Arquitectura de Referencia para la Composición de Servicios en Ambientes de Computación Ubicua

Tesis de Maestría
Víctor Alberto Hermida Quintero

Director: Mag. Oscar Mauricio Caicedo Rendón

Universidad del Cauca
Instituto de Postgrados en Electrónica y Telecomunicaciones
Maestría en Ingeniería, Área Ingeniería Telemática
Departamento de Telemática
Línea de Investigación en Servicios Avanzados de Telecomunicaciones
Popayán, enero de 2011

Resumen

La computación ubica se ha establecido como la siguiente generación de las comunicaciones, ofreciendo a los usuario las capacidades de la redes en cualquier momento y lugar con el fin de soportar sus tarea cotidianas. Sin embargo, el desarrollo de sistemas software para los ambientes de computación ubicua requiere afrontar diversos retos, tales como la movilidad, el dinamismo y la heterogeneidad.

En este contexto, las arquitecturas orientadas a servicios ofrecen una alternativa apropiada para el diseño y despliegue de ambientes de computación ubicua, en la cual cada recurso es encapsulado como un servicio ubicuo. Generalmente, un servicio es relativamente sencillo mientras las tareas de usuario son más complejas y variables, de tal forma que es necesario combinar varios servicios para satisfacerlas. El mecanismo para combinar dos o más servicios y formar un nuevo servicio se denomina composición de servicios.

En este sentido, esta tesis de maestría introduce una arquitectura de referencia para soportar el descubrimiento y composición de servicios en ambientes de computación ubicua. Las principales contribuciones de esta tesis son: a) La arquitectura de referencia para la composición de servicios en ambientes ubicuos; b) una implementación de referencia, que muestra la instanciación de la arquitectura de referencia en un caso de estudio específico, detallando las tecnologías y herramientas empleadas para su implementación; c) las técnicas y algoritmos propuestos para abordar el problema del descubrimiento y composición de servicios en los ambientes de computación ubicua.

Tabla de contenido

Lista de Figuras

Lista de Tablas

Lista de Algoritmos

Capítulo 1

Introducción

1.1 Definición del Problema

La siguiente generación de comunicaciones será diferente a la actual. Se prevén cambios en el tipo de acceso y en los dispositivos empleados para interactuar con las redes. Se considera el empleo de terminales distribuidos, configurables dinámicamente y en proximidades a los usuarios que permitirán un acceso permanente a la información y a las capacidades computacionales, abriendo paso a la visión introducida por Weiser con el nombre de computación ubicua (Weiser, 1991). Según Weiser, la computación ubicua se describe como computadores muy pequeños con capacidad de comunicación y computación que se incrustan de forma casi invisible en cualquier tipo de dispositivo cotidiano. Estos dispositivos se encuentran por todas partes y se integran de forma amigable con los humanos, es decir, las personas interactúan con ellos de forma inconsciente (Almenárez, 2005).

Para lograr los objetivos de la computación ubicua se requiere abordar retos significativos, principalmente relacionados con la heterogeneidad del ambiente, el dinamismo y el contexto del usuario. La computación ubicua debe soportar las tareas de los usuarios por medio de la integración de las funcionalidades de los recursos de red, de tal forma que éstos puedan ser invocados en cualquier momento y lugar (El-Sayed, y otros, 2006). En este contexto, las arquitecturas orientadas a servicios ofrecen una alternativa apropiada para el diseño y despliegue de ambientes de computación ubicua, en la cual cada recurso es encapsulado como un servicio ubicuo. Alcanzar los objetivos mencionados es posible solo si se cuenta con mecanismos de descubrimiento de los servicios ubicuos que están presentes en la red y que son requeridos por el usuario.

En un escenario típico de descubrimiento de servicios, el cliente especifica su requisito tomando en cuenta tres aspectos fundamentales (Corrales, y otros, 2006): i) que el servicio sea capaz de realizar una cierta función (p.ej. crear un carrito de compras), ii) exponer una interfaz en particular (p.ej. adicionar un producto), iii) proporcionar un comportamiento determinado (p.ej. ignorar cualquier petición de remover productos cuando el carrito de compras este vacío). Sin embargo, es muy difícil encontrar exactamente los servicios que busca un usuario, la mayoría de las veces se requiere adaptar los servicios disponibles en la red ubicua o realizar procesos de composición de servicios.

Generalmente, un servicio es relativamente sencillo mientras las tareas de usuario son más complejas y variables, de tal forma que es necesario combinar varios servicios para satisfacerlas. El mecanismo para combinar dos o más servicios y formar un nuevo servicio se denomina composición de servicios. Esta composición puede ser realizada de una manera estática, durante la fase de despliegue, sin embargo, esta solución limita el posible uso de los recursos y no se adapta a las fluctuaciones propias de los ambientes ubicuos (Zhenghui, y otros, 2009).

Por otro lado, la composición dinámica busca combinar los servicios ubicuos disponibles en la red durante la fase de ejecución, brindando una mejor respuesta a los requisitos de heterogeneidad y dinamismo de los ambiente ubicuos. Adicionalmente, esta solución reduce los costos de desarrollo de componentes software ubicuos en términos de dinero, tiempo y recursos humanos, al generar nuevas funciones basándose en servicios ya existentes. De esta forma, la composición de servicios ubicuos se ha convertido en un área de profundo interés en los últimos años (Zhenghui, y otros, 2009).

Precisamente, uno de los objetivos más difíciles de alcanzar en los ambientes de computación ubicua es ayudar a los usuarios en la realización de sus tareas, en cualquier momento y en cualquier lugar, por medio de la composición de los servicios ofrecidos en la red ubicua (Ben Mokhtar, 06). Los dispositivos de la red deben tener la posibilidad de publicar sus servicios para que sean empleados por las aplicaciones cliente. Por otro lado, la red ubicua debe ser capaz de localizar e invocar los servicios necesarios para satisfacer los requisitos expresados por el usuario. Sin embargo, encontrar un servicio que cumpla exactamente con estas peticiones se convierte en un caso excepcional, la mayoría de las veces se requiere de una adaptación a los servicios disponibles en la red o un proceso de composición de diversas capacidades para ejecutar tareas complejas. De este modo, en contextos tan dinámicos y heterogéneos como los ambientes de computación ubicua es imprescindible contar con capacidades de descubrimiento y composición de servicios que permitan explotar los recursos (aplicaciones, archivos, almacenamiento, hardware, etc.) presentes en la red, ofreciendo la suficiente flexibilidad para reconfigurar los servicios de acuerdo a las variaciones del ambiente. Por tanto, el descubrimiento y composición de servicios deberá incorporar las características necesarias para ser tan dinámicos y autónomos como las condiciones de la red ubicua lo requieran. De esta manera, surge la siguiente pregunta de investigación: ¿Cómo soportar la composición autónoma y dinámica de servicios dentro de los ambientes ubicuos?

En el siguiente apartado se presentan dos ejemplos que evidencian la necesidad de contemplar los procesos de composición en los ambientes de computación ubica.

1.2 Escenarios de Motivación

A continuación se presentan dos situaciones que requieren de la composición de servicios en un ambiente de computación ubicua. La primera, se ubica en el contexto de e-salud[1], mientras la segunda hace referencia al contexto de la información turística.

Aplicaciones de E-salud: algunos años atrás, Iván fue diagnosticado con una Diabetes Tipo II. Con el fin de controlar el nivel de azúcar en su sangre, porta un dispositivo de monitoreo, que censa regularmente su condición. Supongamos que Iván se encuentra en un viaje de negocios en la ciudad de Cartagena y que después de un día ocupado se siente indispuesto. Su terminal de e-salud identificará una situación anormal y la reportará a su médico en Bogotá. El médico recibirá el mensaje enviado por el terminal, revisará el estado actual de Iván y la historia clínica respectiva; todo a través de un servicio web, con el propósito de tomar una decisión sobre la condición reportada por el sistema. El médico podrá ordenar algunos exámenes de laboratorio o prescribir una formula. Por su parte, Iván recibirá la respuesta de su médico por medio de su teléfono móvil, con las correspondientes instrucciones para tomar el tratamiento recetado. En el escenario

[1]E-salud se define como la aplicación de las tecnologías de la información y la comunicación (TIC) en el amplio rango de aspectos que afectan el cuidado de la salud

descrito, el ambiente ubicuo deberá localizar e invocar los servicios necesarios para enviar en tiempo real los reportes de salud del paciente y poder encadenar las operaciones necesarias para adelantar los exámenes y obtención de los medicamentos recetados por el médico.

Información turística: supongamos que un usuario necesita: i) un servicio de información turística que ofrezca información relacionada con los sitios de interés y restaurantes ubicados en la ciudad objeto de su visita. ii) Un reporte del clima para poder programar su recorrido en la ciudad. En este escenario, el turista realiza su petición por medio de una aplicación móvil y el ambiente de computación ubicua buscará los servicios de información sobre sitios turísticos, restaurantes y predicción del clima para poder crear un servicio compuesto que responda a sus exigencias.

1.3 Objetivos

Objetivo General

Proporcionar una arquitectura de referencia para la composición autónoma de servicios en ambientes de computación ubicua.

Objetivos Específicos

- Definir las técnicas y algoritmos necesarios para realizar el descubrimiento y composición de servicios dentro de los ambientes de computación ubicua.
- Diseñar una arquitectura que proporcione las funcionalidades necesarias para soportar el despliegue y composición de servicios dentro de una ambiente de computación ubicua.
- Desarrollar prototipos que permitan realizar la evaluación experimental de la efectividad y eficiencia de los enfoques adoptados en la arquitectura para la composición de servicios de información turística dentro de un ambiente de computación ubicua.

1.4 Contribuciones de la Tesis y Estructura del Documento

Las contribuciones más importantes del presente trabajo de grado de maestría están estructuradas de la siguiente manera a lo largo de este documento:

En el capítulo 2, se presenta la visión de la computación ubicua y las principales características de estos ambientes. Posteriormente, se introduce SOA (Arquitectura Orientada a Servicios) como un estilo arquitectónico útil para el desarrollo de ambientes de computación ubicua, resaltando la importancia que cobran las funcionalidades de descubrimiento y composición de servicios en la realización de la visión de los ambientes ubicuos. A continuación, se realiza una revisión de los trabajos relacionados con el descubrimiento y composición de servicios, de igual manera se revisa el uso de las representaciones formales en el desarrollo de estas dos funcionalidades. Los resultados obtenidos en este capítulo son: a) Un análisis y valoración de los mecanismos de descubrimiento de servicios, apuntando a seleccionar una técnica apropiada para los ambientes ubicuos; b) Un análisis y valoración de los mecanismos de composición de servicios, orientados a escoger una técnica apropiada para los ambientes ubicuos; c) Un análisis y valoración de las representaciones formales que pueden ser empleadas para modelar las técnicas y algoritmos de descubrimiento y composición de servicios.

En el capítulo 3 se presenta la arquitectura de referencia que proporciona el presente trabajo de grado para lograr la composición de servicios en ambientes de computación ubicua. Se definen los módulos que componen la arquitectura y la forma como se relaciona para constituir un ambiente ubicuo que proporciona las funcionalidades de descubrimiento y composición. Finalmente, se presenta una implementación de referencia, que corresponde a una instanciación de la arquitectura de referencia, la cual se empleó como caso de estudio para analizar el comportamiento de las soluciones propuestas para el descubrimiento y composición de servicios en ambientes ubicuos. Los resultados obtenidos en este capítulo son: a) Una arquitectura de referencia que proporciona las funcionalidades para la composición de servicios en ambientes ubicuos; b) La arquitectura de la implementación de referencia, incluyendo las herramientas y tecnologías empleadas para instanciar la arquitectura de referencia.

El capítulo 4 contiene la técnica y los algoritmos propuestos para el descubrimiento de servicios en los ambientes de computación ubicua. Se presenta una solución basada en el descubrimiento de servicios atómicos, en la cual se aplica BPEL. La solución se sustenta en una representación de grafos para capturar el comportamiento de los procesos de negocio y los atributos de las actividades básicas y de control que los componen. Finalmente, se presenta una evaluación de la calidad y rendimiento de los algoritmos de descubrimiento presentados, las pruebas se realizaron sobre la implementación de referencia presentada en el capítulo 3. Los resultados obtenidos en este capítulo son: a) Los algoritmos y técnicas de descubrimiento de servicios basados en una representación de grafos, incorporando aspectos relevantes para los ambientes ubicuos como el contexto de entrega; b) La valoración de la calidad y rendimiento de los algoritmos de descubrimiento en un entorno de pruebas basado en la arquitectura de la implementación de referencia.

El capítulo 5 contiene la técnica y los algoritmos propuestos para la composición de servicios en los ambientes de computación ubicua. Se presenta una solución basada en la integración de servicios atómicos guiada por el comportamiento expresado en un proceso de negocio BPEL, la composición usa los grafos como representación formal y se sustenta en la solución de descubrimiento presentada en el capítulo 4. Finalmente, se realizó una evaluación del rendimiento de los algoritmos de composición presentados, las pruebas se realizan sobre la implementación de referencia presentada en el capítulo 3. Los resultados obtenidos en este capítulo son: a) Los algoritmos y técnicas de composición de servicios basados en una representación de grafos; b) La valoración del rendimiento de los algoritmos de composición en un entorno de pruebas basado en la arquitectura de la implementación de referencia.

Los principales resultados se han publicado o están en proceso de evaluación para su publicación en eventos o revistas indexadas de la siguiente manera:

- Discovery and Composition of Services in Ubiquitous Environments: artículo presentado en el Cuarto Congreso Colombiano de Computación 4CCC 2009, su contenido se encuentra en el Anexo D.
- Plataforma para Descubrimiento de Servicios en Ambientes Ubicuos: artículo presentado a la Revista Facultad de Ingeniería Universidad de Antioquia, ISSN 0120-6230. El contenido del artículo se puede encontrar en el Anexo E.
- Plataforma para la Composición de Servicios en Ambientes Ubicuos: artículo que será presentado a evaluación a una revista nacional indexada, Ingeniería y Competitividad de la Universidad del Valle ó Ingeniería y Universidad de la Pontificia Universidad Javeriana. El artículo se puede encontrar en el Anexo F.

Capítulo 2

Computación Ubicua

2.1 Ambientes de Computación Ubicua

Se puede afirmar que el ánimo de la computación ubicua es copar el ambiente con dispositivos electrónicos capaces de asistir a los humanos en el desarrollo de sus tareas diarias, de una manera casi natural y poco intrusiva. En un escenario de estos, los usuarios podrán expresar sus deseos o necesidades y el ambiente se configurará de manera autónoma para satisfacerlos. Las personas interactuarán con una gran cantidad de dispositivos inteligentes de una forma totalmente transparente, sin tener conciencia de la presencia o localización de los dispositivos, y sin manejar interfaces específicas y complejas (Bottaro, y otros, 2007).

Los avances en los dispositivos hardware, los sistemas software y las redes de comunicaciones han permitido que se creen condiciones propicias para alcanzar el ideal de la computación ubicua. Actualmente, se pueden encontrar productos comerciales como computadores portátiles, dispositivos móviles de tercera generación, además de sensores, tecnologías inalámbricas de comunicación, redes de área personal y redes de área corporal que pueden brindar el soporte necesario para la construcción de una ambiente de computación ubicua. La esencia de la computación ubicua es que el ambiente se poblé con facilidades de computación y comunicación integradas a las actividades de los humanos (Satyanarayanan, 2001).

La Figura 1 muestra varios ambientes de computación ubicua (el hogar, la oficina, el automóvil) poblados con diversos dispositivos de red (equipos móviles, computadores, pantallas, sistemas GPS, etc.) La naturaleza de los dispositivos puede ser muy amplia, se pueden encontrar tanto equipos con altas prestaciones de recursos y rendimiento (computadores de escritorio o portátiles) como dispositivos más limitados (PDA, equipos móviles, sensores en el hogar). Estas diferencias entre los dispositivos pueden ser expresadas en términos de procesamiento, memoria, capacidad de almacenamiento, potencia de la batería, ancho de banda. Los dispositivos ubicuos pueden ser estacionarios (servidores multimedia, pantallas de LCD) o móviles (PDA o sistemas embebidos en un automóvil) y proveer funcionalidades hardware o software al ambiente ubicuo, denominada como una funcionalidad ubicua (funcionalidad de desplegar un video en una pantalla o un servidor multimedia). Los dispositivos pueden estar conectados permanente o temporalmente al ambiente ubicuo, el cual puede estar constituido por una red heterogénea incluyendo redes alambradas (WAN, LAN, conexión ADSL) y/o inalámbricas (PAN, Bluetooth, WiFi).

La Figura 1 ilustra un usuario que se desplaza de un ambiente a otro, portando un dispositivo ubicuo, lo cual demanda que los ambientes ubicuos posean capacidades de configuración autónoma para poder crear aplicaciones ubicuas que soporten las tareas que ejecuta el usuario. La configuración autónoma de un ambiente ubicuo no es del todo sencilla, principalmente por las características propias de la computación ubicua que imponen altos retos tecnológicos. Entre las más sobresalientes se encuentran (Almenárez, 2005) (Jaroucheh, y otros, 2009):

1. Invisibilidad. Weiser propone dispositivos "invisibles" que interactúan con los usuarios sin que estos tengan que preocuparse de cómo hacerlo, más aún, dicha interacción ocurre por debajo del nivel de conciencia de las personas.
2. Dinamismo. La libertad de movimiento de los usuarios y el uso de tecnologías inalámbricas hacen que estos ambientes sean altamente dinámicos al permitir que nuevos dispositivos se agreguen o abandonen a la red.
3. Espacios colaborativos. La arquitectura para entornos ubicuos está basada en dispositivos autónomos que interactúan colaborativamente en el espacio físico para dar soporte a los usuarios presentes en la red.
4. Escalabilidad local. Los usuarios disponen de capacidades según el contexto en el que se encuentren, por lo tanto, el concepto de servicios locales es muy importante en computación ubicua, ya que hay servicios que pierden todo sentido cuando el usuario se encuentra fuera de un entorno determinado.
5. Enlace transparente entre redes. De acuerdo con la red disponible, la distribución de los servicios ofrecidos puede variar, por lo tanto, estos servicios deben ofrecerse de forma continua entre las distintas redes por donde transite el usuario. Existe entonces una alta heterogeneidad de las tecnologías integradas en términos de la red, dispositivos e infraestructuras software.

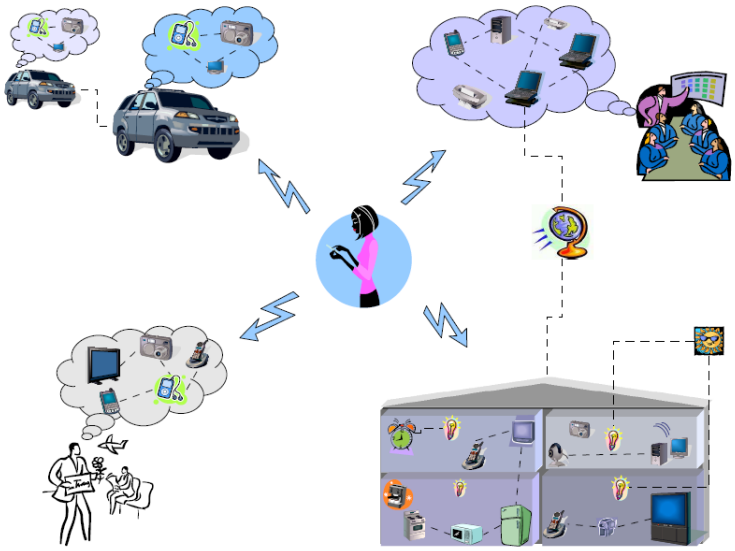

Figura 1. Ambientes de Computación Ubicua

2.2 SOA y la Computación Ubicua

La Arquitectura Orientada a Servicios (SOA) es un estilo arquitectural para el desarrollo de sistemas altamente autónomos y con bajo acoplamiento, capaces de comunicarse, componerse y evolucionar en ambientes abiertos, dinámicos y heterogéneos tales como los ambientes ubicuos (Georgantas, y otros, 2005). Este estilo arquitectural ha sido empleado ampliamente en trabajos relacionados con el descubrimiento y composición de servicios en ambientes de computación ubicua (Ibrahim, y otros, 2010) (Ibrahim, y otros, 2009) (Tigli, y otros, 2009) (Ben Mokhtar, 2007) (Kalasapur, y otros, 2007).

SOA está basado en tres roles principales: el Proveedor del Servicio, asumido por una entidad que ofrece un servicio; Cliente del Servicio, entidad que desea consumir el servicio ofrecido; y el

Registro de Servicio, rol asumido por la entidad que mantiene la información acerca de los servicios disponibles y la forma de accederlos. En (Papazoglou, 2003) se introdujo un nuevo rol dentro de SOA denominado Agregador de Servicio, también llamado Mediador de Composición en (Ibrahim, y otros, 2009), su función es la de componer nuevos servicios a partir de los existentes y ofrecerlos a las aplicaciones clientes. El Mediador de Composición actúa como un Proveedor, ofreciendo los servicios compuestos, y como un Cliente al consumir los servicios existentes. En la Figura 2 se ilustran las relaciones existentes entre los roles de SOA.

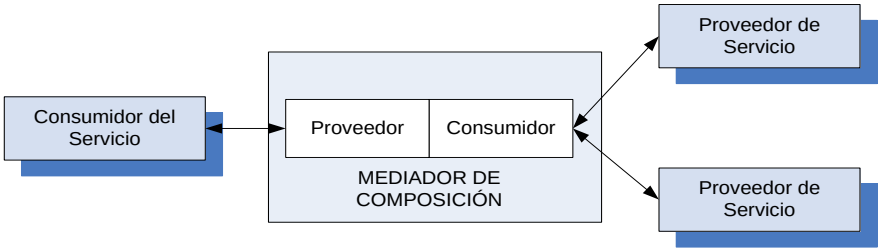

Figura 2. Roles de SOA

La Figura 3 describe en detalle las funciones desempeñadas por cada rol de SOA y sus relaciones. En (Ben Mokhtar, 2007) se describen los elementos conceptuales de SOA de la siguiente manera:

- Publicación del Servicio: permite a los proveedores registrar sus servicios en un registro de servicios.
- Localización de Servicio: permite a los clientes encontrar los servicios deseados en el registro de servicios.
- Emparejamiento de Servicios: funcionalidad ofrecida por el registro de servicios, permite seleccionar los servicios que mejor respondan a las peticiones realizadas por los clientes.
- Acceso a los Servicios: permite a los clientes establecer una conexión con los servicios seleccionados.
- Composición de Servicios: permite la integración de múltiples servicios para conformar un servicio compuesto.

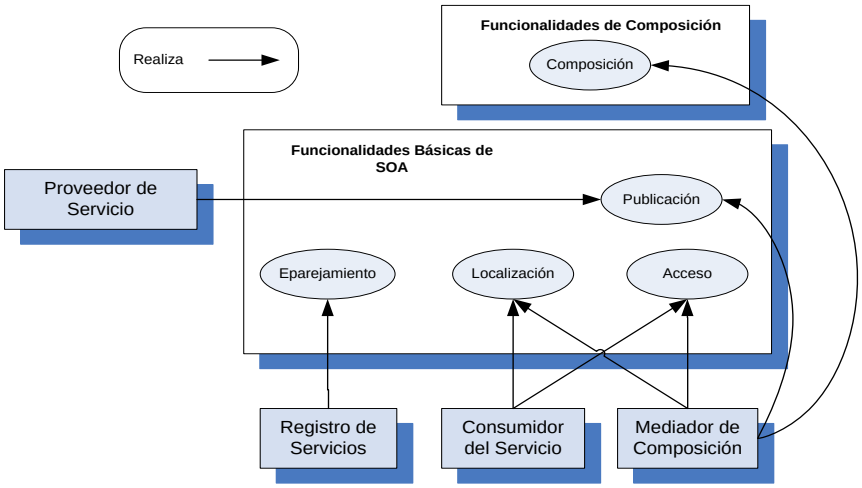

Figura 3. Elementos Conceptuales de SOA

Una vez identificados los elementos conceptuales que conforman el paradigma de SOA se puede realizar una abstracción de estos componentes para soportar el diseño de sistemas ubicuos. En (Ben Mokhtar, 2007) se presenta una abstracción que relaciona los conceptos de SOA con los conceptos involucrados en los ambientes ubicuos, esta abstracción se resume en la Tabla 1.

Tabla 1. Relación entre los Conceptos de SOA y los Ambientes Ubicuos

Computación Ubicua	SOA
Dispositivo Ubicuo	Proveedor del Servicio
Funcionalidad Ubicua	Capacidad de Servicio
Tarea de Usuario	Servicio Compuesto
Usuario	Cliente
Anuncio de Funcionalidad	Publicación del Servicio
Identificación de Funcionalidades	Localización del Servicio
Realización de Tareas de Usuario	Composición de Servicios

Los Proveedores de Servicios agrupan a los dispositivos ubicuos y a las aplicaciones software que proveen las capacidades necesarias para realizar las tareas de usuario. Los dispositivos ubicuos realizan los anuncios de funcionalidad para alertar a los demás dispositivos de las capacidades que ofrecen, estos avisos corresponden a una Publicación de Servicio. Por otra parte, los usuarios del ambiente ubicuo corresponden a un Cliente SOA, y son ellos quienes deben localizar las funcionalidades ofrecidas en el ambiente ubicuo para realizar sus tareas. Las tareas de usuario a su vez son abstraídas como un Servicio Compuesto, que integra las funcionalidades ofrecidas por los dispositivos presentes en la red.

Para la creación autónoma de Servicios Compuestos en las redes ubicuas, la composición deben poseer las siguientes características (Ben Mokhtar, y otros, 2005):

1. Ser dinámica, de acuerdo a los componentes disponibles en un lugar y tiempo.
2. Satisfacer los requisitos funcionales de una manera eficiente dentro de los límites impuestos por las restricciones de recursos de los dispositivos móviles.
3. Adaptarse a la heterogeneidad de las tecnologías.

Las características en mención, exigen que los ambientes ubicuos posean las capacidades necesarias para descubrir e integrar los servicios disponibles en la red, independientemente de la movilidad y heterogeneidad de los dispositivos.

2.3 Mecanismos para el Descubrimiento de Servicios

Los ambientes de computación ubicua se caracterizan por la diversidad y movilidad de las entidades dentro de la red. De esta manera, uno de los mayores retos en la computación ubicua es soportar las tareas de usuario por medio de la integración de las funcionalidades de los servicios de red, de tal forma que puedan ser invocados en cualquier momento y lugar (El-Sayed, y otros,

2006). Para esto se necesita contar con mecanismos de descubrimiento que permitan recuperar los servicios requeridos por el usuario y que están presentes en la red ubicua.

El descubrimiento de servicios puede ser definido como la capacidad de encontrar y utilizar posteriormente un servicio basado en alguna descripción publicada de su funcionalidad y parámetros operacionales (Bandara, y otros, 2007). Se puede clasificar en tres conjuntos (Corrales, 2008): descubrimiento basado en interfaces, descubrimiento semántico y descubrimiento basado en comportamiento.

2.3.1 Descubrimiento de Servicios basado en Interfaces y Semántica

El descubrimiento basado en interfaces de servicio utiliza técnicas de comparación de las palabras clave que las describen. Regularmente las técnicas de comparación de palabras clave requieren coincidencias exactas a nivel sintáctico entre las descripciones de los servicios, lo cual puede resultar en que se descarten servicios equivalentes a nivel lógico (p.ej. dos servicios descritos como Impresora e Impresión pueden ser distintos sintácticamente pero equivalentes lógicamente) (Corrales, 2008).

La descripción semántica de servicios pretende que las máquinas puedan realizar un mayor procesamiento y razonamiento sobre los servicios. La principal diferencia con las descripciones sintácticas es la forma como se representa la información; mientras la sintáctica se enfoca en definir los servicios a partir de los mensajes de entrada y salida, tipos y partes de los mensajes, la semántica busca ofrecer información acerca de la funcionalidad del servicio (Ben Mokhtar, 2007) (Corrales, 2008). La representación semántica del contenido de las descripciones de un servicio permite a las maquinas entender y procesar su contenido, soportando el descubrimiento e integración dinámica de los servicios (Ben Mokhtar, 2007).

En (Sellami, y otros, 2008) (Steller, y otros, 2009) se presentan técnicas de descubrimiento de servicios basadas en comparaciones semánticas dentro de ambientes de computación ubicua, estos trabajos proponen extender la WSDL (*Web Service Description Language*) añadiendo atributos a las descripciones para soportar las comparaciones semánticas. En (Zhang, y otros, 2009) se propone un algoritmo de emparejamiento semántico de servicios web bipartito, empleando anotaciones semánticas que se agregan a la WSDL y al registro UDDI, esta solución fue probada en el dominio de las aplicaciones médicas remotas. Estas aproximaciones al trabajar sobre la WSDL de los servicios no permiten soportar procesos posteriores de composición o adaptación de los servicios recuperados en la red ubicua. Modelos como OWL-S (Ontology Web Language for Services) y WSMO (Web Service Modeling Ontology) que soportan una descripción semántica de los servicios han sido empleados por (Ben Mokhtar, y otros, 2006)(Sayed y otros, 09), permitiendo establecer una descripción más completa de servicios web para propósitos de composición. En (Ibrahim, y otros, 2010) se presenta una técnica de descubrimiento semántico inspirada en las redes sociales humanas, el trabajo propone crear una red social de servicios, en la cual se crean grupos de servicios que posean una interfaz equivalente a nivel semántico, de tal forma que cada grupo de servicios publica una interfaz común pero posee múltiples implementaciones de la interfaz con diferentes propiedades funcionales. Esta agrupación busca reducir el tiempo de consulta en el descubrimiento de servicios.

Los resultados presentados por (Ben Mokhtar, y otros, 2006b) demuestran que la complejidad computacional de los razonadores semánticos es inapropiada para el uso en ambientes de computación ubicua. Los experimentos se realizaron utilizando tres razonadores muy populares

como Racer, FaCT++ y Pellet, la sobrecarga de procesamiento se multiplicó proporcionalmente con el número de servicios comparados con el servicio requerido y el número de ontologías utilizadas para describir los servicios.

En (van Bommel, y otros, 1999) y (Krall, y otros, 1997) se utiliza una codificación que representa la jerarquía de conceptos con el fin de reducir el tiempo necesario para establecer las relaciones entre los conceptos de una ontología. De esta manera, se puede establecer la relación de dos conceptos comparando los códigos numéricos asignados a ellos en lugar de recorrer la jerarquía completa. Estas soluciones implican manejar los problemas típicos de la representación de conceptos, tales como poder generar una codificación libre de conflictos para ontologías considerablemente extensas.

Otras soluciones de descubrimiento de servicios (Constantinescu, y otros, 2003) (Ben Mokhtar, 2007) combinan la codificación de los conceptos y técnicas de indexación para la organización de los registros obteniendo una mejora en los tiempos de respuesta requeridos para las búsquedas. Sin embargo, en estas soluciones la inserción de nuevos conceptos aún sigue siendo un proceso pesado, que requiere un alto grado de procesamiento durante la publicación de los servicios, además, se incrementa sustancialmente la complejidad en la implementación de los registros donde se publican y consultan los servicios ubicuos.

2.3.2 Descubrimiento de Servicios basado en Comportamiento

El descubrimiento de servicios basado en comparaciones sintácticas o semánticas no es suficiente para un gran número de aplicaciones. La tendencia en los trabajos recientes es explotar más el conocimiento sobre los componentes de los servicios y su comportamiento. La necesidad de tener en cuenta el comportamiento de los servicios descritos por medio de un modelo de proceso fue resaltada por diversas investigaciones (Bansal, y otros, 2003) (Bernstein, y otros, 2002) (Wombacher, y otros, 2004). En (Bernstein, y otros, 2002) se mejora la precisión del descubrimiento de servicios web empleando los modelos de proceso para capturar el comportamiento saliente de un servicio.

En (Wombacher, y otros, 2004) los autores brindan una semántica formal al emparejamiento de proceso de negocio basados en autómatas de estados finitos, agregando expresiones lógicas a cada uno de los estados. Sin embargo la complejidad computacional es muy alta y no es muy escalable para repositorios con muchos servicios. En (Wombacher, y otros, 2005) se agrega un índice basado en sistemas de base de datos tradicionales que reduce la complejidad computacional. Por otro lado, la representación de autómata de estados finitos limita la expresividad de los modelos, siendo su principal problema la ejecución en paralelo de diferentes capacidades ya que genera modelos demasiado extensos.

El trabajo de (Shen, y otros, 2005) emplea una representación de autómatas finitos no-determinísticos para modelar el comportamiento de los servicios web y OWL-S para describir el perfil de las actividades de cada servicio, logrando capturar las propiedades semánticas y temporales del comportamiento de los servicios. Estas soluciones no ofrecen un resultado positivo en casos donde no se encuentra un modelo que corresponda exactamente al requerido por el usuario. Además, asumen que los servicios operan con una semántica común para describir los mensajes intercambiados, lo cual limita la efectividad al momento de comparar la similitud entre dos servicios.

En (Corrales, y otros, 2008) se presenta una técnica de emparejamiento que opera sobre modelos de comportamiento y que permite estimar la distancia entre los modelos almacenados en los repositorios de servicios y el modelo requerido por el usuario. Esta solución permite obtener los modelos más similares que se tengan en los repositorios, aún si no existe uno que concuerde exactamente con el requerido por el usuario. Las técnicas son empleadas en el dominio de los servicios web y probados bajo dos protocolos BPEL y WSCL. La limitante de trabajo se concentra en el costo de procesamiento del algoritmo de emparejamiento, ya que el tiempo de comparación crece exponencialmente ($O(m^2n^2)$) cuando se aumenta el número de actividades que conforman el modelo de consulta enviado por el usuario, obteniendo tiempos muy dilatados para modelos extensos (con más de 20 actividades).

En resumen, la necesidad de tener en cuenta el comportamiento de los servicios en los proceso de descubrimiento fue evidenciada por varias investigaciones recientes, sin embargo la complejidad propia de una búsqueda que toma en cuenta la estructura de los procesos hace que su aplicabilidad a los ambientes ubicuos no sea viable, más aún cuando se desea agregar capacidades semánticas, las cuales aumentan los costos de procesamiento y tiempo.

2.3.3 El Contexto y el Descubrimiento de Servicios

Los sistemas ubicuos no solo deben adaptarse al cambio de los recursos disponibles, sino también a los cambios de perfiles y preferencias de los usuarios a través del tiempo y espacios físicos. Esta capacidad usualmente se conoce como percepción del contexto (*context-awareness*) (Issarny, y otros, 2007).

El contexto se constituye en un elemento central para la computación ubicua ya que posibilita la entrega de aplicaciones a los usuarios finales de una manera oportuna y con la mejor calidad posible. La percepción del contexto para la computación ubicua se puede definir como (Rocha, y otros, 2008): la capacidad de un sistema o mediador de proveer acceso en cualquier momento a información heterogénea, distribuida y sin anticipo acerca del contexto en una escala global y sobre diferentes escenarios.

En (Wang, 2003) se identifica el conjunto de elementos funcionales que debe poseer un sistema para soportar una adecuada percepción del contexto:

- **Adquisición del contexto**: concerniente a los mecanismos para obtener los datos del contexto a partir de las diferentes fuentes (usuarios, dispositivos, proveedores).
- **Modelado del contexto:** constituye la base para compartir e interpretar la información adquirida del contexto.
- **Agregación del contexto:** basado en el modelo del contexto, la agregación permite unir la información recolectada desde diferentes fuentes y obtener una mejor interpretación del contexto, evitando una excesiva carga de consultas en los sistemas distribuidos.
- **Interpretación del contexto**: los datos recolectados desde las fuentes debe ser interpretados para derivar en un entendimiento de más alto nivel y posibilitar que los servicios puedan emplear la información extraída sobre el contexto.
- **Consulta del contexto**: abarca los mecanismos necesarios para tener acceso a la información acerca del contexto y que se encuentra esparcida sobre repositorios distribuidos.

En los trabajos de (El-Sayed, y otros, 2006) (Ben Mokhtar, y otros, 2006b) (Steller, y otros, 2006) (Steller, y otros, 2009) se extiende a OWL-S para incluir una descripción semántica de la

información contextual tanto estática como dinámica asociada a los ambientes de computación ubicua. Esta información es coordinada con el motor de emparejamiento para seleccionar los servicios más adecuados de acuerdo a la condición actual del ambiente y el usuario. En el enfoque de (El-Sayed, y otros, 2006) se toman como fuentes de información del contexto tres elementos: el proveedor de servicios (dispositivos ubicuos), el usuario y los servicios, mientras en los trabajos (Steller, y otros, 2006) (Steller, y otros, 2009), la información del contexto es obtenida desde dos fuentes: el usuario y sus dispositivos.

El trabajo de (Henricksen, y otros, 2002) propone un modelado del contexto para ambientes de computación ubicua, el cual se construye a partir de tres entidades fundamentales: las personas, los dispositivos de comunicación y los canales de comunicación. El modelo propuesto busca ofrecer una base formal para representar y razonar acerca de la información relativa al contexto.

En (Dasgupta, y otros, 2009) se define una ontología para capturar los efectos que producen sobre los ambientes ubicuos los servicios y los eventos generados por la ejecución de un proceso desde la perspectiva del contexto. Además, define un conjunto de dimensiones a partir de las cuales se puede capturar la información del contexto de un servicio, dicha información es empleada para estimar la similitud de dos servicios y poder tener más elementos de juicio al momento de valorar el uso de un servicio en un proceso compuesto.

Los trabajos anteriores muestran la necesidad de incorporar información del contexto dentro de las funciones de emparejamiento de servicios para obtener una mayor eficiencia y efectividad en el descubrimiento de servicios, al tiempo que se obtienen sistemas más acordes a las características de los ambientes ubicuos. Coinciden en utilizar como fuentes de información del contexto tres elementos básicos: el usuario, los dispositivos y los servicios, punto de partida para el diseño de los demás módulos funcionales que debe contener un sistema de descubrimiento basado en el contexto. De acuerdo a esto, el presente trabajo de maestría tiene en cuenta estas tres fuentes de información para incorporar la información del contexto de entrega a los mecanismos de descubrimiento y composición de servicios.

2.3.4 Valoración de las Técnicas de Descubrimiento

La Tabla 2 resume la valoración que se realizó de las técnicas de descubrimiento presentadas en la sección 2.3, esta comparación esta soportada en el trabajo de (Corrales, 2008). El símbolo (+) indica que la propiedad evaluada es soportada por la técnica, mientras que el símbolo (-) significa que no es soportada. El símbolo (+/-) señala que la propiedad es soportada de una manera moderada.

Los parámetros evaluados para cada técnica son: stateful, stateless, semántica y tiempo de respuesta. La propiedad Stateful indica que el computador o programa es capaz de mantener el estado de las interacciones, usualmente empleando campos de almacenamiento diseñados para este propósito. La propiedad Stateless se utiliza para programas que no mantienen el estado de interacciones previas, el procesamiento de las peticiones se realiza únicamente con la información que viene incluida en el requerimiento.

En la Tabla 2 se pude apreciar que el descubrimiento basado en interfaces no soporta servicios stateful ya que en este caso los motores de búsqueda solo pueden emparejar servicios atómicos y no manejan aspectos relacionados con el proceso. En esta primera técnica de descubrimiento se pueden incluir algunas características semánticas extendiendo los lenguajes de descripción

empleados con anotaciones especiales como lo realizan (Sellami, y otros, 2008) (Steller, y otros, 2009). El tiempo de respuesta es moderado, ya que las comparaciones sintácticas son más rápidas que las realizadas por los razonadores semánticos.

Tabla 2. Valoración de las Técnicas de Emparejamiento

Técnicas de Emparejamiento	Parámetros			
	Stateless	**Stateful**	**Semántica**	**Tiempo de Respuesta**
Basado en Interfaces	+	-	+/-	Moderado
Basado en Semántica	+	-	+	Alto
Basado en Comportamiento	-	+	+/-	Alto

El descubrimiento basado en la semántica es aplicado también en servicios stateless, es una técnica muy efectiva para encontrar servicios simples, ya que soporta la comparación de las capacidades a un nivel lógico, sin embargo por sí sola no incorpora las características necesarias para realizar una comparación de procesos complejos conformados por varios servicios. Su tiempo de respuesta es muy alto, teniendo en cuenta el costo computacional que demandan los razonadores semánticos.

El descubrimiento basado en comportamiento soporta tanto servicios stateless como stateful, dependiendo de la descripción formal utilizada, es capaz de recuperar estructuras más complejas y ejecutar una búsqueda a nivel de modelos de servicios. Soporta comparaciones semánticas tal como lo realiza (Shen, y otros, 2005), o lingüísticas como en (Corrales, 2008). En cuanto al tiempo de respuesta, esta técnica tiene el desempeño más pobre, ya que la complejidad de los algoritmos empleados es muy alta, por tanto, es poco apropiada para su aplicación en la computación ubicua.

Teniendo en cuenta la valoración realizada en esta sección, se determina que el descubrimiento de servicios basado en comportamiento demanda un alto costo de procesamiento, por lo que en este trabajo de maestría se utiliza una búsqueda de servicios atómicos, propendiendo por unos tiempos de respuesta y procesamiento más acordes a las restricciones de los ambientes ubicuos.

En cuanto a las técnicas de comparación se evidencia la necesidad de abordar una aproximación semántica para lograr un emparejamiento más acertado, sin embargo el costo de procesamiento de los razonadores semánticos constituye una limitante significativa en el contexto de aplicación. Por este motivo, en este trabajo de grado de maestría se optó por una aproximación lingüística, como la empleada en (Corrales, 2008), en la cual se utiliza un analizador que calcula la similitud lingüística entre dos etiquetas basándose en sus nombres. Para obtener esta medida se utilizan los algoritmos Ngram, Check synonym y Check abbreviation. El algoritmo Ngram estima la similitud de acuerdo al número común de qgramas entre las etiquetas (Angell, y otros, 1983). El algoritmo Check synonym utiliza el diccionario lingüístico Wordnet (Miller, 1995), para identificar sinónimos, mientras el algoritmo Check abbreviation (Corrales, 2008) usa un diccionario de abreviaciones adecuado al dominio de aplicación. Por medio de esta solución, se obtiene un nivel de comparación superior al sintáctico, enriqueciendo la comparación de las palabras para medir su

similitud lingüística, aportando como gran ventaja su mejor desempeño y tiempo de respuesta respecto a los razonadores semánticos.

2.4 Mecanismos para la Composición de Servicios

La composición de servicios puede ser definida como la combinación de múltiples servicios en un único servicio compuesto, la cual puede ser lograda en la etapa de diseño (composición estática) o en tiempo de ejecución (composición dinámica) (Ibrahim, y otros, 2010). La composición de servicios se presenta en situaciones donde las peticiones de un cliente no pueden ser satisfechas por los servicios disponibles (Berardi, y otros, 2003).

En general, la composición de servicios se puede dividir en dos aspectos fundamentales la síntesis y la orquestación (Küster, y otros, 2005). La síntesis se refiere a cómo generar un plan para lograr el comportamiento deseado a través de la combinación de múltiples servicios. La orquestación se encarga de la coordinación del flujo de control y datos entre varios componentes, durante la ejecución del plan. El presente trabajo de maestría se enfoca exclusivamente en la síntesis autónoma de servicios, la orquestación de los servicios compuesto es un problema complementario que podrá ser abordado en trabajos futuros.

En (Ben Mokhtar, 2007) se propone una clasificación de soluciones para la composición de servicios basada en dos categorías: la composición basada en interfaces y la composición basada en conversaciones. La primera categoría asume que los servicios son descritos como una lista de capacidades independientes, sin conversaciones asociadas, mientras la segunda categoría asume que los servicios son descritos mediante una conversación asociada.

2.4.1 Composición de Servicios basada en Interfaces

Este tipo de composición es empleado cuando los servicios disponibles en la red y las tareas de usuario son descritos como capacidades individuales sin una conversación asociada, en este modelo de composición los servicios son combinados basándose en la conformidad de sus signaturas (Ben Mokhtar, 2007).

Búsqueda en grafos: en (Zhang, y otros, 2003) se construye un grafo en el que los nodos representan los servicios disponibles y las aristas indican si las salidas de un servicio sirven como entradas a otro servicio. A las aristas se les asigna un peso que mide la correspondencia entre las entradas y salidas de los nodos adyacentes. El problema de esta solución radica en su pobre escalabilidad cuando se incrementa el número de servicios contenidos en los repositorios, su complejidad es $O(n^3)$. En (Hashemian, y otros, 2005) se crea un grafo de dependencia entre los servicios almacenados en un repositorio, los vértices del grafo representan las entradas y salidas que ofrecen los servicios, mientras que las aristas son las dependencias o servicios que permiten pasar de una entrada a una salida. La composición de servicios en este trabajo se inicia empleando el algoritmo de búsqueda en anchura (BFS - Breadth First Search), de tal forma que se encuentra el camino más corto para satisfacer las salidas requeridas por la composición a partir de unas entradas determinadas. El trabajo de (Wang, y otros, 2009) presenta una aproximación para la agregación de servicios basada en grafos y que permite realizar la composición autónoma de servicios en ambientes ubicuos., construyendo un grafo de dos niveles, el primero representa las relaciones funcionales de los servicios y sus parámetros, mientras el segundo contiene las instancias de los servicios y los tipos de datos. Se emplea una modificación del algoritmo DFS (Depth First Search) para obtener la agregación de los servicios. Estas dos últimas soluciones

tienen una complejidad lineal con respecto al número de servicios contenidos en el grafo, de acuerdo a los algoritmos empleados (BFS está acotado por $O(|V| + |E|)$). Sin embargo la complejidad en (Wang, y otros, 2009) está determinada por $O(m^2)$), donde m representa una medida del tamaño de la ontología empleada para las comparaciones semánticas.

Encadenamiento de servicios: existen dos clases de encadenamiento, hacia adelante (forward chaining) y hacia atrás (backward chaining). El primero inicia seleccionando los servicios que respondan a las entradas y precondiciones establecidas por la tarea que se desea realizar, una vez realizada esta selección se pude inferir que servicio encadenar basándose en la compatibilidad de sus signaturas. El encadenamiento finaliza cuando se generan todas las salidas y efectos esperados por la tarea. Por su parte, el encadenamiento hacia atrás inicia buscando aquellos servicios que generan los efectos deseados por la tarea a realizar. Posteriormente, emplea un algoritmo recursivo que le permite encontrar los servicios necesarios para crear las precondiciones requeridas por la tarea. Diferentes propuestas de investigación han adoptado estos enfoques de composición (Ponnekanti, y otros, 2002) (Martínez, y otros, 2004) (Masuoka, y otros, 2003) (Ramasamy, 2006), corroborando su aplicación en contextos tales como los servicios web. La complejidad de estas técnica depende de la cantidad de servicios que se deben encadenar, especialmente en el encadenamiento hacia adelante, donde se pueden crear caminos de encadenamiento infructuosos que no conduzcan al logro de los efectos deseados (Küster, y otros, 2005). Adicionalmente, el proceso de encadenamiento esta guiado únicamente por la compatibilidad en la signatura de los servicios, abriendo la posibilidad de emplear capacidades no apropiadas, generando un grado de incertidumbre con respecto a la manipulación de la información del usuario y la conformación semántica del servicio compuesto.

2.4.2 Composición de Servicios basada en Conversaciones

La composición de servicios basada en conversaciones asume que los servicios poseen un comportamiento complejo descrito a través de una conversación. En (Ben Mokhtar, 2007) se divide esta categoría en tres casos:

La selección de conversaciones dirigidas por la meta: permite seleccionar la conversación de un servicio que satisface las tareas de usuario, tal como se propone en (Bernstein, y otros, 2002). En este trabajo el comportamiento de los servicios es capturado usando modelos de procesos; estos modelos y sus componentes (sub tareas, recursos, etc.) son ubicados dentro de una ontología de procesos. Las consultas se realizan utilizando un Lenguaje de Consulta de Procesos (PQL), el cual permite obtener todos los servicios que contengan un conjunto dado de entidades y relaciones que satisfagan las peticiones del usuario. En este caso se asume que las tareas pertenecientes al servicio, o un subconjunto de ellas, pueden satisfacer las peticiones expresadas por el cliente; sin requerir la integración con otros servicios existentes.

La integración de conversaciones dirigida por la meta: busca la integración de las conversaciones de los servicios existentes para realizar una tarea de usuario expresada como una capacidad requerida. Este caso de composición es empleado por (Brogi, y otros, 2008) (Brogi, y otros, 2005), donde se selecciona un conjunto de conversaciones y se integran de tal forma que el servicio compuesto satisface la tarea de usuario, consumiendo las entradas proporcionadas y generando los efectos requeridos. En estos trabajos, las tareas de usuario se expresan en términos de sus entradas y salidas, buscando una mayor facilidad en la formulación de los requisitos del usuario. En (Shiaa, y otros, 2008) se representan los servicios como un conjunto de metas (entradas, salidas y propiedades no funcionales), los servicios recuperados desde el repositorio son enlazados de

acuerdo a la similitud semántica entre las entradas y salidas, constituyendo un grafo. Este enfoque también emplea una variación del algoritmo BFS para encontrar los caminos entre las entradas y salidas requeridas, finalmente la composición se realiza teniendo en cuenta la similitud semántica entre los nodos, las aristas, el número de servicios empleados y las propiedades no funcionales requeridas. Debido a que los requerimientos del usuario son expresados en términos de las metas (entradas y salidas) se mantiene la incertidumbre sobre el comportamiento del servicio compuesto y la forma como la información del usuario es empleada durante la ejecución.

La integración de conversaciones dirigida por conversación: este modelo de composición asume que tanto los servicios como las tareas de usuario son expresadas por medio de una conversación. (Berardi, y otros, 2005) presenta el framework Colombo, el cual aborda el problema de la composición automática de servicios web. En este trabajo se incorpora la noción de "servicio meta", para denotar el comportamiento que se desea para el servicio requerido, definiendo las transiciones entre los servicios web, los procesos atómicos que se deben invocar y los mensajes que se intercambian. En (Hashemian, y otros, 2005) los servicios y los requisitos de los usuarios son expresados como una terna ordenada compuesta por un conjunto de entradas, un conjunto de salidas y un conjunto de dependencias entre las entradas y las salidas. El conjunto de dependencias expresa el procesamiento que se desea para manejar los datos entregados por el usuario, este procesamiento puede ser realizado por uno o varios servicios atómicos. La composición es realizada por medio de la búsqueda en un grafo de dependencia. En (Ben Mokhtar, 2007) se presenta un mediador para la composición de servicios en ambientes ubicuos, en este trabajo las tareas de usuario son expresadas como una dupla conformada por un autómata que describe la conversación de la tarea y un conjunto de propiedades no funcionales que deben cumplir los servicios. La composición se soporta en una representación de autómatas finitos y busca la integración de servicios descubiertos en una fase previa de emparejamiento semántico.

2.4.3 Valoración de las Técnicas de Composición

La Tabla 3 resume la valoración que se realizó de las técnicas de composición presentadas en esta sección, la comparación realizada está soportada en los trabajos de (Küster, y otros, 2005) y (Ben Mokhtar, 2007). El símbolo (+) indica que la propiedad evaluada es soportada por la técnica, mientras que el símbolo (-) significa que no es soportada. El símbolo (+/-) señala que la propiedad es soportada de una manera moderada.

Tabla 3. Valoración de las Técnicas de Composición

Técnicas de Composición	Parámetros			
	Semántica	**Confiabilidad**	**Comportamiento**	**Escalabilidad**
Búsqueda en Grafos	+/-	-	-	-
Encadenamiento de Servicios	-	-	-	+/-
Selección de Conversaciones dirigida por la meta	+/-	-	+/-	+/-

Integración de Conversaciones dirigida por la meta	+/-	-	+/-	+/-
Integración de Conversaciones dirigida por conversación	+/-	+	+	-

Los parámetros evaluados para cada técnica son: semántica, capacidad de la técnica de describir semánticamente los servicios utilizados en la composición; comportamiento, descripción de las tareas de usuario como un comportamiento complejo, definiendo los servicios que participan y sus interacciones; confiabilidad, capacidad de la técnica de entregar una tarea compuesta lo mas similar al comportamiento que desea el usuario; escalabilidad, rendimiento de la técnica con respecto al número de servicios requeridos para componer las tareas de usuario.

La composición de servicios basada en la búsqueda de grafos incorpora un análisis semántico moderado, se soporta principalmente en la comparación de las interfaces de los servicios desde un nivel sintáctico, teniendo en cuenta su signatura y el tipo de datos. La mayoría de de estas soluciones no tiene en cuenta el comportamiento de las tareas de usuario, por esta razón poseen una baja confiabilidad, se tiene la incertidumbre sobre los servicios atómicos o simples que se agregan dentro del comportamiento del servicio compuesto y la forma como estos emplean la información del usuario durante la ejecución. La escalabilidad de las soluciones basadas en grafos depende directamente del algoritmo empleado para encontrar una posible composición que satisfaga el requisito del cliente. Por su parte, el encadenamiento de servicios se basa principalmente en comparaciones sintácticas de las interfaces de los servicios, no contempla el comportamiento de los servicios teniendo una confiabilidad muy pobre. También presenta una escalabilidad baja, ya que se deben explorar todos los posibles esquemas de encadenamiento, especialmente en el encadenamiento hacia adelante, donde se realizan búsquedas en direcciones innecesarias para alcanzar los efectos deseados.

La selección de conversaciones dirigida por la meta agrega una noción de comportamiento al soportar la publicación y consulta de servicios complejos, sin embargo las tareas de usuario son expresadas aún con base a las entradas y salidas, sin una conversación asociada. De esta manera la confiabilidad es pobre, al desconocer el comportamiento del conjunto de servicios encontrados durante el proceso de consulta. Las capacidades semánticas dependen de la técnica de consulta empleada para seleccionar las conversaciones.

La integración de conversaciones dirigida por la meta permite combinar procesos complejos, conformados por varios servicios atómicos o simples, para obtener una composición que genere las salidas requeridas por el usuario a partir de las entradas especificadas. En esta técnica se ofrece una mayor flexibilidad en la composición al permitir agregar diferentes procesos, a diferencia de la selección de conversaciones dirigida por la meta, en la cual se puede escoger un único proceso para satisfacer los requisitos del usuario. Sin embargo, los requisitos siguen siendo capturados en términos de las entradas y salidas únicamente, sin detallar el comportamiento deseado por el cliente.

La integración de conversaciones dirigida por conversación es la técnica donde se captura el comportamiento en la tarea de usuario. Al expresar el requerimiento del usuario por medio de una conversación se obtiene una mayor confiabilidad, ya que se asegura los servicios que se desea

participen en la composición y los datos que debe manipular cada uno de ellos. La escalabilidad de esta técnica reposa principalmente en la fase de descubrimiento, se puede soportar en la recuperación de servicios simples o estructuras complejas, determinado el tiempo de respuesta para obtener un servicio compuesto.

Teniendo en cuenta la valoración realizada en esta sección se puede concluir que independientemente de la técnica de composición seleccionada, se pueden incluir capacidades sintácticas o semánticas, dependiendo del tipo de comparaciones que se requieran para el emparejamiento de los servicios y sus interfaces. Por otra, parte la confiabilidad va ligada directamente a la capacidad de expresar los requerimientos del usuario como una conversación, ya que permite lograr una mayor aproximación al comportamiento que desea el usuario para sus tareas.

De acuerdo a las conclusiones previas, en este trabajo de grado de maestría se optó por una técnica de composición dirigida por conversación, buscando una mayor confiabilidad en los procesos de composición. Para tener una mejor escalabilidad la composición se soportó en la integración de servicios atómicos, utilizando la técnica de descubrimiento seleccionada en la sección 2.3.4. La técnica de composición se acompañó por un analizador lingüístico, como el empleado en (Corrales, 2008), para realizar las comparaciones a nivel de interfaces y así, poder establecer la conformidad de las interfaces de los servicios a integrar.

2.5 Modelos Formales

En este apartado se realiza un análisis de las técnicas de representación formal para modelos de comportamiento, con el objetivo de ilustrar la selección de la técnica que se empleó para la definición de los algoritmos de descubrimiento y composición propuestos en esta tesis de maestría.

2.5.1 Redes de Petri

Las redes de Petri son un lenguaje formal y gráfico para modelar sistemas concurrentes (Reisig, y otros, 1998), su principal característica es la forma natural como capturan los conceptos básicos de este tipo de sistemas, tanto a nivel matemático como conceptual.

Las redes de Petri son muy populares en el campo de los procesos de negocio ya que permiten capturar los flujos de control de los procesos. En el trabajo propuesto por (Ouyang, y otros, 2005) (Hinz, y otros) se muestra como mapear el flujo de control de un proceso BPEL a una red de Petri, incluyendo el flujo de control para el manejo de excepciones y compensación.

Este modelo formal ha sido empleado para la composición de servicios, en (Narayanan, y otros, 2002) se emplea para la composición de servicios web, se discute la creación de una herramienta para la descripción y verificación automática para la composición de servicios. En (Yi, y otros, 2004) los autores presentan una framework para la composición de servicios, el cual permite visualizar, crear y verificar procesos BPEL.

2.5.2 Procesos Algebraicos

Al igual que las redes de Petri, los procesos algebraicos permiten la verificación automática de ciertas propiedades de comportamiento. Poseen una rica teoría sobre la bisimulación, la cual

permite establecer si dos procesos tienen un comportamiento equivalente. Este tipo de análisis es muy útil para determinar si un determinado servicio puede remplazar a otro dentro de una composición o establecer redundancias de servicios. Ejemplos de esta representación formal son CCS (*Calculus of Communicating Systems*), CSP (*Communicating Sequential Processes*), y p-calculus.

La propuesta de (Ferrara, 2004) combina BPEL y un proceso algebraico denominado LOTOS (Bolognesi, y otros, 1987), con lo cual logra la inclusión de flujos de excepción y compensación. Por otra parte, el trabajo de (Mazzara, y otros, 2006) presenta un lenguaje denominado *web*π_∞, el cual representa una simplificación de WS-BPEL que permite realizar un razonamiento formal sobre la orquestación de procesos. Este lenguaje está compuesto de operadores que permiten expresar el comportamiento, define operadores para secuencia, concurrencia, restricciones, intercambio de mensajes, etc.

2.5.3 Autómata de Estados Finitos (FSA)

Un autómata de estados finitos es definido por un conjunto finito de mensajes, estados, transiciones, un estado inicial y un conjunto de estados finales (Hopcroft, y otros, 2001). Esto puede ser representado como un grafo con un estado inicial único, donde los nodos representan los estados y los arcos las transiciones, las transiciones pueden ser etiquetadas para expresar el conjunto de mensajes. Sin embargo, los FSA en su forma original presentan problemas para representar la semántica de la secuencia de los mensajes, por lo que no soportan adecuadamente la semántica de las ejecuciones en paralelo (Corrales, 2008).

En (Berardi, y otros, 2003) los autores presentan un modelo conceptual basado en FSA para representar el comportamiento de e-Servicios y restricciones temporales. En (Ben Mokhtar, 2007) se utiliza este modelo formal para representar el comportamiento de las tareas de usuario en una ambiente de computación ubicua, el autómata es empleado para desarrollar las funciones de descubrimiento y composición de las capacidades presentes en la red ubicua.

2.5.4 Grafos

Los grafos son una estructura general de datos que sirve para representar objetos y conceptos. En una representación de grafos los nodos corresponden a los objetos o conceptos mientras los arcos definen las relaciones existentes entre los conceptos.

En (Mendling, y otros, 2005), se propone una transformación para representar procesos BPEL como EPC (Event-Driven Process Chains), la cual puede ser definida como un grafo dirigido, con propiedades de cardinalidad y tipos de restricciones. Esta representación se concentra en capturar el comportamiento definido en un proceso BPEL, modelando las actividades básicas y complejas, el flujo de control, el manejo de los flujos de excepción y compensación.

La representación de grafos ha sido empleada en diversos trabajos relacionados con la composición de servicios (Eshuis, y otros, 2006) (Goncalves da Silva, y otros, 2007) (Wang, y otros, 2009). Los trabajo de (Eshuis, y otros, 2006) (Goncalves da Silva, y otros, 2007) realizan una composición guiada por metas, donde se procura realizar la composición de servicios basándose en la conformidad de sus interfaces, teniendo en cuenta la compatibilidad entre las entradas y salidas. Por otra parte (Wang, y otros, 2009) proponen un enfoque para ambientes ubicuos, en el que introducen conceptos del contexto en la composición la cual es guiada por un comportamiento definido por el usuario.

2.5.5 Valoración de las Técnicas de Representación

A continuación se presenta una comparación entre las representaciones formales presentadas, basada en la valoración de modelos formales realizada en (Corrales, 2008) y (Beek, y otros, 2006). Se consideran las siguientes propiedades: integridad (madurez de la notación y una semántica completa para expresar el comportamiento), capacidad de composición (propiedad que permite razonar acerca de un sistema tomando como base sus partes constitutivas, sin necesidad de tener información adicional sobre la implementación de dichas partes), escalabilidad (capacidad de representar un modelo de comportamiento aún cuando el número de partes constitutivas se incremente), manejo de excepciones y compensación, paralelismo (capacidad de modelar procesos concurrentes importante para expresar el comportamiento).

La Tabla 4 resume la valoración que se realizó de los modelos de representación formal presentados en esta sección. El símbolo (+) indica que la propiedad evaluada es soportada por el modelo formal, mientras que el símbolo (-) significa que no es soportada. El símbolo (+/-) señala que la propiedad es soportada de una manera moderada.

Tabla 4. Valoración de las Técnicas de Composición

Parámetros	Modelo Formal			
	Redes de Petri	**Procesos Algebraicos**	**FSA**	**Grafos**
Integridad	+	+	+	+
Capacidad de Composición	+/-	+	+	+
Escalabilidad	+	+	+	+
Manejo de Excepciones y Compensación	+	+	-	+
Paralelismo	+	+	+/-	+

Finalmente, de la valoración presentada y considerado lo expuesto en (Corrales, 2008) se concluye que el modelo de representación de grafos ofrece una simple y madura notación para expresar la semántica de un servicio. La representación de grafos permite analizar un sistema compuesto a nivel de sus partes, sin la necesidad de adicionar información acerca de la implementación de ellas. Además, el paralelismo soportado por el modelo de grafos es un aspecto clave de formalismo que debe cumplirse para modelar con precisión el emparejamiento de servicios y por ende su posterior uso tanto en el descubrimiento como en la composición de servicios.

De acuerdo al anterior párrafo, en esta tesis de maestría se seleccionó la representación formal basada en grafos como herramienta para expresar el comportamiento de las tareas requeridas por el usuario y las conversaciones publicadas en el ambiente ubicuo por los proveedores de servicios. Sobre esta representación formal se soportaron las técnicas y algoritmos propuestos para el descubrimiento y la composición de servicios en ambientes de computación ubicua.

Capítulo 3

Arquitectura de Referencia

3.1 Modelo del Ambiente

La arquitectura de referencia se propone para crear ambientes de computación capaces de ofrecer las funcionalidades de descubrimiento y composición de servicios con el fin de asistir a los usuarios en la realización de sus tareas. Así, el modelo del ambiente del sistema se concibió a partir de los elementos conceptuales y funcionalidades básicas de SOA diagramadas en la Figura 3.

Para la arquitectura de referencia, Las tareas son expresadas por el usuario como un modelo de comportamiento, que detalla el flujo de ejecución, control y datos que se debe seguir para realizar las actividades deseadas. El Mediador de Composición utiliza el modelo de comportamiento como insumo para iniciar las labores de descubrimiento y composición de servicios, buscando generar un comportamiento lo más aproximado posible al requerido por el usuario. De esta manera, el Mediador de Composición desempeña las funciones de descubrimiento y composición de servicios para obtener la mejor aproximación a las tareas demandadas por el usuario, ver Figura 4 (Modelo del Ambiente).

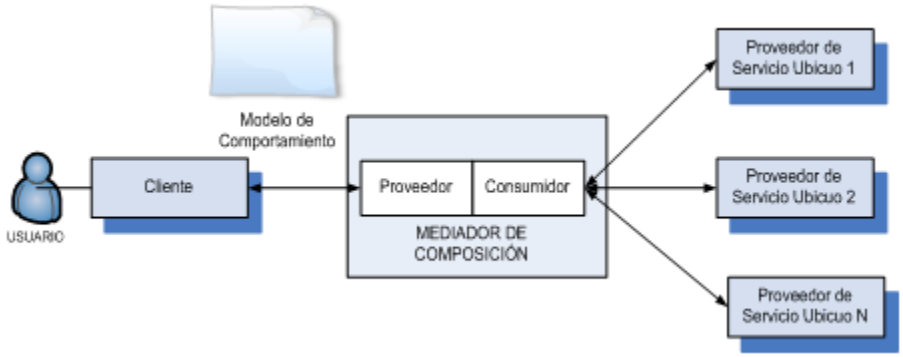

Figura 4. Modelo del Ambiente del Sistema

Tomando como punto de partida el modelo del ambiente del sistema en la siguiente sección se define una arquitectura de referencia para la composición de servicios en ambientes de computación ubicua.

3.2 Descripción General

De acuerdo al RUP: “una arquitectura de referencia, es en esencia, un patrón arquitectural predefinido, o un conjunto de patrones, parcial o totalmente instanciados, diseñados y provistos

para su uso en contextos particulares de negocio o técnicos, junto con artefactos que posibiliten su uso. A menudo, estos artefactos son cosechados de proyectos anteriores" (Reed, 2002).

Una arquitectura de referencia también se puede considerar como el diseño de alto nivel de un sistema, libre de detalles de implementación y que consiste de los siguientes elementos (Space Systems Group, 1996):

- Una descripción de alto nivel de los componentes del sistema
- Definición de las relaciones entre los componentes
- Definición de las relaciones entre los componentes del sistema y los elementos externos al sistema

Considerando los anteriores elementos, el principal problema abordado por una arquitectura de referencia es la definición de la estructura general y las relaciones entre los principales procesos (elementos funcionales) del sistema, indicando el flujo de datos y control entre los procesos e identificando los principales requisitos de desempeño, incluyendo potencia de procesamiento, ancho de banda y capacidad de almacenamiento para asegurar que el sistema sea capaz de manejar las cargas esperadas. Adicionalmente, esta arquitectura debe proveer una definición de alto nivel de las fuentes de datos, almacenamientos de datos e interfaces entre los componentes del sistema. Las últimas generalmente son especificadas como una Interfaz de Programa de Aplicación (API).

La Figura 5 representa la arquitectura de referencia propuesta para la composición de servicios en ambientes de computación ubicua, que se sustenta sobre cuatro niveles: Nivel de Aplicación, Nivel de Modelos, Nivel de Contexto y Nivel de Servicios; estos niveles agrupan los módulos necesarios para la composición y ejecución de servicios dentro de un ambiente ubicuo.

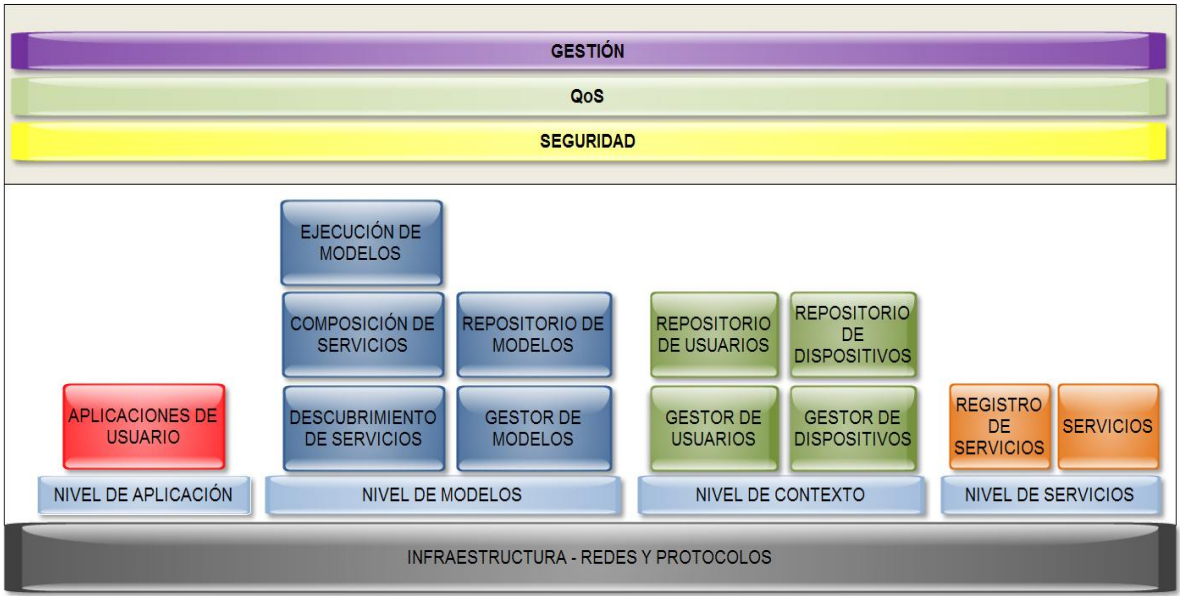

Figura 5. Arquitectura de Referencia Propuesta

A continuación se describe cada uno de los niveles que conforman la arquitectura de referencia.

3.2.1 Nivel de Infraestructura

El nivel inferior corresponde a la Infraestructura que provee las capacidades de comunicación en el ambiente ubicuo. Agrupa los diferentes tipos de redes que pueden coexistir, así como los diferentes protocolos de comunicación a ser empleados por los elementos conectados a la red ubicua. Este nivel representa la heterogeneidad que distingue a los ambientes ubicuos.

3.2.2 Nivel de Servicios

- **Servicios:** las funcionalidades presentes en el ambiente ubicuo se abstraen como servicios, los cuales son los componentes básicos utilizados para estructurar los procesos más complejos que llevan a cabo las tareas requeridas por los usuarios**.**

  Los servicios deben estar compuestos por tres partes: la implementación, la descripción y los métodos de invocación. La implementación del servicio está determinada por las funcionalidades que se requieren del servicio.

  El servicio debe ser descrito a través de una interfaz, que emplee un lenguaje estándar y que asegure una total independencia entre la implementación del servicio y el cliente. La interfaz define el modelo lógico del servicio, el nombre del servicio, las operaciones, las entradas y salidas y las excepciones que se pueden lanzar.

  Los métodos de invocación especifican como se pueden consumir los servicios, definen que protocolos emplear, cómo enviar las entradas requeridas y qué tipo de salidas esperar. Se debe mantener un bajo acoplamiento entre la forma de invocar un servicio y su implementación.

- **Registro de Servicios**: lista los servicios disponibles y provee los metadatos que describen dichos servicios, incluyendo la descripción estándar y los detalles de comunicación (dirección, IP, puerto y protocolo) con el servicio. Este registro permite mantener un desacople entre los proveedores del servicio y sus clientes.

3.2.3 Nivel de Aplicación

En este nivel se agrupan las aplicaciones necesarias para que el usuario interactúe con las funcionalidades que expone el ambiente de computación ubicua.

- **Aplicaciones de Usuario:** este tipo de aplicaciones proporcionan las interfaces necesarias para que el usuario interactué con el ambiente. Cuenta con las capacidades de comunicación que soportan el intercambio de información con las redes ubicuas (Nivel de Infraestructura).

3.2.4 Nivel de Contexto

En este nivel se agrupan los módulos involucrados en almacenar y gestionar la información relativa al contexto del usuario y los dispositivos dentro del ambiente ubicuo. Tomando como referencia los trabajos citados en la sección 2.3.3, a continuación, se puntualizan los siguientes elementos para definir el contexto:

- **Repositorio de Usuario:** almacena los datos relacionados con los usuarios, los cuales pueden incluir preferencias, proporcionadas de manera explícita cuando se utiliza el sistema (perfiles de usuario, características de personalización, etc.), o datos recolectados implícitamente por la arquitectura. Los datos implícitos pueden estar conformados por patrones de uso y comportamiento del usuario (p.ej. servicios consumidos previamente por el usuario, servicios más consumidos, etc.).
- **Repositorio de Dispositivos:** cada usuario interactúa con la arquitectura propuesta a través de diversos dispositivos, los cuales poseen diferentes características tales como procesamiento, tamaños de la pantalla, facilidades de comunicación, etc. Estas características son almacenadas en un repositorio y son consultadas para definir el contexto de entrega de los servicios y mejorar la experiencia de interacción con el ambiente ubicuo.
- **Gestor de Usuarios y Gestor de Dispositivos:** estos módulos agrupan las funcionalidades necesarias para manejar los datos almacenados en los repositorios del contexto. Incluyen las funciones para adicionar, actualizar y eliminar datos de los repositorios mencionados. Cada gestor también ofrece la interfaces necesarias para realizar las consultas pertinentes sobre los repositorios, de tal forma que se pueda obtener los datos relevantes para definir el contexto de un usuario.

3.2.5 Nivel de Modelos

Este nivel constituye el núcleo de la arquitectura, es donde se concentran las capacidades necesarias para lograr el descubrimiento y composición de los servicios requeridos para realizar las tareas propuestas por el usuario.

- **Repositorio de Modelos:** almacena los modelos de comportamiento diseñados a partir de los servicios disponibles en el ambiente ubicuo. Este repositorio debe contener los metadatos que describen los modelos así como aquellos archivos necesarios para poder ejecutar un modelo de comportamiento.
- **Gestor de Modelos:** este módulo permite realizar las funciones de gestión sobre el repositorio de modelos, abarcando las operaciones necesarias para publicar, actualizar o eliminar un modelo en el repositorio. Las operaciones de consulta se realizan a través de este componente y permiten recuperar la información publicada sobre un modelo en el repositorio.
- **Descubrimiento de Servicios:** permite realizar la búsqueda de los servicios más adecuados para llevar a cabo las tareas requeridas por el usuario. El descubrimiento de servicios se realiza a partir del modelo de comportamiento que requiere el usuario, se obtienen los servicios que componen los modelos de comportamiento publicados en el repositorio de modelos, y se ejecuta un proceso de emparejamiento que permite dilucidar cuáles son los servicios que mejor se adaptan a los requisitos del usuario.
- **Composición de Servicios**: se encarga de componer los servicios recuperados por el módulo de descubrimiento, de tal forma que se obtiene una conversación útil para ejecutar las tareas de usuario establecidas en el requerimiento inicial.
- **Ejecución de Modelos:** permite realizar las invocaciones de los servicios que conforman la tarea de usuario que se desea ejecutar. El motor de ejecución debe estar basado en un lenguaje estándar para la descripción de modelos de comportamiento, a partir del cual se estructura la secuencia de invocaciones, el flujo de datos y manejo de errores. Además, debe soportar los métodos de invocación que exponen los servicios y adaptarse a las características heterogéneas de las redes ubicuas tales como los tipos de dispositivos, protocolos y medios de comunicación.

3.3 Relaciones

En esta sección se explican las relaciones entre los módulos de la arquitectura y la interacción con los actores externos. Por tanto, se describe la forma como los módulos colaboran para soportar las funcionalidades básicas (publicación, localización, emparejamiento y acceso) y de composición identificadas para SOA. La Figura 6 presenta el diagrama de relaciones entre los módulos de la arquitectura de referencia.

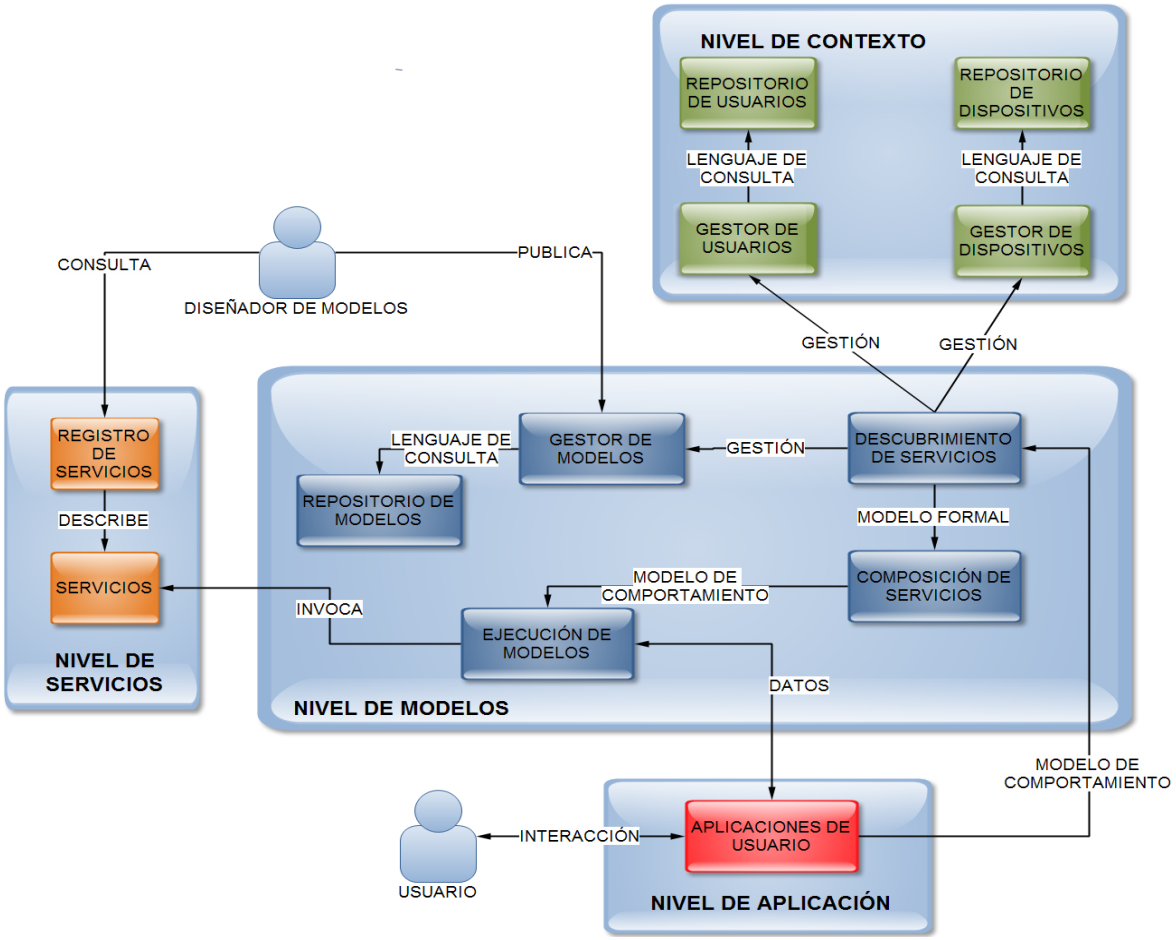

Figura 6. Diagrama de Componentes y Relaciones

Los actores que interactúan con la arquitectura son dos: el Diseñador de Modelos y el Usuario. El Diseñador de Modelos consulta el Repositorio de Servicios para obtener las descripciones de los servicios disponibles y así poder diseñar los modelos de comportamiento que estarán disponibles en el ambiente ubicuo. Las conversaciones diseñadas serán publicadas en el Repositorio de Modelos a través del Gestor respectivo, en este repositorio podrán ser consultadas para su posterior uso en la ejecución de las tareas de usuario.

El Usuario representa a las personas que interactúan con el ambiente ubicuo, y que utilizan las prestaciones que ofrece el sistema para realizar sus tareas. El Usuario emplea las Aplicaciones de Usuario para comunicarse con el sistema y poder hacer uso de los servicios que están disponibles. Las peticiones del usuario son expresadas como un modelo de comportamiento, en el cual se especifica la tarea a realizar.

En los siguientes párrafos se explican las relaciones entre cada uno de los módulos, haciendo especial énfasis en el tipo de información que intercambian.

- **Aplicaciones de Usuario – Descubrimiento de Servicios:** las aplicaciones de usuario son las encargadas de tomar las peticiones del usuario y comunicarlas a los demás módulos para iniciar la composición. Las peticiones del usuario son especificadas como un Modelo de Comportamiento que representa el flujo de ejecución, control y datos que requiere la tarea del usuario. El modelo de comportamiento debe ser expresado en un lenguaje estándar que permita definir la manera como deben ser armonizados los servicios para obtener conversaciones más complejas. La petición realizada por la Aplicación de Usuario es abstracta y no contiene mayor información sobre los detalles de invocación de los servicios requeridos, se limita a describir la secuencia de ejecución, el flujo de control y datos. El modelo de comportamiento es recibido por el módulo de descubrimiento, lugar donde se da el primer paso (Emparejamiento de servicios) para realizar la composición de servicios.
- **Descubrimiento de Servicios – Gestores de Repositorios**: estas relaciones representan las consultas que se realizan a los repositorios de modelos, usuario y dispositivos para implementar las funcionalidades de localización y emparejamiento de servicios. Los gestores deben ofrecer una interfaz de gestión para realizar las consultas sobre los repositorios, especificando un lenguaje para formular las consultas respectivas y así poder interpretar los resultados arrojados.
- **Descubrimiento de Servicios – Composición de Servicios:** la composición de servicios se realiza a partir de los servicios recuperados durante la fase de descubrimiento, el módulo de composición toma como entrada los servicios seleccionados por el componente de descubrimiento. Estos dos componentes operan sobre modelos formales que representan el comportamiento, por esta razón la información intercambiada se expresa en términos de un modelo formal.
- **Composición de Servicios – Ejecución de Servicios**: una vez terminada la composición de servicios el modelo de comportamiento generado debe ser ejecutado par realizar la tarea de usuario requerida. El comportamiento entregado por el módulo de composición es un modelo concreto que contiene la información necesaria para ejecutar la conversación generada, especificando los detalles necesarios para invocar los servicios utilizados en el modelo compuesto.
- **Aplicaciones de Usuario – Ejecución de Modelos**: intercambian los datos requeridos y/o generados durante la ejecución de un modelo de comportamiento. Se aprovecha las facilidades de presentación contenidas por las aplicaciones para capturar los datos necesitados para invocar un servicio y mostrar la información que se genera como resultado después de ejecutar las tareas de usuario.
- **Gestor de Repositorio – Repositorio**: esta relación agrupa las dependencias de gestión entre los componentes Gestor de Procesos – Repositorio de Procesos, Gestor de Usuarios – Repositorio de Usuario y Gestor de Dispositivos – Repositorio de Dispositivos. Básicamente cada repositorio ofrece un lenguaje de consulta por medio del cual se pueden realizar las funciones de consulta y actualización de los datos contenidos en los repositorios.

3.4 Módulos

En esta sección se describe la estructura interna de los módulos que pertenecen a la arquitectura de referencia. Se establecen los componentes y las relaciones que crean para desempeñar sus labores.

3.4.1 Aplicaciones de Usuario

Para las aplicaciones de usuario se propone emplear la arquitectura para sistemas ubicuos presentada por (Muñoz, y otros, 2006), la cual utiliza el patrón arquitectural Modelo-Vista-Control (MVC) para proveer una arquitectura multicapa para sistemas ubicuos, ver Figura 7.

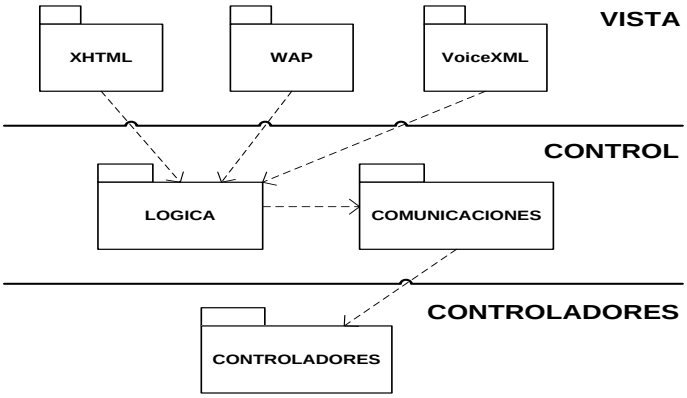

Figura 7. Arquitectura para Aplicaciones de Usuario Ubicuas

- **Controladores**: es el nivel más bajo de la arquitectura, se encarga de encapsular las funciones requeridas para manejar los requerimientos propios de cada recurso hardware o dispositivo. Siguiendo esta estrategia, el nivel de controladores ofrece una interfaz común para cada tipo de hardware o dispositivo, la cual es empleada por los niveles superiores.
- **Comunicaciones**: este nivel permite una comunicación transparente entre los controladores y el nivel de lógica, obteniendo una independencia de los requerimientos propios de cada fabricante. El nivel de comunicaciones ofrece una representación lógica de los elementos físicos para que puedan ser manipulados por el nivel de lógica de la aplicación.
- **Lógica**: provee las funcionalidades que el usuario demanda de las aplicaciones. Los componentes que pertenecen a esta capa hacen uso de la capa de comunicaciones u otros componentes lógicos para realizar sus tareas.
- **Vista:** proporciona las interfaces gráficas para que el usuario interactué con el sistema.

3.4.2 Repositorios y Gestores

La descripción de los repositorios de usuario, dispositivos y modelos, y sus respectivos gestores se realiza de una manera general. Así, a continuación, se establece una arquitectura común (ver Figura 8) para estos tres repositorios identificando los componentes necesarios para gestionar los datos que almacenan.

Los componentes que hacen parte de los repositorios son los siguientes:

- **Sistemas de Archivos:** representan a las estructuras de datos contenidas en una unidad de almacenamiento y que permiten representar la información del contexto o de los modelos de comportamiento publicados.
- **Manejador de Datos**: este componente se encarga de acceder a los datos contenidos en el sistema de archivos. Otorga una independencia a los niveles superiores del medio de almacenamiento escogido en una implementación determinada. Se pueden tener diferentes

alternativas de almacenamiento (bases de datos, archivos planos, XML, etc.) para las cuales se implementa un manejador de datos especifico con total trasparencia para los niveles superiores al momento de manipular los datos guardados.

- **Mediador**: forma parte del núcleo del repositorio y provee una interfaz para los componentes software externos que requieren tener acceso a los datos contenidos en el sistema de archivos. El mediador contiene una API que gestiona los datos almacenados, implementando las funciones de creación, actualización y eliminación de los datos, además maneja las consultas que pueden ser ejecutadas sobre el repositorio para obtener la información del contexto y los modelos comportamiento.

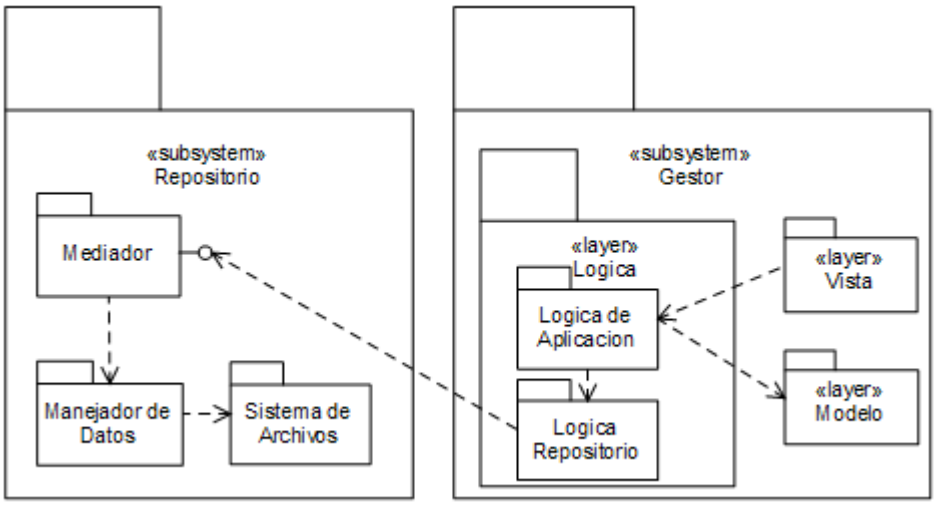

Figura 8. Arquitectura para los Repositorios y Gestores

Los módulos de gestión de los repositorios se basan en un patrón arquitectural Modelo-Vista-Control, las capas se describen a continuación:

- **Modelo:** en esta capa se encapsulan los datos que emplea la aplicación de gestión para su funcionamiento (p.ej. usuarios, perfiles de usuarios, permisos, etc.), exporta las funciones necesarias para que los componentes superiores puedan acceder a los datos y manipularlos.
- **Lógica:** esta capa maneja los eventos generados por el usuario, e implementa la lógica para desarrollar las funcionalidades de las aplicaciones de gestión de los repositorios. La lógica se divide en dos componentes: Lógica de Aplicación y Lógica de Repositorio. Esta separación se realiza para identificar claramente que parte de las funciones están dedicadas a la gestión de los datos contenidos en los repositorios. La Lógica de Repositorio emplea las interfaces expuesta por la API del Repositorio para implementar las funciones de gestión. Por otro lado, la Lógica de Aplicación se encarga de manejar los eventos del usuario, realizar el procesamiento necesario para invocar la Lógica de Repositorio a partir de las acciones realizadas por el usuario o interactuar con el modelo de la aplicación, de acuerdo a las peticiones recibidas.
- **Vista:** contiene las interfaces gráficas de usuario, que permiten desplegar la información correspondiente al modelo de la aplicación o de los repositorios. Ofrece los elementos necesarios para capturar los eventos del usuario y transmitirlos al control para su posterior procesamiento.

A continuación se aborda cada uno de los aspectos relacionados al tipo de información a ser tenida en cuenta en los repositorios definidos en la arquitectura de referencia.

3.4.2.1 Contexto del Usuario y Dispositivo

(Guerrero, y otros, 2010) Propone tres dimensiones que permiten definir un meta-modelo de contexto del usuario, estas dimensiones son tenidas en cuenta dentro de la arquitectura de referencia definida en este trabajo, para establecer el contexto del usuario y sus dispositivos (Figura 11).

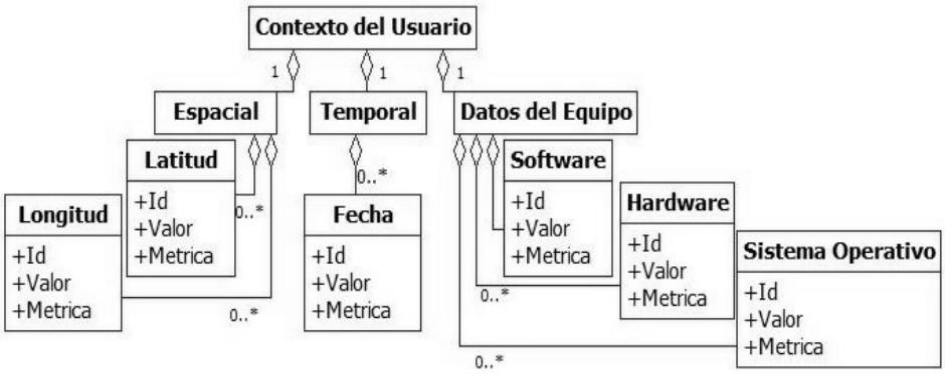

Figura 9. Meta-Modelo de Contexto del Usuario

- Dimensión espacial: contiene todos los parámetros asociados con la longitud y latitud desde la cual un usuario accede al servicio.
- Dimensión temporal: contiene información sobre la hora y la fecha en la cual se hace una petición al sistema o es invocado el servicio.
- Dimensión de los datos del equipo: la información del equipo como software soportado e instalado, hardware, tipo de sistema operativo, son útiles para procesos de adaptación de contenido. Abraca además la información contenida en diferentes estándares utilizados para este propósito como el CC/PP (Composite Capability/Preference Profiles) (Klyne, y otros, 2004).

3.4.2.2 Repositorio de Modelos

En la Figura 10 se presenta un diagrama con las asociaciones entre los conceptos necesarios para constituir un repositorio de modelos.

Se parte del hecho que los modelos van a estar organizados en carpetas (Folder) donde se almacenan los archivos (File) requeridos para describir un modelo de comportamiento. Cada archivo tiene como atributos un nombre y una URI que identifica de manera única su ubicación en el repositorio, además se asocia un tipo de archivo y una extensión que va de acuerdo al tipo. Cada carpeta contiene un descriptor (Descriptor) que lista los archivos contenidos, creando una especie de índice. El tipo de contenido (Content_Type) permite especificar el rol que desempeña un archivo dentro de la descripción de un modelo de comportamiento, teniendo en cuenta que archivos del mismo tipo pueden ser empleados para diferentes propósitos dentro de la especificación de un comportamiento.

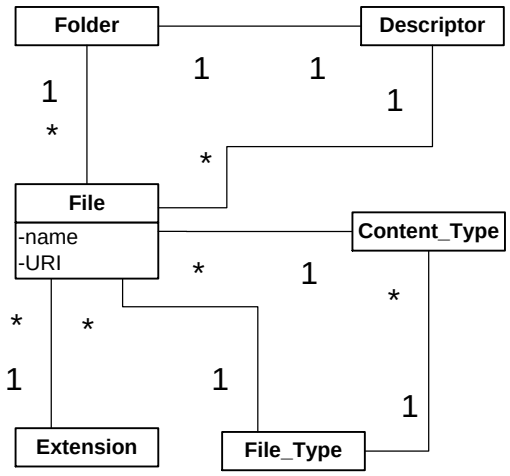

Figura 10. Conceptos del Repositorio de Modelos

3.4.3 Descubrimiento de Servicios

El módulo de descubrimiento de servicios es desglosado en esta sección, definiendo los componentes internos y su interacción para realizar las funciones de localización y emparejamiento de servicios en la arquitectura de referencia (ver Figura 11).

Los componentes incorporados en el módulo de descubrimiento se describen a continuación:

- **Traductor**: soporta las funciones necesarias para realizar la transformación de un modelo de comportamiento expresado en un lenguaje estándar (BPEL, WSCL, etc.) a una representación formal (grafos, máquinas de estados, autómatas finitos, etc.). La implementación de este componente depende directamente del lenguaje de comportamiento y la representación formal seleccionados al momento de realizar una instanciación de la arquitectura de referencia.
- **Control de Descubrimiento**: coordina los componentes internos para realizar la localización y emparejamiento de servicios, encapsula la lógica que determina los pasos a seguir para realizar el descubrimiento de servicios. Este componente exporta una interfaz para recibir las peticiones provenientes desde las aplicaciones de usuario e iniciar el proceso de descubrimiento de servicios basado en el requisito enviado por el usuario. Una vez terminado el descubrimiento de servicio entrega los resultados obtenidos al módulo de Composición para que se complete el requerimiento del usuario. Este componente implementa el Algoritmo 7 presentado en el Capítulo 4.

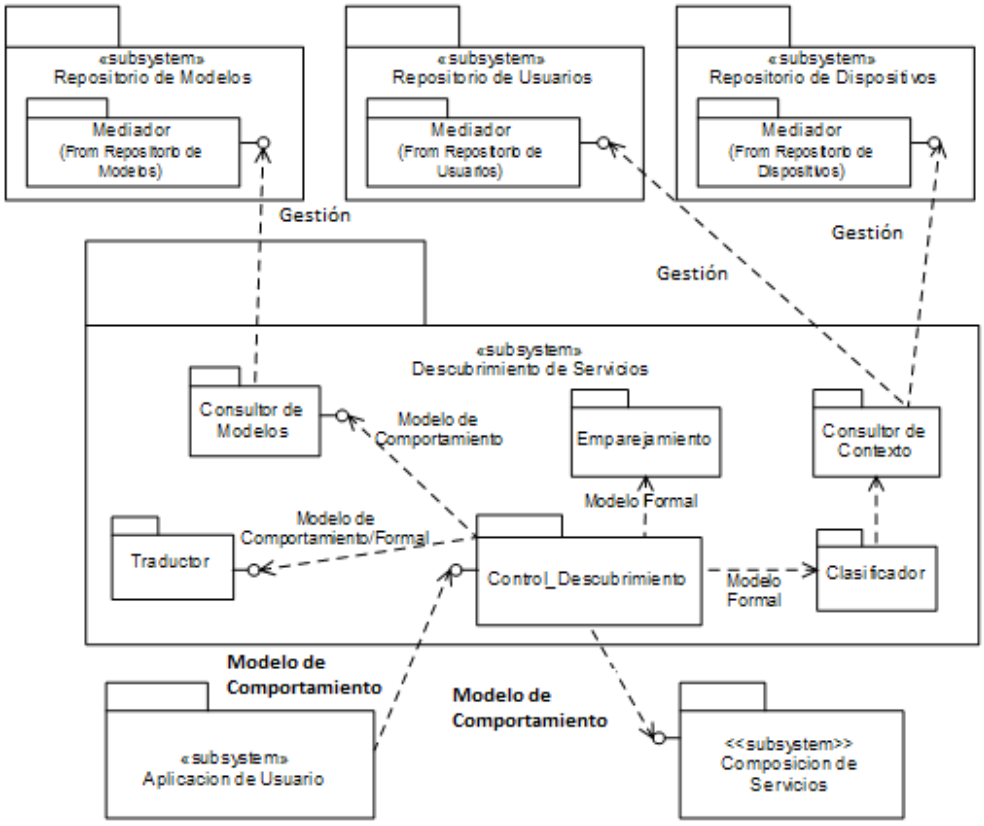

Figura 11. Componentes del Módulo de Descubrimiento de Servicios

- **Consultor de Modelos**: se emplea como frontera de comunicación con el repositorio de conversaciones. Este componente importa las interfaces expuestas por el API del repositorio para implementar las funciones de consultas requeridas por el módulo de descubrimiento para obtener los comportamientos publicados en el repositorio. El consultor se encarga de los detalles propios de comunicación con el repositorio de modelos, abstrayendo a los demás componentes de los diversos requisitos de implementación impuestos por los diferentes tipos de repositorios de modelos que se pudieran emplear. Los resultados de las consultas son entregados como modelos de comportamiento los cuales deben ser procesados para obtener una representación formal para ejecutar las funciones de emparejamiento.
- **Consultor de Contexto**: importa el API del repositorio de usuarios y dispositivos, a través de este consultor se obtiene la información correspondiente al contexto de entrega del usuario. Permite realizar consultas sobre los repositorios de contexto, ocupándose de los detalles de comunicación pertinentes y que dependen directamente del tipo de interfaces que expongan los repositorios y los medios de invocación que exporten. En este componente se agrupan los procedimientos descritos en los Algoritmos 4 y 5 presentados en el Capítulo 4.
- **Emparejamiento**: implementa el Algoritmo 1 presentado en el Capítulo 4, el cual permite seleccionar los servicios más similares a aquellos que requiere el usuario en sus tareas. Las comparaciones se realizan sobre los servicios que conforman las conversaciones publicadas en el repositorio de modelos y el objetivo trazado es estimar cuan similares son los servicios presentes en la red ubicua a los necesitados por el usuario. La similitud de dos servicios va

estar determinada por los elementos que describen a un servicio, por ejemplo las entradas, salidas, nombre del servicio, etc. Las técnicas de emparejamiento operan sobre los modelos formales que representan las conversaciones publicadas y el comportamiento requerido por el usuario.

- **Clasificador**: toma como entrada las distancias de similitud estimadas por el componente de emparejamiento y establece un ranking, permitiendo determinar cuáles servicios son más similares a los pedidos por el usuario. Este componente tiene en cuenta la información del contexto para realizar el ordenamiento de los servicios, se considera el contexto del usuario y de sus dispositivos para crear un ranking que tenga en cuenta el contexto de entrega. De acuerdo al ranking establecido se utilizaran los servicios recuperados para el proceso de composición, dando mayor prioridad a aquellos servicios que sean más similares a los buscados y que se ajusten mejor al contexto de entrega del usuario. La clasificación de los servicios está implementa en el Algoritmo 7 presentado en el Capítulo 4.

A continuación se describe la secuencia de acciones que se realizan durante el proceso de descubrimiento, detallando las interacciones entre los componentes internos (ver Figura 12).

El descubrimiento de servicios es iniciado cuando se recibe una petición desde un cliente, la invocación incluye como parámetro de entrada la conversación definida como la tarea de usuario. El componente Control de Descubrimiento recibe la invocación e inicia el manejo de los demás componentes para realizar localización y emparejamiento de servicios. La localización de los modelos publicados se ejecuta a través del Consultor de Modelos, posteriormente se transforman todos los modelos de comportamiento, conversación de consulta y conversaciones publicadas, a una representación formal para realizar el emparejamiento de servicios. Como último paso se arma un ranking de servicios tomando la información del contexto y las medidas de similitud estimadas en el emparejamiento. El ranking generado es entregado al Control de Descubrimiento, y constituye el producto resultante del proceso de descubrimiento, dicha lista organizada servirá como insumo para las funciones de composición, las cuales son guiadas por el modelo formal obtenido a partir del requisito inicial del usuario.

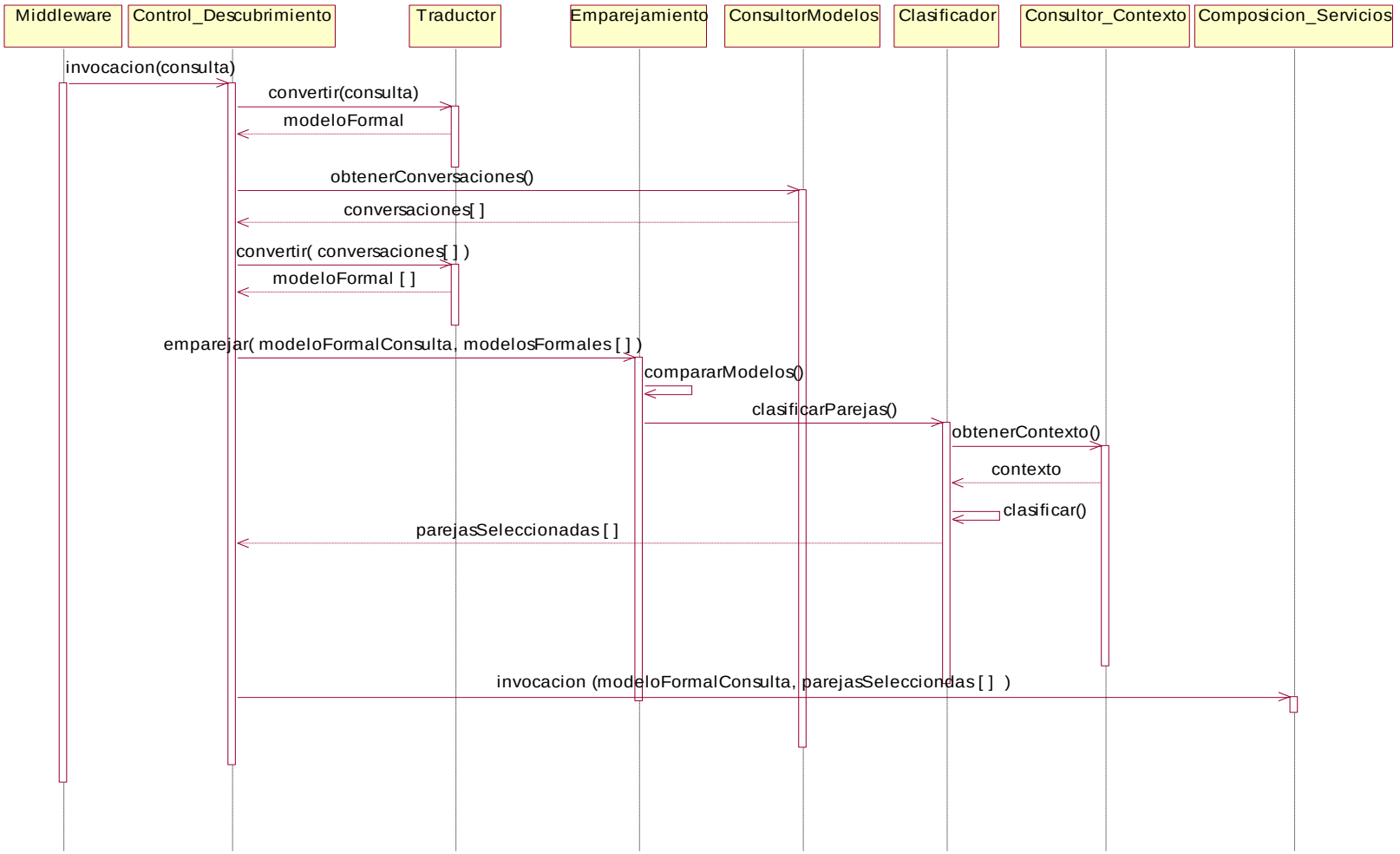

Figura 12. Interacción Componentes de Descubrimiento de Servicios

3.4.4 Composición de Servicios

La Figura 13 ilustra los componentes que hacen parte del módulo de composición de servicios y las relaciones respectivas.

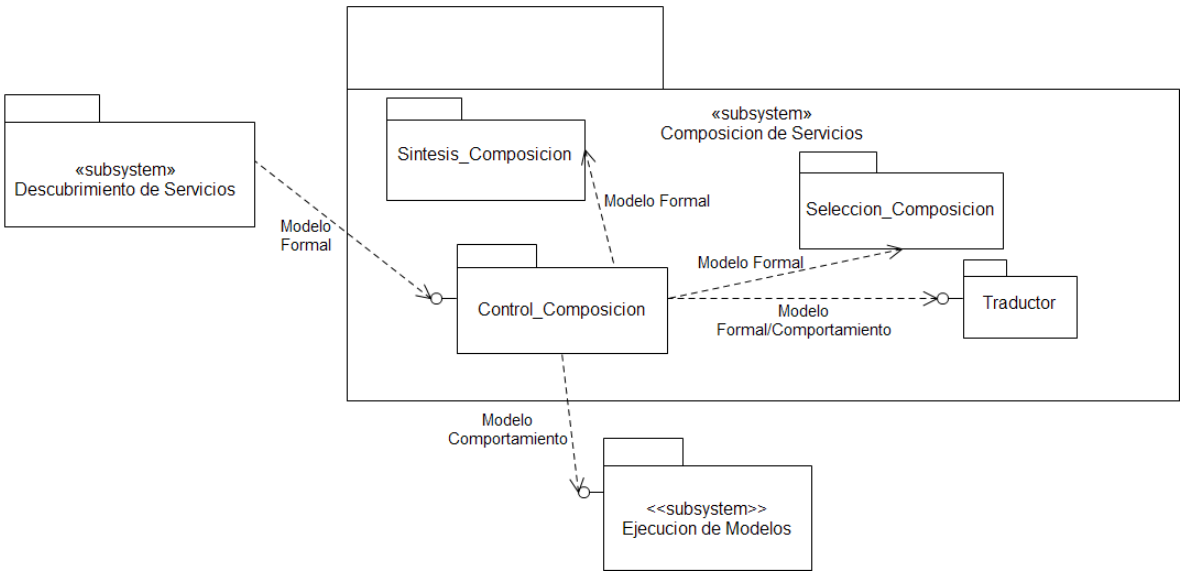

Figura 13. Componentes del Módulo de Composición de Servicios

A continuación se describen los componentes que constituyen el módulo de composición:

- **Control de Composición**: coordina los componentes internos para realizar la composición de servicios. Exporta la interfaz para recibir las peticiones provenientes desde el módulo de descubrimiento e iniciar el proceso de composición de servicios basado en el requisito enviado por el usuario. Este componente implementa el Algoritmo 10 presentado en el Capítulo 5.
- **Síntesis de Composición**: se encarga de sintetizar la tarea del usuario empleando los servicios recuperados por el módulo de descubrimiento, la síntesis consiste en coordinar los servicios recuperados para obtener un servicio más complejo que llene los requerimientos del usuario. La coordinación se lleva a cabo tomando como referencia el flujo de ejecución, control y datos definidos en el modelo de comportamiento enviado como petición desde el usuario. Todo este proceso de composición se realiza sobre los modelos formales que representan el requisito del usuario y los servicios recuperados. Como el módulo de descubrimiento entrega una lista de servicios candidatos para utilizar en la composición se pueden obtener varias síntesis de composición, derivadas de las posibles combinaciones que se realicen entre los servicios recuperados, las combinaciones se realizan tomando en cuenta la compatibilidad de las interfaces de los servicios, las precondiciones de ejecución, etc. Este conjunto de síntesis conforma el producto entregado por este componente y son el insumo para el módulo de selección. Este componente implementa el Algoritmo 18 presentado en el Capítulo 5.
- **Selección de Composición**: recibe las síntesis de composición generadas por el sintetizador y escoge una síntesis para ser enviada como respuesta a la solicitud del usuario. La similitud entre las síntesis candidatas y el requisito del usuario está determinada principalmente por las

distancias estimadas para los servicios recuperados y que hacen parte de las síntesis de composición, la secuencia de ejecución y el flujo de control se preservan y siguen exactamente el orden definido por el requisito del usuario. Finalmente la síntesis seleccionada es pasada al Control de Composición para que sea procesada y enviada al usuario. Este componente implementa el Algoritmo 19 presentado en el Capítulo 5.

- **Traductor**: se encarga de realizar la transformación del modelo formal obtenido después del proceso de composición a un modelo de comportamiento expresado en lenguaje estándar que permita su ejecución.

La interacción de los componentes del módulo de composición es diagramada en la Figura 14. Esta figura detalla la secuencia de pasos que se siguen para realizar el proceso de composición, se muestran las invocaciones realizadas entre los componentes internos y las relaciones con los módulos externos.

1. Se recibe la invocación desde el módulo de descubrimiento y se da inicio al proceso de composición. Los parámetros de entrada son el modelo formal de consulta que representa la petición del usuario y las parejas seleccionadas en el descubrimiento, las parejas contienen la lista de los servicios recuperados y las medidas de similitud estimadas.
2. El componente de control de composición inicia la composición solicitando que se realice la síntesis. El módulo de síntesis recibe como entradas el modelo formal de consulta y las parejas seleccionadas para cada actividad de consulta, utilizando estos insumos se forman múltiples candidatos de composición, conformados por las diferentes combinaciones de servicios que se pueden obtener a partir de las parejas seleccionadas. Las síntesis de composición generadas son entregadas como respuesta al control de composición.
3. Una vez terminado el proceso de síntesis se selecciona uno de los esquemas de composición sintetizados en el paso anterior y se entrega como el modelo de comportamiento de respuesta. La selección de composición busca el candidato que más se aproxime al modelo de consulta, teniendo en cuenta las descripciones de las actividades y sus interfaces. Finalmente, se retorna como salida el modelo formal que representa el comportamiento seleccionado.
4. Cuando el control de composición recibe el modelo formal compuesto seleccionado se continúa realizando su transformación a un modelo de comportamiento que emplee un lenguaje que permita su ejecución en el ambiente ubicuo. Esta transformación es realizada por el componente Traductor.
5. El último paso consiste en entregar el modelo de comportamiento obtenido al módulo de Ejecución de Modelos.

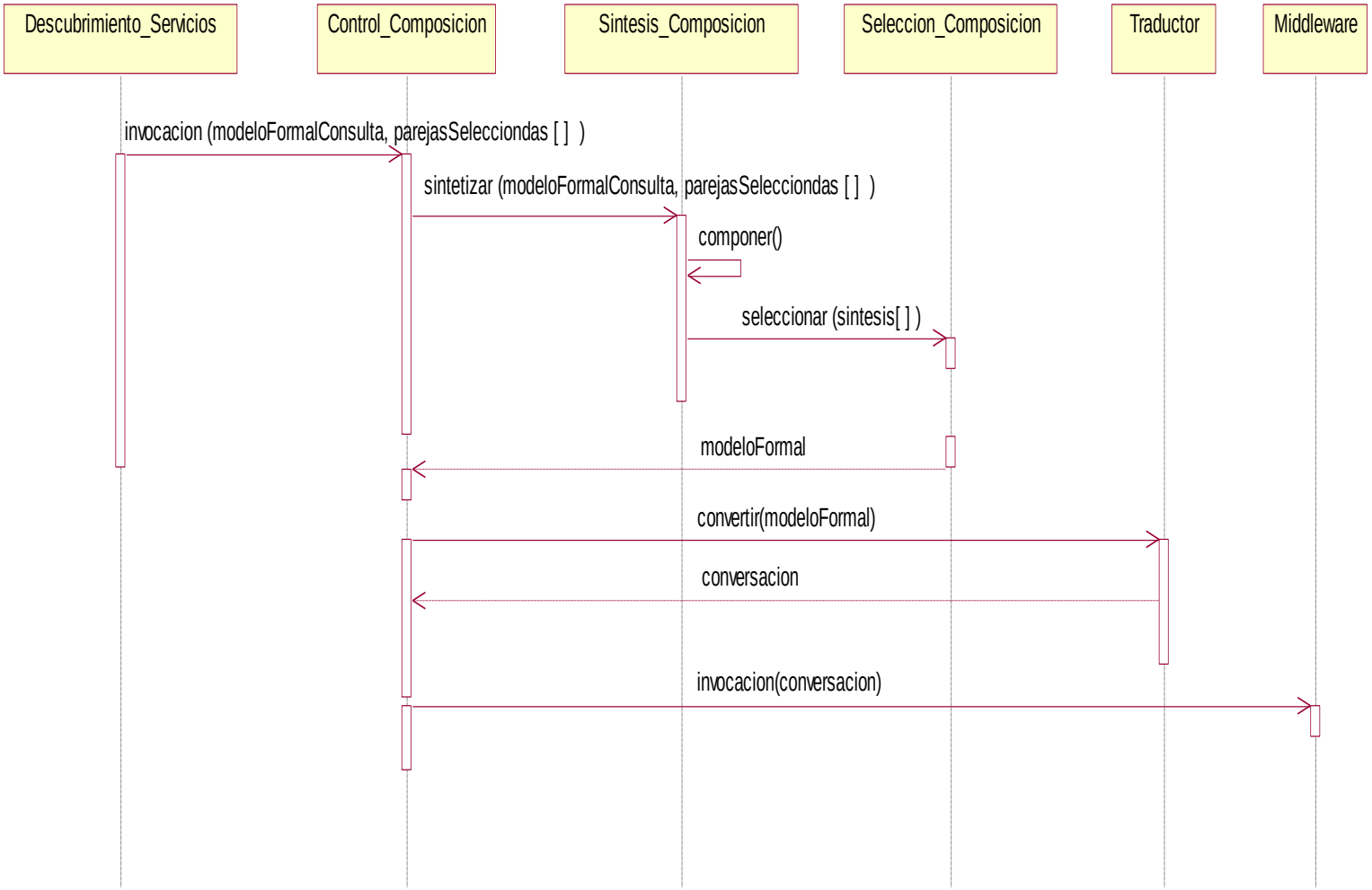

Figura 14. Interacción Componentes de Composición de Servicio

3.5 Implementación de Referencia

En esta sección se describe la implementación de la arquitectura. Como primer paso se define el alcance de la misma y las decisiones de diseño consideradas. Posteriormente, se amplía la descripción de la implementación detallando la forma como se instanciaron los módulos y sus respectivos componentes.

3.5.1 Alcance

La implementación de referencia persigue dos objetivos principales, asociados a los objetivos de esta tesis de maestría:

1. Instanciar la arquitectura de referencia para proporcionar las funcionalidades necesarias para soportar el despliegue y composición de servicios dentro de una ambiente de computación ubicua.
2. Desarrollar prototipos que permitan realizar la evaluación experimental de la efectividad y eficiencia de los enfoques adoptados en la arquitectura para la composición de servicios de información turística dentro de un ambiente de computación ubicua.

3.5.1.1 Características

De acuerdo a los propósitos se definen las siguientes características para la implementación de referencia:

- **Descubrir Servicios**: la implementación de referencia debe soportar las funciones necesarias para realizar el descubrimiento de servicios en un ambiente ubicuo.
- **Componer Servicios**: debe incorporar las funcionalidades de composición basada en comportamiento, las cuales se soportan en el descubrimiento de servicios.
- **Información Turística**: la implementación de referencia debe ser probada en un contexto de aplicación para servicios de información turística, esta característica determina el tipo de servicios que se van a manejar dentro del ambiente de computación ubicua así como los modelos de comportamiento que se diseñan a partir de ellos.

3.5.1.2 Restricciones

En este apartado se listan las consideraciones que se tuvieron en cuenta para realizar la instanciación de la arquitectura para la implementación de referencia del presente trabajo:

1 En el diseño propuesto se han excluido los componentes correspondientes al Registro de Servicios y al proceso de búsqueda de servicios. Se asume que los comportamientos han sido diseñados previamente y publicados en el repositorio de modelos, esto teniendo en cuenta que el proceso de diseño del proceso de negocio BPEL a partir de servicios web publicados en un registro UDDI es una tarea bien conocida y que cuenta con el suficiente soporte tecnológico y teórico para su realización. Los servicios son considerados ya en la etapa de

ejecución de las tareas de usuario, siendo invocados desde el cliente que utiliza el usuario para interactuar con el ambiente.

2 La arquitectura es concebida para un ambiente de dispositivos móviles, los cuales operan como los clientes empleados por el Usuario para interactuar con el ambiente ubicuo. Esta restricción es tomada para limitar claramente el tipo de repositorio que se emplea para representar la información del contexto de los dispositivos, teniendo en cuenta que si se consideran todos los dispositivos que pueden hacer parte de un ambiente ubicuo el espectro de información para el contexto de entrega sería excesivamente amplio para el caso de estudio.

3.5.2 Arquitectura

La Figura 15 muestra la arquitectura de la implementación de referencia, la cual es una instanciación de la arquitectura de referenciada planteada en este capítulo.

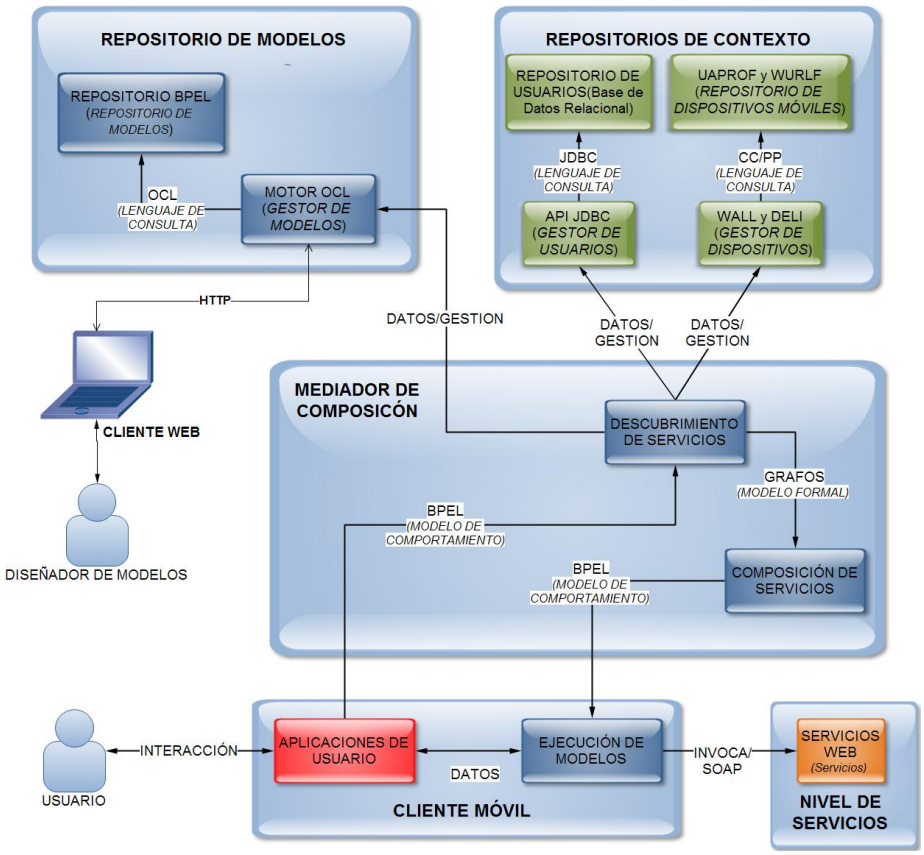

Figura 15. Arquitectura de la Implementación de Referencia

En la arquitectura de la implementación de referencia (Figura 15) las aplicaciones de usuario y el módulo de ejecución de modelos BPEL se localizan en el cliente móvil, estos dos módulos soportan la interacción del usuario con el ambiente, permiten comunicarse con el mediador de composición para obtener un modelo de comportamiento útil para realizar las tareas del usuario, y posteriormente ejecutarlas por medio de la invocación de los servicios web presentes en la red ubicua.

El mediador de composición agrupa los módulos de descubrimiento y composición de servicios de la arquitectura de referencia, por medio de estos dos elementos se implementan las funcionalidades básicas SOA de localización, emparejamiento y composición dentro del ambiente ubicuo.

3.5.3 Herramientas y Tecnologías

En esta sección se presentan las herramientas y tecnologías empleadas para implementar cada uno de los módulos que hacen parte de la implementación de referencia.

3.5.3.1 Servicios

Para la implementación de los servicios en la instanciación de la arquitectura se seleccionaron los servicios web. Un servicio web puede ser definido como una pieza de una lógica de negocio, localizada en cualquier parte de la Internet y que es accesible a través de protocolos estándares tales como HTTP o SMTP. Se seleccionaron los servicios web ya que es una tecnología madura y que se ha impuesto como la más adecuada para la realización de SOA (Juric, y otros, 2007). Se pueden mencionar las siguientes características de los servicios web, las cuales los hacen apropiados para emplearlos en la presente tesis de maestría:

- Auto-contenidos: no requieren que se instalen componentes en el lado del cliente, una vez los servicios web son desplegados el cliente los puede consumir sin necesidad de instalar componentes software en su máquina.
- Auto-descritos: los documentos WSDL describen las interfaces de los servicios web, definen el formato para el intercambio de mensajes y los tipos de datos usados en los mensajes.
- Independencia de los lenguajes, plataforma y protocolos: los servicios web están basados en estándares XML abiertos, un lenguaje neutral, que permite que clientes escritos en cualquier lenguaje y sobre cualquier plataforma puedan invocar el servicio. Estas invocaciones pueden ser realizadas sobre cualquier protocolo estándar de red. Esta independencia es muy importante en ambientes abiertos y heterogéneos como las redes ubicuas.
- Dinamismo: los servicios web pueden ser descubiertos y consumidos en tiempo de ejecución, sin necesidad de tener un conocimiento previo sobre ellos.
- Composición: los servicios web pueden ser agregados para conformar un servicio más complejo, se pueden seguir dos patrones la orquestación o coreografía de servicios.

3.5.3.2 Modelos de Comportamiento

Partiendo del hecho que se seleccionó a los servicios web como implementación de los servicios disponibles en la red ubicua, los modelos de comportamiento se pueden representar con dos tipos de tecnologías: la orquestación y coreografía. Estas tecnologías permiten abordar la búsqueda, agrupación y composición de servicios web para obtener un proceso de negocio más complejo y significativo.

Coreografía de servicios: describe la colaboración entre una colección de servicios para alcanzar una meta común. La coreografía captura las interacciones entre los participantes y las dependencias entre estas, incluyendo: dependencias de flujo de control, exclusión, flujo de datos, correlaciones, restricciones de tiempo, dependencias de transacción, etc.

La coreografía de servicios es descrita formalmente como: el comportamiento externo observable a través de múltiples clientes (los cuales generalmente son servicios web, aunque no exclusivamente), donde el comportamiento se define como la presencia o ausencia de mensajes intercambiados entre los servicios web y sus clientes (Kavantzas, y otros, 2004).

A continuación se listan un conjunto de características comunes entre los lenguajes de coreografía de servicios web WS-CDL (Web Services – Choreography Description Language, WSCI (Web Service Choreography Interface), WSCL (Web Services Conversation Language) (Ivanova, 2006):

- Se concentran en especificar el comportamiento observable los servicios que se están comunicando
- Describe cuales mensajes son intercambiados por los servicios en una situación en particular
- No especifican como se deben implementar los servicios para el intercambio de mensajes

Orquestación de servicios: se refiere a la coordinación de múltiples servicios de tal forma que se obtenga un servicio más complejo o un proceso compuesto por los servicios individuales (Juric, y otros, 2007). Un modelo de orquestación define las acciones de comunicaciones y las acciones internas en las cuales participa un servicio, las cuales incluyen transformación de datos e invocación a módulos internos software.

La orquestación posee las siguientes características (Juric, y otros, 2007):

- Capacidad de invocar servicios de una manera asíncrona, se pueden realizar una ejecución concurrente y no solo secuencial.
- Maneja transacción y compensación, en caso de que un servicio falle el sistema puede realizar la compensación adecuada.
- Manejo de excepciones.
- Entrega segura y confiable de mensajes.
- Separación entre la lógica de procesos abstractos y los servicios web concretos usados. Esta característica posibilita la creación de servicios dinámicos y flexibles.

La orquestación es soportada por toda una familia de lenguajes estándar basados en XML, donde BPEL es el más representativo. Se encuentran otros lenguajes tales como: BEPLJ (BPEL for Java), BPML (Business Process Modelling Language), BPSS (Business Process Specification Schema), jPDL (jBPM Process Definition Language), XPDL (XML Process Definition Language), etc. Sin embargo, es claro que BPEL se ha convertido en el estándar dominante teniendo en cuenta la cantidad de herramientas software comerciales producidas para la composición de servicios que lo soportan.

Por tanto, para el caso de estudio se utilizó BPEL como el lenguaje estándar para especificar los modelos de comportamiento. Así, se tiene como ventaja la capacidad de poder emplear procesos de negocio abstractos que sirven como plantillas para generar posteriormente procesos ejecutables, condición apropiada para expresar los requerimientos de un usuario y luego componer diferentes procesos de negocio que respondan a dicho requerimiento. Estos procesos

abstractos solo se ocupan de aspectos relacionados con el comportamiento externo observable sin tomar recaudos de los detalles de ejecución.

Como el Repositorio de Modelos debe ser instanciado de acuerdo al lenguaje seleccionado para la especificación de los modelos de comportamiento, para la implementación de referencia se utilizó un repositorio que permite almacenar, consultar y gestionar procesos de negocio BPEL. De igual manera, el componente Motor de Ejecución de Modelos corresponde a una instancia que permite ejecutar procesos de negocio BPEL invocando los servicios web contenidos en los modelos compuestos.

3.5.3.3 Repositorio de Modelos

El repositorio de modelos se implementó empleando el trabajo presentado por (Vanhatalo, y otros, 2006), el cual presenta un repositorio BPEL que permite almacenar procesos de negocio junto con otros documentos XML. Por defecto, el repositorio soporta los lenguajes estándares de los servicios web BPEL, WSDL y esquemas XML, sin embargo puede ser extendido para soportar otros tipos de documentos XML, característica esencial para representar información asociada al contexto de entrega de los modelos publicados.

La Figura 16 ilustra la arquitectura propuesta por (Vanhatalo, y otros, 2006). El API del Repositorio es empleado por el componente de Gestión de Módulos para gestionar los procesos almacenados en el repositorio.

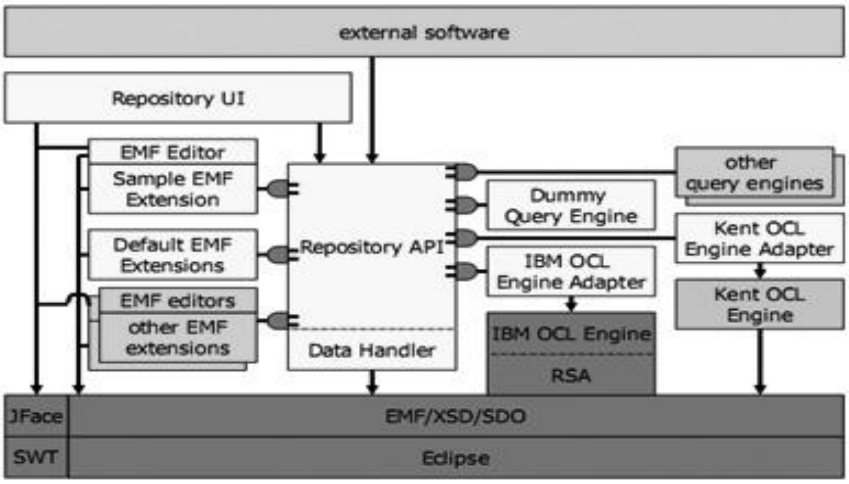

Figura 16. Arquitectura del Repositorio BPEL

El repositorio de procesos organiza los documentos en carpetas que representan una Organización, los archivos almacenados corresponden a: archivos tipo BPEL, WSDL que especifican la interfaz pública del proceso de negocio y las interfaces de los PartnerLinks empleados en proceso de negocio, archivos tipo XSD para la definición de esquemas de datos, y archivos XML que permiten definir meta-modelos para extender la descripción de los procesos de negocio con archivos adicionales. Cada organización tiene un descriptor que lista los archivos contenidos en la carpeta y el rol que desempeña cada uno.

3.5.3.4 Motor de Ejecución de Modelos

El motor de ejecución de modelos debe operar sobre BPEL, ya que es el lenguaje seleccionado para especificar los comportamientos en la implementación de referencia. Adicionalmente, es necesario que el motor de ejecución se acomode a las restricciones impuestas por los ambientes ubicuos: diversidad de dispositivos, heterogeneidad de redes y protocolos, etc.

Como implementación del motor de ejecución de modelos se propone el uso de Sliver (Hackmann, y otros, 2007), el cual es un motor de ejecución de BPEL que emplea SOAP, caracterizado por ser muy liviano y capaz de ser desplegado en un amplio rango de plataformas y dispositivos, cubriendo de dispositivos móviles hasta computadores personales. La Figura 17 muestra la arquitectura de Sliver que tiene entre otras las siguientes características:

- Provee una clara separación entre las comunicaciones y la lógica de procesamiento, los componentes de comunicación pueden ser intercambiados sin afectar la lógica, esto permite soportar una amplia variedad de medios y protocolos de comunicación.
- Adecuado para el uso bajo limitaciones de recursos hardware y de procesamiento. Sliver depende solo de dos librerías externas, las cuales han sido implementadas para ser empleadas en dispositivos móviles.

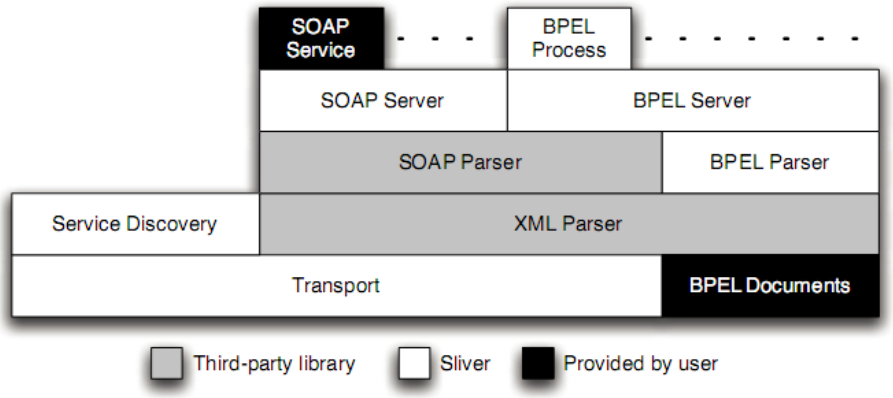

Figura 17. Arquitectura del motor de ejecución Sliver

3.5.3.5 Repositorio de Dispositivos

Para el repositorio de dispositivos se utilizaron dos soluciones que describen las capacidades de los dispositivos móviles: UAProf (User Agent Profile) (Forum, 2001) y WURFL (Wireless Universal Resource) (Passani, 2010).

UAProf: es la implementación de CC/PP para dispositivos móviles. CC/PP es una descripción de las capacidades de los dispositivos y preferencias de los usuarios, usualmente utilizado para definir el contexto de entrega del dispositivo final; está basado en RDF y define un conjunto de atributos y valores que determinan el contexto de entrega del dispositivo.

El sistema consiste en enviar una URL por medio de las cabeceras HTTP, esta URL contiene las capacidades del dispositivo, descritas según el estándar CC/PP. WAP 1.2.1 recomienda para el transporte de la información el protocolo HTTP Extensión Framework. WAP 2.0 define una extensión de HTTP 1.1 llamada W-HTTP. Cada fabricante de dispositivos tiene la responsabilidad de crear su propio UAProf y así permitir que el contenido web sea adaptado correctamente según las capacidades de su producto.

El esquema WAP UAProf consiste en la descripción de los siguientes componentes:

1. HadwarePlatform: una colección de propiedades que describen apropiadamente el dispositivo terminal. Esto incluye, el tipo de dispositivo, número del modelo, tamaño de la pantalla, métodos de entrada y salida de datos, etc.
2. SoftwarePlatform: una colección de atributos relacionados con el entorno de operación del dispositivo. Los atributos proveen información del sistema operativo usado, codificadores de video y audio soportados por el equipo y preferencias del lenguaje del usuario.
3. Browser UA: grupo de atributos para describir el browser HTML.
4. NetworkCharacteristics: información acerca de la infraestructura de red y el entorno en el cual la información transita. Los atributos son expresados en términos del ancho de banda de la red y la accesibilidad del dispositivo.
5. WapCharacteristics: un grupo de atributos pertenecientes a las capacidades WAP soportadas en el dispositivo. Esto incluye las capacidades del Browser WML, Wireless Telephony Aplication, etc.
6. PushCaracteristics: un grupo de características asociadas a las capacidades Push soportadas por el equipo. Esto incluye los MIME Types soportados por el equipo, el tamaño máximo de un mensaje push enviado del dispositivo, el tamaño del "buffer" de mensajes push, etc.

WURLF: es una iniciativa que gestiona un archivo XML de configuración el cual contiene información de las capacidades y características de un gran número de dispositivos móviles; frecuentemente, estas no son consignadas por los fabricantes sino por los desarrolladores que se ven en la necesidad de recopilarlos.

WURLF se basa en el concepto de familias, ya que los dispositivos descienden de un tipo de dispositivo genérico, se puede utilizar una jerarquía de familias para que los dispositivos nuevos hereden las propiedades atribuidas a los más genéricos.

Los dispositivos son descritos por medio de capacidades, se tienen parejas de atributos y valores. Estos atributos son agrupados dependiendo del tipo de las características que describan, entre los grupos de capacidades se tienen los siguientes: información del producto, interfaz de usuario para navegador WML, XHTML-MP, chtml, lenguajes de marcas soportados, características de pantalla (resolución, ancho, alto, etc), seguridad, almacenamiento, wap push, etc.

3.5.3.6 Repositorio de Usuarios

El repositorio de usuarios fue implementado como una base de datos relacional, su estructura está basada en el meta-modelo definido por (Guerrero, y otros, 2010), la Figura 18 presenta el modelo entidad relación de la base de datos diseñada.

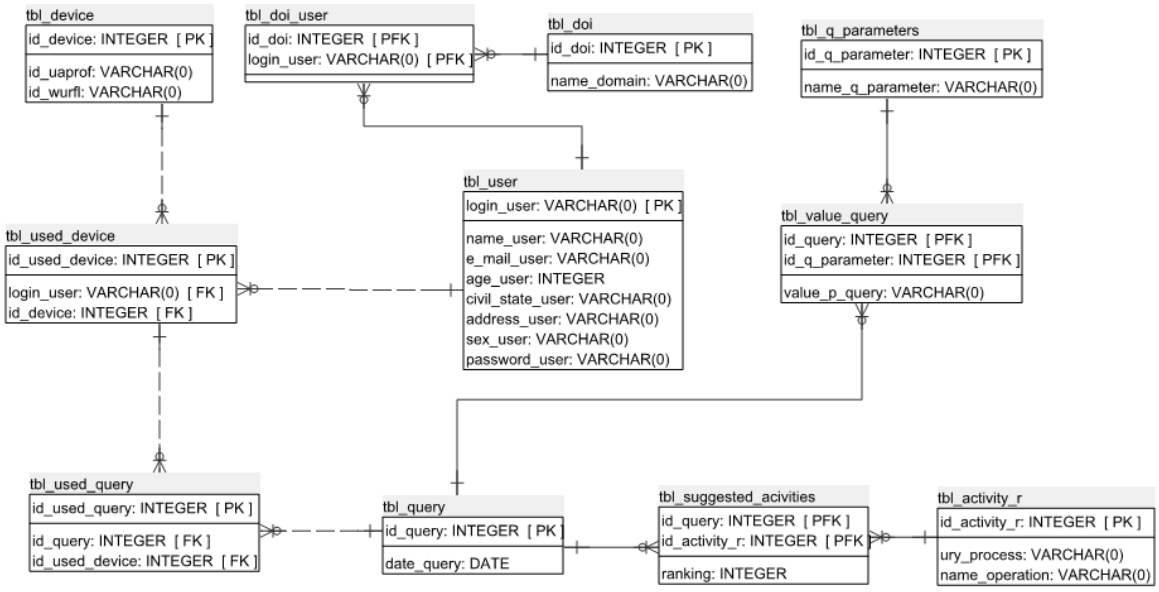

Figura 18. Modelo Entidad Relación del Repositorio de Usuario

La tabla *tbl_user* contiene la información personal del usuario, su nombre, edad, estado civil, sexo, dirección, junto con un login y contraseña que permiten validar su identidad frente al sistema. El usuario tiene asociado un conjunto de dispositivos, los cuales se describen por medio de las URL donde residen sus descripciones UAProf y WURLF, esta información se almacena en la tabla *tbl_device* y se asocia con el usuario por medio de la tabla intermedia *tbl_used_device*.

El repositorio de usuario permite crear un historial de las consultas realizadas por un usuario, almacenando las actividades consultadas y recuperadas en la fase de descubrimiento, esto teniendo en cuenta el dispositivo empleado, ya que es un elemento determinante para la selección final de las actividades candidatas de acuerdo al contexto de entrega. La tabla *tbl_suggested_activities* contiene la relación entre una actividad de consulta y las actividades recuperadas, almacena además la posición en el ranking asignado a la actividad recuperada. La tabla *tbl_activity_r* contiene la información del proceso donde se puede encontrar la actividad recuperada. Por su parte, la *tbl_query* representa un proceso de consulta agrupando todas las actividades de consulta contenidas en dicho proceso y sus parámetros de descripción. El proceso de consulta se asocia a los dispositivos empleados por el usuario, con esto se logra crear una relación entre las actividades recuperadas y el contexto de entrega del dispositivo, asegurando que la actividades puedan ser invocadas desde el cliente empleado por el usuario. Con este repositorio de usuario se crea un historial de actividades solicitadas por el usuario y las parejas recuperadas por el sistema, el uso de este historial y sus beneficios se condensan en la sección 4.3.2.3.

3.5.3.7 Módulos de Descubrimiento y Composición

Los componentes de descubrimiento y composición constituyen el principal aporte del presente trabajo, su desarrollo se realizó a partir de los elementos descritos previamente en esta sección. Su implementación se basa en los componentes definidos en las secciones 3.4.3 y 3.4.4, los detalles de implementación y formalización de los algoritmos empleados se consignan en el Capítulo 4 y el Capítulo 5 respectivamente.

3.5.4 Modelo de Despliegue

En esta sección se presenta el diagrama de paquetes, de implantación y despliegue de la implementación de referencia.

3.5.4.1 Diagrama de Paquetes de Aplicación

La Figura 19 muestra el diagrama de paquetes de aplicación de la implementación de referencia. A continuación se describen cada una de las capas contenidas en el diagrama.

Capa de Aplicación: contiene un paquete correspondiente a la *Aplicación Web* a través de la cual el usuario interactúa con la plataforma de descubrimiento y composición de servicios. El paquete de *Lógica de Negocio* encapsula toda la lógica necesaria para que las aplicaciones de descubrimiento y composición puedan ser empleadas por medio de la aplicación Web. Estos dos paquetes se implementan con el lenguaje de programación Java y se soportan sobre el JDK 1.6.0.

Capa de Modelos: abarca los paquetes de *Descubrimiento* y *Composición de Servicios*, también incorpora el repositorio de procesos BPEL. La implementación de los paquetes de *Descubrimiento* y *Composición de Servicios* y el *Motor de Consultas OCL* se soporta sobre el JDK 1.6.0. El repositorio de procesos BPEL corresponde a la implementación presentada en (Vanhatalo, 2006b). Para la implementación del analizador lingüístico empleado por el paquete de descubrimiento se utilizó la versión 2.0 de WordNet.

Nivel de Contexto: en este nivel se ubican los paquetes correspondientes a los repositorios de usuarios y dispositivos. La implementación del *Repositorio de Usuarios* se soportó sobre el motor de base de datos relacionales PostgresSQL 8.2, para realizar las consultar se empleó el API JDBC. Para el contexto de los dispositivos se emplearon dos librerías: *WALL* y *DELI*.

- *WALL* (Wireless Abstraction Library / Librería de Abstracción Móvil) es una librería de tags JSP que permite diseñar páginas Web que, dependiendo de las capacidades del dispositivo, entrega el contenido en formato WML, C-HTML, y XHTML Mobile Profile.
- *DELI* librería que permite a un Servlet Java conocer el contexto de entrega de un dispositivo a través de CCPP o UAProf. El contexto de entrega ayuda a los servidores a identificar en qué clase de entorno se está realizando la petición y con base en esto puede dar una respuesta más adecuada según el dispositivo.

Finalmente, se debe decir que las aplicaciones fueron desplegadas sobre un servidor de aplicaciones *Glassfish v3 Prelude* instalado en un sistema operativo Linux Ubuntu 9.04 server.

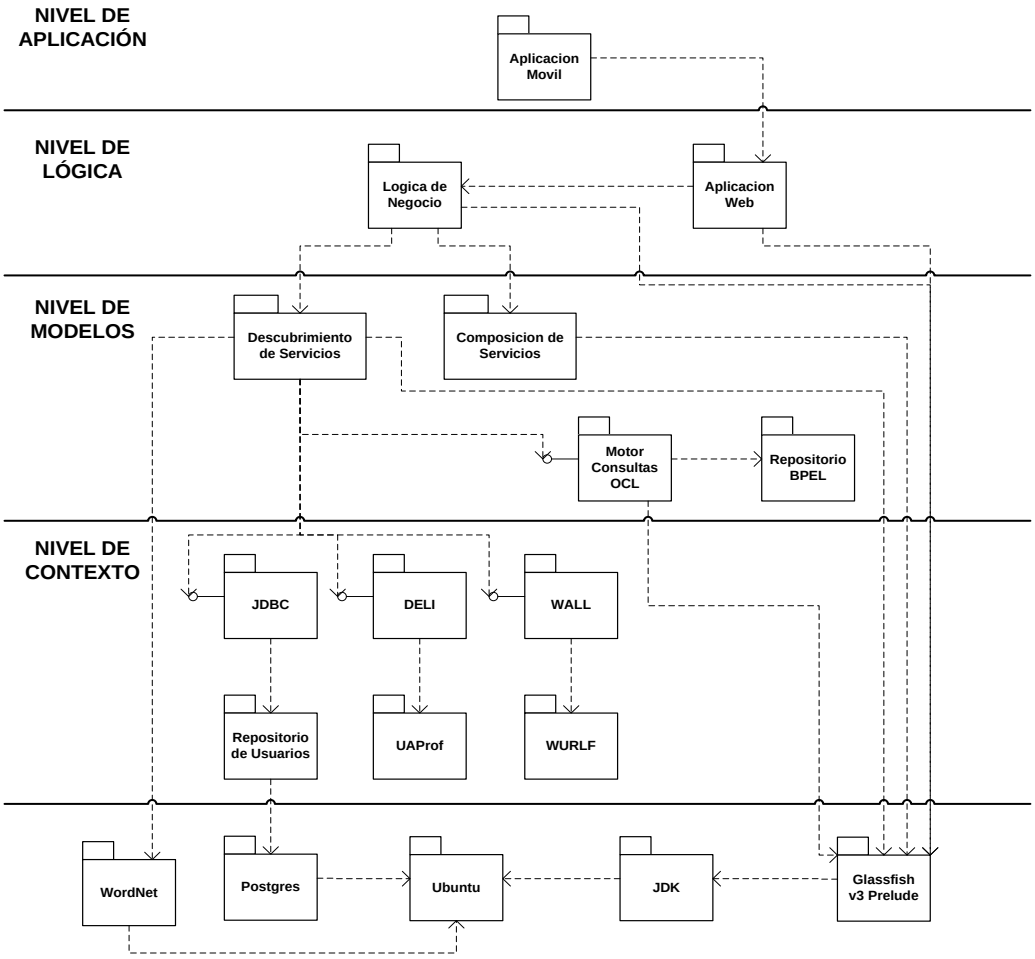

Figura 19. Diagrama de Paquetes de Aplicación de la Implementación de Referencia

3.5.4.2 Diagrama de Implantación

La Figura 20 muestra el diagrama de implantación de la arquitectura diseñada para la implementación de referencia. Se detallan los componentes incluidos en la implementación de referencia y su distribución en los nodos empleados.

En el contenedor Web se despliegan los componentes para el Descubrimiento y Composición de servicios, estos componente son accedido por el usuario a través de la Aplicación Web y la Lógica de negocio. El usuario utiliza un navegador Web para poder enviar las peticiones y recibir las respuestas desde la aplicación Web.

Los repositorios de modelos y contexto han sido distribuidos en diversos nodos indicando los protocolos que se emplean para realizar las consultas. De igual manera se detallan los componentes (DELI y WALL) que participan como medidores entre el componente de descubrimiento de servicios y los repositorios de dispositivos.

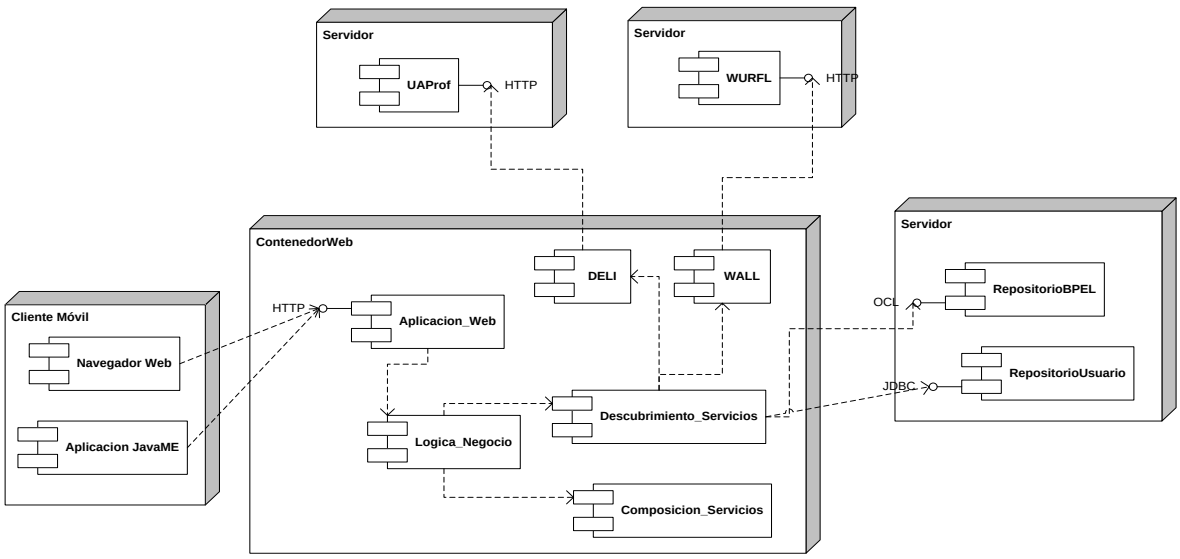

Figura 20. Modelo de Implantación de la Implementación de Referencia

3.5.4.3 Diagrama de Despliegue

El diagrama de despliegue expuesto en la Figura 21 describe la configuración de la implementación de referencia para su ejecución en un ambiente del mundo real, presentando la disposición física de los distintos nodos que intervienen en la composición del Sistema, los protocolos de comunicación empleados y la topología hardware de la implementación.

Los clientes móviles acceden a la plataforma mediante el navegador web o una aplicación Java ME, enviando una URL por medio de las cabeceras HTTP, la cual contiene capacidades básicas del dispositivo descritas según el estándar CC/PP. Ésta petición es llevada hasta el *Servidor Web de la Plataforma de Composición,* que identifica las capacidades básicas del dispositivo que está interactuando con la plataforma por medio de las cabeceras HTTP. Sin embargo, dicha información no es suficiente, es aquí donde interviene el *Repositorio de Dispositivos*, el cual determina el contexto de entrega completo del cliente móvil.

El contexto de entrega es construido con base en tres fuentes de información UAProf, WURFL y las cabeceras HTTP, lo que garantiza una descripción amplia y fiel de las capacidades del dispositivo. El orden en el cual el *Repositorio de Dispositivos* obtiene el contexto de entrega es el siguiente: El Servidor inicialmente usa la información que envía el *Servidor Web de Composición* por medio de las cabeceras HTTP, luego los datos ofrecidos por la librería WURFL y por último usa UAProf. Se ha considerado que este es el orden de fidelidad de la información, ya que el navegador debe conocer fielmente sus capacidades, y la librería WURFL ha sido construida por desarrolladores que mediante pruebas obtienen éstos datos; mientras que en el caso de UAProf suele obtenerse información que no detalla el producto, sino una categoría de dispositivos del fabricante.

Una vez el contexto de entrega del cliente móvil ha sido identificado, el usuario es habilitado para usar la plataforma, permitiéndole consultar su historial de servicios (consumidos y consultados) o ejecutar una búsqueda para recuperar nuevos servicios. Esto es posible gracias a los Repositorios de Usuarios y de Procesos del sistema, donde el primero gestiona los *Perfiles de Usuario*, registrando el comportamiento del mismo, servicios consumidos o historial y patrones de uso; y el segundo provee un poderoso mecanismo de consulta que permite obtener información de los procesos BPEL almacenados, mediante búsquedas en sus propiedades o metadatos relacionados.

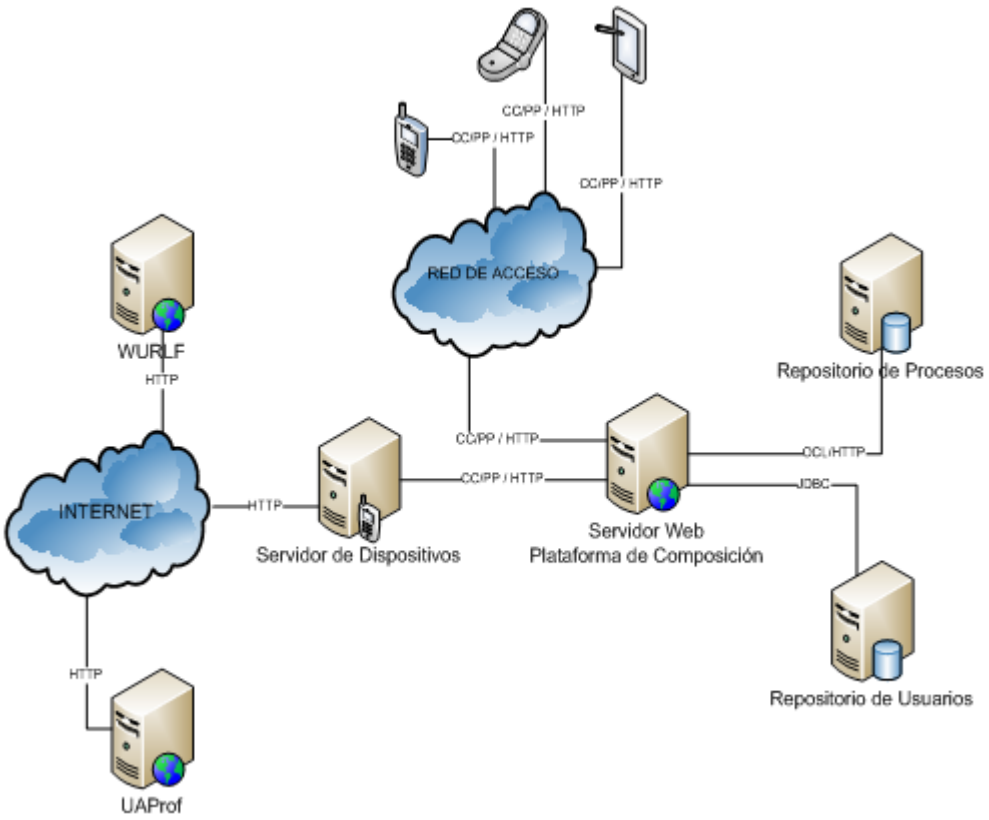

Figura 21. Modelo de Despliegue de la Implementación de Referencia

Protocolos de comunicación y conexión.

Los protocolos son el conjunto de reglas que especifican el intercambio de mensajes durante la comunicación entre las entidades que forman parte de una red. Los utilizados por la plataforma son los siguientes:

- **HTTP** (*HyperText Transfer Protocol*): el Protocolo de transferencia de hipertexto define la sintaxis y la semántica que utilizan los elementos software de la arquitectura web (clientes, servidores) para comunicarse. Es el protocolo de comunicación definido por los repositorios de procesos y de dispositivos (WURLF y UAProf) para realizar las consultas. En cuanto a los dispositivos móviles las cabeceras HTTP permiten identificar las capacidades básicas del dispositivo y realizar posteriores consultas a los repositorios.
- **OCL** (*Object Constraint Language*): este lenguaje permite construir consultas basadas en una representación de objetos e independientes del tipo de almacenamiento empleado en los repositorios (p.ej. bases de datos relacionales o base de datos XML). OCL es definido como el lenguaje de consulta para Repositorio de Procesos presentado en (Vanhatalo, y otros, 2006) y el cual es empleado en la implementación de referencia.
- **JDBC** (*Java Database Connectivity*): es una API que permite la ejecución de consultas sobre bases de datos relacionales desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede. Este protocolo es empleado para consultar el repositorio de usuario cuya implementación se soporto sobre una base de datos relacional.

Capítulo 4

Descubrimiento de Servicios en Ambientes Ubicuos

El presente capitulo expone de manera detallada la propuesta para realizar el descubrimiento de servicios en ambientes de computación ubicua. En primera instancia se realiza una descripción del lenguaje empleado para la descripción de modelos de comportamiento, en este caso particular BPEL. Posteriormente, se presenta la forma como se expresan los modelos de comportamiento a través de una representación formal, mostrando el proceso de transformación de un documento BPEL a una representación en grafos. Empleando dicha representación formal se construyen los algoritmos para el descubrimiento de servicios basados en el comportamiento requerido por el usuario y la información del contexto de entrega. Finalmente se muestran los resultados obtenidos al evaluar los algoritmos propuestos.

4.1 Ejemplo de Procesos de Negocio (BPEL)

Para el desarrollo de este capítulo se utiliza un ejemplo de proceso de negocio BPEL para explicar cada uno de los procedimientos empleados en el descubrimiento de servicios. El ejemplo plantea el siguiente escenario: “un viajero que desea realizar sus reservas en una aerolínea y un hotel para hospedarse el tiempo que dure su viaje. Para realizar estas operaciones el usuario debe suministrar la fecha de salida y regreso, el nombre de la aerolínea con la cual desea viajar y el nombre del hotel en que quiere hospedarse, como condición especial se requiere que se confirmen ambas reservas, de lo contrario deberá cancelarse el viaje. Para el caso exitoso de las reservas, se desea también rentar un vehículo para los días en los cuales el viajero se encuentre en la ciudad de destinos”.

La Figura 22 diagrama un proceso de negocio para realizar la tarea de usuario propuesta en el escenario descrito en el párrafo anterior. Por un lado se tiene el Cliente quien realiza la petición inicial al proceso de negocio, a partir de la cual se realizan las invocaciones a los servicios involucrados. En la petición inicial se envían los datos requeridos por el proceso, tales como las fechas de vuelo, hotel y aerolínea. Por otro lado están los proveedores de servicios, en este caso concreto se tienen tres servicios: Servicio de Aerolínea, Hotel y Renta de Carros. Sobre estos servicios se invocan las operaciones para las reservas del vuelo, habitación y vehículos, están interacciones son representadas como actividades básicas BPEL. Cada operación se nutre de los datos entregados en la petición inicial del usuario o a partir de los resultados entregados por las operaciones previamente invocadas. Igualmente, la información entregada por los servicios permite definir el flujo de ejecución del proceso.

4.2 Transformación de BPEL a Grafos

En éste apartado se describe la equivalencia entre una descripción BPEL y una representación formal de Grafos para procesos de negocio, de acuerdo a lo definido en (Corrales, 2008). En esta transformación, se implementa la estrategia de barrido presentada en (Mendling, y otros, 2005),

la idea central de este algoritmo es recorrer la estructura del flujo de control BPEL de arriba hacia abajo aplicando recursivamente un proceso de transformación específico para cada tipo de actividad estructurada.

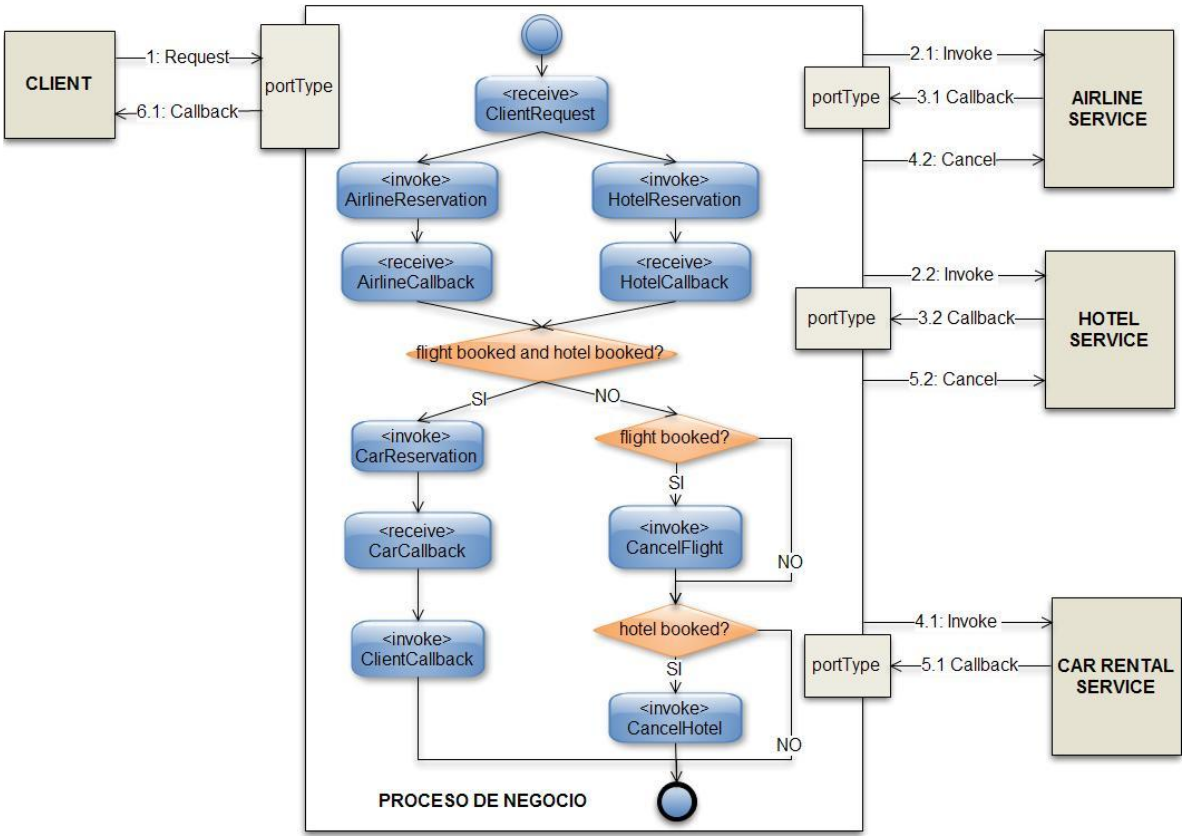

Figura 22. Ejemplo de un Proceso de Negocio

En la transformación se crea un grafo, que tiene un nodo de comienzo y uno o múltiples nodos de finalización. El grafo también posee dos clases de nodos: los nodos regulares, que representan las actividades y los conectores, que representan las reglas de separación o juntura del tipo XOR o AND utilizadas para representar las actividades estructuradas. Los nodos son conectados mediante aristas, las cuales pueden tener un *guard* opcional, los *guards* son condiciones que pueden ser evaluadas como verdaderas o falsas, y permiten determinar el flujo de ejecución del proceso. La Figura 23 muestra la forma como se representan las actividades estructuradas de BPEL con los nodos tipo conector AND y XOR. Los nodos conectores tienen dos atributos: ConnectorType (AND-split, AND-join, XOR-split, XOR-join) y ActivityType (la actividad estructurada BPEL desde la cual fue transformada).

Por su parte las actividades básicas son representadas como simples nodos dentro del grafos, dichos nodos poseen los siguientes atributos: *ActivityType* (*AT*), *Operation* (*Op*), *PortType* (*PT*), *PartnerLink* (*PL*), *Input y Output*. Una actividad básica BPEL puede ser definida formalmente de la siguiente manera:

Definición 1: Actividad, una Actividad A es definida como una tupla (AT, Op, PT, PL, Input, Output) donde:

- **AT:** es el tipo de actividad básica, es un elemento del conjunto T= {invoke, receive, reply, throw, wait}
- **Op**: especifica el nombre de la operación que se invoca a través de la actividad.
- **PT**: define el PortType a través del cual se interactúa con el servicio, relaciona la operación y los mensajes intercambiados
- **PL:** es una referencia concreta al servicio al cual pertenece la operación invocada y con el cual interactúa el proceso BPEL
- **Input**: define el mensaje de entrada de la operación, lo relaciona con una variable definida en el proceso BPEL
- **Output**: define el mensaje de salida de la operación, lo relaciona con una variable definida el proceso BPEL

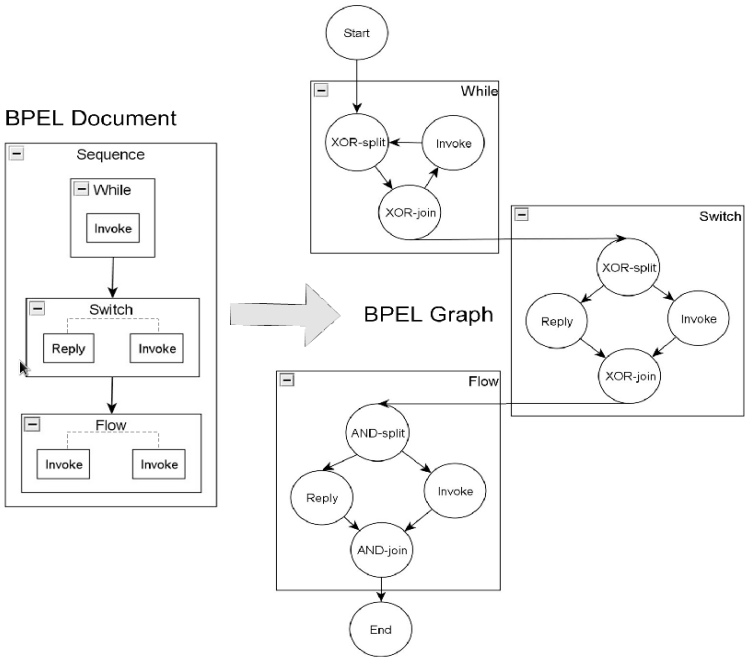

Figura 23. Correspondencia entre elementos BPEL y elementos de Grafos

La Figura 24 muestra la representación en forma de grafos del proceso de negocio presentado en la sección 4.1, se puede apreciar las actividades básicas expresadas como nodos del grafo y la representación de las estructuras complejas *Switch* y *Flow* por medio de los nodos tipo *AND* y *XOR.* Las secuencias se elaboran empleando las aristas que enlazan los diferentes nodos del grafo.

4.3 Mecanismos para el Descubrimiento de Servicios

En esta sección se presentan los diferentes mecanismos para realizar el descubrimiento de servicios en ambientes de computación ubicua. Para esto, se formaliza una serie de algoritmos que definen la manera como se articulan los módulos de la Arquitectura de Referencia para realizar el descubrimiento de servicios teniendo en cuenta el los requisitos del usuario y el contexto de entrega.

Inicialmente se presenta el enfoque utilizado para realizar el emparejamiento de actividades basándose en una representación formal de grafos, posteriormente se aborda la influencia del

contexto en la recuperación de servicios, y finalmente se presenta un algoritmo que reúne estos elementos para obtener una solución completa al problema de descubrimiento.

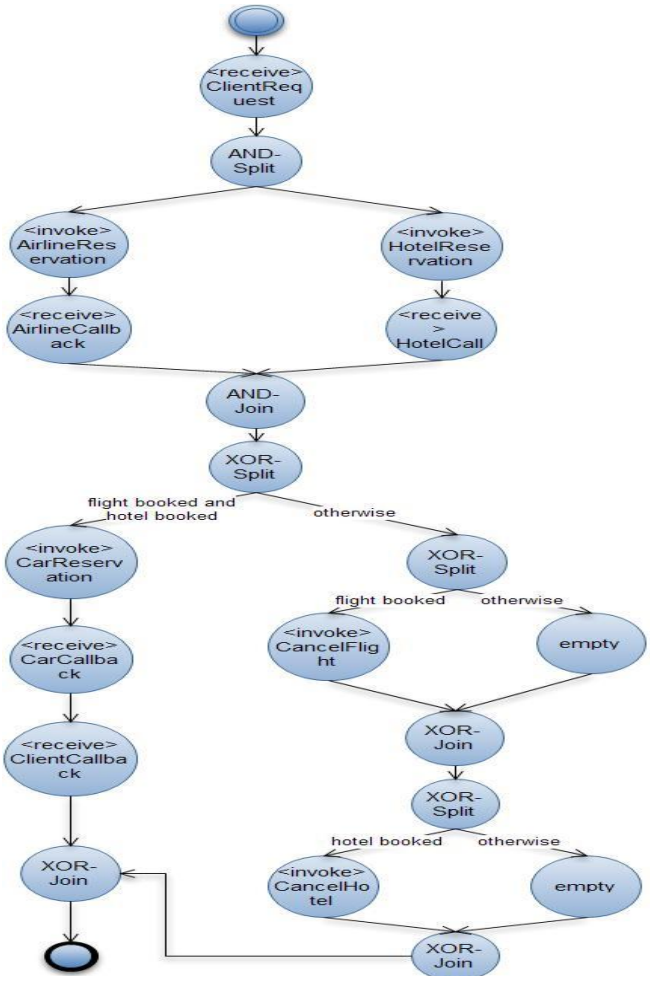

Figura 24. Proceso de Negocio representado como Grafo

4.3.1 Emparejamiento de Actividades Básicas

En esta sección se evidencia la aplicación de un modelo formal de representación de procesos en el emparejamiento atómico de actividades básicas BPEL. El Algoritmo 1 expresa el emparejamiento de actividades básicas, y se fundamenta en los trabajos de (Corrales, 2008) (Caicedo, y otros, 2009). Toma como entradas dos nodos, que representan actividades básicas de BPEL (Receive, Invoke, Reply), y calcula la distancia semántica entre ellos. Cada nodo posee cuatro atributos: ActivityType (AT), Operation (Op), PortType (PT) y PartnerLink (PL), de los cuales tres son considerados en el proceso de emparejamiento (Op, PT y PL), ya que el parámetro ActivityType permite identificar que los nodos a comparar sean del mismo tipo, este atributo es tomado en cuenta en el Algoritmo 2.

La función BasicActivityMatch, inicia dando prioridad a la comparación de Op, si las dos operaciones son similares (SimOperation > 0), se continua con el cálculo de la similitud de los demás parámetros (PT y PL) para estimar la distancia entre las dos actividades, DistanceNode. Los pesos Wop, Wpt y Wpl indican la contribución de la similitud de los atributos Op, PT y PL a la

similitud de las actividades (0 ≤ Wop ≤ 1, 0 ≤ Wpt ≤ 1 y 0 ≤ Wpl ≤ 1). Para calcular la similitud de los atributos se emplea la función Linguistic Similarity (LS), descrita en el Algoritmo 3.

Algoritmo 1. Emparejamiento de Actividades Básicas – BasicActivityMatch

INPUTS: (Nodei, Nodej) Nodei: Struct (*Opi, PTi, PLi, ATi*), Nodej: Struct (*Opj, PTj, PLj, ATj*)
OUTPUT: *DistanceNode*

BEGIN
Calculate Operation Similarity *SimOperation = LS(Opi, Opj)*
if SimOperation = 0 (different Operations) **then**
  return *DistanceNode* = 1
else
  Calculate PortType Similarity *SimPortType = LS(PTi, PTj)*
  Calculate PartnerLink Similarity *SimPartnerLink = LS(PLi, PLj)*
  Calculate *DistanceNode*

$$\text{DistanceNode} = 1 - \frac{w_{op} * \text{SimOperation} + w_{pt} * \text{SimPortType} + w_{pl} * \text{SimPartnerLink}}{w_{op} + w_{pt} + w_{pl}}$$

  return *DistanceNode*
end if

Antes de ejecutar un algoritmo de emparejamiento de los nodos de dos grafos (G_Q y G_T), es necesaria una etapa previa encargada de la clasificación, organización y filtrado de estos, de acuerdo a su tipo de actividad. De esta forma, sólo los nodos que pertenecen al mismo tipo de actividad en G_Q (*grafo Query*) y G_T (*grafo Target*), respectivamente, son comparados, (es decir: el conjunto de actividades *Invoke$_{syn}$, Invoke$_{asyn}$, Receive* y *Reply*). El algoritmo para la clasificación de tipos de actividades es presentado en (Suarez, y otros, 2010), ver Algoritmo 2. El trabajo (Suarez, y otros, 2010) corresponde a un trabajo de grado presentado en la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca, enmarcado en el descubrimiento de servicios en ambientes ubicuos, y el cual soporto dicho ítem en la presente tesis de maetría.

Algoritmo 2. Clasificar nodos por tipo de actividad – ClassifierNodes

INPUTS: BPELGraph: Struct (Star, Receive, Invoke$_{syn}$, Invoke$_{asyn}$, Reply, End)
OUTPUT: set of nodes organized by Activity Type: setInvoke$_{syn}$, setInvoke$_{asyn}$, setReceive and setReply

BEGIN
for each activity node n **in** BPELGraph **do**
  if n ∈ Star **then**
    not consider
  else if n ∈ Receive AND n ∉ setReceive **then**
    add n to setReceive
  else if n ∈ Invoke$_{syn}$ then AND n ∉ setInvoke$_{syn}$ **then**
    add n to setInvoke$_{syn}$
  else if n ∈ Invoke$_{asyn}$ then AND n ∉ set Invoke$_{asyn}$ **then**

add n to setInvoke$_{asyn}$
else if n ∈ Reply then AND n ∉ setReply **then**
add n to setReply
else if n ∈ End **then**
not consider
end if
end for

***Linguistic Similarity* (*LS*):** calcula la similitud lingüística entre dos etiquetas basándose en sus nombres. Para obtener esta medida se utilizan los algoritmos *Ngram, Check Synonym* y *Check Abbreviation.* El algoritmo *Ngram* estima la similitud de acuerdo al número común de *qgramas* entre las etiquetas (Angell, y otros, 1983). El algoritmo *Check Synonym* utiliza el diccionario lingüístico WordNet (Miller, 1995), para identificar sinónimos, mientras el algoritmo C*heck Abbreviation* usa un diccionario de abreviaciones adecuado al dominio de aplicación. Si todos los algoritmos entregan un valor de 1 existe una coincidencia exacta entre las etiquetas, si entregan un valor de 0 no hay similitud entre las palabras. Si los valores entregados por *Ngram* y *Check Abbreviation* son iguales a 0 y el valor de *Check synonym* está entre 0 y 1, el valor total de la similitud es igual a *Check Synonym.* Finalmente, si los tres algoritmos arrojan un valor entre 0 y 1, la similitud lingüística es el promedio de los tres. La función LS es descrita en el Algoritmo 3.

Algoritmo 3. Función Linguistic Similarity (LS)

$$\boldsymbol{LS} = \begin{cases} \mathbf{1} & \boldsymbol{si}\ (\ \boldsymbol{m1} = \mathbf{1} \vee \boldsymbol{m2} = \mathbf{1} \vee \boldsymbol{m3} = \mathbf{1}\ ) \\ \boldsymbol{m2} & \boldsymbol{si}\ (\mathbf{0} < m\mathbf{2} < 1\ \wedge\ m\mathbf{1} = \boldsymbol{m3} = \mathbf{0}) \\ \mathbf{0} & \boldsymbol{si}\ (\ \boldsymbol{m1} = \boldsymbol{m2} = \boldsymbol{m3} = \mathbf{0}\ ) \\ \dfrac{\boldsymbol{m1} + \boldsymbol{m2} + \boldsymbol{m3}}{\mathbf{3}} & \boldsymbol{si}\big(\boldsymbol{m1}, \boldsymbol{m2}, \boldsymbol{m3}\ \in (\mathbf{0}, \mathbf{1})\big) \end{cases}$$

Donde: **m1** = Sim(NGRAM), **m2** = Sim(Synonym Matching), **m3** = Sim(Abbreviation Expansion)

4.3.2 Información del Contexto

En este apartado se incorpora la información del contexto al proceso de descubrimiento de servicios, se establece la forma como se manipulan los datos almacenados en los repositorios para obtener la información necesaria para determinar qué servicios son más pertinentes al momento de responder al requisito del usuario tomando en cuenta su perfil y los dispositivos que emplea.

4.3.2.1 Contexto del Proceso

El Repositorio de Procesos, soportado por el trabajo presentado en (Vanhatalo, y otros, 2006), permite almacenar documentos BPEL y otro tipo de archivos basados en el estándar XML. Este repositorio tiene como característica la posibilidad de extender el tipo de contenidos que almacena, definiendo un nuevo meta-dato que permita expresar la información con la cual se quiere enriquecer los procesos de negocio BPEL. Por lo tanto, en este caso es necesario definir un meta-dato XML que describa el contexto en el cual el servicio puede ser consumido. Dicha información es entregada por los proveedores o desarrolladores de servicios. Cumpliendo así con

unas de las variables de contexto establecidas en (Steller, y otros, 2006) denominada: *requerimientos de contexto del servicio.*

Para que el *Repositorio de Procesos* soporte la característica de contexto definida anteriormente, es necesario extender su funcionalidad mediante un nuevo modelo EMF (*Eclipse Modeling Framework*) que permita describir las restricciones del servicio especificadas por el proveedor. El repositorio puede ser extendido por medio de tres tipos de modelos: un modelo Rational Rose, un esquema XML o anotaciones Java (Vanhatalo, 2006b). En esta tesis, para definir el contexto de los procesos se ha utilizado un esquema XML en el cual se definen las características y atributos que permiten expresar las restricciones o requerimientos del contexto del servicio. El esquema es definido a partir de la dimensión de dispositivos establecida en el meta-modelo de (Guerrero, y otros, 2010) tomando atributos para las capacidades requeridas por los servicios a nivel de capacidades software, hardware y de sistema operativo. Para el caso de estudio, se limita la descripción del contexto de los procesos a un solo atributo denominado tipo acceso, el cual está asociado a las características de red requeridas para el acceso a los servicios contenidos en un proceso. Se debe aclarar que la descripción del contexto de los procesos puede ser extendida y tener en cuenta un mayor número de atributos agregando etiquetas XML que los representen. El esquema definido para representar el contexto del servicio es mostrado en la Tabla 5.

El meta-dato se definió por medio de un esquema de datos muy básico, el cual permite establecer una estructura compleja denominada *networkaccesses,* la cual posee un conjunto de elementos llamados *networkaccess* que son del tipo *EString*. La intención de estos elementos es enriquecer los procesos de negocio especificando los tipos de redes de acceso que soportan los proveedores de servicios e identificar si los dispositivos del usuario tienen la capacidad de conectarse a dichas redes y consumir los servicios. Estos datos del contexto son consignados en un tipo de archivo adicional, el cual acompaña al archivo BPEL y las WSDL publicadas por el proveedor, el tipo de contenido tienen una extensión del tipo *context* y es referenciado desde los archivos de descripción asociados a cada organización.

Tabla 5. Meta-Dato de los Requerimientos del Contexto del Servicio

```
<?xml version="1.0" encoding="UTF-8"?>
<emfcontext xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:emfcontext="http:///com/ibm/bpia/repository/extensions/emfcontext.ecore"
name="Holidays">
    <networkaccesses>
        <networkaccess>BLUETOOTH</networkaccess>
        <networkaccess>WIFI</networkaccess>
        <networkaccess>EGPRS</networkaccess>
        <networkaccess>GSM</networkaccess>
    </networkaccesses>
</emfcontext>
```

4.3.2.2 Verificación del Contexto de Entrega

Una vez definidos los atributos que definen los requerimientos del contexto de los servicios se puede establecer una función que permite aprovechar esta información y contrastarla con las características de los dispositivos del usuario. La función se ha denominado Verificar Contexto (*CheckDeliveryContext*) y se especifica en el Algoritmo 4.

La función Verificar Contexto toma la lista del ranking de los servicios obtenidos en la fase de emparejamiento de actividades básicas y verifica los requerimientos de contexto de los servicios recuperados en el *Repositorio de Procesos*. Las restricciones de contexto del usuario son determinas a través del *Repositorio de Dispositivos* mediante las cabeceras HTTP, UAProf y WURFL. Así, las características de los dispositivos son relacionadas con las restricciones del proveedor de servicios (protocolos de acceso y redes). En primer lugar, el perfil del dispositivo es obtenido desde el *Repositorio de Dispositivos*, luego, el descriptor *features.context* (archivo que contiene las restricciones publicadas por el proveedor) es recuperado para cada servicio contenido en el *Repositorio*; finalmente, la función verifica que el dispositivo es compatible con las tecnologías de acceso definidas por el proveedor de servicios en el descriptor features.context.

Algoritmo 4. Función Verificar Contexto - CheckDeliveryContext

INPUTS: (listRankedServices)
OUTPUT: rankedFilteredServices

BEGIN
Device Profile *deviceProfile* = *lookupDeviceProfile(deviceURL)*
for each node in *listRankedServices* **do**
  EmfContext *features.context* = *lookupFeatures.context (node.URI)*
  for each networkaccess in *features.context* **do**
    if *deviceProfile contains networkaccess* (network access supported) **then**
      Add node to rankedFilteredServices
      break for
    end if
  end for
end for

return *rankedFilteredServices*

4.3.2.3 Contexto del Usuario

La definición de los algoritmos para la inclusión del contexto del usuario es tomada del trabajo presentado por (Suarez, y otros, 2010). De (Guerrero, y otros, 2010) se tiene que los diferentes conjuntos de datos que conforman el Perfil de usuario y que se han denominado Dimensiones del Perfil de usuario son: datos personales y dominio de intereses. En la Figura 25, se presenta el meta-modelo general del Perfil de usuario propuesto. A continuación se explica cada una de estas dimensiones.

- Dominio de interés: la información de esta dimensión obtenida de un usuario depende del ámbito de aplicación. Varios estudios utilizan diferentes métodos de obtención y manipulación de la información dependiendo de la aplicación: Web mining: (Tocarruncho, y otros, 2007), clustering: (Zhang, y otros, 2009), vectores de registros (logs) de aplicaciones (Abbar, y otros, 2008), etc., cada uno de estos mecanismos genera un conjunto de parámetros con sus posibles valores para un dominio de interés dado. La definición de estos parámetros y valores no se establece en éste trabajo debido al alto nivel de análisis y la desvinculación a un ámbito específico.

- Datos personales: la dimensión de los datos personales se divide en dos categorías, la de los Datos de Identificación y los Datos Demográficos, esto se establece debido a los requerimientos de protección de la identificación del usuario.

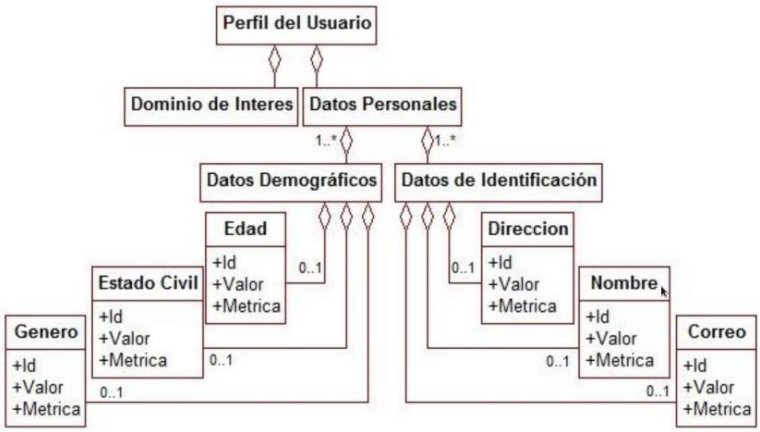

Figura 25. Meta-modelo del perfil de usuario

Las anteriores descripciones se almacenan en el Repositorio de Usuarios. A partir de esta descripción, el sistema de descubrimiento de servicios realiza un proceso de sugerencia de servicios que tiene como finalidad recomendar un grupo de servicios ya utilizados por el usuario y/o personas con perfiles similares, y así disminuir el tiempo que toma el descubrimiento realizado sobre todo el repositorio de procesos. Este procedimiento de búsqueda y recomendación precede a la fase de descarte realizada por la función de verificación de contexto, expuesta en la sección anterior. El primer paso en la fase de sugerencia es encontrar usuarios similares, para después utilizar los servicios consultados y consumidos por ellos.

La función FindSimilarProfiles es la encargada de organizar una búsqueda sobre los perfiles contenidos en el repositorio de usuario, ver Algoritmo 5. Los perfiles de usuario están definidos de acuerdo al meta-modelo de perfil de usuario ya presentado. Para determinar la distancia entre los perfiles la función FindSimilarProfiles hace uso de la función basicProfileMatch.

Algoritmo 5. Function FindSimilarProfiles

INPUTS: (*queryProfile*)
OUTPUT: *similarProfiles List*

```
BEGIN
List Profiles existingProfiles = lookupServiceRepository(non-operational information)
 for each profilei in candidateService do
    DistanceProfile distance = basicProfileMatch(profilei, queryProfile)
    if distance < 1 then
        Add (profilei, distance) to similarProfiles
    end if
end for
return similarProfiles.
```

La función BasicProfileMatch (Algoritmo 6) calcula la distancia entre una pareja de perfiles. Este cálculo se realiza sobre valores de características que describen el perfil de usuario. Debido al aumento en procesamiento y tiempo, se descartó la posibilidad de utilizar razonadores en esta comparación. Por eso la distancia es calculada utilizando un diccionario léxico como WordNet, de forma similar al matching básico de actividades. Esta función comienza evaluando la similitud de las edades *SimAge,* de la misma manera, luego evalúa la similitud entre el estado civil *SimCS* y la similitud de género *SimG* continua con el cálculo de la similitud de valores de la lista de Dominios de interés SimDoIij para estimar la similitud SimDoI general, por último se calcula distancia entre los dos perfiles, *DistanceProfile*. Los pesos *Wage*, *Wcs, Wg y* W_{DoI} indican la contribución de la similitud de los atributos DoIlisti, Agei, CSi, y Gi a la similitud de los perfiles ($0 \leq Wage \leq 1$, $0 \leq Wcs \leq 1$, $0 \leq Wg \leq 1$ y $0 \leq W_{DoI} \leq 1$). Para calcular la similitud de los valores de DoI se emplea la función *Linguistic Similarity* (*LS*), descrita en el Algoritmo 3.

Algoritmo 6. Function BasicProfileMatch

INPUTS: (Profilei, Profilej) Profilei: Struct (DoIlisti, Agei, CSi, Gi), Profilei: Struct (DoIlisti, Agei, CSi, Gi)
OUTPUT: DistanceProfile

BEGIN
Calculate Age Similarity *SimAge* = 1-{[|*Agei – Agej*|/[(*Agei*+*Agej*)/2]}
SimCS =0
SimG =0
SimDoI = 0

if CSi = CSj **then**
    SimCS =1
end if
if Gi = Gj **then**
    SimG =1
end if
for each value *vi* in *DoIlisti* **DO**
    for each value *vj* in *DoIlist*j **do**
        Calculate DoIij Similarity *SimDoIij = LS(vi,vj)*
        if *SimDoIij* > *SimDoI* **then**
            SimDoI = *SimDoIij*
        end if
    end for
end for

$$\text{DistanceProfile} = 1 - \frac{w_{age} * \text{SimAge} + w_{cs} * \text{SimCS} + w_{g} * \text{SimG} + w_{\text{DoI}} * \text{SimDoI}}{w_{age} + w_{cs} + w_{g} + w_{\text{DoI}}}$$

En el Algoritmo 7 se expone el procedimiento para conseguir los servicios sugeridos antes del refinamiento de la búsqueda por parte de la función de verificación de contexto. Se presentan dos funciones que evidencian la utilidad del perfil de usuario en el proceso de sugerencia de servicios;

listRankedServicesQueried y consumedNodes. Internamente estas dos funciones utilizan a la función findSimilarProfiles.

El Algoritmo 7 empieza verificando si el nodo de consulta ya ha sido previamente solicitado, en caso afirmativo, recupera del repositorio de usuarios la información de los servicios retornados en esa oportunidad. En este punto, es importante distinguir la diferencia entre un servicio de consulta y un servicio sugerido. El servicio de consulta es entregado por el usuario, contiene una descripción de sus requerimientos, y puede ser real o no. En cambio los servicios sugeridos, obligatoriamente deben poseer una implementación. La importancia de los servicios consultados radica en que si bien no poseen una implementación real, tienen relacionada una lista de servicios reales que les fueron sugeridos. Pero si el servicio de consulta no ha sido solicitado en ocasiones anteriores, se realiza una primera búsqueda sobre aquellos nodos que han sido consumidos por el usuario y/o usuarios con perfiles similares. Estos nodos son organizados con base en su distancia al nodo de consulta queryNode. Una vez terminada esta organización la función badSuggest verifica que los nodos de listRankedServices cumplan con unas condiciones (superando un umbral y un número de servicios, mínimos requeridos por el usuario) para ser entregados al usuario como los más aproximados a sus requerimientos. Si esta lista no cumple con las condiciones, la búsqueda comienza desde cero analizando todo el repositorio de servicios.

Algoritmo 7. Servicios Sugeridos - SuggestServices

```
INPUTS: (queryNode, usrProfile)
OUTPUT: listRankedServices

BEGIN
if searchServicesPreviouslyConsulted (queryNode) then
    listRankedServices = listRankedServicesQueried(queryNode)
else
    List Nodes suggestServices = consumedNodes(usrProfile)
    for each nodei in suggestService do
        DistanceNode distance = basicActivityMatch(nodei, queryNode)
        if distance < 1 then
            Add ( nodei, distance) to listRankedServices order by distance
        end if
    end for
end if

if badSuggest(listRankedServices) then
    listRankedServices = null
    List Nodes candidateService = lookupServiceRepository(non-operational information)
    for each nodei in candidateService do
        DistanceNode distance = basicActivityMatch(nodei, queryNode)
        if distance < 1 then
            Add ( nodei, distance) to listRankedServices order by distance
        end if
    end for
end if
return listRankedServices
```

4.4 Experimentación y Evaluación

Inicialmente en esta sección se presenta la metodología empleada para la experimentación, posteriormente las medidas que se establecen como criterios de evaluación, y por último se consignan los resultados y el análisis respectivo.

4.4.1 Metodología de Experimentación

Para medir el desempeño del algoritmo de descubrimiento de servicios se procedió de la siguiente manera:

1. Se construyó un banco de pruebas para comparar un conjunto de actividades de consulta contra un repositorio de servicios, este banco de pruebas constituye una verdad absoluta sobre la similitud de las actividades comparadas.
2. Se compararon los resultados arrojados por el algoritmo de descubrimiento contra el banco de pruebas, con el objeto de medir su desempeño, para ello se definieron previamente los criterios para cuantificar el rendimiento del sistema.

Banco de Pruebas.
Un banco de pruebas puede definirse como un proceso sistemático y continúo para evaluar comparativamente los productos, servicios y procesos de trabajo en organizaciones (Spendolini, y otros, 1994). Con el propósito de tener una base para comparar los resultados del algoritmo de búsqueda de actividades de procesos de negocios se estableció un banco de pruebas conformado a partir de 30 actividades básicas BPEL agrupadas en 5 dominios: Vacaciones (5 actividades), Compras (7), Pagos (4), Disponibilidad de productos (7) e Información de productos (7).

El banco de pruebas se creó comparando las 30 actividades básicas BPEL contra las 144 actividades almacenadas en el repositorio de servicios. Las comparaciones fueron realizadas por 5 evaluadores, obteniendo 5530 comparaciones por cada uno, para un total de 27.650 comparaciones. Las comparaciones realizadas para las actividades BPEL evaluaron la similitud de cinco atributos: Activity Type, Operation, Portype, PartnerLink y Access Type, ver Figura 26, los cuatro primeros atributos pertenecen una actividad básica en un documento BPEL y son empleados por el algoritmo de comparación explicado en la sección 4.3.1. El atributo Access Type corresponde a una extensión que se realizó a la descripción de los servicios, el cual fue empleado por las funciones de verificación del contexto de entrega. El evaluador realizó la comparación entre las actividades asignando una calificación a cada uno de los atributos según su similitud. El valor de la calificación está entre 0 y 5, 0 mínima similitud y 5 máxima similitud. El experto evaluador fijó un peso de acuerdo a la importancia de cada uno de los atributos, la suma total de los pesos debe ser igual a 1.

El valor de similitud para cada comparación es calculado de la siguiente manera:

a. Evaluación del Emparejamiento por usuario:

$$EMu = \sum_{i=1}^{n} (Wi * Sui) \quad (1)$$

Donde: **Wi** (es el peso asignado por el evaluador según el grado de relevancia del atributo del servicio evaluado), **Sui** (Calificación) y **n** (cantidad de atributos a considerar)

El valor *EMu* es la similitud estimada por un evaluador para una pareja de servicios.

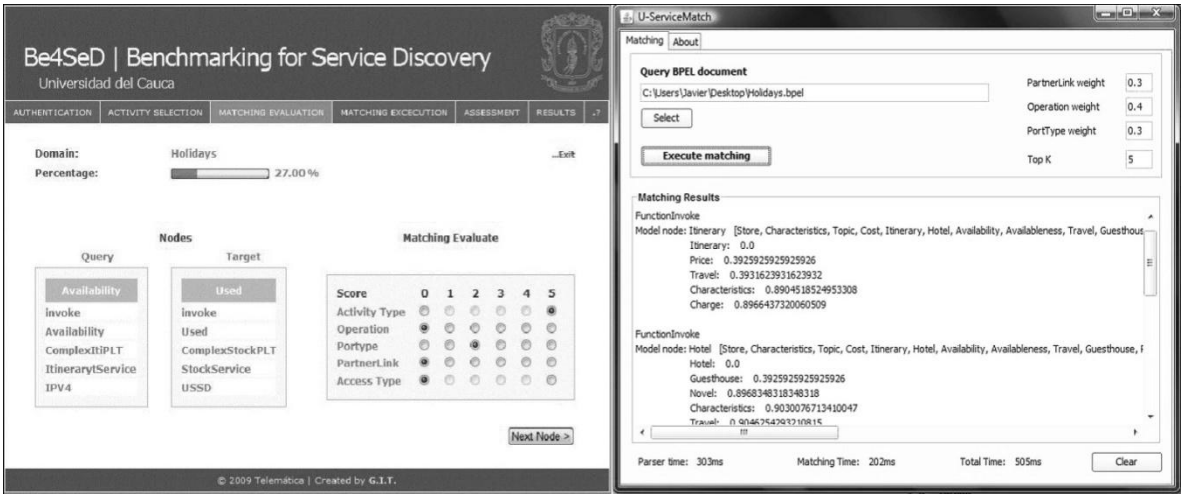

Figura 26. Evaluación de Similitud de Actividades

La media de similitud para cada uno de los servicios evaluados es la siguiente:

b. Evaluación Total:

$$TE(EM) = \frac{\sum_{u=1}^{n} EMu}{n} \qquad (2)$$

Donde **EMu** es la similitud de una comparación y **n** es el número de evaluadores.

Para cada una de las 30 actividades evaluadas en el banco de pruebas se obtuvo una lista ordenada conformada por las 144 actividades del repositorio con su respectivo valor de similitud media (*TE(Em)*), identificando las actividades relevantes que debieron ser recuperadas del repositorio. De esta forma se obtuvo el banco de datos que permitió realizar las medidas de desempeño del algoritmo de emparejamiento.

4.4.2 Análisis Estadístico

El desempeño general del sistema se establece utilizando las medidas: *recall* (*r*), *precisión* (*p*), *overall* (*o*), *top-k precisión* (P_k) *y p-presición* (P_p). Para evaluar la calidad del algoritmo de descubrimiento, se comparan los servicios (*P*) retornados por el *Algoritmo* con los servicios (R) obtenidos en el *Banco de Servicios*. De esta forma, se determina un conjunto de verdaderos positivos (*I*), servicios correctamente identificados; igualmente se establece un conjunto de falsos positivos, servicios falsos recuperados (*F = P/I*), y falsos negativos, es decir servicios relevantes no recuperados (*M = R/I*) (Corrales, 2008). $Retrel_k$ es el conjunto de servicios relevantes para un *top k* de servicios recuperados, mientras *Rel-p* determina cuantos de los servicios de $Retrel_k$ están en la misma posición del ranking de referencia del *Banco de Servicios* (Zhang, y otros, 2004). Con base en la cardinalidad de estos conjuntos se tiene:

$$p = \frac{|I|}{|P|} \qquad (3)$$

$$r = \frac{|I|}{|R|} \tag{4}$$

$$o = r * \left(2 - \frac{1}{p}\right) \tag{5}$$

$$p_k = \frac{|Retrel_k|}{k} \tag{6}$$

$$p_p = \frac{[Retrel_k]Rel - p}{k} \tag{7}$$

La medida *precisión* estima la fiabilidad de los servicios relevantes recuperados por el algoritmo, en tanto *recall* especifica el porcentaje de los servicios relevantes recuperados. Por su parte la medida *overall* valora la calidad del emparejamiento, teniendo en cuenta el esfuerzo necesario para la eliminación de falsos positivos y los servicios no recuperados. Las medidas establecidas anteriormente se calculan para cada una de los servicios empleados en el *Banco de Servicios*. Para estimar la *precisión* y el *recall* de todo el sistema, se emplean los métodos *macro-promedio* y *micro-promedio* (Lewis, 1992), así:

Macro-promedio: es la media de la precisión y recall de los emparejamientos individuales.

$$P = \frac{\sum_{i=1}^{n} p_i}{n} \tag{8}$$

$$R = \frac{\sum_{i=1}^{n} r_i}{n} \tag{9}$$

Donde: *n* es el número de emparejamientos realizados.

Micro-promedio: tiene en cuenta los verdaderos positivos y los falsos positivos. La precisión y el recall se calculan utilizando los valores globales.

$$P = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n}(TP_i + FP_i)} \tag{10}$$

$$R = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n}(TP_i + FN_i)} \tag{11}$$

Donde: *TPi*: Son los verdaderos positivos, *FPi*: Son los Falsos positivos, *FNi*: Son los Falsos Negativos.

A partir de los resultados obtenidos, se realizó un completo análisis estadístico que permitió determinar la calidad del algoritmo de emparejamiento de servicios. Este análisis se fundamentó en la evaluación de las siguientes relaciones: *Precision vs. Recall, Overall, K-Precision vs. K, P-Precision vs. K,* aplicadas en tres escenarios: i) evaluación de las medidas de desempeño comparando servicios de entrada contra los de un mismo dominio contenidos en el repositorio, ii) comparación de los servicios de entrada contra aquellos almacenados que pertenecen a un dominio diferente, y iii) comparación de los servicios de consulta contra todos los servicios contenidos en el repositorio. Los resultados de las pruebas se presentan en la sección 4.4.5.

4.4.3 Criterios de Evaluación

Con el fin de asegurar y garantizar el correcto funcionamiento de la implementación, se realizaron pruebas que permiten determinar la calidad de los resultados (eficacia) y el rendimiento la plataforma (eficiencia).

Se definieron pruebas enfocadas a la evaluación de dos aspectos fundamentales: la calidad de los resultados y el rendimiento de la plataforma. El primero, evalúa la eficacia de los algoritmos que intervienen en el proceso de recuperación y emparejamiento de la plataforma, y el segundo evalúa los tiempos de respuesta de las solicitudes realizadas a la plataforma.

Finalmente la evaluación y clasificación de los resultados de las pruebas, se realizó a partir de un conjunto de reglas que determinan los criterios de tiempos de respuesta para aplicaciones que están directamente relacionadas a las características cognitivas de los humanos (Joines, y otros, 2002):

- Los usuarios no notan un retardo de menos de 0.1 segundos. Por tanto, un tiempo de respuesta que esté bajo el umbral de 0.1 segundos se clasifica como óptimo.
- Un tiempo de respuesta de menos de 1 segundo no interrumpe el flujo del pensamiento de un usuario, pero deja notar un retardo. Por tanto, un tiempo de respuesta que esté bajo el umbral de 1 segundo se clasifica como bueno.
- Los usuarios aún esperarán la respuesta si está por debajo del umbral de 10 segundos, pero el retardo es notorio. Por tanto, un tiempo de respuesta que esté bajo el umbral de 10 segundos se clasifica como aceptable.
- Después de 10 segundos los usuarios pierden la concentración y no continúan esperando la respuesta del sistema. Por tanto, un tiempo de respuesta que esté por encima del umbral de 10 segundos se clasifica como deficiente.

4.4.4 Plan de Pruebas

Las pruebas de calidad y rendimiento, se hicieron sobre los módulos que realizan las funcionalidades más importantes y críticas de la fase de descubrimiento, estas son: Recuperación de Actividades Básicas BPEL y Recuperación de Actividades Básicas BPEL adaptado al contexto.

A continuación en la Tabla 6 se describen las pruebas de eficacia y eficiencia realizadas a las aplicaciones software que conforman la plataforma de descubrimiento.

Tabla 6. Plan de Pruebas

Recuperación de Actividades Básicas BPEL	**Prueba de Calidad de los Resultados PC1:** determina la calidad de los resultados arrojados por la aplicación, y por ende la eficacia de los algoritmos que intervienen en el proceso de emparejamiento. **Prueba de rendimiento PR1**: Determina el tiempo que tarda el módulo de recuperación en encontrar los servicios más similares para un determinado requerimiento del usuario.
Recuperación de Actividades Básicas BPEL adaptado al Contexto	**Prueba de Calidad de los Resultados PC2:** determina la calidad de los resultados arrojados por la aplicación, y por ende la eficacia de los algoritmos que intervienen en el proceso de emparejamiento, los cuales consideran el contexto, tanto de los requerimientos del servicio, como las restricciones del usuario.

	Prueba de rendimiento PR2: determina el tiempo que tarda el módulo de recuperación en encontrar los servicios más similares para una determinada solicitud, considerando el contexto del usuario.

4.4.5 Resultados y Discusión

Se implementó un sistema de descubrimiento de servicios sobre un repositorio de documentos BPEL (Vanhatalo, y otros, 2006), el cual almacena 17 archivos BPEL y cuenta con un total de 144 actividades básicas BPEL. El algoritmo de emparejamiento se implementó en el lenguaje Java y los experimentos fueron realizados en un computador con procesador Pentium 4 2,30GHz, 1.000 MB de RAM bajo el S.O. Linux Ubuntu. En el Anexo B se encuentran consignadas todas las medidas realizadas para estimar el rendimiento y calidad de los algoritmos de descubrimiento.

4.4.5.1 Prueba de Rendimiento PR1

En esta parte, se presenta los resultados del estudio del tiempo de ejecución del algoritmo de matching de actividades básicas BPEL. Para esta medida se contó con 17 archivos BPEL publicados en el Repositorio de Servicios. Se tomaron 5 documentos Query, conformados por actividades básicas, denominadas nodos Query que fueron comparados con los nodos Target, encontrados en el Repositorio de Servicios (ver Tabla 7).

Tabla 7. Archivos de Consulta BPEL

BPEL Query Files	Actividades Básicas
Holidays	5
Payment	4
ProductDisponibility	7
ProductInformation	7
Purchase	7
TOTAL	30

De la Figura 27 se puede concluir que el rendimiento del sistema es proporcional al número de documentos BPEL publicados en el repositorio de servicios. Específicamente, el tiempo de respuesta del algoritmo de emparejamiento propuesto varía con relación a los servicios publicados en el repositorio. Igualmente el Transformador BPEL – Grafos presenta un comportamiento similar, a medida que los nodos de entrada del sistema se van incrementando, el tiempo también lo hace. Este comportamiento es esperado, ya que el proceso de transformación de BPEL a grafos no es trivial.

Los resultados de rendimiento del proceso de emparejamiento fueron linealizados, generalizando su comportamiento en función del número de nodos consultados, y de esta manera se evaluó su rendimiento de acuerdo con lo expuesto en (Joines, y otros, 2002). Así, se determinó que la propuesta tiene un comportamiento:

- **Óptimo** (tiempo de respuesta menor o igual a 0,1s.): cuando el número de nodos comparados es menor o igual a 51,2.

- **Bueno** (tiempo de respuesta entre 0,1s. y 1s.): cuando el número de nodos comparados es mayor que 51,2nodos y menor que 527,9.
- **Aceptable** (tiempo de respuesta entre 1s. y 10s.): cuando el número de nodos comparados es mayor que 527,9nodos y menor que 5294,8.
- **Deficiente** (tiempo de respuesta mayor a 10s.): cuando el número de nodos comparados es superior a 5294,8.

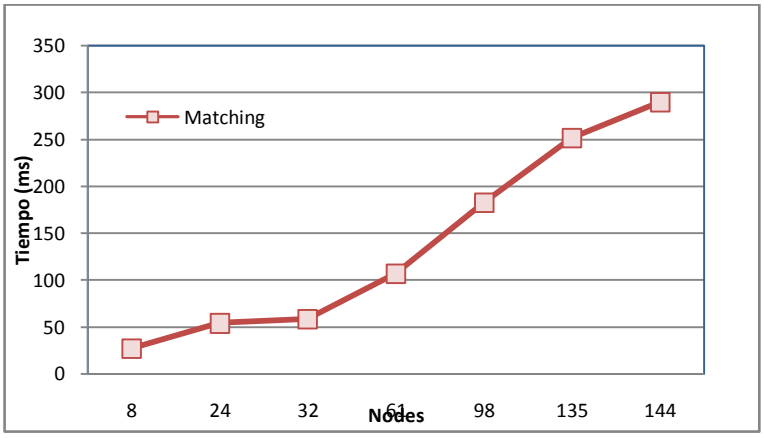

Figura 27. Rendimiento del proceso de Emparejamiento de las actividades básicas BPEL

4.4.5.2 Prueba de Calidad de los Resultados PC1

Se evaluaron las medidas de *precisión, recall, total, Top-k precisión, P-Precisión K* tomando como referencia un banco de pruebas de 30 actividades, para las cuales se obtuvieron las actividades más similares de acuerdo al algoritmo de descubrimiento. Los valores globales de las medidas se calcularon por medio de las técnicas de Macro-promedio y Micro-promedio.

Para el análisis de los resultados se plantearon tres escenarios: i) evaluación de las medidas de desempeño comparando las actividades de entrada contra las actividades del repositorio que pertenecen al mismo dominio (caso 1), ii) evaluación comparando las actividades de entrada contra las actividades almacenadas que pertenecen a un dominio diferente (caso 2), y iii) evaluación comparando las actividades de consulta contra todas las actividades contenidas en el repositorio (caso 3). La Figura 28 ilustra los resultados obtenidos para los dominios utilizados en el banco de pruebas.

En la Figura 28(A) se puede observar la relación entra las medidas de precisión y recall para los tres escenarios de prueba. En el caso 1 se tiene una mejor relación entre la medida de precisión y recall, esto se puede atribuir al hecho que se comparan actividades de un mismo dominio, teniendo el sistema una mayor posibilidad de diferenciar entre las actividades relevantes y las que se deben descartar. Por su parte el caso 3 toma el mismo comportamiento que el caso 1, sin embargo su eficacia es disminuida por el ruido que agregan las actividades pertenecientes a otros dominios. El caso 2, presenta un comportamiento particular al ser muy preciso, ya que las actividades recuperadas son todas relevantes para la búsqueda, sin embargo esta precisión implica que se descarten muchas actividades que podrían ser candidatas relevantes en la búsqueda, hecho por el cual el valor de recall es bajo y no crece más allá de un 0,2. El desempeño del caso 2 se presenta como el más pobre de los tres escenarios, ya que se tiene una gran diferencia entre los valores de precisión y recall.

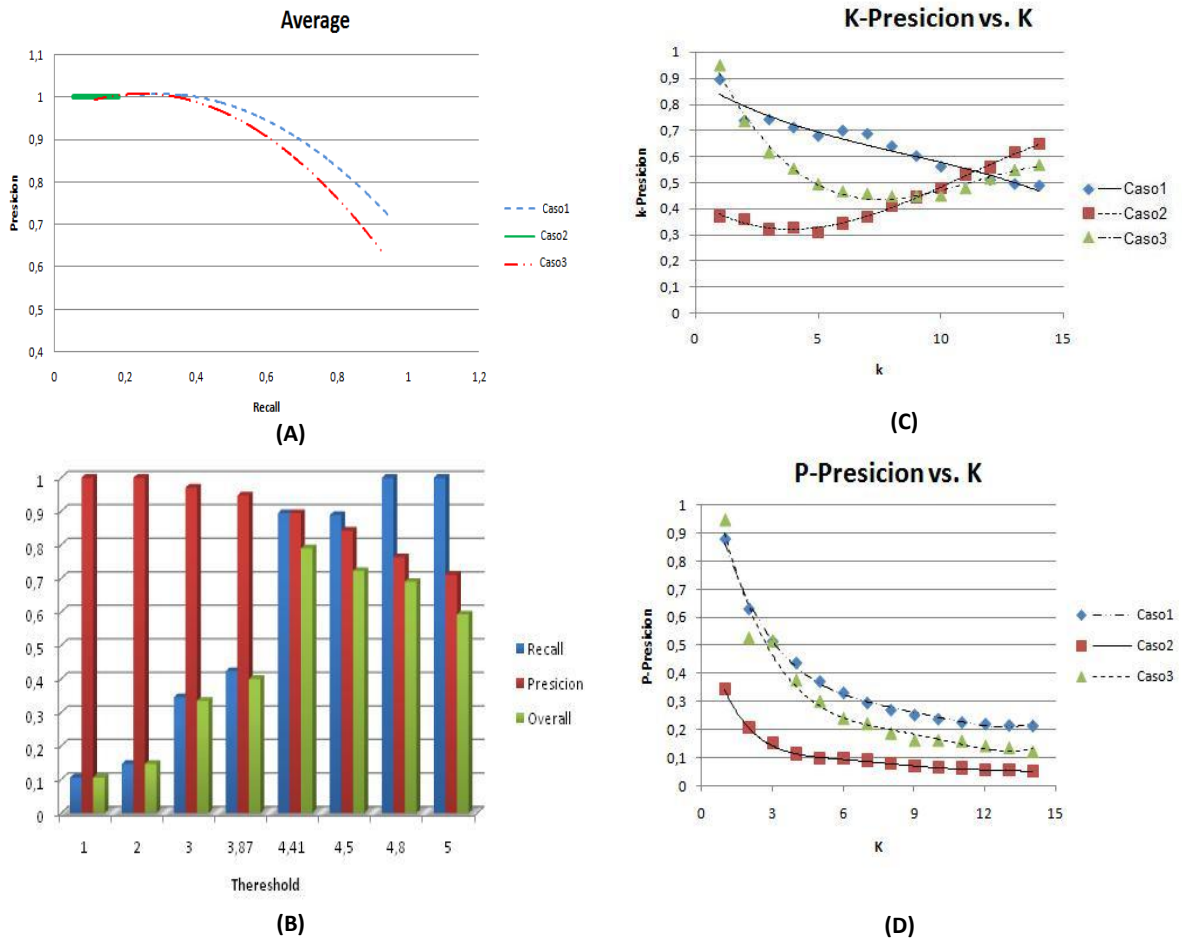

Figura 28. Evaluación de Similitud de Actividades

El desempeño total del sistema se puede apreciar en la Figura 28(B) donde se identifica el umbral de similitud óptimo para el algoritmo de emparejamiento, equivalente a 4,41, punto en el cual la medida total alcanza su tope en un valor cercano al 80%. Igualmente se aprecia que en los valores de similitud superiores a 4,41 el sistema obtiene un mejor desempeño que aquellos ubicados en la parte inferior, siendo más favorable trabajar con valores altos de similitud, donde la medida de recall realiza un mayor aporte al desempeño general. El algoritmo trabaja de una manera muy selectiva, tomando valores de precisión muy altos, y recuperando un alto porcentaje de servicios relevantes cuando el umbral de similitud está por encima de 4.

La medida de precisión con respecto al número de actividades recuperadas por el algoritmo es presentada en la Figura 28(C). Para valores bajos de k (número de actividades recuperadas) se tiene que el caso 1 es más preciso, dado que la búsqueda bajo el mismo dominio de consulta incrementa la posibilidad de encontrar las actividades más similares a una actividad requerida. Por el contrario el caso 3, inicia con valores pequeños de precisión para un k bajo, pero su precisión crece a medida que se aumenta el número de actividades recuperadas, hecho atribuido a la posibilidad de obtener actividades con un valor menor de similitud. El caso 2 presenta el desempeño más pobre, la precisión es considerablemente menor en comparación a los otros casos cuando el número de actividades recuperadas es bajo ($k < 5$). Esto deja en evidencia que el algoritmo de recuperación no opera bien cuando las actividades comparadas no pertenecen a un mismo dominio.

De acuerdo a la medida Total el mejor desempeño del sistema se obtiene para valores de similitud superiores a 4,41, rango en el cual la precisión oscila entre 0,7 y 0,9. Tomando este rango como referencia el k óptimo del sistema se encuentra entre 1 y 7 actividades recuperadas, valores en los que la precisión se ajusta a los parámetros requeridos para el mejor desempeño. Para alcanzar estos valores se debe implementar un filtrado de actividades por dominio que ayude a obtener un comportamiento similar al caso 1 de la evaluación realizada.

La gráfica de P-Precisión muestra de igual manera que el caso 1 acierta con una mayor eficacia la posición y las actividades recuperadas de acuerdo al banco de pruebas generado. Para esta medida las curvas de la Figura 28(D) son decrecientes a medida que se incrementa el número de actividades k, comportamiento presentado en los tres casos. Las mejores presiones de posición se alcanzan para un valor de k inferior a 3, teniendo como valor mínimo de precisión 0,7 rango en el cual se tiene un desempeño total óptimo. En cuanto a los casos 2 y 3, se tiene una medida P-Precisión por debajo del caso 1, propiciado por que la precisión se afecta negativamente cuando se incorporan actividades de diferentes dominios, más aún cuando se debe tener en cuenta las posiciones del ranking entregado por el algoritmo de emparejamiento.

4.4.5.3 Prueba de Rendimiento con Contexto PR2

Las pruebas correspondientes al desempeño del prototipo de emparejamiento de servicios tomando en cuenta el contexto de los dispositivos y usuario se basaron en el trabajo de (Suarez, y otros, 2010). Como se puede observar en la Figura 29, el tiempo de respuesta para el proceso de emparejamiento a nivel atómico considerando el contexto posee el mismo comportamiento y el tiempo de ejecución es similar a la prueba de rendimiento PR1.

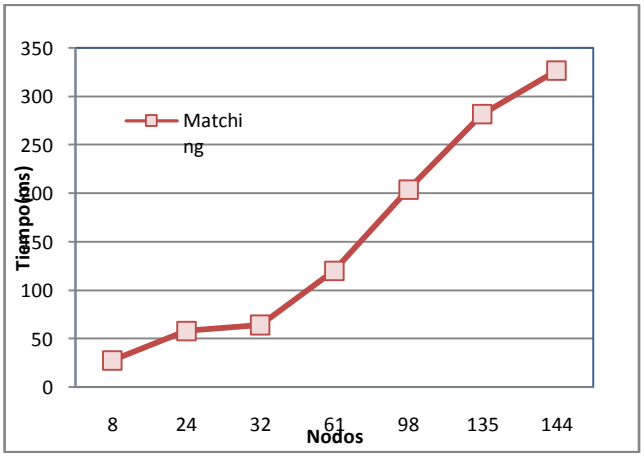

Figura 29. Rendimiento del proceso de emparejamiento de actividades considerando el contexto.

Existe un incremento en el tiempo de respuesta correspondiente a la fracción que toma la identificación del contexto del dispositivo móvil, asociado con los tiempos necesarios para consultar los repositorios y la consideración de parámetros adicionales en el emparejamiento.

Tomando como referencia (Joines, y otros, 2002) se determinó el siguiente comportamiento:

- **Óptimo** (tiempo de respuesta menor o igual a 0,1s.): cuando el número de nodos comparados es menor o igual a 46,1.

- **Bueno** (tiempo de respuesta entre 0,1s. y 1s.): cuando el número de nodos comparados es mayor que 46,1 nodos y menor que 466,4.
- **Aceptable** (tiempo de respuesta entre 1s. y 10s.): cuando el número de nodos comparados es mayor que 466,4 nodos y menor que 4669,8.
- **Deficiente** (tiempo de respuesta mayor a 10s.): cuando el número de nodos comparados es superior a 4669,8.

4.4.5.4 Prueba de Calidad de los Resultados con Contexto PC2

Las pruebas correspondientes a la calidad de recuperación del prototipo de emparejamiento de servicios tomando en cuenta el contexto de los dispositivos y usuario se basaron en el trabajo de (Suarez, y otros, 2010), la Figura 30 muestra los resultados presentados en dicho trabajo.

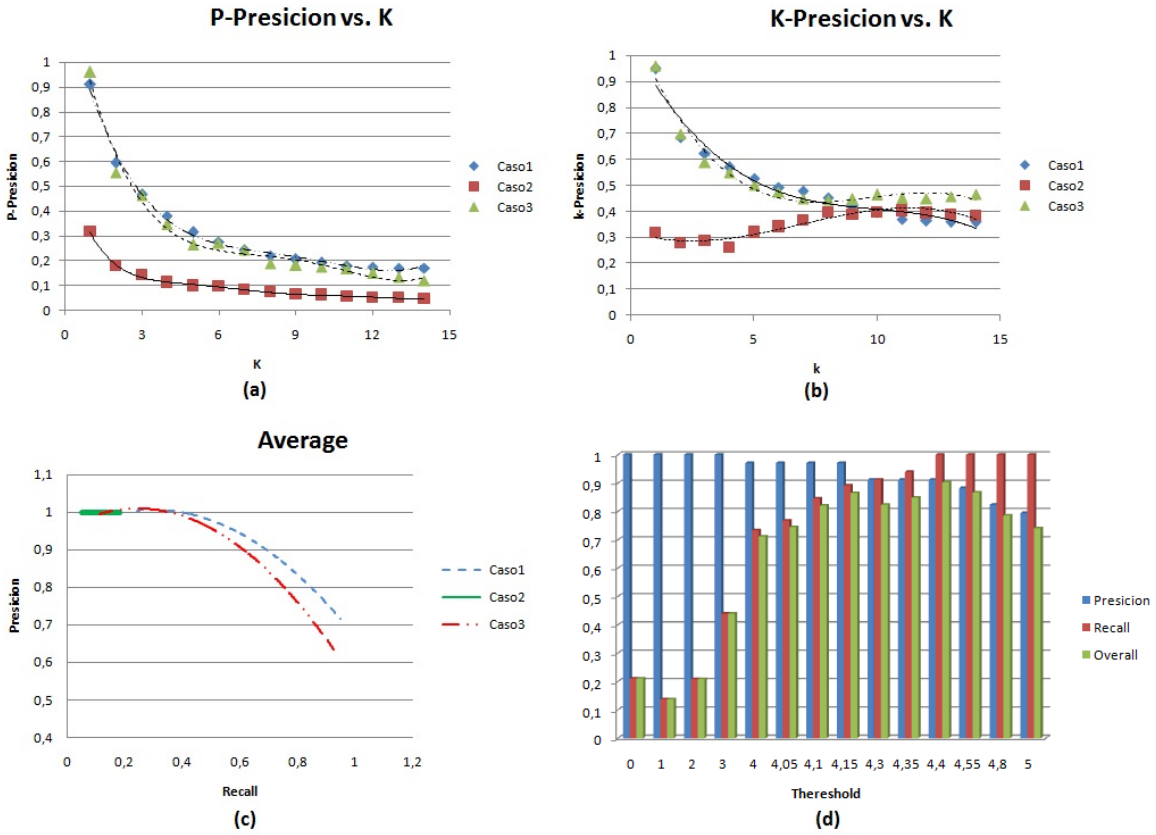

Figura 30. Evaluación de Similitud de Actividades Considerando el Contexto de Entrega

De los resultados obtenidos se puede mencionar que el comportamiento del sistema es muy similar a las medidas obtenidas en el emparejamiento de actividades, sin tener en cuenta el contexto, teniendo como gran ventaja que se mejora sustancialmente el desempeño general de algoritmo de emparejamiento (*overall*), ver Figura 30 (D), obteniendo un desempeño total superior del 0,7 en un rango de similitud mayor a 4,0, siendo más amplio al que se obtuvo en las pruebas anteriores. En este rango de similitudes se tiene un comportamiento superior y más sostenido en términos de desempeño Total del algoritmo, demostrando el impacto favorable de la incorporación de la información del contexto en las funciones de recuperación de servicios.

Capítulo 5

Composición de Servicios

El presente capitulo expone de manera detallada la propuesta para realizar la composición de servicios en ambientes de computación ubicua. En primera instancia se realiza una descripción de la técnica de codificación empleada para identificar de manera única las actividades involucradas en el proceso de composición. Posteriormente, se presenta la forma como se analiza la conformidad de los servicios que se emplearán en la composición, a través de la construcción de un grafo de dependencia de datos que permita establecer la compatibilidad de los servicios a nivel de sus interfaces. Luego, se presentan los algoritmos propuestos para obtener las síntesis de composición, basada en el grafo de dependencia de datos, y la posterior selección de un plan de composición para la tarea de usuario requerida. Finalmente, se muestran los resultados obtenidos al evaluar los algoritmos propuestos.

5.1 Codificación de las Actividades

Para la composición de servicios se realiza una codificación de los nodos que representan las actividades básicas BPEL, consistente en asignar un número primo diferente a cada uno de los nodos. Esta asignación se realiza tanto para los nodos que componen el proceso de consulta como para aquellos que son recuperados en la etapa de descubrimiento. La codificación se lleva a cabo para identificar de una manera ágil las dependencias de datos y ejecución que existen en el proceso de consulta y que deben ser tomadas en cuenta al momento de componer los servicios recuperados.

Para el proceso de consulta se recorre el grafo respectivo y se asigna un número primo único a cada uno de los nodos que representa una actividad básica BPEL, iniciando desde el 3 hasta el número primo que sea necesario de acuerdo al número de actividades básicas contenidas en el proceso. Con este número primo asignado se identifica de manera única a cada actividad de consulta durante la composición de servicios. De igual manera, para las actividades recuperadas se realiza una codificación empleando los números primos desde el 3 hasta el número primo que se requiera de acuerdo a la cantidad de actividades recuperadas.

5.2 Conformidad de los Servicios

La conformidad de los servicios hace referencia al filtrado que se debe realizar para descartar aquellos servicios recuperados por el módulo de descubrimiento que no son útiles para la composición de las tareas del usuario. Específicamente, la conformidad de los servicios compara los datos de entrada y/o salida de las actividades requeridos por el usuario contra las actividades recuperadas para determinar cuáles de estas son compatibles a nivel de interfaz para el proceso de composición (sección 5.2.1), posteriormente se toman en cuenta las restricciones de precedencia entre las operaciones que establece la conversación de consulta (sección 5.3.2), y que se deben cumplir para una adecuada ejecución del proceso compuesto.

5.2.1 Restricciones y Flujo de Datos

Los modelos de comportamiento permiten representar el flujo de datos entre los servicios que participan de un proceso. El flujo de datos entre dos actividades se establece cuando la información de salida de una operación constituye la información de entrada de otra que se ejecuta posteriormente, en tal caso se establece una dependencia de datos entre ellas. Un tipo especial de flujo de datos se presenta cuando se requiere que las operaciones que intercambian datos pertenezcan a un mismo servicio, en este caso se tiene una restricción de datos.

5.2.1.1 Comparación de Datos

Para determinar la distancia entre los datos utilizados en las interfaces de dos operaciones se debe realizar una comparación de los parámetros de entrada y salida que componen dichas interfaces. Las comparaciones se realizan teniendo en cuenta el tipo de datos y el nombre asignado a los parámetros que pertenecen a la interfaz.

Los procesos de negocio definen variables, las cuales contienen los datos de entrada y/o salida de las operaciones invocadas, estos datos son los mismos que se intercambian entre actividades. Estas variables se pueden definir de la siguiente manera:

Definición 2: Variable, una Variable V es definida como una dupla (Param, FuncType) donde:

- **Param** es el conjunto de los nombres de los datos
- **FuncType**: Param -> Tipos Datos; es una función que asigna a cada parámetro un tipo de dato. Para un proceso de negocio BPEL Tipos Datos es el conjunto de tipos de datos XML.

Partiendo de la definición anterior se puede establecer la similitud de dos actividades a nivel de sus interfaces:

Definición 3: Similitud de mensajes, dos actividades $queryActivity_{ik}$ = ($Operation_{ik}$, $PortType_{ik}$, $Type_{ik}$, $Input_{ik}$, $Output_{ik}$) y $targetActivity_{jl}$ = ($Operation_{jl}$, $PortType_{jl}$, $Type_{jl}$, $Input_{jl}$, $Output_{jl}$) poseen mensajes similares si:

1. $\forall\, p \in Param\ in\ Input_{ik}, \exists\, p' \in Param\ in\ Input_{jl} \mid p\ sea\ similar\ a\ p'\ y\ FuncType(p) = FuncType(p')$

2. $\forall\, p \in Param\ in\ Output_{ik}, \exists\, p' \in Param\ in\ Output_{jl} \mid p\ sea\ similar\ a\ p'\ y\ FuncType(p) = FuncType(p')$

La **Definición 3** establece la forma de determinar si dos actividades son similares a nivel de sus interfaces, realizando un emparejamiento de los datos de entrada y salida. La similitud de los nombres y tipos asignados a los datos se mide de acuerdo al tipo comparación que se realice (sintáctica o semántica). El Algoritmo 8 muestra la manera de realizar una comparación lingüística del nombre de los parámetros y una comparación sintáctica del tipo de dato, el algoritmo entrega como resultado una medida de la similitud de los datos de entrada (línea 9). La similitud de los datos es calculada empleando la función *LS* (línea 8) descrita en el Algoritmo 3.

Algoritmo 8. Función Emparejamiento de Datos - DataMatch

```
1.  INPUTS: (Datai, Dataj) Datai: Struct (Namei, Typei), Dataj: Struct (Namej, Typej)
2.  OUTPUT: DistanceData

3.  BEGIN
4.  DataDistance = 0

5.  If Typei ≠ Typej then
6.      return 1
7.  else
8.      DataDistance = LS(Namei, Namej)
9.      return DataDistance
10. end if
```

El emparejamiento de datos descrito en el Algoritmo 8 se debe realizar para cada uno de los parámetros que componen los mensajes de entrada y/o salida de las actividades que se desean comparar. El Algoritmo 9 realiza el emparejamiento de los mensajes, este procedimiento se aplica para determinar que tan similares son los mensajes de entrada y/o salida de dos actividades. El Algoritmo 9 entrega como resultado un conjunto de duplas (línea 15), las cuales contienen los datos emparejados y la distancia calculada por la función *DataMatch* (línea 8).

Algoritmo 9. Función Emparejamiento de Mensajes - MessageMatch

```
1.  INPUTS: (Nodei, Nodej); Nodei: Struct (Messagei), Nodej: Struct (Messagej)
2.  OUTPUT: Matches[][]

3.  BEGIN
4.  Matches[]= ∅

5.  for each parami ∈ Param in Messagei do
6.      BestMatch = 1
7.      for each paramj ∈  Param in Messagej do
8.          DistanceData =
              DataMatch([parami, FuncType(parami)], [paramj, FuncType(paramj)])
9.          if DistanceData < BestMatch then
10.             BestMatch = DataDistance
11.             Matches[parami] =( paramj, DistanceData)
12.         end if
13.     end for
14. end for
15. return Matches[][]
```

5.2.1.2 Grafo de Dependencia de Datos

Tomando como punto de partida la comparación de datos se construye un grafo de dependencia de datos con el objetivo de identificar los servicios que pueden combinarse para componer la

tarea de usuario solicitada. El grafo de dependencia es construido empleando el concepto de Híper-grafos (Gallo, y otros, 1993). A continuación se definen formalmente los Híper-grafos.

Definición 4: Híper-grafo, es una dupla H = (V, E), donde V= v_1, v_2, v_3,…,v_n es el conjunto de vértices o nodos, y E= E_1, E_2, E_3,…,E_m con $E_i \subseteq V$ para *i*= 1,2,3,…,*m* es el conjunto de híper-aristas. Cuando $|E_i| = 2$, *i*= 1,2,…, *m* el híper-grafo es una grafo convencional.

Definición 5: Híper-grafo dirigido, es un híper-grafo con híper-aristas dirigidas. Una híper-arista dirigida es una pareja ordenada E= (X, Y), donde X es la cola de E, denotada por T(E), mientras Y representa la cabeza H(E).

Definición 6: BF-Híper-grafo, un BF-híper-grafo, o simplemente BF-grafo, es un híper-grafo cuyas híper-aristas son del tipo B-arista o F-arista. Una B-arista (*backward*) es una híper-arista E= ((T(E),H(E)) con |H(E)|= 1). Una F-arista (*forward*) es una híper-arista E= ((T(E),H(E)) con |T(E)|= 1).

Para la construcción del grafo de dependencias se emplean los nodos que representan las actividades recuperadas y se crean nuevos nodos para representar los datos que conforman los mensajes de entrada y/o salidas de dichas actividades. Por tanto, las híper-aristas del grafo de dependencias poseen los siguientes tipos:

- E_{pd}= ({p}, O). F-arista desde el nodo *p* (tipo actividad) a un conjunto de nodos *O* que contiene los datos de salida de *p*.
- E_{dp}= (I, {p}). B-arista desde el conjunto de datos *I* al nodo *p* (tipo actividad), *I* representa el conjunto de datos de entrada de *p*.
- E_{dd}= ({d}, D). F-arista desde el nodo *d* (tipo dato) al conjunto de nodos *D* (tipo datos), este tipo de híper-arista se utiliza para representar la asignación del valor de un dato a otro(s).

En la siguiente sección se explica la forma como se construye el híper-grafo de dependencias a partir del grafo de consulta y los nodos recuperados, posteriormente se detalla la manera de analizar el híper-grafo para obtener las posibles síntesis de composición.

5.2.1.3 Construcción del Grafo de Dependencia de Datos

La construcción del híper-grafo de dependencias es guiada por el grafo de consulta, se realiza un recorrido ordenado de los nodos de consulta y se van agregando los nodos que representan los datos de entrada y/o salida de los nodos de consulta así como los nodos recuperados. A medida que se agregan los diferentes nodos se van creando las relaciones de dependencia a través de las híper-aristas que conectan los vértices. Los procedimientos para la construcción del híper-grafo se describen a continuación.

Procedimiento 1: Recorrer el Grafo de Consulta, la construcción del híper-grafo se realiza siguiendo el orden establecido en el grafo de consulta, se hace de esta forma para establecer las relaciones de dependencia de datos y ejecución entre las actividades básicas. Para recorrer el grafo se utiliza el algoritmo de búsqueda en profundidad (DFS *Depth First Search*), el cual permite recorrer un grafo de manera ordenada pero no uniforme. La falta de uniformidad constituye un problema para crear las dependencias de datos, para solucionar esto se agrega una variación al algoritmo para realizar un recorrido cuasi-uniforme de los grafos cuando hay presencia de estructuras complejas (AND-Join y XOR-Join). La variación implementada consiste recorrer todas las ramas derivadas de una estructura compleja antes de continuar la búsqueda en profundidad,

de esta manera las actividades básicas se agregan de acuerdo a un orden de ejecución, asegurando que cuando un nodo tipo actividad sea agregado al híper-grafo, previamente se hayan adicionado aquellas actividades que generan los datos de entrada para dicho nodo.

El Algoritmo 10 presenta la función para realizar el recorrido DFS Modificado para un grafo de consulta. El recorrido inicia por los nodos tipo *Start* contenidos en el grafo de consulta (línea 6), se agrega el nodo inicial al híper-grafo (línea 11) y se colocan en una pila (línea 20) los vecinos del nodo que se está procesando y se les asigna un estado visitado (línea 15), para asegurar que el nodo sea procesado una sola vez. Este procedimiento se repite por cada nodo que haya sido agregado a la pila (línea 9), cada vez que se procesa un nodo se retira de la pila (línea 10). De manera alterna se construye una cola (*QueueConnector*) para almacenar temporalmente los nodos tipo conector *AND-Join* y *XOR-Join* (línea 18), esto se realiza para controlar la profundidad del recorrido, al encolar estos conectores en un arreglo diferente se logra que se recorran todas las ramas de una estructura compleja antes de aumentar la profundidad mas allá de los nodos donde convergen dichas ramas. Con esto se logra que al visitar un nodo determinado se hayan visitado todas aquellas actividades predecesoras que pertenecen a ramas concurrentes o excluyentes.

Algoritmo 10. Función DFS Modificado – DFS Modified

```
1.  INPUTS: (queryGraph)
2.  OUTPUT: ∅

3.  BEGIN
4.  StackNextNode[] = ∅
5.  QueueConnector[]= ∅

6.  for each startNode ∈ StartNodes in queryGraph do
7.      state[startNode] = visited
8.      StackNextNode add startNode
9.      while StackNextNode ≠ ∅ do
10.         node = StackNextNode remove first Node
11.         AddQueryNode (node)
12.         Neighbors[] = queryGraph->getNeighbors(node)
13.         for each neighbor in Neighbors do
14.             if state[neighbor] ≠ visited then
15.                 state[neighbor] = visited
16.                 parent[neighbor]= node
17.                 if type[neighbor]== AND-Join or type[neighbor]== XOR-Join then
18.                     QueueConnector add neighbor
19.                 else
20.                     StackNextNode add neighbor
21.                 end if
22.             end if
23.         end for
24.         if StackNextNode == ∅ and QueueConnector ≠ ∅ then
25.             StackNextNode add QueueConnector remove last Node
26.         end if
27.     end while
28. end for
```

A continuación se detalla el procedimiento para adicionar un nodo de consulta al híper-grafo de composición.

Procedimiento 2: Agregar Nodo de Consulta, el Algoritmo 12 describe el procedimiento para adicionar un nodo de consulta al híper-grafo de composición. Este procedimiento consiste básicamente en detectar que tipo de nodo se quiere agregar (*Start, BasicActivity, AssignNode, AND, XOR, End*) y de acuerdo a esto definir los pasos que se deben seguir. En este caso, se profundizará en el tratamiento requerido para adicionar un nodo tipo *BasicActivity* (línea 11) y *Assign* (línea 14), debido a que estos nodos determinan el flujo y restricciones de datos en el proceso BPEL. Los demás nodos reciben un tratamiento básico, simplemente se agregan al híper-grafo y se crean las híper-aristas para conectarlos con el nodo padre, el cual ha sido identificado durante el recorrido DFS Modified del grafo de consulta (Algoritmo 10 línea 16).

Algoritmo 11. Función Adicionar Nodo de Consulta –AddQueryNode

```
1.  INPUTS: (Node_q); Node_q: Struct (Op_q, PT_q, Type_q, Input_q, Output_q)
2.  OUTPUT: Ø

3.  BEGIN
4.  if Node_q ∈ Start then
5.      Hypergraph add Start Node_q
6.  end if

7.  if Node_q ∈ Connector then
8.      Hypergraph add Connector Node_q
9.  end if

10. if Node_q ∈ BasicActivity then
11.     AddActivityNode(Node_q)
12. end if

13. if Node_q ∈ Assign then
14.     AddAssignNode(Node_q)
15. end if

16. if Node_q ∈ End then
17.     Hypergraph add End Node_q
18. end if
```

En el Procedimiento 3 se describe el algoritmo para adicionar una actividad básica y el Procedimiento 4 explica el tratamiento que se realiza para los nodos que realizan la asignación de valores entre las variables de un proceso de negocio.

Procedimiento 3: Agregar Nodo Actividad Básica, este procedimiento consiste en agregar los nodos tipo dato que representan las entradas y/o salidas de la actividad básica de consulta, y sumar los nodos recuperados al híper-grafo creando las relaciones de dependencia entre los datos y las actividades predecesoras. El Algoritmo 12 especifica cada uno de los pasos que se siguen para adicionar un nodo de consulta al híper-grafo.

Algoritmo 12. Función Adicionar Actividad de Consulta –AddActivityNode

1. **INPUTS**: ($Node_q$); $Node_q$: Struct (Op_q, PT_q, PL_q, $Type_q$, $Input_q$, $Output_q$)
2. **OUTPUT**: ∅

3. **BEGIN**
4. *IndexQuery[$Node_q$] = get Next Prime Number for Query Nodes*
5. *TargetSelected[[IndexQuery[$Node_q$]] = 1*
6. **if** *PL_q ∈ QueryServices[]*
7. *QueryServices[PL_q]= QueryServices[PL_q] * IndexQuery[$Node_q$]*
8. **else**
9. *QueryServices[PL_q]= IndexQuery[$Node_q$]*
10. **end if**

11. **for each** *matchNode* **in** *ListRankedServices* **do**
12. *TargetNode = matchNode get TargetNode*
13. *IndexTarget[targetNode] = get Next Prime Number for Target Nodes*
14. *TargetSelected[IndexQuery[$Node_q$]] =*
    *TargetSelected[[IndexQuery[$Node_q$]] * IndexTarget[targetNode]*
15. *OutputDistance = AddOutputData($Node_q$, TargetNode)*
16. *InputDistance = AddInputData($Node_q$, TargetNode)*
17. *NodeDistance[TargetNode] = matchNode[Distance] + InputDistance + OutputDistance*
18. *Hypergraph-> AddNodeTarget(matchNode, DataMatch)*
19. **end for**

El Algoritmo 12 empieza asignando un código al nodo de consulta (línea 4), el código corresponde a un número primo único para identificar al nodo, el código se va incrementando a medida que se agregan nuevos nodos. Posteriormente, se recorren las parejas entregadas por el módulo de descubrimiento (línea 6) para agregar los nodos recuperados al híper-grafo. A cada nodo recuperado se le asigna un número primo único para identificarlo (línea 8) y se procede a adicionarlo al híper-grafo (línea 18). Cuando se procesan los nodos recuperados se lleva a cabo la comparación de las interfaces de entrada y salida con el nodo de consulta (línea 10 y 11), con esto se crean las aristas de dependencias entre los datos y las actividades y se establece la compatibilidad entre las actividades candidatas a participar en la composición. Se pude observar que en el algoritmo se construye un índice denominado como *TargetSelected* con el propósito de relacionar a los nodos de consulta con los nodos recuperados, para ello se obtiene el producto de todos los índices asignados a los nodos recuperados y se emparenta con el índice del nodo de consulta correspondiente (línea 9).

La función *AddOutputData* invocada en la línea 10 del Algoritmo 12 se describe en el Algoritmo 13. Esta función tiene como objetivo determinar que tan similares son los datos de salida de los nodos de consulta ($Node_q$) y los nodos recuperados ($Node_t$). Como primer paso se realiza un emparejamiento en los datos de salida de los dos nodos (línea 4) empleando la función *MessageMactch* (Algoritmo 9), con esta función se emparejan los datos y se obtienen una medida de su similitud. Acto seguido, se recorren las parejas encontradas para crear la relación entre el nodo recuperado y los datos de salida del nodo de consulta, esta relación se representa como una híper-arista (F-arista línea 16) la cual tiene un peso equivalente a la similitud estimada por el algoritmo de emparejamiento de mensajes (línea 17) y un atributo denominado *MatchData* que indica con que dato del nodo recuperado fue emparejado el dato de consulta. Antes de crear la

relación se verifica si el dato de consulta ya ha sido agregado al híper-grafo (línea 9), si no es ese el caso se crea un nuevo nodo del tipo Dato y se lo adiciona al híper-grafo (línea 10). Al nodo tipo dato se le asocia un índice nombrado como *TargetDataTail* el cual permite saber cuáles nodos recuperados generan este dato dentro de sus interfaces de salida. Esta información es importante para determinar la compatibilidad entre operaciones que intercambia datos. El índice *TargetDataTail* es calculado como el producto de los índices de todos aquellos nodos tipo actividad que producen el dato entre sus salidas. Finalmente, el algoritmo entrega una medida total de la distancia entre los datos de salida de los dos nodos comparados, esta distancia corresponde a una suma de las distancias individuales de los datos de salida.

Algoritmo 13. Función Adicionar Datos de Salida – AddOutputData

1. **INPUTS**: ($Node_q$, $Node_t$); $Node_q$: Struct (Op_q, PT_q, $Type_q$, $Input_q$, $Output_q$), $Node_t$: Struct (Op_t, PT_t, $Type_t$, $Input_t$, $Output_t$)
2. **OUTPUT**: DistanceData
3. **BEGIN**
4. *DataMatch[][] = MessageMatch($Output_q$, $Output_t$)*
5. **for each** *dataMatch* **in** *DataMatch[][]* **do**
6. *QueryData = dataMatch get query data*
7. *TargetData = dataMatch get target data*
8. *DataDistance = dataMatch get data distance*
9. **if** *QueryData* $\notin$ *Hypergraph* **then**
10. *H-NodeDQ = Hypergraph add new Hyper Node for Query Data*
11. *TargetDataTail [H-NodeDQ] = 1*
12. **else**
13. *H-NodeDQ = Hypergraph get Hyper Node for Query Data*
14. **end if**
15. *QueryDataTail[H-NodeDQ] = IndexQuery[$Node_q$]*
16. *F-edge = Hypergraph add new F-edge E_{pd} = ({$Node_t$}, O ∪ {H-NodeDQ})*
17. *Cost[F-edge] = DataDistance*
18. *MatchData[F-edge] = TargetData*
19.
20. **if** *DataDistance < 1* **then**
21. *TargetDataTail[H-NodeDQ] = TargetDataTail[H-NodeDQ] * IndexTarget[$Node_t$]*
22. **end if**
23. *DistanceTotal = DistanceTotal+ DataDistance*
24. **end for**
25. **return** *DistanceTotal*

Igualmente, se deben procesar los datos de entrada del nodo de consulta para establecer la similitud de las interfaces de entrada con las actividades recuperadas en la fase de descubrimiento, el Algoritmo 14 define la secuencia de pasos para este propósito. Se reciben los nodos (*$Node_q$ y $Node_t$*) que se desean comparar a nivel de sus entradas y se entrega como resultado la medida de similitud entre los datos.

El Algoritmo 14 empieza realizando un emparejamiento en los datos de entrada de los dos nodos (línea 5), empleando la función *MessageMactch* (Algoritmo 9), con la cual se emparejan los datos y se obtiene una medida de su similitud. Terminado el emparejamiento se verifica si todas las entradas del nodo recuperado ($Node_t$) tienen un par correspondiente en los datos del nodo de consulta (línea 7), si esta condición no se cumple se entiende que al menos uno de los datos necesarios para la invocación del nodo recuperado no estará presente en el proceso compuesto, por lo que esta operación no podrá ser ejecutada (línea 8) y se le asigna una distancia equivalente a ∞ (línea 9) indicando que este nodo debe ser ignorado al momento de realizar la composición de servicios.

Si todas las entradas del nodo recuperado fueron emparejadas se recorren las parejas encontradas para crear la relación entre el nodo recuperado y los datos de entrada del nodo de consulta. Si se encuentra una pareja de datos con una distancia de similitud igual a 1 (línea 15) se concluye que la actividad recuperada no tiene como entrada uno de los datos requeridos en la operación de consulta, bajo esta condición se declara al nodo como no ejecutable bajo las restricciones expresadas en el proceso de negocio y se lo descarta para una posible composición (línea 37).

Si se cumplen todas las condiciones de compatibilidad entre los datos de los dos nodos comparados se crea una híper-arista (B-arista línea 22) para representar la relación entre los datos de entrada y el nodo recuperado, se le asigna un peso a la híper-arista equivalente a la similitud estimada por el algoritmo de emparejamiento de mensajes (línea 23) y un atributo denominado *MatchData* (línea 24) que indica con que dato del nodo recuperado fue emparejado el dato de consulta.

Algoritmo 14. Función Adicionar Datos de Entrada – AddInputData

```
1.  INPUTS: (Node_q, Node_t); Node_q: Struct (Op_q, PT_q, Type_q, Input_q, Output_q),
    Node_t: Struct (Op_t, PT_t, Type_t, Input_t, Output_t)
2.  OUTPUT: DistanceData

3.  BEGIN
4.  DistanceTotal = 0
5.  DataMatch[][] = MessageMatch(Input_q, Input_t)
6.  DataFlow[][] = ∅

7.  if ∃input_t ∉ DataMatch[][] then
8.      state[Node_t]= Not Executable
9.      return ∞
10. end if

11. for each dataMatch in DataMatch[][] do
12.     QueryData = dataMatch get query data
13.     TargetData = dataMatch get target data
14.     DataDistance = dataMatch get data distance

15.     if DataDistance < 1 then
16.         if QueryData ∉ Hypergraph then
17.             state[Node_t]= Not Executable
```

```
18.             return ∞
19.         else
20.             T-NodeDQ = Hypergraph get Hyper Node for Query Data
21.         end if
22.         B-edge = Hypergraph add new B-edge E_dp = (I ∪ {T-NodeDQ}, {Node_t})
23.         Cost[B-edge] = DataDistance
24.         MatchData[B-edge] = TargetData
25.         DistanceTotal = DistanceTotal+ DataDistance
26.         queryTail = QueryDataTail[T-NodeDQ]

27.         if queryTail ∈ DataFlow[][] then
28.             lastMCD = DataFlow[queryTail]
29.             mcd = MCD(lastMCD, TargetDataTail[T-NodeDQ] )
30.         else
31.             mcd = TargetDataTail[T-NodeDQ]
32.         end if

33.         DataFlow[queryTail] = mcd
34.         targetSelected = TargetSelected[queryTail]
35.         XOR4Composed[Node_t]  = targetSelected / mcd
36.     else
37.         state[Node_t]= Not Executable
38.         return ∞
39.     end if
40. end for

41. return DistanceTotal
```

El algoritmo crea un conjunto de dependencias nombrado como *DataFlow*, ahí se lleva una memoria para identificar cuáles datos de entrada deben ser producidos por una misma actividad predecesora. Esto es importante para conocer qué actividades agregadas previamente generan datos compatibles con las entradas requeridas por el nodo recuperado que se está procesando, permitiendo deducir cuáles pueden ser combinadas con dicho nodo y cuáles no. Para esto se inicia averiguando el índice del nodo de consulta *QNP* (Nodo de Consulta Predecesor) que genera el dato de entrada (*queryTail* línea 26), este índice se establece cuando se agregan los datos de salida del nodo de consulta, luego se verifica si ese nodo predecesor *QNP* ya fue incluido en el flujo de datos (línea 27), si es así, quiere decir que existen otros datos de entrada que son generados por ese nodo *QNP*. Ahora se debe identificar cuáles nodos recuperados para el nodo *QNP* producen todos los datos requeridos como entrada para el nodo $Node_t$, esto se obtiene calculando el máximo común divisor (*MCD*) de los índices *TargetDataTail* (Algoritmo 13 línea 21) asignados a los nodos data. El siguiente ejemplo aclara el funcionamiento de esta lógica.

Ejemplo: Considere un escenario de flujo de datos como el que se plantea en la Figura 31, en el grafo de consulta se define un intercambio de datos (*departure* y *return*) entre dos actividades *ClientRequest* y *AirlineReservation*. En el lado derecho de la gráfica se representa la conformación del híper-grafo con los nodos recuperados y los nodos tipo dato. En primera instancia se puede observar que a cada nodo se le ha agregado un atributo índice, para los nodos de consulta se

utiliza una serie de números primos y para los nodos recuperados otra. El ejemplo toma una fracción del proceso de negocio expuesto en la sección 4.1.

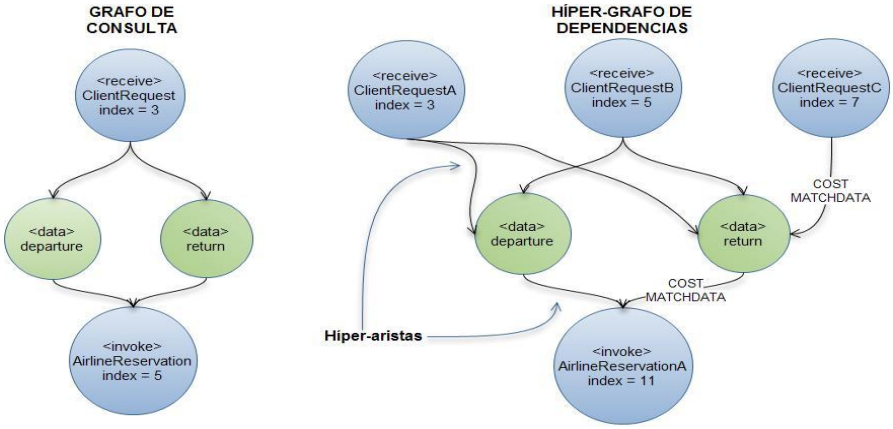

Figura 31. Ejemplo Flujo de Datos

La Tabla 8 contiene los valores de índices para los nodos tipo actividad básica contemplados en el ejemplo. El valor obtenido para el índice *TargetSelected* se asocia directamente con el nodo de consulta correspondiente, y es equivalente al producto de los índices asignados a los nodos recuperados para dicho nodo de consulta.

Tabla 8. Valores de Índices para las Actividades Básicas

Índice Nodos de Consulta	Índice Nodos Recuperados	Índice *TargetSelected*
ClientRequest = 3	*ClientRequestA = 3* *ClientRequestB = 5* *ClientRequestC = 7*	*(ClientRequestA * ClientRequestB * ClientRequestC) = 3*5*7 = 105*
AirlineReservation = 5	*AirlineReservationA= 11*	

Cuando se agregan los nodos recuperados para la actividad de consulta *ClientRequest* (*QNP*) se realiza la comparación de las interfaces de entrada (Algoritmo 14). Se contrastan las salidas de la actividad *ClientRequest* con las salidas de *ClientRequestA, ClientRequestB y ClientRequestC*; cuando se encuentra una salida equivalente en las actividades recuperadas se agrega una híper-arista entre la actividad recuperada y el dato de consulta, ver Figura 31. Al ejecutar el Algoritmo 14 se generan los índices contenidos en la Tabla 9, estos valores son asignados a los datos de salida de la actividad *ClientRequest* al momento de agregarlos al híper-grafo.

Tabla 9. Valores de Índices para las Actividades Básicas

Nodo Dato	*QueryDataTail*	*TargetDataTail*
departure	*3 (ClientRequest)*	*(ClientRequestA * ClientRequestB) = 3*5 = 15*
return	*3 (ClientRequest)*	*(ClientRequestA * ClientRequestB * ClientRequestC) = 3*5*7 = 105*

Para los datos se tiene que el índice *QueryDataTail* es igual a 3, correspondiente al índice asignado a la actividad de consulta (*QNP*) que los genera en el proceso de consulta. El índice *TargetDataTail* para el dato *departure* es 15, producto de la multiplicación de los índices de las actividades recuperadas que generan dicho dato (*ClientRequestA* y *ClientRequestB*), mientras para el dato *return* el *TargetDataTail* es 35 *(ClientRequestB * ClientRequestC)*.

Al momento de agregar la segunda actividad del ejemplo (*AirlineReservation*) se utilizan los índices generados previamente, al agregar las actividades recuperadas y el emparejamiento de datos. Como primera medida se agregan los nodos recuperados para la actividad de consulta *AirlineReservation* y se establece la similitud con los datos de entrada establecidos en el flujo de consulta; para el ejemplo constituye la relación de los datos *departure* y *return* con la actividad *AirlineReservationA*. Una vez creadas las híper-aristas entre los datos y el nodo recuperado, ver Figura 31, se procede a determinar la compatibilidad de la actividad *AirlineReservationA* con las actividades que generan sus datos de entrada, en este caso *ClientRequestA, ClientRequestB* y *ClientRequestC.*

En primer lugar se identifican que datos proceden de una misma actividad predecesora, esto se hace comparando los índices *QueryDataTail* asignado a los datos. En el ejemplo propuesto los datos *departure* y *return* poseen el mismo valor de *QueryDataTail* que indican que proceden del nodo de consulta *ClientRequest*. Para efectos de la composición de servicios se debe averiguar cuáles de las actividades recuperadas producen estos dos datos, esto se realiza obteniendo el máximo común divisor (MCD) de los índices *TargetDataTail* (Algoritmo 14 línea 29) asignados a los datos cuando se relacionaron como salidas de los nodos recuperados en el híper-grafo. Para el ejemplo se tendría lo siguiente:

- *TargetDataTail[departure]=15*
- *TargetDataTail[return]=105*
- MCD(*TargetDataTail[departure], TargetDataTail[return]) = 15*

El MCD calculado representa el producto de los índices de los nodos recuperados que generan las entradas requeridas por el nodo que se está procesando. Para el ejemplo, se tiene que las actividades *ClientRequestA* y *ClientRequestB* producen las dos entradas que requiere la actividad *AirlineReservation*, por tal motivo son compatibles a nivel de sus interfaces, por el contrario la actividad *ClientRequestC* es descartada ya que no genera uno de los datos necesitados para invocar la actividad procesada. Con estos datos se puede calcular matemáticamente que actividades predecesoras no son compatibles a nivel de interfaces, para esto se debe obtener el conjunto de actividades recuperadas predecesoras (para el ejemplo *ClientRequestA, ClientRequestB* y *ClientRequestC*), en el Algoritmo 14 se hace consultando el índice *TargetSelected* de la actividad *QNP*. Seguido a esto se procede a dividir el *TargetSelected* con el MCD calculado para los índices *TargetDataTail* de los datos, el resultado de esta división representa los índices de los nodos que no son compatibles a nivel de datos con el nodo que se está evaluando. De acuerdo a esto se tiene:

- *TargetSelected[ClientRequest]=105*
- MCD(*TargetDataTail[departure], TargetDataTail[return]) = 15*
- *TargetSelected[ClientRequest] / MCD= 105/15 = 7*

En el ejemplo se tiene que la actividad recuperada con índice 7 es incompatible con la operación *AirlineReservation*, tal como se había dicho anteriormente. En el Algoritmo 14 estos índices de incompatibilidad son almacenados en un registro (*XOR4Composed [Node$_t$]* línea 35) con el fin de determinar posteriormente durante la síntesis que servicios pueden ser combinados y cuáles no.

En resumen el procedimiento para agregar una actividad básica crea una estructura en el híper-grafo en la cual se relacionan las actividades recuperadas y los datos especificados en el proceso de consulta, con esto se determina en primera instancia que actividades recuperadas producen o consumen los datos requeridos, y cuales actividades se pueden combinar de acuerdo a sus interfaces para llevar a cabo la composición de servicios.

Procedimiento 4: Agregar Nodo de Asignación de Datos, el procedimiento se encarga de manejar los nodos de asignación (*Assign*) de datos entre variables de un proceso de negocio y consiste en crear una híper-arista entre los datos que se relacionan (Algoritmo 15 línea 14), la dirección de la arista va orientada desde el dato de origen hacia el dato de destino. Al relacionar los datos se copian los índices *QueryDataTail* y *TargetDataTail* (Algoritmo 15 línea 15 y 16) del dato de origen al dato de destino, esto se hace para representar las actividades básicas de donde proceden originalmente los datos.

Algoritmo 15. Función Adicionar Relación de Datos – AddAssignNode

1. **INPUTS**: (Assign$_i$); Assign$_i$: Struct (Data$_{from}$, Data$_{to}$)
2. **OUTPUT**: ∅

3. **BEGIN**
4. **If** *Data$_{from}$* ∈ *Hypergraph* **then**
5. *T-NodeD = Hypergraph get Data Node for Data$_{from}$*
6. **else**
7. *T-NodeD = Hypergraph add new Data Node for Data$_{from}$*
8. **end if**

9. **If** *Data$_{to}$* ∈ *Hypergraph* **then**
10. *H-NodeD = Hypergraph get Data Node for Data$_{to}$*
11. **else**
12. *H-NodeD = Hypergraph add new Data Node for Data$_{to}$*
13. **end if**

14. *F-edge = Hypergraph add new F-edge E_{dd} = ({T-NodeD}, D ∪ {H-NodeD})*
15. *QueryDataTail[H-NodeD] = QueryDataTail[T-NodeD]*
16. *TargetDataTail[H-NodeD] = TargetDataTail[T-NodeD]*

5.2.2 Conformación de los Servicios

En el apartado anterior se abordo la creación del híper-grafo de dependencias entre actividades básicas, generado a partir de la compatibilidad entre las interfaces de entrada y salida de las operaciones recuperadas. En esta sección se complementan los pasos necesarios para la generación del híper-grafo atacando las relaciones entre las actividades básicas, se detallan la manera como se manejan las restricciones de datos y las restricciones de servicios y su impacto en la selección de servicios para la composición de una tarea de usuario.

Para desarrollar esta sección se parte del ejemplo planteado en la Figura 32, se puede observar que las tres operaciones invocadas en las actividades básicas (*AirlineReservation, AirlineCallback y CancelFlight*) pertenecen a un mismo servicio (*AirlineService*), esto constituye una restricción de servicio. Para considerar un servicio recuperado como apto para la composición se debe cumplir que dicho servicio posea todas y cada una de las operaciones requeridas en el proceso de negocio de consulta. Por otro lado, se puede establecer una restricción de datos entre las actividades *AirlineCallback* y *CancelFlight*, si en un momento determinado del proceso de negocio se tiene que ejecutar la actividad *CancelFlight* se debe asegurar que el dato de entrada *reservation* provenga de la actividad *AirlineCallback* y que estas dos actividades pertenezcan al mismo servicio, cumpliendo esta restricción se logra un comportamiento adecuado en el proceso de negocio compuesto.

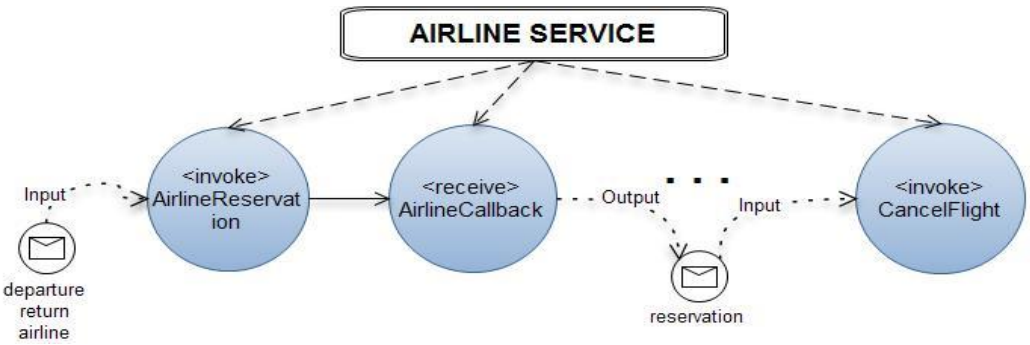

Figura 32. Ejemplo de Restricciones de Servicio y Datos

A continuación se describen los procedimientos para procesar las restricciones de servicios y datos.

Procedimiento 5: Restricciones de Servicios, el propósito de este procedimiento es identificar las actividades básicas recuperadas que pertenecen a un mismo servicio y agruparlas, con esto se puede verificar durante la síntesis de composición si un servicio recuperado posee todas las operaciones que se requieren. El Algoritmo 16 describe la secuencia de pasos para agregar una actividad básica recuperada al híper-grafo e identificar las operaciones que pertenecen a un mismo servicio.

Algoritmo 16. Función Adicionar Nodo Recuperado– AddTargetNode

1. **INPUTS:** ($Node_q$, $Node_t$); $Node_q$: Struct (Op_q, PT_q, $Type_q$, $Input_q$, $Output_q$), $Node_t$: Struct (Op_t, PT_t, PL_t)
2. **OUTPUT**: Ø

3. **BEGIN**
4. *TargetServices[]*
5. *QueryMatches[]*
6. *Hypergaph add new node* $Node_t$
7. **If** PL_t *in* $Node_t \in$ *TargetServices* **then**
8. $TargetServices[PL_t]= TargetServices[PL_t]* IndexTarget[Node_t]$
9. $QueryMatches[PL_t]= QueryMatches[PL_t]* IndexQuery[Node_q]$
10. $XORServices[PL_t]= XORServices[PL_t] * XOR4Composed[Node_t]$
11. **else**
12. $TargetService[PL_t]= IndexTarget[Node_t]$

13. *QueryMatches [PL_t]= QueryMatches [PL_t]* IndexQuery[$Node_t$]*
14. *XORServices[PL_t]= XOR4Composed[$Node_t$]*
15. **end if**

Para iniciar se agrega un nuevo nodo al híper-grafo (línea 6) para representar la actividad recuperada que se está procesando. Luego se crean dos índices que van a servir para identificar que actividades pertenecen a un mismo servicio. Se debe mencionar que el servicio al cual pertenece la operación que se invoca en la actividad básica esta referenciado por el *PartnerLink (PL)* definido en la actividad dentro del proceso BPEL. El índice *TargetServices* se emplea para agrupar las actividades recuperadas que pertenecen a un mismo *PL*, para ello se relaciona al *PartnerLink PL* con el producto de los índices asignados a cada una de sus operaciones (línea 8). El segundo índice que se crea es *QueryMatches*, este se emplea para conocer qué actividades de consulta fueron emparejadas con alguna de las operaciones del servicio recuperado, el *QueryMatches* es el resultado del producto de los índices de las actividades de consulta que fueron emparejadas con alguna de las operaciones que pertenecen al *PartnerLink PL*. Estos dos índices serán empleados durante el proceso de síntesis y servirán para conocer qué servicios de los recuperados poseen las operaciones necesitadas por el proceso de negocio.

Las restricciones de los datos se cumplen por medio de la combinación de las restricciones de servicios y flujo de datos. Haciendo uso de las restricciones de servicios se asegura que las operaciones seleccionadas pertenezcan a un mismo servicio, ahora esas operaciones podrán ser combinadas únicamente si sus interfaces son compatibles, como resultado bastará con comprobar que todas las operaciones de un servicio sean compatibles entre sí para cumplir con las restricciones de datos, esta comprobación se realizará durante la fase de síntesis de composición.

La Tabla 10 muestra un ejemplo útil para entender el manejo de las restricciones de servicios. En la primera columna se tiene el servicio de consulta *AirlineService*, el cual posee tres operaciones utilizadas en el proceso de negocio citado en la sección 4.1. En la segunda y tercera columna se tiene dos servicios candidatos cuyas operaciones fueron recuperadas como parejas para algunas de las operaciones del servicio de consulta. En frente de cada operación se relaciona el índice que se les asigna durante el procesamiento que se realiza para agregarlos al híper-grafo.

Tabla 10. Valores de Índices para las Actividades Básicas

Servicio de consulta: AirlineService	**Servicio A**	**Servicio B**
AirlineReservation = 5 ***AirlineCallback = 17*** ***CancelFlight = 19***	*AirlineReservationA = 11* *AirlineCallbackA = 13* *CancelFlightA = 17*	*AirlineReservationB = 19* *AirlineCallbackB = 23*

Para los servicios recuperados que se listan en la Tabla 10 se calculan los siguientes valores:

- *TargetService[**Servicio A**]=11*13*17 = 2431*
- *QueryMatches[**Servicio A**]=5*17*19 = 1615*
- *TargetService[**Servicio B**]=19*23 = 437*
- *QueryMatches[**Servicio B**]=5*17 = 85*

En el ejemplo se observa que el *Servicio A* ofrece las tres operaciones que se especifican para el servicio de consulta *AirlineService*, este servicio es un candidato apto para ser empleado en la composición del proceso de negocio. Por otro lado, el *Servicio B* solo contiene dos de las tres operaciones requeridas, este servicio debe ser descartado ya que no cumple con las restricciones especificadas para el proceso de negocio que se desea componer. Por medio de los índices calculados *TargetService* y *QueryMatches* se puede identificar cuáles servicios son útiles, la forma como se procesan estos índices es explicada en la siguiente sección donde se aborda la síntesis de de la composición.

5.3 Síntesis de la Composición

Basado en la conformidad de los servicios, flujo y restricciones de datos y restricciones de los servicios, se pueden generar diferentes síntesis de composición como respuesta al requerimiento enviado por el cliente. En esta etapa se desea encontrar soluciones concretas que ofrezcan un proceso de negocio ejecutable conformado por los servicios recuperados y que sigan el comportamiento definido inicialmente en la tarea de usuario solicitada.

5.3.1 Definición del Problema

Considere que se tiene un conjunto de servicios S_q= {sq_1, sq_2,…, sq_n}, el cual representa los servicios requeridos en el proceso de consulta enviado por el cliente, y el conjunto S_t= {st_1, st_2,…, st_m} que representa el conjunto de todos los servicios recuperados para los cuales se cumple que todas sus operaciones hayan sido definidas como ejecutables durante la construcción del hípergrafo, se precisa encontrar una función de síntesis *f:* S_q->S_t donde se cumpla:

1. Para cada servicio sq ∈ S_q existe una pareja st ∈ S_t, tal que para todas las operaciones que pertenecen sq exista una operación similar y con interfaces equivalentes que pertenezca a st.
2. Los servicios que pertenecen al rango de la función *f* no posean operaciones que sean excluyentes a nivel de sus interfaces de entrada y/o salida.

La función que se define hace un emparejamiento entre los servicios definidos en el proceso de consulta y los servicios que fueron recuperados y que poseen operaciones ejecutables. La primera condición definida hace referencia a las restricciones relacionadas con las operaciones que debe contener un servicio para ser considerado como útil en el proceso de composición. La segunda condición busca obtener una consistencia a nivel de las interfaces de entrada y/o salida de los servicios que se seleccionan para combinar en el proceso compuesto, no podrán coexistir en un mismo proceso dos servicios que tengan operaciones que no son compatibles a nivel de sus interfaces y que por el diseño del requerimiento del cliente deban intercambiar datos.

En la síntesis de composición se busca encontrar esta función que señale el emparejamiento entre los servicios de consulta y recuperados, a partir de estas parejas se podrá construir el proceso de negocio concreto que dará respuesta al requerimiento del usuario. A continuación se presenta el proceso que se sigue para encontrar la función de síntesis de servicios y posteriormente la especificación del proceso de negocio basada en las operaciones de los servicios seleccionados.

5.3.2 Integración de Servicios

La integración de servicios tiene como objetivo explorar las posibles combinaciones de los servicios recuperados tomando en cuenta el flujo de datos definido en el proceso requerido por el

usuario. La integración se realiza verificando si los servicios recuperados no poseen incompatibilidades a nivel de sus interfaces de entrada y/o salida, posteriormente se escoge la combinación que se asemeja más al requisito del usuario, con dichos servicios recuperados se construye un proceso de negocio concreto siguiendo el flujo de control y datos especificado en el proceso abstracto que representa la tarea del usuario.

En la Figura 33 se ilustran dos conjuntos de ejemplo, el de la columna izquierda contiene los servicios que pertenece al proceso de consulta citado en la sección 4.1, mientras que el conjunto de la derecha agrupa los servicios recuperados. Los servicios recuperados para cada servicio de consulta han sido organizados por filas en la gráfica.

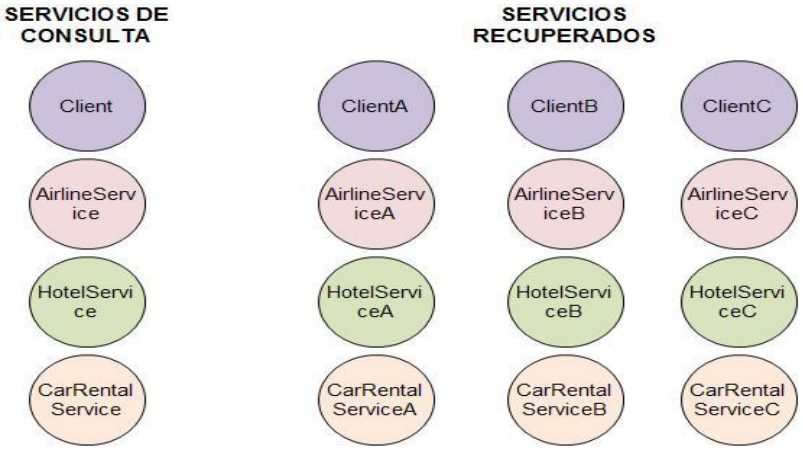

Figura 33. Ejemplo de Conjuntos de Servicios de Consulta y Recuperados

Tomando como entradas los servicios de consulta y los recuperados se debe encontrar una función que relacione los dos conjunto tal como se describió en la sección anterior. La conformación de la función se inicia realizando un filtrado de los servicios recuperados para determinar que candidatos ofrecen las mismas operaciones que los servicios requeridos. El Algoritmo 17 especifica la función para filtrar los servicios y crear una lista ordenada de acuerdo a la similitud entre los servicios recuperados y los de consulta.

Algoritmo 17. Función Ordenar Servicios– SortTargetServices

1. **INPUTS:** (QueryServices[], TargetServices[])
2. **OUTPUT**: $\mathbf{ServicesMatches}$

3. **BEGIN**
4. *ServicesMathces[] = ∅*
5. **for each** PL_q **in** *QueryServices[]* **do**
6. *Sq = QueryServices[*PL_q*]*
7. *Matches = {}*
8. **for each** PL_t **in** *TargetServices[]* **do**
9. *St = TargetServices[*PL_t*]*
10. *Qm = QueryMatches[*PL_t*]*
11. **if** *Sq == Qm* **then**
12. **if** *(*$\forall\ \boldsymbol{node_t} \in \boldsymbol{PL_t}$ *state[*$\boldsymbol{node_t}$*] ≠ Not Executable)* **then**
13. *serviceDistance =* $\sum \boldsymbol{NodeDistance}[\ \boldsymbol{node_t}] \mid\ \boldsymbol{node_t} \in \boldsymbol{PL_t}$
14. *Matches = Matches ∪* PL_t*, ordered by serviceDistance*

15. **end if**
16. **end if**
17. **end for**
18. *ServicesMatches add (PL_q, Matches)*
19. **end for**
20. **return** *ServicesMatches[]*

Las entradas del Algoritmo 17 son los conjuntos *QueryServices* (este conjunto es armado cuando se procesan los nodos de consulta en el Algoritmo 12 línea 6) y *TargetServices* (este conjunto se obtiene en el Algoritmo 16 línea 8); los dos conjuntos tienen entre sus elementos los *PartnerLinks* de consulta y los recuperados respectivamente. El algoritmo empieza a recorrer los servicios de consulta (Algoritmo 17 línea 5) y los compara contra los servicios recuperados, la comparación busca identificar aquellos servicios recuperados que poseen operaciones similares y compatibles a nivel de interfaces con las operaciones que requiere el servicio de consulta procesado. Para realizar la comparación a nivel de operaciones se hace uso de dos índices construidos previamente *QueryServices* (Algoritmo 12 línea 6) y *QueryMatches* (el Algoritmo 16 línea 9), el índice Q*ueryServices* indica que actividades de consulta pertenecen a un determinado servicio, *QueryMatches* indica que actividades de consulta fueron emparejadas con una de las operaciones que pertenecen a un servicio recuperado, de acuerdo a esto si estos dos índices tienen el mismo valor quiere decir que el servicios de consulta y el recuperado son equivalentes a nivel de sus operaciones (Algoritmo 17 línea 11). Una vez verificado que el servicio recuperado es equivalente al de consulta se constata que todas las operaciones emparejadas del servicio recuperado sean ejecutables. Finalmente, se crea un conjunto (*Matches*) en el que se almacenan todos los servicios recuperados compatibles con el servicio de consulta que se está procesando, los elementos del conjunto *Matches* se ordena de acuerdo a la similitud estimada para cada servicio recuperado. La similitud de dos servicios es igual a la sumatoria de las similitudes de sus operaciones (Algoritmo 17 línea 13). Al finalizar el algoritmo entrega un arreglo de parejas ordenadas, cada pareja contiene un servicio de consulta y el conjunto de servicios recuperado ordenados por su similitud.

Después de seleccionar y ordenar los servicios recuperados, que serán aptos para llevar a cabo la composición del proceso de negocio, se debe explorar las posibles combinaciones que permitirán obtener un proceso concreto basado en los servicios recuperados. Las combinaciones de los servicios se generan como un árbol, donde los nodos representan los servicios recuperados. La Figura 34 muestra la conformación de un árbol empleando los conjuntos de servicios descritos en la Figura 33, los servicios recuperados que se encuentran a una misma profundidad, medida desde el nodo raíz, son equivalentes a un mismo servicio de consulta. Cada uno de los caminos que se puedan recorrer en el árbol desde el nodo raíz hasta la última hoja, con una profundidad igual al número de servicios de consulta requeridos, constituyen una alternativa de composición de servicios para realizar la tarea de usuario requerida.

El Algoritmo 18 presenta la secuencia de pasos para construir el árbol de servicios recuperados, a través del cual se seleccionará la síntesis de composición para responder al requisito enviado por el usuario. La función construir árbol de servicios recibe como entrada el conjunto de parejas ordenadas que genera el Algoritmo 17 y entrega como resultado un árbol que contiene los servicios seleccionados para sintetizar el proceso de consulta recibido. El Algoritmo 18 no genera todas las ramas que puede contener el árbol, su ejecución se detiene cuando se alcanza un camino que contenga todos los servicios requeridos para componer la tarea de usuario. Debido a que los servicios recuperados son agregados en orden, de acuerdo a su similitud, se tiene que los caminos

generados inicialmente poseen una mejor similitud que aquellos que se generan posteriormente. Esto se hace así, para evitar generar todas las posibles combinaciones de servicios y disminuir los tiempos de ejecución del algoritmo.

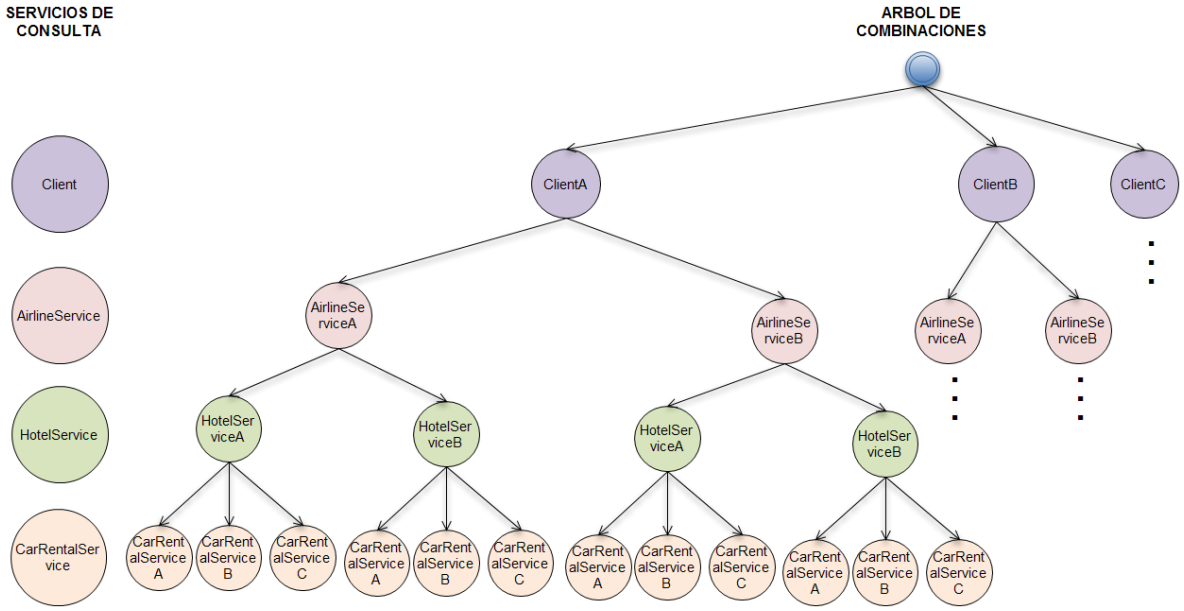

Figura 34. Árbol de Servicios Recuperados

Algoritmo 18. Función Construir Árbol de Servicios– BuildServicesTree

1. **INPUTS:** (ServicesMatches[])
2. **OUTPUT**: Tree

3. **BEGIN**
4. *TreeDepth = Number of Query Services*
5. *Depth= 0*
6. *Tree = ∅*
7. *TreeIndexXOR = 1*
8. *TreeIndexAND = 1*
9. **while** *Depth < TreeDepth* **do**
10. *(queryService, Matches) = ServicesMatches get Query Service in (Position = Depth)*
11. *MatchService = GetNextServiceMatch(Matches, Tree get node at Depth)*
12. **if** *MatchService ≠ ∅* **then**
13. *Tree add MatchService in (Position = Depth)*
14. *TreeIndexXOR = TreeIndexXOR * XORServices [MatchService]*
15. *TreeIndexAND = TreeIndexAND * TargetService[MatchService]*
16. **else**
17. *Depth = Depth -1*
18. **if** *Depth >=* 0 **then**
19. *RemovedService = Tree get node at Depth*
20. *TreeIndexXOR = TreeIndexXOR / XORServices [RemovedService]*
21. *TreeIndexAND = TreeIndexAND / TargetService[RemovedService]*

```
22.          else
23.              return ∅
24.          end if
25.      end if
26.  end while
27.  return Tree
```

El Algoritmo 18 inicia definiendo la profundidad que deberá tener el árbol, en este caso debe ser equivalente al número de servicios requeridos (línea 4). A continuación, se recorre el conjunto de parejas ordenadas que contienen los servicios de consulta y los recuperados, en este paso se debe seleccionar un servicio recuperado por cada servicio de consulta (línea 11), esto es realizado por la función *GetNextServiceMatch* (Algoritmo 19). Si la función *GetNextServiceMatch* entrega un servicio recuperado que sea compatible a nivel de sus interfaces con los servicios previamente agregados en el árbol (línea 12) se procede a adicionar el nuevo servicio recuperado al árbol. Cuando se agrega un nuevo servicio se crea un nodo dentro del árbol y se actualizan los índices de composición, *TreeIndexXOR* contiene los índices de aquellos servicios recuperados que no pueden ser agregados posteriormente al árbol ya que sus operaciones no son compatibles con las operaciones que ya fueron agregadas; el índice *TreeIndexAND* contiene el producto de los índices de todas las actividades recuperadas que hacen parte de los servicios agregados al árbol. En caso de que no se pueda obtener un servicio recuperado para agregar al árbol (línea 16) se debe retroceder un nivel e intentar generar una nueva combinación de servicios, esto se hace hasta cuando la profundidad sea igual cero, si se rebasa este límite quiere decir que no es posible obtener una síntesis de composición entre los servicios recuperados por la incompatibilidad de sus interfaces, en esta situación se termina la ejecución del algoritmo y se retorna un valor de vacio (línea 23).

El Algoritmo 19 describe la función para seleccionar el servicio recuperado más similar a un servicio de consulta y que sea compatible con aquellos servicios que previamente fueron agregados al árbol de composición. Recibe como entradas el conjunto de servicios recuperados (*Matches*) y el servicio seleccionado previamente (*MatchSelected*) para el servicio de consulta procesado. Inicialmente, se verifica si previamente se seleccionó un servicio recuperado (línea 6), si no es así se escoge el primer servicio en el conjunto *Matches*, en caso contrario se selecciona el servicio recuperado que está a continuación del servicio seleccionado en el conjunto *Matches*. De esta forma se van escogiendo los servicios de una manera ordenada de acuerdo a la similitud estimada. Una vez seleccionado el servicio recuperado, se verifica si es compatible con los servicios ya agregados al árbol de composición, esto se hace por medio de los índices *TreeIndexXOR* y *TreeIndexAND;* obteniendo el MCD entre los índices de incompatibilidad del servicio y del árbol se puede detectar si las operaciones del servicios que se quiere agregar pueden coexistir con los servicios seleccionados previamente. Cuando se encuentra un servicio apto se detiene el bucle de ejecución y se retorna el nuevo servicio seleccionado. En caso contrario, se selecciona al siguiente servicio recuperado de la lista y se repite el proceso. Si se recorre la lista por completo y no se encuentra un servicio recuperado compatible se entrega una respuesta vacía para que se replantee la construcción del árbol desde un nivel superior.

Algoritmo 19. Función Obtener Siguiente Servicio– GetNextServiceMatch

```
1. INPUTS: (Matches[], MatchSelected)
2. OUTPUT: Service
```

```
3.  BEGIN
4.  isSelected = false

5.  do
6.      if MatchSelected == ∅ then
7.          newMatchService = Matches get first Target Service
8.      else
9.          if Matches has next Target Service (MatchSelected) then
10.             newMatchService = Matches get next Target Service (Tree get node at Depth)
11.         else
12.             return ∅
13.         end if
14.     end if
15.     mcd1 = MCD(TreeIndexXOR, TargetService[newMatchService])
16.     mcd2 = MCD(TreeIndexAND, XORServices [newMatchService])
17.     if mcd1==1 and mcd2==1 then
18.         isSelected = true
19.     end if
20.     MatchSelected = newMatchService
21. while isSelected == false
22. return newMatchService
```

5.3.3 Grafo Compuesto

Una vez seleccionada la síntesis de composición que se utilizará para componer la tarea de respuesta se debe generar un nuevo grafo que presente el comportamiento requerido por el usuario utilizando los servicios escogidos. La creación del nuevo grafo se realiza siguiendo el flujo de control y datos definido en el grafo de consulta. El recorrido del grafo de consulta se realiza utilizando un algoritmo DFS similar al presentado en el Algoritmo 10. El procesamiento de los nodos se realiza siguiendo el Algoritmo 20.

El Algoritmo 20 recibe como entrada el nodo de consulta que se desea procesar. De acuerdo al tipo de nodo se realiza un procesamiento diferente. Los nodos tipo *Start, End* y *Connector* se adicionan de forma directa al grafo de composición. Los nodos tipo *BasicActivity* son remplazados por el nodo recuperado, esto se hace consultando los servicios seleccionados y obteniendo la operación correspondiente al nodo de consulta. Los nodos tipo *Assign* son actualizados incluyendo el nombre de las variables que serán utilizadas en el proceso de negocio de acuerdo a las interfaces de los servicios seleccionados.

Finalmente, una vez remplazados los nodos en el grafo compuesto se crean las aristas que representan la secuencia de ejecución y control para el proceso de negocio. Una vez completado el grafo de composición debe ser traducido a un modelo de comportamiento en el lenguaje de ejecución escogido para la implementación, en este caso particular BPEL.

Algoritmo 20. Función Crear Grafo Compuesto –BuildComposedGraph

1. **INPUTS**: ($Node_q$); $Node_q$: Struct (Op_q, PT_q, PL_q, Typeq, $Input_q$, $Output_q$)

2. **OUTPUT**: ∅

3. **BEGIN**
4. **if** $Node_q \in Start$ **then**
5. *GraphComposed add Start* $Node_q$
6. **end if**

7. **if** $Node_q \in Connector$ **then**
8. *GraphComposed Connector* $Node_q$
9. **end if**

10. **if** $Node_q \in BasicActivity$ **then**
11. *MatchService = Tree get Match for* PL_q
12. $Node_t$ *= MatchService get Activity for* $Node_q$
13. *GraphComposed AddActivityNode(*$Node_q$*)*
14. **end if**

15. **if** $Node_q \in Assign$ **then**
16. *AddAssignNode(*$Node_q$*)*
17. **end if**

18. **if** $Node_q \in End$ **then**
19. *GraphComposed add End* $Node_q$
20. **end if**

5.4 Experimentación y Evaluación

En esta sección se expone la metodología de evaluación usada, las medidas que se establecieron como criterios de evaluación, y las pruebas ejecutadas a los algoritmos propuestos para establecer su rendimiento.

5.4.1 Metodología de Experimentación

Para medir el rendimiento de los algoritmos de composición se procedió de la siguiente manera:

1. Se construyó un repositorio de procesos de pruebas, los procesos contenidos en el repositorio se especificaron como procesos abstractos, se adicionaron los archivos WSDL para describir las operaciones junto con sus entradas y salidas.
2. Se midió el rendimiento de los algoritmos variando el número de actividades que conforman el proceso de consulta, para estas pruebas se mantuvieron constantes el número de actividades a recuperar del repositorio para el proceso de composición.
3. Se midió el rendimiento de los algoritmos variando el número de actividades que se desean recuperar del repositorio de procesos, para este caso se mantuvo constante el número de actividades que componen el proceso de consulta.

El repositorio de procesos se conformó con 22 documentos BPEL, distribuidos en cinco dominios diferentes relacionados con procesos de información turística: *CarRental* (6 procesos de negocio), *Guesthouse* (2 procesos), *Holidays* (4 procesos), *hotel* (4 procesos) y *Travel* (6 procesos). Estos 22 procesos BPEL incluyen un total de 148 actividades básicas las cuales servirán como operaciones

publicadas y que estarán a disposición para ser empleadas en los procesos de descubrimiento y composición de servicios.

5.4.2 Criterios de Evaluación

Los criterios de evaluación para los algoritmos de composición de servicios se basan en el conjunto de reglas sobre los tiempos de respuesta para aplicaciones presentada por (Joines, y otros, 2002) y que se citaron previamente en la sección 4.4.3. Como ya se explicó estos criterios fijan unos umbrales de tiempo y unas categorías de respuesta (óptimo, bueno, aceptable y deficiente) que permiten valorar la eficiencia de un sistema o una aplicación.

5.4.3 Plan de Pruebas

Las pruebas de rendimiento se hicieron sobre los módulos que implementan las funcionalidades de la fase de composición, estas son: la conformidad de los servicios, la síntesis y selección de la composición. Las pruebas se ejecutaron sobre una aplicación que integra todos los algoritmos que hacen parte de la solución de composición presentada en este capítulo.

A continuación, en la Tabla 11 se describen las pruebas de eficiencia realizadas a las aplicaciones software que conforman la solución de composición.

Tabla 11. Plan de Pruebas

Composición de Servicios	**Prueba de rendimiento PR3**: Determina el tiempo que tarda el módulo de composición en entregar un resultado ante un proceso de consulta entrante. Se varía el número de actividades de consulta y se mantiene constante el número de actividades recuperadas. **Prueba de rendimiento PR4**: Determina el tiempo que tarda el módulo de composición en entregar un resultado ante un proceso de consulta entrante variando número de actividades recuperadas. Para estas pruebas se mantiene constante la cantidad de actividades que conforman el proceso de consulta.

5.4.4 Resultados y Discusión

Se implementó un sistema de descubrimiento de servicios sobre un repositorio de documentos BPEL (Vanhatalo, y otros, 2006), el cual almacena 22 archivos BPEL y cuenta con un total de 148 actividades básicas BPEL. El algoritmo de composición se implementó en el lenguaje Java y los experimentos fueron realizados en un computador con procesador Pentium 4 2,30GHz, 1.000 MB de RAM bajo el S.O. Linux Ubuntu.

5.4.4.1 Prueba de Rendimiento PR3

En esta sección, se presentan los resultados del estudio del tiempo de ejecución del algoritmo de composición midiendo su comportamiento cuando se varía el número de actividades básicas y datos de entrada y/o salida que hacen parte del documento BPEL de consulta. Para esta prueba se contó con 18 archivos BPEL de consulta (ver Tabla 12) en los cuales se incrementó gradualmente el número de actividades básicas empleadas en el proceso de negocio. Este incremento significó un aumento en el número de datos que se emplearon y se intercambiaron las operaciones especificadas. En la Tabla 12 se muestra el cálculo del número total de nodos procesados por el algoritmo de composición al momento de crear el híper-grafo de acuerdo a los parámetros de entrada de los documentos BPEL de consulta, este número total corresponde a la suma del número de actividades básicas y el número de datos que se declaran en el documento BPEL al momento de definir las variables de entrada y salida de las actividades.

Tabla 12. Archivos de Consulta BPEL para Pruebas de Composición

BPEL Consulta	Actividades Básicas	Datos	Total Nodos		BPEL Consulta	Actividades Básicas	Datos	Total Nodos
BPEL 1	3	8	**11**		**BPEL 10**	12	47	**59**
BPEL 2	4	10	**14**		**BPEL 11**	13	50	**63**
BPEL 3	5	14	**19**		**BPEL 12**	14	30	**44**
BPEL 4	6	30	**36**		**BPEL 13**	15	34	**49**
BPEL 5	7	27	**34**		**BPEL 14**	16	53	**69**
BPEL 6	8	36	**44**		**BPEL 15**	17	56	**73**
BPEL 7	9	36	**45**		**BPEL 16**	18	56	**74**
BPEL 8	10	20	**30**		**BPEL 17**	19	59	**78**
BPEL 9	11	44	**55**		**BPEL 18**	20	62	**82**

Para las pruebas se ingresaron los documentos de consulta a la aplicación de descubrimiento y composición y se midió el tiempo que tarda en realizar las tareas de composición, por cada documento de consulta se tomaron cinco muestras y se promediaron los tiempos para obtener el tiempo de ejecución. Las pruebas se realizaron en diferentes escenarios de acuerdo al número de actividades que se deseaban recuperar del repositorio de procesos, variando el número *K* de actividades recuperadas desde 3 hasta 14. En la Figura 35 se muestran las gráficas obtenidas del comportamiento de la aplicación de composición para tres casos, cuando se recuperaron 4, 9 y 14 actividades del repositorio de procesos, en el Anexo C se incluyen todos los datos y graficas obtenidas en la ejecución de las pruebas de rendimiento PR3.

En las gráficas presentadas en la Figura 35 se puede observar que el tiempo de respuesta de los algoritmos de composición varía linealmente a medida que se incrementan la cantidad de actividades básicas y datos que pertenecen al proceso de consulta.

Para los escenarios probados (3 a 14 actividades recuperadas) la relación del tiempo de respuesta contra los nodos de entrada se puede generalizar como una recta:

- ***t = m*q + C***; donde ***t*** tiempo, ***m*** pendiente de la recta, ***q*** nodos de consulta y ***C*** una constante

La pendiente (*m*) de la recta varía de acuerdo a la cantidad de actividades recuperadas, entre mayor sea el número de actividades recuperadas mayor es la pendiente de la recta. El comportamiento de la pendiente (*m*) se analizará en las pruebas de rendimiento PR4, cuando se obtenga el comportamiento del algoritmo al variar el número de actividades recuperadas.

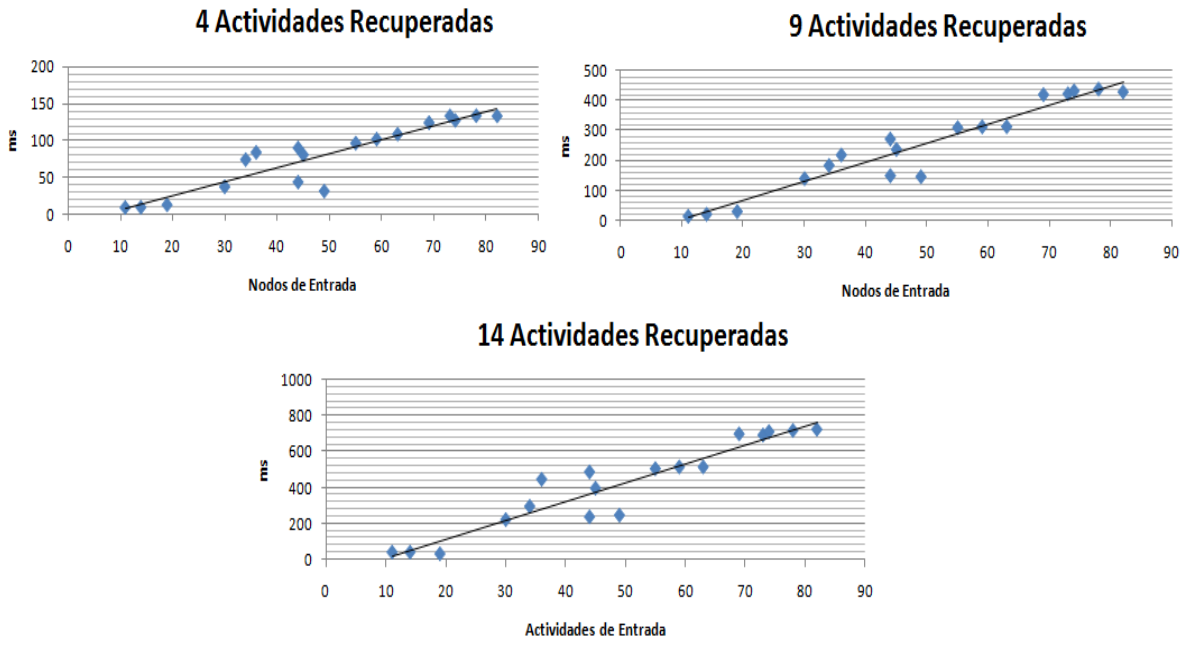

Figura 35. Gráficas de Rendimiento del Algoritmo de Composición en Función de los Nodos de Entrada

Tomando como referencia (Joines, y otros, 2002) se determinó los siguientes tipos de comportamiento:

- **Óptimo** (tiempo de respuesta menor o igual a 0,1s.): cuando el número de nodos de consulta es inferior a 20. Esto se mantiene para todos los escenarios donde se varió el número de actividades recuperadas (de 3 hasta 14 actividades recuperadas).
- **Bueno** (tiempo de respuesta entre 0,1s. y 1s.): en esta categoría se encuentra el comportamiento del algoritmo para documentos de consulta que posean una cantidad de nodos de entrada, entre actividades básicas y datos, entre 20 y 82. Para todos los escenarios de pruebas con distinto número de actividades recuperadas se alcanzaron tiempos de respuesta inferiores a 1 segundo en la ejecución de los algoritmos de composición.

5.4.4.2 Prueba de Rendimiento PR4

En este apartado se presentan el comportamiento de los algoritmos de composición cuando se varía el número de actividades básicas recuperadas del repositorio de procesos. Estas pruebas de rendimiento se realizaron con 18 archivos BPEL de consulta (ver Tabla 12), los cuales emplean un número diferentes de actividades básicas en el proceso de negocio. Se midieron los tiempos de respuesta de los algoritmos a medida que se incrementaba el número de actividades recuperadas en un rango de 3 a 14 actividades, mientras se mantenía constante las actividades y datos que componen el proceso de consulta. Para cada incremento en el número de actividades recuperadas se tomaron cinco muestras de tiempos, las cuales se promediaron para obtener el tiempo de ejecución.

En la Figura 36 se muestran las gráficas obtenidas del comportamiento de la aplicación de composición para tres casos, cuando se tienen procesos BPEL de consulta de 11, 59 y 82 nodos de entrada (entre actividades básicas y datos), en el Anexo C se incluyen todos los datos y graficas obtenidas en la ejecución de las pruebas de rendimiento PR4.

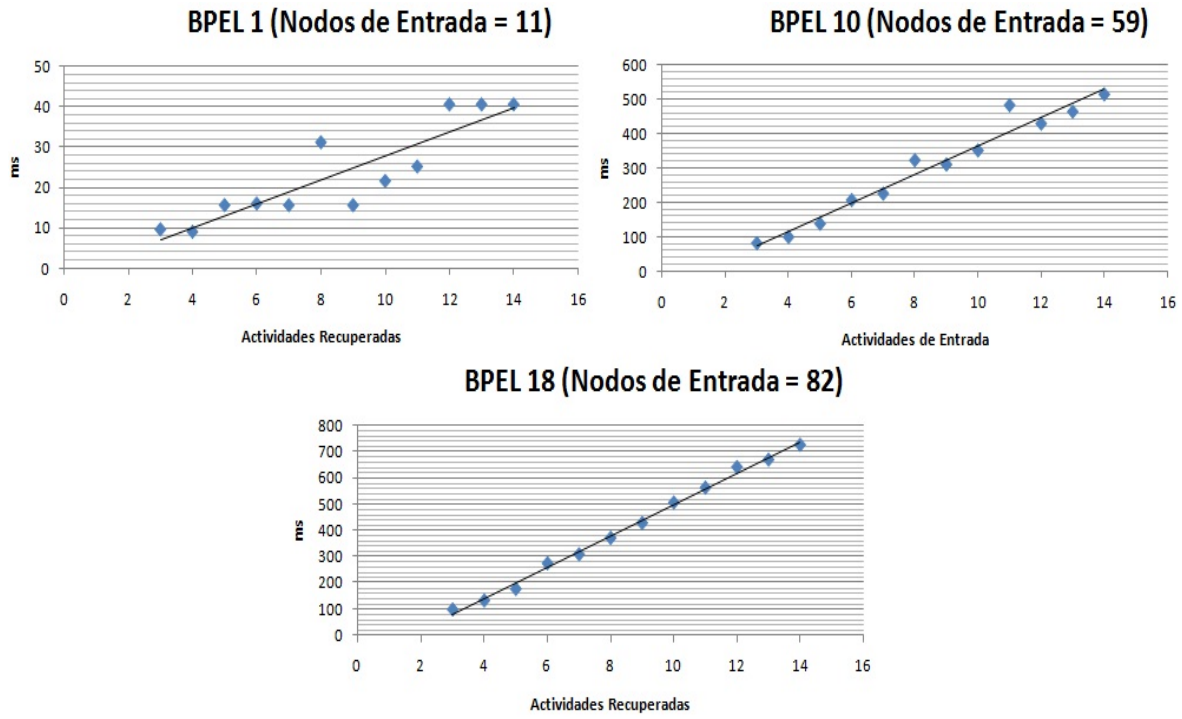

Figura 36. Gráficas de Rendimiento del Algoritmo de Composición en Función de las Actividades Recuperadas

En las gráficas presentadas en la Figura 36 se puede observar que el tiempo de respuesta de los algoritmos de composición varía linealmente a medida que se incrementan la cantidad de actividades básicas recuperadas.

Para los escenarios probados (11 a 82 nodos de entrada) la relación del tiempo de respuesta contra las actividades recuperadas se puede generalizar como una recta:

- ***t = p*k + C***; donde ***t*** tiempo, ***p*** pendiente de la recta, ***k*** nodos recuperados y ***C*** una constante

La pendiente (*p*) de la recta varía de acuerdo a la cantidad de actividades y datos de entrada, entre mayor sea el número de nodos de entrada mayor es la pendiente de la recta.

Considerando el comportamiento de los algoritmos de composición frente a variaciones en la cantidad de actividades de consulta y el número de actividades recuperadas que entrega el módulo de descubrimiento se puede concluir que la complejidad de la solución de composición está dada por:

- ***t = q*k***; donde ***q*** es el número de actividades y datos de consulta, y ***k*** el número de actividades recuperadas

La complejidad computacional de los módulos de descubrimiento y composición está determinada por el producto del número de actividades de consulta y la cantidad de actividades recuperadas, comportamiento esperado ya que los algoritmos realizan comparaciones uno a uno de las actividades que pertenecen a los dos conjuntos. El número de comparaciones es disminuido por medio del filtrado de candidatos basadas en el tipo de actividad y dominio de los procesos de negocio, especialmente en la fase de descubrimiento.

Tomando las conclusiones obtenidas en el Capítulo 4 para el módulo de descubrimiento, donde se determinó que el rango de referencia para el k óptimo del sistema se encuentra entre 1 y 7 actividades recuperadas, valores en los que la precisión se ajusta a los parámetros requeridos para el mejor desempeño. Se puede constatar que el algoritmo de composición presenta en general un comportamiento bueno según los criterios de (Joines, y otros, 2002), cuando se lleva el *k* hasta el umbral de 7 actividades recuperadas el tiempo máximo de respuesta fue 309 ms cuando se ingresaron 82 nodos de consulta (ver Figura 37). Por otro lado, el algoritmo entrega una respuesta óptima (Joines, y otros, 2002) cuando los nodos de entrada son inferiores a 30 y se tiene un *k* igual a 7 actividades recuperadas, si se reduce el número *k* de actividades recuperadas la respuesta óptima es lograda para un mayor número de nodos de consulta, para el caso de *k* igual a 3 se logra una respuesta óptima hasta en procesos de negocio con 82 nodos de consulta.

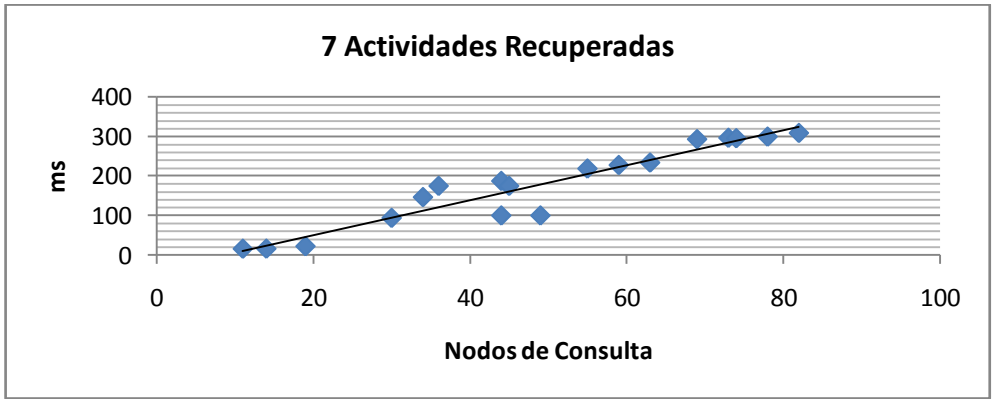

Figura 37. Gráficas de Rendimiento del Algoritmo de Composición para un *k*=7 Actividades Recuperadas

Resumiendo, se puede concluir que los algoritmos de composición presentaron un comportamiento bueno en las pruebas de rendimiento. Se contó con una muestra importante de evaluaciones soportadas en un repositorio de procesos orientados al dominio turístico, el cual fue provisto con un una cantidad de documentos BPEL y actividades básicas significativas de acuerdo a los trabajos relacionados. Se probaron las aplicaciones con documentos de consultas conformados con una cantidad de actividades semejante a otros trabajos relacionados y que están acordes con el tamaño de los requisitos de un usuario en un ambiente ubicuo.

Se comprobó que el algoritmo de composición funciona de una manera buena cuando el número de actividades recuperadas *k* se encuentra en el rango de desempeño óptimo (3 a 7 actividades atómicas recuperadas). Este comportamiento es el ideal, ya que se desea por un lado asegurar la calidad en la similitud entre los requisitos del usuario y las actividades recuperadas, y por el otro una respuesta rápida dentro de los umbrales para un correcto funcionamiento de una aplicación desde el punto de vista del usuario.

Capítulo 6

Conclusiones

6.1 Aportes

En esta tesis de maestría se propone una arquitectura de referencia para la composición de servicios basada en modelos de comportamiento. La arquitectura de referencia se sustenta sobre cuatro niveles, el nivel de servicios donde se concentran todas las funcionalidades del ambiente ubicuo, estas son abstraídas como servicios y puestas a disposición de los usuarios; el nivel de modelos, capa donde se encuentran las funcionalidades necesarias para adelantar el descubrimiento y composición de servicios, en este nivel se manejan los modelos de comportamiento que permiten expresar los requisitos del cliente y los procesos de negocio publicados en la red ubicua; el nivel de contexto agrupa los elementos para definir el contexto de entrega del usuario, permite incorporar nuevas características al procesos de descubrimiento con el fin de recuperar servicios más pertinentes para el usuario de acuerdo a su perfil y las capacidades asociadas a sus dispositivos; el nivel de aplicaciones contempla todas aquellas aplicaciones que pueden ser empleadas por el usuario para interactuar con el ambiente y realizar sus tareas.

La arquitectura de referencia fue instanciada para un ambiente ubicuo basado en dispositivos móviles. La implementación de referencia presentada abarca el descubrimiento y composición de servicios dentro de un ambiente ubicuo, dejando por fuera los procesos de diseño y publicación de los modelos de comportamiento disponibles en la red ubicua; el alcance de la implementación referencia está limitado por los objetivos planteados al iniciar el presente trabajo de maestría. A partir de esta implementación de referencia se construyeron los prototipos empleados para realizar las pruebas de desempeño y rendimiento de las soluciones presentadas para el descubrimiento y composición de servicios.

En la implementación de referencia se seleccionó a los servicios web como la tecnología para implementar las funcionalidades que se publican en la red ubicua y que serán consumidos por los usuarios. De acuerdo a lo anterior, BPEL fue escogido como el lenguaje para expresar los modelos de comportamiento que representan las tareas del usuario y los procesos de negocio que publican los proveedores de servicios. Las herramientas y tecnologías para la implementación de los componentes de la arquitectura de referencia fueron escogidas teniendo en cuenta el alcance propio de la implementación de referencia: dos repositorios de dispositivos móviles UAProf (Forum, 2001) y WURLF (Passani, 2010) para determinar el contexto de entrega de los dispositivos empleados por el usuario; un repositorio de documentos BPEL (Vanhatalo, y otros, 2006), en el cual se publican los procesos de negocio disponibles en la red ubicua; y el motor de ejecución Sliver (Hackmann, y otros, 2007), el cual soporta la ejecución de procesos de negocio BPEL sobre dispositivos de bajos recursos.

Una vez diseñada la arquitectura de referencia, se procedió a definir las aproximaciones para soportar las funcionalidades de descubrimiento y composición de servicios basadas en los

componentes que conforman la arquitectura de referencia. La aproximación de composición se soportó en una fase de descubrimiento previa, la cual utiliza técnicas de emparejamiento que operan sobre actividades atómicas y que permiten obtener una estimación de la distancia lingüística entre las actividades recuperadas y las requeridas por el usuario. En este sentido, si no existe un servicio que satisfaga exactamente los requisitos del usuario se proponen aquellos más similares y que fueron recuperados desde la red ubicua. El emparejamiento de actividades se complementó con una fase de filtrado que considera la características de los dispositivos empleados por el usuario, con esto logró seleccionar actividades que fueran realmente relevantes al contexto de entrega del usuario, y que pudieran ser consumidas desde el dispositivo de acceso considerando sus capacidades hardware y software.

Para la aproximación de composición se utilizó una técnica de integración de servicios atómicos guiada por el comportamiento definido en las tareas de usuario. Con esta técnica se logró generar la síntesis de composición, la cual se rige por el comportamiento expresado por el usuario, identificando de manera precisa los servicios que deben participar y la información que puede circular durante la ejecución de los procesos, incorporando un alto grado de confiabilidad en el proceso de composición. Por otro lado, la integración de servicios atómicos fue escogida teniendo en cuenta su mayor escalabilidad, en comparación con la integración de comportamiento, aspecto que se vio reflejado en el buen desempeño de los algoritmos respecto a los tiempos de respuesta, según los criterios establecidos por (Joines, y otros, 2002).

Las aproximaciones para el descubrimiento y composición de servicios se sustentaron sobre una representación formal de grafos, bajo esta representación se construyeron los algoritmos empleados para solucionar el problema de composición de servicios en los ambientes de computación ubicua. Los grafos fueron empleados para expresar de una manera formal el comportamiento definido en un proceso de negocio BPEL, se representó cada actividad básica como un conjunto de elementos que corresponden a los atributos especificados en BPEL, el flujo de datos fue capturado por medio de las entradas/salidas de cada actividad y las variables o mensajes que se definen para cada proceso de negocio, las estructuras complejas fueron abstraídas por medio de operadores lógicos AND y XOR, finalmente las actividades básicas y complejas fueron enlazadas por medio de aristas, permitiendo representar el flujo de ejecución de los procesos.

Para la síntesis de composición se empleó los híper-grafos como herramienta para establecer las dependencias entre los servicios e identificar las alternativas de síntesis que se podrían generar a partir de los servicios recuperados desde la red ubicua. El grafo de dependencias permitió establecer qué servicios podían ser integrados para alcanzar el comportamiento deseado, basándose en la compatibilidad de sus interfaces y en las restricciones de precedencia y ejecución de las actividades. La selección de la síntesis de composición se realizó construyendo un árbol con los servicios candidatos, donde los caminos se crean empleando servicios que son compatibles y que pueden ser ejecutados bajo un mismo proceso de negocio. En la selección se busca un camino que contenga todos los servicios requeridos para la ejecución de la tarea de usuario, el algoritmo de búsqueda se detiene cuando encuentra un camino con todos los servicios deseados.

En general se pudo concluir que los algoritmos propuestos para el descubrimiento y composición de servicios presentaron una complejidad $O(mn)$. Para el caso del descubrimiento m y n representan el número de actividades publicadas y las actividades de consulta respectivamente. En este caso, el comportamiento fue el esperado para una solución que realiza comparaciones uno a uno, permitiendo recuperar actividades atómicas. Para el caso de estudio, estas comparaciones

se redujeron haciendo una clasificación previa de acuerdo al tipo de actividades, de tal forma que solo se compararon actividades que poseían el mismo tipo de actividad básica BPEL. Con esto se disminuyó significativamente el número de comparaciones realizadas, en especial cuando se incrementa la cantidad de actividades presentes en el repositorio de procesos. Al considerar el contexto en la fase de descubrimiento se presentó un incremento en el tiempo de respuesta, correspondiente a la fracción que toma la identificación del contexto del dispositivo móvil, asociado principalmente con los tiempos necesarios para consultar los repositorios externos (UAProf y WURLF) y la consideración de parámetros adicionales en el emparejamiento.

Para el algoritmo de composición la complejidad *O(mn)* está determinada por el número de actividades básicas de consulta, datos de consulta y la cantidad de actividades recuperadas. Para este caso, es importante resaltar que se debe considerar la cantidad de datos de entrada y salida de las actividades básicas, ya que durante la conformación del grafo de dependencias se realiza una comparación uno a uno entre los datos de las actividades de consulta y las recuperadas.

En resumen, tomando las conclusiones obtenidas en el Capítulo 4 y 5, se determinó lo siguiente:

1.) El mejor desempeño total del sub-sistema de descubrimiento se obtiene para valores de similaridad superiores a 4,41, rango en el cual la precisión oscila entre 0,7 y 0,9. Tomando este rango como referencia el k óptimo del sistema se encuentra entre 1 y 7 actividades recuperadas, valores en los que la precisión se ajusta a los parámetros requeridos para el mejor desempeño general. Para alcanzar estos valores se debe implementar un filtrado de actividades por dominio que ayude a obtener un comportamiento similar al caso 1 de la evaluación realizada para el descubrimiento de servicios.

2.) La incorporación de la información del contexto al proceso de descubrimiento permitió obtener un desempeño total superior del 0,7 en un rango de similitud mayor a 4,0, siendo más amplio al que se obtuvo en las pruebas sin contexto. En este rango de similitudes se tiene un comportamiento superior y más sostenido en términos de desempeño Total del algoritmo, demostrando el impacto favorable de la incorporación de la información del contexto en las funciones de recuperación de servicios.

3.) El algoritmo de composición en general presenta un comportamiento bueno según los criterios de (Joines, y otros, 2002) cuando el rango de referencia para el *k* se encuentra entre 1 y 7 actividades recuperadas. Cuando se lleva el k hasta el umbral de 7 actividades recuperadas el tiempo máximo de respuesta fue 309 ms cuando se ingresaron 82 nodos de consulta. Por otro lado el algoritmo de composición entrega una respuesta óptima (Joines, y otros, 2002) cuando los nodos de entrada son inferiores a 30 en el caso de un k igual a 7 actividades recuperadas; si se reduce el número k de actividades recuperadas, la respuesta óptima es lograda para un mayor número de nodos de consulta, en el caso de k igual a 3 se logra una respuesta óptima hasta en procesos de negocio que contienen 82 nodos de consulta. Con esto se puede constatar que es posible obtener una buena sintonía entre la calidad de las actividades recuperadas y los tiempos de respuesta necesarios para cumplir con las tareas de descubrimiento y composición, bajo unos umbrales de eficiencia como los definidos por (Joines, y otros, 2002).

Los principales aportes de esta tesis de maestría fueron:

- **Análisis detallado de los mecanismos de descubrimiento y composición de servicios**. Los principales resultados de este análisis fueron: a) Un análisis y valoración de los mecanismos de descubrimiento de servicios, apuntando a seleccionar una técnica apropiada para los ambientes ubicuos; b) Un análisis y valoración de los mecanismos de composición de servicios, orientados a escoger una técnica apropiada para los ambientes ubicuos; c) Un análisis y valoración de las representaciones formales que pueden ser empleadas para modelar las técnicas y algoritmos de descubrimiento y composición de servicios.
- **Arquitectura de referencia para la composición de servicios en un ambiente ubicuo**. Se propuso una arquitectura de referencia enfocada a proporcionar las funcionalidades de descubrimiento y composición de servicios para los ambientes ubicuos. Se consideró la inclusión de aspectos relacionados con el contexto de entrega, que aseguran el uso de servicios más pertinentes para la realización de las tareas de usuario.
- **Implementación de Referencia**. Se realizó una instanciación de la arquitectura de referencia, diseñando una implementación de referencia acotada para ambientes conformados por dispositivos móviles y un ámbito de aplicación limitado a los servicios de información turística. La implementación de referencia se llevó a un prototipo que permitió evaluar la eficiencia y eficacia de las aproximaciones propuestas para el descubrimiento y composición de servicios en los ambientes ubicuos.
- **Los algoritmos propuestos para el descubrimiento de servicios**. Se consolidó un conjunto de algoritmos y técnicas para el descubrimiento de servicios basados en una representación de grafos, y que incorpora aspectos relevantes para los ambientes ubicuos como el contexto de entrega.
- **Los algoritmos propuestos para la composición de servicios**. Se propuso un conjunto de algoritmos y técnicas para la composición de servicios soportados en una fase previa de descubrimiento. La composición se basó en una representación formal de grafos para capturar el comportamiento de los procesos de negocio, generar y seleccionar las síntesis de composición.

6.2 Trabajos Futuros

Esta tesis de maestría ha aportado soluciones al problema del descubrimiento y composición de servicios, adaptando sus algoritmos a las necesidades y restricciones que imponen los entornos ubicuos. Así, con relación al campo de estudio de este proyecto de grado se propone los siguientes trabajos futuros:

Estudio y definición de nuevas características que extiendan la descripción del usuario en un entorno ubicuo

El avance de diversos proyectos relacionados con la personalización de sistemas de información (Guerrero, y otros, 2010), (Abbar, y otros, 2008), permite explorar nuevas dimensiones en la descripción de las características de usuario, aplicadas a la búsqueda de servicios. La adición de estas características puede brindar recursos para soportar nuevas funcionalidades en sistemas de descubrimiento de servicios. Así, se propone continuar este estudio para encontrar nuevas técnicas que perfeccionen el descubrimiento de servicios en ambientes ubicuos tomando en cuenta un mayor número de elementos relativos al contexto del usuario.

Técnicas de indexación para el repositorio de procesos

La indexación es empleada en repositorios considerablemente grandes donde las búsquedas de servicios puede ser un proceso muy complejo desde el punto de vista computacional. Teniendo en cuenta que la fase de descubrimiento está basada en una representación de grafos, y considerando que diversos trabajos han propuesto técnicas de indexación para grafos (Williams, y otros, 2007) (Cheng, y otros, 2007) (He, y otros, 2006), un trabajo futuro puede ser la exploración y selección de una de estas técnicas para aplicarla a la solución de descubrimiento propuesta en esta tesis de maestría.

Evolución de la solución de composición

En el presente trabajo de maestría se abordó exclusivamente la síntesis de composición, entendida como la generación de un plan para lograr el comportamiento deseado a través de la combinación de múltiples servicios, dejando a un lado la orquestación de los servicios definida por (Küster, y otros, 2005) como la coordinación del flujo de control y datos entre varios componentes durante la ejecución del plan. En este sentido, un trabajo futuro puede estar enfocado en agregar los elementos necesarios al prototipo desarrollado en esta tesis para que se alcance la fase de ejecución de las tareas de usuario, estudiando la posibilidad de adicionar nuevos procedimientos a los algoritmos de composición, para soportar capacidades de reconfiguración dinámica de las síntesis generadas en casos donde se requiera modificar el plan en tiempo de ejecución.

Experimentación en un ambiente real

Por último se propone realizar la experimentación de la plataforma en un ambiente real, para medir el grado de satisfacción de los usuarios, y especialmente determinar su desempeño dentro de un ambiente de computación ubicua.

Bibliografía

Abbar, S., y otros. 2008. A personalized access model: concepts and services for content delivery platforms. Linz, Austria : ACM New York, NY, USA, 2008, págs. 41-47.

Almenárez, F. 2005. *Arquitectura de Seguridad para Entornos de Computación Ubicua Abiertos y Dinámicos.* 2005.

Angell, R.C., Freund, G. y Willett, P. 1983. Automatic spelling correction using a trigram similarity measure. *Information Processing and management.* 1983, Vol. 19, págs. 255–261.

Bandara, A., y otros. 2007. *A Semantic Approach for Service Matching in Pervasive Environments.* Southampton : Universidad de Southampton, 2007.

Bansal, S. S. y Vidal, J. M. 2003. Matchmaking of web services based on the DAML-S service model. *Proceedings of AAMAS.* Melbourne : s.n., 2003, págs. 926–927.

Beek, M., Bucchiarone, A. y Gnesi, S. 2006. *A Survey on Service Composition Approaches: From Industrial Standards to Formal Methods.* s.l. : In Technical Report 2006TR-15, Istituto, 2006.

Ben Mokhtar, S, y otros. 2006b. Efficient semantic service discovery in pervasive computing environments. *Proceedings of ACM/IFIP/USENIX 7th International Middleware Conference (Middleware'06).* 2006b.

Ben Mokhtar, S. 2007. *Semantic Middleware for Service-Oriented Pervasive Computing.* Paris, Francia : tesis presentada a la Universidad DEVANT L'UNIVERSIT´ E DE PARIS 6 para optar al grado de DOCTEUR DE L'UNIVERSIT´ E DE PARIS 6, 2007.

Ben Mokhtar, S., Georgantas, N. and Issarny, V. 2005. Ad hoc Composition of User Tasks in Pervasive Computing Environments. *In Proceedings of the 4th International workshop on Software Composition ETAPS'05.* Edinburgh. Scotland : Springer Berlin / Heidelberg, 2005.

—. 2006. Cocoa: Conversation-based service composition for pervasive computing environments. *Proceedings of ICPS.* Lyon, France : s.n., 2006, págs. 29-38.

Berardi, D., y otros. 2003. Automatic composition of e-services that export their behavior. *In: Proc. of 1st Int. Conf. on Service Oriented Computing (ICSOC-03).* Trento, Italia : Springer Berlin / Heidelberg, 2003.

Berardi, D., y otros. 2005. Automatic Composition of Web Services in Colombo. *Proceedings of the Thirteenth Italian Symposium on Advanced Database Systems, SEBD.* Brixen-Bressanone, Italy : s.n., 2005.

Berardi, D., y otros. 2003. *Finite state automata as conceptual model for e-services.* 2003.

Bernstein, A. y Klein, M. 2002. Discovering Services: Towards High-Precision Service Retrieval. *In Proceedings of The First International Semantic Web Conference(ISWC'02).* Toronto : s.n., 2002.

Bolognesi, T. y Brinksma, E. 1987. Introduction to the ISO Specification Language LOTOS. *Computers Networks.* 1987, págs. 25-59.

Bottaro, A., Gerodolle, A. y Lalanda, P. 2007. Pervasive Service Composition in the Home Network. *Advanced Information Networking and Applications AINA '07. 21st International Conference on.* 2007, págs. 596-603.

Brogi, A y Popescu, R. 2005. Towards Semi-automated Workflow-Based Aggregation of Web Services. *Service-Oriented Computing - ICSOC 2005.* s.l. : Springer Berlin / Heidelberg, 2005.

Brogi, Antonio y Corfini, Sara, Popescu, Razvan. 2008. Semantics-based composition-oriented discovery of Web services. *ACM Transactions on Internet Technology (TOIT).* 4. 2008, Vol. 8.

Caicedo, O.M., y otros. 2009. Discovery and Composition of Services in Ubiquitous Environments. *Cuarto Congreso Colombiano de Computación 4CCC.* 2009.

Cheng, J., y otros. 2007. FG-Index: Towards Verification-Free Query Processing on Graph Databases. *SIGMOD'07.* Beijing, China : s.n., 2007.

Constantinescu, I y Faltings, B. 2003. Eficient matchmaking and directory services. *In Proceedings of the IEEE/WIC International Conference on Web Intelligence (WI'03).* 2003.

Corrales, J.C. 2008. *Behavioral matchmaking for service retrieval.* Versailles, France : tesis presentada a la University of Versailles Saint-Quentin-en-Yvelines para optar al grado de Doctor of Philosophy in Sciences, 2008.

Corrales, J.C., Grigori, D. y Bouzeghoub, M. 2006. BPEL Processes Matchmaking for Service Discovery. *COOPIS 2006, LNCS 4275.* Montepllier, France : s.n., 2006.

Corrales, J.C., y otros. 2008. Bematch: A platform for matchmaking service behavior models. *Proceedings of EDBT.* Nantes, France : s.n., 2008, págs. 695-699.

Dasgupta, S., Bhat, S. y Lee, Y. 2009. Event driven service composition for pervasive computing. *PERCOM '09: Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications.* Washington, DC, USA : IEEE Computer Societ, 2009, págs. 1-6.

El-Sayed, Abdur-Rahman y Black, J. 2006. Semantic-based context-aware service discovery in pervasive-computing environments. *Proceedings of SIPE.* Lyon, France : s.n., 2006.

Eshuis, R., Grefen, P. y Till, S. 2006. Structured service composition. [ed.] S. Dustdar, J.L. Fiadeiro y A. Sheth. *Proc. of the 4th International Conference on Business Process Management (BPM 2006).* s.l. : Springer, 2006, Vol. 4102 of Lecture Notes in Computer Science, págs. 97-112.

Ferrara, A. 2004. Web Services: a Process Algebra Approach. *In: Proc. ICSOC'04.* s.l. : ACM Press, 2004, pp. 242-251.

Forum, Wireless Application Protocol. 2001. Wireless Application Protocol Forum. [Online] 2001. [Cited: Agosto 21, 2010.] http://www.wapforum.org/.

Gallo, G., y otros. 1993. Directed hypergraphs and applications. *Discr. Appl. Math.* 1993, Vol. 2, págs. 177–201.

Georgantas, N., y otros. 2005. Semantic-aware Services for the Mobile Computing Environment. *Architecting Dependable Systems III.* s.l. : Springer Verlag, 2005.

Goncalves da Silva, E.M., Ferreira Pires, L. y van Sinderen, M.J. 2007. An Algorithm for Automatic Service Composition. *In: 1st International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing, ICSOFT 2007.* Barcelona, España : INSTICC Press, 2007, págs. 65-74.

Guerrero, Esteban, Corrales, Juan Carlos y Ruggia, Raul. 2010. Selección de Servicios basado en Metamodelos del Perfil, Contexto y QoS. 2010.

Hackmann, G., Gill, C. y Roman, G.-C. 2007. Extending BPEL for interoperable pervasive computing. *Proceedings. Of ICPS.* 2007.

Hashemian, S.V. y Mavaddat, F. 2005. A Graph-Based Approach to Web Services Composition. *Proceedings of the The 2005 Symposium on Applications and the Internet (SAINT'05).* 2005.

He, Huahai y Singh, Ambuj K. 2006. Closure-Tree: An Index Structure for Graph Queries. *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06).* 2006.

Henricksen, K., Indulska, J. y Rakotonirainy, A. 2002. Modeling Context Information in Pervasive Computing Systems. *Pervasive Computing In First International Conference.* 2002, Vol. 2414, págs. 79-117.

Hinz, S., Schmidt, K. y Stahl, C. Transforming BPEL to Petri Nets. *In: Proc. BPM'05.* s.l. : LNCS, Springer, Vol. 3649, págs. 220-235.

Hopcroft, R., Motwani, J.E. y Ullman, J.D. 2001. *Introduction to automata theory, languages, and computation.* 2001.

Ibrahim, N. y Le Mouel, F. 2009. A Survey on Service Composition Middleware in Pervasive Environments. *International Journal of Computer Science Issues, IJCSI 2009.* 2009, Vol. 1, págs. 1-12.

Ibrahim, Noha, Frenot, Stephane y Le Mouel, Frederic. 2010. User-Excentric Service Composition in Pervasive Environments. *The 24th IEEE International Conference on Advanced Information Networking and Applications, AINA 2010.* Perth-Australia : s.n., 2010, págs. 682-689.

Issarny, V., Caporuscio, M. y Georgantas, N. 2007. A Perspective on the Future of Middleware-based Software Engineering. *FOSE '07: 2007 Future of Software Engineering.* s.l. : IEEE Computer Society, 2007.

Ivanova, E. 2006. BPEL - based Web Services Composition. *International Scientific Conference "Informatics In The Scientific Knowledge 2006".* Varna, Bulgaria : s.n., 2006, pp. 54-67.

Jaroucheh, Zakwan, Liu, Xiaodong Liu y Smith, Sally. 2009. A Perspective on Middleware-Oriented Context-Aware Pervasive Systems. *in Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference.* 2009, Vol. II, págs. 249-254.

Joines, S., Willenborg, R. y Hygh, K. 2002. Performance Analysis for Java Websites. s.l. : Addison-Wesley, ISBN-13: 978-0201844542, 2002.

Juric, M. B., y otros. 2007. *SOA Approach to Integration.* Birmingham, B27 6PA, UK : Packt Publishing Ltd, 2007. 978-1-904811-17-6.

Kalasapur, S, Kumar, M. y Shirazi, B. 2007. Dynamic Service Composition in Pervasive Computing. *IEEE Transactions on Parallel and Distributed Systems.* 2007, Vol. 18, págs. 907-918.

Kavantzas, N. y Burdett, D. 2004. *Ws choreography model overview.* s.l. : W3C - Web Services Choreography, 2004.

Klyne, Graham, y otros. 2004. Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0. [En línea] W3C, 15 de Enero de 2004. [Citado el: 5 de Septiembre de 2010.] http://www.w3.org/TR/CCPP-struct-vocab/.

Krall, A., Vitek, J. y Horspool, N. 1997. Near optimal hierarchical encoding of types. *In 11th European Conference on Object Oriented Programming (ECOOP'97).* s.l. : Springer, 1997, págs. 128–145.

Küster, U., Stern, M. y König-Ries, B. 2005. A classification of issures and approaches in service composition. *In Proceedings of the First International Workshop on Engineering Service Compositions (WESC05).* Amsterdam, Netherlands : s.n., 2005.

Lewis, D. 1992. Representation and learning in information retrieval. *Ph.D. Thesis.* University of Massachusetts : Department of Computer and Information Science, 1992.

Martínez, E. y Lespérance, Y. 2004. Web service composition as a planning task: Experiments using knowledge-based planning. *In: Proc. of the 14th Int. Conf. on Automated Planning and Scheduling (ICAPS 2004).* Whistler, BC, Canada : s.n., 2004.

Masuoka, R., Parsia, B. y Labrou, Y. 2003. Task computing the semantic web meets pervasive computing. *In 2nd International Semantic Web Conference (ISWC2003).* 2003.

Mazzara, M. y Lucchi, R. 2006. A pi-calculus based semantics for WS-BPEL. *Journal of Logic and Algebraic Programming.* 2006.

Mendling, J. y Ziemann, J. 2005. Transformation of bpel processes to epcs. *In Proc. of the 4th GI Workshop on Event-Driven Process Chains (EPK2005).* Hamburg. Germany : s.n., 2005, Vol. 167, págs. 41-53.

Miller, G. 1995. Wordnet: A lexical database for English. *Communications of the ACM.* 11. 1995, Vol. 38, pp. 39–41.

Muñoz, J. y Pelechano, V. 2006. Applying Software Factories to Pervasive Systems: A Platform Specific Framework. *8th International Conference on Enterprise Information Systems (ICEIS 2006).* Paphos (Cyprus) : s.n., 2006, págs. 23-27.

Narayanan, S. y McIlraith, S. 2002. Simulation, Verification and Automated Composition of Web Services. *WWW'02,.* s.l. : ACM Press, 2002, págs. 77-88.
Ouyang, C. y al., et. 2005. *Formal Semantics and Analysis of Control Flow in WS-BPEL.* s.l. : Technical Report BPM-05-15, BPM Center, 2005.
Papazoglou, M. P. 2003. Service-oriented computing. *Communications of the ACM.* s.l. : ACM Press, 2003.
Passani, L. 2010. [Online] Agosto 21, 2010. http://wurfl.sourceforge.net/.
Ponnekanti, S.R. y Fox, A. 2002. SWORD: A developer toolkit for web service composition. *In: Proc. of the 11th Int. WWW Conf. (WWW2002).* Honolulu, HI, USA : s.n., 2002.
Ramasamy, V. 2006. Syntactical and semantical web services discovery and composition. *In Proceedings of the cec-eee'06 conference.* 2006.
Reed, Paul. 2002. IBM. [Online] Rational Software Corporation, Septiembre 15, 2002. [Cited: Septiembre 1, 2010.] http://www.ibm.com/developerworks/rational/library/2774.html.
Reisig, W. y Rozenberg, G. 1998. *Lectures on Petri Nets I: Basic Models & II:Applications.* s.l. : LNCS. Springer, 1998. Number 1491-1492.
Rocha, R. C., Endler, M. y Siqueira, T. S. 2008. Middleware for Ubiquitous Context-Awareness. *MPAC '08: Proceedings of the 6th international workshop on Middleware for pervasive and ad-hoc computing.* 2008.
Satyanarayanan, M. 2001. Pervasive computing: vision and challenges. 2001, pp. 10-17.
Sellami, M., Tata, S. y Defude, B. 2008. Service Discovery in Ubiquitous Environments: Approaches and Requirement for Context-Awareness. *Advances in Semantics for Web services Workshop, BPM Workshops.* Milan, Italia : s.n., 2008.
Shen, Z. y Su, J. 2005. Web services discovery based on behavior signatures. *Proceedings of IEEE SCC.* Orlando, Florida : s.n., 2005, págs. 279-286.
Shiaa, M.M., Fladmark, J.O. y Thiell, B. 2008. An Incremental Graph-based Approach to Automatic Service Composition. *Proceedings of the 2008 IEEE International Conference on Services Computing.* Washington, DC, USA : IEEE Computer Society, 2008.
Space Systems Group. 1996. Standard Satellite Control Segment Reference Architecture. [Online] The Aerospace Corporation, 1996. [Cited: Septiembre 1, 2010.] http://sunset.usc.edu/research/reference_architecture/index.html.
Spendolini y J., M. 1994. *Benchmarking.* New York : AMACON, 1994. págs. 3-50.
Steller, L. y Krishnaswamy, S. 2009. Efficient Mobile Reasoning for Pervasive Discovery. *Proceedings of the 2009 ACM symposium on Applied Computing.* Honolulu, Hawaii : s.n., 2009.
Steller, L., Krishnaswamy, S. y Newmarch, J. 2006. Discovering relevant services in pervasive environments using semantics and context. *Proceedings of IWUC 2006.* Paphos, Cyprus : s.n., 2006.
Steller, Luke A, Krishnaswamy, Shonali y Gaber, Mohamed M. 2009. Cost Efficient, Adaptive Reasoning Strategies for Pervasive Service Discovery. *ICPS'09.* London, United Kingdom : s.n., 2009.
Suarez, L.J. y Rojas, L.A. 2010. *tesis: Descubrimiento de Servicios en Ambientes Ubicuos.* Popayán : Universiadad del Cauca, 2010.
Tigli, Jean-Yves, y otros. 2009. Lightweight Service Oriented Architecture for Pervasive Computing. *IJCSI International Journal of Computer Science Issues.* 2009, Vol. 4, págs. 1-9.
Tocarruncho, Sandra Patricia, Aponte Novoa, Fredy Andrés y Tocarruncho, Arturo. 2007. EXTRACCIÓN DE PERFILES BASADA EN AGRUPAMIENTO GENETICO PARA RECOMENDACIÓN DE CONTENIDO. 2007.
van Bommel, M. F. y Beck, T. J. 1999. Incremen encoding of multiple inheritance hierarchies. [aut. libro] In Proceedings of the eighth international conference on Information and knowledge management(CIKM '99). 1999, págs. 507–513.

Vanhatalo, J. 2006b. *BPEL Repository User Guide.* s.l. : IBM Corporation 2006, 2006b.
Vanhatalo, J., Koehler, J. y Leymann, F. 2006. Repository for business processes and arbitrary associated metadata. *Proceedings of the BPM Demo Session at the Fourth International Conference on Business Process Management.* Viena, Austria : s.n., 2006.
Wang, Xiaohang. 2003. The Context Gateway: A Pervasive Computing Infrastructure for Context Aware Service. *Research Report.* Singapore : School of Computing, National University of Singapore & Context -Aware Dept., Institute for Infocomm Research, 2003.
Wang, Zhenghui, y otros. 2009. A Parameter-Based Scheme for Service Composition in Pervasive Computing Environment. *2009 International Conference on Complex, Intelligent and Software Intensive Systems.* 2009, págs. 543-548.
Weiser, M. 1991. The Computer for the 21st Century. 1991, pp. 94–104.
Williams, D.W., Huan, J. y Wang, W. 2007. Graph database indexing using structured graph decomposition. *In 23rd International Conference on Data Engineering (ICDE).* 2007.
Wombacher, A., Mahleko, B. y Fankhauser, P. 2005. A grammar-based index for matching business processes. *Proceedings of ICWS.* Washington DC : s.n., 2005.
Wombacher, A., y otros. 2004. Matchmaking for business processes based on choreographies. *Proceedings of EEE.* Washington DC : s.n., 2004, págs. 359-368.
Yi, X. y Kochut, K. 2004. A CP-nets-based Design and Verication Framework for Web Services Composition. 2004, págs. 756-760.
Zhang, Mi y Hurley, Neil. 2009. *Novel Item Recommendation by User Profile Partitioning.* Milan, Italy : s.n., 2009.
Zhang, R., Arpinar, I.B. y Aleman-Meza, B. 2003. Automatic composition of semantic web services. *Proc. of the 2003 Int. Conf. on Web Services (ICWS'03).* Las Vegas, NV, USA : s.n., 2003.
Zhang, Y., Liu, B. y Wang, H. 2009. A Method of Web Service Discovery based on Semantic Message Bipartite Matching for Remote Medical System. *Journal of Theoretical and Applied Electronic Commerce Research.* 2009, Vol. 4, págs. 78-79.
Zhang, Y., y otros. 2004. Similarity Search for Web Services. *Proceedings of the 30th VLDB conference.* Toronto : s.n., 2004, Vol. 30, págs. 372–383.
Zhenghui, Wang, y otros. 2009. A Parameter-Based Scheme for Service Composition in Pervasive Computing Environment. *International Conference on Complex, Intelligent and Software Intensive Systems.* 2009, págs. 543-548.