con el fin de evitar ciclos indeseados y reducir los tiempos de composición MEC.

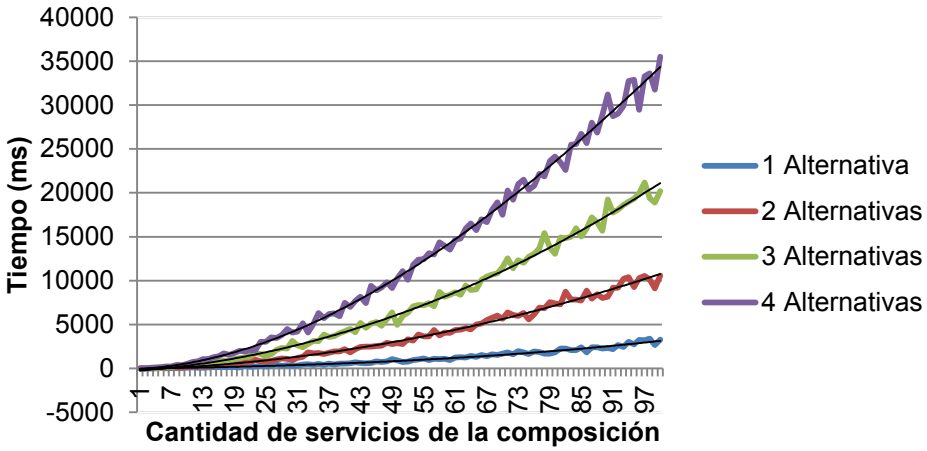


Figura 25. Tiempo de composición MEC

más adelante se muestran los primeros tiempos obtenidos para esta prueba, cuyo objetivo es demostrar claramente la eficiencia del sistema en esta sección, además de permitir el análisis más cómodo entre el modulo total y el aporte de este sub-modulo.

No. de Servicios de Petición	Tiempo (ms)			
	1 Alternativa	2 Alternativas	3 Alternativas	4 Alternativas
2	8.5	1.2	6	14.9
4	4.5	37.8	53.8	106.8
6	15.7	61.9	98.8	220.7
8	33.5	107.9	248.1	393.9
10	59.9	199.6	452.8	527.8
12	81.1	192.9	452.9	793.9
14	114.8	237.2	511.7	1042.1
16	144.7	384.4	702	1385.8
18	146.9	539.3	824.2	1529.8
20	129.4	515.4	1152.3	2020.6
22	196.4	683.5	1340.5	2041.1

Tabla 23. Tiempo de composición MEC

En la Figura 26 y la Tabla 24 se consideran los parámetros de los servicios sintácticamente semejantes, con el fin de evaluar el modulo MEC cuando no es necesario utilizar WordNet, para establecer los puntajes de los enlace, es decir cuando se realizan enlaces únicamente sintácticos, además de que permite apreciar el tiempo que esta herramienta de comparación le agrega al proceso.

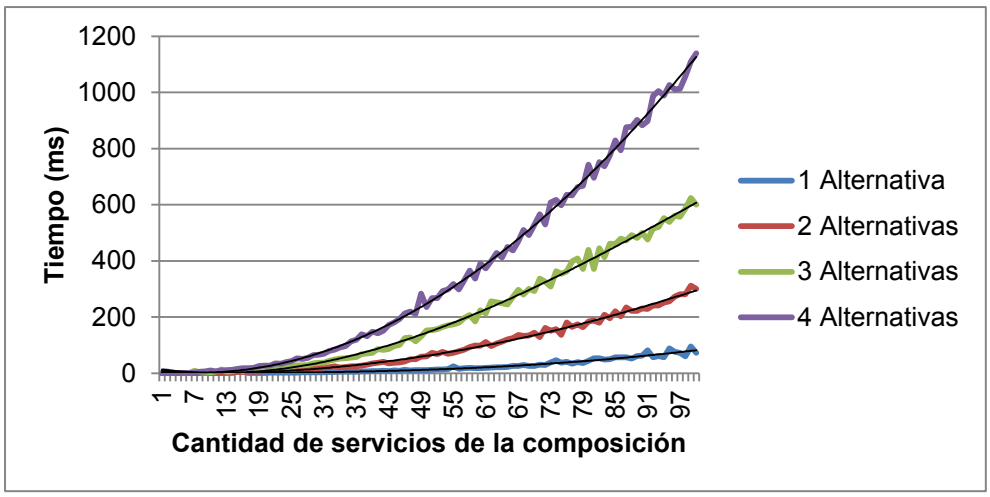


Figura 26. Tiempo de composición MEC (servicios con parámetros ideales)

No. de Servicios de Petición	Tiempo (ms)			
	1 Alternativa	2 Alternativas	3 Alternativas	4 Alternativas
2	7.3	0.01	0.01	0.01
4	0.3	0.01	0.01	0.01
6	0.2	0.01	0.8	1.1
8	0.3	0.4	2.9	5.3
10	0.3	2.1	3	10.4
12	0.4	1.8	14.1	10.4
14	7	4.5	7.9	12.3
16	1.1	4.1	8.4	17.7

Tabla 24. Tiempo de composición MEC (servicios con parámetros ideales)

La Figura 27 corresponde a un proceso interno en este sub-modulo, que presenta gran importancia en el desempeño del mismo, en ella se observa el tiempo que tarda el analizador sintáctico en comparar parámetros a medida que aumentan. Los resultados obtenidos generan una curva que tiende a estabilizarse en 2.5 segundos considerando más de 10.000 parámetros, esto se debe a que el mecanismo que se usa en el algoritmo para este fin, considera las comparaciones ya realizadas con anterioridad y les asigna el puntaje obtenido previamente, sin necesidad de realizar el proceso de comparación.

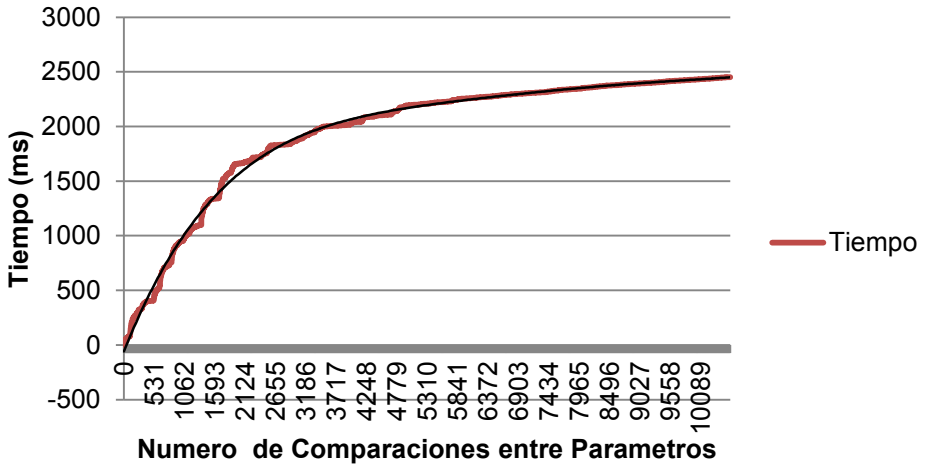


Figura 27. Tiempo de comparación entre parámetros

6.5.1.3. Resultados Generador de Grafos

Cuando se han establecido todos los enlaces posibles entre servicios Web RESTful, es preciso armar los grafos. El tiempo que tarda este modulo en generar todos los grafos posibles, teniendo en cuenta la cantidad de servicios que integran la petición inicial de usuario experto se muestra en la Figura 28, en donde se vuelve a encontrar el patrón de crecimiento exponencial observado en graficas anteriores, debido a que la cantidad de servicios que integran la petición de usuario aumenta.

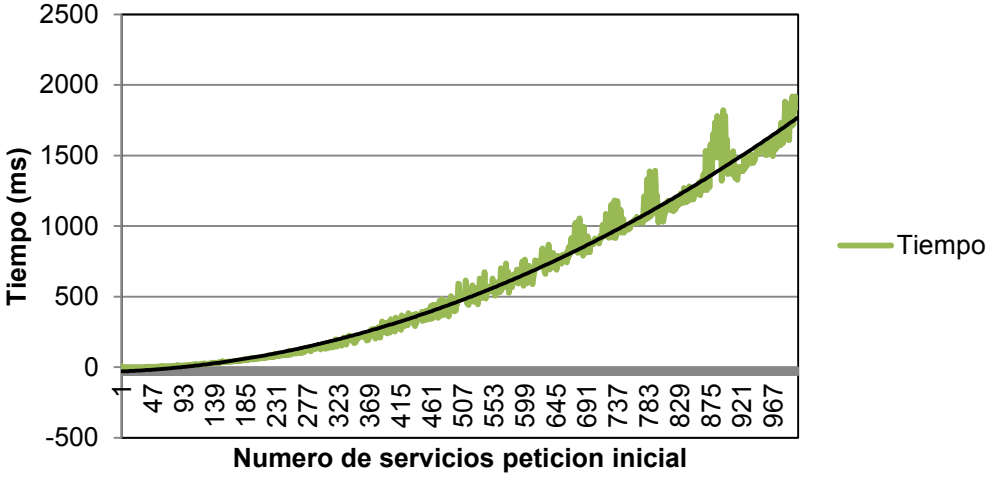


Figura 28. Tiempo de generación de grafos

El tiempo utilizado por este modulo, no sobrepasa los 2 segundos cuando la petición inicial considera una composición con 1000 servicios Web RESTful. En la tabla se muestran los valores obtenidos para los primeros valores al generar los grafos.

No. de Servicios de Petición	Tiempo (ms)
2	0.01
4	0.1
6	0.1
8	0.1
10	0.1
12	0.1
14	0.1

Tabla 25. Tiempo de generación de grafos

6.5.1.4. Resultados Ranking

A diferencia de los demás gráficos, el Ranking arroja resultados que generan una curva con características lineales (Figura 29), que varían su pendiente a medida que la cantidad de servicios que integran los grafos varían. Se observa que existen cambios abruptos en los tiempos a lo largo de las curvas, esto se debe a la naturaleza aleatoria del mecanismo que genera los puntajes de los servicios para la prueba, ya que el algoritmo utiliza cuatro criterios para la clasificación de manera jerárquica para establecer el orden, en otras palabras, si se obtienen resultados contundentes desde el primer nivel de clasificación, no es necesario procesar los siguientes niveles, con lo cual el tiempo comprendido es menor.

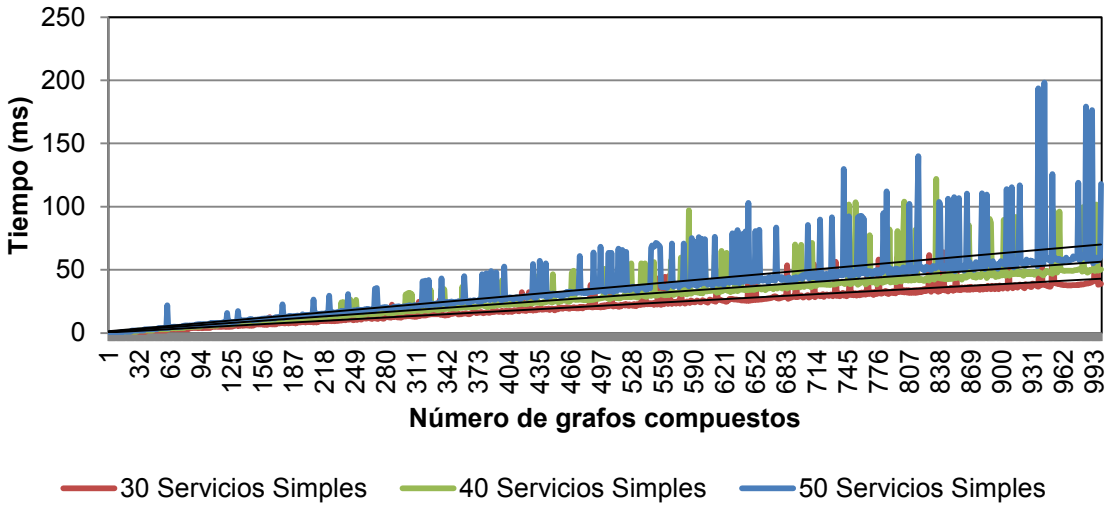


Figura 29. Tiempo para el ranking de los grafos compuestos

Para grafos con pocos servicios en su haber, se obtienen valores tan pequeños y cercanos a cero, que hacen de este proceso el que emplea menos tiempo en todo el sistema.

6.5.2. Modulo dinámico

Las graficas resultantes para cada una de las pruebas planteadas para este modulo se presentan de forma ordenada a continuación.

6.5.2.1. Reconfiguraciones independientes del Usuario

Estos eventos se generan independientemente del usuario experto o del algoritmo para la composición, los cuales son las fallas que sufren los servicios simples, desde su servidor de origen (Fallas tipo I) o las fallas que están relacionadas al contexto (Fallas tipo II) del dispositivo móvil.

En la Figura 30 se tienen los tiempos que emplea el algoritmo en su parte dinámica, para recomponer un servicio compuesto por 100 elementos, en donde se varia la cantidad de grafos en el almacén de grafos, es decir el tiempo se mide en función de la cantidad de alternativas que el algoritmo tiene que evaluar, antes de llegar al sustituto correcto. Además, el tiempo varía también en función del porcentaje del servicio compuesto ejecutado, ya que en base a esta condición, el algoritmo determina si al recomponer un servicio, es posible continuar con el flujo de datos o es necesario retroceder para obtener una reconfiguración acorde con la que fue ejecutada.

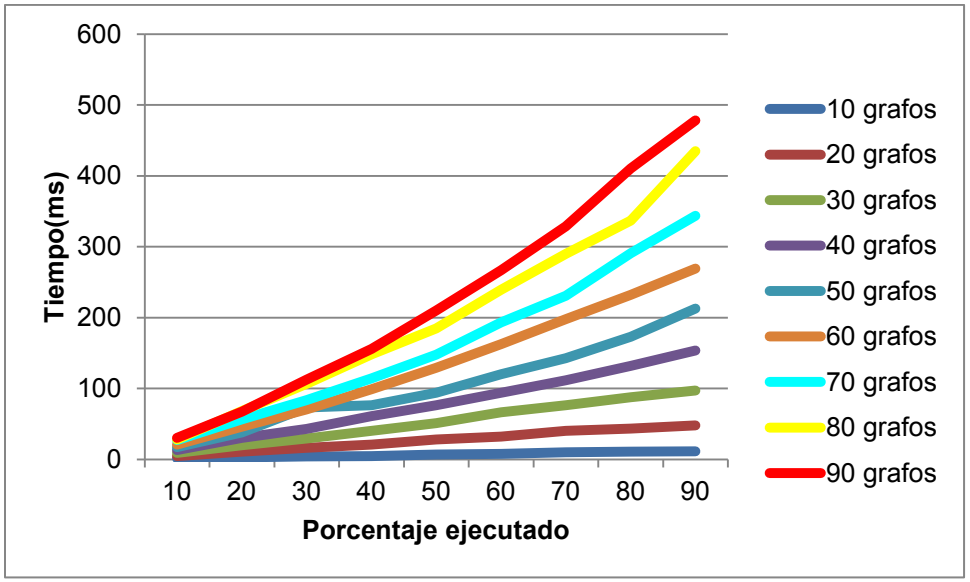


Figura 30. Tiempo de reconfiguración debido a fallas en servicios simples

Para las pruebas de reconfiguración a causa de Fallas tipo II, se obtienen los resultados de la Figura 31, en donde se cuenta con la variación de los servicios Web RESTful finales (los últimos en los grafos) y la cantidad de representaciones por cada uno.

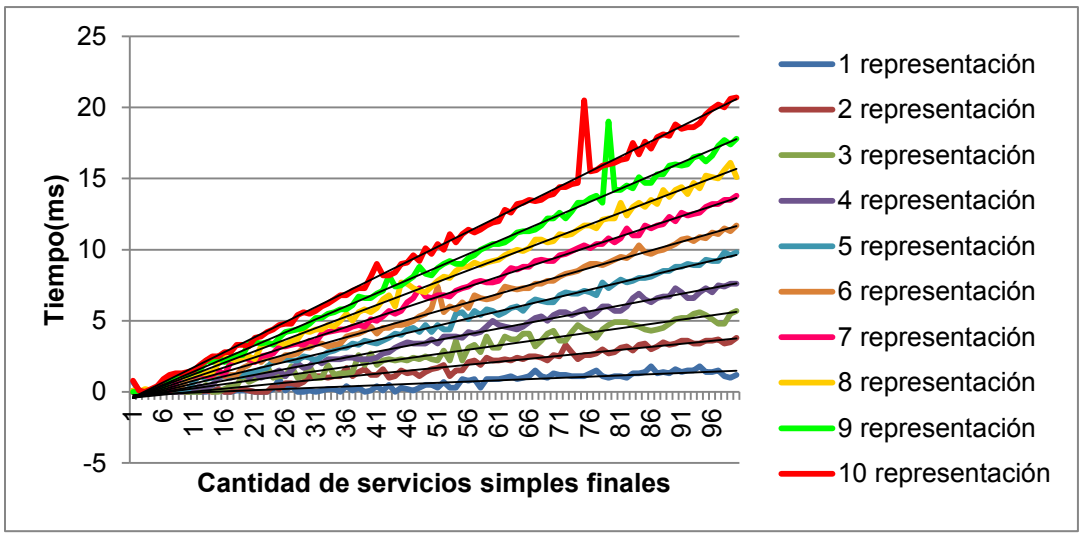


Figura 31. Tiempo de reconfiguración debido a cambios en la velocidad de la red

Los resultados a los valores más pequeños para cada una de las pruebas, se exponen en la Tabla 26 y la Tabla 27.

Porcentaje Ejecutado	Tiempo (ms)				
	1 Rep.	2 Rep.	3 Rep.	4 Rep.	5 Rep.
10%	1.3	4.5	9.3	13	17.3
20%	2.8	10.8	19.9	30.5	37.9
30%	4.2	16.2	29.3	43.2	73.3
40%	4.7	20.9	40.1	61	76.5
50%	6.9	27.9	51.3	76.3	94.1
60%	7.6	32.3	66.6	93.8	120

Tabla 26. Tiempos de reconfiguración frente a fallas tipo I

Servicios Finales	Tiempo (ms)				
	1 Rep.	2 Rep.	3 Rep.	4 Rep.	5 Rep.
2	0,1	0	0	0,1	0
4	0,1	0	0	0	0,1
6	0	0	0	0,1	0
8	0	0	0	0	0,1
10	0	0	0	0,1	0,1
12	0	0	0	0,4	1,2

Tabla 27. Tiempos de reconfiguración frente a fallas tipo II

6.5.2.2. Reconfiguraciones dependientes del Usuario

Estas reconfiguraciones se generan debido a órdenes directas del usuario en tiempo de ejecución, lo cual le permiten interactuar con la estructura del servicio que el mismo compuso, tales interacciones son la adición, la eliminación o la sustitución de servicios simples.

La Figura 32 muestra el tiempo empleado por el sistema para adicionar servicios simples, considerando la cantidad que se agregan al mismo tiempo. Esta prueba se realizó con un servicio base, compuesto por 10 servicios simples; la variación en el tiempo ocurre de forma lineal con pendiente positiva, presentando un crecimiento constante a medida que la cantidad de servicios simples que se adicionan aumenta, esto se debe a que los servicios cuando son adicionados, generan un crecimiento proporcional en la cantidad de enlaces que se realizan de forma secuencial.

En la Figura 33 se muestran los resultados de realizar el proceso inverso a la adición. En este caso se eliminan servicios simples del compuesto, teniendo en cuenta la integridad de la composición, ya que existe una limitante en la cantidad de servicios que pueden ser borrados para restablecer una composición funcional. Este evento se manifiesta en los resultados a medida que el porcentaje aumenta, es decir, cuando la cantidad de servicios que se sustraen es cada vez mayor, la complejidad para reconfigurar el servicios ejecutado crece, causando aumentos en los tiempos empleados para la reconfiguración.

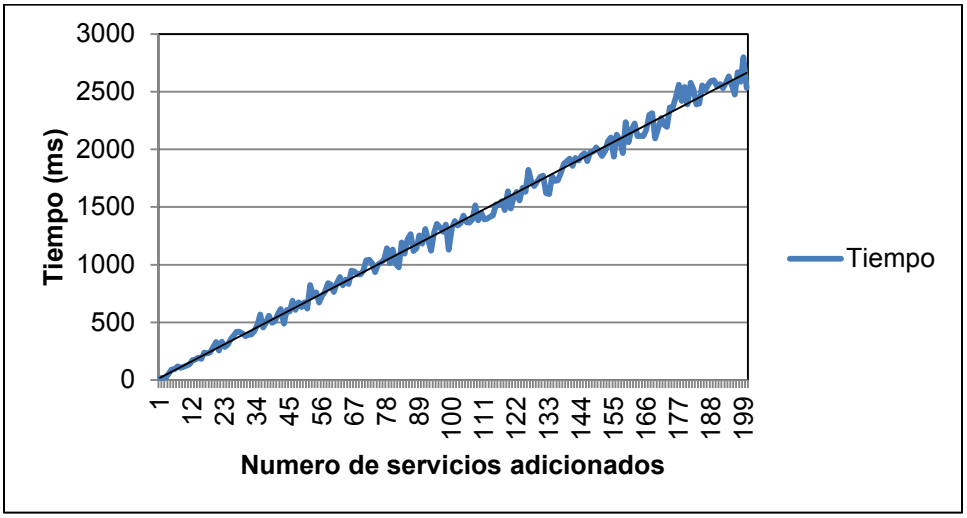


Figura 32. Tiempo de reconfiguración por adición de servicios simples

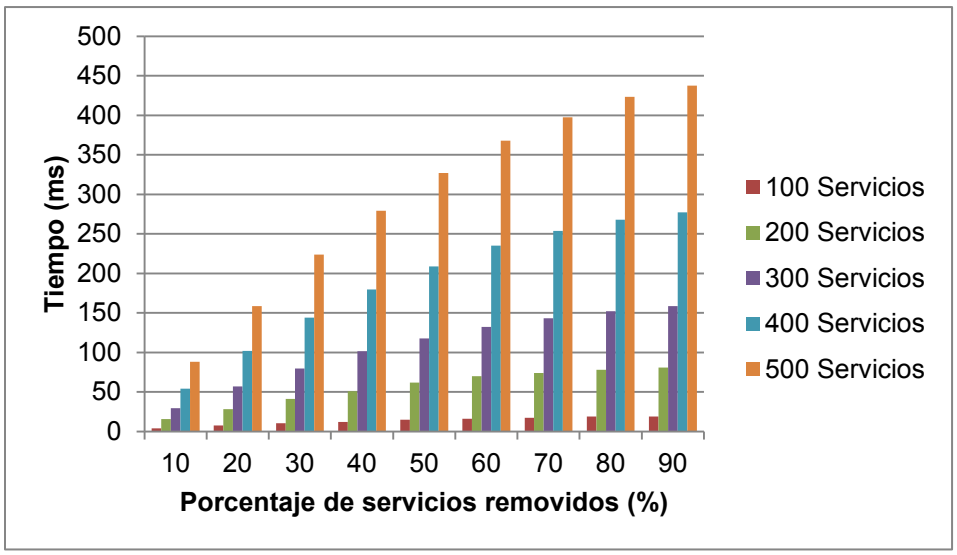


Figura 33. Tiempo de reconfiguración por eliminación de servicios simples

La sustitución de servicios se realiza de uno a uno, es decir que solo se pueden reemplazar un servicio simple por uno similar en su categoría y funcionalidad, que a diferencia de cuando ocurre un fallo, este evento se genera por una petición directa del usuario experto. Para esta prueba se cambio el porcentaje de servicios simples sustituidos, con relación a la cantidad de servicios que integran el compuesto, además de variar la cantidad de opciones para reemplazar tal servicio, aumentando el número de grafos disponibles en el almacén de grafos compuestos. Los resultados a esta prueba se observan en la Figura 34 y reflejan un comportamiento creciente al igual que las dos pruebas anteriores.

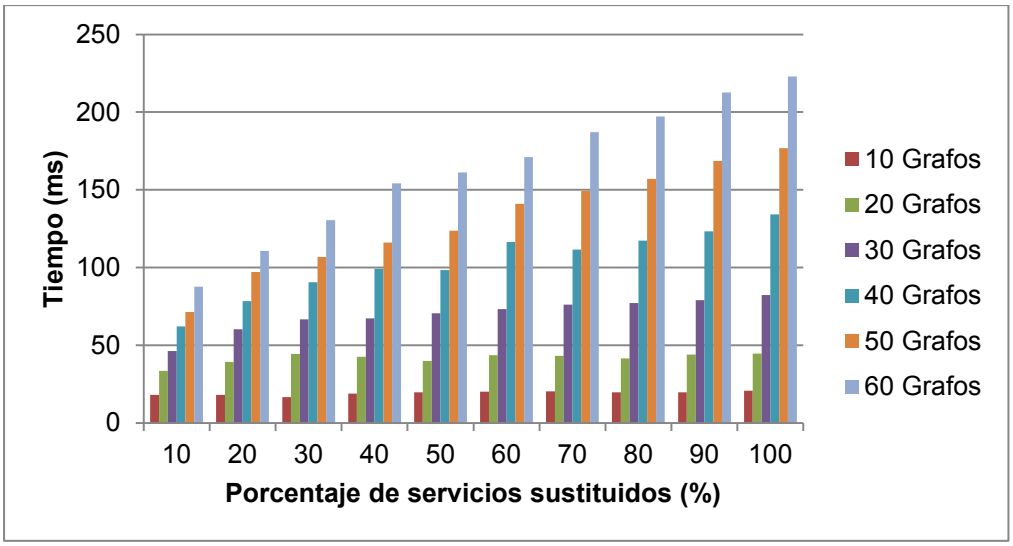


Figura 34. Tiempo de reconfiguración por sustitución de servicios simples

Los valores más pequeños, obtenidos para cada una de las pruebas del modulo dinámico debido a reconfiguraciones por parte del usuario experto, se resumen en las siguientes tablas:

Servicios	Tiempo (ms)
1	16.2
3	22.5
5	90.9
7	120.7
9	115.5
11	136.5

Tabla 28. Tiempo de reconfiguración por adición de servicios simples

Porcentaje Removido	Tiempo (ms)		
	100 Servicios	200 Servicios	300 Servicios
10	3,8	15,8	29,4
20	7,4	28,3	57
30	10,2	41	79,4
40	11,9	51	101,4
50	14,7	61,6	117,7

Tabla 29. Tiempo de reconfiguración por eliminación de servicios simples

Porcentaje Sustituido	Tiempo (ms)					
	10 Grafos	20 Grafos	30 Grafos	40 Grafos	50 Grafos	60 Grafos
10	18,1	33,5	46,2	62,1	71,4	87,7
20	18	39,3	60,2	78,3	97,1	110,7
30	16,6	44,5	66,7	90,5	106,8	130,4
40	18,9	42,5	67,3	99,3	116	154,2
50	19,7	39,9	70,6	98,4	123,6	161,2

Tabla 30. Tiempo de reconfiguración por sustitución de servicios simples

6.6. Implementación del prototipo

Para demostrar el correcto funcionamiento del algoritmo y la arquitectura para la composición de dinámica de servicios Web RESTful, se propone un caso de uso especifico que abarca tanto los conceptos del entorno Mobile 2.0, como los requerimientos impuestos por un usuario experto en lenguajes de programación o mecanismos de composición de servicios Web RESTful.

Servicios Disponibles	
Categoría	**Servicio**
Geolocalización	Google Geocoding
	Bing Geocoding
	MapQuest Geocoding
	GPS Móvil
Eventos musicales	Last FM Event
	Songkick Location Search
Mapas	Yahoo Map image
	Bing Maps
	Google Static Maps
	MapQuest
Notificaciones de Twitter (Este servicio se utiliza	Twitter Search

para el primer evento, por lo cual no hace parte de la composición inicial)	

Tabla 31. Servicios Disponibles para la Composición

El escenario parte de una serie de servicios Web RESTful que coinciden con la petición inicial de un usuario los cuales se especifica en Tabla 31. Esta petición está conformada por tres servicios Web, cuyas metas respectivamente y de forma ordenada son: i) geolocalización (un servicio cuya respuesta sean coordenadas a partir de una dirección o lugar) ii) eventos musicales (servicio Web cuya respuesta sean conciertos o eventos musicales cercanos a una ubicación dada en coordenadas) y iii) mapa (servicio Web cuya respuesta sea una imagen donde se desplieguen las ubicaciones de los lugares a partir de coordenadas), cabe la pena aclarar que todos los servicios son RESTful. La secuencia de ejecución para los servicios, esperado por el usuario experto se muestra en la Figura 35.

Para implementar el prototipo y ejecutar el escenario descrito, se desarrolla una interfaz Web para el dispositivo móvil Figura 36, con el fin de observar de forma amigable los resultados. Este proceso se lleva a cabo en un dispositivo móvil Samsung Galaxy S2 GT I9100, el cual cuenta con el navegador Web Opera Mobile con soporte para Java Script.

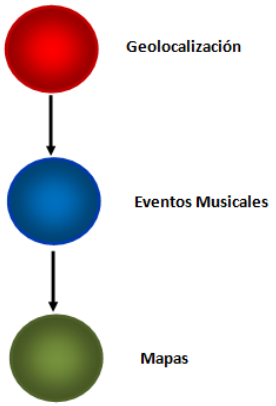


Figura 35. Petición inicial de usuario

Figura 36. Herramienta para la Composición de servicio Web RESTful

6.6.1. Composición de la petición inicial de usuario

Siguiendo los lineamientos propuestos por la Figura 35, se realiza el diseño de la petición formal en el dispositivo móvil, como se observa en la secuencia de imágenes de izquierda a derecha en la Figura 37.

El algoritmo es ejecutado, desde el momento en que se recibe la petición formal en el servidor (Figura 12), ejecutando el proceso de composición utilizando los servicios disponibles (gracias a descubrimiento) y las características del dispositivo y el contexto, cuyo resultado es la generación de diferentes grafos de forma ordenada que cumplen con los requisitos impuestos. Como por ejemplo, en la Figura 38 y la Tabla 32 se muestran los grafos obtenidos para esta implementación. En la Tabla 32 se puede observar además que desde la posición 17 en adelante, los grafos son considerados muy pobres en nivel de enlaces, por lo cual no son considerados para ser ofrecidos al usuario como composición inicial, aunque si pueden ser considerados, en caso de fallas o cambios en tiempo de ejecución. Además se puede observar que tanto la primera como la segunda posición tienen igual puntuación de enlaces, pero la preferencia dada por el usuario para google geocoding define la contienda.

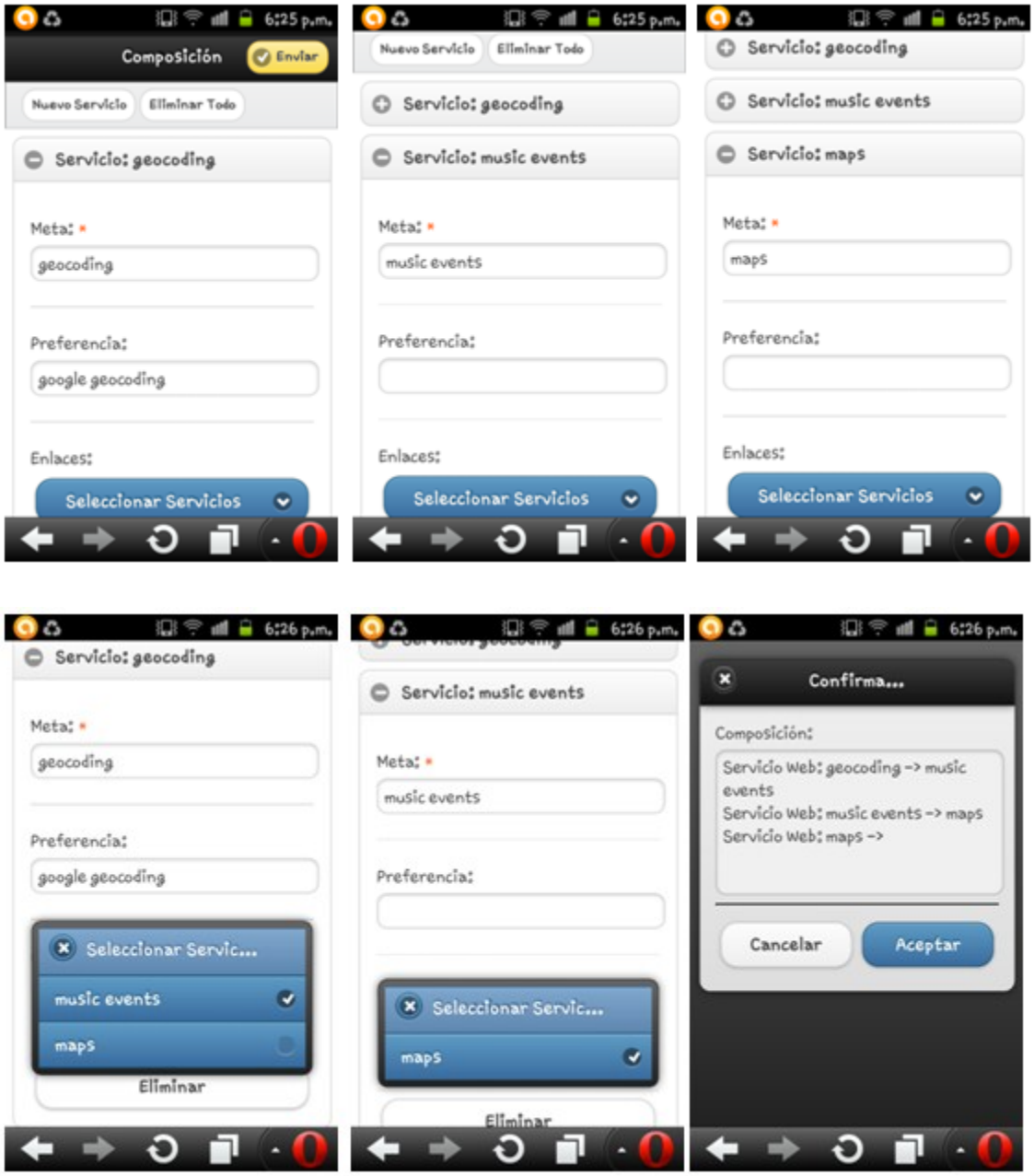


Figura 37. Diseño de la petición formal

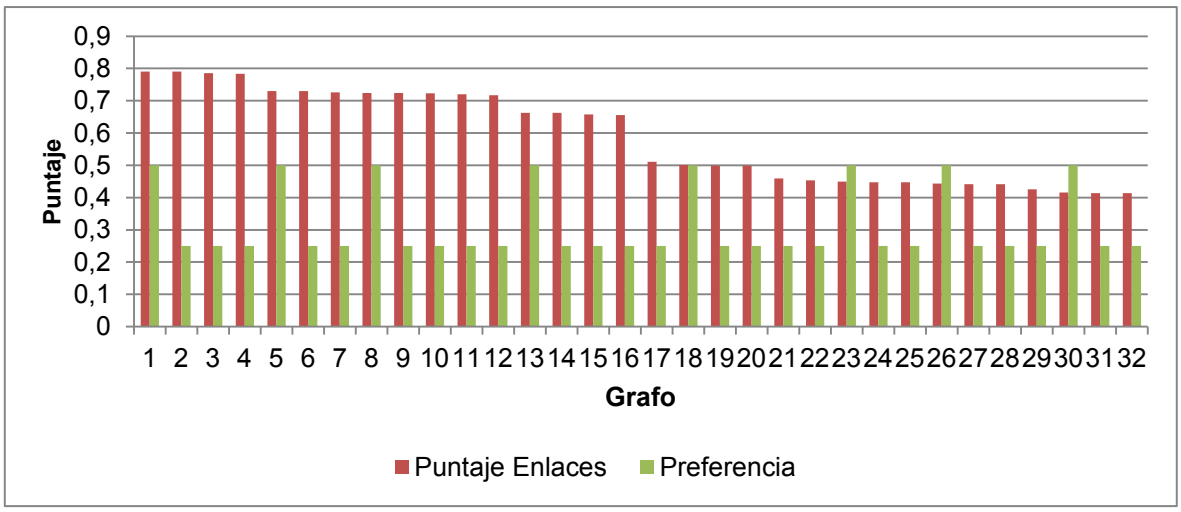


Figura 38. Puntuación de los grafos del prototipo

Pos	Enlaces	P. Enlaces	Pref.
1	**google:geocoding, songkick:events, bing:maps**	**0,790787169**	**0,5**
2	sensor:gps, songkick:events, bing:maps	0,790787169	0,25
3	bing:geocoding, songkick:events, bing:maps	0,786059896	0,25
4	mapQuest:geocoding, songkick:events, bing:maps	0,783787169	0,25
5	google:geocoding, songkick:events, google:maps	0,730478152	0,5
6	sensor:gps, songkick:events, google:maps	0,730478152	0,25
7	bing:geocoding, songkick:events, google:maps	0,72575088	0,25
8	google:geocoding, songkick:events, mapQuest:maps	0,724493304	0,5
9	sensor:gps, songkick:events, mapQuest:maps	0,724493304	0,25
10	mapQuest:geocoding, songkick:events, google:maps	0,723478152	0,25
11	bing:geocoding, songkick:events, mapQuest:maps	0,719766031	0,25
12	mapQuest:geocoding, songkick:events, mapQuest:maps	0,717493304	0,25
13	google:geocoding, songkick:events, yahoo:maps	0,662743305	0,5
14	sensor:gps, songkick:events, yahoo:maps	0,662743305	0,25
15	bing:geocoding, songkick:events, yahoo:maps	0,658016033	0,25
16	mapQuest:geocoding, songkick:events, yahoo:maps	0,655743305	0,25
17	sensor:gps, lastfm:events, bing:maps	0,510925021	0,25
18	google:geocoding, lastfm:events, bing:maps	0,501064174	0,5
19	mapQuest:geocoding, lastfm:events, bing:maps	0,498897507	0,25
20	bing:geocoding, lastfm:events, bing:maps	0,498897507	0,25
21	sensor:gps, lastfm:events, google:maps	0,459674853	0,25
22	sensor:gps, lastfm:events, mapQuest:maps	0,453690004	0,25
23	google:geocoding, lastfm:events, google:maps	0,449814005	0,5
24	mapQuest:geocoding, lastfm:events, google:maps	0,447647338	0,25

25	bing:geocoding, lastfm:events, google:maps	0,447647338	0,25
26	google:geocoding, lastfm:events, mapQuest:maps	0,443829156	0,5
27	bing:geocoding, lastfm:events, mapQuest:maps	0,441662489	0,25
28	mapQuest:geocoding, lastfm:events, mapQuest:maps	0,441662489	0,25
29	sensor:gps, lastfm:events, yahoo:maps	0,425339853	0,25
30	google:geocoding, lastfm:events, yahoo:maps	0,415479005	0,5
31	mapQuest:geocoding, lastfm:events, yahoo:maps	0,413312338	0,25
32	bing:geocoding, lastfm:events, yahoo:maps	0,413312338	0,25

Tabla 32. Puntuación de los grafos del prototipo

En el lado del cliente móvil, se obtuvieron la secuencia de imágenes que está comprendida por: Figura 39, corresponde al complemento de parámetros o entradas por parte del usuario y la Figura 40 corresponde a la representación obtenida de ejecutar el servicio compuesto, en donde se observa el mapa donde se llevara a cabo el evento musical.

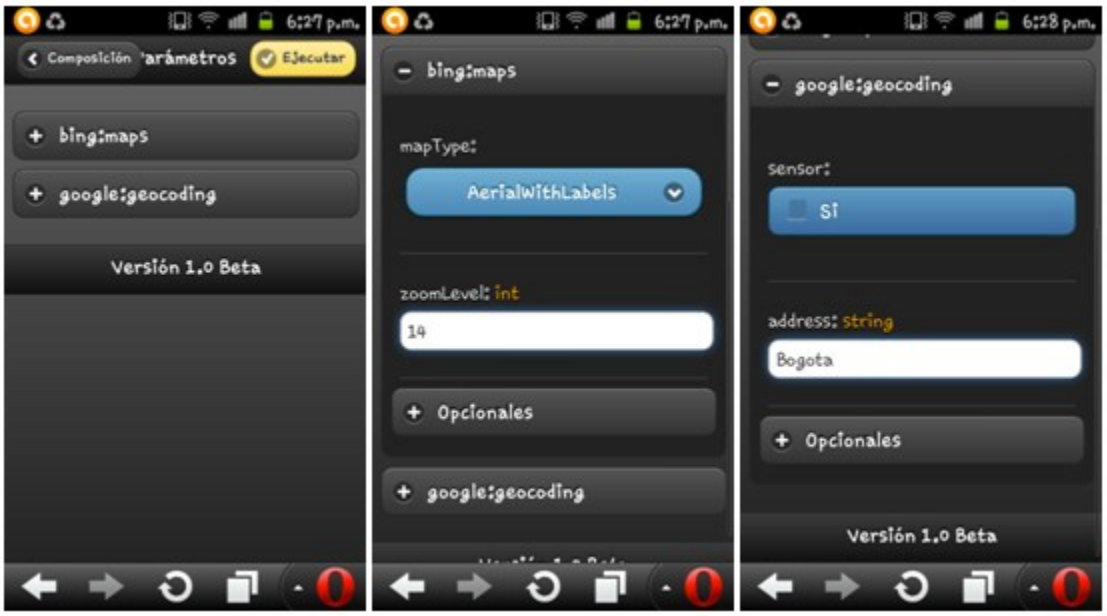


Figura 39. Ingreso de parámetros de usuario

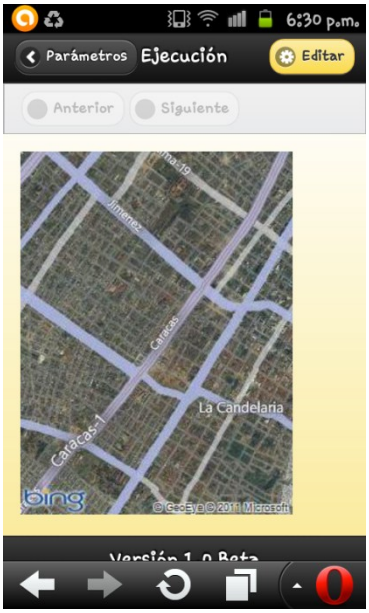


Figura 40. Representación del servicio compuesto

6.6.2. Sustitución del servicio Web RESTful de mapas.

En este paso se comprueba la disposición del algoritmo para sustituir un servicio por otro en tiempo de ejecución. Para ello se realiza la sustitución de dos servicios del compuesto (maps y geocoding).

La primera sustitución se evidencia en las imágenes de la Figura 41, donde se ha editado el servicio compuesto ejecutado, mediante el reemplazo del servicio de mapas de bing por el de google, pero el resultado muestra la misma ubicación.

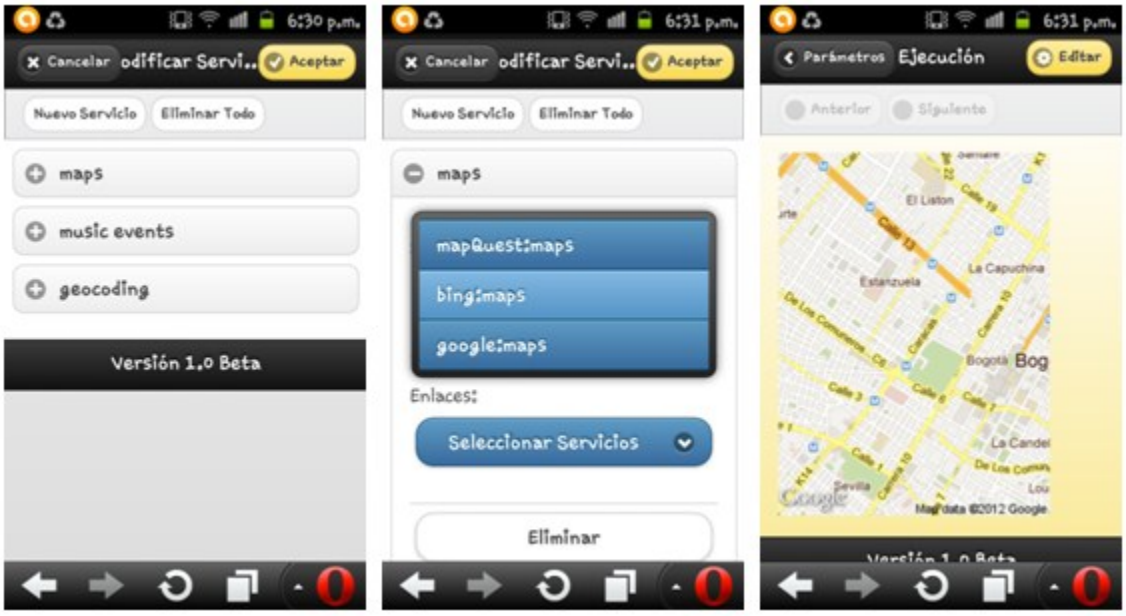


Figura 41. Sustitución del servicio de mapas

6.6.3. Adición del servicio Web RESTful de notificaciones de Twitter

Se adiciona el servicio de notificaciones de Twitter (previamente se encuentra disponible), justo después del encargado de los eventos musicales (songkick según la composición efectuada). Esta modificación tiene como fin, mostrar al usuario experto si existen notificaciones en Twitter acerca de un sitio o de algún evento arrojado por el servicio Web RESTful de eventos musicales. Estos resultados por el lado del móvil se observan a continuación.

Figura 42. Adición de un nuevo servicio Web RESTful

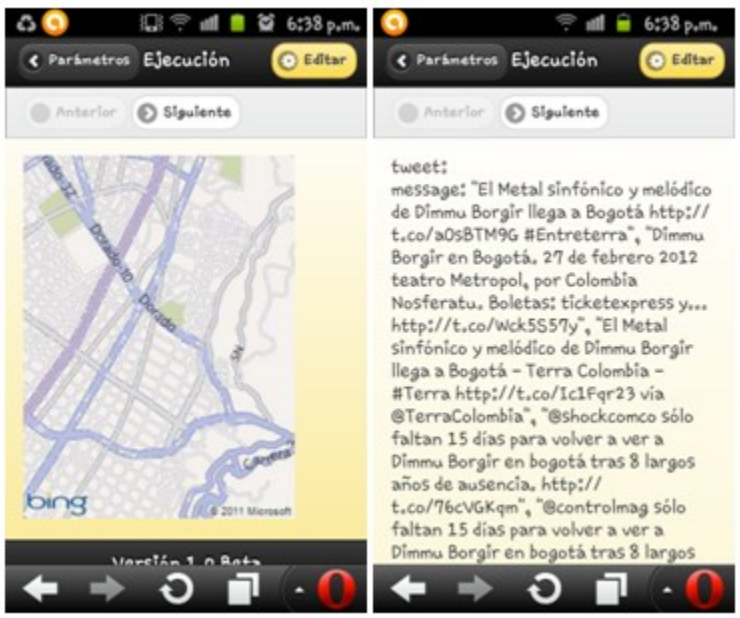


Figura 43. Representaciones del servicio compuesto modificado

La secuencia de imágenes es similar al proceso de composición y esto es debido a que al agregar u nuevo servicio, se debe establecer su posición dentro del grafo en ejecución, mediante la fijación de los enlaces e ingresar los parámetros del mismo. Además como el nuevo servicio compuesto tiene dos servicios Web finales, se obtienen dos representaciones para el usuario.

6.6.4. Falla y eliminación del servicio Web RESTful de eventos musicales

Para este evento se considera inicialmente la sustitución del servicio de google geocoding, por el sensor GPS del dispositivo móvil (una de las características que lo hace idóneo para un entorno Mobile 2.0), como se observa en la Figura 44**,** pero debido a que ningún servicio de eventos musicales es capaz de obtener resultados en la ubicación de pruebas (Popayán-Cauca), la composición no puede reconfigurarse (por causas ajenas al sistema) y genera un error (Figura 45). En la Figura 46 se aprecia un fragmento de la respuesta del servidor que contiene el proceso de reconfiguración de la falla del servicio al obtener estados 500 y 503 respectivamente por parte de los dos servicios de eventos musicales disponibles aunque sin éxito, por lo tanto el usuario decide eliminar el servicio de eventos musicales y obtener el mapa de su ubicación, en base a los dos servicios restantes (sensor GPS y google maps) como se muestra en la Figura 47.

GPS es utilizado mediante un código en Java Script que se ejecuta en el lado del cliente, el cual es tratado como un servicio Web, ya que posee una descripción que se asemeja a la utilizada por los demás servicios Web RESTful.

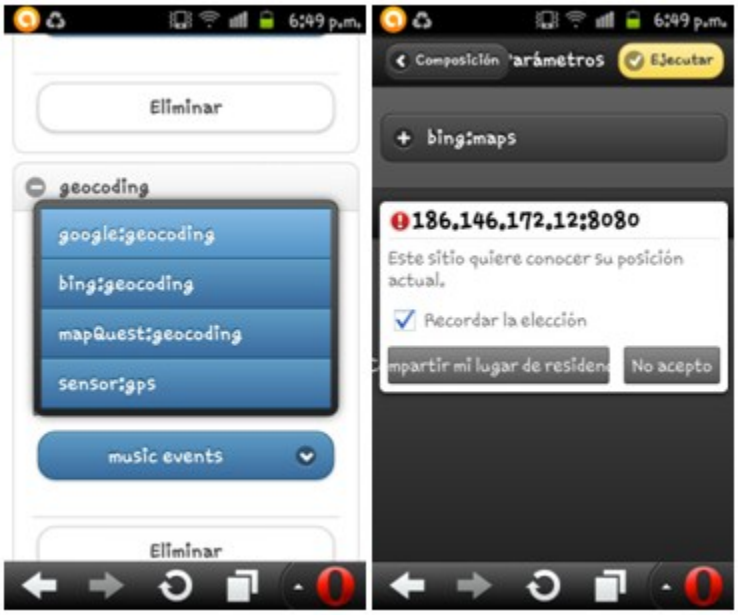


Figura 44. Sustitución de google geocoding por sensor GPS

Figura 45. Error de reconfiguración del servicio compuesto

```
9 > GET
http://api.songkick.com/api/3.0/events.b25b959554ed76058ac220b7b2e0a026?location=geo%3A2.4529995%2C-
76.63055&apikey=1gzXO1QGt9pIOY2O
9 > Accept: application/json
|#]
[#|2012-02-12T06:49:00.162-
0500|INFO|glassfish3.1.1|com.sun.jersey.api.client.filter.LoggingFilter|_ThreadID=18;_ThreadName=Thread-
2;|10 * Client in-bound response
9 < 500
9 < Status: 500
9 < Date: Sun, 12 Feb 2012 17:24:58 GMT
9 < Vary: Accept-Encoding
9 < Content-Length: 1105
9 < Content-Type: text/html; charset=utf-8
9 < Connection: close
9 < Server: nginx/0.8.34 + Phusion Passenger 2.2.11.1 (mod_rails/mod_rack)
9 < X-Powered-By: Phusion Passenger (mod_rails/mod_rack) 2.2.11.1
9 < Cache-Control: no-cache
9 <
//Se realizo una petición a Songkick pero no se obtuvo ninguna respuesta válida para la ubicación proveniente del
sensor GPS del dispositivo móvil. Por lo tanto se busca otro servicio para los eventos que arroje resultados
positivos.
```

```
10 > GET http://ws.audioscrobbler.com/2.0/?method=geo.getevents&location=&lat=2.4529995&long=-
76.63055&distance=&limit=2&page=1&api_key=b25b959554ed76058ac220b7b2e0a026
10 > Accept: text/xml
|#]
[#|2012-02-12T06:49:58.633-
0500|INFO|glassfish3.1.1|com.sun.jersey.api.client.filter.LoggingFilter|_ThreadID=18;_ThreadName=Thread-
2;|9 * Client in-bound response
10 < 503
10 < Age: 0
10 < Expires: Sun, 12 Feb 2012 17:25:08 GMT
10 < Access-Control-Max-Age: 86400
10 < Access-Control-Allow-Methods: POST, GET, OPTIONS
10 < Connection: close
10 < Server: Apache/2.2.17 (Unix)
10 < Cache-Control: max-age=10
10 < X-Varnish: 1977272927
10 < X-Web-Node: www173
10 < Date: Sun, 12 Feb 2012 17:24:58 GMT
10 < Access-Control-Allow-Origin: *
10 < Via: 1.1 varnish
10 < Content-Type: text/xml; charset=utf-8;
10 <
 //Se realizo el intento con Last.FM pero no se obtuvieron datos para esa ubicación, por lo
tanto al no existir mas servicios que cumplan con este propósito el servicio no puede ser
reconfigurado y genera un error.
```

Figura 46. Registros de reconfiguración frente a falla del servicio simple de eventos musicales

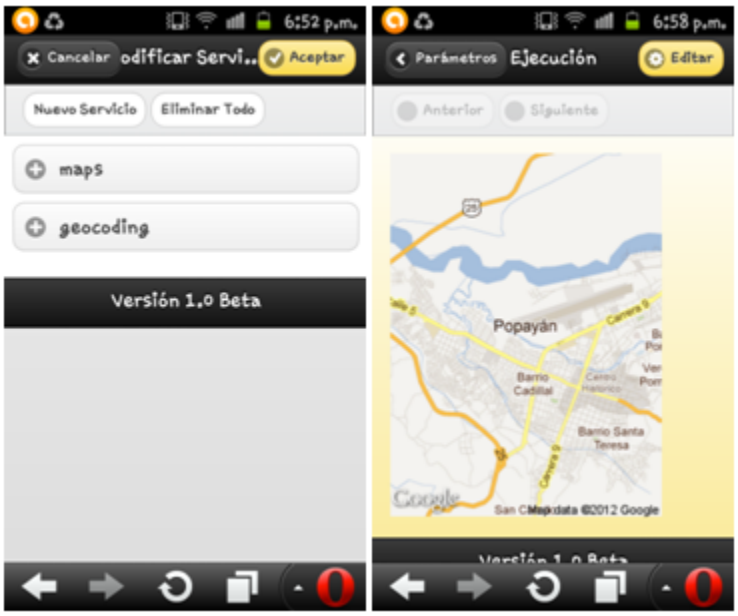


Figura 47. Representación del servicio Web reconfigurado

6.6.5. Cambio en el nivel de la señal de la Red

El otro tipo de falla considerado para este sistema, independiente de la naturaleza del mismo, son los comportamientos de red que afectan el formato de las respuestas del servicio compuesto, por lo tanto, con el fin de tener una representación que se adapte al contexto (velocidad de la red) donde se encuentra el dispositivo móvil, se realiza una reconfiguración en tiempo de ejecución que no afecta la estructura del servicio compuesto como el caso anterior, si no su formato de representación.

El proceso de contingente del algoritmo frente a este evento, recae en reemplazar o sustituir el formato de representación de la respuesta obtenida, por uno más apropiado a las características de red, es decir, si un servicio Web RESTful puede generar una imagen en PNG o GIF, esta varia dinámicamente en relación a las variaciones de la tasa de transferencia de datos de la red, por lo tanto, si se presenta una red con buenas prestaciones, PNG es el formato más adecuado, en caso contrario GIF será la mejor opción.

Capitulo 7

Conclusiones

En este capítulo se presentan los aportes realizados, las conclusiones que se obtuvieron al plantear, diseñar, adaptar, implementar y analizar el mecanismo de composición dinámica de servicios Web RESTful y las necesidades que surgen en tales procesos, que se convierten en un foco de atención para posibles trabajos futuros. Todos estos aspectos se tienen en cuenta bajo diferentes puntos de vista, tanto el usuario, tecnológico, académico, entre otros.

7.1. Aportes

El algoritmo para la composición de servicios Web RESTful, surge de la adaptación de investigaciones y trabajos previamente realizados por diferentes organizaciones y grupos, en los cuales se consideran aspectos como, procesamiento de lenguaje natural, descubrimiento de servicios, gestión de fallas en tiempo de ejecución, gestión de ontologías (análisis semántico), repositorios de servicios, enlaces semánticos o sintácticos, entre otros, cuyas estructuras fueron manipuladas con el fin de procesar servicios Web RESTful. No todos los bloques fueron considerados, debido a factores temporales y/o de alcance (fuera de los objetivos planteados), como es el caso del procesamiento de lenguaje natural o enlaces semánticos.

De la misma forma se realizó la adaptación de una arquitectura, que da soporte a la ejecución de servicios compuestos en un entorno móvil, teniendo en cuenta los aspectos más relevantes en este medio, como lo son algunas de las características de los dispositivos y las velocidades o tasas de transferencia de datos de la red, además de considerar los cambios dinámicos que se dan a causa de errores en los servicios Web RESTful y las reconfiguraciones hechas por peticiones directas de usuario.

Los aportes generados por este proyecto según su clasificación, son los siguientes:

Campo	Aportes
Académico e Investigativo	• Un algoritmo para la composición dinámica de servicios Web RESTful, que se constituye en referencia para trabajos futuros relacionados con el tema. • Una arquitectura para la ejecución de servicios compuestos Web RESTful. • Un concepto de composición dinámica, basada en la reconfiguración de un servicio compuesto en tiempo de ejecución, teniendo en cuenta los requerimientos de usuario. • Un modulo STE (Sistema para la Transición de Estados), que realiza la estructura interna de un servicio Web RESTful automáticamente. • Un prototipo para la implementación del algoritmo para la composición dinámica de servicios Web RESTful.

Para el Usuario	• Un mecanismo para la composición de servicios Web RESTful, accesible mediante un navegador Web, con el cual se pueden acceder a servicios ajustados a una petición de usuario. • Una nueva forma de obtener servicios a la medida, que le da a los usuarios la libertad de poder acceder a servicios que cumplen con sus necesidades utilizando el potencial existente en la Web. • La posibilidad de elegir entre diferentes opciones se servicios Web, que le da al usuario la sensación de tener el control sobre sus actividades en un dispositivo móvil. • La ejecución de servicios Web RESTful compuestos, que modifiquen su estructura o sus representaciones de forma dinámica, ya sea por peticiones del usuario o por fallos en los servicios ejecutados. • Una interfaz para la interacción con el usuario, que le permita la manipulación de los servicios de manera directa
Para el desarrollador	• Un mecanismo que les permite a los desarrolladores de servicios Web RESTful, aprovechar las ventajas que ofrece la Web y los servicios RESTful. • Interoperabilidad de contenido entre dispositivos móviles, ya que todo se soporta en la Web y en servicios RESTful, que permiten la flexibilidad mediante diferentes tipos de representaciones. • Un elemento importante que puede generar una idea de negocio que brinde grandes beneficios económicos.

Tabla 33. Aportes

7.2. Conclusiones

- El concepto general de ambiente móvil, plantea como principal característica la movilidad, que se constiuye en una clara ventaja frente al entorno estatico, y que permite el desarrollo de diferentes aspectos, tales como dispositivos personales, portabilidad de tecnología software y hardware, creación y consumo de contenidos en tiempo de inspiración, entre otros, generando un entorno altamente flexible, adaptativo y escalable. Otra premisa importante que ha sido concebida en los últimos años y se ha convertido en un objetivo por defecto para el entorno móvil, es la ubicuidad y la posibilidad de tener todo un conjunto multifuncional en la palma de la mano, que combinado con el concepto original de movilidad permite que la tecnología sea una extensión física y abstracta de nuestras vidas. Es por esto que abarcar los problemas de dicho entorno se constituyen en una fuente de progreso para la tecnología y por ende para la humanidad.

- Los servicios Web RESTful, presentan características idóneas para la implementación de mecanismos de composición en el ámbito móvil, ya que se adaptan a las capacidades limitadas de los dispositivos y las redes inalámbricas, para que los usuarios puedan acceder a la Web, mediante un cliente navegador que cumpla con ciertos requerimientos. Pero aunque se presentan dichas ventajas, es importante aclarar que el ambiente móvil a pesar de sus diversas mejorías en los últimos años, presenta debilidades en algunos aspectos, tales como interoperabilidad entre plataformas, capacidades físicas reducidas en comparación con el entorno estático,

variedad de versiones de un mismo sistema operativo, etc. Lo cual imposibilitó el desarrollo del algoritmo en el lado del móvil, trasladándose al servidor y aprovechando así las ventajas que brinda el protocolo HTTP y el modelo cliente servidor.

- En cuanto a la eficiencia del sistema, ésta se ve afectada principalmente por los tiempos que tardan los procesos realizados por el modulo MEC, debido a las comparaciones entre parámetros hecha por el analizar sintáctico, más exactamente cuando en el proceso entra a formar parte WordNet. La combinación de parámetros sin una terminología estándar y su débil relación con parámetros de otros servicios Web RESTful, ocasionan demoras significativas. Otro aspecto que aporta retardos en el tiempo de respuesta a este modulo, es la cantidad de parámetros que tiene cada servicio simple, y su heterogeneidad entre servicios, esto se debe a que cada desarrollador le añade un valor agregado sus servicios, independientemente que pertenezcan a una categoría en particular.

- A diferencia de los servicios Web tradicionales basados en operaciones, los servicios Web RESTful están orientados a recursos fundamentados en información y en los posibles enlaces que pueden establecerse entre ellos. Esto impidió la aplicación directa de algoritmos existentes para la composición de servicios tradicionales y requirió un tratamiento diferente. Para esto, los servicios RESTful fueron modelados como Sistemas de Transiciones de Estados, permitiendo aprovechar técnicas existentes en este campo, para encontrar una solución de composición y ejecución a nivel de los servicios individuales, los cuales se convierten en la base de los restantes procesos de composición analizados.

- La ejecución de servicios Web RESTful compuestos que consideran características del entorno Mobile 2.0, como las capacidades brindadas por los dispositivos móviles (por ejemplo el sensor GPS), requiere de una interacción más compleja entre los dispositivos y el servidor. Esto se ve reflejado por ejemplo, en el hecho de que la ejecución del servicio compuesto debe realizarse en diferentes etapas según se requiera: unas en el lado del cliente, como la consulta de la posición del usuario y otras en el lado del servidor, relacionadas directamente con los servicios Web. La interacción descrita requiere la interrupción y posterior continuación de los procesos de ejecución de un servicio compuesto en un servidor.

- Todo mecanismo o proceso tecnológico, generado para suplir una necesidad, debe ser sometido a evaluación, con el fin de establecer el grado de aceptación que puede llegar a alcanzar, valorando principalmente, si cumple con el objetivo general para el que fue concebido y si encaja con los estándares y propuestas contemporáneas a la idea. Es por esto que los resultado obtenidos de las diferentes pruebas y evaluaciones realizadas al sistema para la composición dinámica de servicios Web RESTful, establecen una métrica que refleja su nivel frente a proyectos similares y modos de obtener composiciones en la actualidad, posicionando el algoritmo y la arquitectura desarrollados a lo largo de este documento, como herramientas apropiadas, que mejoran el proceso actual, y abre un nuevo camino en el campo de los servicios Web RESTful.

- Hoy en día la creación de servicios Web RESTful, se ha tornado en un objetivo principal para las empresas tales como Google, Facebook, Microsoft, entre otras, cuyo

fin es satisfacer necesidades especificas del mercado móvil o brindar nuevas experiencias a los usuarios, por lo tanto el uso de mecanismos de composición de servicios, como el planteado en este trabajo de grado, permite el acceso a dichos servicios e integrarlos para que cumplan una función particular de cada usuario.

7.3. Trabajos Futuros

- El mecanismo para la composición de servicios Web RESTful, exige grandes capacidades de procesamiento a la hora de aplicar una lógica, para realizar enlaces entre servicios simples, es por esto que en este proyecto se implemento un análisis sintáctico el cual brinda eficiencia en tiempos de respuesta en ejecución, pero posee debilidades en cuanto a calidad entre enlaces, que son inherentes al proceso mismo de composición. Por lo tanto con la incursión de la tecnología en el campo de la Web 3.0 y las mejorías en las capacidades hardware y software de los dispositivos, es claro que surge la necesidad de aumentar los niveles de calidad del servicio compuesto en tiempo de ejecución, mediante el análisis semántico, lo cual permite generar enlaces más fuertes, que combaten la interoperabilidad entre servicios Web RESTful de diferentes desarrolladores.

- Diseñar o adaptar un lenguaje para la descripción de servicios basados en REST, que integre todas sus características, tales como manejo de recursos y representaciones, enlaces entre los recursos, gestión de métodos CRUD, etc. Debido a que tecnologías existentes en la actualidad solo toman parte de las características, o intentan describir estos servicios como servicios Web tradicionales.

- El algoritmo aquí desarrollado para la composición, toma como punto de partida una petición inicial generada por un usuario experto en el área de los lenguajes para la composición de servicios Web, por lo cual otro punto de interés que se convierte en idea de trabajo a seguir, es el desarrollo de un modulo de análisis de lenguaje natural o de lenguaje humano para generar, lógicas de composición de servicios, tomando en cuenta la petición de usuarios que no posean un alto grado de conocimiento sobre la tecnología usada para dicho fin. Este módulo debe considerar las características de los servicios Web RESTful enfocados en el contexto de la Mobile 2.0.

- Desde un punto de vista más social que tecnológico, es promover la creación y desarrollo de servicios Web RESTful en el ámbito local y nacional, que cubra necesidades en el campo económico, cultural, educacional, de la salud, etc. Utilizando el mecanismo obtenido mediante este trabajo de grado, para componer servicios de cualquier índole.

Bibliografía

[1] B. van Schewick, "Internet Architecture and Innovation," *MIT Press Books,* vol. 1, 2010.
[2] T. Oreilly, "What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software," *Communications & Strategies, No. 1, p. 17, First Quarter 2007*.
[3] T. Pollet, "How the Web Radically Transforms Communication Networks," presented at the Proceedings of the 2007 Workshop on INnovative SERvice Technologies, Rome, Italy, 2007.
[4] ProgrammableWeb.com. (2010, Sept. 5, 2010). *API Directory* [Online]. Available: http://www.programmableweb.com/apis
[5] A. Barros, M. Dumas, and P. Bruza, "The Move to Web Service Ecosystems," *BPTrends Newsletter,* vol. 3, 2005.
[6] M. Sauter, "Mobile Web 2.0, Applications and Owners," in *Beyond 3G - Bringing Networks, Terminals and the Web Together: LTE, WiMAX, IMS, 4G Devices and the Mobile Web 2.0*, ed Nortel, Germany: John Wiley & Sons, 2009.
[7] P. Farley and M. Capp, "Mobile Web Services," *BT Technology Journal,* vol. 23, pp. 202-213, 2005.
[8] S. Cho, H. Kim, D. Jung, and H. Park, "Dynamic Mashup Platform for Mobile Web Applications," 2009.
[9] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web Services Architecture," *W3C Working Group Note,* vol. 11, pp. 2005-1, 2004.
[10] M. Tian, T. Voigt, T. Naumowicz, H. Ritter, and J. Schiller, "Performance Considerations for Mobile Web Services," *Computer Communications,* vol. 27, pp. 1097-1105, 2004.
[11] H. Hamad, M. Saad, and R. Abed, "Performance Evaluation of RESTful Web Services for Mobile Devices," 2009.
[12] L. Richardson and S. Ruby, *RESTful web services: web services for the real world*: O'Reilly Media, 2007.
[13] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. big'web services: making the right architectural decision," 2008, pp. 805-814.
[14] N. Blum and F. Schreiner, "Smart Bit Pipes: Open APIs and their Role in Emerging SOA SDPs for Converging Networks," ed: Fraunhofer Institute FOKUS, 2010.
[15] J. Brandt and H. Kuklisnki, "Mobile Web 2.0: Marco Teórico y Tendencias de Desarrollo," 2008.
[16] M. Stanley, "Anual Report: Mobile Internet Report," December 15, 2009 2009.
[17] D. Guinard, M. Fischer, and V. Trifa, "Sharing Using Social Networks in a Composable Web of Things," 2010.
[18] N. Neves and W. K. Fuchs, "Adaptive recovery for mobile environments," *Communications of the ACM,* vol. 40, pp. 68-74, 1997.
[19] B. Fling, "Mobile 2.0," in *Mobile Design and Development*, O'Reilly, Ed., 1 ed, 2009.
[20] S. P. Crespo. (2007, May 25, 2011). Mobile 2.0, el segundo intento de llevar la web al móvil. Available: http://sociedadinformacion.fundacion.telefonica.com/DYC/SHI/seccion=1188&idioma=es_ES&id=2009100116310010&activo=4.do?elem=4177

[21] Qualcomm. (2007, Evolution of Wireless Applications and Services. Available: http://www.qualcomm.com/media/documents/evolution-wireless-applications-and-services-whitepaper
[22] H. Cerón and G. Vela, "Arquitectura de Referencia para Sitios de Redes Sociales en Ambientes Móviles," Ingenieria Electrónica y Telecomunicaciones, Dpto. Telemática, Universidad del Cauca, Popayán, 2009.
[23] H. P. Kuklinski, J. Brandt, and J. P. Puerta, "Mobile Web 2.0. Theoretical-technical framework and developing trends," *International Journal of Interactive Mobile Technologies (iJIM),* vol. 2, pp. pp. 54-61, 2008.
[24] D. Appelquist. (2006, June 15, 2011). What is “Mobile 2.0″.
[25] K. Chang, C. Chen, J. Chen, and H. Chao, "Challenges to Next Generation Services in IP Multimedia Subsystem," 2010.
[26] F. Martire, G. Bartolomeo, C. Trubiani, and S. Salsano, "Location Based Services Architecture for Simple Mobile Services," *Information Society Technologies, Italy,* August 2008.
[27] T. Ryhänen, M. Uusitalo, O. Ikkala, and A. Kärkkäinen, *Nanotechnologies for Future Mobile Devices*: Cambridge Univ Pr, 2010.
[28] J. Wilson, "3G to Web 2.0? Can mobile telephony become an architecture of participation?," *Convergence: The International Journal of Research into New Media Technologies,* vol. 12, p. 229, 2006.
[29] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible Markup Language (XML) 1.0," 5th Edn ed: W3C Recommendation, 2008.
[30] D. Crockford, "The application/json media type for javascript object notation (json)," 2006.
[31] J. Webber, "REST in Practice," *Software Architecture,* pp. 7-7, 2010.
[32] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Citeseer, 2000.
[33] J. Webber, "HATEOAS-The Confusing Bit from REST," ed.
[34] H. Overdick, "Towards resource-oriented BPEL," *Emerging Web Services Technology, Volume II,* pp. 129-140, 2008.
[35] C. Pautasso, "RESTful Web Service Composition with BPEL for REST," *Data & Knowledge Engineering,* vol. 68, pp. 851-866, 2009.
[36] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," *W3C Working Draft,* vol. 26, 2004.
[37] M. Hadley, "Web Application Description Language (WADL)," ed: Sun Microsystems Inc., 2009.
[38] J. Kopecký, K. Gomadam, and T. Vitvar, "hrests: An html microformat for describing restful web services," 2008, pp. 619-625.
[39] J. Lathem, K. Gomadam, and A. Sheth, "Sa-rest and (s) mashups: Adding semantics to restful services," 2007.
[40] R. Alarcon and E. Wilde, "Linking Data from RESTful Services."
[41] C. Pautasso, "BPEL for REST," *Business Process Management,* pp. 278-293, 2008.
[42] F. Curbera, M. Duftler, R. Khalaf, and D. Lovell, "Bite: Workflow Composition for the Web," ed, 2007, pp. 94-106.
[43] J. Domingue and D. Fensel, "SOA4All, enabling the SOA revolution on a world wide scale," 2008, pp. 530-537.

[44] F. Schnabel, M. Born, L. Xu, R. Gonzales, F. Lecue, N. Mehandjiev, and T. Pariente, "D6.3.1. First Specification of Lightweight Process Modelling Language," ed: SOA4ALL, 2009.
[45] A. Duke, S. Stincic, J. Davies, G. Álvaro Rey, C. Pedrinaci, M. Maleshkova, J. Domingue, D. Liu, F. Lecue, and N. Mehandjiev, "Telecommunication mashups using RESTful services," *Towards a Service-Based Internet,* pp. 124-135, 2010.
[46] S. Dustdar and W. Schreiner, "A Survey on Web Services Composition," *International Journal of Web and Grid Services,* vol. 1, pp. 1-30, 2005.
[47] M. Anastasopoulos, H. Klus, J. Koch, D. Niebuhr, and E. Werkman, "DoAmI-a middleware platform facilitating (re-) configuration in ubiquitous systems," 2006.
[48] F. C. Pop, M. Cremene, J. Y. Tigli, S. Lavirotte, M. Riveill, and M. Vaida, "Natural Language based On-demand Service Composition," 2010.
[49] F. AlShahwan and K. Moessner, "Providing SOAP Web Services and RESTful Web Services from Mobile Hosts," 2010, pp. 174-179.
[50] F. Lécué, E. Silva, and L. F. Pires, "A framework for dynamic web services composition," *Emerging Web Services Technology, Volume II,* pp. 59-75, 2008.
[51] K. Boumhamdi and Z. Jarir, "Yet another approach for dynamic web service composition," pp. 1-5.
[52] L. Passani. (August 15, 2011). *WURFL*. Available: http://wurfl.sourceforge.net/
[53] R. Johnson, *Expert one-on-one J2EE Design and Development*: Wrox, 2003.
[54] K. Fujii and T. Suda, "Component service model with semantics (cosmos): A new component model for dynamic service composition," 2004.
[55] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory & Practice*: Morgan Kaufmann Publishers, 2004.
[56] F. Lecue and A. Leger, "Semantic web service composition based on a closed world assumption," 2006, pp. 233-242.
[57] F. Lécué, Y. Gorronogoitia, R. Gonzalez, M. Radzimski, and M. Villa, "SOA4All: An innovative integrated approach to services composition," 2010, pp. 58-67.
[58] K. Fujii and T. Suda, "Semantics-based dynamic service composition," *Selected Areas in Communications, IEEE Journal on,* vol. 23, pp. 2361-2372, 2005.
[59] F. H. Khan, M. Y. Javed, S. Bashir, A. Khan, and M. S. H. Khiyal, "QoS Based Dynamic Web Services Composition & Execution," *International Journal of Computer Science and Information Security,* vol. 7, February 2010 2010.
[60] X. Liu, Y. Hui, W. Sun, and H. Liang, "Towards service composition based on mashup," 2007, pp. 332-339.
[61] R. M. Miguel, "Desarrollo web orientado a dispositivos móviles," 2008.
[62] C. Pautasso, "Title," unpublished|.
[63] F. O. Martínez, G. A. Uribe, and F. L. Mosquera, "OneWeb: plataforma de adaptación de contenidos web basada en las recomendaciones del W3C Mobile Web Initiative," *INGENIERÍA E INVESTIGACIÓN,* vol. 31, pp. 117-126, 2011.
[64] C. Kiss, "Composite capability/preference profiles (cc/pp): Structure and vocabularies 2.0," *W3C Working Draft,* vol. 8, 2006.
[65] A. C. Augé and J. L. F. Riera. (2004, June 21, 2004). HACIA UNA WEB INDEPENDIENTE DEL DISPOSITIVO MEDIANTE CC/PP.

Anexo A

Métodos HTTP para el sistema

Las siguientes tablas muestran los métodos HTTP más usados en la implementación de servicios basados en REST.

Método:	**GET**
Operación común:	Leer
Operación CRUD:	READ
Seguro:	Si
Ídem-potente:	Si
Descripción:	Retorna una representación del estado de un recurso. Debido a que es un método seguro, las respuestas obtenidas pueden ser almacenadas en un caché, a menos que se especifique lo contrario por medio de metadatos en los mensajes HTTP.

Método:	POST
Operación común:	Crear
Operación CRUD:	CREATE
Seguro:	No
Ídem-potente:	No
Descripción:	Por convención, crea un recurso subordinado al recurso identificado en la petición que contiene este método. Aunque, de acuerdo con su definición formal permite implementar operaciones cuya semántica no coincide con ninguno de los significados de los otros métodos.

Método:	PUT
Operación común:	Reemplazar o Crear
Operación CRUD:	UPDATE
Seguro:	No
Ídem-potente:	Si
Descripción:	Reemplaza el estado de un recurso con la información enviada en el mensaje de la petición, únicamente si la URI referenciada corresponde a un recurso existente. En cualquier otro caso, el servidor de origen puede crear un nuevo recurso con el estado de la representación enviada. Debido a que el método es ídem-potente, una petición de este tipo puede repetirse sin temor a causar efectos indeseados.

Método:	DELETE
Operación común:	Eliminar
Operación CRUD:	DELETE
Seguro:	No
Ídem-potente:	Si
Descripción:	Elimina un recurso del servidor de origen. La realización de este tipo de petición no implica la eliminación efectiva del recurso en el servidor. Debido a que el método es ídem-potente, una petición de este tipo puede repetirse sin temor a causar efectos indeseados.

Método:	PATCH
Operación común:	Actualizar
Operación CRUD:	UPDATE
Seguro:	No
Ídem-potente:	No
Descripción:	Permite la realización de actualizaciones parciales al estado de un recurso determinado. El documento enviado en el cuerpo del mensaje debe indicar cuáles son los cambios a realizar al recurso. HTTP no impone ningún tipo de formato para el documento de actualización. Debido a que el método se aceptó recientemente como extensión al protocolo HTTP, su utilización aún no se ha extendido significativamente.

Método:	HEAD
Seguro:	Si
Ídem-potente:	Si
Descripción:	Equivalente al método GET excepto en que la respuesta no contiene una representación del recurso solicitado. Se utiliza para obtener información relacionada de un recurso sin el costo de enviar una representación.

Método:	OPTIONS
Seguro:	Si
Ídem-potente:	Si
Descripción:	Envía información sobre las capacidades de un servidor o recurso, principalmente, en cuanto a métodos que sean actualmente soportados y que un cliente puede realizar.

Anexo B

Herramientas para la detección de dispositivos

B.1. UAProf 2.0 [61]

La especificación de UAProf (*User Agent Profile*), propuesta por OMA (Open Mobile Alliance. Anteriormente conocida como WAP Fórum [62]), se encarga de capturar información de dispositivos móviles inalámbricos, la cual es utilizada por los desarrolladores de contenido, para proveer aplicaciones y/o servicios que se adapten sus características especificas.

UAProf brinda un marco, para que los fabricantes de teléfonos móviles, generen especificaciones particulares para cada uno de los dispositivos que producen, mediante el uso de XML y CC/PP (*Composite Capabilities/ Preferences Profile*), el cual brinda un marco mucho mayor que UAProf. Aunque hoy en día existe un gran inconveniente para el uso de UAProf, el cual es la heterogeneidad de los dispositivos que existen y por lo tanto de los conceptos definidos para CC/PP, generando consigo el uso de términos diferentes para conceptos iguales, o la exclusión de algunos de ellos debido a que los fabricantes lanzan especificaciones aisladas de otros para sus propios dispositivos.

UAProf utiliza las cabeceras HTTP, las cual contiene las características del dispositivo descritas en las especificaciones del CC/PP. Los inconvenientes que se dan debido a la heterogeneidad son[63]:

- No todos los dispositivos tienen un UAProf definido.
- Puede causar retardos en la navegación.
- No existe un estándar para los datos en cada UAProf.
- Las cabeceras UAProf pueden planearse erróneamente.
- Algunos fabricantes crean estos perfiles para un grupo de dispositivos y no para cada producto en particular.

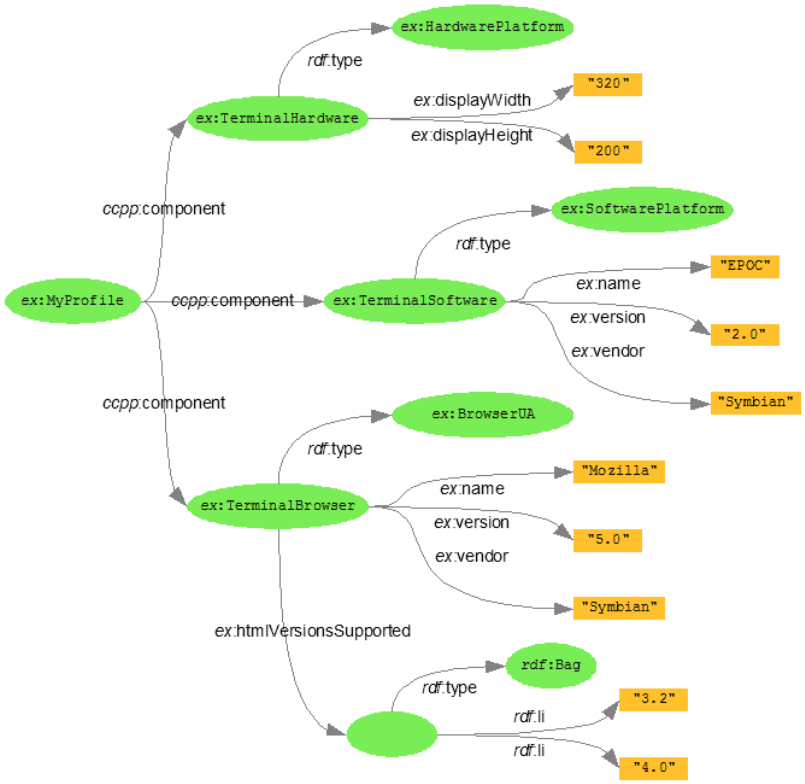
B.2. CC/PP 2.0 [64]

Como se menciono anteriormente CC/PP, brinda un marco mucho más grande que UAProf, ya que no solo permite expresar la funcionalidad de los dispositivos, sino también las preferencias de usuario, que al igual que UAProf, se basa en rdf debido a que los perfiles de agentes de usuario se intercambian entre los agentes de usuario y los creadores de datos de recursos, de manera semántica.

Un perfil de CC/PP contiene los nombres de los atributos y valores asociados que son utilizados por un servidor, para determinar la forma más apropiada de entregar un recurso al cliente, lo cual le permite acceder al contenido según sus capacidades. El conjunto de nombres de atributos, valores permitidos y significados asociados de CC/PP, se constituyen en su vocabulario, compatible con los perfiles definidos para UAProf 2.0.

La siguiente figura se muestran los componentes de un perfil de CC/PP: el hardware sobre el cual el software se está ejecutando, el software sobre el cual las aplicaciones

están soportadas y una aplicación tal como un Browser que permite acceder ha contenido Web. La estructura de CC/PP en general está definida jerárquicamente, donde un perfil tiene asociado un número de componentes (hardware, software y browser), con uno o más atributos.



Gracias a estar basado en RDF, CC/PP es flexible y extensible que permite que se añadan conceptos a medida que los dispositivos van cambiando, lo cual se convierte en una desventaja cuando la cantidad de conceptos se hace muy grande, ya que los fabricantes los definen de acuerdo a sus características particulares [61, 65].

Anexo C

Composición STE

C.1. Sistema de Transición de Estados (STE)

Un STE es un modelo conceptual usado para describir sistemas dinámicos [55]. Una de las aplicaciones de los STE, que resulta de interés en el contexto de la composición de servicios, es su utilización como una herramienta para representar formalmente sistemas controlables en el área de la planeación automática. Área que se especializa en buscar un conjunto de acciones que ejecutadas en un orden particular, permiten cumplir con un objetivo determinado.

Formalmente un STE puede definirse como una 4-tupla $\Sigma = (S, A, E, \gamma)$, donde:

- $S = \{s_1, s_2, \ldots$, es un conjunto finito o recursivamente numerable de estados[9];
- $A = \{a_1, a_2, \ldots$, es un conjunto finito o recursivamente numerable de acciones;
- $E = \{e_1, e_2, \ldots$, es un conjunto finito o recursivamente numerable de eventos y
- $\gamma : S \times A \times E \to 2^S$, es la función de transición de estados[10].

En la definición anterior:

- Un estado identifica una situación particular de un sistema, la cual puede modificarse mediante la aplicación de una acción o la ocurrencia de un evento.
- Una acción identifica un posible efecto causado por un elemento externo al sistema y solo puede aplicarse en estados con características particulares. La constante especial no-op representa una acción neutral que no causa ningún efecto al sistema.
- Un evento identifica un posible efecto causado por la dinámica interna del sistema y solo puede presentarse en estados con características particulares. A diferencia de una acción, un evento no puede controlarse. La constante especial ε identifica un evento neutral que no causa ningún efecto.
- La función de transición describe la dinámica del sistema, por medio de la asociación de un estado o estados futuros con la aplicación de una acción o la ocurrencia de un evento. Por comodidad la transición causada únicamente por la acción a en el estado s ($\gamma(s, a, \varepsilon)$) se escribe $\gamma(s, a)$ y similarmente, la transición causada únicamente por el evento e en el estado s ($\gamma(s, \text{no-op}, e)$) se escribe $\gamma(s, e)$.

Dado un STE, el objetivo de la planeación es el de encontrar una secuencia de acciones (denominadas plan), que aplicadas ordenadamente sobre el sistema, lo conduzcan a un estado particular. Este último estado se conoce como el objetivo o la meta de un problema

[9] Un conjunto recursivamente numerable es aquel, no necesariamente finito, que puede ser generado mediante un algoritmo.

[10] 2^S es el conjunto potencia de S formado por todos los posibles subconjuntos de S.

de planeación y puede ser especificado de forma directa, o por medio de una serie de condiciones.

C.2. Representación en variables de estado de un STE

En esta sección se presenta una representación que permite codificar un STE de forma compacta. La representación en variables de estado de un STE se basa en la definición de un conjunto de constantes, variables y operadores, que pueden usarse para generar de forma dinámica cada uno de los estados, acciones, eventos y transiciones del sistema modelado. A continuación se describen cada uno de los elementos que constituyen este tipo de representación.

C.2.1. Términos

Un término identifica un individuo o grupo de individuos en el sistema modelado y puede ser, tanto un símbolo de constante como un símbolo de variable[11]. Los símbolos de constante identifican siempre el mismo individuo y aquellos que identifican objetos con características similares, son agrupados en colecciones o grupos de constantes, denominadas clases. Los símbolos de variable pueden modificar el valor referenciado de acuerdo a un rango determinado, denominado el tipo de la variable. Dicho rango está formado por una o varias clases de constantes particulares.

Respecto a la notación, el conjunto de todas las constantes del sistema es denotado con el símbolo D y las clases de constantes con el símbolo T, definidas como $T \subseteq D$. Una variable referenciada mediante el símbolo v, se define por su rango $D^v = T_1 \cup \ldots$. Las variables de constante permiten escribir expresiones que no dependen de valores particulares, por lo que facilitan la descripción compacta de un STE.

C.2.2. Variables de estado

Una variable de estado se define como una expresión de la forma $x(t_1,...,t_k,s)$, donde cada t_k es un término y s es un estado. Una variable de estado es un elemento de una función:

$$x : D_1^x \times \ldots \quad \mathrm{S} \rightarrow D_{k+1}^x,$$

Dónde $\left(D_i^x = T_1^x \cup \ldots \quad = D\right.$.

El estado s en las variables de estado se omite generalmente, debido a que en una misma expresión, todas dependen del mismo estado. De esta forma, la expresión:

$$x\left(t_1,\ldots \quad {}_{+1}\right.$$

Hace referencia implícitamente al valor de la variable de estado en el estado actual.

Una variable de estado $x(c_1,\ldots$ se dice que es substituida si todo c_i es una constante en D_i^x. Por el contrario, una variable de estado $x(v_1,\ldots$ se dice que es no substituida si

[11] No se consideran términos más complejos, como los formados por funciones, pues aumentan la complejidad del modelo sin un beneficio aparente en el dominio de los Servicios Web RESTful.

uno o más de los símbolos $v_1,\ldots$ son variables. De acuerdo con lo anterior, un estado s es especificado por los valores de todas las variables de estado substituidas en s.

Más exactamente, para cada variable de estado substituida $x(c_1,\ldots$, un estado s, incluye una expresión sintáctica de la forma $x(c_1,\ldots\ _{k+1}$, tal que c_{k+1} es el valor de $x(c_1,\ldots$ en s, con cada c_i siendo una constante en el rango apropiado.

C.2.3. Relaciones rígidas

Una relación rígida es un predicado, cuyo valor se mantiene independiente del estado en el que pueda encontrarse el sistema. Debido a estas características, las relaciones rígidas únicamente se especifican en el planteamiento de un problema de planeación y no en cada estado.

Una relación rígida es una expresión de la forma $r(t_1,\ldots$, definida por

$$r \subseteq D_1^r \times \ldots$$

Donde $\left(D_i^r = T_1 \cup \ldots\ \ = D\right.$ y cada t_k es un término.

C.2.4. Operadores, acciones y eventos

Un operador es una especificación genérica de una acción o un evento. Formalmente se define como una 3-tupla

$$o = \left\langle \text{nombre}(o), \text{precondición}(o), \text{efecto}(o) \right\rangle$$

Donde:

- $\text{nombre}(o)$, es una expresión de la forma $n(v_1,\ldots$, donde n es un símbolo que identifica unívocamente un operador y $v_1,\ldots$ son símbolos de variable que aparecen en la definición de o.
- $\text{precondición}(o)$, es un conjunto de expresión en variables de estado y relaciones. Las precondiciones especifican los requisitos para aplicar una acción o para que pueda presentarse un evento.
- $\text{efecto}(o)$, es un conjunto de expresiones de asignación de constantes a variables de estado de la forma $x(t_1,\ldots\ t_{k+1}$, donde cada t_k es un término en el rango adecuado. Los efectos especifican como debe modificarse el estado actual, para generar el estado siguiente.

Con el fin de separar los operadores usados para generar acciones, de los usados para generar eventos, se introduce los conjuntos de operadores O_A, como el conjunto de todos los operadores cuyas substituciones son acciones, y O_E, como el conjunto de todos los operadores cuyas substituciones son eventos. El conjunto de todos los operadores se define como $O = O_A \cup O_E$.

C.2.5. Lenguajes, dominios y problemas de planeación

Un lenguaje de planeación está definido como una 3-tupla $\mathcal{L} = (D, R, X)$, donde:

- D, es el conjunto de todos los símbolos de constante.
- R, es el conjunto de todas las relaciones rígidas.
- X, es el conjunto de todas las variables de estado $x(c_1, \ldots$ substituidas.

Un dominio de planeación en $\mathcal{L}$, es un STE $\Sigma = (S, A, E, \gamma)$, tal que:

- $S \subseteq \prod_{x \in X} D_x$, donde D_x es el rango de la variable de estado x. Un estado tiene la forma $s = \{(x = c) \mid x \in X\}$, donde $c \in D_x$.
- A, es el conjunto de todos los operadores en O_A substituidos.
- E, es el conjunto de todos los operadores en O_E substituidos.
- $\gamma(s, a) = \{(x = c) \mid x \in X\}$, donde c está dado por una asignación $(x \leftarrow c) \in \text{efecto}(a)$ si tal asignación existe, o en cualquier otro caso $(x = c) \in s$. Este caso es aplicable si ningún evento puede presentarse en s.
- $\gamma(s, e) = \{(x = c) \mid x \in X\}$, donde c está dado por una asignación $(x \leftarrow c) \in \text{efecto}(e)$ si tal asignación existe, o en cualquier otro caso $(x = c) \in s$. Este caso se presenta si ninguna acción es aplicable en s.
- En el caso de que tanto una acción como un evento puedan presentarse en s, entonces:
  i. Si la acción a es ejecutada antes que la ocurrencia del evento e, entonces $\gamma(s, a, e) = \gamma(s, a)$;
  ii. Si el evento e ocurre antes que la ejecución de la acción a y a no es aplicable en $\gamma(s, e)$, entonces $\gamma(s, a, e) = \gamma(s, e)$;
  iii. Si e es ejecutado primero y a es aplicable en $\gamma(s, e)$, entonces $\gamma(s, a, e) = \{(x = c) \mid x \in X\}$, donde c está dado por la secuencia estricta de asignaciones $(x \leftarrow c) \in \text{efecto}(e)$ y $(x \leftarrow c) \in \text{efecto}(a)$ si tales asignaciones existen, o en cualquier otro caso $(x = c) \in S$.
- s es cerrado en γ, es decir para cualquier acción o evento aplicable en un estado s, entonces $\gamma(s, a, e) \in S$.

Un problema de planeación es una tripleta $\mathcal{P} = (\Sigma, S_0, S_g)$, donde:

- Σ, es el dominio de planeación;
- S_0, es el conjunto de estados iniciales y
- S_g, es el conjunto de estados finales.

C.2.6. Sentencia de un problema de planeación

La sentencia de un problema de planeación P, contiene toda la información requerida para solucionar un problema de planeación, a saber $P = (O, R, S_0, S_g)$, donde:

- $O = O_A \cup O_E$, es el conjunto de todos los operadores;
- R, es el conjunto de todas las relaciones rígidas;
- S_0, es el conjunto de estados iniciales y
- S_g, es el conjunto de estados finales.

C.3. Rutinas auxiliares

A. Pre-Imagen débil

La pre-imagen débil de un estado s se encuentra conformada por el conjunto de estados de la aplicación, en los que las transiciones que pueden presentarse en dichos estados, ya sean por ejecución de un enlace u ocurrencia de un evento, puedan llevar la aplicación a dicho estado s, aunque no haya garantía de ello. Formalmente:

$$\text{preImagenDébil}(S) = \{(s,a,e) \mid \gamma(s,a,e) \cap S \neq \varnothing\}.$$

B. Pre-Imagen fuerte

La pre-imagen fuerte de un estado s se encuentra conformada por el conjunto de estados de la aplicación, en los que las transiciones que pueden presentarse en dichos estados, ya sean por ejecución de un enlace u ocurrencia de un evento, lleven la aplicación a dicho estado s, con garantía de esto. Formalmente:

$$\text{preImagenFuerte}(S) = \{(s,a,e) \mid \gamma(s,a,e) \subseteq S\}.$$

C. Eliminación de estados

La eliminación de estados es aplicada a una política particular y está definida por:

$$\text{eliminarEstados}(\pi, S) = \{(s,a,e) \in \pi \mid s \notin S\}.$$

D. Eliminación de estados inalcanzables

Las políticas resultantes de los algoritmos que serán presentados más adelante, presentan estados inalcanzables, es decir estados que no hacen parte de ninguno de los posibles caminos de ejecución de una política. Esta rutina está dada por:

$$\text{eliminarInalcanzables}(\pi, S) = \bigcup_{s \ldots} \ldots) \in \pi \mid s \in \hat{\Gamma}(s')\},$$

donde $\hat{\Gamma}(s)$ es el conjunto de estados que es posible alcanzar desde $s \in S$, definido por:

$$\hat{\Gamma}(s) = \Gamma(s) \cup \Gamma^2(s) \cup \ldots,$$

en el cual $\Gamma(s)$ es el conjunto de los estados sucesores de s, dado por

$$\Gamma(s) = \{\gamma(s,a,e) \mid a \in A(s) \wedge e \in E(s)\},$$

$$\Gamma^2(s) = \Gamma(\Gamma(s)) = \bigcup_{s' \in \Gamma(s)} \quad \text{y}$$

$$\Gamma^3(s) = \Gamma(\Gamma^2(s)) = \ldots$$

E. Eliminación de transiciones salientes

La eliminación de transiciones salientes de una política π y un conjunto de estados S, elimina todas las transiciones pertenecientes π que no derivan en estados pertenecientes a S. Formalmente:

$$\text{eliminarSalientes}(\pi, S) = \{(s,a,e) \in \pi \mid \gamma(s,a,e) \subseteq (S \cup S_\pi)\}.$$

F. Eliminación de transiciones desconectadas

La eliminación de transiciones desconectadas de una política π y un conjunto de estados S, elimina todas las transiciones pertenecientes a π que no derivan en estados pertenecientes a la pre-imagen débil de S. Esta rutina está definida por el algoritmo mostrado en la siguiente lista.

Eliminación de transiciones desconectadas

1. $\text{eliminarDesconectadas}(\pi, S)$
2. $\pi' \leftarrow \varnothing$, $\pi'' \leftarrow \text{falla}$
3. **mientras** $\pi' \neq \pi''$ **hacer**
4. $\pi'' \leftarrow \pi'$
5. $\pi' \leftarrow \pi \cap \text{preImagenDébil}(S \cup S_{\pi'})$
6. **fin mientras**
7. **retornar** π'
8. **fin**

G. Eliminación de transiciones no progresivas

La eliminación de transiciones no progresivas de una política π y un conjunto de estados S, elimina las transiciones pertenecientes a π que no contribuyen en alcanzar los estados en S. Esta rutina está dada por la siguiente lista.

Eliminación de transiciones no progresivas

1. $\text{eliminarNoProgresivas}(\pi, S)$
2. $\pi' \leftarrow \varnothing$, $\pi'' \leftarrow \text{falla}$
3. **mientras** $\pi' \neq \pi''$ **hacer**
4. $\text{preImagen} \leftarrow \pi \cap \text{preImagenDébil}(S \cup S_{\pi'})$
5. $\pi'' \leftarrow \pi'$
6. $\pi' \leftarrow \pi' \cup \text{eliminarEstados}(\text{preImagen}, S \cup S_{\pi'})$
7. **fin mientras**

8. **retornar** π'
9. **fin**

C.4. Algoritmos de planeación.

C.4.1. Algoritmo para generar soluciones débiles

El algoritmo en términos generales, realiza una búsqueda en amplitud (en inglés *BFS - Breadth First Search*) hacia atrás en el espacio de estados, iniciando en los estados de la aplicación finales y retrocediendo hasta llegar a los estados iniciales. El algoritmo aplica la rutina preImagenDébil para determinar los estados de la aplicación en los que existen enlaces, cuya selección o ejecución permiten llevar la aplicación a un estado deseado. El algoritmo se presenta a continuación.

El algoritmo presentado construye la política resultante mediante la adición repetida de las transiciones derivadas de la pre-imagen débil de los estados finales esperados. El algoritmo termina cuando no haya más transiciones que adicionar o cuando los estados iniciales estén en los estados de la política.

Algoritmo para generar soluciones débiles

1. $\pi \leftarrow \varnothing$, $\pi' \leftarrow \text{falla}$
2. **mientras** $\pi \neq \pi'$ **y** $S_0 \not\subset \left(S_g \cup S_\pi\right)$ **hacer**
3. $\text{preImagen} \leftarrow \text{preImagenDébil}\left(S_g \cup S_\pi\right)$
4. $\pi'' \leftarrow \text{eliminarEstados}\left(\text{preImagen}, S_g \cup S_\pi\right)$
5. $\pi' \leftarrow \pi$
6. $\pi \leftarrow \pi \cup \pi''$
7. **fin mientras**
8. **si** $S_0 \subseteq \left(S_g \cup S_\pi\right)$ **entonces**
9. **retornar** $\text{eliminarInalcanzables}\left(\pi, S_0\right)$
10. **sino**
11. **retornar** falla
12. **fin si**

C.4.2. Algoritmo para generar soluciones fuertes

El algoritmo para generar soluciones fuertes es idéntico al algoritmo usado para generar soluciones débiles, excepto que la rutina usada para determinar la pre-imagen de $S_g \cup S_\pi$ es preImagenFuerte y no preImagenDébil.

C.4.3. Algoritmo para generar soluciones cíclicas-fuertes

El algoritmo para la generación de soluciones cíclicas fuertes inicia con la política universal, la cual se define como la política, cuyos elementos son todas las posibles tripletas, estado-acción-evento, permitidos en el servicio, es decir:

$$\text{PolíticaUniversal} = \left\{(s,a,e) \mid a \in A(s) \wedge e \in E(s)\right\}.$$

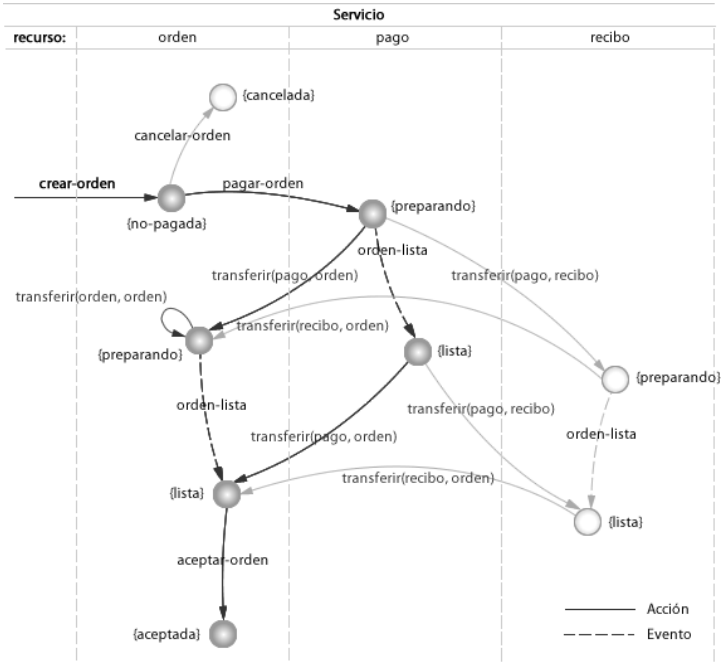
Posteriormente, el algoritmo elimina gradualmente tripletas estado-acción-evento de acuerdo con las rutinas: eliminarSalientes , eliminarDesconectadas y eliminarNoProgresivas . La siguiente lista presenta el algoritmo.

Algoritmo para generar soluciones cíclicas-fuertes

1. $\pi \leftarrow \text{PolíticaUniversal}$, $\pi' \leftarrow \varnothing$
2. **mientras** $\pi \neq \pi'$ **hacer**
3. $\pi' \leftarrow \pi$
4. $\pi \leftarrow \text{eliminarDesconectadas}\left(\text{eliminarSalientes}\left(\pi, S_g\right), S_g\right)$
5. **fin mientras**
6. **si** $S_0 \subseteq \left(S_g \cup S_\pi\right)$ **entonces**
7. **retornar** $\text{eliminarInalcanzables}\left(\text{eliminarNoProgresivas}\left(\pi, S_g\right), S_0\right)$
8. **fin si**

C.5. Ejemplo.

Para ilustrar la composición STE se presenta un ejemplo basado en la aplicación de comercio electrónico presentada en [31]. En dicha aplicación es posible realizar órdenes o pedidos, efectuar un pago de la orden solicitada y tomar un recibo opcional. La orden es un recurso complejo que depende de la lógica interna del servicio y de su propio estado. Una orden puede estar no-pagada, cancelada, preparando, lista o aceptada y de acuerdo con dichos estados, se pueden obtener diferentes enlaces. Por ejemplo, cuando la orden se encuentre no-pagada, el servidor enviará una representación de la orden junto con un enlace para realizar el pago.



El STE del servicio de comercio electrónico presentado anteriormente, estaría definido por:

Constantes

Se definen las constantes:

$$D = \{\text{orden, pago, recibo, no-pagada, cancelada, preparando, lista, aceptada}\},$$

Cuyas clases quedan definidas por las expresiones

$$\textit{recurso}(\text{orden}),\ \textit{recurso}(\text{pago}),\ \textit{recurso}(\text{recibo}),\ \textit{estadoOrden}(\text{no-pagada}),\ \textit{estadoOrden}(\text{cancelada}),$$
$$\textit{estadoOrden}(\text{preparando}),\ \textit{estadoOrden}(\text{lista}),\textit{estadoOrden}(\text{aceptada}).$$

Relaciones rígidas

Las relaciones rígidas son las siguientes:

$$\textit{enlazado}(\text{orden},\text{orden}),\ \textit{enlazado}(\text{pago, orden}),$$
$$\textit{enlazado}(\text{pago, recibo}),\ \textit{enlazado}(\text{recibo, orden}).$$

Variables de estado

Las variables de estado son: *recursoActual* , la cual ya fue definida, y *orden.estado* que corresponde al atributo estado de una orden en el servicio Web RESTful y se define con la función de variable de estado:

$$\textit{orden.estado} : S \rightarrow \textit{estadoOrden}.$$

Acciones

A continuación se muestran las acciones derivadas del operador *transferir* , las cuales cambian únicamente el recurso actual en el cliente y cumplen las relaciones rígidas planteadas. Además, se incluyen las acciones específicas de la lógica del servicio.

Acción crear-orden

```
1.  acción crear-orden
2.      efecto:          recursoActual ← orden,
3.                       orden.estado ← no-pagada
```

Acción cancelar-orden

```
1.  acción cancelar-orden
2.      precondición:    recursoActual = orden,
3.                       orden.estado = no-pagada
4.      efecto:          orden.estado ← cancelada
```

Acción pagar-orden

```
1.  acción pagar-orden
2.      precondición:    recursoActual = orden,
3.                       orden.estado = no-pagada
4.      efecto:          recursoActual ← pago,
5.                       orden.estado ← preparando
```

Acción transferir(pago, orden)

```
1. acción transferir(pago, orden)
2.     precondición:   recursoActual = pago,
3.     efecto:         recursoActual ← orden
```

Acción transferir(recibo, orden)

```
1. acción transferir(recibo, orden)
2.     precondición:   recursoActual = recibo,
3.     efecto:         recursoActual ← orden
```

Acción transferir(pago, recibo)

```
1. acción transferir(pago, recibo)
2.     precondición:   recursoActual = pago,
3.     efecto:         recursoActual ← recibo
```

Acción aceptar-orden

```
1. acción aceptar-orden
2.     precondición:   recursoActual = orden,
3.                     orden.estado = lista
4.     efecto:         recursoActual ← recibo,
5.                     orden.estado = aceptada
```

Eventos

Se tiene un único evento, el cual puede presentarse una vez la orden sea pagada. Este evento se debe a que la orden no puede prepararse instantáneamente y requiere de un periodo de tiempo indeterminado para estar lista. La siguiente lista muestra la definición de este evento.

Evento orden-lista

```
1. evento orden-lista
2.     precondición:   orden.estado = preparando
3.     efecto:         orden.estado ← lista
```

Un caso de composición

Estado inicial

$$S_0 = \{s_0\},\ s_0 = \{\text{recursoActual} = \text{orden}, \text{orden.estado} = \text{no-pagada}\},$$

Para un problema de planeación donde

$$S_g = \{s_g\},\ s_g = \{\text{recursoActual} = \text{orden}, \text{orden.estado} = \text{aceptada}\},$$

teniendo en cuenta que dicho servicio puede presentar eventos internos, el algoritmo a utilizar en primera instancia es aquel que genera soluciones fuertes. Sin embargo, en este caso, debido a que la única forma de llegar al estado final es a través de estados en los que se presentan eventos que pueden ocasionar transiciones que deriven en estados fuera de los estados objetivos o finales, entonces no existe una solución fuerte. Por esta razón, es necesario aplicar el algoritmo que produce soluciones cíclicas-fuertes.

La solución arrojada por el algoritmo es entonces:

$$\{\text{recursoActual} = \text{orden}, \text{orden.estado} = \text{no-pagada}\} \rightarrow \{(\text{pagar-orden}, \text{nulo})\}$$

$$\{\text{recursoActual} = \text{orden}, \text{orden.estado} = \text{preparando}\} \rightarrow \{(\text{transferir}(\text{orden}, \text{orden}), \text{orden-lista})\}$$

$$\{\text{recursoActual} = \text{orden}, \text{orden.estado} = \text{lista}\} \rightarrow \{(\text{aceptar-orden}, \text{nulo})\}$$

$$\{\text{recursoActual} = \text{pago}, \text{orden.estado} = \text{preparando}\} \rightarrow \{(\text{transferir}(\text{pago}, \text{orden}), \text{orden-lista})\}$$

$$\{\text{recursoActual} = \text{pago}, \text{orden.estado} = \text{lista}\} \rightarrow \{(\text{transferir}(\text{pago}, \text{orden}), \text{nulo})\}$$

Implementación

El proceso del ejemplo anterior fue implementado usando un servicio Web RESTful definido en [31]. El cliente que consumió el servicio estableció la política necesaria para el consumo efectivo del servicio según las metas y estados iniciales vistos. A continuación se presenta un registro de dicho cliente.

```
INFO: 1 * Client out-bound request
1 > POST http://localhost:9998/order
1 > Content-Type: application/vnd.restbucks+xml
<?xml version="1.0" encoding="UTF-8"?>
<order xmlns="http://schemas.restbucks.com">
    <item>
        <milk>skim</milk>
        <size>medium</size>
        <drink>cappuccino</drink>
    </item>
    …
    <location>inStore</location>
</order>
9/12/2011 05:31:22 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 1 * Client in-bound response
1 < 201
1 < Date: Fri, 09 Dec 2011 22:31:22 GMT
1 < Content-Length: 1104
1 < Location: http://localhost:9998/order/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23
1 < server: grizzly/1.8.1
1 < Content-Type: application/vnd.restbucks+xml
1 <
9/12/2011 05:31:22 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 2 * Client out-bound request
2 > GET http://localhost:9998/order/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23
2 > Accept: application/vnd.restbucks+xml

9/12/2011 05:31:22 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 2 * Client in-bound response
2 < 200
2 < Date: Fri, 09 Dec 2011 22:31:22 GMT
2 < Content-Length: 1104
2 < server: grizzly/1.8.1
2 < Content-Type: application/vnd.restbucks+xml
2 <
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<order xmlns="http://schemas.restbucks.com" xmlns:rel="http://schemas.restbucks.com/dap">
    <rel:link mediaType="application/vnd.restbucks+xml"
              uri="http://localhost:9998/order/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23"
              rel="http://relations.restbucks.com/cancel"/>
    <rel:link mediaType="application/vnd.restbucks+xml"
              uri="http://localhost:9998/payment/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23"
              rel="http://relations.restbucks.com/payment"/>
    <rel:link mediaType="application/vnd.restbucks+xml"
              uri="http://localhost:9998/order/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23"
              rel="http://relations.restbucks.com/update"/>
```

```
    <rel:link mediaType="application/vnd.restbucks+xml"
              uri="http://localhost:9998/order/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23"
rel="self"/>
    <item>
        <milk>skim</milk>
        <size>medium</size>
        <drink>cappuccino</drink>
    </item>
    …
    <location>inStore</location>
    <cost>4.0</cost>
    <status>unpaid</status>
</order>
9/12/2011 05:31:22 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 3 * Client out-bound request
3 > PUT http://localhost:9998/payment/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23
3 > Accept: application/vnd.restbucks+xml
3 > Content-Type: application/vnd.restbucks+xml
<?xml version="1.0" encoding="UTF-8"?>
<payment xmlns="http://schemas.restbucks.com">
    <amount>4.0</amount>
    <cardholderName>Harry Valley</cardholderName>
    <cardNumber>1234-1234-1234-12</cardNumber>
    <expiryMonth>12</expiryMonth>
    <expiryYear>14</expiryYear>
</payment>

9/12/2011 05:31:22 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 3 * Client in-bound response
3 < 201
3 < Date: Fri, 09 Dec 2011 22:31:22 GMT
3 < Content-Length: 697
3 < Location: http://localhost:9998/payment/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23
3 < server: grizzly/1.8.1
3 < Content-Type: application/vnd.restbucks+xml
3 <
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<payment xmlns:rel="http://schemas.restbucks.com/dap" xmlns="http://schemas.restbucks.com">
    <rel:link mediaType="application/vnd.restbucks+xml"
          uri="http://localhost:9998/order/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23"
          rel="http://relations.restbucks.com/order"/>
    <rel:link mediaType="application/vnd.restbucks+xml"
          uri="http://localhost:9998/receipt/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23"
          rel="http://relations.restbucks.com/receipt"/>
    <amount>4.0</amount>
    …
</payment>
9/12/2011 05:31:22 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 4 * Client out-bound request
4 > GET http://localhost:9998/order/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23
4 > Accept: application/vnd.restbucks+xml

9/12/2011 05:31:22 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 4 * Client in-bound response
4 < 200
4 < Date: Fri, 09 Dec 2011 22:31:22 GMT
4 < Content-Length: 600
4 < server: grizzly/1.8.1
4 < Content-Type: application/vnd.restbucks+xml
4 <
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<order xmlns="http://schemas.restbucks.com" xmlns:rel="http://schemas.restbucks.com/dap">
    <rel:link mediaType="application/vnd.restbucks+xml"
              uri="http://localhost:9998/order/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23"
rel="self"/>
    <item>
        <milk>skim</milk>
        <size>medium</size>
        <drink>cappuccino</drink>
    </item>
```

```
    …
    <location>inStore</location>
    <cost>4.0</cost>
    <status>preparing</status>
</order>
9/12/2011 05:31:22 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 5 * Client out-bound request
5 > GET http://localhost:9998/order/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23
5 > Accept: application/vnd.restbucks+xml

9/12/2011 05:31:22 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 5 * Client in-bound response
5 < 200
5 < Date: Fri, 09 Dec 2011 22:31:22 GMT
5 < Content-Length: 600
5 < server: grizzly/1.8.1
5 < Content-Type: application/vnd.restbucks+xml
5 <
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<order xmlns="http://schemas.restbucks.com" xmlns:rel="http://schemas.restbucks.com/dap">
    <rel:link mediaType="application/vnd.restbucks+xml"
              uri="http://localhost:9998/order/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23"
rel="self"/>
    …
    <status>preparing</status>
</order>
9/12/2011 05:31:22 PM rwsc.model.service.execution.context.PolicyExecutor nextLink
INFO: Waiting 5 seconds before executing link /restbucks/resource/order/link/self-order

9/12/2011 05:31:27 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 6 * Client out-bound request
6 > GET http://localhost:9998/order/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23
6 > Accept: application/vnd.restbucks+xml

9/12/2011 05:31:27 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 6 * Client in-bound response
6 < 200
6 < Date: Fri, 09 Dec 2011 22:31:27 GMT
6 < Content-Length: 600
6 < server: grizzly/1.8.1
6 < Content-Type: application/vnd.restbucks+xml
6 <
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<order xmlns="http://schemas.restbucks.com" xmlns:rel="http://schemas.restbucks.com/dap">
    <rel:link mediaType="application/vnd.restbucks+xml"
              uri="http://localhost:9998/order/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23"
rel="self"/>
    …
    <status>preparing</status>
</order>
9/12/2011 05:31:27 PM rwsc.model.service.execution.context.PolicyExecutor nextLink
INFO: Waiting 5 seconds before executing link /restbucks/resource/order/link/self-order

9/12/2011 05:31:32 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 7 * Client out-bound request
7 > GET http://localhost:9998/order/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23
7 > Accept: application/vnd.restbucks+xml

9/12/2011 05:31:32 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 7 * Client in-bound response
7 < 200
7 < Date: Fri, 09 Dec 2011 22:31:32 GMT
7 < Content-Length: 632
7 < server: grizzly/1.8.1
7 < Content-Type: application/vnd.restbucks+xml
7 <
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<order xmlns="http://schemas.restbucks.com" xmlns:rel="http://schemas.restbucks.com/dap">
    <rel:link mediaType="application/vnd.restbucks+xml"
              uri="http://localhost:9998/receipt/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23"
```

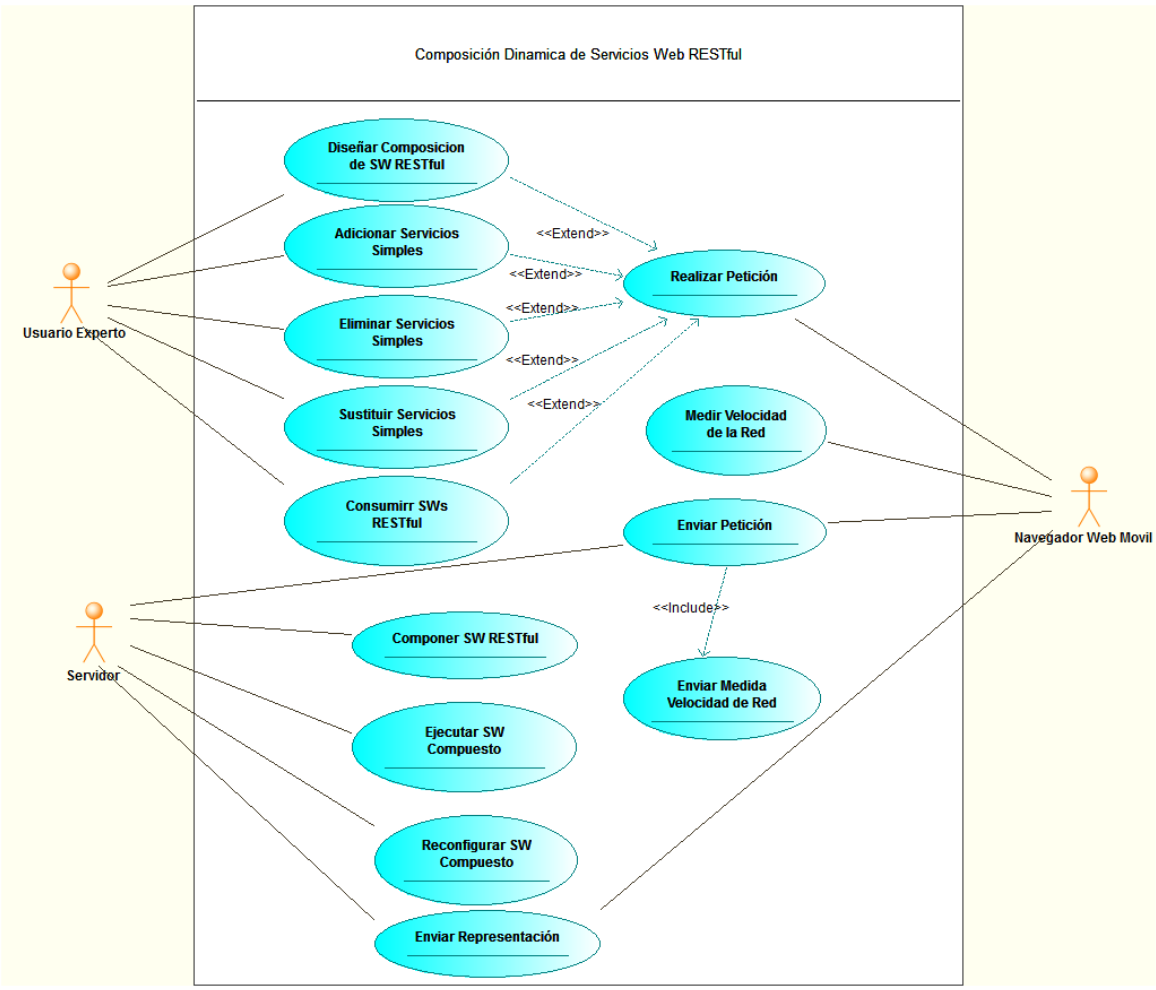
```
                rel="http://relations.restbucks.com/receipt"/>
    …
    <status>ready</status>
</order>
9/12/2011 05:31:32 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 8 * Client out-bound request
8 > DELETE http://localhost:9998/receipt/bc36bedc-c5c0-4ba0-bd4b-5d533c030e23
8 > Accept: application/vnd.restbucks+xml

9/12/2011 05:31:32 PM com.sun.jersey.api.client.filter.LoggingFilter log
INFO: 8 * Client in-bound response
8 < 200
8 < Date: Fri, 09 Dec 2011 22:31:32 GMT
8 < Content-Length: 461
8 < server: grizzly/1.8.1
8 < Content-Type: application/vnd.restbucks+xml
8 <
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<order xmlns:rel="http://schemas.restbucks.com">
    <item>
        <milk>skim</milk>
        <size>medium</size>
        <drink>cappuccino</drink>
    </item>
    …
    <location>inStore</location>
    <cost>4.0</cost>
    <status>taken</status>
</order>
```

Anexo D

Diseño e implementación del prototipo para la composición dinámica de servicios Web RESTful

El prototipo para la composición dinámica de servicios Web RESTful, implementa los casos de uso mostrados en la siguiente figura.

D.1. Casos de uso



La descripción de los casos de uso, se muestra a continuación.

D.1.1. Caso de uso diseñar composición.

Caso de uso No:	**1**
Caso de uso:	Diseñar Composición de SW RESTful
Propósito:	Crear un servicio Web RESTful a partir de un orden establecido por el usuario.
Actor:	Usuario Experto
Precondiciones:	Ninguna

Flujo normal de eventos

Usuario Experto	**Sistema**
1. El usuario adiciona servicios, los enlaza y selecciona la opción Enviar.	2. El sistema realiza la petición de usuario al servidor.
	3. El sistema muestra los parámetros requeridos para consumir el servicio.

Flujo de excepciones

Usuario Experto	**Sistema**
	3. No existe una solución al problema de composición. El sistema muestra un mensaje notificando al usuario de la solución.

Postcondiciones:	El servicio ha sido generado.

D.1.2. Consumir composición

Caso de uso No:	**2**
Caso de uso:	Consumir SW RESTful
Propósito:	Consumir un servicio Web RESTful previamente creado
Actor:	Usuario Experto
Precondiciones:	El servicio Web RESTful debe haberse generado

Flujo normal de eventos

Usuario Experto	**Sistema**
1. El usuario ingresa los parámetros requeridos del servicio compuesto y selecciona la opción ejecutar.	2. El sistema envía la petición con los datos ingresados por el usuario.
	3. El sistema muestra la representación que resulta como respuesta del servicio invocado por el usuario.
4. El usuario selecciona la opción Siguiente o Anterior para observar diferentes respuestas.	5. El sistema consulta las respuestas correspondientes y las muestra al usuario.

Flujo de excepciones

Usuario Experto	Sistema
	3. Existe un error irrecuperable (la sustitución de servicios no fue posible) en la ejecución de un servicio. El sistema muestra un mensaje de error.

Postcondiciones:	El servicio ha sido consumido.

D.1.3. Adicionar servicio

Caso de uso No:	**3**
Caso de uso:	Adicionar servicios simples
Propósito:	Adicionar un nuevo servicio a la composición desarrollada
Actor:	Usuario Experto
Precondiciones:	El servicio compuesto debe haberse consumido

Flujo normal de eventos

Usuario Experto	Sistema
1. El usuario selecciona la opción editar servicio en la interfaz de resultados de ejecución de un servicio compuesto.	2. El sistema muestra la interfaz para la edición o modificación de un servicio compuesto, la cual da la alternativa de adicionar servicios.
3. El usuario adiciona uno o más servicios indicando sus metas y preferencias y selecciona la opción aceptar edición.	4. El sistema envía la petición de edición con los nuevos servicios solicitados. El servidor modifica el servicio compuesto y realiza la ejecución.
	5. El sistema muestra la respuesta del servicio compuesto modificado.

Flujo de excepciones

Usuario Experto	Sistema
	5. La modificación no pudo ser ejecutada. El sistema muestra un mensaje de error indicándole al usuario.

Postcondiciones:	El servicio ha sido modificado.

D.1.4. Eliminar servicio

Caso de uso No:	**4**
Caso de uso:	Eliminar servicios simples
Propósito:	Eliminar un servicio existente en la composición de usuario
Actor:	Usuario Experto
Precondiciones:	El servicio compuesto debe haberse consumido

Flujo normal de eventos

Usuario Experto	**Sistema**
1. El usuario selecciona la opción editar servicio en la interfaz de resultados de ejecución de un servicio compuesto.	2. El sistema muestra la interfaz para la edición o modificación de un servicio compuesto, la cual da la alternativa de eliminar servicios.
3. El usuario elimina uno o más servicios y selecciona la opción aceptar edición.	4. El sistema envía la petición de edición indicando que servicios deben eliminarse. El servidor modifica el servicio compuesto y realiza la ejecución.
	5. El sistema muestra la respuesta del servicio compuesto modificado.

Flujo de excepciones

Usuario Experto	**Sistema**
	5. La modificación no pudo ser ejecutada. El sistema muestra un mensaje de error indicándole al usuario.

Postcondiciones:	El servicio ha sido modificado.

D.1.5. Sustituir servicio

Caso de uso No:	**5**
Caso de uso:	Sustituir servicios simples
Propósito:	Sustituir o remplazar un servicio simple de un servicio compuesto
Actor:	Usuario Experto
Precondiciones:	El servicio compuesto debe haberse consumido

Flujo normal de eventos

Usuario Experto	Sistema
1. El usuario selecciona la opción editar servicio en la interfaz de resultados de ejecución de un servicio compuesto.	2. El sistema muestra la interfaz para la edición o modificación de un servicio compuesto, la cual da la alternativa de sustituir servicios.
3. El usuario sustituye el servicio seleccionado para uno o más metas del servicio compuesto.	4. El sistema envía la petición de edición indicando que servicios deben sustituirse. El servidor modifica el servicio compuesto y realiza la ejecución.
	5. El sistema muestra la respuesta del servicio compuesto modificado.

Flujo de excepciones

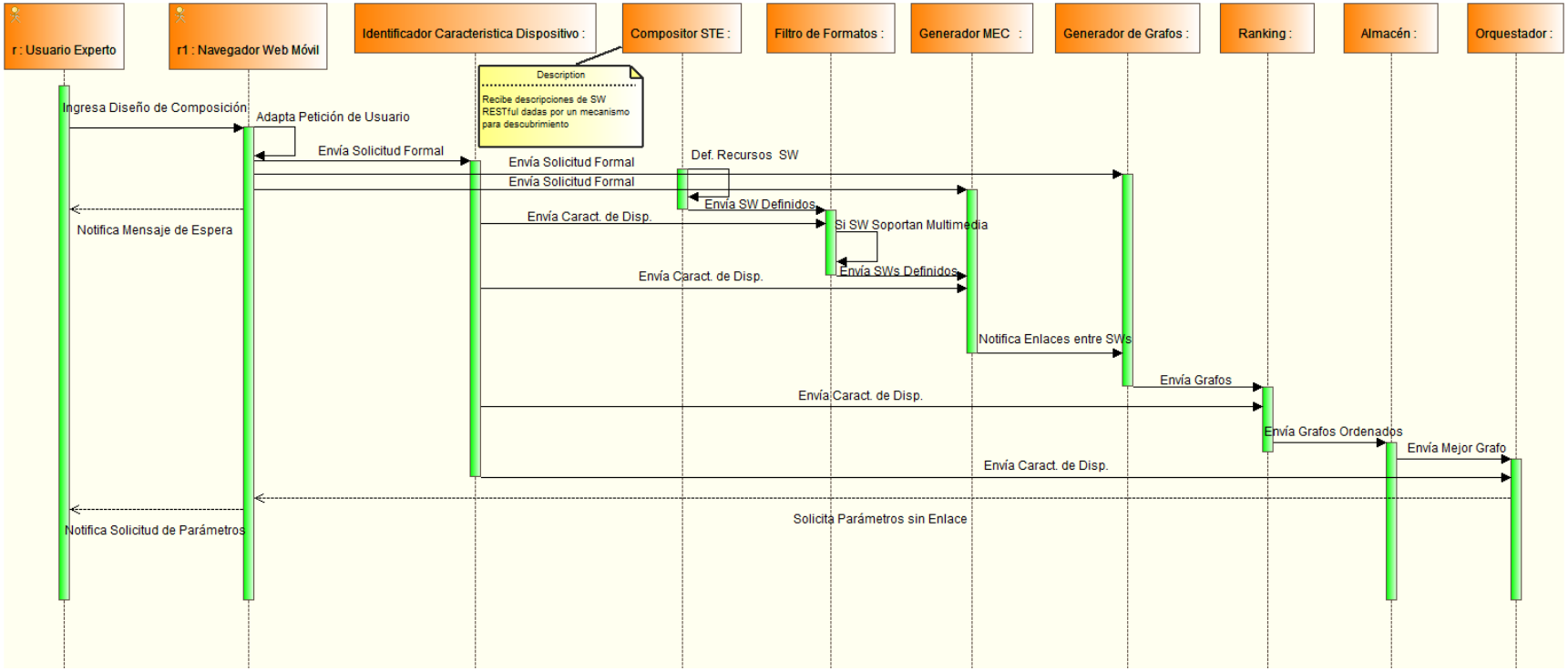
Usuario Experto	Sistema
	5. La modificación no pudo ser ejecutada. El sistema muestra un mensaje de error indicándole al usuario.

Postcondiciones:	El servicio ha sido modificado.

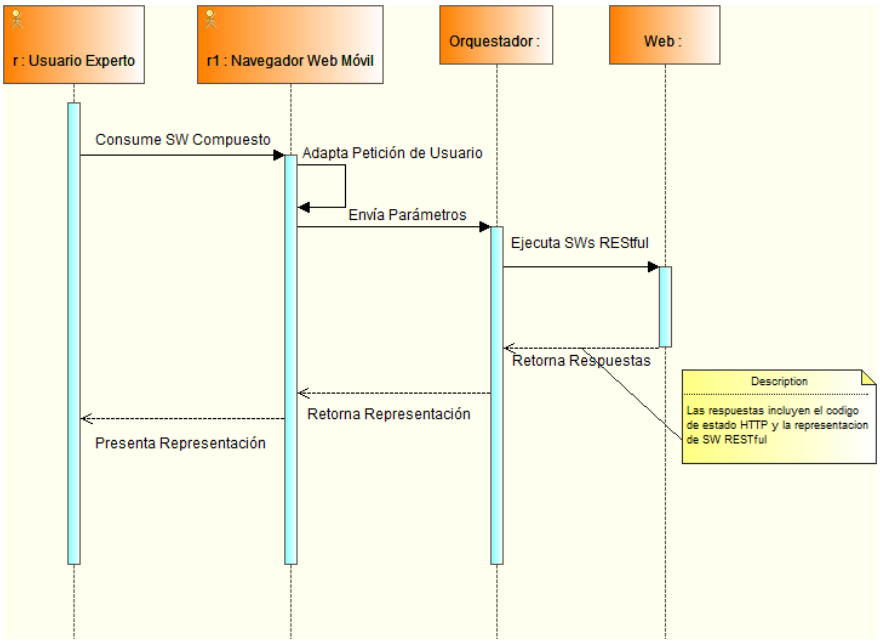
D.2. Diagramas de secuencia

Los diagramas de secuencia, reflejan el flujo normal de eventos para cada caso de uso, expuesto anteriormente.

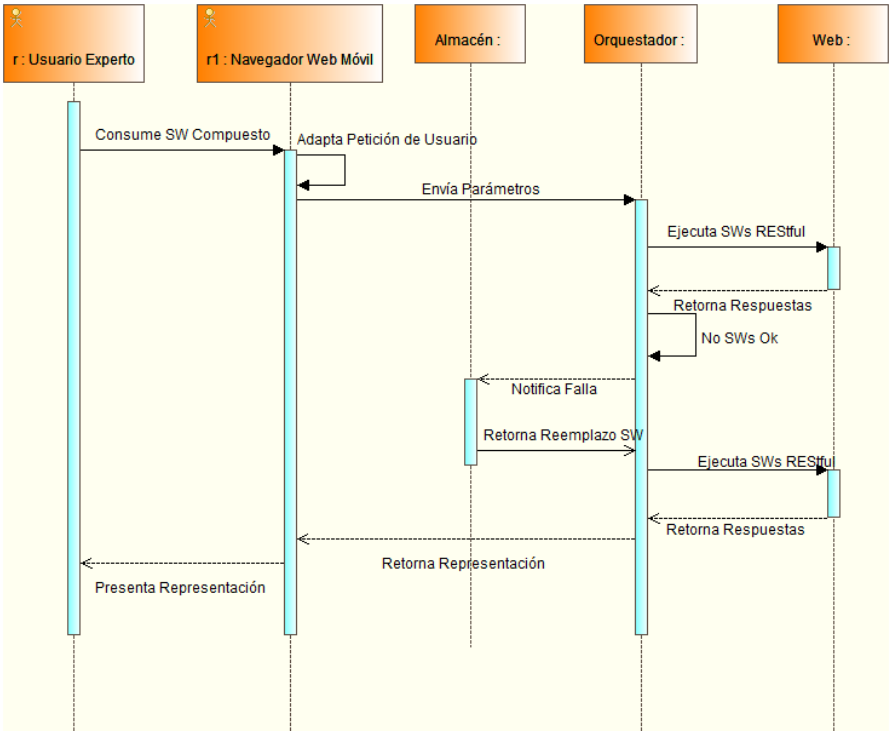
D.2.1. Diagrama de secuencia proceso de composición automático



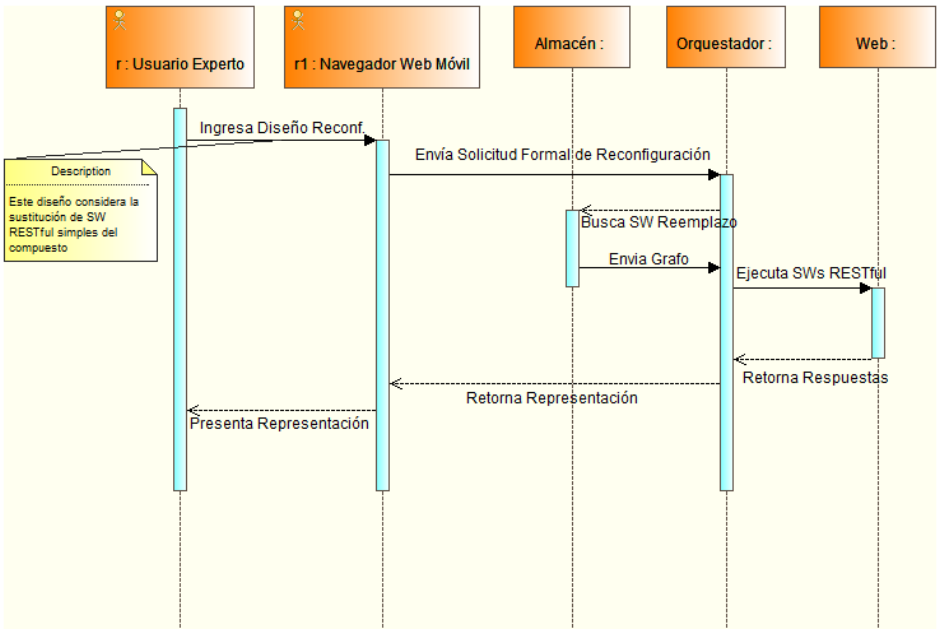
D.2.2. Diagrama de secuencia consumo servicio compuesto



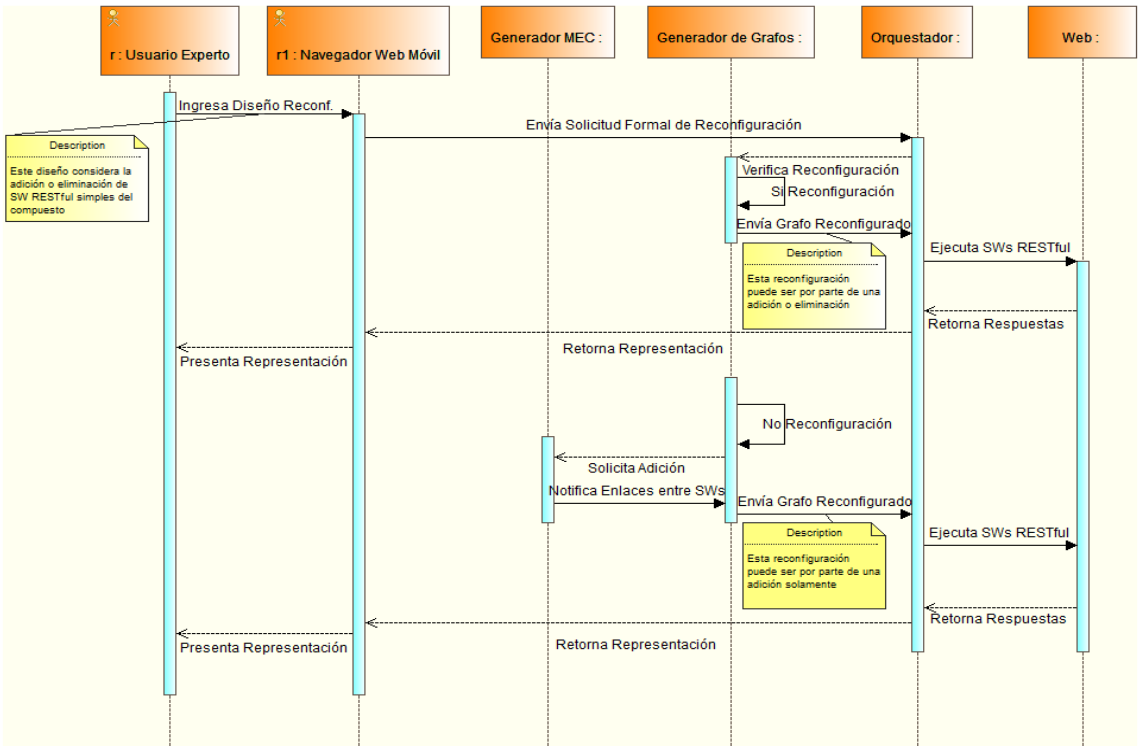
D.2.3. Diagrama de secuencia sustitución de servicio debido a fallas de servicio



D.2.4. Diagrama de secuencia sustitución de servicios debido a cambios del usuario

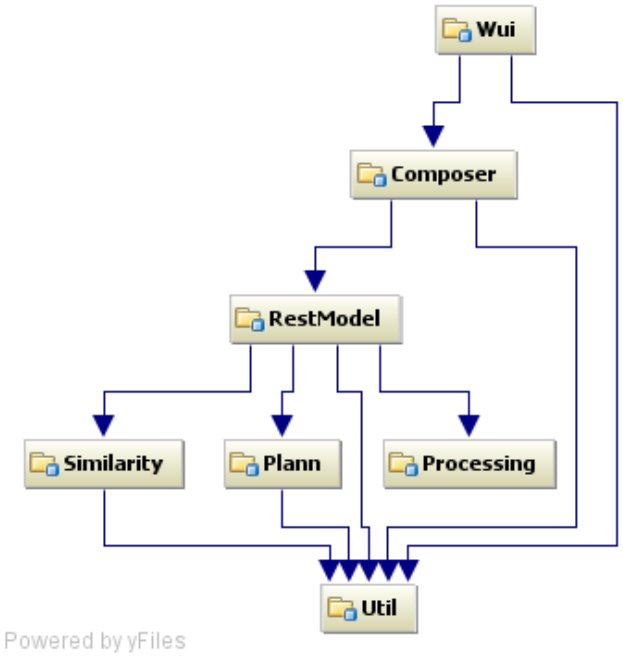


D.2.5. Diagrama de secuencia eliminación o adición de servicios debido a cambios del usuario



D.3.Diagrama de paquetes

Este diagrama muestra una vista general del sistema, y sus relaciones entre módulos.

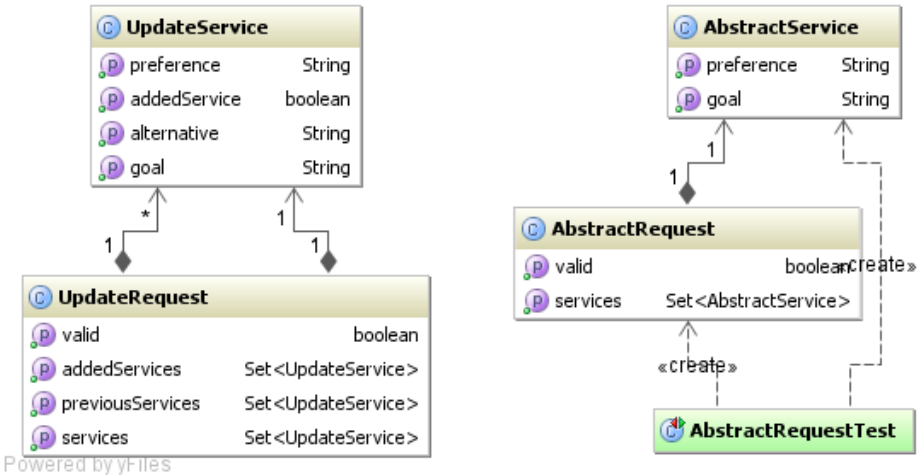


D.4. Diagrama de clases

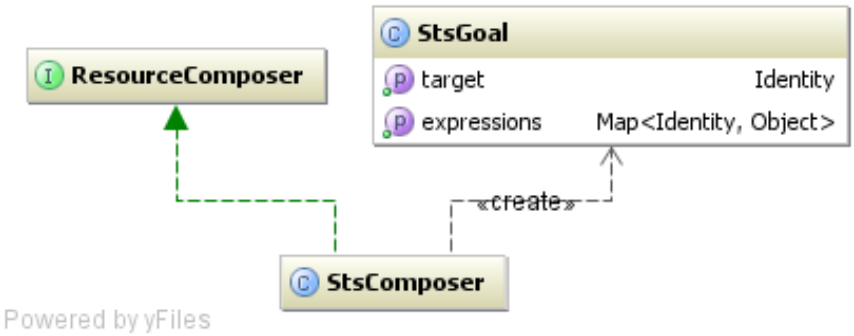
Los diagramas de clase muestran la estructura de cada modulo para su implementación, estas se muestran a continuación, ordenadas en base al modulo al que pertenecen.

Módulo de composición.

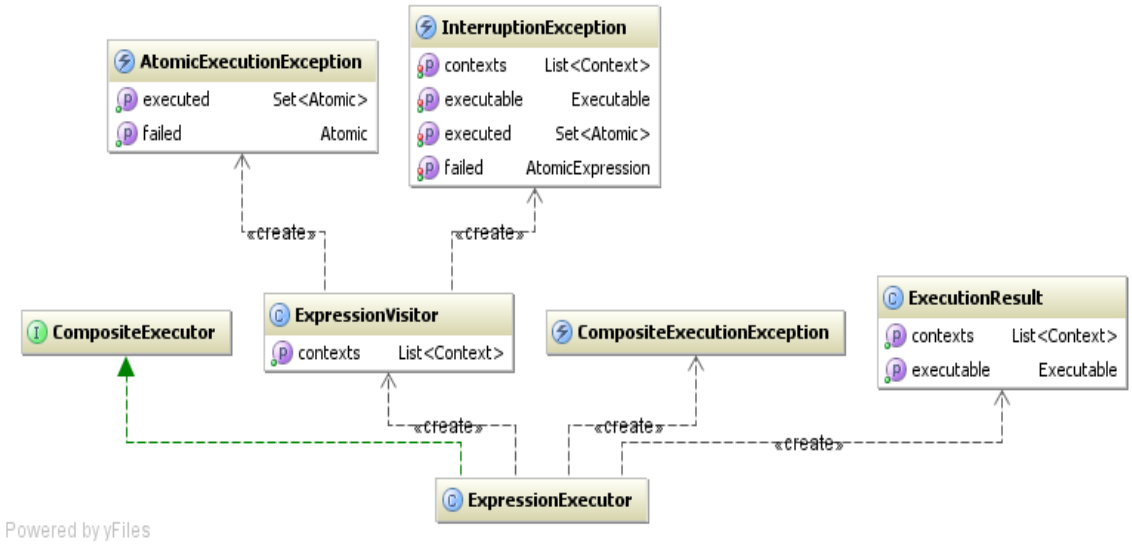
1. **Paquete petición.**



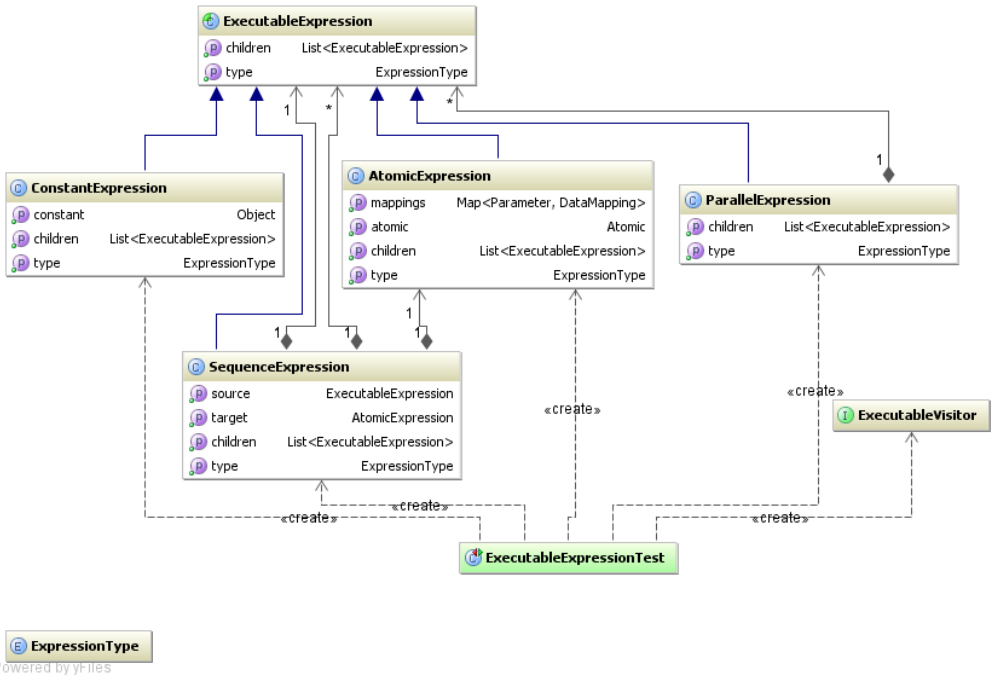
2. **Paquete recurso.**



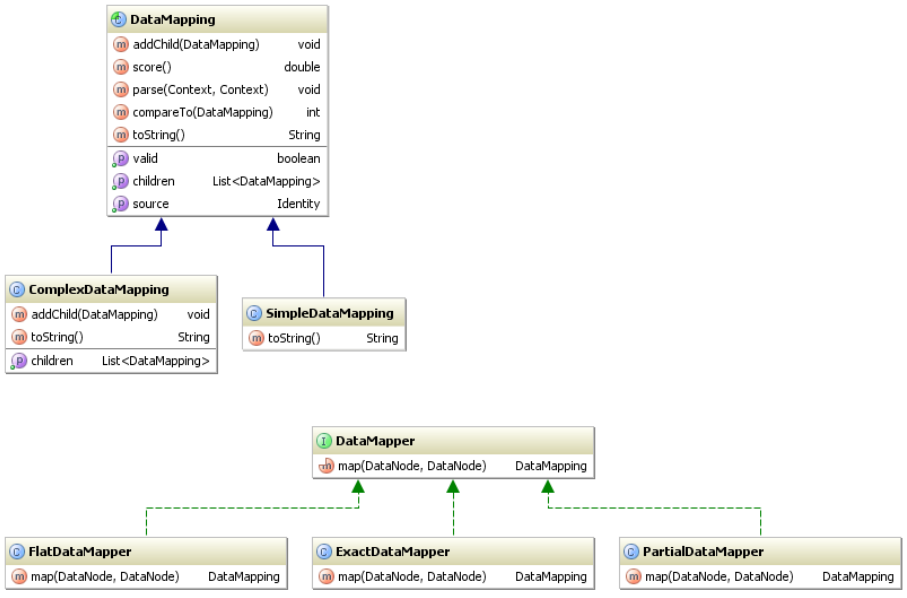
3. **Paquete ejecución.**



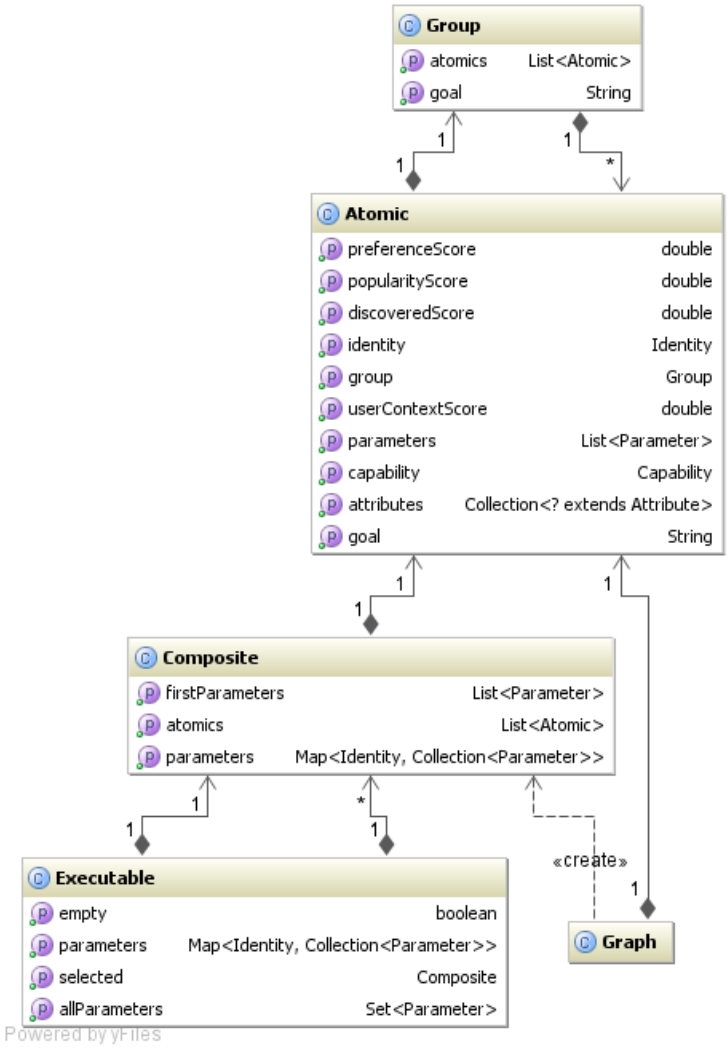
4. Paquete expresión.



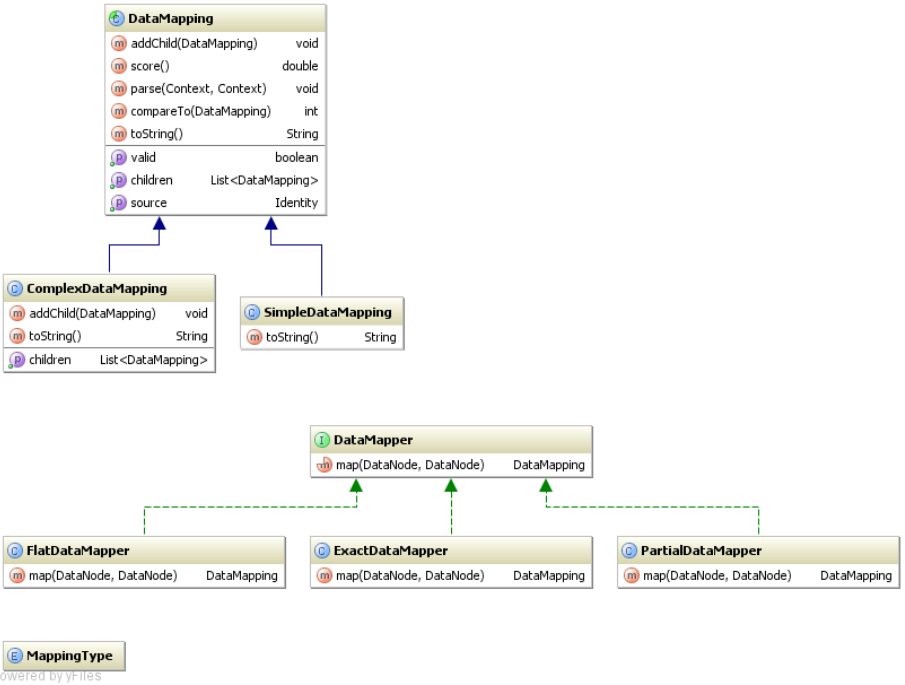
5. Paquete mapeo de datos



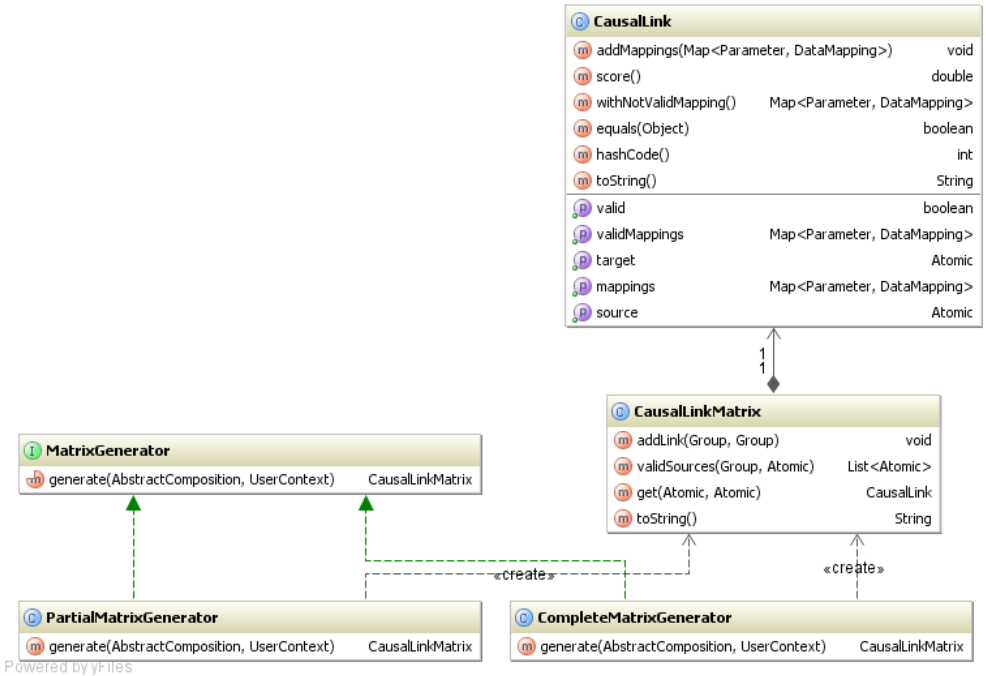
6. Paquete grafos



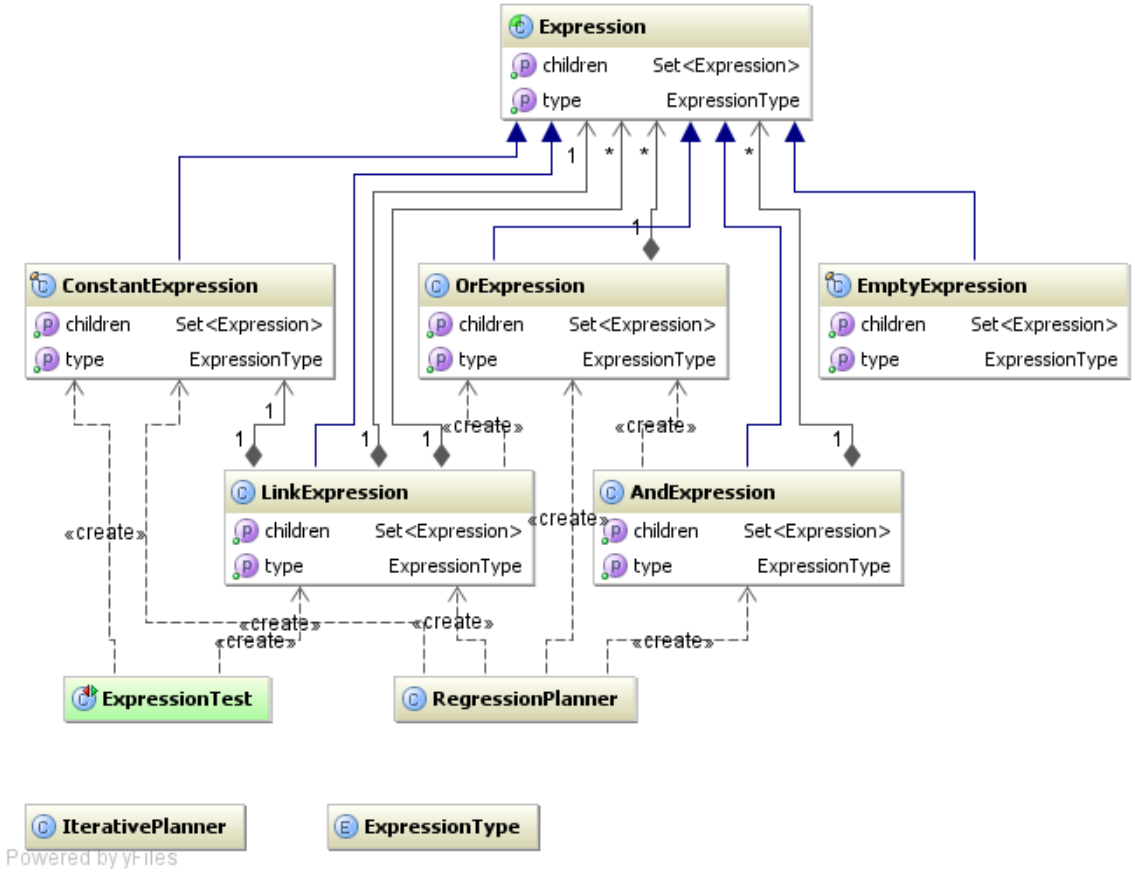
7. Paquete mapeo de datos



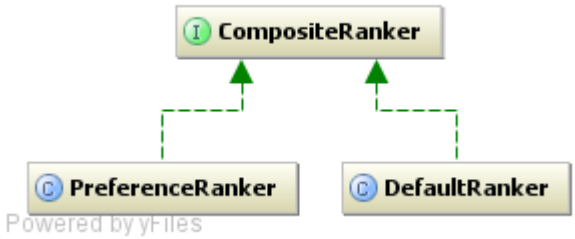
8. Paquete MEC



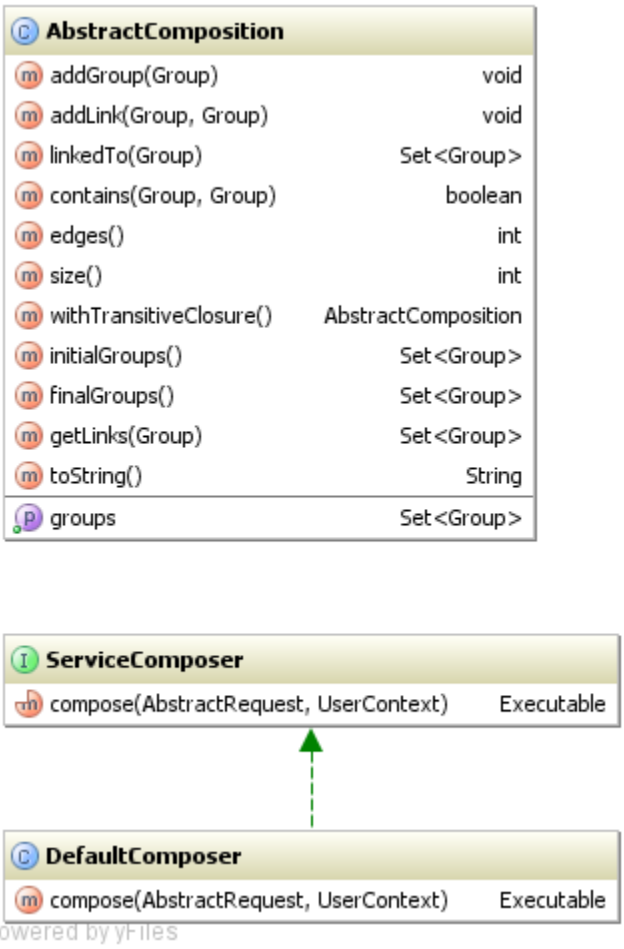
9. Paquete expresión planeador



10. Paquete clasificación

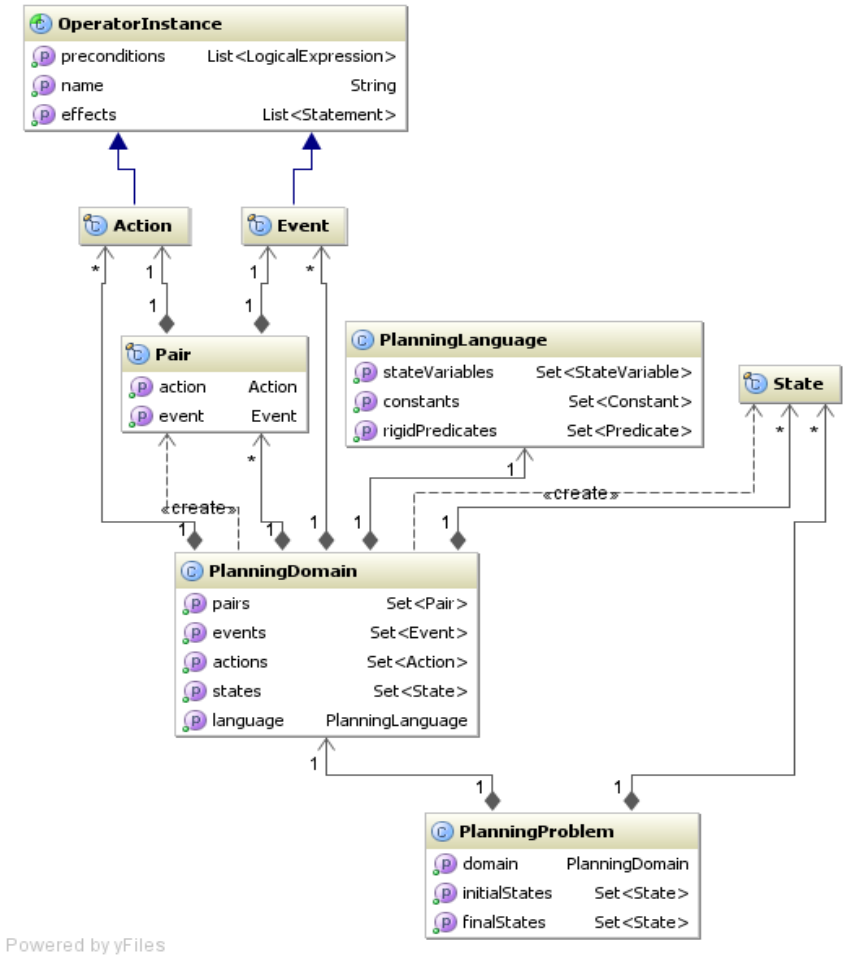


11. Paquete composición

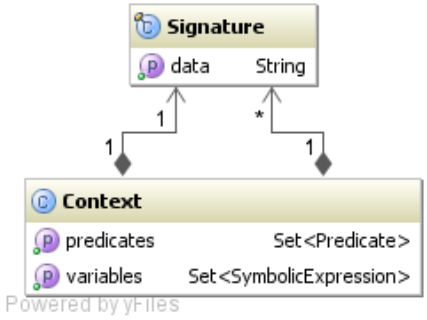


Modulo Composición STE

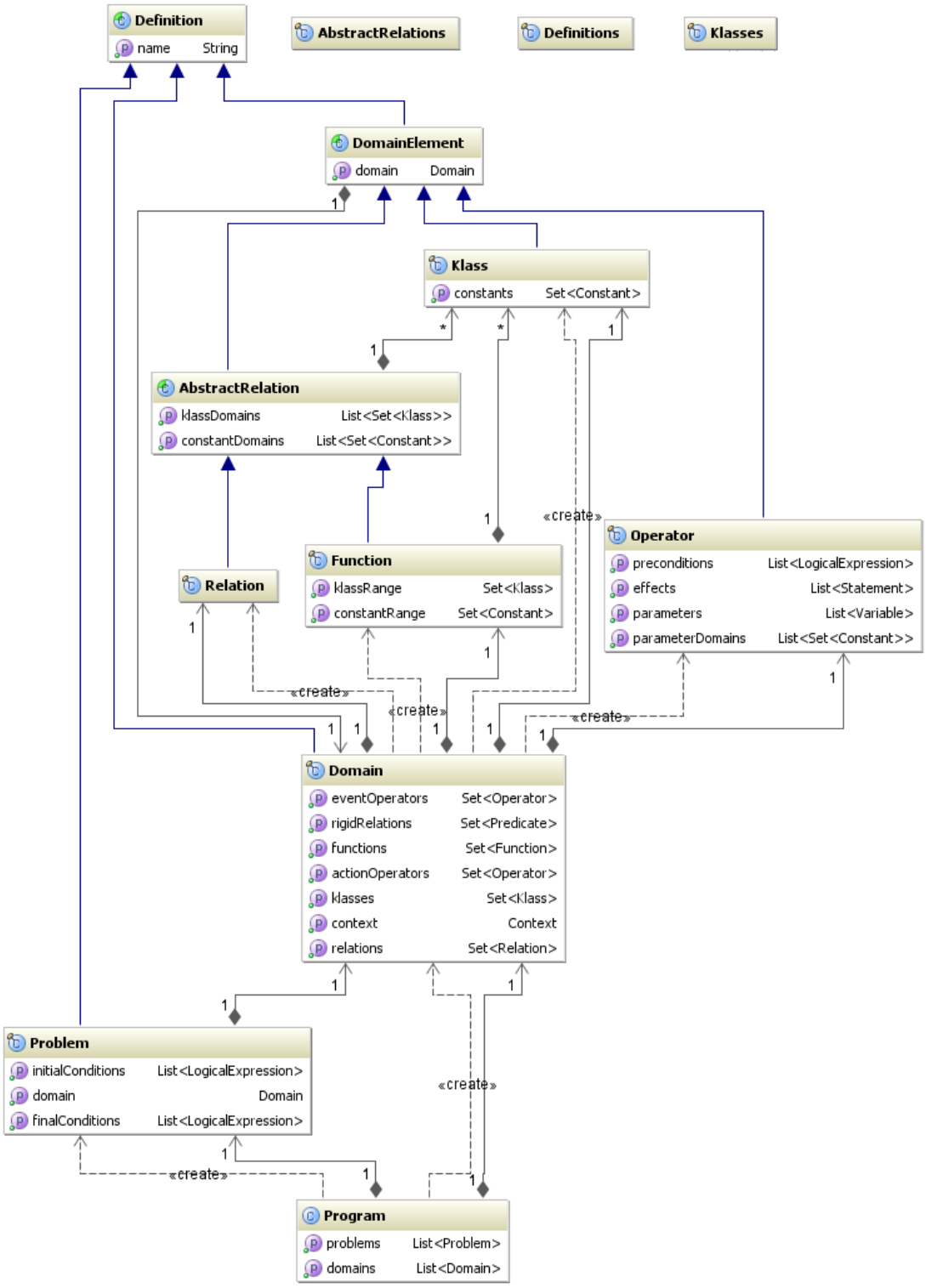
1. Paquete compilación



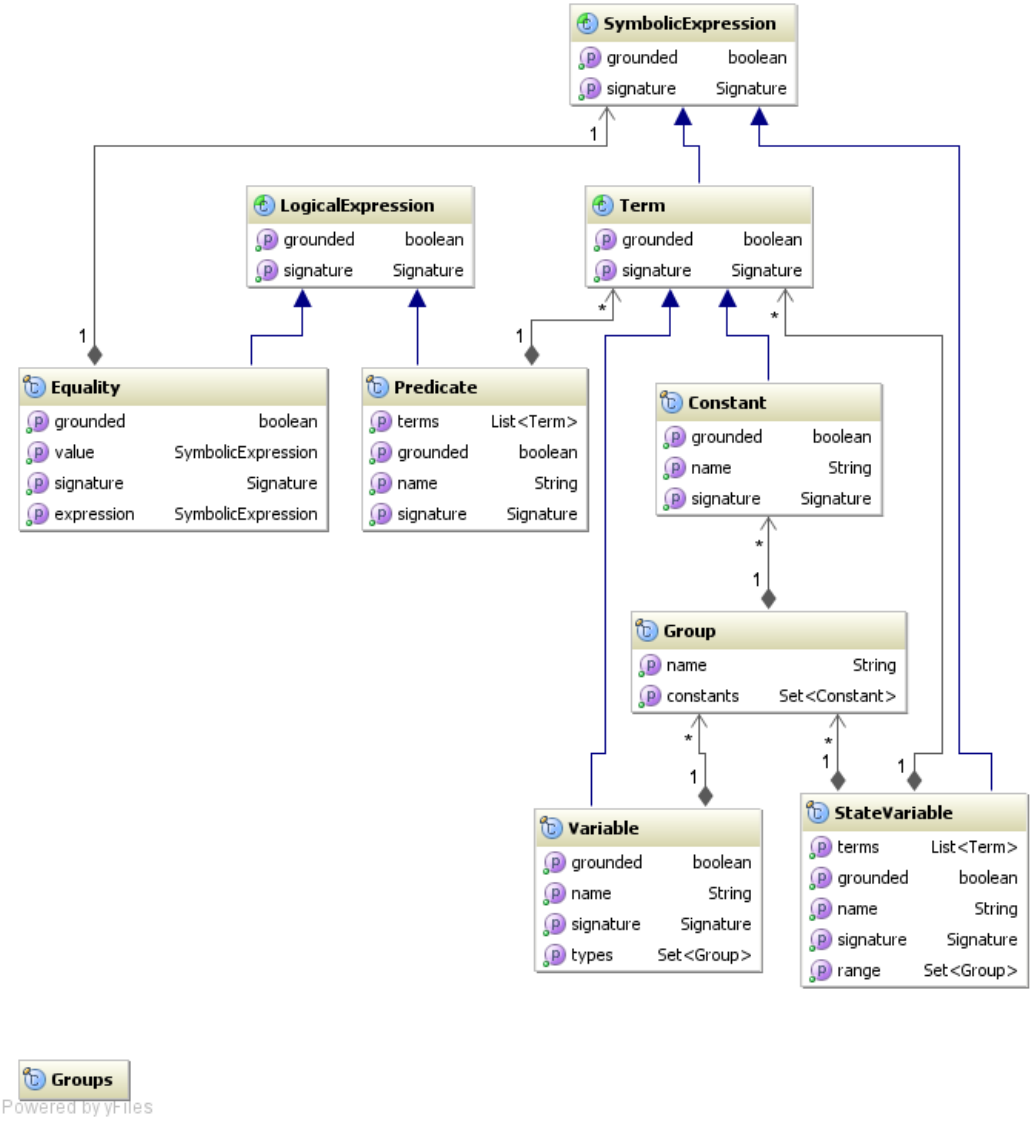
2. Paquete evaluación



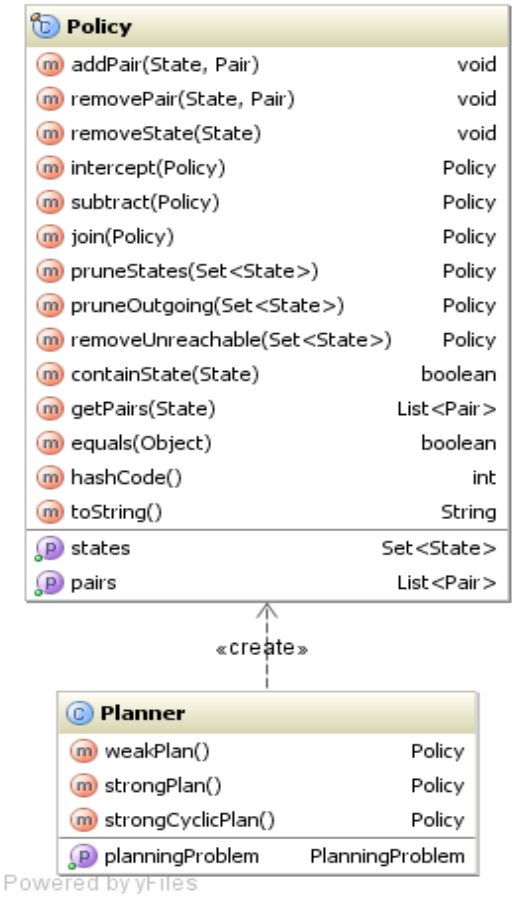
3. Paquete definición



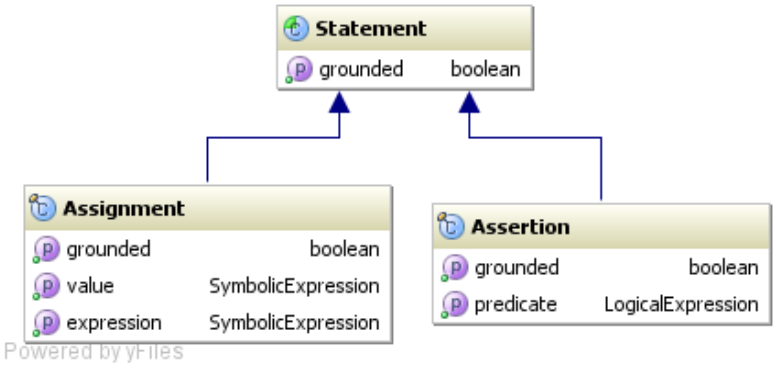
4. Paquete expresiones



5. Paquete solucionador

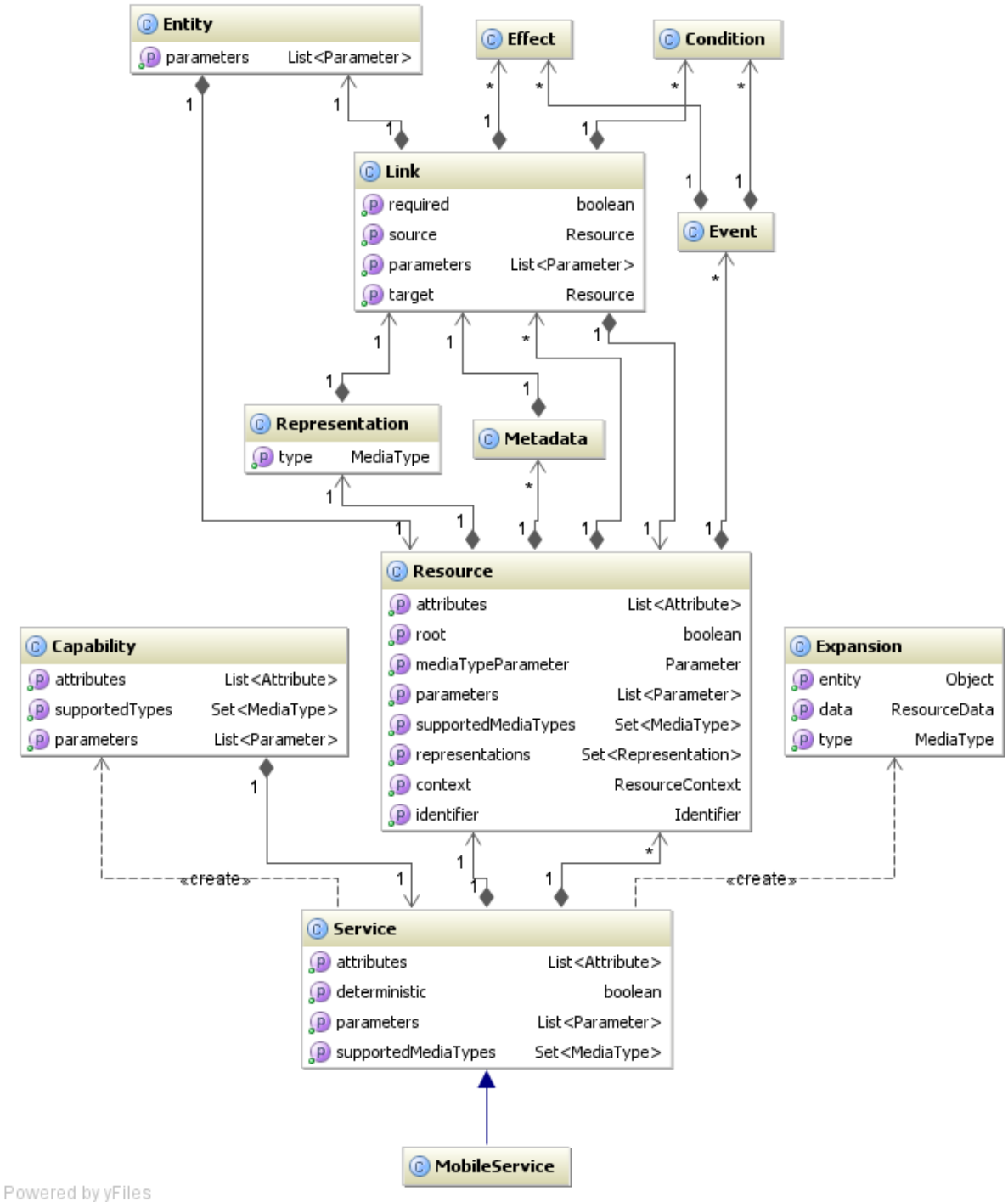


6. Paquete sentencias

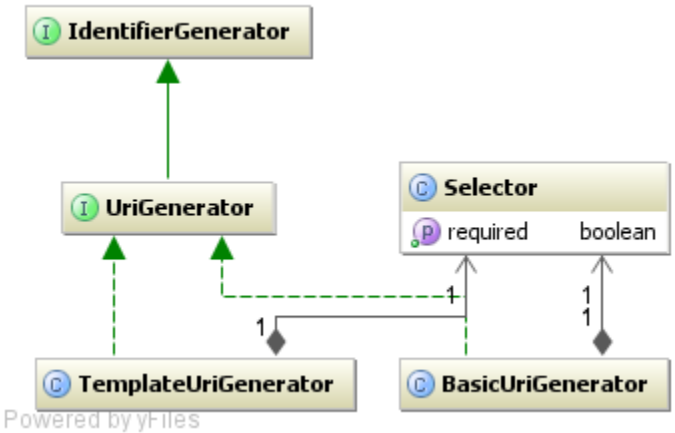


Modulo modelo RESTful

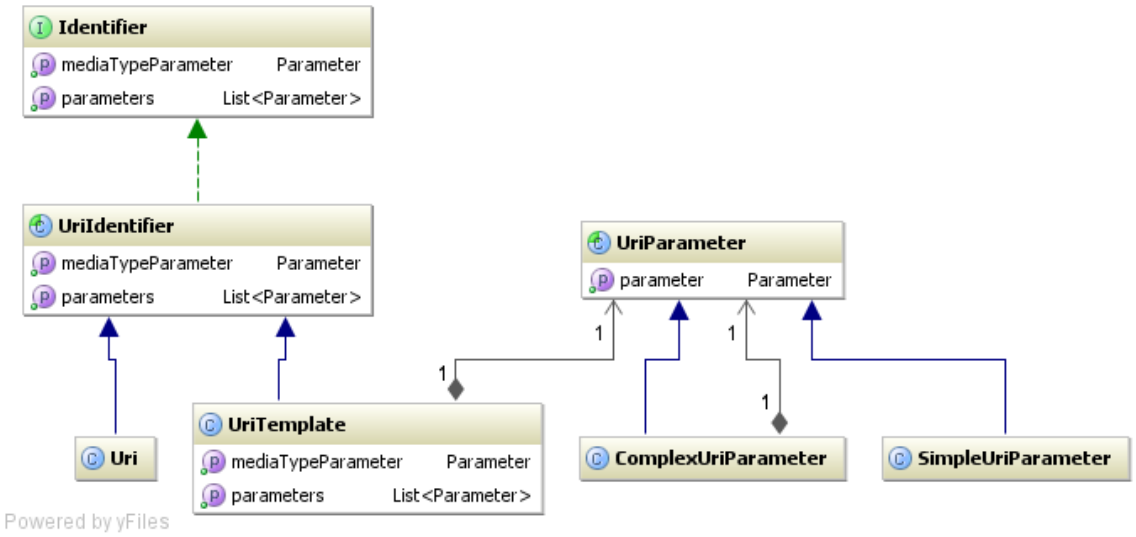
1. Paquete servicio RESTful



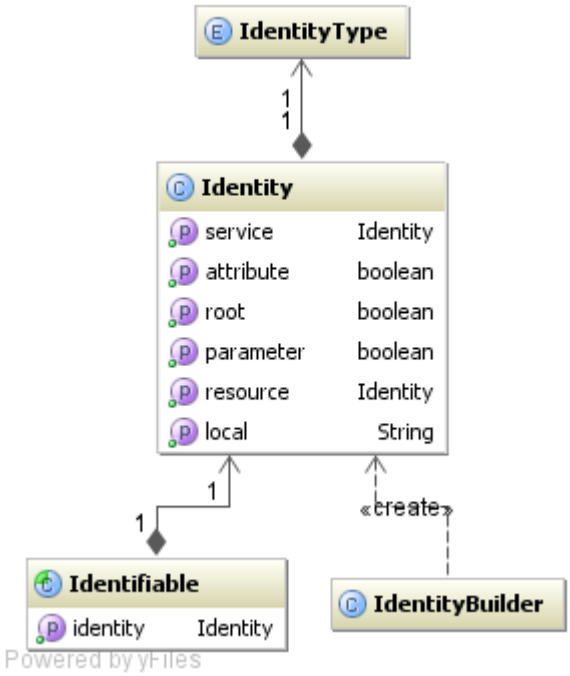
2. Paquete generador



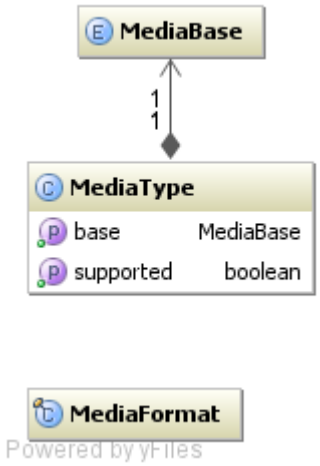
3. Paquete identificador



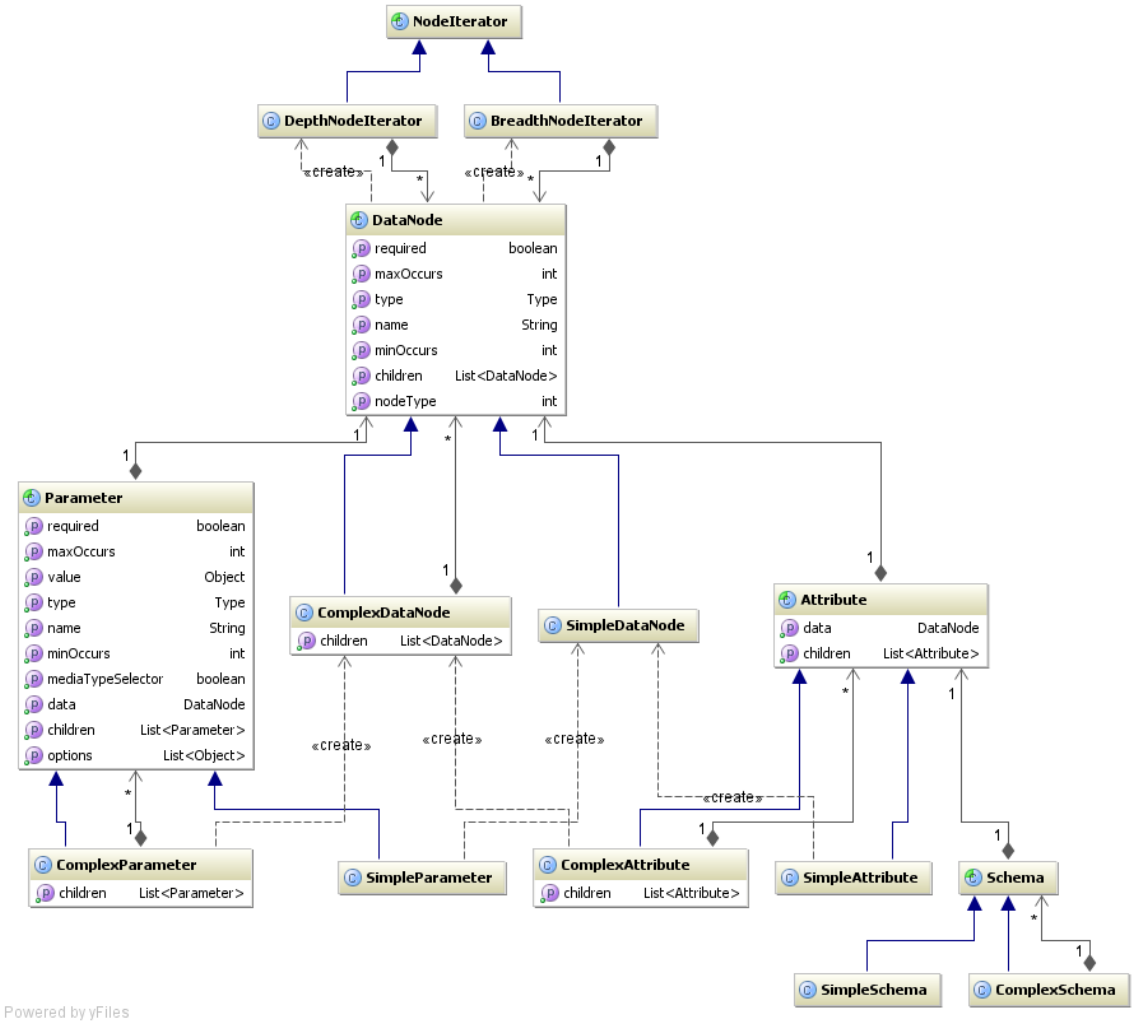
4. Paquete identidad



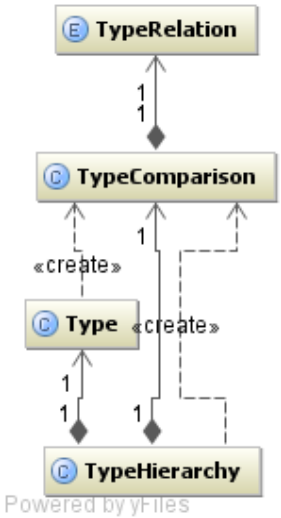
5. Paquete media



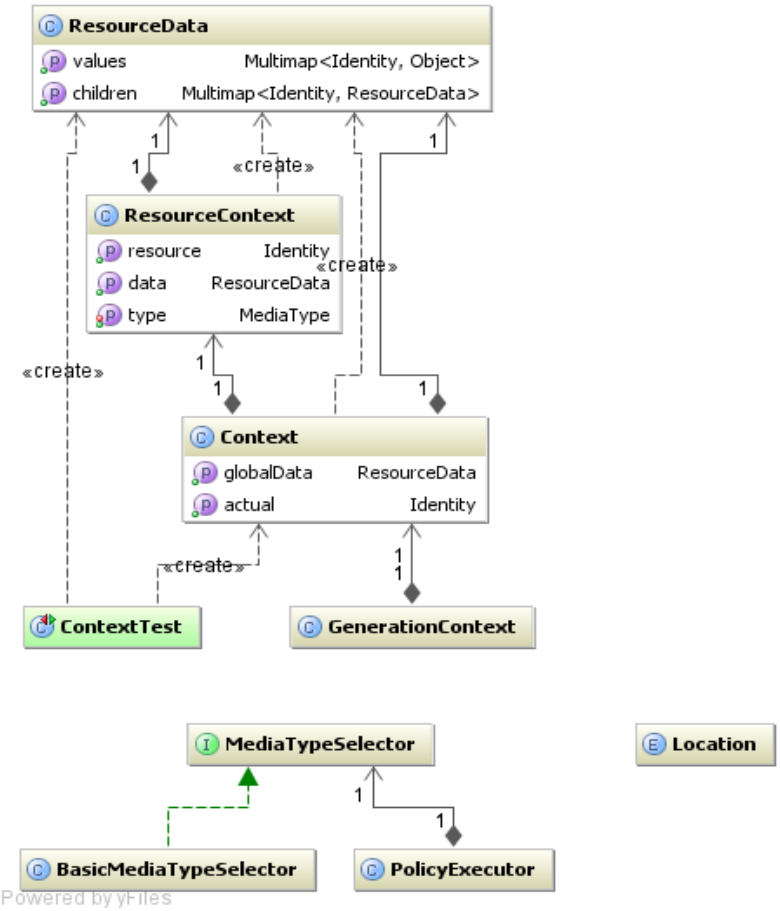
6. Paquete de datos



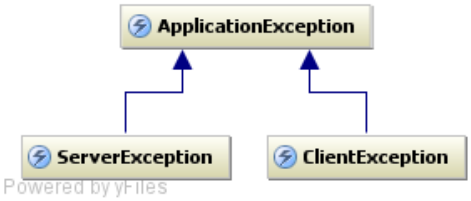
7. Paquete tipos de datos



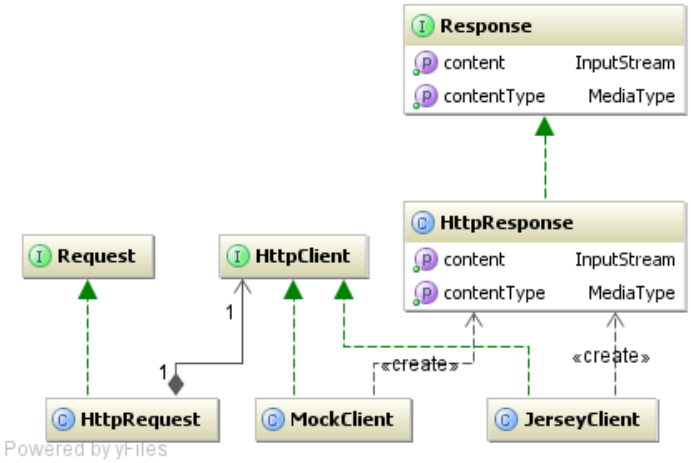
8. Paquete contexto



9. Paquete excepciones

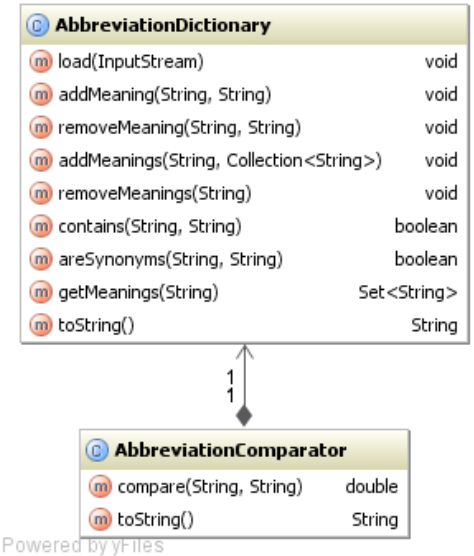


10. Paquete peticiones respuestas

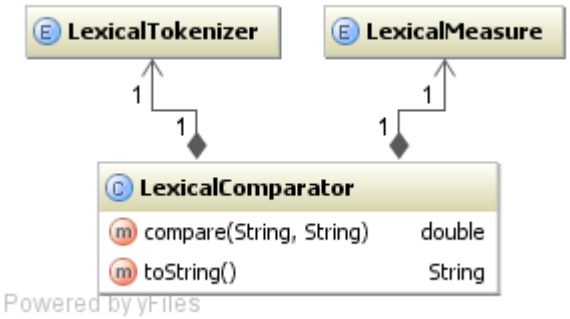


Modulo Similitud

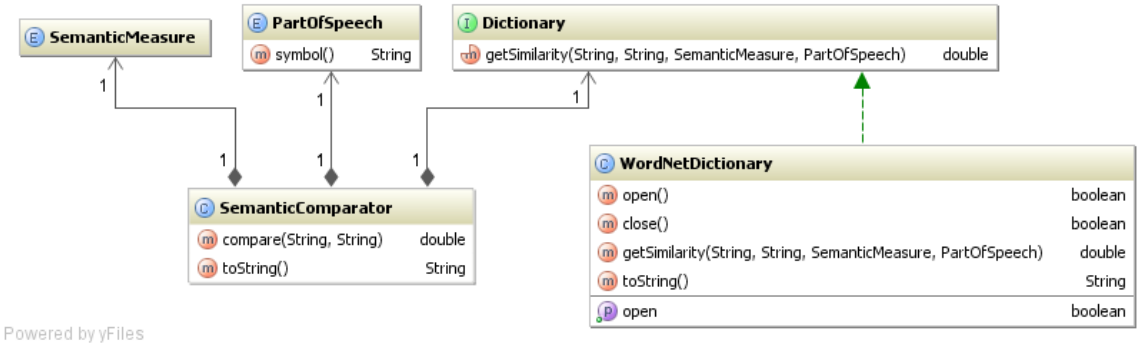
1. Paquete abreviaciones



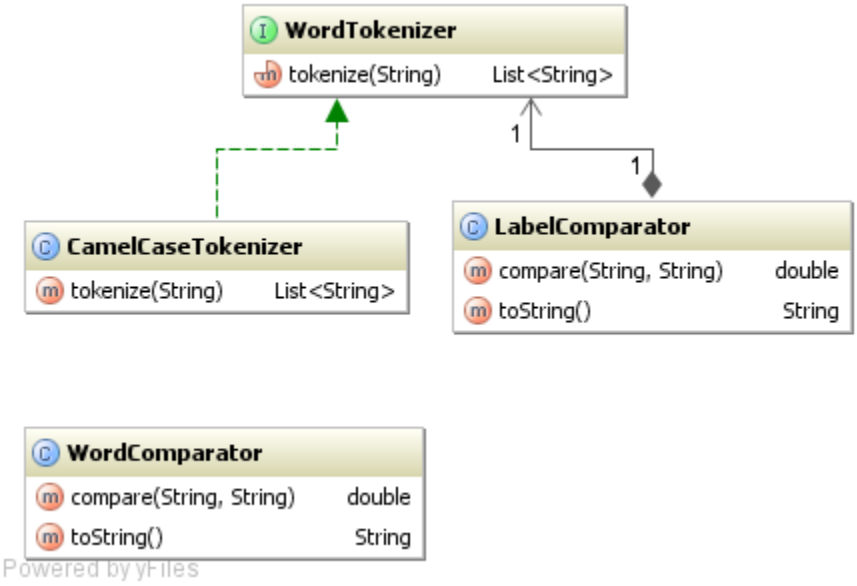
2. Paquete léxico



3. Paquete semántico

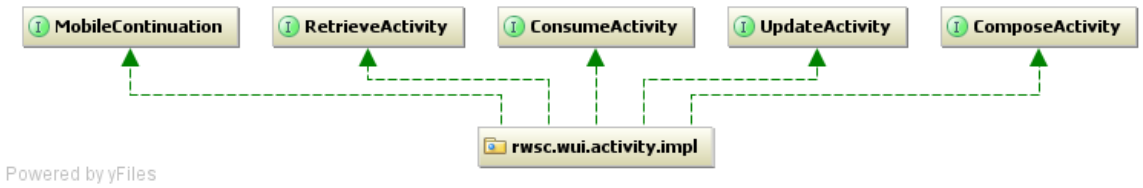


4. Paquete combinación

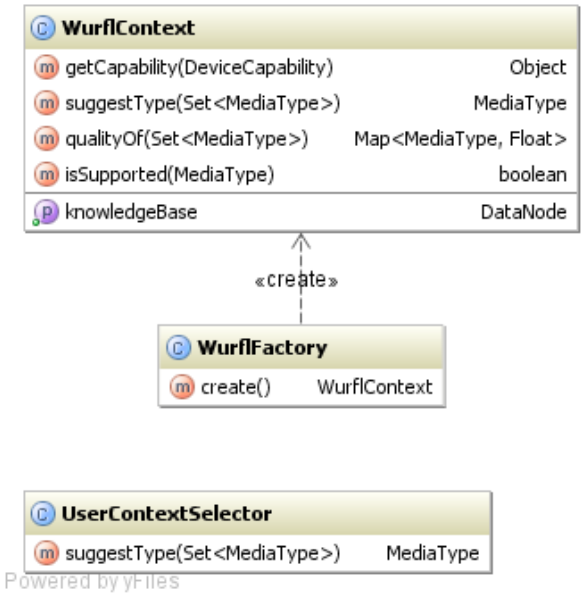


Modulo interfaz Web

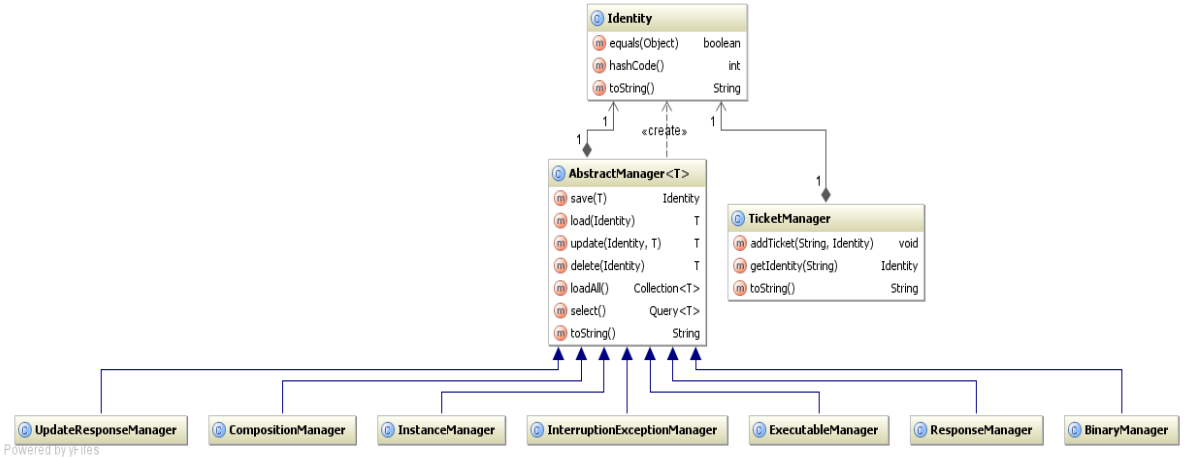
1. Paquete actividades



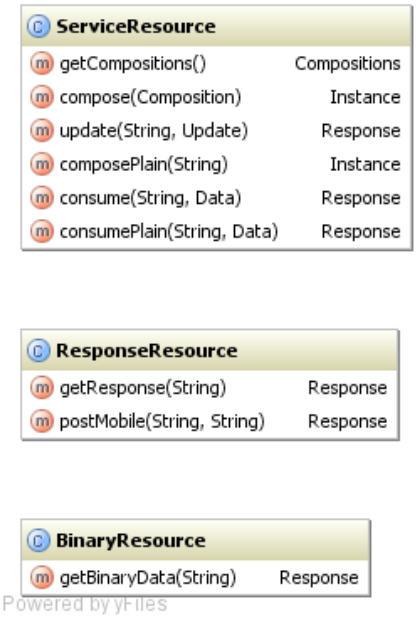
2. Paquete información dispositivo



3. Paquete persistencia

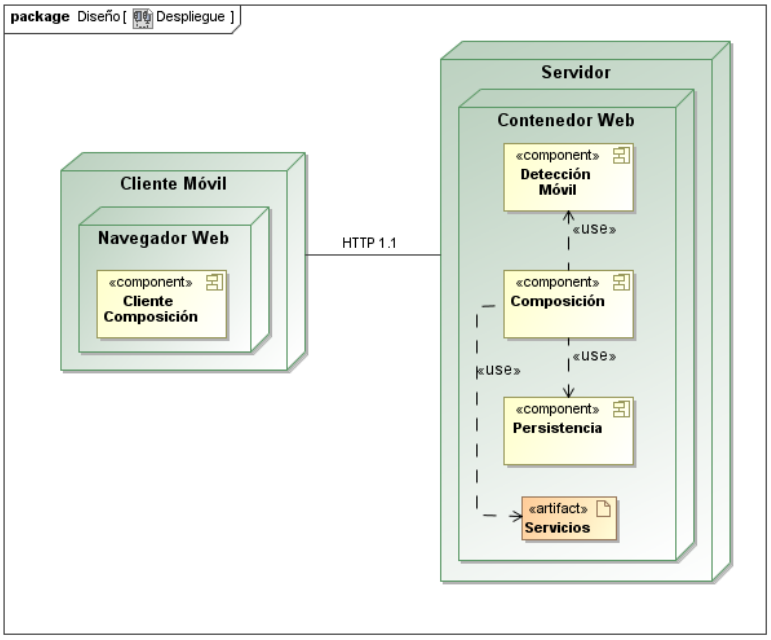


4. Paquete recursos



D.5. Diagrama de despliegue

La siguiente figura muestra el diagrama de despliegue de componentes del sistema, el cual describe la distribución de los módulos en cada uno de los componentes hardware del sistema.



A continuación se describen detalladamente cada uno de los nodos componentes del sistema.

D.5.1. Nodos

Nodo:	Cliente
Descripción:	Dispositivo móvil con navegador Web con soporte del lenguaje Java Script.

Nodo:	Servidor
Descripción:	Servidor Web Glassfish versión 3.1 edición de código abierto, soportado en un sistema operativo Windows 7 Ultimate de 32bit. Equipo de 2GB de memoria RAM, procesador Intel Core 2 Duo a 2GHz.

D.5.2. Componentes

Componente:	Cliente Composición
Descripción:	Componente desarrollado en el lenguaje Java Script, el cual genera las peticiones de composición al servidor, muestra las representaciones de respuesta y permite la interacción entre el usuario experto y el sistema.

Componente:	Detección Móvil
Descripción:	Este componente se encarga de determinar las características del dispositivo de un usuario a partir de información enviada por el navegador Web en las peticiones de la aplicación móvil. Está basado en Wurfl versión 1.3.1.1.

Componente:	Composición
Descripción:	Se encarga de los procesos de composición dinámica. Manipula las peticiones del usuario experto y retorna las representaciones de respuesta, obtenidas al invocar los diferentes servicios RESTful o móviles que se utilicen en una composición.

Componente:	Persistencia
Descripción:	Mantiene un conjunto de gestores de entidades en memoria, las cuales permiten almacenar los resultados de matrices, grafos o peticiones según el proceso realizado.

D.5.3. Artefactos

Artefacto:	Servicios
Descripción:	Conjunto de descripciones de servicios Web RESTful usados en la composición de servicios.

Anexo E

Características de WURFL y Clasificación de Formatos

Wurfl es una herramienta muy importante para generar servicios compuestos que se adapten a las características multimedia soportadas por el dispositivo móvil, pero es una base de datos muy extensa que posee características de miles de dispositivos, muchas de las cuales no son necesarias para los procesos relacionados con la composición dinámica de servicios Web RESTful en un entorno Mobile 2.0, a continuación se muestran las características especificas utilizadas por el algoritmo tanto para la clasificación en el ranking, como para la reconfiguración a causa de fluctuaciones en los niveles de velocidad de red.

E.1. Características WURFL

Característica	Descripción
mobile_browser	Identifica el cliente navegador desde el cual se accede al servidor.
mobile_browser_version	Cuál es la versión del navegador
resolution_width	Presenta el ancho de la pantalla del dispositivo móvil en pixeles
resolution_height	Presenta la altura de la pantalla del dispositivo móvil en pixeles
max_image_width	Este campo se encarga del tamaño máximo del ancho permitido para una imagen.
max_image_height	Este campo se encarga de la altura máxima permitida para una imagen.
has_cellular_radio	Muestra si el dispositivo soporta redes móviles celulares
max_data_rate	Presenta el ancho de banda soportado por el dispositivo (según la red móvil celular: HSDPA, UMTS,GPSR, EDGE,etc)
wbmp	Verifica si el dispositivo soporta el formato de imagen wbmp
bmp	Verifica si el dispositivo soporta el formato de imagen bmp
giff	Verifica si el dispositivo soporta el formato de imagen giff
jpg	Verifica si el dispositivo soporta el formato de imagen jpg
png	Verifica si el dispositivo soporta el formato de imagen png
tiff	Verifica si el dispositivo soporta el formato de imagen tiff
wav	Verifica si el dispositivo soporta el formato de audio wav
au	Verifica si el dispositivo soporta el formato de audio au

wma	Verifica si el dispositivo soporta el formato de audio wma
amr	Verifica si el dispositivo soporta el formato de audio amr
awb	Verifica si el dispositivo soporta el formato de audio awb
aac	Verifica si el dispositivo soporta el formato de audio aac
mp3	Verifica si el dispositivo soporta el formato de audio mp3
playback_real_media	Verifica si el dispositivo soporta el formato de video real media
playback_3gpp	Verifica si el dispositivo soporta el formato de video 3gpp
playback_3g2	Verifica si el dispositivo soporta el formato de video 3g2
playback_flv	Verifica si el dispositivo soporta el formato de video flv
playback_mp4	Verifica si el dispositivo soporta el formato de video mp4
playback_wmv	Verifica si el dispositivo soporta el formato de video wmv
playback_wov	Verifica si el dispositivo soporta el formato de video wov

E.2. Clasificación formatos multimedia

E.2.1. Clasificación de formatos de imagen

En la tabla se exponen los formatos de imagen utilizados para la reconfiguración de las representaciones del servicio compuesto en tiempo de ejecución, esta clasificación surge en base a http://www.library.cornell.edu/preservation/tutorial/presentation/table7-1.html y http://paleo.org/BKsamples/ .

Formato	Clasificación	Calidad
tiff	1	alta
png	2	alta
jpg	3	buena
gif	4	media
bmp	5	baja
wbmp	6	baja

E.2.2. Clasificación de formatos de audio

Para los servicios Web RESTful cuya representación está constituida por formatos de audio, se presenta la tabla que se encuentra más adelante en donde se clasifican de acuerdo a la calidad, el tamaño de sus archivos y la latencia de cada formato. Estos criterios son obtenidos de http://en.wikipedia.org/wiki/Comparison_of_audio_formats.

Formato	Clasificación	Grupo de Formato	Latencia
wav	1	Sin Compresión	
au	2	Sin Compresión	
wma	3	Compresión sin perdidas	>100ms
amr	4	Compresión con perdidas	25ms
awb	5	Compresión con perdidas	25ms
aac	6	Compresión con perdidas	20−405ms
mp3	7	Compresión con perdidas	>100ms

E.2.3. Clasificación de formatos de video

Finalmente para completar los formatos de media, se tiene la clasificación de videos soportados por los dispositivos móviles, que pueden ser parte de la representación final de un servicio compuesto. Tal clasificación se detalla en la siguiente tabla, los cuales son obtenidos, después de convertir tres videos en formato avi con diferentes tamaños, a los formatos requeridos, estableciendo configuraciones de video, audio y texto en base a las usadas por los dispositivos móviles.

Formato	Clasificación	Tamaño (AVI 2.7Mb)	Tamaño (AVI 8.25Mb)	Tamaño (AVI 21.2Mb)
mov	1	2,69	8,64	24,10
mp4	2	2,66	8,68	24,00
flv	3	2,58	7,56	18,50
wmv	4	2,30	7,83	18,10
Real media	5	2,15	6,86	16,90
3gpp	6	0,62	1,57	3,60
3g2	7	0,36	1,09	2,73

E.2.3. Clasificación de formatos con respecto a la velocidad de la red

Los formatos multimedia, se distribuyen según la tasa de transferencia en el enlace de bajada para todas las redes.inlambricas.la siguiente tabla muestra las redes móviles celulares más importantes y sus respectivas velocidades de menor a mayor, en donde ninguna presenta restricciones para el transporte de contenido, pero si son menos eficientes cuando se utilizan formatos pesados en redes bajas, o subutilizadas cuando se utilizan formatos livianos en redes altas.

Interfaz inalámbrica	Velocidad bajada (Kbps)	Formatos imagen	Formatos audio	Formatos video
GSM CSD	14.4	No aplica	No aplica	No aplica
HSCSD	57.6/14.4	gif, bmp, wmp	amr, awb, aac, mp3 muy livianos	No aplica
GPRS	57.6/28.8	gif, bmp, wmp	amr, awb, aac, mp3 livianos	No aplica
CDMA 2000	153	jpg, gif, bmp, wmp	amr, awb, aac, mp3	3gpp, 3g2 muy livianos
EDGE	236.8	jpg, gif, bmp, wmp	amr, awb, aac, mp3	3gpp, 3g2 muy livianos
UMTS	384	jpg, gif, bmp, wmp	amr, awb, aac, mp3	3gpp, 3g2

EDGE evol.	1.184/474	png,jpg, gif, bmp, wmp	wma, amr, awb, aac, mp3	Real media, 3gpp, 3g2
HSDPA	14.400/1.800	Todos (png y tiff recomendados)	Todos (wav, au y wma recomendados)	mp4 y mov livianos, flv, wmv, real media, 3gpp, 3g2
HSOPA	100.000	Todos (png y tiff recomendados)	Todos (wav, au y wma recomendados)	Todos (mov, mp4 y flv recomendados)
LTE(MIMO 2x2)	173.000/58.000	Todos (png y tiff recomendados)	Todos (wav, au y wma recomendados)	Todos (mov, mp4 y flv recomendados)
LTE(MIMO 4x4)	326.000/86.000	Todos (png y tiff recomendados)	Todos (wav, au y wma recomendados)	Todos (mov, mp4 y flv recomendados)

Vale la pena aclarar, que tanto el servicio Web como el dispositivo móvil deben soportar el formato multimedia que permite la red.