UNIVERSIDAD NACIONAL DEL SUR

Tesis de Magíster
en Ciencias de la Computación

Composición de Servicios Web Semánticos

Ana Carolina Alonso de Armiño

Bahía Blanca Argentina

2010

Prefacio

Esta Tesis es presentada como parte de los requisitos para optar al grado académico de Magíster en Ciencias de la Computación, de la Universidad Nacional del Sur y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otras. La misma contiene los resultados obtenidos en investigaciones llevadas a cabo en el Departamento de Ciencias de la Computación de la Universidad Nacional del Comahue, bajo la dirección del Dr. Pablo Fillottrani, Profesor Adjunto del Departamento de Ciencias e Ingeniería de la Computación de la Universidad Nacional del Sur.

Lic. Ana Carolina Alonso de Armiño

UNIVERSIDAD NACIONAL DEL SUR

Secretaría General de Posgrado y Educación Continua

La presente tesis ha sido aprobada el .../.../..., mereciendo la calificación de .....(.....)

Resumen

Los servicios Web constituyen una tecnología que ha modificado las prácticas de diseño, desarrollo e implementación de aplicaciones, permitiendo la integración de diversas empresas, entornos heterogéneos y diferentes implementaciones. Además los servicios Web posibilitan el reuso de componentes y el desarrollo de nuevos servicios más complejos a través de la integración de servicios existentes. Este trabajo se enfoca en las tecnologías desarrolladas en torno a la Web semántica y los servicios Web en general, y en especial en las cuestiones relacionadas con la composición de servicios Web semánticos.

Para lograr una mejor comprensión de los conceptos, estándares y tecnologías describimos un ejemplo de aplicación para ser implementada como servicios Web, se trata una aplicación de reservación de pasajes de colectivo que hace uso de diversos servicios y ontologías para brindar al usuario la mejor opción a la hora de solicitar la reservación para viajar de una ciudad a otra, optimizando el recorrido y la coordinación de los trayectos recorridos.

Para finalizar concluimos con un capítulo destinado a analizar y comparar las propuestas surgidas en torno a la composición de servicios Web y se describen los lineamientos a seguir en trabajos futuros.

Abstract

Nowadays, there exists a great expansion of Web services that have modified the application's design, development and implementation, and allow the enterprise integration even on the heterogeneous environments and implementations.

In addition, Web services that make reuse of components and development of new services more complex possible across integration of existing services. This work focuses in the technologies developed concerning the semantic web and web services, and especially in the questions related to the web services composition.

To achieve a better comprehension of the concepts, standards and technologies we describe an example of application to be implemented as Web services. The application's objective is the travel reservation using web services and ontologies to select the best option to travel from a city to other one, optimizing the tour.

Finally we formulate our conclusion where we analyze and compare the proposals concerning the web services composition and we describe the works to do in the future.

Agradecimientos

En primer lugar quiero agradecer a mi familia, en especial a mi esposo por su tiempo y apoyo, y porque sin su ayuda hubiese sido imposible para mi terminar esta tesis, y a mis hijas porque iluminan cada día de mi vida.

También quiero agradecer a la Universidad Nacional del Comahue en su conjunto por brindarme la oportunidad y el apoyo para mi desarrollo, en especial a los docentes de la Facultad de Informática.

Finalmente, un agradecimiento muy especial al director de esta tesis, Pablo Fillottrani por su dedicación, apoyo y enseñanzas durante la realización de este trabajo.

Índice de contenido

Índice de Figuras

Índice de tablas

XX

Capítulo 1:

Introducción

En este Capítulo se presenta una descripción sobre el contexto en el cual se realiza el trabajo de investigación, las metas que se persiguen y la estructura de este trabajo.

1.1 Motivación

El advenimiento de los servicios Web y la arquitectura orientada a servicios (SOA) están provocando importantes cambios en la construcción de los sistemas y la forma en que interactúan con otros sistemas externos. Esta tecnología se encuentra en la cúspide en cuanto a compatibilidad de los componentes de software, lo cual permitirá la reducción de los costos de desarrollo de sistemas de software a la vez que permite mejorar las características y funcionalidad ofrecidas.

Los servicios Web junto con la arquitectura orientada a servicios permitirán hacer sistemas de información mas flexibles y sensibles, con mas opciones respecto a la tecnología de información, reducir costos de desarrollo y mantenimiento, integrar y reusar componentes de software, etc.

El futuro del software implica algún tipo de arquitectura orientada a servicios, en la cual el software esté empaquetado, pudiendo ser usado como un servicio interno o externo, y disponible en Internet. Los sistemas del futuro serán desarrollados a través de la conexión de estos servicios, requiriendo de esta manera menos software específico y más creatividad para hacer la conexión entre los servicios. Esta es la evolución natural de la tecnología del software y es sobre lo que se avocarán nuestros esfuerzos.

Una arquitectura orientada a servicios es esencialmente una colección de servicios que se

comunican entre si. Dicha comunicación, representada en la Figura 1.1, puede implicar el intercambio de datos simples o la coordinación de la actividad de algunos servicios. La definición de servicio es: una función bien definida que no depende del contexto o del estado de otros servicios.

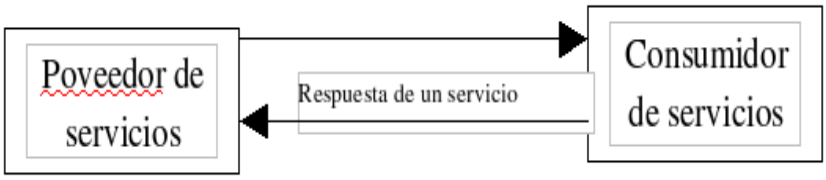

Fig.1.1 Comunicación básica entre proveedores y consumidores de servicios Web.

La distinción entre servicios internos y externos no es tan clara en una arquitectura orientada a servicios. Los servicios permiten la creación de ambientes dinámicos, donde los vendedores de software compiten usando las características e innovaciones independientemente de las conexiones existentes. Por ejemplo las interfaces de usuario, agentes de software automatizados, sistemas basados en reglas o perfiles de usuario permitirán interacciones altamente personalizadas.

Los servicios Web constituyen la tecnología de conexión del futuro. La representación de la Figura 1.2 denota la colaboración de los servicios Web sin considerar detalles de cómo estos servicios son representados y descubiertos, forma en que son invocados e interacción, lenguajes, protocolos, etc. Web Service Description Language (WSDL) forma la base de los servicios Web. Es un formato para describir la interfaz de un servicio Web. Es una forma de describir servicios y de ligarlos a una dirección de red. Una especificación WSDL tiene tres partes:

- Definición: en general está escrita en XML, incluye la definición de tipos de datos y los mensajes que usan esos tipos.

- Operaciones: describen acciones para los mensajes.

- Service bindings: conecta tipos de puerto con los puertos. Un puerto es definido asociando una dirección de red a un tipo de puerto. Una colección de puertos definen un servicio. En general el binding se hace usando XML.

La falta de descripciones semánticas ofrecidas por WSDL implica que sea imposible desarrollar clientes de software que puedan encontrar e invocar servicios en forma dinámica y sin intervención humana. Las especificaciones WSDL deben ser interpretadas por los programadores, que interpretan las palabras claves de los elementos de los mensajes usando documentación de soporte para integrar los servicios a las aplicaciones clientes. Los servicios web semánticos buscan que los clientes puedan encontrar y utilizar servicios descubiertos sin programación adicional. Los clientes pueden de esta manera interactuar con cualquier servicio que describa sus operaciones WSDL usando una representación semántica. Esto permite que los servicios sean descubiertos desde el repositorio de servicios gracias a sus descripciones semánticas sin intervención humana.

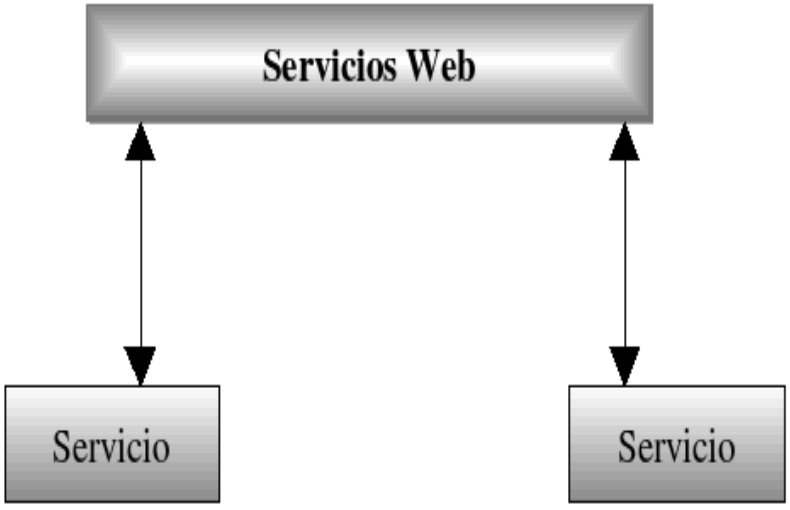

Fig. 1.2 Tecnología de servicios Web

En la representación mostrada anteriormente está implícito el directorio. El directorio en general es un repositorio UDDI (Universal Description, Discovery, and Integration) que permite descubrir servicios Web descriptos usando WSDL. La idea principal es que el registro sea usado

para encontrar información de contacto de los servicios disponibles de varias organizaciones.

Los mensajes que son intercambiados entre los servicios, tanto entre quien requiere y quien provee, como con el registro UDDI, son enviados usando SOAP (Simple Object Access Protocol). SOAP esencialmente provee el envoltorio para el envío de mensajes entre servicios Web. Contiene dos partes principales:

1. Header: son opcionales, provee información de autenticación, codificación de datos, información sobre como debe ser procesado el mensaje, etc.

2. Body: el cuerpo contiene el mensaje, puede ser definido usando WSDL.

Organizaciones de cualquier envergadura pueden beneficiarse del uso de los servicios Web. El impacto que puede causar el uso de servicios Web puede ser de cualquier nivel. En un primer nivel podemos hablar del uso de los servicios Web para crear un sitio Web o un portal. Desde un nivel más complejo podemos decir que los servicios Web pueden formar la base para software de negocios (transacciones business-to-business). A la vez, el uso de los servicios Web introducirá formas revolucionarias de usar Internet para usos tanto comerciales como personales.

Los servicios Web y la arquitectura orientada a servicios, pueden ser vistos como un proceso en evolución. Los cambios en los sistemas han comenzado y evolucionarán a medida que más gente gane experiencia y conocimiento y se desarrollen los estándares necesarios.

Uno de los principales objetivos es el estudio de las nuevas tecnologías, específicamente la tecnología de servicios Web. Como se ha mencionado, la evolución de los sistemas de software usando dicha tecnología permite el desarrollo de paquetes de software más reusables, reduciendo los costos a la vez que reduce el tiempo de desarrollo. La adopción de cada nueva tecnología implica la dedicación y predisposición al cambio por parte de quien debe incorporarlas. La adaptación de los sistemas implica que se reformulen las prácticas antiguas de desarrollo y cambien la forma de concepción de dichos sistemas. También implica que los desarrolladores y organizaciones se adapten y compitan en el desarrollo y uso de los servicios Web.

La visión de un mundo donde los servicios estén integrados, donde usuarios y proveedores interactúan a través de esos servicios, donde la tecnología es implementada según los marcos usados globalmente y usando estándares interoperables, será posible de alcanzar en unos pocos años a través de la evolución de las tecnologías.

Esta investigación está centrada en la evaluación de las distintas tecnologías relacionadas con la información y lacomunicación, más específicamente con la implementación de los servicios Web, y las diferentes arquitecturas y estándares relacionados con ella. Dada la gran cantidad de lenguajes que han surgido en torno a la Web semántica y los servicios Web, en esta tesis se ha detectado la necesidad de realizar un estudio y clasificación de los mismos. El análisis realizado es de especial interés para los investigadores que se inician en el tema ya que les permite tener una visión clara desde donde partir. Además para lograr una mejor comprensión de los conceptos, estándares y tecnologías se describe una aplicación de ejemplo. Todo esto junto con el capitulo 6, donde se analizan y comparan las propuestas originadas en torno a la composición de servicios Web, son un gran aporte para los investigadores para que puedan realizar su propio mapa conceptual, que de otra manera implicaría un gran esfuerzo la interpretación, comparación y organización de toda la información disponible.

1.2 Conceptos Básicos

1.2.1 Servicio y SOC

Un **servicio** es una pieza de software que interactúa con los clientes (humanos u otros servicios) para realizar una tarea. Cuando un servicio es ejecutado realiza ciertas acciones y delega otras interactuando con otros servicios. Esto constituye el paradigma conocido como **Computación Orientada a Servicios** (**SOC**) cuyos conceptos básicos son las aplicaciones distribuidas y el reuso de servicios existentes .

El *esquema* de un servicio está constituido por sus características funcionales que representan el *que* del servicio y no funcionales tales como calidad, seguridad y rendimiento.

La *implementación* del servicio se enfoca en cómo se logra brindar el servicio, es decir, la funcionalidad.

Una *instancia* de servicio es una ocurrencia del servicio ejecutándose e interactuando con un cliente. Para un mismo esquema pueden existir mas de una instancia, cada una se ejecuta independientemente de las otras.

Para ejecutar un servicio un cliente debe activar una instancia, para lo cual debe realizar una acción y esperar la respuesta desde la instancia para seguir interactuando hasta que la tarea sea completada.

Características de los servicios

- Están disponibles en un endpoint particular en la red, reciben y envían mensajes y exhiben su comportamiento de acuerdo a su especificación. Proveen una funcionalidad especifica que es desarrollada de acuerdo con la calidad apropiada.

- Los aspectos funcionales son especificados usando WSDL, las restricciones y condiciones asociadas a un servicio son especificadas a través de políticas que se pueden asociar a partes del WSDL.

Las interfaces y las políticas describen los términos y condiciones que gobiernan el uso de un servicio.

1.2.2 XML

XML es un metalenguaje definido usando Unicode e independiente de la plataforma.

- Elementos: construcción con nombre que tiene una serie de atributos e hijos, los hijos pueden ser otros elementos, texto, comentarios, etc. Se escriben usando <>.
  Ej: <foo> .. </foo>
- Atributos: son pares de nombre-valor asociados a los elementos.
  Ej: <foo name1=”value1” name2=”value2”…>

- Comentarios: se encierran entre <!-- y -->
- Texto literal: son caracteres unicote contenidos en un elemento.
- Documento: un documento XML es una unidad de empaquetamiento XML que consiste exactamente de un elemento (el elemento documento).

Muchos de los lenguajes que tienen como objetivo la descripción de la estructura de los documentos se basan en XML. Mediante etiquetas determinan cuales son los elementos que definen la estructura del documento, si hay una jerarquía entre ellos, en que orden pueden aparecer los elementos, cuales son sus propiedades, tipos, valores, etc.

GRDDL (**G**leaning **R**esource **D**escriptions from **D**ialects of **L**anguages) es una técnica para obtener datos RDF desde documentos XML y páginas XHTML. Se pueden asociar explícitamente los documentos con los algoritmos de transformación, que en general están representados en XSLT, usando los elementos link y head del documento. Alternativamente la información necesaria para obtener la transformación puede estar en un documento de metadatos (profile) [28].

POWDER (Protocol for Web Description Resources) es un mecanismo para proveer significado a individuos u organizaciones para describir un grupo de recursos a través de la publicación de metadatos entendibles por las máquinas. Un Description Resources encapsula tales metadatos, los cuales son representados generalmente en XML siendo relativamente entendibles por el humano. El significado de tales DRs se apoya en una semántica formal, accesible realizando la transformación GRDDL[29].

1.2.3 DTD

Mediante etiquetas XML especifica restricciones en la estructura y sintaxis de un documento.

Permite especificar qué etiquetas son permitidas y el contenido de dichas etiquetas, el orden en que pueden aparecer dentro de un documento y qué etiquetas van dentro de otras.

Tiene algunas limitaciones como no permitir definir elementos locales que sólo sean

válidos dentro de otros elementos. Es poco flexible la definición de elementos con contenido mixto y no es posible indicar a qué tipo de dato corresponde un atributo o el texto de un elemento.

1.2.4 XML Esquema

Este lenguaje pretende aumentar la potencia expresiva que provee el DTD. El principal aporte de XML Schema es el gran número de los tipos de datos que incorpora incluyendo tipos de datos complejos como fechas, números y strings.

Define los Elementos y atributos que pueden aparecer en un documento, cuales elementos son hijos de otros elementos, el número de hijos y el orden de los elementos. Si un elemento es vacío o puede incluir texto, los tipos de datos de los elementos y de los atributos y los valores por defecto de ambos [15].

1.2.5 RDF (Resource Description Framework)

La idea sobre la que se basa RDF es la posibilidad de identificar cosas usando identificadores Web (Uniform Resource Identifiers, URIs[1]), y describir los recursos en términos de propiedades y el valor de estas propiedades. Una descripción RDF se define como una tripleta (sujeto, predicado, objeto), donde el sujeto representa el recurso (por ejemplo http://www.w3c.es/Personal/Martin), el predicado representa una propiedad (por ejemplo *creado*) y el objeto es un literal (por ejemplo Martin Alvarez): <sujeto> tiene <predicado><objeto>

Soporta tipos de datos literales: decimal, entero, binario, string y es posible representar el tipo de los recursos y de las propiedades. Además permite crear tipos y propiedades para representar grupos de recursos (Contenedor, Colección y Reification). También permite representar relaciones binarias y relaciones de mayor aridad [3].

---

[1] URI:
La Web es un espacio de información donde las URIs son ítems en este espacio, esta es la única tecnología para nombrar/direccionar en la Web. Uniform Resource Identifiers (URIs, aka URLs) son pequeños strings que identifican recursos en la Weby posibilita el acceso a los mismos dentro de una variedad de esquemas de nombres y métodos de acceso tales como HTTP, FTP e Internet mail addressable.

SPARQL es un lenguaje para el acceso a la información RDF. Algunas características de este lenguajes son: limita el número de resultados, elimina duplicados, los ordena, etc.; especifica múltiples fuentes de datos (vía URIs) en una misma consulta, usando conjunción y disyunción. SPARQL es una "recomendación" de la W3C [30].

EaRL (Evaluation and Report Language) es un vocabulario usado para describir resultados de pruebas. Este lenguaje se basa en RDF, siendo una colección de sentencias sobre recursos. Cada sentencia tiene un sujeto, un predicado y un objeto. Desde ellas se puede conocer: **Quien** ejecuta la prueba, conocido como Assertor; el **recurso** que está siendo evaluado; el **criterio** que se usa para hacer la evaluación; y los **resultados** de realizar las pruebas.

EaRL tiene una naturaleza semántica que busca facilitar la extracción y comparación de los resultados de las pruebas hechas por personas y por herramientas. EaRL permite la creación de una forma estandarizada de producir reportes de pruebas, intercambiar los reportes hechos por diferentes evaluadores, comparar los resultados, etc. [31].

1.2.6 Arquitectura Orientada a Servicios (SOA)

La tecnología de Servicios Web está basada en estandares y centrada en XML y es una realización de un SOA. La arquitectura de servicios distribuidos SOA se caracteriza por las siguientes propiedades:

- Vista lógica: el servicio es abstracto, la vista lógica de los programas actuales, bases de datos, procesos del negocio, etc son definidos en términos de qué hace el servicio, en general llevando a cabo una operación a nivel del negocio.

- Orientado a Mensajes: los servicios son formalmente definidos en término de los mensajes intercambiados entre los agentes proveedores y los agentes que requieren. La estructura interna de los agentes, incluyendo sus características como el lenguajes en que son implementados, la estructura de los procesos y la estructura de la base de datos son abstraídos en SOA: usando SOA no hace falta conocer como un agente implementa un

servicio.

- Orientado a la descripción: un servicio es descrito por metadatos procesables por las máquinas. Tal descripción soporta la naturaleza publica de SOA: solo aquellos detalles que son públicos e importantes para el uso del servicio deberían ser incluidos en la descripción. La semántica del servicio debería ser documentada, directa o indirectamente por tal descripción.

- Granularidad: los servicios tienden a usar un pequeño número de operaciones con una cantidad de mensajes relativamente grandes y complejos.

- Orientado a la Red: los servicios tienden a ser usados en una red.

- Independiente de la plataforma: los mensajes son enviados independientemente de la plataforma, con un formato estandarizado a través de las interfaces, generalmente XML.

El escenario donde funcionan los servicios Web está constituido por tres componentes de software que interactúan: proveedor de servicios, registro y el cliente. El proveedor es un servidor que hospeda servicios y que provee su interfaz en el nivel de aplicación sobre un protocolo de transporte. El proveedor es quien se encarga de la publicación del servicio y del intercambio de mensajes con el cliente que solicita el servicio. El registro es básicamente una aplicación que guarda la información necesaria para identificar los servicios. El registro es fundamental para la búsqueda y descubrimiento de servicios. El cliente es la aplicación que solicita un servicio concreto, conectándose con el proveedor del mismo para luego operar con el servicio.

En este escenario hay algunas etapas que deben desarrollarse secuencialmente para hacer posible el uso de un servicio: descripción y anotación semántica de servicios, publicación y registración, descubrimiento, invocación, composición y ejecución.

1.2.6.1 Descripción y anotación semántica de los servicios Web

Para comprender cómo se debe interactuar con un servicio Web determinado, es necesario

proporcionar una descripción que defina las interacciones que admite el servicio Web. La descripción de un servicio está contenida en un documento XML. El lenguaje mas comúnmente usado para realizarla es WSDL, el cual establece el formato de los mensajes que se admiten de manera que el cliente sepa cómo interactuar con el.

WSDL no provee una forma uniforme de interpretar los datos de entrada y de salida de las operaciones, por lo que el descubrimiento e invocación de los servicios en forma automática no es posible. El uso de ontologías permite que un servicio Web sea anotado con información semántica. Esto consiste en asociar conceptos y relaciones de una ontología con los parámetros y las operaciones del servicio, haciendo posible automatizar las tareas ligadas con el uso de los servicios.

1.2.6.2 Publicación y registración de servicios Web

El registro es una aplicación que mantiene información sobre los servicios, haciendo posible identificar y descubrir servicios.

El registro UDDI (Universal Description, Discovery, and Integration) provee una forma estandarizada para publicar y descubrir información sobre los servicios Web. UDDI fue propuesto originalmente como un estándar para servicios Web, puede ser consultado a través de mensajes SOAP y provee la descripción de los servicios en WSDL, brindando los protocolos y formatos de mensajes requeridos para interactuar con los servicios.

El repositorio UDDI está lógicamente centralizado pero físicamente distribuido: cada socio del consorcio es dueño de un registro UDDI pero la información de cada registro UDDI está replicada en los otros. Así, un proveedor de servicios puede publicar un servicio en un registro de cualquiera de los socios y luego esa información será replicada en cada uno de los registros de los demás socios, siendo posible que un usuario invoque un servicio desde cualquiera de los registros.

Los tres elementos principales del registro UDDI son: las páginas blancas, que contienen información sobre el proveedor de servicios; las páginas amarillas, que contienen una clasificación

de los servicios; y las páginas verdes que proveen información técnica sobre los servicios publicados.

Un cliente que quiera usar un servicio debe realizar la búsqueda usando el nombre del servicio y si lo encuentra obtendrá su descripción y un link a la página del proveedor de servicios, de esta forma el cliente puede invocar el servicio

UDDI soporta la búsqueda por nombre de servicios, pero si el cliente no conoce el nombre y conoce su funcionalidad no será capaz de descubrirlo ni invocarlo. UDDI no soporta la búsqueda semántica de servicios y esto hace que su fucionalidad sea limitada.

1.2.6.3 Descubrimiento de servicios Web

El descubrimiento incluye la localización de documentos que describen servicios específicos. En algunas especificaciones si los clientes conocen la ubicación de la descripción del servicio pueden omitir el descubrimiento, directamente podrían invocarlo.

En una LAN puede ser hecho a través de una base de datos de servicios tal como lo hace el sistema **DySCO** [124]. En una red mayor se requiere un sistema de búsqueda jerárquico para lograr escalabilidad, tal como lo hace el sistema **eFlow** [87]. Si no se encuentra un servicio que realice la tarea el servicio **Broker** preguntará a otros broker si conocen el servicio, reenviando el requerimiento.

El descubrimiento puede ser realizado en forma centralizada o descentralizada. La pila de red de los sistemas operativos provee servicios de anuncios, descubrimiento de servicios, etc. Un dispositivo UPNP usa una versión multicast de HTTP para preguntar a todos los dispositivos de la red si tienen habilitado UPNP, las respuestas son unicast UDP. El descubrimiento de servicios es **descentralizado** por lo que no sirve para grandes redes. **JINI** [125] es un sistema basado en JAVA, donde el descubrimiento de servicios se hace a través de un servicio de búsqueda que usa una base de datos **centralizada**. A través de unicast se hace la consulta de un servicio, si se conoce

el servicio de búsqueda se contacta con él brindando el nombre de host y el número de puerto.

Brokering es el proceso de seleccionar cual servicio debería ser elegido, para lo cual se debería tener alguna descripción de los servicios seleccionados y usar ciertos criterios para tomar la decisión: tiempo que requiere el servicio para completar su ejecución, rol que puede cumplir el servicio dentro de la composición y precio que hay que pagar al proveedor.

En [84] se proponen algoritmos para descubrir servicios especificados en WSDL, tales algoritmos se basan en encontrar similitudes, para lo cual se consideran las operaciones y los parámetros de entrada y de salida de las mismas.

En [85] se presenta un algoritmo de descubrimiento de servicios basado en tareas y usando la similaridad semántica de los servicios para finalmente realizar la composición de servicios. En este trabajo se define una tarea como una función o proceso del negocio que puede ser realizada por uno o por un grupo de servicios juntos.

1.2.6.4 Invocación de servicios Web

Para habilitar la comunicación entre servicios se utilizan protocolos que puede comprender cualquier sistema compatible con los estándares Web más utilizados. SOAP es el protocolo principal para la comunicación de servicios Web, en este caso la invocación a un servicio será a través del envío de un documento XML según lo requiera la descripción del servicio, en general en WSDL, que define el contrato entre el servicio y quien requiere del mismo al definir las interacciones necesarias determinadas por la secuencia de mensajes que deben ser intercambiados.

1.2.6.5 Composición de servicios Web

La composición de servicios requiere que se realicen algunas tareas en forma manual, lo cual no es adecuado para la composición dinámica de los mismos. Para superar este obstáculo han surgido algunas técnicas adaptivas que usan e-learning, información de perfil del usuario, etc. para lograr formas de composición mas dinámica y flexible.

Para lograr una composición de servicios adecuada según los requerimientos del usuario se requiere modelar el mundo real en el cual el usuario expresa sus necesidades. Las ontologías proveen una forma de capturar e intercambiar modelos del mundo real que los agentes automatizados utilizan para realizar la composición.

En [85] se presenta una forma de componer servicios en forma flexible gracias al descubrimiento semántico de los mismos. En esta propuesta se realiza un grafo que representa la composición y a través de una búsqueda que maximiza las cualidades de QoS se determina la composición final.

1.2.6.6 Ejecución de servicios Web

La ejecución de servicios Web requiere de una plataforma donde puedan correr y responder a las solicitudes de los clientes. Se debe asegurar:

- Atomicidad: todas las acciones en una transacción deben ser completadas exitosamente o ninguna de ellas debe ser completada.
- Consistencia: si una transacción es interrumpida o si es terminada, el estado del sistema debe permanecer consistente con lo ocurrido.
- Aislamiento: mientras una transacción es ejecutada, los cambios no deben ser visibles desde otras transacciones.
- Durabilidad: los resultados obtenidos tras una transacción deben ser durables.

Hay varias propuestas :

BPWS4J (IBM Business Process Execution Language for Web Services JavaTM Run Time), incluye una plataforma donde pueden ejecutarse procesos de negocios escritos en BPEL4WS además de una herramienta de validación de documentos BPEL4WS.

Active EndPoints ActiveBPEL es un entorno de ejecución capaz de ejecutar procesos cuyas definiciones están en BPEL.

Apache ODE (Orchestration Director Engine) ejecuta procesos de negocios escritos en WS-BPEL.

Intalio BPMS soporta procesos escritos en BPEL y se basa en otros motores como Apache Geronimo y Apache ODE.

1.3 Objetivos

El objetivo principal de este trabajo es analizar los problemas y posibles soluciones de la composición automática de los servicios Web, para lo cual se estudiarán las diferentes propuestas y la evolución de las mismas y se realizará una comparación de dichas alternativas.

Otras metas implicadas son:

1. Estudiar las particularidades de los servicios Web semánticos.
2. Estudiar las propuestas existentes para realizar la integración automática de servicios Web.
3. Analizar los problemas existentes en las propuestas de integración y sus posibles soluciones.
4. Comparar las alternativas previamente estudadas y realizar alguna clasificación.

1.4 Contribuciones de esta Tesis

Los contenidos estudiados durante el desarrollo de esta tesis han permitido realizar diferentes producciones que han sido divulgadas en varias oportunidades:

- "Una plataforma de Servicios Web". Ana Alonso de Armiño, Pablo Rubén Fillotrani. Publicado en XI Workshop De Investigadores De Ciencias De La Computación. San Juan. 2009
- "Incorporación de Semántica en plataformas para e-learning". Lidia Marina López. Ana

Alonso de Armiño. Presentado en Jornadas de difusión científica de la Facultad de Economía y Administración. Neuquén. 2009

- "Incorporación de Semántica en plataformas para e-learning". Lidia Marina López. Ana Alonso de Armiño. Publicado en III Congreso de Tecnología en Educación y Educación en Tecnología. Bahía Blanca. 2008

- "Clasificación de los Lenguajes definidos en torno a Servicios Web y Web Semántica". Ana Alonso de Armiño, Pablo Rubén Fillotrani. Publicado en X Workshop De Investigadores De Ciencias De La Computación. General Pico 2008.

1.5 Estructura

La estructura de la tesis esta organizada de la siguiente manera:

En el Capítulo 2 se describen las principales características de las tecnologías de la Web Semántica y Servicios Web. También se definen los servicios Web como una implementación de la arquitectura SOA y se describen los lenguajes que han surgido en torno de ellos. El Capítulo se centra en la integración de ambas tecnologías en lo que se conoce como Servicios Web Semánticos, para lo cual se analiza el concepto de ontología y los lenguajes surgidos para crear ontologías.

En el Capítulo 3 se estudian los conceptos fundamentales de la composición de servicios, tales como coreografía y orquestación, se analizan los problemas y cuestiones actualmente en estudio y también se analizan las dos técnicas principales de composición: basadas en workflow y basadas en planning. Al final del Capítulo se presentan los estándares y propuestas surgidas en torno al tema de composición y se resume el estado del arte.

El Capítulo 4 se centra en el estudio de las cuestiones y los problemas relacionados con la composición automática de servicios Web. También se analizan las diferentes técnicas para realizar la composición usando Máquinas de Estado Finito, Planning y Workflow.

En el Capítulo 5 se describen las principales características que deben tener las plataformas para composición, ejecución y orquestación de los servicios Web. También se describe la tecnología de agentes, las plataformas de implementación y ejecución de agentes, y finalmente la relación de estas tecnologías con la implementación de los servicios Web. Se analiza un ejemplo y la planificación para su futura implementación.

En el capítulo 6 se realiza la comparación de las tecnologías y propuestas para realizar el diseño, ejecución y adaptación de lo servicios Web compuestos, y se presentan las conclusiones y trabajos futuros.

Capítulo 2:

Web Semántica y Servicios Web

En este Capítulo se describen las principales características de las tecnologías de la Web Semántica y Servicios Web. También se definen los servicios Web como una implementación de la arquitectura SOA y se describen los lenguajes que han surgido en torno de ellos (los cuales han sido clasificados en el trabajo presentado en WICC 2007).

El Capítulo se centra en la integración de ambas tecnologías en lo que se conoce como Servicios Web Semánticos, para lo cual se analiza el concepto de ontología y los lenguajes surgidos para crear ontologías.

2.1 Web Semántica

La Web ha tenido un crecimiento exponencial que nos coloca ante una enorme cantidad de datos. Allí anidan problemas bien conocidos como la dificultad para realizar consultas semánticas, elevados costos de mantenimiento de los sitios Web, contenidos dependientes del lenguaje, alta dependencia del factor humano para depurar la información buscada, etc.

Es así que surge la necesidad de construir una Web Semántica (WS) [27] concebida como "una extensión de la actual (Web), en la cual la información está dada por significados bien definidos, mejorando la relación entre las computadoras y los humanos para su trabajo cooperativo" [25], que debe permitir que sea usada para un "descubrimiento más efectivo, automatización, integración y reutilización entre varias aplicaciones"[26].

Los mecanismos de representación del conocimiento, las ontologías y los agentes inteligentes, son las tecnologías requeridas en la construcción de esta nueva Web. El consorcio de

la W3C, realiza la tarea de estandarización de estas tecnologías e impulsa proyectos que tienden a masificar su uso.

La visión propuesta, una Web con semántica, plantea el reto de garantizar que las máquinas "comprendan" e interactúen en un medio certificado bajo estrictos niveles de confianza. Las tecnologías de la Web semántica permiten la creación de repositorios de datos en la Web, la construcción de vocabularios y la escritura de reglas para manejar esos datos, así los datos son relacionados gracias a tecnologías como RDF, SPARQL, OWL, y SKOS.

En este contexto se define el término inferencia como el descubrimiento de nuevas relaciones [73], para ello los datos deben ser modelados como un conjunto de relaciones entre recursos. La inferencia permite generar nuevas relaciones entre los recursos a través de procedimientos automáticos basados en los datos y otra información adicional como un conjunto de reglas.

Los vocabularios o conjuntos de reglas son el recurso que posibilita la extracción de información adicional. Ambos enfoques requieren técnicas de representación del conocimiento aunque se diferencian en:

- Las Ontologías se basan en métodos de clasificación para lo cual definen clases y subclases. También definen la forma en que los recursos se relacionan con dichas clases y las características que relacionan las clases y sus instancias. Los estándares o recomendaciones de la W3C son RDF, OWL y SKOS.

- Las Reglas se basan en definir mecanismos para descubrir y generar nuevas relaciones desde las ya existentes, similar a los programas lógicos como los de Prolog. El estándar recomendado por la W3C es RIF.

2.2 Servicios Web

Los servicios Web han nacido en el 2000 con la primera versión de XML messaging-

SOAP, WSDL 1.1, y una versión inicial de UDDI como un registro de servicios. Este conjunto básico de estándares ha provisto las bases para lograr la interoperabilidad de los componentes (servicios Web) que es independiente de su ubicación en la red, y la base para detalles de implementación específicos del servicio y su infraestructurade soporte.

El valor de esta tecnología ha sido demostrada en la práctica ya que permite resolver algunos problemas importantes. Los desarrolladores destacan el alcance de la interoperabilidad lograda a traves del intercambio de mensajes, requiriendo una infraestructura de alto nivel que lo soporte. Las aplicaciones actualmente son construidas asumiendo un modelo de programación específico. Son desarrolladas en plataformas (sistemas operativos o middleware) que proveen una infraestructura de servicios como soporte de un modelo de programación, que oculta la complejidad, y simplifica los problemas que debe encarar el desarrollador. Por ejemplo, un middleware tipicamente soporta transacciones, seguridad, intercambio de mensajes confiable (se entrega una y solo una vez). Por otro lado, no hay middleware estandarizados, lo que hace difícil la construccion de aplicaciones desde componentes que usan diferentes modelos de programación (Microsoft COM, OMG CORBA, Java 2 Platform, Enterprise Edition –J2EE, Enterprise Java Beans). Las diferentes suposiciones en cuanto a la infraestrctura de servicios, tal como transacciones y seguridad, hacen que la interoperabilidad entre plataforma heterogeneas sea difícil.

La comunidad de los servicios Web ha trabajado sobre este aspecto y se han hecho algunas especificaciones relacionadas con los servicios Web. En la Figura 2.1 se representa en forma general un ejemplo de interacción entre un proveedor y un consumidor de servicios. Varios estándares han sido definidos para que esto sea posible. En primer lugar SOAP (Protocolo Simple de Acceso a Objetos) es un protocolo que permite la interacción entre varios dispositivos y que tiene la capacidad de transmitir información compleja. Los mensajes SOAP especifican el protocolo de transmisión de datos usado para transmitir los datos, por ejemplo HTTP o SMTP. En segundo lugar WSDL (Lenguaje de Descripción de Servicios Web) es un estándar que permite que un servicio y un cliente establezcan un acuerdo sobre la forma de transporte y contenido de los mensajes, describiendo la sintaxis y los mecanismos de intercambio de mensajes. Este contrato

entre el proveedor y el cliente (que solicita el servicio) se especifica en un documento procesable por dispositivos. WSDL representa una especie de contrato entre el proveedor y el que solicita.

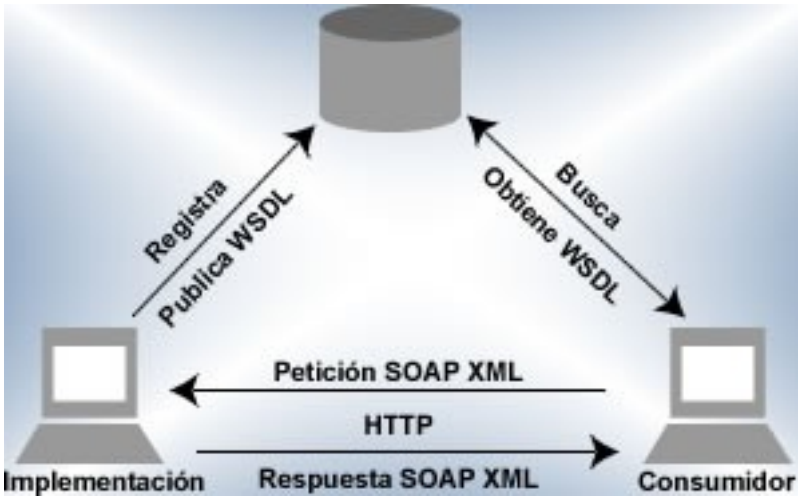

Fig. 2.1. Interacción entre un proveedor y un consumidor de servicios

Se han ido desarrollando otros mecanismos para potenciar a los servicios Web, tales como las anotaciones semánticas que permite describir y encontrar los Servicios Web que mejor se adapten a las necesidades y mecanismos para permitir la composición de varios Servicios Web individuales, lo que se conoce como coreografía.

En [75] se presenta una arquitectura de servicios Web. Se define servicio Web como un sistema de software que soporta la interacción entre máquinas sobre una red, con una interfaz descrita en un formato procesable por las máquinas (en WSDL). Otros sistemas interactúan con el servicio, según la forma prescrita en la descripción del servicio, usando mensajes SOAP típicamente sobre HTTP con una serialización XML. En esta arquitectura se definen los siguientes elementos:

- Agentes: un servicio es una noción abstracta que debe ser implementado por un agente concreto, el cual es una pieza de software o hardware que envía y recibe mensajes. El agente que implementa el servicio puede cambiar mientras el servicio Web debe permanecer intacto.

- Solicitantes y proveedores: el objetivo de un servicio es proveer cierta funcionalidad, el

proveedor es una persona o entidad que provee el agente que implementa un servicio particular. La entidad que solicita es una persona o entidad que hace uso de un servicio Web. El agente que solicita intercambia mensajes con el agente del proveedor. Para que este intercambio sea exitoso debe realizarse un acuerdo tanto semántico como del intercambio de mensajes a realizar.

- Descripción del servicio: los mecanismos de intercambio de mensajes son documentados en una descripción del servicio Web (WSD), lo cual es una especificación procesable por las máquinas de la interfaz de los servicios Web y está escrita en WSDL. Esta descripción define el formato de los mensajes, tipos de datos, protocolos de transporte y serialización. También se especifican los puntos en la red en los cuales el agente proveedor puede ser invocado así como información sobre el patrón de intercambio de mensajes esperado. Esta descripción representa el acuerdo que gobierna el mecanismo de interacción con el servicio.

- Semántica: la semántica de un servicio Web es el comportamiento que se espera que un servicio tenga en respuesta a los mensajes que recibe. Es un contrato informal entre la entidad que requiere y la entidad que provee dilucidando el propósito y las consecuencias de la interacción.

- Ejecución de un servicio Web: los pasos básicos son 1) las entidades solicitante y proveedor se conocen; 2) las entidades acuerdan sobre la descripción y semántica del servicio que gobierna la interacción que tendrán los agentes solicitante y proveedor; 3) la descripción y semántica del servicio son enviadas a los agentes; y 4) ambos agentes intercambian mensajes.

2.2.1 Servicios Web como una realización de SOA

La Arquitectura Orientada a Servicios (SOA) representa un concepto arquitectural abstracto. Es un enfoque para construir sistemas de software basados en componentes con poco

acoplamiento (servicios) que han sido descriptos de una forma uniforme y que pueden ser descubiertos y compuestos. La Figura 2.2 muestra una representación de esta arquitectura donde se identifican los elementos que la integran [126]:

1. El *transporte*, el cual constituye el mecanismo para llevar solicitudes y respuestas entre el consumidor el proveedor de servicios.

2. El *protocolo de comunicación*, el cual representa un acuerdo para llevar a cabo la comunicación entre proveedores y consumidores de servicios.

3. La *descripción de los servicios*, lo cual incluye información sobre el *qué* del servicio, *cómo* debe invocarse, y *qué datos* requiere el servicio para poder ser invocardo.

4. Los *servicios* disponibles.

5. Los *procesos de negocio*, integrados por una colección de servicios que al ser invocados en una secuencia particular y con un conjunto específico de reglas, pueden satisfacer un requisito de negocio.

6. El *registro de servicios*, es un repositorio de descripciones de servicios donde los proveedores pueden publicar los servicios que brinda y los consumidores de servicios hallar servicios disponibles que necesitan.

7. Las *políticas*: integran un conjunto de reglas bajo las cuales un proveedor puede ofrecer un servicio.

8. *Seguridad*: se definen reglas para acceder en forma segura a los servicios, éstas incluyen reglas para la identificación, autorización y control de acceso.

9. Las *transacciones*: es el conjunto de atributos que podrían aplicarse a un grupo de servicios para que el resultado sea consistente (por ejemplo si se usan tres servicios para realizar una función todos deben completarse o bien ninguno de ellos).

10. *Administración*: se define un conjunto de atributos que podrían aplicarse para manejar los servicios ofrecidos.

Los 6 primeros elementos se relacionan con las aspectos funcionales de la arquitectura y los 4 restantes con aspectos de la calidad del servicio ofrecido.

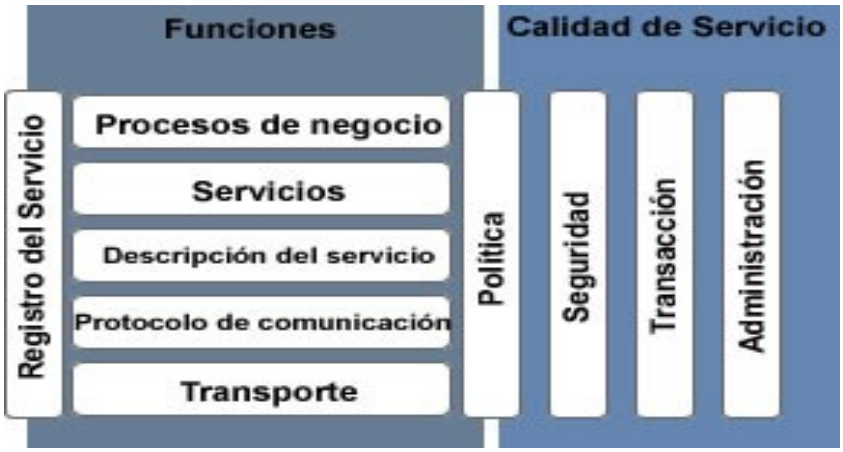

Fig. 2.2. Elementos que integran la arquitectura SOA.

Los servicios Web representan un enfoque para realizar un SOA. La definición de la W3C de Servicios Web es : U*n sistema de software diseñado para soportar la interacción de las máquinas sobre una red. Que tiene una interfaz descripta en un formato procesable por las máquinas (WSDL). Otros sistemas interactúan con el servicio Web en una forma prescrita por esa descripción usando mensajes SOAP, típicamente usando HTTP y XML entre otros estándares.*

La tecnología de servicios Web no es el unico enfoque para implementar SOA pero es el que la industria de la IT ha potenciado. Con los servicios Web se ha apuntado a una cuestion fundamental en lo que es computación distribuida: proveer una forma uniforme para descrbir componentes o servicios dentro de una red, encontrarlos y accederlos. La diferencia entre los servicios Web y el enfoque tradicional (Tecnología de objetos distribuida como Object Management Group – Common Object Request Broker Architecture –OMG CORBA o Microsoft Distributed Component Object Model –DCOM) recae en el aspecto de bajo acoplamiento de esta arquitectura. En vez de construir aplicaciones que resultan en una colección de objetos o componentes, los cuales son conocidos y entendidos en tiempo de desarrollo, el enfoque total es

mucho mas dinámico y adaptable a cambios. Además, la industria de las IT apoya el uso de la tecnología y las especificaciones de una forma abierta, con el respaldo de la W3C y la Organization for the Advancement of Structured Information Standards (OASIS) y basados en estándares y tecnología que son las bases de Internet.

Los servicios Web son básicamente una arquitectura de mensajes interoperables, y la tecnología de transporte es la base para esta. Los mensajes de los servicios Web pueden ser transportados usando los protocolos de la Web como HyperText Transport Protocol (HTTP) o Secure HTTP (HTTPS) o cualquier otro protocolo de comunicación, incluso propietarios si fuera necesario. Los protocolos son fundamentales para los servicios Web y son un factor que define la interoperabilidad.

2.2.1.1 Messaging Services

El componente mensajería de servicios incluye las especificaciones y tecnologías mas importantes, entre ellas eXtensible Markup Language (XML), SOAP, yWS-Addressing. Estas tecnologías en conjunto forman la base para la mensajería de los servicios Web. XML provee un formato interoperable para describir el contenido de los mensajes entre los servicios Web y es el lenguaje básico en el que se definen las especificaciones de los servicios Web.

2.2.1.1.1 SOAP

SOAP provee un mecanismo relativamente liviano para el intercambio de información estructurada y tipada entre servicios Web. Fue diseñado para reducir el costo y complejidad de la integración de aplicaciones que son construidas sobre diferentes plataformas.

SOAP es un mecanismo que estructura el intercambio de mensajes entre servicios Web. Un mensaje SOAP es un documento XML que contiene 3 elementos distintos: envelope, header y body. Una representación [127] puede observase en la Figura 2.3. El *envelope* es el elemento raíz del mensaje. El *header* permite agregar otras características a SOAP. Cada elemento hijo del header es llamado *header block*. SOAP define atributos para indicar como debe ser tratado un

header block y si su procesamiento es opcional u obligatorio. El *body* es el ultimo elemento del envelope y contiene el verdadero contenido del mensaje (*payload*).

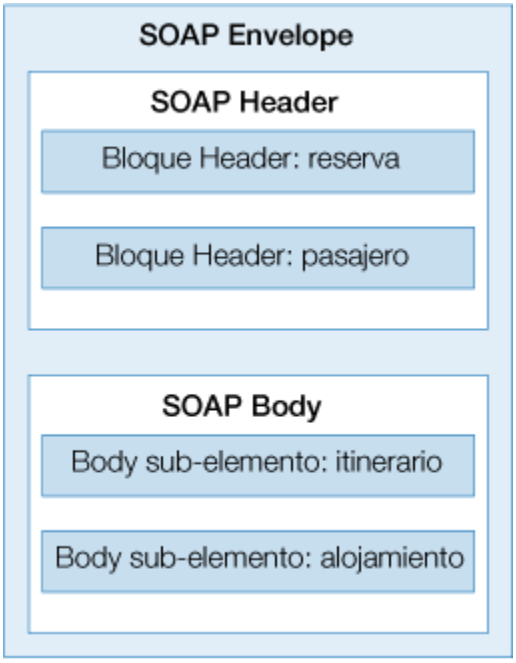

Fig. 2.3. Estructura de un mensaje SOAP.

SOAP se define independientemente del mecanismo de transporte de mensajes utilizado, por lo que podría utilizarse cualquier mecanismo alternativamente y su elección se puede hacer en el momento de ejecución, es decir, en el momento de enviar el mensaje. Además el mecanismo podría cambiar mientras el mensaje esta siendo enviado entre diferentes nodos. Aunque exista esta independencia en cuanto al mecanismo de transporte usado, el mas utilizado es HTTP.

Los mensajes SOAP son transmitidos desde el emisor al receptor, lo que se conoce como one way. Estos mensajes pueden combinarse en esquemas de mensajes mas sofisticados como el patrón de pares de mensajes request/response sincrónicos, según lo requieran las diferentes aplicaciones distribuidas (como RPC que usa el esquema antes mencionado). Cuando no se puede controlar la latencia se requiere la mensajería asincrónica. Cuando se usa el esquema request/response asincrónico senecesitan mensajes de control (mandatory).

Los mensajes pueden ser ruteados según el contenido del header y el body. XML provee las herramientas para construir e inspeccionar mensajes completos, mientras que en arquitecturas

como DCOM, CORBA y Java Remote Method Invocation (RMI) no puede accederse a los header desde las aplicaciones. Cualquier agente de software que envíe o reciba un mensaje es llamado un nodo SOAP. Quien inicia la emisión es llamado el original sender, el nodo final que consume y procesa el mensaje es llamado ultimate receiver. Cualquier nodo entre ellos que procese el mensaje es llamado intermediary. El conjunto de intermediarios y el nodo final son llamados colectivamente message path.

Cada nodo que participa en el message path asume un rol: Next, es el rol universal de cada nodo SOAP; y ultimate receiver es el rol del ultimo nodo, típicamente una aplicación, quien ve el body del mensaje, a diferencia del header que es visto por cada nodo en el path [35].

2.2.1.1.2 Addressing

Esta especificación permite identificar servicios Web y mensajes de servicios Web independientemente del protocolo de transporte utilizado, haciendo posible hacer que las peticiones a servicios Web puedan ser transmitidas a través de redes compuestas por nodos de manera que la información de transporte sea independiente del protocolo utilizado para la transmisión del mensaje.

Esta especificación incluye dos tipos de información:

- **Endpoint References**, identifican el punto donde deben ser dirigidas las peticiones para acceder a un servicio.

- **Message Information Headers**, son cabeceras que contienen información para la identificación de los mensajes independientemente del protocolo de transporte.

Se usan prefijos pertenecientes a ciertos espacios de nombres (namespace) para identificar cada ítem de información definido en estaespecificación.

Prefijo	Namespace
S	http://www.w3.org/2003/05/soap-envelope
S11	http://schemas.xmlsoap.org/soap/envelope
wsa	http://schemas.xmlsoap.org/ws/2004/08/addressing
wsp	http://schemas.xmlsoap.org/ws/2002/12/policy
xs	http://www.w3.org/2001/XMLSchema

Tabla 3.1: Namespace usados en WS-Addressing. [39]

2.2.2 REST (Representational State Transfer)

REST [76] es una arquitectura que consiste básicamente de clientes y servidores, donde los clientes envían solicitudes a los servidores, quienes las procesan y retornan una respuesta. Las solicitudes y las respuestas se basan en la transferencia de una representación de los recursos, siendo un recurso cualquier concepto que pueda ser referenciado y su representación es un documento que captura el estado del mismo.

Un cliente puede atravesar por diferentes estados de una aplicación, se dice que el cliente está en un rest-state, donde es capaz de interactuar con los usuarios pero no de crear ni consumir nada desde el almacenamiento de los servidores, mientras se espera la respuesta de una solicitud enviada. En la representación de los estados de una aplicación hay enlaces que pueden ser usados para realizar la siguiente transición de estados.

REST fue descrito inicialmente en el contexto de HTTP pero no está limitado únicamente a este protocolo. La arquitectura RESTful se basa en otros protocolos en la capa de aplicación que deben proveer un vocabulario rico y uniforme para expresar la representación de los estados.

La arquitectura REST tiene las siguientes características:

- Los clientes están separados de los servidores por interfaces uniformes. Esto permite que el código de los clientes sea portables, ya que por ejemplo desconocen todo respecto al almacenamiento de los datos. Los servidores a su vez pueden ser mas simples y escalables al ignorar las cuestiones relacionadas con las interfaces o estados de los clientes. Asi mismo los clientes y servidores pueden ser desarrollados y reemplazados independientemente.

- La comunicación entre clientes y servidores no está restringida por el estado de los clientes, toda la información requerida para poder responder a una solicitud está contenida en ella misma. Así los servidores son mas confiables ante fallas en la red.

- Los clientes son capaces de almacenar las respuestas, esto puede ser hecho explícita o implícitamente. Esto reduce el número de interacciones entre clientes y servidores, mejorando la escalabilidad y el rendimiento.

- Los clientes ignoran si la conexión con un servidor es directa o no, los servidores intermediarios aumentan la escalabilidad del sistema ya que posibilitan el balanceo de carga.

- Los servidores pueden extender su funcionalidad o configurarla de acuerdo a la necesidad de un cliente, quien tiene la posibilidad de enviar la lógica que se requiere que ejecute el servidor (como applets de java o script en JavaScript). Para restringir esto se configuran constraint.

- Los recursos son identificados por ejemplo usando URIs. Cuando un cliente conoce la representación de un recurso tiene entonces la posibilidad de modificarlo o borrarlo del servidor según tenga los permisos.

- Cada mensaje incluye la información necesaria que describe como debe ser procesado, por ejemplo si hay que invocar a un parser.

REST intenta minimizar la latencia y comunicación sobre la red y maximizar la independencia y escalabilidad de la implementación de los componentes a través de las restricciones sobre los conectores semánticos. Además REST permite guardar y reusar las interacciones, sustituir dinámicamente componentes y procesar las acciones por intermediarios, permitiendo sistemas de hypermedia distribuidos y escalables [76].

2.2.3 Lenguajes para la descripción semántica de los servicios Web

El objetivo principal de los lenguajes para la descripción semántica de los servicios Web es el de automatizar las operaciones relacionadas con los servicios Web: descubrimiento, selección, composición y ejecución. Algunos de estos lenguajes han sido pensados para atender algunas de cuestiones más específicas de los servicios Web como es la composición.

2.2.3.1 OWL-S

Con el fin de proveer diferente tipo de conocimiento sobre los servicios esta ontología se estructura en tres partes: Profile: permite enfocarse en qué hace el servicio desde la perspectiva del cliente; Modelo de procesos que responde a cómo se usa; y Grounding que especifica cómo es la forma de interactuar con el servicio (especificación concreta que incluye: protocolo y formato de mensajes, serialización, transporte, direccionamiento) [11].

Mientras que WSDL permite la especificación de las operaciones de los servicios Web, las cuales proveen la estructura organizacional sobre la que se especifican los patrones y la sintaxis de los mensajes de entrada/salida, OWL-S [11] provee una construcción análoga conocida como proceso atómico, los cuales están caracterizados por sus entradas, salidas, precondiciones y efectos.

Los tipos de las entradas y salidas de un proceso atómico corresponden al sistema de tipos de OWL, el cual permite el uso de conceptos definidos y compartidos en la Web semántica. En el ejemplo a continuación se muestra el código de una declaración de proceso atómico con su especificación de entradas/salidas, donde el tipo de AtomicProcess, input, output, y parameter

pertenecen al espacio de nombres del modelo de procesos OWL-S. Se asume en este ejemplo que Human, BookTitle, e ISBN son clases definidas en ontologías del dominio apropiado con otro espacio de nombres.

```
<AtomicProcess ID="AuthorSearch">
  <hasInput>
    <Input ID="Author">
      <parameterType resource="#Human">
    </Input>
  </hasInput>
  <hasInput>
    <Input ID="Title">
      <parameterType resource="#BookTitle">
    </Input>
  </hasInput>
  <hasOutput>
    <Output ID="BookID">
      <parameterType resource="#ISBN">
    </Output>
  </hasOutput>
 /At i P
```

El grounding para este proceso debería establecer la correspondencia con una operación WSDL y la correspondencia de cada elemento de E/S a un elemento de un mensaje WSDL en particular (para hacer esto puede ser necesario un script XSLT de transformación).

Las precondiciones de un proceso son fórmulas lógicas que deben ser satisfechas por quien requiere el servicio. Los efectos son fórmulas lógicas que se deben cumplir tras la ejecución del servicio. OWL-S permite el uso de lenguajes mas expresivos que OWL para la especificación de las precondiciones y efectos, tales como RuleML, DRS o OWL Rules Language. por ejemplo una precondición podría ser que para hacer una compra la cuenta y la tarjeta de crédito deben ser válidas.

2.2.3.2 WSMF (Web Service Modeling Framework)

WSMF es un formato XML para describir servicios en la red como un endpoint operando sobre mensajes que contienen información orientada a documentos u orientada a procedimientos. Un endpoint está constituido por puertos, los puertos constituyen un servicio. La asignación de un

protocolo a un puerto constituye un binding.

El modelo WSMF consiste de cuatro elementos diferentes: *ontologías* que proveen la terminología usada por otros elementos, *metas* que representan los problemas que deberían ser resueltos por los servicios Web, *descripciones de servicios Web* que definen varios aspectos de un servicio, y *mediadores* que resuelven los problemas de interoperabilidad. [5]

2.2.3.3 WSMO (Web Service Modeling Ontology) y WSML (Web Service Modeling Language)

WSMO se basa en WSMF (Web Service Modeling Framework) refinando y extendiéndolo, y diseñando una ontología formal y un conjunto de lenguajes. Los cuatro elementos principales de esta propuesta son: ontologías, descripciones de servicios, metas y mediadores[6]. Utiliza el lenguaje WSML [13] que provee una sintaxis formal y semántica.

2.2.3.4 WSDL-S

Define un mecanismo para anotar documentos WSDL con información semántica, las cuales se refieren a conceptos semánticos que definen el significado de las operaciones del servicio así como el de sus entradas y salidas. Las anotaciones semánticas son definidas como un conjunto de elementos de extensión de WSDL y atributos que no están ligados a ningún lenguaje de representación de ontologías en particular. También se agrega un mecanismo para especificar precondiciones y efectos de los servicios Web [7].

2.2.3.5 SAWSDL (Semantic Annotations for WSDL and XML Schema)

La extensión de WSDL y XML Schema permite agregar semántica a los componentes de WSDL. SAWSDL no representa un lenguaje para especificar modelos semánticos, sino que provee mecanismos por los cuales se pueden referenciar los conceptos del modelo semántico, que están especificados fuera del documento WSDL, desde los componentes WSDL y XML Schema.

Define una forma de anotar interfaces y operaciones WSDL con información de

categorización, anotaciones sobre los tipos Schema, define atributos que pueden ser aplicados tanto a los elementos WSDL como elementos XML Schema, etc. El mecanismo de anotación es independiente del lenguaje ontdógico, también es independiente de los lenguajes de mapeo [8].

2.3 Web Semántica de Servicios

La evolución de la Web viene acompañada con el desarrollo de nuevos estándares y tecnologías, dentro de las cuales se destacan la **Web Semántica** y los **Servicios Web**.

La Web Semántica pretende conseguir una comunicación efectiva entre computadoras, para lo cual centra sus esfuerzos en la búsqueda de descripciones enriquecidas semánticamente para los datos en la Web. Para lograrlo se requieren descripciones que incluyan no sólo las estructuras de datos, sino también las relaciones existentes con otros conceptos, las restricciones, reglas que permitan realizar inferencias, etc. En este contexto también se promueve la definición y reutilización de vocabularios u ontologías de conceptos que faciliten el procesamiento por parte de las máquinas.

Los Servicios Web promueven la interacción entre aplicaciones. El servicio es un componente software que puede procesar documentos XML recibidos a través de protocolos de transporte y de aplicación.

Ambas tecnologías constituyen lo que se conoce como la Web Semántica de Servicios, la cual proveerá otras funciones y requerirá otras técnicas y modelos que afectarán todos los ámbitos: educativos, e-comerce, colaboración y comunicación, etc.

El desarrollo de sistemas más complejos o que cumplan con las características de ser correctos, claros y verificables es todo un desafío. Esto muchas veces no se logra y en su lugar ocurren aumento del presupuesto, extensión de los plazos establecidos, software de baja calidad, software que no satisface las expectativas del cliente. Con el intento de mejorar las prácticas de desarrollo es que existen varios paradigmas de entre los que se puede seleccionar el que resulte más adecuado para el caso: programación estructurada,orientada a objetos, componentes, etc. Pero

muchos opinan que para lograr un cambio más radical es necesario adoptar otro paradigma. Uno en el que el software deje de ser considerado como un producto manufacturado para empezar a verlo como un servicio. Los servicios Web son la base de este cambio y permitirán fortalecer las características de interoperabilidad, confiabilidad, escalabilidad y disponibilidad.

La tecnología de servicios Web proveera el entendimiento automatico de las funciones provistas por un servicio Web mientras que la tecnología de la Web semántica soporta la definición de la semántica de un documento, de forma de sustituir las técnicas actuales de búsqueda basadas en la coincidencia de palabras (keyword matching) y recuperación de información (information retrieval) por búsquedas mas inteligentes. Las representaciones procesables por las máquinas de la semántica de los datos estructurados, soportará la transformación automática y matching de diferentes formatos de la misma pieza de información.

Combinar ambas tecnologías permitirá por ejemplo descubrir que dos servicios distintos proveen la misma funcionalidad aunque los tipos de puerto, operaciones y mensajes sean llamados distinto. La tecnología de Web semántica podrá determinar que dos puertos distintos están llevando la misma información y que son usados en la misma forma dentro de las operaciones. Las anotaciones semánticas adicionales que describen cada uno de los servicios son la base para el razonamiento sobre la funcionalidad de las operaciones.

Un concepto relevante en la Web semántica es el de ontología, lo cual es una representación de conceptos (o entidades) del dominio de una aplicación y todas las relaciones entre esos conceptos. Las anotaciones a través de ontologías hacen procesable ese conocimiento.

La tecnología de servicios Web semánticos es considerada otra forma de conocimiento para describir el significado de los servicios Web. Por ejemplo, cuando el orden de las operaciones es importante las anotaciones con el modelo de procesos BPEL puede asegurar menos errores.

2.4 Ontologías

Los lenguajes para la definición de vocabulario (ontologías) permiten definir ontologías

otorgando un significado claro y bien definido a cada elemento mencionado en una descripción.

2.4.1 RDFS o RDF Schema

El modelo de datos RDFS permite describir las interrelaciones de los recursos en términos de propiedades y valores. Las propiedades RDF pueden ser los atributos de los recursos o pueden representar relaciones entre los recursos [1].

Los documentos RDF-S mantienen la sintaxis y la estructura de los documentos RDF. Es posible usar tipos de datos simples y otros definidos, jerarquías de clases y de propiedades, definir restricciones y relaciones específicas entre clases, restringir el dominio de una propiedad y el rango, etc. La semántica de la información es expresada a través de la definición de propiedades.

2.4.2 DAML

El lenguaje DAML ha sido desarrollado como una extensión de XML y RDF, agregando otras propiedades tales como equivalencia o que propiedades particulares son únicas [2].

Puede especificarse la cardinalidad de los elementos, enumerar los valores de las propiedades, pueden especificarse las propiedades y rangos global y localmente, usarse tipos de datos básicos y estructurados. Se puede expresar negación: daml:complementOf y permite especificar la relación de equivalencia y la unicidad de ciertas propiedades.

2.4.3 DAML+OIL

Se basa en otros estándares como RDF y RDF Schema. Una ontologías DAML+OIL está compuesta por varios componentes, algunos de los cuales son opcionales y algunos de los cuales pueden estar repetidos: permite definir cero o mas headers, en el cual se puede incluir información sobre la versión y referencias a otras ontologías DAML+Oil permite definir cero o mas elementos clase e instancias.

En cuanto a las clases permite expresar jerarquía, clases disjuntas, equivalencia, enumeración de instancias, etc. En cuanto a las propiedades permite expresar jerarquías, el

dominio y el rango, equivalencia, transitividad, unicidad, etc. En cuanto a las instancias existen sentencias que permiten verificar si dos objetos son el mismo individuo o no [12].

2.4.4 OWL o DAML-S

OWL provee tres sublenguajes incrementalmente expresivos: OWL Lite soporta una jerarquía de clasificación y restricciones simples. OWL DL soporta máxima expresividad mientras retiene la completitud computacional (todas las conclusiones se garantiza que son computables) y decidibilidad (todos los cómputos terminan en un tiempo finito). OWL Full soporta máxima expresividad y sintaxis proveniente de RDF sin garantías computacionales. OWL Full permite que una ontología sea aumentada con significado predefinido de vocabulario (RDFu OWL).

OWL extiende RDFS para permitir la expresión de relaciones complejas entre diferentes clases RDFS, y mayor precisión en las restricciones de clases y propiedades específicas. Esto incluye por ejemplo la posibilidad de limitar las propiedades de clases con respecto a número y tipo, expresar relaciones uno-a-uno, varios-a-uno o uno-a-varios, permite expresar relaciones entre clases definidas en diferentes documentos en la Web, construir nuevas clases a partir de uniones, intersecciones y complementos de otras, así como restringir rangos y dominios para especificar combinaciones de clases y propiedades [17].

SKOS (Simple Knowledge Organization System) es un modelo de datos para compartir y enlazar el conocimiento de sistemas vía Web. Muchos sistemas de conocimiento de las organizaciones, tales como tesaurios (thesauri) y taxonomías, esquemas de clasificación y subject heading systems, comparten una estructura similar y son usados en aplicaciones similares. SKOS captura estas similitudes y las hace explícitas para permitir compartir datos y tecnología a través de diversas aplicaciones.

Este modelo provee estándares y bajo costo de migración de los sistemas de conocimiento de las organizaciones a la Web semántica. También provee un lenguaje bastante intuitivo para desarrollar y compartir nuevo conocimiento en dichos sistemas. Esto puede ser usado en

combinación con lenguajes formales derepresentación de conocimiento como OWL [41].

2.4.5 SWRL(Semantic Web Rule Language)

Es un lenguaje de reglas para la Web Semántica, que combina sublenguajes de OWL Web Ontology (OWL DL and Lite) con lenguajes Rule Markup (Unary/Binary Datalog).

Este lenguaje se usa en el contexto de los servicios Web. Consiste de reglas de la forma de una implicación entre antecedente (body) y consecuente (head). El significado se puede leer como: si las condiciones especificadas en el antecedente se verifican, luego las condiciones especificadas en el consecuente también deben verificarse [4].

El antecedente y el consecuente consisten de cero o más átomos. Los átomos son de la forma C(x), P(x,y), sames(x,y) o differentFrom(x,y). Donde C es una descripción OWL, P es una propiedad, x e y son variables, individuos OWL o valores. La sintaxis de este lenguaje es una extensión de la sintaxis OWL en la que se combina BNF. Es posible expresar que la combinación de dos propiedades implica otra propiedad, expresar condiciones de existencialidad, etc.

2.5 Resumen

En este capítulo se repasan las diferentes tecnologías y estándares surgidos en torno a la Web semántica y los Servicios Web. Los servicios Web semánticos constituyen el resultado de la integración de ambas tecnologías. La incorporación de semántica en la descripción de los servicios Web permite el descubrimiento de los servicios que implementan la funcionalidad requerida. Se han analizado también los lenguajes surgidos para permitir la descripción semántica de los servicios Web: OWL-s, WSMF, WSMO, WSDL-S y SAWSDL.

Las ontologías permiten crear vocabularios de conceptos y relacionarlos, de esta manera es posible la integración de servicios Web. Han surgido diferentes lenguajes para la definición de ontologías, tales como RDFS, DAML, DAML+OIL, OWL y el lenguaje que ha surgido para ser usado específicamente en el contexto de los servicios Web es SWRL. Cada uno tiene sus

particularidades y diferente nivel de expresividad.

El entorno en el que se ejecutan los servicios Web debería tener la habilidad de soportar la integración de los servicios así como la integración delos datos. La integración a nivel de servicios requiere una infraestructura que permita la composición de los mismos. En el siguiente capítulo se analizan las cuestiones relacionadas con la composición de servicios Web así como los lenguajes surgidos para que esto sea posible.

Capítulo 3:

Composición de Servicios Web

En este Capítulo se estudian los conceptos fundamentales de la composición de servicios, tales como coreografía y orquestación, se analizan los problemas y cuestiones actualmente en estudio y también se analizan las dos técnicas principales de composición: basadas en workflow y basadas en planning.

Al final del Capítulo se presentan los estándares y propuestas surgidas en torno al tema de composición y se resume el estado del arte de esta temática para continuar profundizando en el siguiente capítulo.

3.1 Introducción

La composición de servicios implica que un servicio pueda ser usado como un servicio básico dentro de una composición o que brinde su funcionalidad para satisfacer la funcionalidad que un cliente requiere. La descripciones de los servicios tales como meta-datos y terminología estándar permiten realizar la composición. Luego se debe coordinar la ejecución de la composición, controlando los flujos de datos y de control entre los servicios que interactúan. Ello implica el establecimiento de políticas y acuerdos de composición. En [81] se presenta la composición de servicios en cuatro dimensiones que se caracterizan por:

- Grado de participación del usuario en la definición de la composición.
- Composición basada en plantillas o en las instancias actuales de servicios.
- Dinamicidad de la composición.

- Grado de participación del usuario en la dinamicidad o adaptación de la composición, en una composición automática es el sistema el que controla los flujos de control y de datos de los servicios ensamblados.

La última dimensión es todo un desafío debido a la dificultad de mapear las necesidades del usuario en una colección de servicios correlacionados donde la salida de ellos responda a los requerimientos del usuario. La base de la composición, en cualquiera de las dimensiones, es crear una composición usando las instancias actuales de los servicios o las plantillas predefinidas en las que se ensamblan los servicios individuales siendo buscados e integrados automáticamente en tiempo de ejecución.

El verdadero potencial de los servicios está en el hecho de poder integrarlos en servicios compuestos que aporten un valor agregado. Permitiendo brindar soluciones de negocios electrónicas y dinámicas, en este contexto han surgido dos conceptos que serán analizados a continuación: orquestación y coreografía, los cuales se han usado para describir los protocolos de interacción implicados en la colaboración de los servicios.

Para que la composición de servicios sea posible varios trabajos de investigación han realizado sus aportes para definir una arquitectura que incluya los estándares, técnicas, modelos y lenguajes que permitan realizarla. La arquitectura debe permitir realizar el proceso de composición de servicios Web, donde en primer lugar los proveedores de servicios deben describir y publicar los servicios y los usuarios deben realizar las solicitudes. El proceso de definición y composición puede requerir la traducción a un lenguaje más formal. Como resultado el generador del proceso de composición puede retornar mas de un modelo de composición que satisfaga los requerimientos de una solicitud particular, y deberán ser evaluados usando para ello los atributos no funcionales. El servicio compuesto que es seleccionado será ejecutado.

Para que se inicie el proceso de composición debe existir una solicitud que no pueda ser satisfecha por ningún servicio existente por lo cual se requiere la combinación de varios servicios.

Se pueden diferenciar entre los lenguajes de especificación de servicios internos y los lenguajes externos. Los lenguajes externos o de diseño permiten representar la composición de servicios de forma mas entendible, a diferencia de ellos los lenguajes internos son generalmente lenguajes más precisos y formales.

Al realizar una composición se debe verificar si los servicios que la integran verdaderamente se pueden componer. Por un lado las operaciones que se invocan desde un cliente deben tener una correspondencia en los parámetros que se esperan desde el servicio.

La integración de servicios en forma manual es muy difícil debido a que el número de servicios disponibles crece exponencialmente y a que no existe un lenguaje universal tal que se describa un servicio con un único significado, es decir que tenga una única interpretación. Para llevar a cabo la composición automática o en forma semi-automática hay dos paradigmas: workflow y planning de inteligencia artificial que serán estudiados en este capítulo.

3.2 Coreografía y Orquestación

Coreografía (Web Services Choreography) y **Orquestación** (Web **Serv**ice Orchestration) son dos conceptos que suelen confundirse, aunque tienen similitudes y pueden estar relacionados pero tienen objetivos diferentes.

La *Orquestación* puede verse como la ejecución de un proceso de negocio, por lo que el lenguaje que se relaciona con esto es WS-BPEL [79] (este estándar proviene de BPEL4WS), describe cómo un servicio interactúa con otro a nivel de mensajes, incluyendo la lógica del negocio y el orden de ejecución de las interacciones desde la perspectiva de uno de los implicados. En cambio una *Coreografía* se relaciona con la colaboración entre actores, es decir, con las interacciones entre servicios Web. Está asociado con el intercambio de mensajes, las reglas de interacción y los acuerdos que ocurren entre los procesos implicados.. El lenguaje que se relaciona con esto es WS-CDL [80] (Web Services Choreography Description Language). Podríamos ver la orquestación como un mecanismo para el intercambio de mensajes a través de un proceso (un flujo

de control) y la coreografía como una descripción de los actores que intercambian mensajes para ejecutar varios procesos particulares (varios flujos de control).

En [86] se presentan dos tipos de orquestación, el primero es un enfoque basado en la topología hub-and-spoke, en el cual un servicio juega el rol de mediador, de forma que todas las interacciones pasan a través de este servicio; el segundo es un enfoque peer-to-peer donde cada servicio interactúa directamente con los otros, no hay un control centralizado.

Las siguientes propuestas se pueden clasificar como basados en el enfoque mediado:

- En [87] la orquestación es considerada como un meta-servicio llamado Composite Service (CES). Un proveedor puede brindar un nuevo servicio como resultado de la coordinación de otros, para ello se registra el nuevo servicio en el CES, el cual controlará luego su ejecución. En ese trabajo eFlow soporta la definición de servicios compuestos adaptivos y dinámicos.

- En [88] la coordinación de los servicios se realiza modelando los procesos como diagramas de estados. La Figura 3.1 muestra la especificación de un workflow como un diagrama de estados.

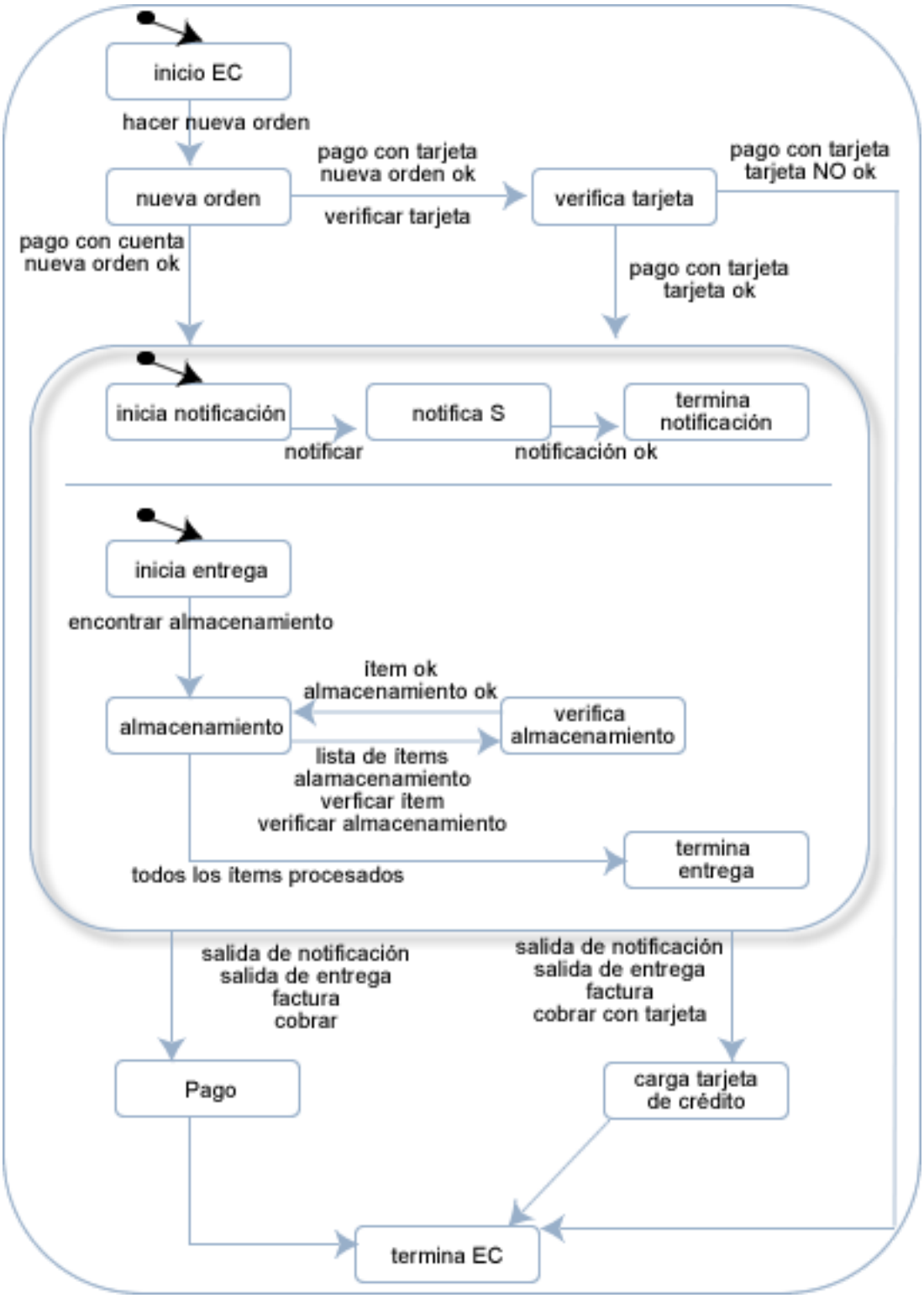

Fig. 3.1 Representación de workflow usando diagramas de estados

- En [90] la orquestación de los servicios se realiza usando Redes de Petri, la Figura 3.2 muestra los elementos básicos de este modelo.

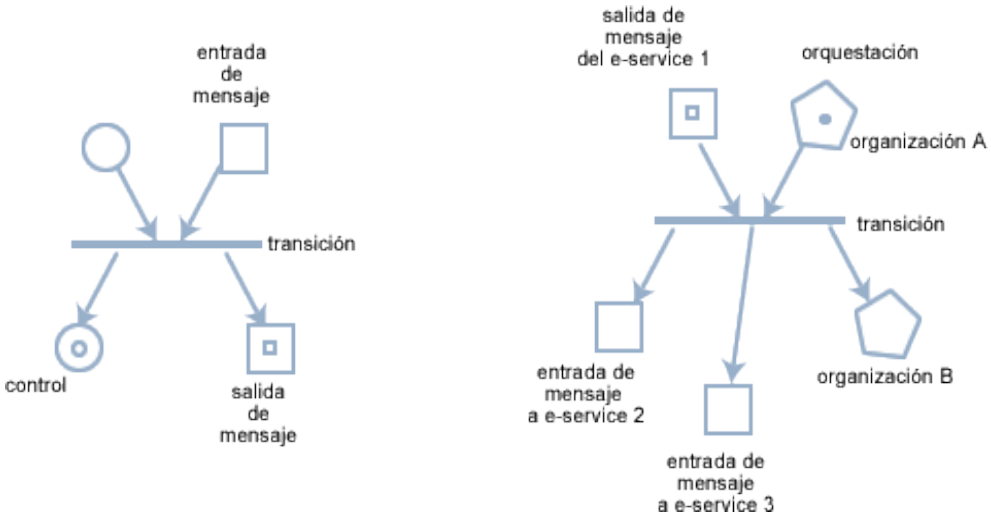

Fig. 3.2 Orquestación usando redes de Petri

Una propuesta basada en el enfoque peer-to-peer se presenta en [91], donde los servicios compuestos se modelan como diagramas de actividad y la coordinación de los servicios se realiza en forma descentralizada por los estados coordinadores de cada componente y del servicio compuesto.

WS-BPEL es un lenguaje de orquestación ya que se centra en la composición de nuevos servicios a partir de otros, y permite describir qué necesitan los servicios para trabajar conjuntamente.

WS-CDL es un lenguaje coreográfico y se enfoca en la interoperabilidad de los servicios para crear otros más complejos, y permite especificar reglas de unión de los servicios Web. La especificación del contrato entre las partes representa el comportamiento observable de los servicios que colaboran. Este lenguaje se basa en XML y su modelo conceptual en calculo II, este lenguaje provee construcciones para representar la comunicación, alternativas, concurrencia e iteración, con una semántica formal.

3.2 Problemas de la Composición

Un gran cambio se da en la industria de desarrollo al adoptar el enfoque orientado a

servicios, donde se deben investigar nuevas tecnologías, métodos y herramientas que soporten la composición flexible, confiable, fácil de usar, de bajo costo y tiempo, y dinámica de las aplicaciones.

El desarrollo orientado a servicios implica la implementación de interfaces bien definidas para las unidades funcionales de software de forma que puedan ser integradas. La composición de servicios apunta a las técnicas para la integración de esos servicios .

La composición de servicios es la orquestación de un número de servicios para proveer otros servicios mas ricos que satisfagan algunos requerimientos del usuario. Las técnicas de composición implican expresar tanto los servicios elementales como los compuestos, así como la definición de un servicio compuesto requiere expresar los flujos de control y la información que se intercambia entre los servicios que lo componen.

La composición puede ser realizada en forma manual en tiempo de diseño conociendo previamente qué servicios intervendrán en la composición. La composición automática en cambio es de naturaleza más dinámica, permite optimizar algunos parámetros como ancho de banda y seleccionar de entre los servicios que están disponibles en ese momento, es una forma de modelado mas flexible, lo cual es importante para los escenarios del mundo real. Además el número creciente de servicios disponibles y la necesidad de realizar composiciones complejas hacen que la composición manual sea inadecuada.

Otra clasificación de la composición de servicios diferencia la **obligatoria** (mandatory) donde se requiere el correcto funcionamiento de todos los componentes, y la composición **opcional** donde los servicios pueden no estar en operación en el momento en que se realiza la ejecución del servicio compuesto sin afectar la funcionalidad del mismo.

Para que la composición de servicios sea posible se requieren formas comunes de pasar la información entre los servicios. Estándares tales como WSDL, SOAP y HTTP constituyen la

infraestructura de comunicación mas común y junto con XML permiten gran flexibilidad para la codificación y transformaciónde los datos.

La composición automática tiene la ventaja de que es posible la sustitución de servicios concretos cuando ocurre una falla por ejemplo, pero no es posible actualmente adaptarse y manejar los nuevos servicios que ofrecen nuevas funcionalidades o asegurar el funcionamiento de una composición si los servicios seleccionados son de-registrados durante la ejecución de la composición.

La composición automática de servicios se realiza sobre un requerimiento de un servicio en particular, por lo que es posible considerar las peculiaridades de la solicitud. En cambio la composición manual se realiza sobre un requisito general que luego responderá a las necesidades de varios usuarios, debiendo ser por este motivo, lo suficientemente general como para poder responder a un conjunto de requisitos.

Para que la automatización sea posible es necesario contar con ontologías que provean los conceptos y contar con lenguajes de descripción adecuados que ofrezcan construcciones del lenguaje apropiadas, de forma que se pueda realizar la descripción semántica de las propiedades funcionales y no funcionales delos servicios.

Algunas cuestiones que deben verificarse al realizar una composición son:

- Especificación: la notación usada para especificar la composición debe ser correcta, para lo cual existen estándares para realizarla, tales como BPEL4WS que ha surgido para competir con WSFL (Web Service Flow Language propuesto por IBM) y XLANG (propuesto por Microsoft).

- Acuerdo: debe existir un acuerdo entre el servicio compuesto y los servicios individuales, éstos últimos pueden especificar políticas con relación a la participación en un servicio compuesto [19].

- Soporte de cambios: si la interfaz de un servicio cambia, ésto no debe afectar el servicio compuesto; una solución es construir una capa de funcionalidad sobre la pila de estándares de composición, como se describe en [93].

Todo proceso de composición de servicios Web, representado en la Figura 3.3, debe ser iniciado por un servicio solicitante, que puede representar al usuario. El proceso de definición y composición de servicios puede requerir la traducción de lenguajes, de lenguajes de diseño a un lenguaje más formal que es utilizado por el generador del proceso de composición. Pueden generarse mas de un modelo de composición, en especial debido a que varios servicios tienen funcionalidad similar por lo que luego los servicios compuestos deben ser evaluados usando la información de los atributos no funcionales, y luego de seleccionar un servicio compuesto éste será ejecutado. En el siguiente capítulo se analizan las posibilidades de automatizar la composición de servicios Web.

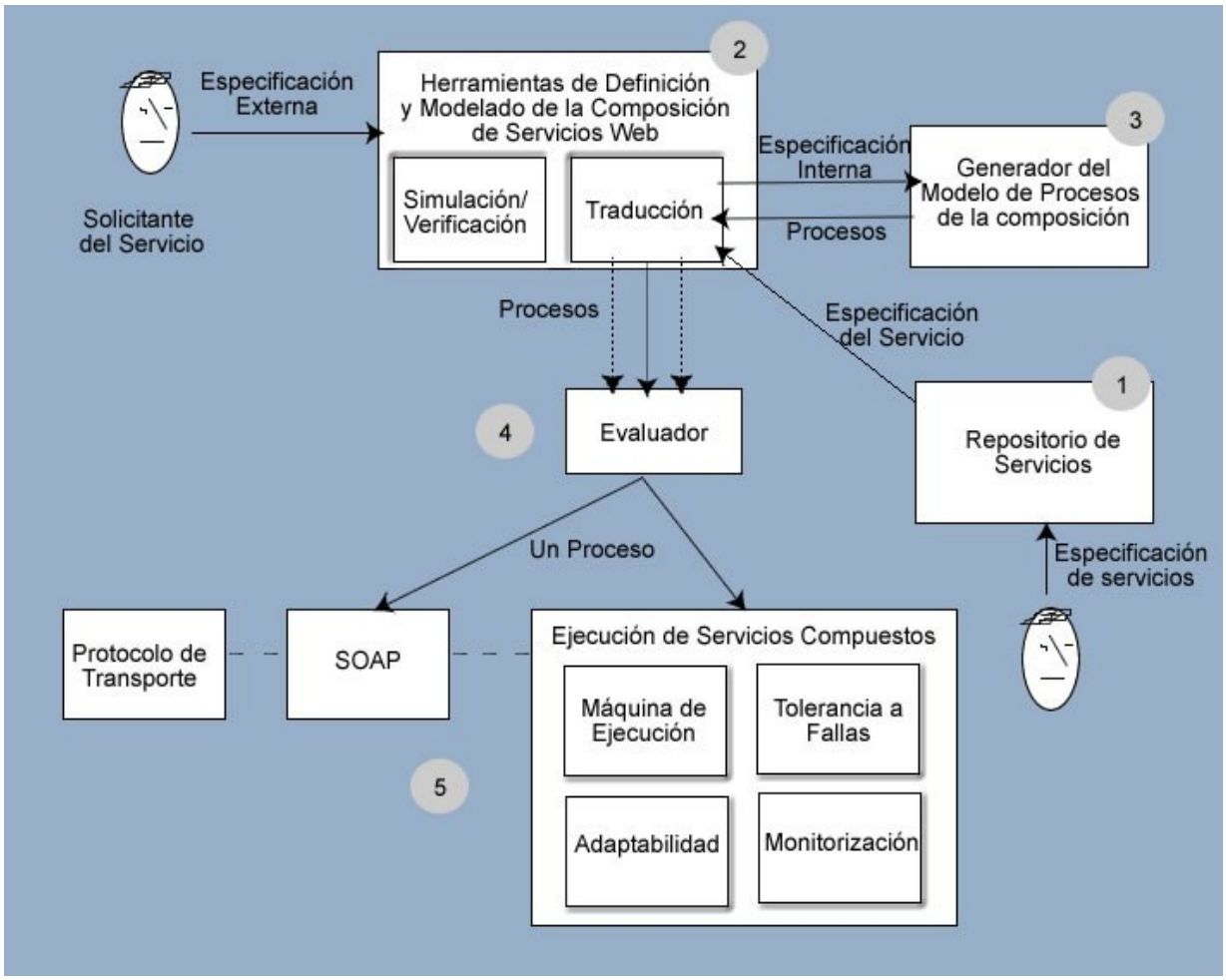

Fig.3.3 Arquitectura genérica de composición.

3.3 Workflow

Workflow es la tecnología que permite coordinar actividades humanas a través de la definición de roles, actividades y reglas de negocio, donde los flujos de trabajo constituyen el elemento principal del wokflow. El usuario y otros sistemas pueden intervenir en la ejecución de un flujo de trabajo para determinar si se dan las condiciones y el flujo de trabajo a seguir.

Un proceso de negocio es definido en [83] como un conjunto de tareas inter-relacionadas ligadas a una actividad. Los procesos de negocio tienen un punto de comienzo y puntos de finalización y son repetibles. Crear procesos de negocios para una empresa permite capturar el conocimiento en ellos. Aplicar workflow a procesos de negocio permite extender esta definición y

agregar reglas del negocio.

Un workflow puede dar respuesta a las preguntas Qué? Cómo? y Cuándo? de un proceso de negocios. Por un lado pueden representarse *quiénes* son los participantes en los flujos del proceso, qué roles juegan, cómo se organizan, etc. Los servicios Web pueden ser los participantes en un workflow. También se puede representar que es lo *que* hace cada participante. Algunas actividades son manuales y otras automáticas, pero cada vez puede automatizarse un número mayor de tareas, por ejemplo llamar a un proveedor para solicitar precios puede ser una actividad manual o puede ser reemplazada por un servicio Web que obtenga la lista de precios del vendedor. Por último se puede especificar *cuándo* un participante puede comenzar y cuándo su trabajo ha concluido, así como el orden en que los participantes realizan las tareas y bajo que circunstancias están habilitados para hacerlas.

Un workflow [82] pretende reflejar y automatizar la organización del sistema de información, para lo cual se deben establecer los mecanismos de control y seguimiento. Una tarea en un workflow es una unidad de trabajo que puede ser ejecutada por una o varias entidades (tal como un DBMS u otra aplicación del sistema). Una tarea está caracterizada por un conjunto de estados, uno de ellos es el estado inicial y uno o mas de ellos pueden ser estados finales. Además un conjunto de eventos son los responsables de modificar el estado de una tarea. En el workflow también pueden especificarse las condiciones que afectan la ejecución de las tareas, por ejemplo la dependencia entre ellas. Una estrategia para que esto sea posible es especificar las precondiciones que se deben satisfacer para ejecutar una tarea, lo cual puede incluir: el estado de ejecución de otras tareas, los valores de salida de otras tareas, variables externas (como tiempo), etc. Algunas tareas pueden ser dinámicas, es decir que se crean a medida que se ejecutan las tareas y son determinadas por un conjunto de reglas de ejecución. Una tarea se comunica con otra a traveś de variables, el flujo de datos está determinado por el valor de las variables de entrada y las variables de salida. La diferencia en la representación y formato de los datos que constituyen la salida de una tarea y la entrada de otra puede solucionarse con una función de traducción.

Un workflow también puede especificar y manejar los requerimientos de atomicidad y dependencias con otros workflows. Un planificador determina cuales son las tareas que pueden ejecutarse según el estado y las entradas del usuario.

Un servicio compuesto puede ser visto como un workflow ya que incluye un número de servicios atómicos que se comunican generando flujos de control y de datos. La ventaja de esta propuesta es que workflow es una técnica ampliamente estudiada, por lo que los avances en relación con la generación de workflow flexibles y que se adapten automáticamente pueden ser aprovechados al diseñar técnicas de composición usando esta herramienta.

El prototipo de Mentor-lite [89] es un sistema de workflow liviano que incluye: el *Intérprete* de las especificaciones del workflow, lo cual está constituido por diagramas de estado y de actividad; el *Communication Manager* y el *Log Mananger.* Los tres elementos constituyen el motor del workflow. La instancia de ejecución del workflow puede estar distribuida en varios motores en diferentes sitios. La Figura 3.4 muestra los elementos de esta arquitectura.

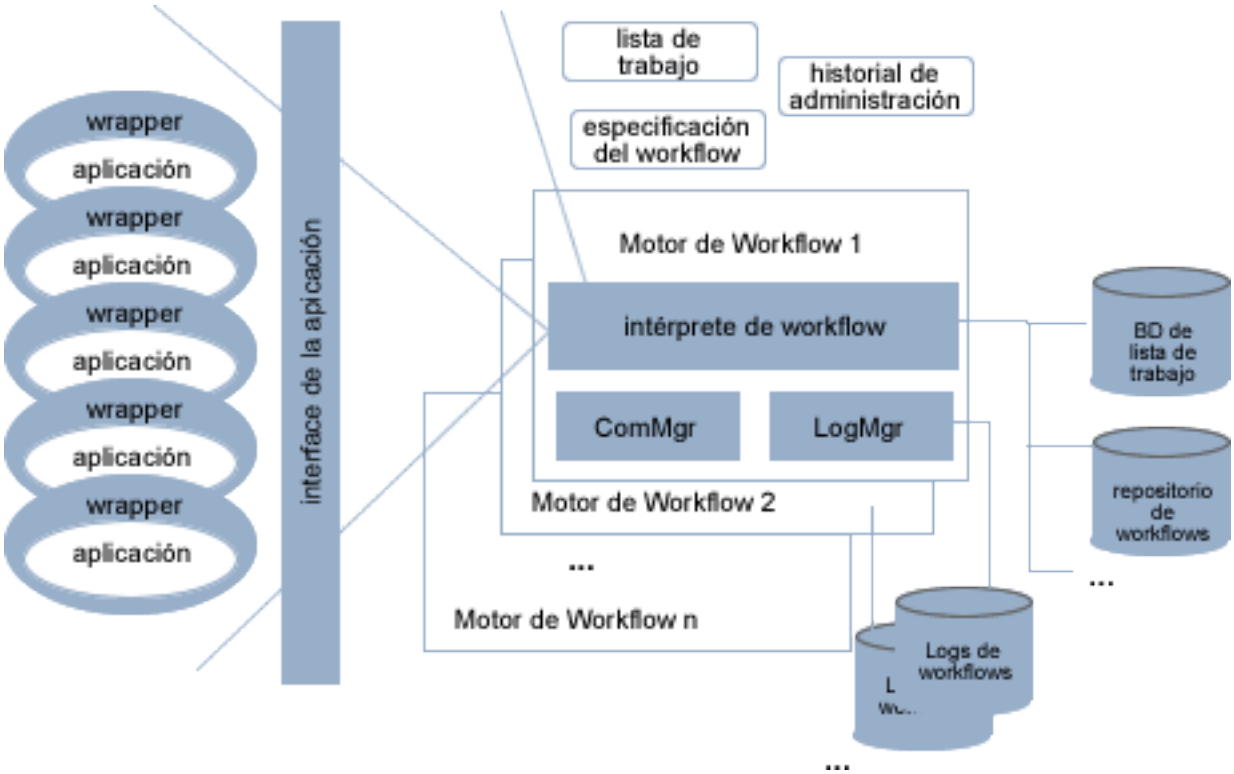

Fig. 3.4 Arquitectura del sistema Mentor-lite

En [94] se presenta una herramienta para automatizar la configuración de WFMS (Workflow Management System). Esta herramienta se basa en modelos matemáticos para estimar el grado de replicación de los servidores para lograr la performance y disponibilidad necesaria en los sistemas tales como Mentor-lite.

Los avances en el campo de los servicios electrónicos requieren que la tecnología de workflow sea cada vez mas eficiente, flexible y fácil de usar para que pueda integrarse con otras tecnologías como XML y servicios Web. En [88] se discute una arquitectura basada en XML para la administración de worflow distribuidos que son implementados en el sistema Mentor-lite. El elemento primordial es un mediador XML que maneja el intercambio de flujos de control y de datos con los clientes usando mensajes XML sobre http. El principal beneficio de esta arquitectura es la integración entre las aplicaciones clientes y los workflows. La Figura 3.5 muestra la arquitectura del sistema.

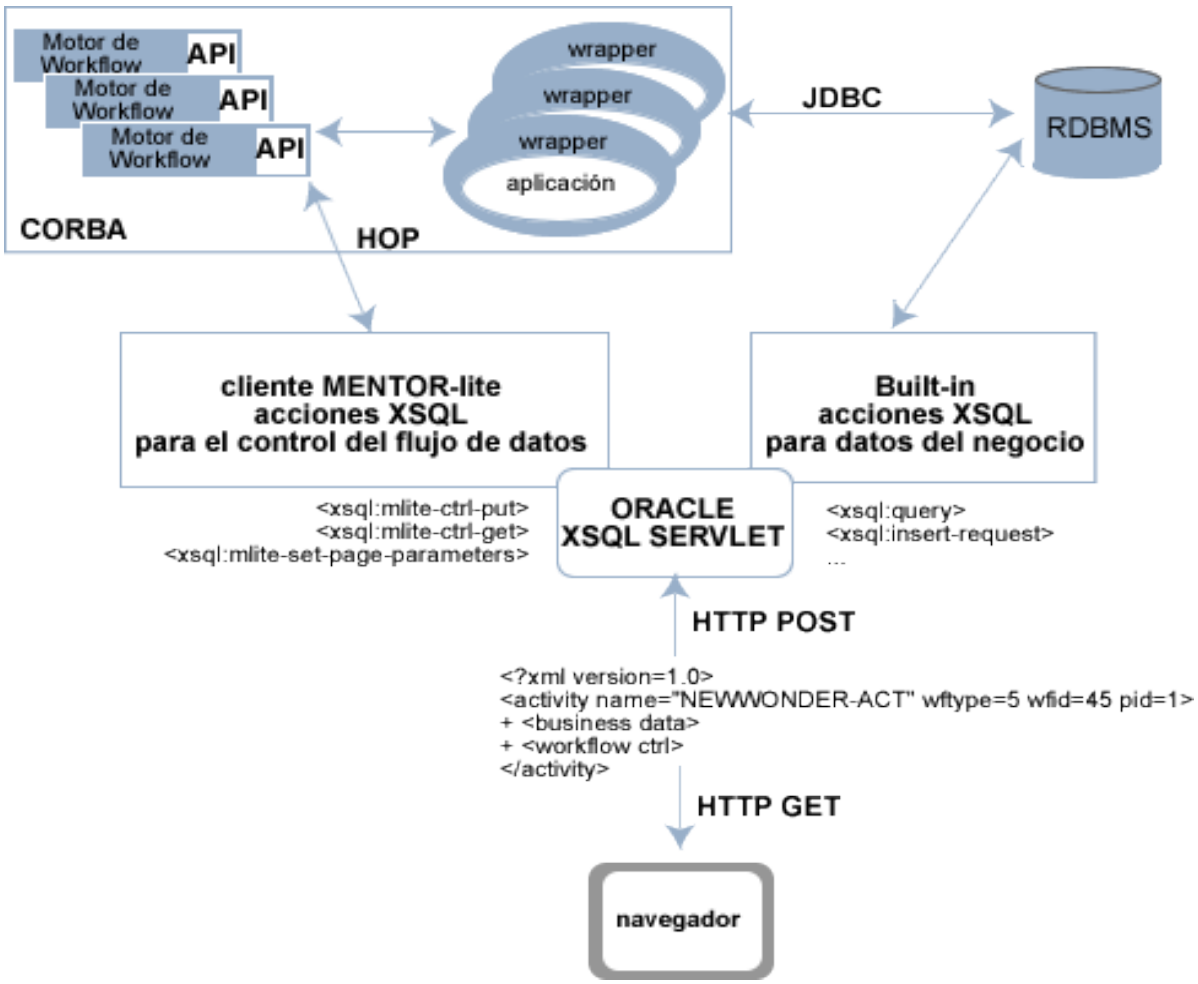

Fig. 3.5 Arquitectura para la administración de workflow

3.4 Planning

Esta técnica supone que cada servicio puede ser especificado en términos de sus precondiciones y efectos, los cuales corresponderían a los parámetros de entrada y de salida respectivamente. El estado del mundo integra a las precondiciones, el estado antes de la ejecución de un servicio, y el estado resultante tras la ejecución del servicio integra los efectos.

Un agente es un sistema de software que percibe el entorno y es capaz de realizar acciones para maximizar el éxito de las mismas. Los agentes deben ser capaces de alcanzar la meta visualizando el futuro, deben ser capaces de realizar una representación del estado del mundo y de hacer predicciones sobre cómo sus acciones producirán cambios, siendo responsables de hacer las elecciones que maximicen el valor de las mismas. En los problemas clásicos de planning, cada

agente asume que es el único actuando en el mundo y puede saber de esta manera cuales serán las consecuencias de sus acciones. Pero como esto no es así en la realidad debe chequear continuamente que se cumplan sus predicciones y a veces puede ser necesario que cambie su plan, por lo cual el agente debe razonar dentro de cierta incertidumbre.

Es posible de esta manera generar el plan de los procesos que deben ocurrir, usando las precondiciones y los efectos especificados. Además se pueden agregar restricciones, por ejemplo para reflejar la lógica del negocio.

En el **framework presentado** por Rao y Su en [18] intervienen dos clases de participantes: los proveedores y los consumidores de servicios. Otros componentes de esta propuesta son: *traductor*, es el que traduce del lenguaje interno usado por el generador de procesos y el lenguaje usado por los consumidores; *generador de procesos*: por cada requerimiento intenta generar un plan que a través de la composición de los servicios existentes en el repositorio intentará cumplir con la solicitud; *repositorio de servicios*: es donde reside la información de los servicios disponibles; *evaluador*: si se genera mas de un plan se evalúa y elije el mejor para ser ejecutado; *motor de ejecución*: ejecuta el plan y retorna los resultados al proveedor de servicios. En la Figura 3.6 está representado este sistema.

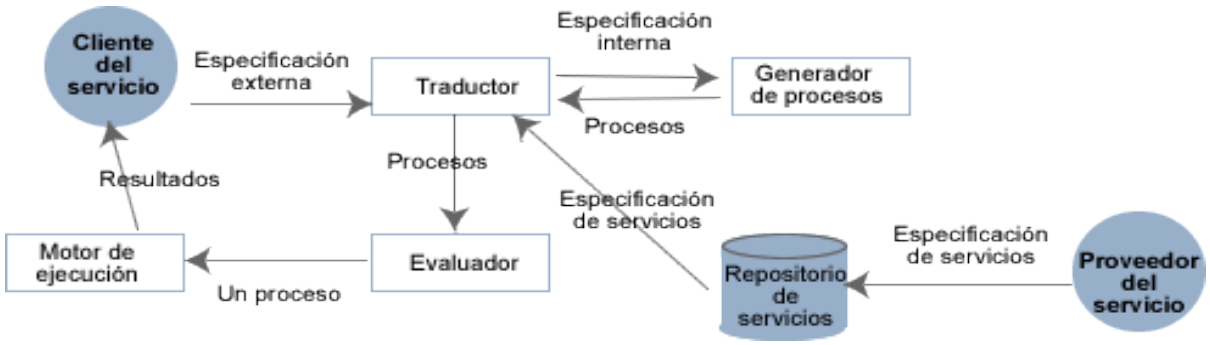

Fig. 3.6 Elementos implicados en una tarea de Planning

Las etapas implicadas son:

- Presentación de un servicio: se especifican las entradas, salidas y excepciones, así como la transformación a realizar sobre los datos de entrada y otros valores no funciones. La especificación en DAML-S se guarda en un registro UDDI.

- Traducción: los lenguajes externos (WSDL, DAML-S) usados para describir los servicios o especificar los requerimientos difieren del lenguaje interno usado por el generador del proceso, por lo que un traductor debe intervenir.

- Generación del modelo de composición: tomando la funcionalidad de los servicios genera un modelo de procesos que contiene los servicios seleccionados y la especificación de los flujos de control y de datos.

- Evaluación de los servicios compuestos: si se han propuesto mas de un forma de composición debe seleccionarse una usando los atributos no funcionales.

- Ejecución del servicio compuesto: implica la ejecución de los servicios de a cuerdo al modelo de procesos.

A continuación se describen los métodos de composición basados en IA Planning.

3.4.1 Cálculo de situación

Es un método para modelar sistemas dinámicos basado en lógica de primer orden, modela en forma explícita el hecho de que distintas situaciones se puedan dar en el tiempo. Se definen agentes que razonan sobre los servicios Web y los usuarios especifican los requerimientos y restricciones. Golog [33] es un lenguaje de programación lógico que declara un formalismo natural para la composición automática de tareas primitivas de los Servicios Web. Este lenguaje razona sobre las acciones y cambios producidos, el estado del mundo es representado por funciones y relaciones (fluentes).

En [32] se presenta una adaptación de Golog para la programación en la Web Semántica donde se deben manejar acciones sensibles. Para manejar las acciones sensibles se considera un tipo especial de conocimiento *K*, *K(s',s)* denota que si se conoce *s* pero se cree que *s'* también puede ocurrir. *do(a,s)* mapea desde la situación *s* y ocurriendo la acción *a* hacia una nueva situación. Una situación es una secuencias de acciones ocurridas desde la situación inicial S_0.

Para personalizar los programas Golog se introducen dos nuevos predicados: *Desirable(a,s)* que denota que estando en la situación *s* es deseable que ocurra la acción *a*; *Poss(a,s)* que denota que es posible que estando en la situación *s* se realice la acción *a*. Estas dos construcciones permiten incorporar restricciones semánticas a los programas.

También se incorporan dos predicados para modificar la semántica computacional del lenguajes: *Trans(δ,s,δ',s')* y *Final(δ,s)*; el primero denota que un programa δ que está en la situación *s* puede terminar en la situación *s'* y el resto del programa δ'; el segundo denota que un programa δ que está en la situación *s* puede terminar.

Se incorporan otras dos construcciones para manejar cuestiones de orden. La primera es *":"* que denota que dos acciones son realizadas en orden si esto es posible, es decir, $a_1:a_2$ si *Poss*(a_2,*do*(a_1,*s*)). La otra construcción agregada es el predicado *achieve* que denota la posibilidad

de alcanzar una situación.

3.4.2 Planning basado en reglas

En este método se usan reglas de composición sintácticas y semánticas. Se distinguen cuatro fases para llevar a cabo la composición: especificación (especificación a alto nivel de la composición), matchmaking (generación de planes), selección (selección de un plan usando parámetros de calidad) y generación (genera una descripción de la composición a realizar).

En [19] se presenta una técnica para generar servicios compuestos desde una descripción declarativa en alto nivel. Dado que el lenguaje de descripción de servicios Web mas difundido es WSDL y éste no da la posibilidad de expresar información semántica, en este trabajo se define una ontología en DAML+OIL la cual es modelada usando un grafo donde los nodos representan los conceptos WSDL y los arcos representan las relaciones entre los conceptos. En este enfoque se consideran tres tipos de participantes, los proveedores, composer y consumidores.

Se plantea un modelo de composabilidad representado en la Figura 3.7 en el cual se usan reglas para determinar si dos servicios son compatibles en sus parámetros de entrada y salida, para su posterior composición. Dichas reglas se dividen en sintácticas y semánticas. Dentro de las primeras se incluyen: mode composability, que compara el modo de operación (notificación, one-way, solicitud-respuesta, requerimiento-respuesta); binding composability, que compara los protocolos de los servicios que interactúan. Dentro de las reglas semánticas se incluyen: message composability, que compara el número de mensajes, parámetros, tipos de datos, etc.; operation semantics composability, que compara la semántica de las operaciones; qualitative composability, que compara las propiedades de calidad de los servicios web; y composition soundness, que verifica si es posible combinar los servicios.

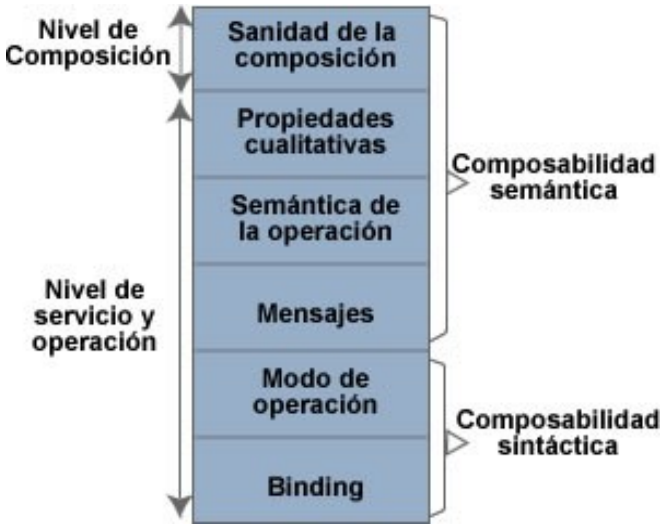

Fig. 3.7 Modelo de composabilidad

En la fase de especificación se usa el lenguaje CSSL (Composite Service Specification Language) para representar la composición luego de verificar la composabilidad.

SWORD [23] es una herramienta para construir composiciones de Servicios Web usando reglas. Las principales ideas en las que se basa SWORD son:

1. Los servicios individuales son definidos en término de sus entradas y salidas constituyendo un modelo de entidad-relación. Se define una *regla* según las entradas y salidas de un servicio.

2. Cuando un desarrollador quiere crear un nuevo servicio compuesto debe especificar las entradas y salidas del mismo.

3. Usando las reglas SWORD determina si es posible realizar el servicio compuesto usando los servicios existentes. Si es posible genera el plan de composición.

4. El servicio compuesto representa la secuencia de servicios que deben ser invocados para implementar el servicio compuesto, es decir, cada regla corresponde a la invocación de un servicio.

5. Para manejar las posibles alternativas en la ejecución de un servicio compuesto en esta propuesta se puede optar por solicitar la intervención del usuario o utilizar la elección por código.

A continuación ejemplificamos el funcionamiento de SWORD. Primero se describe el modelo de servicios, el modelo de composición y finalmente el mecanismo de generación del plan basado en reglas.

Modelo de Servicios

El ejemplo concreto corresponde a un servicio de correo y un servicio de direcciones.

```
EMail service:
Entities involved: X, Y
Condition Inputs:
  EMailMessage(X) – indicates that X is an email message
Data Inputs:
  subject(X), body(X), emailaddress(Y)
Condition Outputs:
  MailSent(X, Y)
Data Outputs: none
```

```
Driving Directions service:
Entities involved: X, Y
Condition Inputs:
  none
Data Inputs:
  streetaddress(X), city(X), state(X),
  streetaddress(Y), city(Y), state(Y)
Condition Outputs:
  none
Data Outputs:
  drivingdirections(X, Y)
```

El modelo estará completo con mas información disponible en tiempo de ejecución.

Composición

Suponemos que queremos crear un servicio que busca el camino para ir de la casa de una persona a la de otra dadas sus direcciones. Las entradas y salidas del servicio compuesto serían:

```
Names to Driving Directions service:
Entities involved: X, Y
Condition Inputs: Person(X), Person(Y)
Data Inputs:
    firstname(X), lastname(X), city(X), state(X),
    firstname(Y), lastname(Y), city(Y), state(Y)
Condition Outputs: none
Data Outputs: drivingdirections(X, Y)
```

El servicio compuesto puede ser implementado componiendo el servicio de búsqueda con el servicio de direcciones. El problema de planning puede ser modelado de la siguiente manera:

- Cada servicio puede ser modelado como una acción.
- Las precondiciones de una acción es la conjunción de todas las condiciones de entrada y los hechos conocidos que corresponden a los datos de entrada de los servicios.
- Las postcondiciones de una acción es la conjunción de las salidas y hechos conocidos correspondientes a los datos de salida de los servicios.
- El estado inicial es una conjunción de las condiciones de entrada del servicio compuesto y los hechos conocidos correspondientes a los datos de entrada del servicio compuesto.
- El estado final deseado es la conjunción de todas las condiciones de salida del servicio compuesto y hechos conocidos correspondientes a los datos de salida del servicio compuesto.

```
People lookup service:
precondition:
  Person(X) & Known(firstname, X) & Known(lastname, X)
  & Known(city, X) & Known(state, X)
postcondition:
  Known(streetaddress, X), Known(phone, X)

Driving directions service:
precondition:
  Known(streetaddress, X), Known(city, X), Known(state, X)
  Known(streetaddress, Y), Known(city, Y), Known(state, Y)
```

```
postcondition:
  Known(drivingdirections, X, Y)

Names-to-driving-directions composite service:
Initial state:
  Person(X), Person(Y), Known(firstname, X),
  Known(lastname, X), Known(city, X), Known(state, X)
  Known(firstname, Y), Known(lastname, Y),
  Known(city, Y), Known(state, Y)
Desired final state:
  Known(drivingdirections, X, Y)
```

Generación del plan basado en reglas

La generación del plan se logra usando un sistema experto que usa los hechos conocidos en el estado inicial y un conjunto de reglas if-then y deriva mas información. Las reglas correspondientes al servicio de búsqueda y al servicio de direcciones son:

```
Person(X) & Known(firstName, X) & Known(lastName, X) &
Known(city, X) & Known(state, X)
=> Known(streetaddress, X) & Known(phone, X)

Known(streetaddress, X) & Known(city, X) & Knwon(state, X) &
Known(streetaddress, Y) & Known(city, Y) & Knwon(state, Y)
=> Known(drivingdirections, X, Y)
```

Estas reglas deben ser escritas manualmente una vez, luego cada vez que se cree un servicio compuesto solo se deben especificar los hechos del estado inicial y final. SWORD usa el siguiente algoritmo para determinar el plan que implemente el servicio compuesto.

1. Se configuran las reglas correspondientes a todos los servicios disponibles (en Jess). En este caso se agregarían las 2 reglas mostrada recientemente.

2. Se configuran los hechos correspondientes al estado inicial del servicio compuesto.

```
Person(A)
Person(B)
Known(firstname, A)
Known(lastname, A)
Known(city, A)
```

```
Known(state, A)
Known(firstname, B)
Known(lastname, B)
Known(city, B)
Known(state, B)
```

3. Se ejecutan las reglas que sean posibles, cuando para se consulta por el estado final del servicio compuesto. Para el ejemplo la consulta sería:

```
Known(drivingdirections, A, B)
```

4. Se derivan los hechos correspondientes al estado final y se construye un plan inferido de dicha derivación.

Fig.3.8: Representación del plan simplificado

5. El plan construido puede ser visto por el desarrollador usando una herramienta gráfica, algo similar a lo que muestra la Figura 3.8.

3.4.3 Model Checking

En esta técnica se utiliza un procedimiento para determinar en forma automática si las especificaciones son satisfechas por los grafos que representan el estado actual del problema y cómo se comportan las soluciones parciales con respecto a las metas. En [21] ha sido empleada para detectar errores en distintos sistemas.

En [52] se plantea el uso de esta técnica para afrontar el tema de composición automática

de servicios Web. Este enfoque se basa en el chequeo del modelo de sistemas de transición de estado, en el cual se modela el comportamiento del sistema como un conjunto de estados. Por otro lado los servicios Web pueden especificar su comportamiento por el intercambio de mensajes y en general el estado del servicio es desconocido. Por lo tanto es necesario mapear el paradigma basado en mensajes a un paradigma orientado a estados para poder usar esta técnica. Esto puede lograrse encapsulando los cambios ocurridos tras la ejecución de una operación dentro de un estado. Para realizar la composición no solo deben considerarse los estados sino también los flujos de mensajes, roles del negocio, dominio del servicio y requerimientos de calidad.

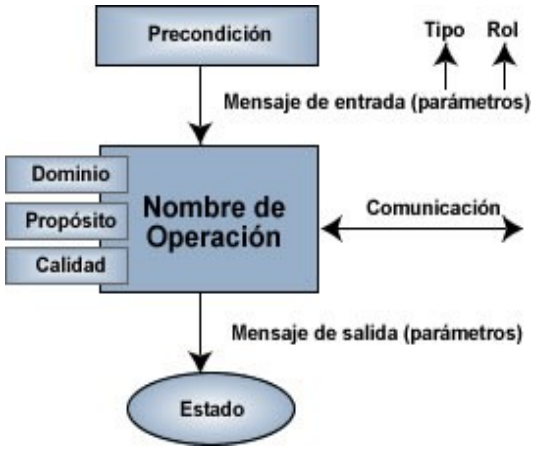

Fig.3.9 Modelo de una operación

El servicio y los requerimientos son las entradas del algoritmo de model checking y el plan es la salida. Además se consideran precondiciones y postcondiciones. En general esta técnica permite validar un modelo formal de un sistema contra su especificación lógica. El problema es descrito en una tripleta M = (S, R, L) que representa un sistema de estados finitos y una fórmula lógica temporal que expresa la especificación deseada. Encontrar el conjunto de soluciones que satisfaga que desde el estado inicial s ($s \in S$) se alcance la meta.

Este framework consta de 4 fases:

1. Especificación de la meta de planning (el dominio de interés, la meta y los límites de búsqueda) y descripción del conocimiento inicial sobre el cliente (mensaje de entrada

inicial y estado inicial) como entrada para la siguiente fase. Se inicializa la lista de condiciones y parámetros.

2. Extracción del modelo, en forma automática se seleccionan servicios Web desde el repositorio para construir el plan. El repositorio almacena servicios del dominio, ontologías y direcciones Web.

3. Ejecución del algoritmo para buscar un plan, se pueden obtener varios planes como resultado.

4. Composición física. El cliente elije el mejor plan (o se podría elegir uno en forma automática según propiedades no funcionales) y se genera un plan ejecutable, se controla la ejecución y en caso de falla se elige un plan alternativo de la fase 3.

El algoritmo propuesto usa la técnica de búsqueda Depth-First y tiene la ventaja de ser independiente del lenguaje de ontologías usado, permite la especificación de metas simples (esto debe ser mejorado), permite la generación y ejecución de planes, y es altamente escalable y reusable.

3.4.4 HTN (Hierarchical Task Network)

En este enfoque las tareas son el concepto central y el sistema de Planificación HTN[22] se basa en la descomposición de las tareas en un conjunto de subtareas y estas a su vez en un nuevo conjunto de subtareas, el proceso continúa hasta que se llega a tareas atómicas (o primitivas) las cuales pueden ser ejecutadas directamente por la invocación de algunas operaciones atómicas.

Un estado es un conjunto de átomos que son verdaderos en dicho estado. Las acciones, llamadas tareas primitivas, corresponden a las transiciones de estados. Una acción es un mapeo parcial desde un conjunto de estados a otro conjunto de estados. En este tipo de planificados se busca hacer un plan que resuelva una *tarea de red* via descomposición de tareas. Una tarea de red

es una colección de tareas que deben ser realizadas juntas con restricciones sobre el orden en el cual pueden ser realizadas dichas tareas, la forma en que las variables son instanciadas y sobre los literales que deben ser verdaderos antes o después de que cada tarea sea realizada.

Una tarea en red puede incluir una sola tarea, y en tal caso es llamada tarea primitiva y puede ser ejecutada directamente, o incluir varias tareas, en tal caso es llamada tarea compuesta y puede ser resulta por un método.

La información semántica puede ser incorporada a través de construcciones como *efectos*, *condiciones*, y *restricciones y criticos*. En el enfoque presentado solo las tareas primitivas pueden tener efectos. Las condiciones son construcciones que se usan dentro de los métodos y se definen: use-when, o condiciones filtro sirven para determinar el método a usar para resolver una tarea minimizando el factor de branching; supervised, son metas de las tareas; y unsupervised, son condiciones que deben ser verdaderas para realizar una tarea pero que pueden ser alcanzadas por otra parte de la tarea de red.

Los criticos permiten el análisis de problemas potenciales y permiten evitar fallas y backtracking, en este trabajo se utiliza una función que toma como entrada un estado y una tarea de red y da como salida otra tarea de red. Las restricciones pueden ser de dos tipos: temporales y de estado, que denotan interacciones temporales e interacciones eliminadas respectivamente. Las restricciones también son usadas para guardar información de control.

En [129] se describe SHOP2, un sistema de planning HTN que usa descripciones OWL-S de los servicios Web. También se presenta un algoritmo para traducir las descripciones OWL-S al dominio de SHOP2 y se implementa un sistema que hace el plan sobre un conjunto de descripciones OWL-S usando SHOP2 y lo ejecuta.

Una diferencia entre SHOP2 y otros planificadores HTN es que el primero hace el plan de tareas en el mismo orden en el que mas tarde serán ejecutadas, lo que hace posible conocer el

estado del mundo en cada paso del proceso de planning y habilita un mecanismo de evaluación de precondiciones que da lugar a realizar inferencias y razonamiento así como de llamar a programas externos. SHOP2 necesita conocer el dominio, su base de conocimiento consiste de operadores, que representa lo que debe ser hecho para completar una tarea primitiva, y métodos que le dicen como descomponer algunas tareas en un conjunto parcialmente ordenadode subtareas.

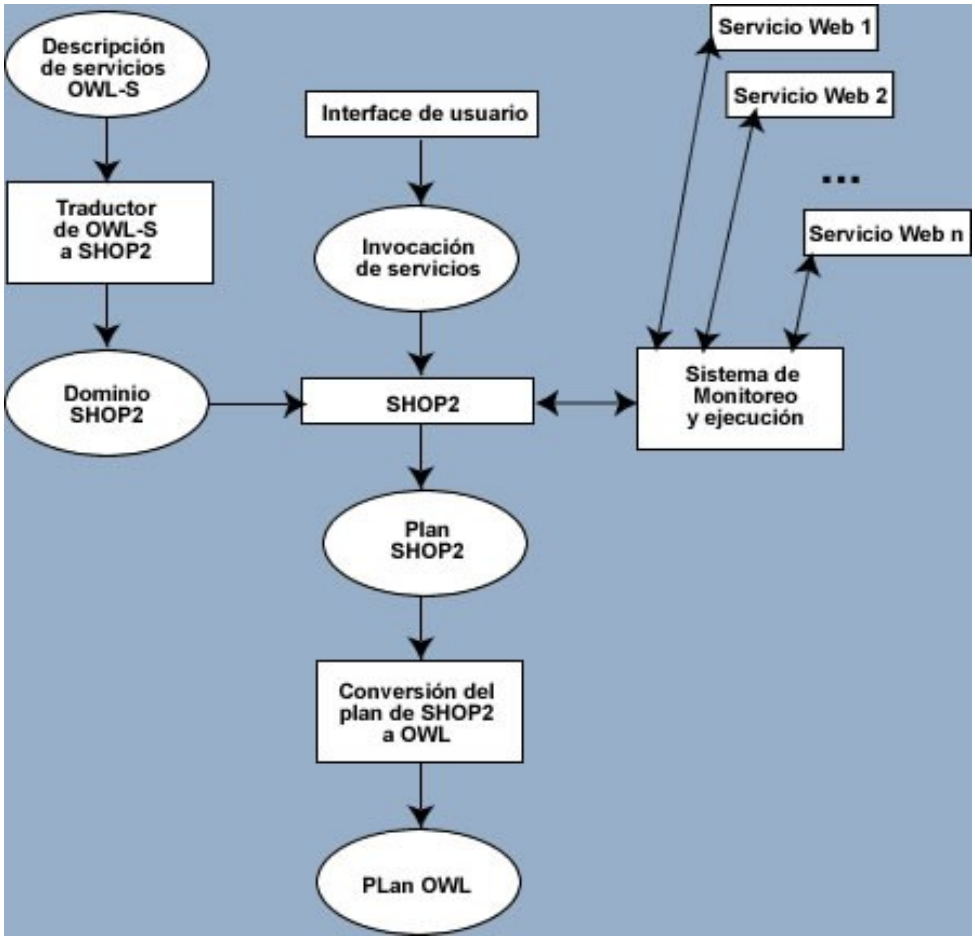

Fig.3.10 Arquitectura Shop2

Algo a destacar en la arquitectura de SHOP2, Figura 3.4, es que todos los parámetros proceden de la traducción desde las descripciones OWL-S. Los servicios seleccionados en la interfaz gráfica constituyen una lista abstracta que representan tareas que pueden ser alcanzadas de diversas maneras: éstas deben ser descompuestas en acciones (o servicios) concretas que estén disponibles y puedan ser invocadas. SHOP2 descompone las tareas de mayor nivel en subtareas

por lo que hay muchas alternativas al hacerlo, el plan generado puede realizarse con el aporte de información de los servicios que proveen información que son invocados por el planificador. El plan es una secuencia de servicios Web atómicos y ejecutables, y se le da al usuario la posibilidad de verlo y modificarlo.

3.5 Lenguajes para la especificación de la composición de servicios Web

3.5.1 BPEL4WS

Éste es un estándar para especificar procesos del negocio y los protocolos de interacción del negocio. BPEL define un estado y la lógica de coordinación entre esas interacciones y formas semánticas de tratar las condiciones excepcionales. El modelo de procesos definido por BPEL se basa en el modelo de descripción de servicios de WSDL [9].

A diferencia de WSDL en este lenguaje cada operación de cada tipo de puerto no mapea una pieza lógica separada sino que el conjunto de tipos de puertos son implementados como un solo proceso BPEL4WS [135] y los puntos de entrada correspondientes a la invocación externa de los usuarios está indicada en la descripción BPEL4WS. BPEL4WS solo soporta las operaciones de entrada y de entrada-salida de WSDL (no salida ni salida-respuesta).

Un proceso BPEL4WS es un diagrama de flujo donde cada paso del proceso es una actividad. Hay una colección de actividades primitivas: invocar una operación de un servicio Web (`<invoke>`), esperar un mensaje de una operación (`<receive>`), generar una respuesta de una operación de E/S (`<reply>`), esperar (`<wait>`), copiar datos de un lugar a otro (`<assign>`), indicar un error (`<throw>`), terminar una instancia de un servicio (`<terminate>`), o no hacer nada (`<empty>`).

Las actividades primitivas pueden ser combinadas usando estructuras provistas por el

lenguajes: definir una secuencia ordenada de pasos (<sequence>), posibilidad de decidir usando un enfoque “case-statement" (<switch>), iterar (<while>), ejecutar una de varias alternativas (<pick>), y posibilidad de indicar que una colección de pasos pueden ser ejecutados en paralelo (<flow>). Estas estructuras pueden además ser combinadas en forma recursiva.

Para manejar y recuperarse de los errores BPEL4WS define las construcciones <throw> y <catch>. Además BPEL4WS soporta la idea de *compensación*, que permite al diseñador de procesos realizar acciones compensatorias cuando ocurren acciones irreversibles. Por ejemplo en el caso de un proceso de reservación de pasajes, una vez que la reserva ha sido confirmada para poder cancelarla se deben hacer acciones explícitas.

3.5.2 WSFL (Web Services Flow Language)

WSFL es un lenguaje XML para la descripción de la composición de los servicios Web. WSFL considera dos tipos de composiciones de servicios Web: el primer tipo (modelo de flujos) especifica un patrón de uso de una colección de servicios Web, de forma tal que la composición resultante describa cómo se alcanza una meta en particular. El segundo tipo (modelo global) especifica un patrón de interacción de una colección de servicios Web, en este caso el resultado es una descripción del patrón de interacción total [16].

En [136] se describe la sintaxis del lenguajes. Un documento WSFL contiene la definición de uno o mas modelos de flujos y modelos globales, y se puede especificar opcionalmente los tipos de puerto WSDL. La raiz del documento es un elemento <definitons>.

El modelo de flujos es definido usando los elementos:

- <flowSource> y <flowSink> que definen las entradas y salidas del modelo.
- <serviceProvider> que representa los servicios participantes de la composición.

- <activity> que representa el uso de operaciones individuales de un proveedor de servicios dentro del modelo.

- <controlLink> y <dataLink> que representan conexiones de control y de datos entre las actividades.

Un ejemplo se muestra a continuación:

```
<complexType name="flowModelType">
   <sequence>
      <element name="flowSource"
                  type="wsfl:flowSourceType"
                  minOccurs="0"/>
      <element name="flowSink"
                  type="wsfl:flowSinkType"
                  minOccurs="0"/>
      <element name="serviceProvider"
                  type="wsfl:serviceProviderType"
                  minOccurs=”0” maxOccurs="unbounded"/>
      <group ref="wsfl:activityFlowGroup"/>
   </sequence>
   <attribute name="name" type="NCName" use="required"/>
   <attribute name="serviceProviderType" type="Qname”/>
</complexType>
<group name="activityFlowGroup">
   <sequence>
      <element name="export" type="wsfl:exportType"
                  minOccurs="0" maxOccurs="unbounded"/>
      <element name="activity" type="wsfl:activityType"
                  minOccurs=”0” maxOccurs="unbounded"/>
      <element name="controlLink" type="wsfl:controlLinkType"
                  minOccurs="0" maxOccurs="unbounded"/>
      <element name="dataLink" type="wsfl:dataLinkType"
                  minOccurs="0" maxOccurs="unbounded"/>
   </sequence>
</group>
```

El modelo global define las interacciones entre las partes en termino de relaciones entre las definiciones públicas de las operaciones clientes y servidores. El modelo global es una colección de proveedores de servicios que interactúan y ligan a través de los tipos de puertos de sus operaciones, indicando cual es el originador de una operación y cual responde.

El ejemplo siguiente es sobre un servicio de venta de libros.

```
<globalModel name="orderingSomeBooks"
                serviceProviderType="compoundBookOrder">
   <serviceProvider name="bookseller01" type="bookseller">
      <locator type="static" service="muchToRead.com"/>
   </serviceProvider>
   <serviceProvider name="bookseller02" type="bookseller">
      <locator type="static" service="allYouCanRead.com"/>
   </serviceProvider>
   <serviceProvider name="bookLover" type="bookLoverPublic">
      <export>
           <source portType="lifeCycle"
                     operation="spawn"/>
           <target portType="lifeCycle"
                     operation="buy"/>
      </export>
   </serviceProvider>
   <plugLink>
      <source serviceProvider="bookLover"
                portType="bookRequester"
                operation="orderDictionary"/>
      <target serviceProvider="bookseller01"
                portType="processOrder"
                operation="receiveOrder"/>
   </plugLink>
   <plugLink>
      <source serviceProvider="bookLover"
                portType="bookRequester"
                operation="orderPoetry"/>
       <target serviceProvider="bookseller02"
                portType="processOrder"
                operation="receiveOrder"/>
   </plugLink>
   <plugLink>
       <source serviceProvider="bookseller01"
                portType="processOrder"
                operation="sendBooks"/>
       <target serviceProvider="bookLover"
                portType="bookRequester"
                operation="receiveDictionary"/>
       <locator type="mobility"
                 operation="receiveOrder"
                 message="bookOrder"
                 messagePart="whatever"
                 dataField="customer"/>
   </plugLink>
   <plugLink>
       <source serviceProvider="bookseller02"
                portType="processOrder"
                operation="sendBooks"/>
       <target serviceProvider="bookLover"
                portType="bookRequester"
                operation="receivePoetry"/>
       <locator type="mobility"
```

```
                    operation="receiveOrder"
                    message="bookOrder"
                    messagePart="whatever"
                    dataField="customer"/>
    </plugLink>
</globalModel>
```

3.5.3 DAML-S / OWL-S

DAML-S [36] es un lenguaje ontológico lo suficientemente expresivo como para representar las capacidades y propiedades de los Servicios Web. Se basa en DAML+OIL y permite hacer el descubrimiento, invocación, composición y monitoreo de los servicios Web. Define una ontología para describir servicios usando clases y propiedades básicas. En DAML-S cada servicio puede ser visto como un proceso, el Modelo de Procesos es usado para controlar las interacciones con los servicios. Otras dos ontologías, ProcessOntology y ProcessControlOntology, capturan los detalles de la operación de los servicios Web. La primera describe las entradas, salidas, precondiciones, efectos y los sub-procesos que componen un servicio. La segunda es usada para monitorear la ejecución de un servicio, aunque se ha dejado de usar en versiones mas nuevas de DAML-S.

Se definen tres tipos de procesos: atómicos, no tienen sub-procesos y pueden ser ejecutados directamente; procesos simples, son una abstracción para representar procesos atómicos o compuestos; y compuestos, son procesos compuestos por otros usando estructuras de control que especifican cómo son las entradas y las salidas de los subprocesos.

En el ejemplo de Amazon.com [11] se provee un servicio Web que permite a los clientes buscar libros y otros productos y comprarlos. Los clientes pueden hacer búsquedas por autor, fabricante de un producto o director de una película. La especificación en WSDL no provee información semántica sobre las operaciones provistas, por ejemplo las entradas y salidas son todas de tipo strings. Con OWL-S se puede especificar la función de una operación así como el

tipo semántico de cada tipo de entrada/salida. OWL-S asume que las definiciones de los conceptos semánticos están disponibles a través de URIs en la Web semántica, por lo que clientes y proveedores comparten los mismos términos.

OWL-S soporta la especificación de procesos compuestos, y permite la invocación e interoperación entre los clientes y los proveedores de servicios. En el área de composición de servicios se han propuesto una variedad de enfoques para razonar sobre las entradas, salidas, precondiones y efectos usando OWL-S, soportando composición manual, semiautomática y automática de servicios Web.

En el ejemplo presentado en http://www.daml.org/services/owl-s/1.0/examples.html se proponen los documentos correspondientes a los servicios que brinda una línea aérea (Bravo Air Line). A continuación mostramos parte del service Profile donde se describe un proceso compuesto.

```
BravoAir_Process is a composite process.

It is composed of a sequence whose components are 2 atomic
processes, GetDesiredFlightDetails and SelectAvailableFlight,
and a composite process, BookFlight.
-->
-
<process:CompositeProcess rdf:ID="BravoAir_Process">
<rdfs:label> This is the top level process for BravoAir </rdfs:label>
-
<process:composedOf>
-
<process:Sequence>
-
<process:components rdf:parseType="Collection">
<process:AtomicProcess rdf:about="#GetDesiredFlightDetails"/>
<process:AtomicProcess rdf:about="#SelectAvailableFlight"/>
<process:CompositeProcess rdf:about="#BookFlight"/>
</process:components>
</process:Sequence>
</process:composedOf>
</process:CompositeProcess>
```

3.5.4 WSCI (Web Service Choreography interface)

WSCI [128] es un lenguaje para describir el comportamiento de un servicio Web en un contexto donde se intercambian mensajes (por ejemplo en un workflow). Define el flujo de mensajes intercambiados por un servicio, lo cual representa su comportamiento. Se pueden especificar las dependencias temporales y lógicas en los mensajes intercambiados, lo cual permite conocer en forma no ambigua la manera de interactuar con un servicio (colaboración). Esta descripción orientada a mensajes complementa la descripción de la interfaz provista por WSDL, describiendo la coreografía de las operaciones en un contexto. Puede usarse RDF para anotar la interfaz definida por WSCI con información semántica.

El ejemplo que se presenta incluye un servicio Web que implementa la funcionalidad de un agente de viaje y un sistema de reservación de viajes. Las interacciones entre el agente de viaje y el que viaja responden a la siguiente coreografía:

1. Un viajante solicita un viaje al agente de viajes.

2. Luego de un tiempo el viajante confirma la solicitud.

3. El agente de viajes le cobra al viajante.

```
<? xml version = "1.0" ?>
<wsdl:definitions name = "Travel Agent Dynamic Interface"
   targetNamespace = "http://example.com/consumer/TravelAgent"
   xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema"
   xmlns:tns = "http://example.com/consumer/TravelAgent"
   xmlns = "http://www.w3.org/2002/07/wsci10">                    1.

   <!-- WSDL complex types -->                                    2.
   <!-- WSDL message definitions -->
   <!-- WSDL operations and port types -->
   <!-- selectors -->

   <correlation name = "itineraryCorrelation"                     3.
            property = "tns:itineraryID">
   </correlation>

   <interface name = "TravelAgent">                               4.
```

```
    <process name = "PlanAndBookTrip"                                  5.
         instantiation = "message">                                    6.

       <sequence>                                                      7.
       <action name = "ReceiveTripOrder"                               8.
           role = "tns:TravelAgent"                                    9.
           operation = "tns:TAtoTraveler/OrderTrip">                  10.
       </action>

       <action name = "ReceiveConfirmation"                           11.
           role = "tns:TravelAgent"
           operation = "tns:TAtoTraveler/bookTickets">
          <correlate correlation="tns:itineraryCorrelation"/>         12.

         <call process = "tns:BookSeats" />                           13.
       </action>

       <action name = "SendStatement"                                 14.
           role = "tns:TravelAgent"
           operation = "tns:TAtoTraveler/SendStatement"/>
       </action>
       </sequence>
    </process>

    <process name = "BookSeats" instantiation = "other">              15.
       <action name = "bookSeats"
          role = "tns:TravelAgent"
          operation = "tns:TAtoAirline/bookSeats">
       </action>
    </process>
  </interface>
</wsdl:definitions>
```

La definición de la interfaz WSCI se encuentra dentro del elemento *wsdl:definitions*. Detalles del código

1. Incluye las definiciones de tipos, mensajes, operaciones y tipos de puerto WSDL.

2. *itineraryCorrelation* indica que los mensajes con el mismo *itineraryID* se refieren al mismo viaje, esto es importante cuando hay varias ejecuciones concurrentes.

3. *interface* contiene las definiciones WSCI de los procesos que describen el comportamiento dinámico de los servicios en el contexto de un intercambio de mensajes. Un servicio puede exponer varias interfaces las cuales son distinguidas por un nombre (name).

4. *PlanAndBookTrip* modela el comportamiento observable del agente de viajes en el contexto del sistema de reservación. *process* es la unidad básica de reuso, por lo cual una *interfaz* solo puede contener definiciones de procesos.

5. Un proceso con el atributo "`instantiation=message`" puede iniciar su ejecución tras una acción. Por ejemplo *PlanAndBookTrip* puede ser disparado cuando el mensaje *tripOrderRequest* sea recibido por el agente de viajes.

6. *sequence* indica que las acciones a continuación son ejecutadas secuencialmente.

7. *action* identifica una unidad de trabajo asociada con una operación dada. Por ejemplo la interfaz WSCI describe que el agente de viajes ejecuta la operación *OrderTrip* mientras realiza la acción *ReceiveTripOrder*.

8. Un servicio puede representar mas de un rol, por ejemplo el atributo "`role=travelAgent`" especifica que cuando la acción *ReceiveTripOrder* es ejecutada el rol del servicio es *travelAgent*.

9. Un atributo de operación asocia una acción WSCI con una operación WSDL con lo que es posible determinar el tipo de la operación (1-way o 2-way), los mensajes intercambiados y el bindings.

10. *ReceiveConfirmation* es la segunda acción en secuencia tras haber aceptado la solicitud del viajente. Cuando el agente de viajes recibe el mensaje *tripOrderRequest* se crea una nueva instancia del mismo proceso para manejarla.

11. Las acciones *ReceiveConfirmation* y *ReceiveTripOrder* debrían estar correlaciondas, lo cual se logra con *itineraryCorrelation*.

12. El proceso *BookSeats* es llamado como parte de la acción WSCI. *call* permite reusar

definiciones de procesos.

13. *SendStatement* no requiere correlación ya que referencia a una operación.

14. *BookSeats* es definido con "`instantiation=other`", lo que significa que el proceso será ejecutado cuando sea explícitamente incocado (y no tras la recepción de un mensaje).

3.5.5 WSCL (Web Services Conversation Language)

Provee una forma para modelar los procesos públicos de un servicio, habilitando a los servicios para participar en ricas interacciones. WSCL ha sido desarrollado como un complemento de WSDL. Este último especifica como enviarle mensajes a un servicio sin establecer el orden en el cual se pueden enviar esos mensajes, WSCL define la secuencia de documentos intercambiados entre los servicios Web [10].

El objetivo principal de WSCL es definir una conversación, contiene el mínimo conjunto de elementos y atributos, lo cual es suficiente para muchas conversaciones pero no para interacciónes complejas.

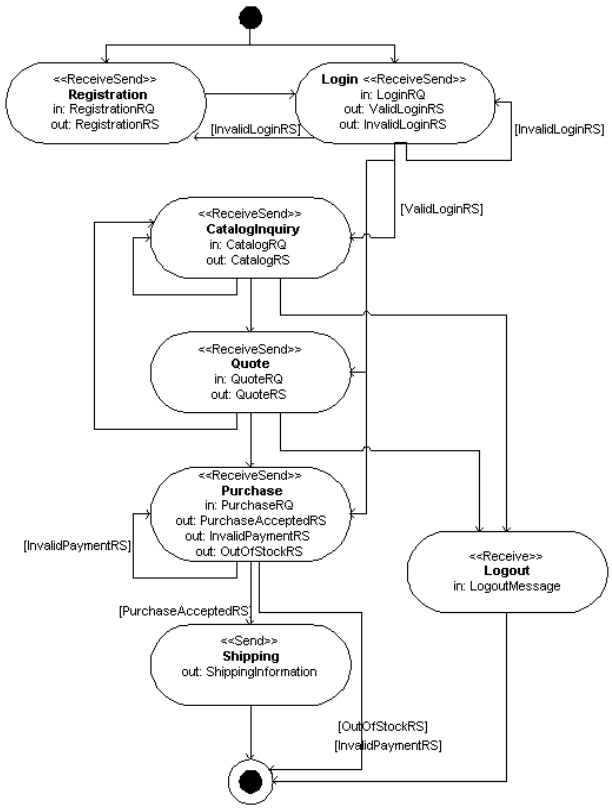

Fig. 3.11. Diagrama UML de actividad de una conversación

La Figura 3.11 muestra una conversación de compra desde la perspectiva del vendedor. La conversación se inicia con la recepción de LoginRQ o RegistrationRQ. Una vez recibido uno de estos documentos se responde con ValidLoginRS, InvalidLoginRS, o RegistrationRS, dependiendo del tipo y contenido del mensaje recibido. Parte de la especificación WSCL es mostrada a continuación:

```
<?xml version="1.0" encoding="UTF-8"?>
<Conversation name="StoreFrontServiceConversation"
        xmlns="http://www.w3.org/2002/02/wscl10"
        initialInteraction="Start"  finalInteraction="End"  >
  <ConversationInteractions>
    <Interaction interactionType="ReceiveSend" id="Login">
        <InboundXMLDocument
hrefSchema="http://conv123.org/LoginRQ.xsd"
                id="LoginRQ"/>
        <OutboundXMLDocument
hrefSchema="http://conv123.org/ValidLoginRS.xsd"
                id="ValidLoginRS"/>
        <OutboundXMLDocument
hrefSchema="http://conv123.org/InvalidLoginRS.xsd"
                id="InvalidLoginRS" />
```

```
    </Interaction>
    <Interaction interactionType="ReceiveSend" id="Registration">
        <InboundXMLDocument
href Schema="http://conv123.org/RegistrationRQ.xsd"
                id="RegistrationRQ"/>
        <OutboundXMLDocument
href Schema="http://conv123.org/RegistrationRS.xsd"
                id="RegistrationRS"/>
    </Interaction>
    <Interaction interactionType="ReceiveSend" id="Logout">
        <InboundXMLDocument
href Schema="http://conv123.org/Logout.xsd"
                id="LogoutMessage"/>
        <OutboundXMLDocument
href Schema="http://conv123.org/LogoutRS.xsd"
                id="LogoutRS"/>
    </Interaction>
    <Interaction interactionType="ReceiveSend" id="CatalogInquiry"
>
        <InboundXMLDocument
href Schema="http://conv123.org/CatalogRQ.xsd"
                id="CatalogRQ"/>
        <OutboundXMLDocument
href Schema="http://conv123.org/CatalogRS.xsd"
                id="CatalogRS" />
    </Interaction>
    <Interaction interactionType="ReceiveSend" id="Quote" >
        <InboundXMLDocument
href Schema="http://conv123.org/QuoteRQ.xsd"
                id="QuoteRQ" />
        <OutboundXMLDocument
href Schema="http://conv123.org/QuoteRS.xsd"
                id="QuoteRS" />
    </Interaction>
    <Interaction interactionType="ReceiveSend" id="Purchase" >
        <InboundXMLDocument
href Schema="http://conv123.org/PurchaseOrderRQ.xsd"
                id="PurchaseOrderRQ" />
        <OutboundXMLDocument  id="PurchaseOrderAcceptedRS"
                href Schema="http://conv123.org/PurchaseOrderAccept
edRS.xsd" />
        <OutboundXMLDocument  id="InvalidPaymentRS"
                href Schema="http://conv123.org/InvalidPaymentRS.xs
d" />
        <OutboundXMLDocument  id="OutOfStockRS"
                href Schema="http://conv123.org/OutOfStockRS.xsd"
/>
    </Interaction>
    <Interaction interactionType="Send" id="Shipping" >
        <OutboundXMLDocument  id="ShippingInformation"
                href Schema="http://conv123.org/ShippingInformation
.xsd" />
```

```
    </Interaction>
    <Interaction interactionType="Empty" id="Start" />
    <Interaction interactionType="Empty" id="End" />
  </ConversationInteractions>

  <ConversationTransitions>
    <Transition>
      <SourceInteraction href="Start"/>
      <DestinationInteraction href="Login"/>
    </Transition>
    <Transition>
      <SourceInteraction href="Start"/>
      <DestinationInteraction href="Registration"/>
    </Transition>
    <Transition>
      <SourceInteraction href="Registration"/>
      <DestinationInteraction href="Login"/>
    </Transition>
    <Transition>
      <SourceInteraction href="Login"/>
      <DestinationInteraction href="Registration"/>
      <SourceInteractionCondition href="InvalidLoginRS"/>
    </Transition>
    <Transition>
      <SourceInteraction href="Login"/>
      <DestinationInteraction href="Login"/>
      <SourceInteractionCondition href="InvalidLoginRS"/>
    </Transition>
    <Transition>
      <SourceInteraction href="Login"/>
      <DestinationInteraction href="CatalogInquiry"/>
      <SourceInteractionCondition href="ValidLoginRS"/>
    </Transition>
...
 </ConversationTransitions>
</Conversation>
```

Algunas características que deberían incorporarse a este lenguaje son [10]: descripción explícita de roles de los participantes, conversaciones múltiples entre 3 o mas participantes, expresar timeouts y otras características de calidad de servicio de las interacciones individuales, eventos que causen que se produzcan ciertas interacciones en cualquier momento dentro de una conversación, conversaciones recursivas, conversaciones dentro de otras mayores, subtipos para extender las definiciones de las conversaciones,etc.

3.5.6 BPML

Su objetivo es expresar procesos de negocios abstractos y ejecutables. Permite definir procesos entre empresas, servicios Web complejos y colaboración entre múltiples partes. Un *proceso* en BPML [137] es una composición de actividades que realizan funciones específicas. Las *actividades* tienen atributos que las caracterizan y pueden ser de un tipo simple o de un tipo compuesto. Los procesos dirigen la ejecución de estas actividades. A su vez pueden ser parte de otra composición definiéndose como parte de un proceso padre o invocándolo desde otro proceso.

El *contexto* define el entorno de ejecución de las actividades. Las actividades que se ejecutan dentro del mismo contexto intercambian información y coordinan su ejecución. Una actividad puede acceder y modificar el valor de las propiedades definidas dentro del contexto, instancias procesos anidados, sincronizar señales, etc. Las definiciones del contexto especifican el comportamiento común de las actividades que se ejecutan dentro del contexto, el cual define como se resuelven las condiciones excepcionales y errores, provee una semántica para acceder a las propiedades e intercambiar mensajes, etc. Las definiciones locales de un contexto son usadas por las entidades de ese contexto y no pueden ser vistas desde afuera.

Las *señales* son usadas para coordinar la ejecución de las actividades que se ejecutan en el mismo contexto. Por ejemplo para sincronizar el comienzo de una actividad y la completitud de otra. Las señales también reflejan condiciones que se dan tras la ejecución de otras actividades y permiten que otras actividades se ejecuten y reacciones a estas condiciones. Las señales tienen alcance únicamente dentro del contexto en el que ocurren.

Un *schedule* representa una serie de eventos en el tiempo, y se dispara a un determinado evento de tiempo. El evento puede depender de los valores de las propiedades de las instancias. El schedule puede ser modificado, cancelado o disparado frente a otros eventos. Cuando el schedule se dispara invoca a un proceso. El schedule puede también representar una restricción lanzando

una falla que cause que la actividad o proceso sea abortado.

Los *protocolos de transacciones* permiten que dos procesos que interactúan a través del intercambio de mensajes coordinen sus actividades.

Las especificaciones en BPML soportan la importación de definiciones en WSDL.

3.5.7 PDDL

Planificación de regresión [24] estimada es una técnica en la cual la búsqueda en el espacio es guiada por un estimador, y el espacio resultante es mucho mas pequeño y es representado por un grafo de regresión que representa como el conjunto de metas puede ser alcanzado con una secuencia mínima de acciones. Esta técnica ha sido aplicada a dominios de planning clásicos donde se considera que se tiene la información completa del mundo. Hay problemas que no cumplen con esta idea, tal es el caso de los servicios Web, donde un servicio realiza acciones para obtener la información que desconoce. El paradigma es válido ya que las acciones tienen precondiciones y efectos, por lo que se puede estimar la contribución de cada acción para llegar a la meta, esto es lo que se conoce como *regresión*.

Una de las cuestiones que deben resolverse en este dominio es cómo representar la transmisión de información. Otras acciones pueden representarse en forma estándar, usando PDDL:

```
(:action login
       :parameters (a - Agent pw - String)
       :precondition (password pw a)
       :effect (logged-in a))
```

PDDL no tiene forma de especificar la información que se adquiere tras ejecutar una acción. Por ejemplo el envío de un mensaje a un agente A puede tener como objetivo generar un ID para el mensaje que sea usado en el futuro, a esto se lo llama el *valor de una acción*. Esta necesidad de pasar información desde un paso del plan a otro requiere técnica para adquirir la

información. Se propone en [24] un modelo en el cual cada paso tiene asociado un campo `:value` que representa el valor retornado.

```
(:action send
       :parameters (?agt - Agent ?sent - Message)
       :value Message - id
       :precondition (Web-agent ?agt)
       :effect (rep
             ?agt (step-value this-step)
             ?sent))
```

En el caso de los servicios Web no se puede tomar la suposición que hacen los planificadores, la suposición del mundo cerrado, por lo que se deben agregar construcciones a PDDL para denotar la información que se conoce por un lado (`know-val (price bogotron)`) y la información que se aprende por otro (`Learnable Money`), así como acciones que sirven para aprender (`compute term` donde el termino contiene subexpresiones que son aprendibles) y predicados para decidir que hacer según la información obtenida (`test X PT PF` en caso que la expresión booleana X de verdadero se ejecuta la acción PT, sino se ejecuta la acción PF).

En [38] se propone una herramienta basada en PDDL para la composición automática de Servicios Web. Se presenta un método para manejar la composición de servicios Web mediante el uso de herramientas basadas en técnicas de planning. Para realizar esto introduce elementos para marcar o anotar los servicios (la descripción WSDL):

```
<serivce-annotation
serviceNamespace="http://www.borland.com/soapServices/"
name="IEmailServiceservice">
  <op-def name="SendMail" portType="IEmailService">
    <var-def var="?to" message="SendMailRequest" part="ToAddress"/>
    <var-def var="?from" message="SendMailRequest" part="FromAddress"/>
    <var-def var="?subject" message="SendMailRequest" part="ASubject"/>
    <var-def var="?msg" message="SendMailRequest" part="MsgBody"/>
    <var-def var="?result" message="SendMailResponse" part="return"/>
    <input>(have-email ?from)</input>
    <effect success-condition="(eval-JSTL '?result==0')">
      (sent-mail ?to ?subject ?msg)
    </effect>
```

```
  </op-def>
</service-annotation>
```

El problema del proceso de Composición de Servicios Web es presentado mediante PDDL, lo cual es soportado en la mayoría de los planificadores existentes y por tanto puede ser ejecutado.

3.5.8 BPSS (Business Process Specification Schema)

BPSS (Business Process Specification Schema) [138] es un marco en el que se pueden configurar y ejecutar sistemas de negocios que colaboran a través de transacciones. Esta especificación soporta transacciones de negocios y coreografías de dichas transacciones, definiendo los patrones que determinan el intercambio de documentos de negocios entre las partes para implementar las transacciones de comercio electrónico requeridas.

EbXML BPSS es un estándar para definir un framework basado en XML que permite encontrar procesos de negocios y conectarlos usando mensajes, permite representar modelos de colaboración entre procesos de negocios usando la sintaxis XML. Algunos conceptos fundamentales son:

- Colaboraciones del negocio, es un conjunto de transacciones del negocio entre las partes, donde cada parte juega uno o mas roles en dicha colaboración.

- Transacción del negocio, es una unidad atómica de trabajo entre dos partes del negocio que juegan un rol complementario en una transacción (el que requiere y el que responde).

- Flujo de documentos del negocio, una transacción se implementa como un flujo de documentos que se intercambian entre quien juega el rol de requerir y quien juega el rol de responder. Siempre hay una solicitud por un documento del negocio y opcionalmente hay un documento de respuesta según la semántica de la transacción (one-way notification vs. two-way conversation).

- Coreografía, describe el orden y las transiciones entre las transacciones y colaboraciones, puede representarse usando los conceptos del diagramade actividad UML

- Patrones, combinan flexibilidad y consistencia para facilitar el diseño, implementación y procesamiento.

```
<BusinessTransaction name="Create Order">
     <RequestingBusinessActivity name=""
                isNonRepudiationRequired="true"
                timeToAcknowledgeReceipt=”P2D"
                timeToAcknowledgeAcceptance=”P3D">
           <DocumentEnvelope
                BusinessDocument="Purchase Order"/>
     </RequestingBusinessActivity>
     <RespondingBusinessActivity name=""
                isNonRepudiationRequired="true"
           timeToAcknowledgeReceipt=”P5D">
           <DocumentEnvelope isPositiveResponse="true"
                BusinessDocument="PO
           Acknowledgement"/>
           </DocumentEnvelope>
     </RespondingBusinessActivity>
</BusinessTransaction>
```

No hay una forma de decir explícitamente cómo son los flujos de datos entre las transacciones, pero si se puede especificar la QoS para la transacción, parámetros como autenticación, reconocimiento, umbrales de tiempo, etc.

3.6 Resumen

En este capítulo comenzamos analizando dos conceptos fundamentales que se relacionan con la composición de servicios Web: **orquestación** y **coreografía**. Se distingue uno del otro principalmente por el objetivo que persigue cada uno de ellos: la orquestación se centra en la composición de servicios, concentrándose en el intercambio de mensajes o flujos de control de un proceso; en cambio la coreografía se centra en la forma en que los actores (que participan en una composición) intercambian mensajes paraejecutar varios procesos.

Para que la composición de servicios sea posible se requieren formas comunes para el intercambio de información para lo cual son fundamentales los estándares tales como WSDL, SOAP y HTTP. También surgen a partir de aquí otros estándares y propuestas de lenguajes tanto para expresar la semántica de los servicios como para especificar la composición.

También se estudió en este capítulo las características de las arquitecturas orientadas a servicios, las cuales posibilitan registrar, descubrir e invocar a los servicios dinámicamente. En este escenario hay tres componentes de software que interactúan: proveedor de servicios, registro y el cliente. El proveedor es un servidor que hospeda servicios y que provee su interfaz y se encarga de la publicación. El registro guarda la información para identificar los servicios y es usado para la búsqueda y descubrimiento de servicios. El cliente es la aplicación que solicita un servicio concreto, conectándose con el proveedor del mismo para luego operar con el servicio.

En este escenario hay algunas etapas que deben desarrollarse secuencialmente para hacer posible el uso de un servicio: descripción y anotación semántica de servicios, publicación y registración, descubrimiento, invocación, composición y ejecución. Dadas las características dinámicas del entorno y los problemas de composición descritos en este capítulo, se resalta la necesidad de contar con técnicas de composición más dinámicas y adaptivas.

En el siguiente capítulo se continúa estudiando el tema de la composición y más específicamente su automatización, se profundizan las técnicas de composición automática usando máquinas de estados finitos, workflows y técnicas de composición usando planning.

Capítulo 4:

Composición Automática

Este Capítulo se centra en el estudio de los conceptos fundamentales de la composición automática de servicios Web, se analizan los problemas y cuestiones actualmente en estudio. También se analizan las diferentes técnicas para realizar la composición usando Máquinas de Estado Finito, Planning y Workflow.

El Capítulo finaliza con un resumen donde se presentan las conclusiones.

4.1 Introducción

La composición de servicios Web surge como una respuesta a la necesidad de combinar servicios para responder a los requerimientos de los usuarios que no son alcanzables desde un único servicio. Para llevar a cabo el proceso de composición se requiere en primer lugar identificar los requerimientos del usuario, para seleccionar luego el conjunto de servicios que combinados puedan alcanzar las metas.

En [56] se define como *Síntesis de Composición* a la especificación de cómo coordinar los servicios componentes para cumplir con los requerimientos de los clientes. Tal especificación puede ser hecha automáticamente usando herramientas que implementen algoritmos de composición o manualmente. Además se debe hacer la orquestación de los servicios, lo cual se relaciona con la coordinación entre los servicios, ejecutando la especificación producida en la síntesis de composición y supervisando y monitorizando tanto los flujos de control como de datos entre los servicios implicados.

Hay algunas cuestiones que deben atenderse para llevar a cabo la composición de servicios

Web. En primer lugar la especificación del cliente, la cual puede ser representada por un modelo de interacción y variar en el grado de completitud y observabilidad. La especificación del cliente es un requerimiento atómico que representa el servicio en término de sus entradas y salidas (tanto el tipo de dato como los valores), y posiblemente las precondiciones y efectos. También pueden especificarse en término de las acciones que se espera que el servicio compuesto realice. Identificamos el primer punto en conflicto: el servicio compuesto y el requerimiento del cliente no necesariamente comparten el mismo modelo de interacción, el cliente no sabe si su solicitud es realizada por un solo servicio o por un conjunto de servicios que son coordinados para alcanzar la meta. Cuando una solicitud es realizada por un conjunto de servicios hay varias situaciones que pueden darse: la mas simple es cuando los servicios son ejecutados secuencialmente, la ejecución de un servicio comienza cuando el anterior ha sido completado, aquí el flujo de control y de datos tienen una secuencia lineal; un poco mas complejo es cuando la ejecución de servicios puede ser intercalada y realizarse en forma concurrente, los flujos de control y de datos también son mas complejos ya que hay varios servicios activos a la vez alternando la ejecución de sus operaciones.

En segundo lugar se debe considerar la forma en que será realizada la composición, en forma manual o automática. Las principales propuestas desarrolladas están constituidas por técnicas para realizar la orquestación de los servicios compuestos, donde la composición se realiza en forma manual y los servicios son representados en WSDL, representados en BPEL4WS (Business Process Execution Language for Web Services) y CDL (Choreography Description Language).

Las propuestas actuales relacionadas con la composición no alcanzan a cumplir con las expectativas, en especial por las dificultades para flexibilizar la forma de realizar la composición de forma tal que se reduzca la necesidad de intervención manual.

Los principales problemas para implementar la composición automática de servicios son la selección y la interoperación de los mismos. Como se menciona en [19] el composer debe centrarse en el qué de la composición, y no en cómo interactúan los servicios, además de ser

transparente al usuario. Es así que para que la selección de los servicios sea la adecuada se deben considerar, no solo las características sintácticas (como el número de parámetros de las operaciones) sino que también debe considerarse la semántica de los servicios. Para ello las ontologías son de gran utilidad, ya que constituyen una conceptualización compartida y capturan la semántica de los servicios Web.

La tercera cuestión se refiere a la arquitectura de orquestación la cual puede realizarse en forma distribuida, como los sistemas peer-to-peer, o en forma centralizada, donde existe un mediador que realiza la orquestación. Esto será retomado en el capítulo 5.

En el framework presentado en [19], la composición de servicios se centra en tres elementos:

- Modelo de Composabilidad: composabilidad se refiere al proceso de verificar que si un servicio va a ser usado en una composición, éste debe poder interactuar con cada uno de los otros servicios en la composición. Se propone un modelo para comparar sintáctica y semánticamente los servicios.

- Generación automática de servicios compuestos: propone una técnica para generar la descripción de un servicio compuesto usando como entrada una lista de operaciones a ser realizadas.

- Prototipo de implementación: usando UDDI, WSDL y SOAP.

Este modelo usa reglas de composición sintácticas, semánticas y ontologías para hacer la composición. Se define el elemento Composer de la siguiente manera:

El Composer es quien elije los servicios a componer basado en el qué de los mismos y en los parámetros. La descripción de los servicios es hecha en WSDL y se define una ontología en DAML+OIL. Un servicio Web es definido instanciando cada concepto en la ontología. La funcionalidad de un servicio Web es accesible a través de sus operaciones. A sus vez las

operaciones tienen distintos modos de operación:

1. Notificación: el servicio envía un mensaje de salida y no espera respuesta
2. One-way: recibe un mensaje de entrada, lo consume y no produce mensajes de salida
3. Solicitud-respuesta: genera un mensaje de salida y recibe un mensaje de entrada
4. Requerimiento-respuesta: recibe un mensaje de entrada, lo procesa y envía un mensaje de salida

Los parámetros en un mensaje tienen un nombre, tipo, unidad y rol. Cada operación es descripta por su propósito (definido en términos de su función, sinónimos y especialización) y categoría (definida en términos de su dominio, sinónimos y especialización). Además puede definirse la calidad de la operación sobre los parámetros costo, seguridad y privacidad. Para verificar si dos operaciones se pueden componer, se deben analizar las reglas:

- Sintácticas: modo y binding (protocolos que soportan las operaciones para el intercambio de mensajes).
- Y semánticas: analizar los mensajes (número de parámetros, tipos, roles, unidades), las operaciones, las propiedades cualitativas y la sanidad de la composición.

Las fases de este modelo están representadas en la Figura 4.1, éstas son:

- Especificación: se realizan descripciones a alto nivel de la composición deseada en CSSL (Composite Service Specification Language). CSSL extiende WSDL permitiendo agregar semántica y permite describir el flujo de control entre las operaciones del servicio compuesto.
- Matchmaking: consiste en la generación de planes de composición usando las especificaciones en el repositorio UDDI (donde están almacenadas las descripciones de los servicios en WSDL).

- Selección: se realiza la selección de servicios que brindan la funcionalidad deseada usando los parámetros de calidad.

- Generación: se genera la descripción del servicio compuesto (en WSFL, XLANG, BPEL4WS).

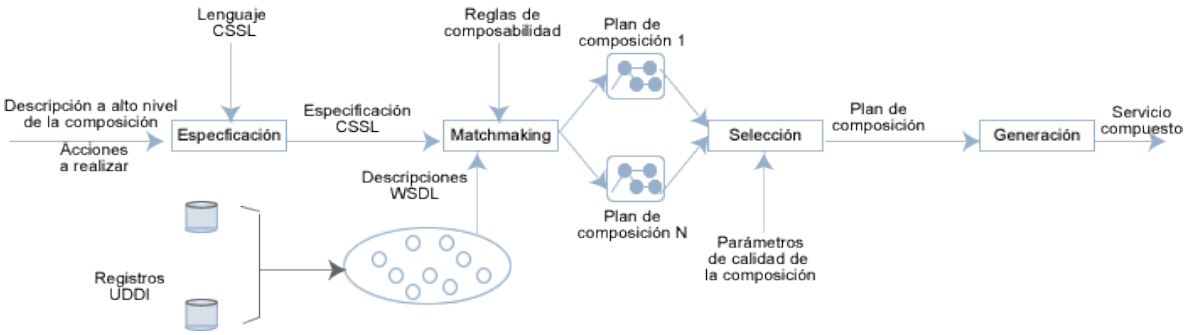

Fig. 4.1. Etapas del enfoque de composición de servicios propuesto.

En [56] se presentan dos posibles enfoques para realizar la síntesis de composición: para un solo uso o para múltiples usos. En el primer enfoque, para un solo uso descrito en [57], el cliente especifica el comportamiento del servicio que desea, el algoritmo de composición crea un servicio compuesto y éste es ejecutado una vez para cumplir con el requerimiento. Si en el futuro alguien quiere ejecutar un servicio similar, el algoritmo de composición debe generar nuevamente el servicio compuesto (que posiblemente no será el mismo). En el segundo enfoque, para múltiples usos presentado en [60], el comportamiento deseado es especificado como un sistema de transición reusable. El algoritmo de composición es ejecutado una vez para crear el servicio compuesto y el servicio resultante puede ser ejecutado muchas veces por diferentes clientes.

En [95] se define el concepto de ***servicio componente*** que empaqueta juntos servicios complejos y define una manera consistente y uniforme de presentar sus interfaces y operaciones en forma de definición de una clase abstracta. Los servicios que son usados en el contexto de un servicio componente son llamados ***servicios constituyentes***. Los servicios componentes son construidos, reusados, especializados o extendidos y pueden ser publicados e invocados como si fueran servicios Web. La interfaz de un servicio componente es pública e incluye los mensajes y operaciones que realiza, pero la lógica del negocio que está representada por los servicios

contenidos es privada.

También se presenta un framework para manejar el ciclo de vida de los servicios componentes desde su definición abstracta, planificación, construcción y ejecución. La composición lógica define cómo los servicios componentes pueden ser combinados, sincronizados y coordinados, permitiendo expresar la lógica del negocio. El estándar que se propone para representarla es BPEL. La composición lógica se refiere a la forma en que los servicios componentes son construidos en término de los servicios constituyentes, para lo cual se asume que los servicios disponibles son descritos en WSDL. La composición lógica considera dos tipo de construcciones:

- Tipo de composición, indica si los servicios constituyentes tienen que ser ejecutados en algún orden, en serie o en paralelo, o si los servicios pueden ser invocados en forma alternativa.

- Dependencia de mensajes, la dependencia de los mensajes limita el orden en que pueden ser ejecutados los servicios.

En este framework se identifican tres tipos de composición con distinto grado de flexibilidad:

- Composición explorativa: la composición de servicios es realizada sobre los requerimientos expresados por los clientes. Los requerimientos son expresados en un lenguaje de alto nivel y luego se deben seleccionar de entre los servicios registrados en el UDDI, dando lugar a varias alternativas de composición. La selección de una de ellas será determinada por los parámetros no funcionales como costo, disponibilidad y performance. Este tipo de composición requiere la orquestación de los servicios que la integran. Este enfoque está representado en la Figura 4.2.

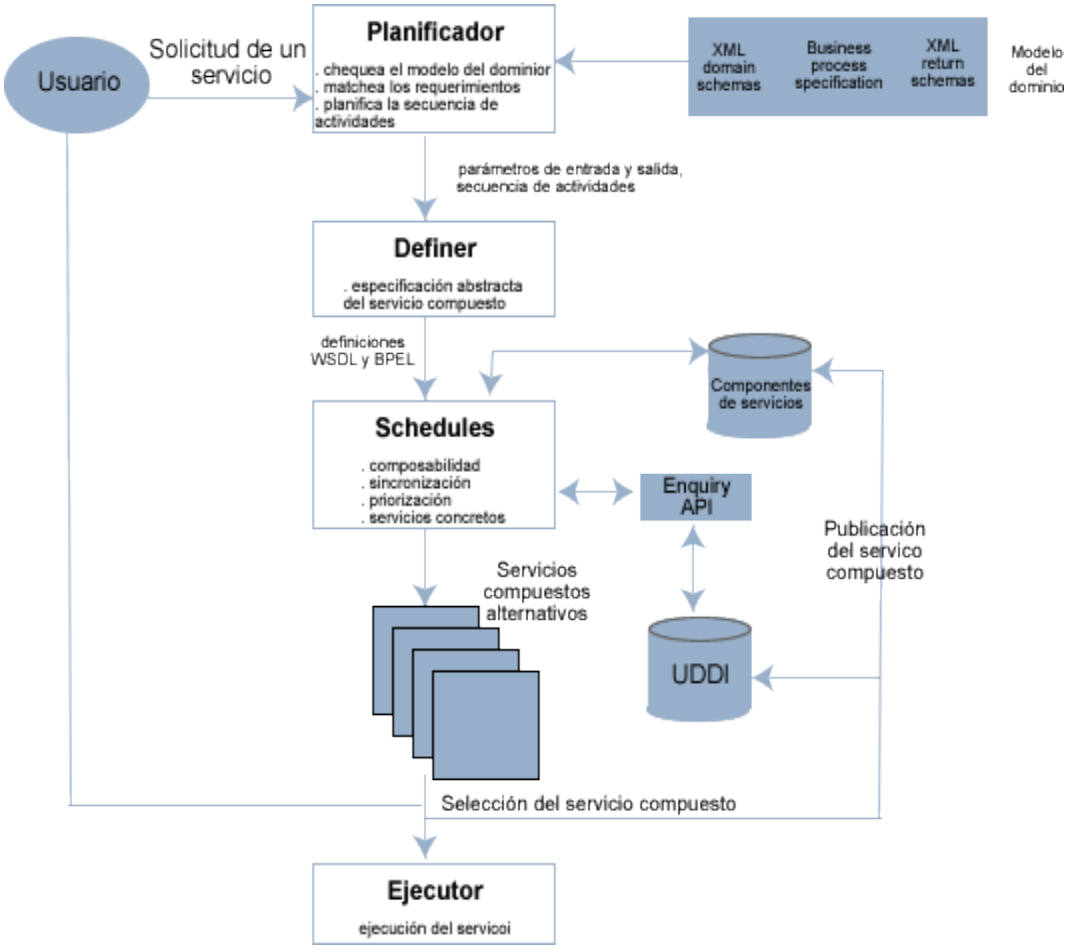

Fig. 4.2. Fases de la composición de servicios explorativa.

- Composición semi-fija: requiere que la composición sea expresada estáticamente pero los servicios serán ligados durante la ejecución, es decir, que al ser invocado un servicio compuesto se realizará la composición basada en los servicios constituyentes previamente definidos y los servicios actualmente disponibles. La definición del servicio componente puede ser almacenada en un repositorio y volver a usarse en cualquier momento. Este enfoque se muestra en la Figura 4.3.

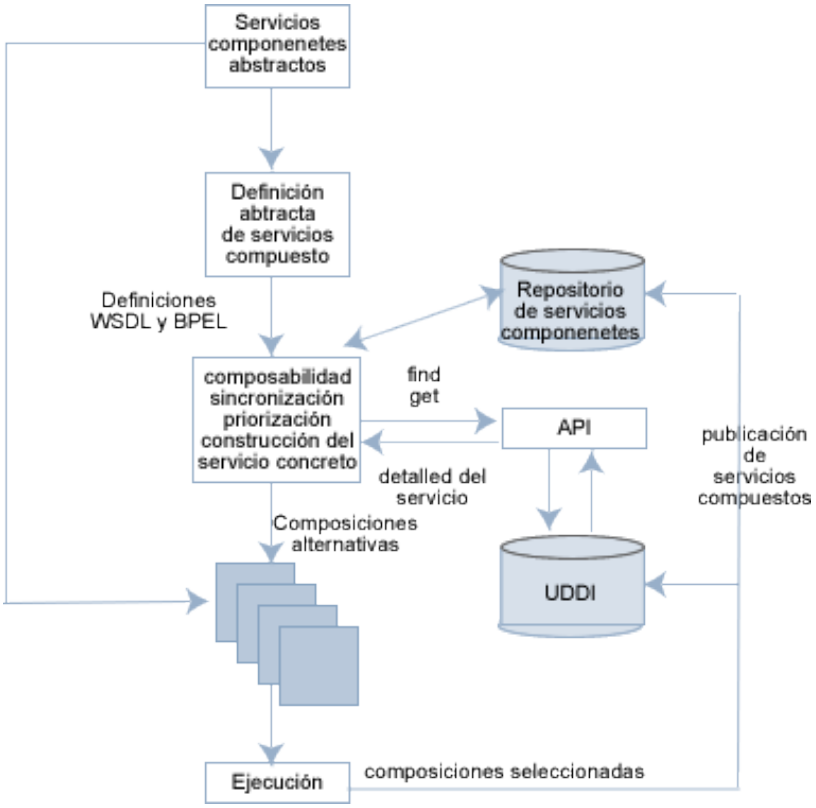

Fig. 4.3. Fases de la composición de servicios semi-fija.

- Composición fija: los servicios constituyentes de un servicio compuesto son identificados previamente y de manera estática.

Este trabajo asume que la composición es realizada según alguno de los últimos dos enfoques. El ciclo de vida que se pretende automatizar está constituido por las siguientes fases:

- Planificación: en esta fase se determinan las operaciones de los servicios que deben realizarse para alcanzar los requerimientos del cliente.

- Definición: en esta fase se describe en forma abstracta el servicio compuesto. Para lo cual se usan las descripciones en WSDL, lenguajes de orquestación como BPEL y pueden usarse las clases abstractas de los servicios componentes.

- Scheduling: en esta fase se determina cuando y cómo deben ejecutarse los servicios. En esta fase se concretiza la composición, verificando la composabilidad de los servicios (correlación de mensajes y operaciones). Varias alternativas de composición pueden ser

ofrecidas para su elección, las cuales son especificadas en SSL (Service Scheduling Language).

- Construcción: esta fase da como resultado una composición concreta en SCSL, con la elección de los servicios constituyentes disponibles, lista para ser ejecutada.

- Ejecución: implementa la composición con la ejecución de los servicios implicados. Para esto se genera un grafo en SGEG (Service Composition Execution Graph) donde cada nodo en el grafo es un servicio compuesto y sus nodos hijos representan los servicios constituyentes.

En [59] se presenta un modelo llamado ***Colombo*** que combina cuatro aspectos fundamentales de los servicios Web:

- El *estado del mundo*, es visto como una base de datos que instancia esquemas de una base de datos relacional, esquemas del mundo. Esto es similar a los fluentes en el modelo OWL-S [13] y es común en cálculo de situación [16].

- Los *procesos atómicos* (operaciones), pueden acceder y modificar el estado del mundo, y pueden tener efectos condicionales y no determinísticos.

- El *pasaje de mensajes*, incluye la noción de puerto y link de los estándares como WSDL y BPEL4WS [2]) .

- El *comportamiento de los servicios Web* (incluye múltiples invocaciones a procesos atómicos y pasaje de mensajes) es especificado usando sistemas de transición de estados [60,61,62].

Los primeros tres elementos coinciden con el lenguaje de ontologías SWSL (Semantic Web Service Language) [63]. Los cuatro elementos permiten modelar los procesos internos de un servicio Web, usando la base de datos relacional. Tal como lo hace Colombo se pueden modelar las relaciones individuales que son accesibles solo por un servicio o por un conjunto de ellos. Los

últimos dos, el patrón de comportamiento y secuencia en el pasaje de mensajes, se relacionan con los conceptos de orquestación y coreografía definidos en WSMO (Web Service Modeling Ontology) [62].

Colombo provee un enfoque para aprovechar la característica de los servicios para acceder a los datos y filtrar datos, para lo cual se hacen las siguientes suposiciones:

- Cada instancia de un servicio tiene un almacenamiento local para capturar los valores de los parámetros de los mensajes de entrada y salida de los procesos atómicos y para setear los parámetros de los mensajes de entrada y salida de los mismos. Las opciones en un servicio Web serán elegidas según los valores almacenados localmente.

- Considera restricciones de integridad sobre los estados del mundo.

Un cliente interactúa con un servicio Web enviando y recibiendo mensajes hasta alcanzar una cierta situación, por lo que el comportamiento del cliente también puede ser representado como un sistema de transición. Un ejemplo de tal representación puede observarse en la Figura 4.4.

Para apuntar al problema de composición automática de servicios se introduce el concepto de **meta**, que denota el comportamiento deseado del nuevo servicio, el cual es especificado como transiciones de estado de un servicio interactuando con un cliente que invoca procesos atómicos. Se propone construir un mediador que usando los mensajes interactúe con servicios ya existentes (registrados en repositorio UDDI) tal que el comportamiento total sea orquestado por éste.

Colombo usa y extiende las técnicas basadas en PDL (Propositional Dynamic Logic) presentadas en [60] permitiendo automatizar la composición de servicios.

Fig. 4.4. Sistema de transición de estados del servicio meta.

En [96] se enfocan en la composición automática de servicios que son vistos como fuentes de datos. Los sistemas de este tipo tradicionales toman la consulta del usuario y la re-formulan en una combinación de consultas a las fuentes de datos que respondan la consulta del usuario. La extensión presentada en el framework consiste en permitir al usuario especificar una clase de consulta a la que el servicio debería responder.

Este enfoque se divide en tres pasos:

- Los expertos del dominio definen un conjunto de predicados del dominio y describen los servicios Web como vistas sobre los predicados del dominio. A su vez el sistema mediador describe las fuentes de datos como vistas de los predicados del dominio.

- Los expertos del dominio también deben proveer al mediador de las reglas sobre las dependencias funcionales en el modelo del dominio.

- El mediador utiliza un algoritmo de regla inversa para generar nuevos servicios. El algoritmo propuesto permite optimizar los planes de integración para crear los servicios compuestos reduciendo el número de consultas enviadas a un servicio Web. Además inserta reglas sensitivas que permiten optimizar la ejecución de los servicios compuestos.

La plataforma de composición de servicios StarWSCoP (Star Web Services Composition Platform) [145] se enfoca en al composición dinámica de los servicios, la cual contempla cuatro etapas: publicación de los servicios en el registro; división de la solicitud en requerimientos menores que son enviados como requerimientos SOAP al registro para encontrar los servicios que los satisfagan; el registro retorna un conjunto de servicios concretos; y por último el motor realiza la composición enviando mensajes SOAP a los servicios concretos. Para llevar a cabo este proceso la plataforma incluye varios módulos: un sistema inteligente que descompone los requerimientos de los usuarios en descripciones abstractas de servicios; un registro de servicios que provee un repositorio de servicios; un motor para descubrir servicios apropiados según los requerimientos; un motor de composición que planifica la ejecución del servicio compuesto; traductores para mejorar la interoperabilidad de servicios heterogéneos de diferentes proveedores; una librería de ejecución que almacena información sobre los servicios compuestos ejecutados; una estimación de calidad para estimar la QoS de un servicio compuesto; y un monitor de eventos.

En esta plataforma WSDL es mejorado con atributos de QoS, tales como tiempo, costo y confiabilidad, para permitir la composición dinámica de los servicios. Además se incorpora el uso de ontologías al UDDI para lograr la conformidad semántica de los servicios Web. Un segundo traductor (o wrapper) permite resolver inconsistencias en los tipos de datos, políticas de seguridad o contenido de los servicios Web. Este adiciona un administrador de la comunicación para traducir los mensajes entre diferentes protocolos de transporte tales como HTTP o SMTP,un administrador de la seguridad para soportar firewalls y manejar la autenticación y autorización, una administrador de contenido para convertir entre diferentes representaciones de los documentos, un administrador de conversaciones para manejar las diferencias conversacionales y un monitor de la QoS para controlar las métricas de calidad de los servicios Web.

SELF-SERV [145] es una plataforma que se basa en lenguajes declarativos para componer servicios según los diagramas de estado modelados en Unified Modeling Language (UML). En esta plataforma se define el concepto de *comunidad* para apuntar a cuestiones de la composición de un gran número de servicios Web dinámicos, las comunidades son contenedores de servicios alternativos.

La ejecución del servicio compuesto es coordinado por varios peer llamados coordinadores, los cuales se encargan de iniciar y controlar el estado asociado mientras colaboran con otros para manejar la ejecución del servicio. El conocimiento requerido en tiempo de ejecución por cada coordinador es extraído estáticamente desde el diagrama de estados y es representado en tablas de ruteo, las cuales contienen precondiciones y postprocesamiento. Las primeras son usadas para determinar cuando un servicio debería ser ejecutado mientras que las segundas son usadas para determinar que debería ser hecho luego de la ejecución del servicio.

4.2 *Composición* usando Máquinas de Estados Finitos (Mealy Machines)

La tendencia a la computación distribuida sobre una red y al incremento de la complejidad de la comunicación, han incentivado la investigación en búsqueda de métodos que permitan manejar y reducir la complejidad.

Las redes de Petri constituyen un modelo que no tiene gran poder expresivo en comparación con algunos lenguajes. Las Máquinas de estado finito son una alternativa que permiten describir los procesos componentes y los canales de interconexión, pero solo ciertos protocolos pueden ser descritos, por ejemplo no puede describirse un protocolo que permite un número arbitrario de mensajes.

En el modelo presentado en [104] se usan máquinas de estados finitos para representar procesos y las colas implícitas para representar canales, a través de los cuales un proceso envía mensajes a otro. Las colas tienen capacidad ilimitada para permitir un número arbitrario de

mensajes en tránsito. En este modelo se supone que cada proceso es representado por una máquina de estados finitos y los canales son usados para comunicación.

En la representación gráfica la transmisión de un mensajes se representa con el signo ***menos(-)***, para denotar el envío, y con un signo ***mas(+)*** para representar la recepción de un mensaje. Cuando un proceso está en un cierto estado desde el cual sale un arco rotulado ***m***, puede cambiar de estado enviando un mensaje ***m*** al proceso destino a través del canal que los conecta. No hay suposiciones del tiempo que lleva procesar el mensaje en un cierto estado antes de enviar otro mensaje, ni tampoco del tiempo que el mensaje tarda en atravesar el canal hasta llegar al destino.

Colombo [59] es un sistema que usa un modelo basado en autómatas para representar el comportamiento interno de los servicios Web donde las transiciones corresponden a la ejecución de procesos atómicos, envío y recepción de mensajes.

En [105] se representa el comportamiento de un servicio Web como un sistema de transición de estados (LTS), el cual tras la recepción de un requerimiento puede dar una respuesta adecuada (de error o el resultado de ejecutar una acción). Los servicios son representados por un conjunto de estados S (gráficamente representados por círculos) y transiciones entre ellos (representados por flechas) y acciones (rótulos de las flechas) que pueden ser envío o recepción de mensajes. Además una función de transición determina a qué estado se puede pasar desde un estado particular al ejecutarse cada una de las posibles acciones. La Figura 4.5. muestra un ejemplo de dicha representación.

Fig. 4.5. Sistema de transición de estados de un servicio simple

En dicho trabajo se analizan algunas características de la composición de servicios Web como compatibilidad y sustitutabilidad, y se presentan algunas variantes del modelo propuesto en los cuales se incorporan: el comportamiento interno y/o externo no determinístico y comunicación asincrónica.

El modelo OWL-S se enfoca en qué hace un servicio o una composición en término de las entradas y salidas de los servicios y en su impacto en el mundo, pero no se enfoca en cómo debería interactuar un servicio en una composición, la noción *conversación* apunta a esta cuestión. En [106] se supone que existe un conjunto de nombres de servicios (*P*) y un conjunto de clases de mensajes (*M*), donde cada clase tiene un nombre de servicio como fuente y un nombre de servicio como destino. Un esquema de composición es un par *(P,M)* que tiene un conjunto de nombres de servicios y un conjunto de clases de mensajes. Un servicio compuesto para un esquema de composición dado (*P,M*) es una asociación de los servicios actuales a cada uno de los nombres de servicios en el conjunto *P*. Cada servicio tiene una cola FIFO para almacenar los mensajes de entrada, lo cual permite representarla comunicación asincrónica.

Dado el servicio compuesto *S* sobre el esquema de composición *(P,M)*, una *Conversación* sobre *S* es la secuencia de mensajes enviados durante una ejecución exitosa de *S*. Se define también *Protocolo de Conversación* sobre el esquema de composición *(P,M)* como una máquina de estados finitos sobre el conjunto de clases de mensajes en *M*. Esto puede verse representado en la Figura 4.6. también se define la noción de *servicio Mealy* como la implementación de un nombre de servicio que define algunas propiedades del comportamiento del servicio y las restricciones locales del comportamiento que se deberían satisfacer.

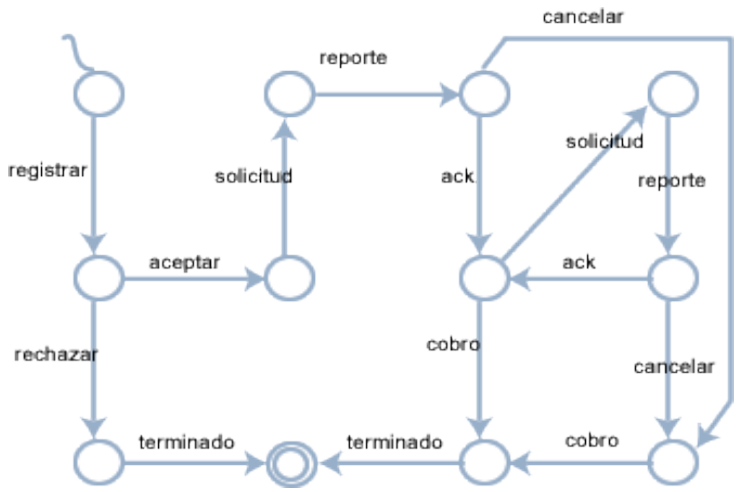

Fig. 4.6. Ejemplo de un protocolo de conversación

En [98] se presenta un framework para modelar y especificar el comportamiento global de la composición de servicios Web, dentro del cual los peer que corresponden a servicios individuales se comunican a través de mensajes asincrónicos y cada peer tiene una cola de mensajes entrantes. Un *observador* global lleva la pista de los mensajes intercambiados, lo cual constituye la noción de *conversación* que es usada para diseñar la composición de servicios.

Los peers son representados como máquinas de Mealy, es decir máquinas de estado finito con entradas y salidas, las conversaciones pueden tener un comportamiento inesperado dependiendo del entorno.La Figura 4.7 muestra un ejemplo de dicha representación.

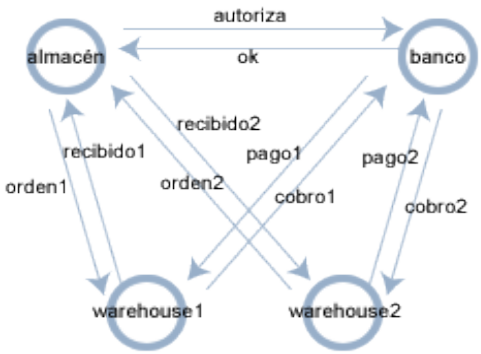

Fig. 4.7. Representación de un servicio compuesto.

Los mensajes son organizados dentro de colecciones de *clases de mensajes* que sirven para simplificar y organizar la especificación de las acciones. Una clase de mensaje consiste de un

nombre y un conjunto finito de acciones. Un mensaje de una clase consiste de un identificador del servicio al que pertenece y un identificador del mensaje en si mismo, el emisor, el receptor y una función que asigna a cada atributo un valor del tipo apropiado. La implementación de un peer puede ser vista como un programa que decide, basado en los mensajes recibidos y los mensajes enviados, si un nuevo mensaje debería ser enviado y/o si la sesión debe terminar.

En este trabajo se formaliza la noción de *esquema de composición* (e-composition schema) incluyendo un conjunto de mensajes, un conjunto de peers y un conjunto de canales de comunicación.

Para hacer la composición de servicios se enfoca en primer lugar el comportamiento global de la composición, limitando la forma en que cada servicio interactúa con otros, lo cual se relaciona con la coordinación de los mensajes. Para resolver esto se definen familias de secuencias de mensajes entre los peers. La implementación de Mealy de un peers, máquinas de estados finitos, reacciona según la clase de mensajes sin considerar su contenido. En la implementación propuesta la comunicación es asincrónica utilizando un canal común y cada peer utiliza una cola para almacenar los mensajes que llegan hasta ser procesados.

4.3 Composición usando métodos de planning

Los métodos de composición basados en planning clásico son afectados por la suposición de que el comportamiento de los servicios Web es determinístico. Por esto se requiere un overhead para recuperarse del comportamiento no esperado de los servicios.

Muchos de los métodos de planning propuestos no son escalables, además son ineficientes para manejar las descripciones de los servicios Web, específicamente las precondiciones y efectos los cuales pueden ser especificados usando lógica de primer orden. La propuesta presentada en [45] llamada Haley, distingue los servicios como procesos y aprovecha la jerarquía natural de los mismos para hacer la composición. En la Figura 4.8 puede observarse la representación propuesta. Este sistema se basa en lógica de primer orden y puede manejar la incertidumbre a la vez que

garantiza la optimización de los costos.

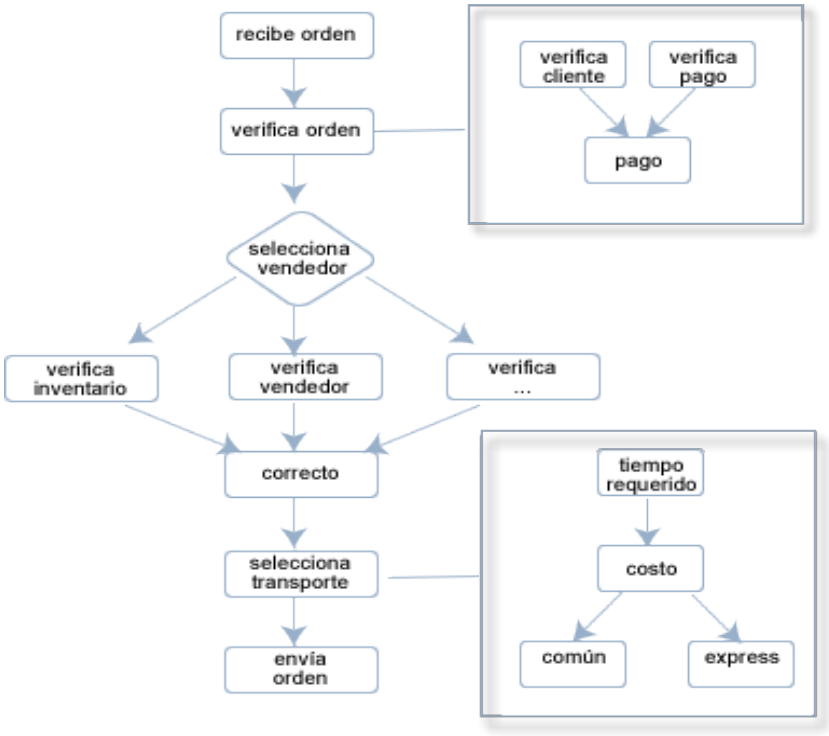

Fig. 4.8. Escenario para manejar órdenes donde el servicio VerificarOrden es un proceso.

Este enfoque jerárquico se basa en la propuesta presentada en [49] y mejora la propuesta [50] al permitir realizar la composición a nivel lógico. El sistema Haley usa lógica de primer orden para representar las precondiciones y efectos de los servicios. Para esto se modela cada nivel de la jerarquía de procesos usando FO-SMDP (First order semi-Markov decision process) que extiende SMDP para operar directamente sobre las sentencias de lógica de primer orden, y provee así una visión lógica del espacio de estados. Además provee métodos para derivar el modelo de parámetros de alto nivel desde los parámetros de menor nivel.

En el modelo Markov clásico (MDP) se modela el entorno a través de la representación del estado, las acciones que representan invocaciones a servicios Web, una función de transición que considera los efectos inciertos usando una distribución probabilística y una función que representa el costo de las invocaciones. Hallar una solución consiste en computar un *policy,* el cual mapea una acción que resulte óptima a cada estado, siendo un estado la asignación de valores a las variables. Cada acción afecta a algunas de ellas y permite pasar a un nuevo estado si se cumplen las precondiciones. Para resolver un MDP se deben enumerar explícitamente todos los pares

estado-acciones, lo cual es un desafío cuando se componen grandes procesos.

Las descripciones estándares tales como OWL-S y SA-WSDL expresan precondiciones y efectos de los servicios usando lenguajes basados en lógica de primer orden tal como Rule ML. Aunque es posible aplicar MDP para realizar la composición de servicios haciendo proposiciones con las descripciones de los servicios, el número de proposiciones crece exponencialmente con el número de servicios, por lo que es necesario un framework que opere simbólicamente sobre las descripciones en lógica de primer orden.

En el framework se usa lógica de primer orden para representar acciones y cambios así como para razonar sobre ellos. Los elementos que intervienen son:

- Acciones: son términos en lógica de primer orden que representan las invocaciones a los procesos.
- Situación: representa el estado del mundo, es una secuencia de acciones.
- Fluentes*:* representan los cambios desde una situación a otra a causa de las acciones.
- Elección: una acción puede dividirse en múltiples acciones determinísticas, la elección puede hacerse en forma aleatoria.
- Probabilidad de la elección: es la probabilidad de que una acción pueda ocurrir.
- Precondiciones: son axiomas que caracterizan las precondiciones de una acción.
- Efectos: son axiomas que representan los efectos o estado sucesor de los fluentes.
- Regresión: es un mecanismo para proveer consecuencias en cálculo de situación.

SMDP modela explícitamente la evolución del sistema en el tiempo, representando el tiempo que transcurre en un estado particular mientras se realiza una acción según una distribución probabilística. El resultado es un policy que le asigna a cada estado del proceso acciones que se

espera sean óptimas para el período considerado. Extender este modelo a lógica de primer orden implica evitar enumerar los pares estados-acciones y permitir operar directamente sobre las precondiciones y efectos de los servicios Web en lógica de primer orden.

4.3.1 Planificación usando Cálculo de situación

Cálculo de Situación es un lenguaje de primer orden para axiomatizar el mundo dinámico. Es una formalización de cálculo de predicados para describir estados, acciones y efectos de las acciones sobre los estados. Es una técnica de planning que responde a la pregunta "Existe un estado tal que satisfaga la meta?".

Las **acciones** son representadas por una función con un argumento, permiten expresar los hechos que son ciertos en cada estado, una **situación** denota una secuencia de acciones, desde un estado se pasa a un nuevo estado o situación al ejecutar una acción, también se pueden expresar los efectos de las acciones como pares de literales-acciones, y los **fluentes** son relaciones cuyo valor de verdad pueden variar de estado en estado [68].

En [97] se identifica el problema de composición de servicios como un problema de planning donde las acciones son los servicios. En [99] los mismos autores muestran como un servicio puede ser visto como un conjunto de operaciones, que representan la evolución de una tarea compleja, y pueden ser tratados como acciones primitivas al realizar el plan. El problema de composición, al igual que el problema de planning, tiene la dificultad de contar con información incompleta, por lo que se requieren servicios que busquen información.

En general los planes hallados constituyen un espacio de búsqueda corto pero ancho, el cual puede ser recortado con las entradas y restricciones del usuario. Además se argumenta que el número de actividades que los usuarios pueden querer hacer en la Web pueden ser vistos como la personalización de procedimientos genéricos reusables por lo que la mayor complejidad recae en la selección de los servicios adecuados según las restricciones y preferencias de los usuarios. En este trabajo se pretende construir dichos procedimientos para que puedan ser compartidos en

ontologías (en DAML-S) de forma que muchos usuarios puedan acceder a ellos. Un usuario seleccionaría de esta manera un procedimiento desde la ontología para realizar una tarea específica. El agente que lo ejecuta lo personalizará según las preferencias del usuario, el estado del mundo y los servicios disponibles. Para esto se propone una adaptación del lenguaje Golog.

Golog es un lenguaje construido sobre la base del Cálculo de Situación, donde el mundo es descrito por funciones y relaciones relativas a una situación, y donde es posible pasar de una situación a otra a través de las acciones. El lenguaje es extendido con la incorporación de nuevos predicados como *deseable*, *posible* y *legal*, que permiten incorporar las restricciones de los usuarios. También se agregan otras construcciones para indicar el orden de ejecución de las acciones, lo cual permitiría alcanzar las precondiciones para posibilitar que la siguiente acción sea ejecutada.

También se definen la noción de programas *autosuficientes*, los cuales son ejecutados con una cantidad mínima de suposiciones sobre el estado inicial de los agentes de información o el estado del mundo.

Ejemplo (extraído de [97])

El ejemplo consiste en la planificación de un viaje para asistir a una conferencia. Si esto se realizara usando los servicios existentes se debería primero obtener información sobre la conferencia para saber donde se realiza y en qué fecha y hora. Sabiendo donde queda se debe seleccionar el medio de transporte adecuado. Si el medio es aéreo se debe verificar si hay vuelos usando los servicios Web, hacer la reserva y determinar cómo se llega al aeropuerto a través de otros servicios Web, por ejemplo podría rentar un auto. Finalmente se debe reservar un lugar en la conferencia, etc.

Se ha definido un procedimiento general que captura los aspectos del ejemplo: se selecciona y reserva un medio de transporte (aéreo o auto), hotel, transporte local, armado del itinerario y actualización de las cuentas on-line. Luego se personalizará con las restricciones del

usuario.

Procedimiento genérico en Golog para reservar un vuelo:

```
proc(bookRAirTicket(O, D, D1, D2),
[
   poss(searchForRFlight(O, D, D1,D2)) ?,
   searchForRFlight(O, D, D1, D2),
   [ pi(price,
             [ rflight(ID, price) ?,
                (price < usermaxprice) ?,
                buyRAirTicket(ID, price) ])
]
]);
```

El siguiente procedimiento usa los procedimientos para reservar hotel, vuelo, etc.:

```
proc(travel(D1, D2, O, D),
[
   [
         bookRAirticket(O, D, D1, D2),
         bookCar(D, D, D1, D2)
   ]
   bookCar(O, O, D1, D2),
   bookHotel(D, D1, D2),
   sendEmail,
   updateExpenseClaim
]).
```

4.3.2 Planificador como Model Checking

El lenguaje BPEL4WS (Business Process Execution Language for Web Services) se relaciona con esta técnica de planning, es un estándar para la especificación y ejecución de servicios orientados a procesos de negocios. BPEL4WS ha sido diseñado con dos funciones en mente: programas ejecutables en este lenguaje permiten la especificación y ejecución de procesos internos de una organización, por otro lado, especificaciones abstractas en este lenguajes pueden ser usadas para especificar y publicar invocaciones y la interacción con servicios externos.

En [64] se presenta una técnica de planning para composición y monitoreo automático de servicios Web especificados como servicios en BPEL4WS. Esto permite proveer servicios que se combinan con otros servicios, posiblemente distribuidos, para alcanzar una meta, para lo cual se

usa la descripción de los protocolos externos (especificaciones abstractas BPEL4WS) y los requerimientos, el planificador sintetiza en forma automática el código que implementa los procesos internos y usa servicios externos paraalcanzarla.

Un monitor de procesos es una pieza de software que es capaz de detectar fallas y un proveedor externo se comporta consistentemente según los protocolos especificados. Esto es importante para la aplicaciones de servicios Web, ya que el comportamiento erróneo en ejecución afecta a los servicios compuestos automáticamente, por ejemplo fallas en la infraestructura de pasaje de mensajes, o cambios en la especificación externa de un servicio Web.

Para alcanzar la meta el planificador debe considerar algunos problemas:

- No determinismo, hay cuestiones que no se pueden determinar de antemano, por ejemplo en una interacción no se puede saber a priori si la respuesta a una solicitud será positiva o no, si el usuario confirmaráo no un servicio, etc.

- Obserbabilidad parcial, el planificador solo puede observar la comunicación con los procesos externos perono puede acceder a sus estados internos.

- Metas extendidas, los requerimientos de un negocio a menudo implican condiciones complejas sobre el comportamiento de los procesos. Así los requerimientos necesitan expresar preferencias sobre diferentes metas a alcanzar.

La técnica de planificación model checking considera estas cuestiones. Una máquina de estados finitos representa los servicios externos que están disponibles. Los requerimientos del negocio son expresados en EaGLe y son usados para desarrollar el plan constituido por procesos internos que definirán el servicio compuesto. También se generan monitores que son autómatas que hacen el seguimiento de la evolución de los procesos externos de acuerdo a las observaciones de las interacciones.

Un dominio es un sistema genérico con su propia dinámica, el plan también es modelado

como un sistema con una dinámica interna el cual controla la evolución del dominio a través de observaciones que describen la parte visible del estado del dominio y el cual controla las evoluciones a través de acciones. El primer paso para construir el modelo del dominio es identificar cada uno de los protocolos de los proveedores como un modelo de planning. Los estados del dominio son usados para codificar los estados del protocolo, los valores de las variables y el contenido de las entradas y salidas. Las acciones corresponden a operaciones de lectura y escritura y son accesibles desde un agente externo. Una acción *receive* es ejecutada en un canal de salida y una acción *send* en un canal de entrada.

Un planificador del dominio es definido en término de sus estados, las acciones que acepta y las posibles observaciones que el dominio exhibe. Algunos estados constituyen el conjunto de estados iniciales válidos. Una función de transición describe como la ejecución de una acción desde un estado permite pasar a diferentes estados, dependiendo de las observaciones actuales y del contexto. Una función de observación está asociada a cada estado del dominio.

Un problema de este enfoque es que en la mayoría de los casos el dominio se vuelve exponencialmente mayor por lo que el enfoque no es viable, a través de heurísticas se resuelve este aspecto, evitando generar todo el conocimiento del dominio.

A través de expresiones en el lenguaje EaGLe se pueden expresar condiciones sobre los estados. Resolver el problema de planning implica usar algoritmos que consideren la observabilidad parcial con metas EaGLe o considerar que la observabilidad es completa dentro de un nivel de conocimiento del dominio.

Ejemplo (extraído de [64])

El servicio del ejemplo consiste en la fabricación y entrega de muebles, el cual consiste de dos servicios separados, por un lado la fabricación de muebles y por otro el servicio de entrega.

Los protocolos para interactuar con los servicios son: 1) para interactuar con el servicio de fabricación de muebles en primer lugar debe existir un requerimiento por parte de un cliente, el

ítem solicitado puede estar disponible o no, en el último caso el protocolo termina con una falla. Si el ítem está disponible el cliente recibe mas información sobre el mismo y el cliente responderá si continúa con la solicitud o no. Luego el cliente recibe información sobre el precio y tiempo de fabricación a los cual también dará su respuesta para continuar o no con la solicitud. 2) para interactuar con el servico de entrega lo primero es que exista un requerimiento de transporte de un objeto de un cierto tamaño a un lugar dado. Si no es posible hacer el transporte el protocolo termina con una falla, si es posible hacerlo se informa el costo y tiempo de entrega y el cliente debe responder si acepta el servicio o no.

La idea es combinar ambos servicios para que el cliente interactúe con el servicio de fabricación y entrega, el cliente solicitaría un ítem y que éste sea llevado a un cierto lugar, obtendría el costo completo y tiempo (que incluye fabricación y entrega) y decidiría si aceptarlo o no. Una interacción típica de este servicio al cual llaman P&S es:

1. El usuario solicita a P&S un ítem I y quiere que sea transportado a un lugar L.

2. P&S consulta con el fabricante el tiempo y costo de fabricación, así como el tamaño del mueble.

3. P&S consulta al transporte el costo y tiempo de entrega de un objeto de tal tamaño a un lugar L.

4. P&S le hace al usuario una propuesta brindándole el costo y tiempo total.

5. El usuario confirma la orden a P&S.

En [66] se presenta una propuesta de planificación basada en Modelos (MBP). Este tipo de planificación se caracteriza por proveer un marco general para tratar con diferentes problemas de planificación en dominios no determinísticos y permite implementar algoritmos que pueden manejar grandes espacios de estados. Se asume un modelo general en el cual la incertidumbre se puede dar en la situación inicial, en los efectos de las acciones, y en el estado en el cual se den las

acciones.

Este modelo considera dos cuestiones a la hora de hacer la planificación para alcanzar las metas: la observabilidad, puede darse en uno de los siguientes niveles: completa, parcial o nula; y la expresividad de las metas, puede darse que las metas expresen el estado que se desea alcanzar o que existan metas extendidas que expresen que estados se deben atravesar al ejecutarse el plan para alcanzar el estado deseado. MBP puede crear planes secuenciales, condiciones, interactivos y planes que tengan en cuenta la historia de ejecución previa.

MBP se basa en Planning via Symbolic Model Checking. Por lo que el plan es hecho buscando a través de autómatas de estado finito cuyos estados son expresados como fórmulas proposicionales. La forma de encontrar el plan depende del grado de observabilidad del mundo (determina la forma en que se hará la búsqueda en profundidad, en lo ancho primero, etc) [66].

4.3.3 Planificador HTN (Hierarchical Task-Network)

Uno de los principales obstáculos que tiene la planificación HTN es la falta de un marco teórico que lo sustente. En el trabajo presentado en [65] se propone un formalismo que permite evaluar el poder expresivo de este tipo de planificadores y desarrollar algoritmos de planning HTN que sean correctos.

```
Algoritmo Genérico de Planning HTN

1. Input a planning problem P.
2. If P contains only primitive tasks, then
     resolve the con icts in P and return the result.
     If the con icts cannot be resolved, return failure.
3. Choose a non-primitive task t in P.
4. Choose an expansion for t.
5. Replace t with the expansion.
6. Use critics to nd the interactions among the tasks in P,
     and suggest ways to handle them.
7. Apply one of the ways suggested in step 6.
8. Go to step 2.
```

Esta técnica de planning usa acciones y estados del mundo, cada uno de los cuales son representados por un conjunto de átomos que son verdaderos en ese estado. Las acciones, también llamadas tareas primitivas, corresponden a las transiciones de estado, ya que cada acción es una

función que va desde un estado a un conjunto de estados.

Un planificador HTN busca un plan para realizar una tarea en red, el cual se realiza a través de la descomposición de tareas y la resolución de conflictos para alcanzar las metas.

Una tarea en red es una colección de tareas que deben ser realizadas junto con ciertas restricciones sobre el orden en el cual pueden ser llevadas a cabo, la forma en que las variables son instanciadas y cuales literales deben ser verdaderos antes o después de que la tarea sea realizada. Una tarea en red puede contener solo tareas primitivas y en ese caso es llamada tarea en red primitiva. El caso mas general es una tarea en red que contenga tareas no primitivas, las cuales no pueden ser ejecutadas directamente porque representan actividades que pueden implicar la ejecución de otras tareas.

Los métodos son construcciones que permiten llevar a cabo las tareas no primitivas y expresan que una tarea puede ser resuelta para alcanzar todas las tareas en una tarea red sin violar las restricciones impuestas. Cada tarea no primitiva puede ser reducida a través de un método que encuentre la tarea red que la sustituya. Cada reducción debe chequear y resolver los conflictos que pueden surgir, al detectarse tempranamente se reduce así el backtracking.

Este método de planning tiene una sintaxis y semántica bien definidas, presentada en [65], tal que permiten el desarrollo de algoritmos basados en él tal como el formalismo llamado UMCP (Universal Method-Composition Planner).

```
Procedure UMCP:

1. Input a planning problem P = hd; I; Di.
2. if d is primitive, then
      If compd; I; D 6= ;, return a member of it.
      Otherwise return FAILURE.
3. Pick a non-primitive task node n :  in d.
4. Nondeterministically choose a method m for .
5. Set d := reduced; n; m.
6. Set , := d; I; D.
7. Nondeterministically set d := some element of ,.
8. Go to step 2.
```

Strips se diferencia de los otros planificadores HTN en la forma en que se constituye el

plan y los objetivos del mismo. En este caso el objetivo es encontrar una secuencia de acciones que cambien el mundo a un estado tal que satisfaga ciertas condiciones y que alcance ciertas metas. Este método procede buscando operaciones que causen los efectos deseados a la vez que transforman las precondiciones de esas operaciones en submetas.

SHOP2 es un sistema de planning HTN independiente del dominio, la diferencia con otros sistemas de planning HTN es que SHOP2 planifica las tareas exactamente en el orden en el que luego serán ejecutadas, esto permite conocer el estado actual del mundo en cada paso en el proceso de planning y desarrollar mecanismos que basados en la evaluación de precondiciones pueden aumentar la capacidad para hacer inferencias y razonamientos. Además es posible hacer uso de programas externos, por lo cual SHOP2 es ideal para planning integrando fuentes de información externas y considerando el entorno [67].

Ejemplo (extraído de[49])

La madre de Bill y Joan's realiza la siguiente secuencia de actividades: 1)Obtener una receta de un anti-inflamatorio, 2) Realizar una radiografía y tomografía para diagnosticar las posibles causas de los síntomas, 3) Ir al doctor para que vea los resultados de los estudios.

Bill y Joan deben deben realizar las siguientes actividades: 1) Completar la prescripción en la farmacia, 2) Pedir los turnos para los estudios, 3) Sacar un turno con el doctor.

Para cada uno de los pasos hay preferencias y restricciones con respecto a los dos tratamientos: es preferible que ambos turnos sean lo mas cercanos posible el mismo día y en sitios cercanos, o que ambos turnos sean días distintos. El turno con el doctor debe ser posterior a los dos estudios. Cada uno de los turnos debe ser adecuado para los horarios del hijo que acompañe a la madre.

Se asume que existen servicios Web para sacar los turnos, pero puede ser difícil para Bill y Joan seleccionar los turnos manualmente ya que deben verificar la existencia de los turnos adecuados, cercanos en horario y lugar. Para solucionar esto suponemos que se usa una ontología

DAML-S para realizar la descripción de cómo componer servicios Web para realizar tal tarea.

4.4 Workflow

Los sistemas de información deben soportar procesos de negocios para lo cual no es suficiente enfocarse simplemente en las tareas sino que deben atenderse cuestiones relacionadas con el control, monitoreo y soporte de los aspectos logísticos de los procesos de negocios; es decir que deben administrar los flujos de trabajo a través de la organización. Esto dio lugar a la necesidad de identificar conceptos, técnicas y herramientas para soportar la administración de workflow.

El workflow se enfoca en las operaciones de una actividad de trabajo, identificando la forma en que se estructuran las tareas para que sean realizadas en el orden correcto y lograr que se sincronicen. Además identifica los flujos de información y permite realizar el seguimiento de la ejecución de las tareas. Una forma común para modelar workflow es usando redes de Petri.

Los sistemas de administración de workflow (WFMS) son herramientas que permiten la definición, ejecución, registración y control de workflow, lo que posibilita que una aplicación no conozca a las otras aplicaciones con que debe interactuar flexibilizando de esta manera la programación de las mismas. El principal motivo de un workflow es controlar procesos por lo que es importante que sea modelado correctamente. Las redes de Petri permiten modelar dichos procesos y tienen algunas características destacables: semántica formal, naturaleza gráfica, expresividad, etc.

En [109] se proponen las redes de Petri como herramientas para modelar y analizar procesos. Las redes de Petri pueden usarse como un lenguaje de diseño para la especificación de workflow y además pueden usarse para verificar la correctitud de los procedimientos de un workflow. Para modelar workflow de procesos en redes de Petri las *tareas* son modeladas como *transiciones*, las *condiciones* como *sitios* y los *casos* como *tokens*. En la Figura 4.9 se muestra un ejemplo.

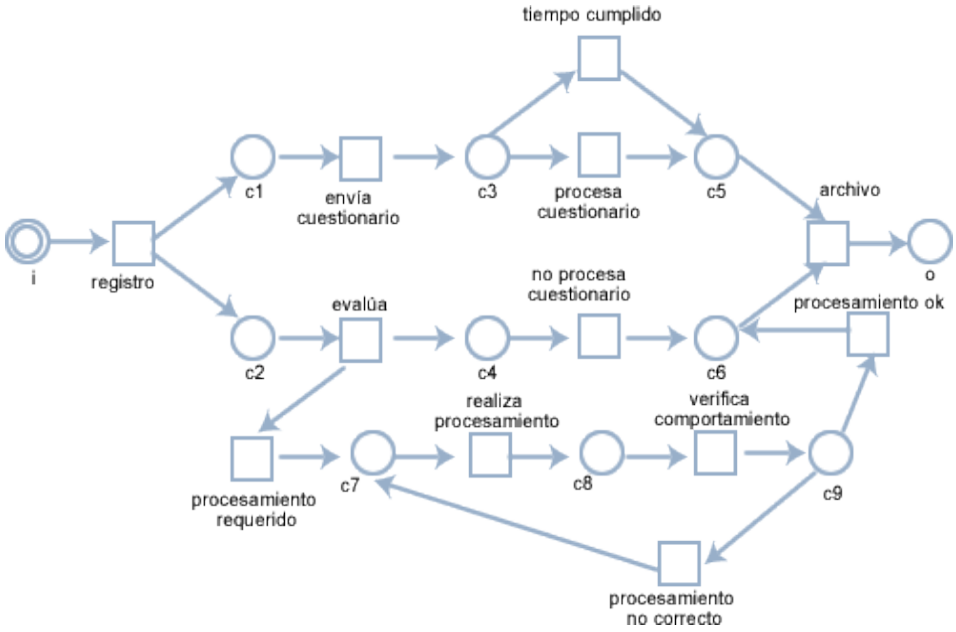

Fig. 4.9. Ejemplo de procesos modelados con redes de Petri.

Los sistemas de administración de workflow eran usados inicialmente para componer servicios en forma manual, para automatizar dicha tarea se requiere resolver algunas cuestiones de compatibilidad y adaptación, que pueden ser apuntadas gracias a la Web semántica, así como las cuestiones que tienen que ver con la QoS requerida. En el enfoque presentado en [108] se realiza la composición de servicios usando workflow para lograr la integración automática. Se presenta una forma de combinar servicios sobre la base de *patrones de composición* que representan los elementos estructurales de una composición tales como secuencia, iteración y ejecución paralela. Se definen también los patrones de composición y se contemplan las dimensiones de QoS. El mecanismo propuesto determina la calidad de una composición usando un modelo de workflow genérico para hacer la combinación de servicios considerando las dimensiones locales de calidad. Los patrones de workflow que se proponen en este trabajo han sido creados para definir un conjunto de requerimientos para los sistemas de administración de workflows y permiten comparar dichos sistemas por su funcionalidad y por sus aspectos no funcionales.

Algunos sistemas de workflow intentan combinar las tecnologías de bases de datos con sistemas basados en eventos para implementar workflow distribuidos dirigidos por eventos, tal es el caso del motor **WIDE** [110] y **EVE** [111].

WIDE extiende la tecnología de base de datos enfocándose en el manejo de transacciones, lo que aumenta la flexibilidad y permite mejorar la semántica de los workflow de procesos y soporte de reglas, que permite un comportamiento reactivo frente a los eventos del workflow. En la Figura 4.10 se muestra la arquitectura global del sistema WIDE, en la Figura 4.11 se muestra la arquitectura de administración de transacciones y de soporte de reglas.

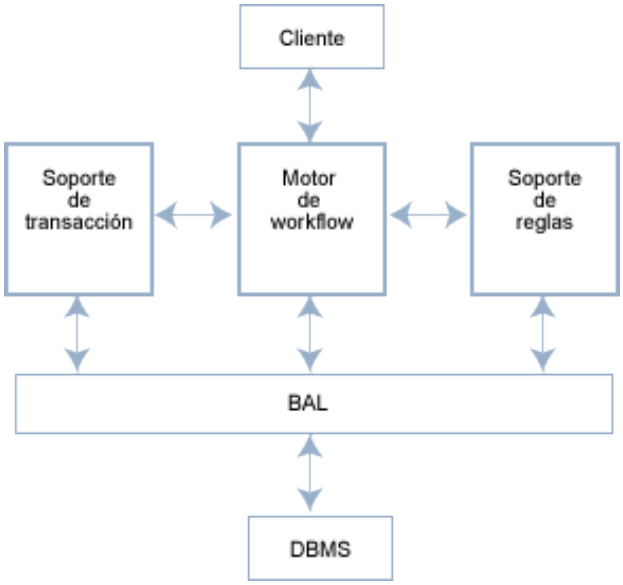

Fig. 4.10. Arquitectura global de WIDE

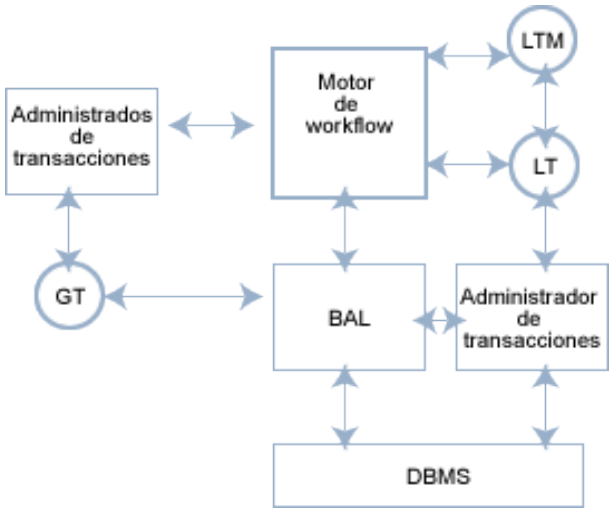

Fig 4.11. Arquitectura de soporte de transacciones y arquitectura de soporte de reglas.

EVE usa un modelo basado en un Broker y Servicios para describir la arquitectura del sistema de workflow, lo cual permite simplificar la integración de tareas y mejorar la definición de los aspectos funcionales, informacionales y de comportamiento del sistema. Se definen módulos

para la detección de eventos, para la ejecución de reglas y la ejecución de servicios. Cuando se detecta un evento se activa la regla correspondiente (ECA-rules), se verifican las condiciones y si se cumplen se realiza una acción determinada. Las acciones pueden generar eventos que son detectados por el módulo de detección de eventos. En la Figura 4.12 se muestra la estructura de un workflow y en la Figura 4.13 el proceso de ejecución de workflows en EVE.

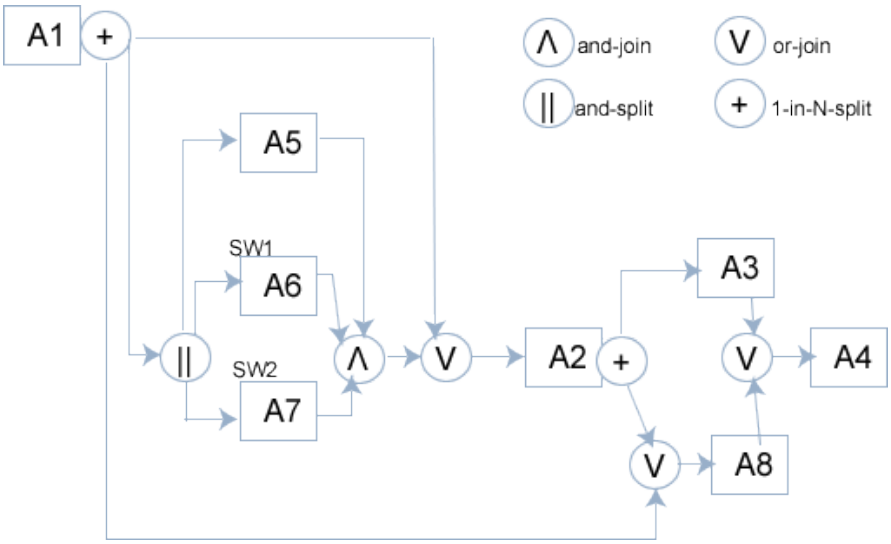

Fig. 4.12. Estructura de un workflow.

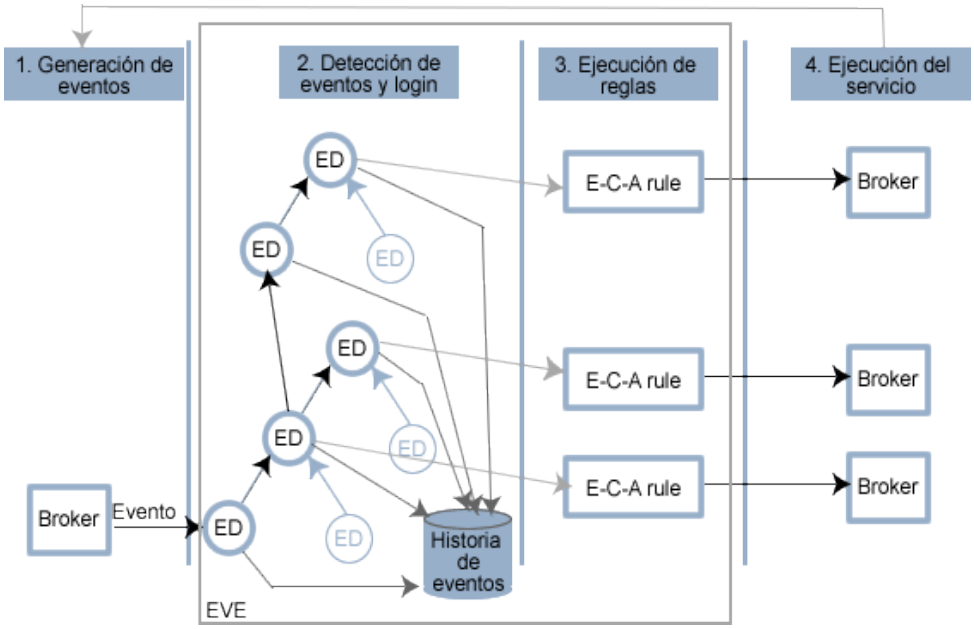

Fig. 4.13. Proceso de ejecución de workflows en EVE.

En [107] se distinguen entre dos tipos de servicios Web: porun lado los *servicios discretos*, los cuales son servicios de corta duración y no pueden responder a eventos externos asincrónicos,

ejemplos de tales servicios son agregar un ítem a un carrito de compras o cargar una tarjeta de crédito; por otro lado los *servicios orientados a sesión* son servicios de mayor duración y pueden responder a eventos, ejemplos de tales servicios son teleconferencia, chat colaborativo o video streaming.

Componer servicios discretos es relativamente simple y solo requiere usar el modelo de procesos, pero componer servicios orientados a sesión es mas complejo ya que se deben considerar eventos asincrónicos y determinar cómo impactan en la sesión activa. En este trabajo los autores proponen un modelo y una arquitectura flexible para la composición y ejecución de servicios discretos y servicios orientados a sesión. Dicho modelo permite la especificación de *diagramas de flujos activos* que pueden ser disparados por eventos asincrónicos que afectan la sesión activa. También se pueden especificar *esquemas de procesos* que combinan múltiples servicios y describen la ejecución de los mismos.

AZTEC, presentado en [107], es un sistema de workflow donde las sesiones de los servicios son vistas como objetos de sesión que generan eventos y llamadas a funciones. Se usan wrappers para traducir entre la representación interna de las funciones y eventos a la interfaz SOAP que es soportada por dichos servicios. Los wrappers pueden transformar las llamadas asincrónicas a las funciones en llamadas sincrónicas. El propósito de una composición es especificar como interactúan los objetos de sesión. El framework AZTEC propuesto permite usar diagramas de flujo para especificar cómo responder a los eventos de los objetos de sesión y determinar cómo el estado de un objeto impacta en otros. Los diagramas de flujo proveen las mismas ventajas que los modelos de workflow, tales como separación de la lógica de control de las tareas que son realizadas, pero también soportan construcciones para la manipulación de datos. Debido a que no puede predecirse cuando un evento u otro ocurrirá, se incorpora en este modelo el paradigma dirigido por eventos ya que un diagrama de flujos puede ser representado en un workflow. Este modelo soporta la adaptabilidad a través de la modificación del contenido del repositorio que representa el contexto durante la ejecución de una instancia de workflow. En la Figura 4.14 se muestra la arquitectura del sistema AZTEC.

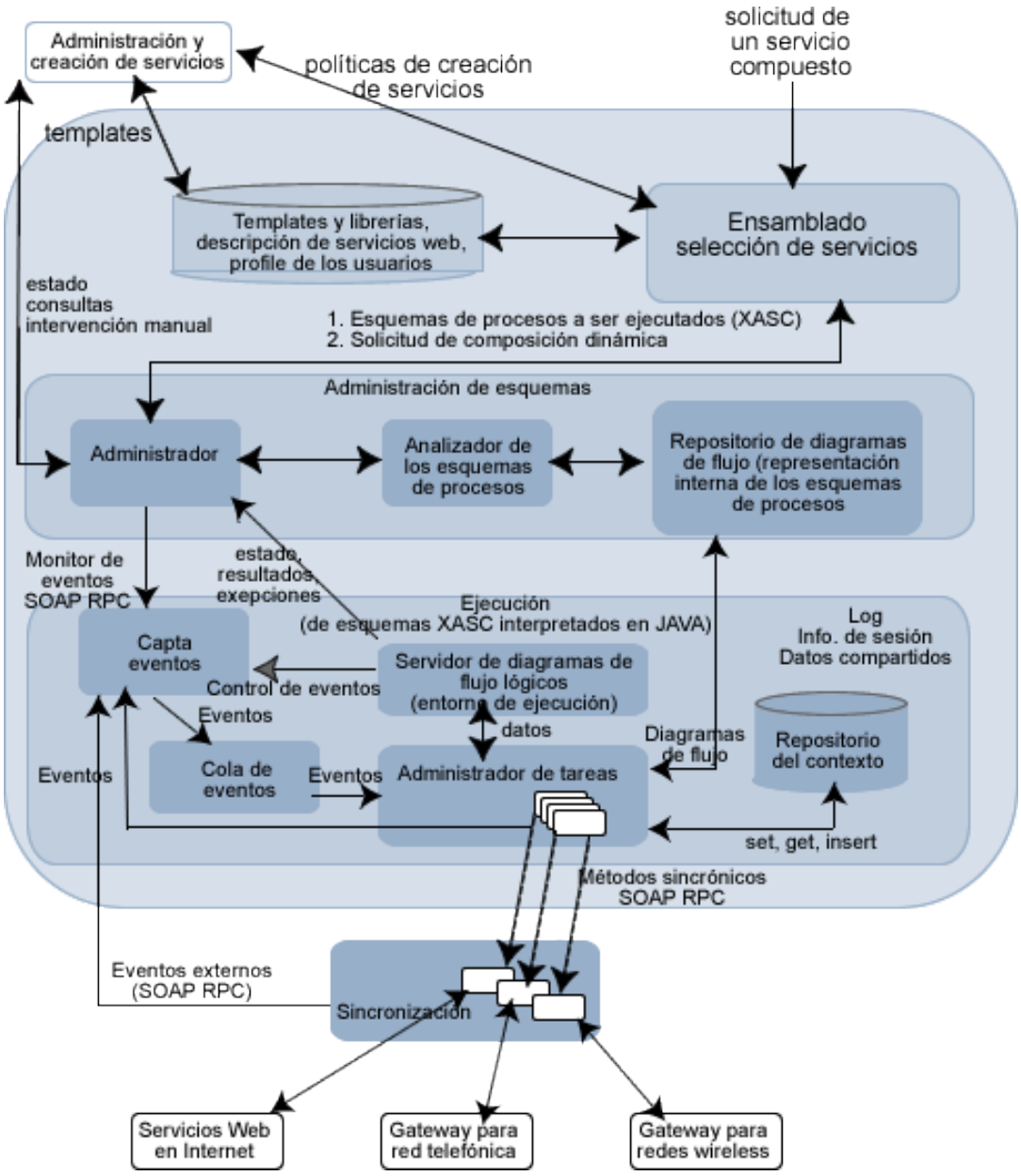

Fig. 4.14. Arquitectura del sistema AZTEC.

En [112] se presenta una propuesta para modificar dinámicamente las instancias del workflow que se ejecutan usando reglas y predicados (drop, replace, check, delay, process). Este enfoque se basa en la detección de eventos para decidir automáticamente las modificaciones en la instancia del workflow que se ejecuta.

En [113] se presenta un modelo orientado a servicios que pretende solucionar los problemas originados de las complejas relaciones existentes entre las diferentes organizaciones y sus servicios. En esta propuesta se modela la ejecución de

un workflow como servicios que cooperan, permitiendo que las organizaciones interactúen a través de interfaces bien definidas. Un workflow es modelado como un grafo con actividades como nodos y los arcos representan los flujos de control y de datos. Además se consideran los parámetros de calidad (QoS) tales como duración y costo máximo permitido. Las actividades son realizadas por agentes que pueden representar humanos, programas o proveedores de servicios externos. Un servicio tiene cuatro interfaces: entradas y salidas, que representan los flujos de datos, y control y notificaciones, que representan los flujos de control y la notificación de eventos. En la siguiente sección se analizan diferentes propuestas para realizar la adaptación dinámica de workflows que representan servicios Web para responder a los cambios en el entorno que pueden ocurrir durante la ejecución.

4.4.1 Workflow para servicios Web

El entorno en el que se ejecutan los servicios Web es cambiante, por lo que tanto los datos de entrada como los parámetros de calidad pueden cambiar en cualquier momento durante el ciclo de vida de un proceso. Esto es llamado *Volatilidad de los datos*, o *Volatilidad de los componentes* en el dominio de workflow, y puede afectar la performance de la composición de servicios Web. Una forma de afrontar el problema es con técnicas de adaptación de workflow. Varias propuestas han sido presentadas como en [44] donde la técnica de adaptación se basa en el monitoreo de los cambios en el entorno, en [42] se estudia el impacto de los cambios ocurridos sobre el workflow y se considera la posibilidad de adaptación mientras se ejecuta el workflow. La adaptación llamada A-WSCE presentada en [42] usa un enfoque de tres etapas para la composición y ejecución de workflow, refinando la propuesta presentada en [46]. La arquitectura del sistema A-WSCE puede verse en la Figura 4.15. La forma de adaptación se basa en la generación de múltiples workflows los cuales pueden alternarse ante la falla de uno de ellos o ante cambios de los parámetros de QoS. En esta propuesta se chequea periódicamente la QoS ofrecida por los proveedores en forma aleatoria, es una estrategia simple para decidir si el workflow actual es óptimo pero se presenta como una limitación que es superada definiendo el Valor de la Información Cambiada (VOC) asociado a cada workflow para determinar el workflow que sustituirá al anterior, mejorando la

performance de la composición.

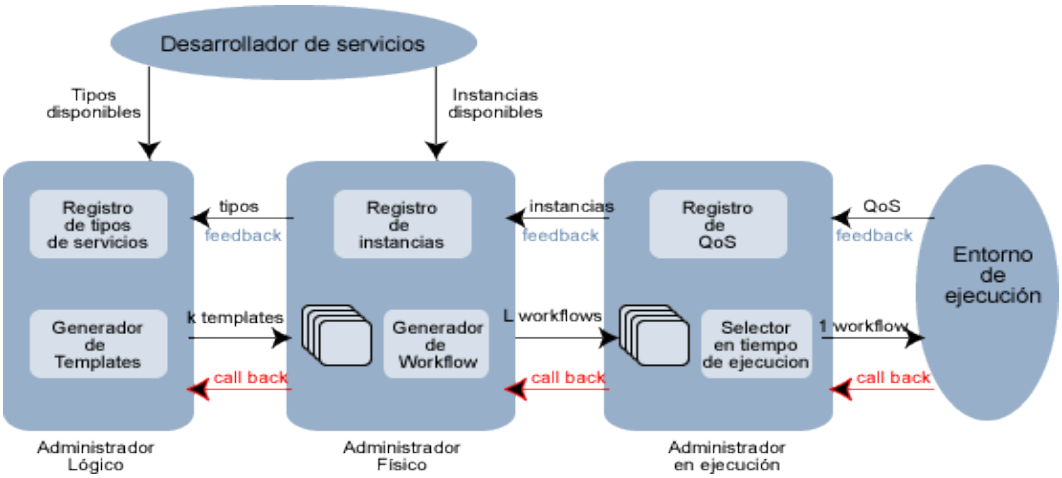

Fig. 4.15. Arquitectura del sistema A-WSCE

Los métodos de adaptación propuestos consisten en monitorear los cambios ocurridos y medir el impacto de esos cambios en los workflow, y permiten que la adaptación de los workflows se realice mientras se hace la composición. En el framework se consultan frecuentemente los parámetros de calidad de los servicios para seleccionar el que resulte en la composición óptima. Esto impone dos limitaciones: algunas consultas pueden no tener información nueva que sea significativa para requerir algún cambio; la información obtenida puede implicar grandes costos para hacer la adaptación sin tener una ganancia significativa. Para apuntar a esto se usan modelos estadísticos de volatilidad de los parámetros, el mecanismo VOC permite analizar el impacto esperado de hacer un cambio.

Este framework se basa en el enfoque de composición de servicios Synthy [43] que diferencia entre *tipos* de servicios, agrupando servicios similares según la funcionalidad, y las *instancias* de los mismos que corresponden a los proveedores. Para cada acción en el workflow, correspondiente a los tipos de servicio, pueden haber múltiples instancias del servicio que implementen la misma funcionalidad y pueden variar en sus características no funcionales. Además los valores de QoS de las instancias de un servicio pueden variar dependiendo de las condiciones de ejecución del sitio proveedor.

En [43] se generan los workflow ejecutables seleccionando una instancia por cada tipo de

patrón, optimizando los valores de calidad, para lo cual se usa la función Q (0<Q<1) que captura los valores de cada dimensión de calidad (costo, tiempo y disponibilidad) según sus pesos relativos. En cambio, en el framework presentado en [42] se generan múltiples instancias en la etapa lógica y se seleccionan L.

La adaptación puede hacerse en varios niveles. Primero se pueden buscar instancias de un servicio alternativo cuando uno falla, tal que ofrezca la misma funcionalidad con diferentes garantías de QoS. El segundo nivel de adaptación se da cuando ninguna de las instancias satisface los requerimientos, entonces se debe buscar otro template que lo logre. En la Figura 4.16 se representa la composición lógica y en la Figura 4.17 la composición física.

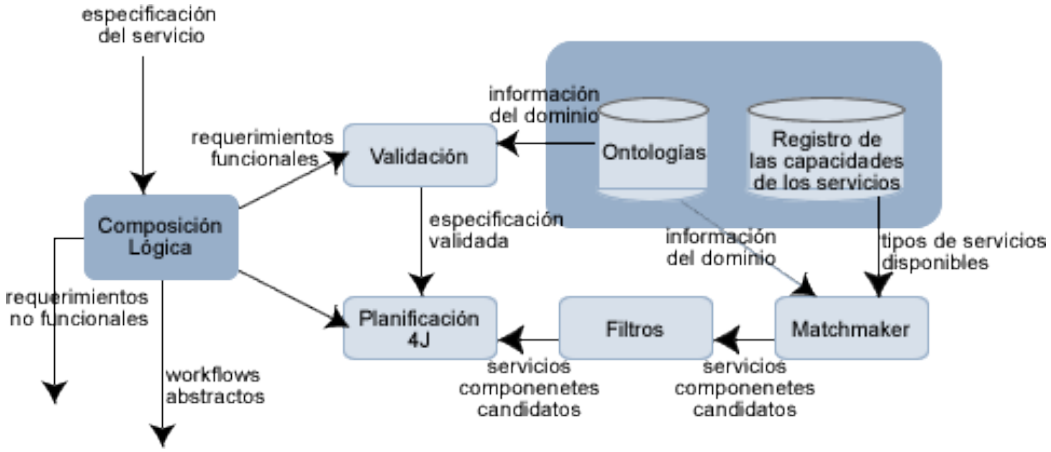

Fig. 4.16. Composición Lógica.

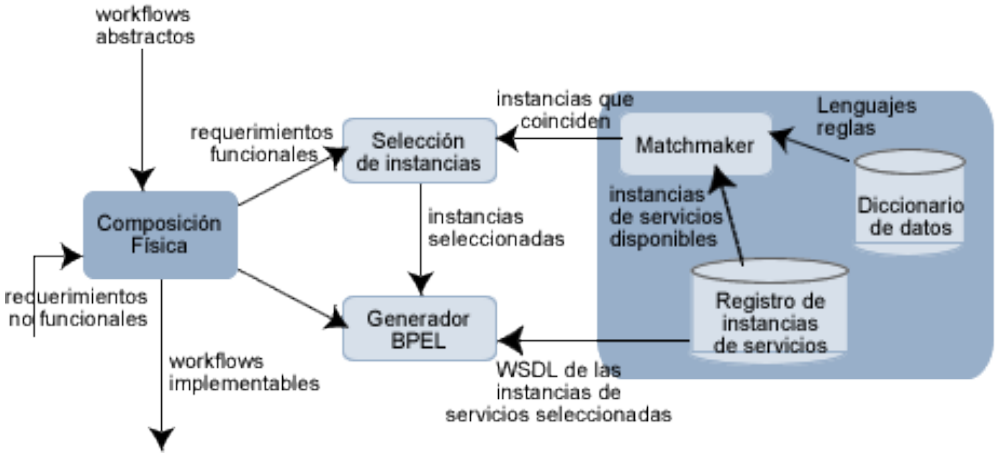

Fig. 4.17. Composición Física.

Los cambios mas frecuentes deberían ser por cambios en los valores de QoS que por la

necesidad de reemplazar una instancia que falla. Para que la adaptación sea efectiva frente a los cambios en el entorno: un servicio compuesto debería poder responder a eventos externos, las técnicas de adaptación deberían considerar las probabilidades de cambios en el entorno, y la adaptación debería ser aplicada incrementalmente para evitar pérdidas de tiempo.

Se configura un *Logical Manager* (LM) y un *Phisical Manager* (PM) con detalles sobre los tipos de servicios y las instancias de los mismos. El *Runtime Manager* (RM) mantiene el registro de la información recibida sobre los cambios de los parámetros de calidad o fallas de las instancias a través de un componente de monitoreo.

Este enfoque pretende aumentar el número de elecciones generando múltiples workflows en diferentes etapas que realizan la misma composición en términos funcionales de la siguiente manera:

- Generación de múltiples templates en el LM. Estos wokflows abstractos o templates son creados manualmente o usando técnicas de planning que trabajan sobre servicios Web anotados semánticamente.

- Generación de múltiples workflows ejecutables para cada template. Se genera una instancia para cada tipo de template desde los cuales luego se generarán los workflow ejecutables.

- Selección de un workflow. Habiendo generado múltiples workflow para cada template el PM selecciona un número L de los cuales solo uno será elegido para su ejecución buscando maximizar los valores de QoS, y los otros quedarán al alcance para recuperarse rápidamente frente a fallos o cambios importantes de calidad.

Durante el proceso de composición el LM invoca a un Template Generator para crear los workflows según los requerimientos funcionales. El proceso de selección es llevado a cabo por el Workflow Generator buscando optimizar los requerimientos no funcionales en el PM. En [46] una función R_L le asigna un puntaje a cada workflow abstracto, usando para esto la longitud del workflow, la comprensibilidad por parte del usuario, etc., este puntaje será usado para seleccionar

los workflow que pasan a la siguiente etapa.

Los patrones son almacenados e instanciados en workflow ejecutables en el PM. En esta etapa una función R_p le asigna un puntaje a cada workflow ejecutable, usando los valores de QoS del workflow instanciado [46]. El RM almacena estos workflow ejecutables e invoca al *Runtime Selector* para elegir el que tenga mejor QoS actual.

Mientras un workflow se ejecuta pueden ocurrir cambios en el entorno, como fallas de servicios o cambios en los parámetros de QoS. El proceso de recuperación se debe realizar en forma incremental. Desde la etapa de ejecución, si ningún workflow disponible puede ser ejecutado, el RM invoca al PM para seleccionar uno nuevo del conjunto de workflow ejecutables. Desde la etapa física, el PM puede pedir al LM que genere nuevos templates si ninguno de los workflows disponibles pueden ser concretizados en uno ejecutable.

La infraestructura de monitoreo envía información sobre los cambios de calidad y fallas de las instancias a la etapa de ejecución en un intervalo de tiempo T_M. El RM incorpora los cambios cada intervalos de tiempo T_R y selecciona el mejor a ejecutar. El PM es invocado cada T_P para generar el conjunto de workflows ejecutables y esto es pasado al RM. El LM crea nuevos templates cada período de tiempo T_L Siendo $T_M<T_R<T_P<T_L$. Además de esto el *callback mode* se usa cuando todos los workflows en una etapa fallan.

Cuando hay disponibles nuevos servicios en término de funcionalidad esto debe informarse a la etapa de composición lógica. Si nuevas instancias de un tipo de servicio están disponibles esto debe informarse a la etapa de composición física. Si hay cambios importantes en la QoS de un servicio esto debe informarse a la capa de composición de ejecución. En cada etapa se debe evaluar el conjunto de alternativas y cambiar la selección según los cambios en el entorno. En [42] se define un *feedback chanel* que permite mantener el registro de los tipos e instancias disponibles en cada etapa. También se provee un *callback mode* que permite superar situaciones de fallas de todos los workflows disponibles. En [46] se definen tres funciones para mantener el sistema actualizado: F_R registra información de QoS y fallas de los servicios, lo cual es usado en la etapa

de ejecución; F_P registra información sobre QoS en un período de tiempo, esta información se usa en la etapa física; F_L registra información sobre los tipos de servicio lo cual es de utilidad en la etapa lógica.

La composición puede ser vista como un problema de planning donde cada servicio corresponde a una acción y los estados inicial y la meta se corresponden a los requerimientos del nuevo servicio. En el framework presentado en [48] la semántica del plan de composición P es formalizada en término del conjunto de candidatos denotado por <<P>>, el cual es el conjunto de secuencias de acciones que son compatibles con las restricciones del plan. El propósito del algoritmo de planning es encontrar el candidato mínimo, reduciendo en cada etapa el espacio de búsqueda de la siguiente manera:

- La etapa lógica genera un conjunto P con K workflows abstractos seleccionados, donde cada uno tiene |Pi| tipos de servicio y el conjunto de candidatos es $<<P_i>>$
- En la etapa física el espacio de búsqueda se reduce de $<<P>>= \sum<<P_i>>$. En esta etapa cada plan es instanciado en un conjunto de workflow ejecutables obteniendo el conjunto <<W>>, incluido en el conjunto <<P>>, de candidatos para ser ejecutados.
- En la etapa de ejecución solo uno de los workflows es ejecutado, preseleccionando L candidatos para una adaptación inmediata. Si no se pueden alcanzar los requisitos desde esta etapa se informa a la etapa física, la cual no deberá considerar todo el espacio de búsqueda sino solo <<P>> - <<W>> candidatos. Si ésta tampoco puede satisfacer los requisitos le informa a la capa lógica que tampoco deberá considerar todo el espacio sino que podrá remover los <<P>> workflows ya considerados.

El costo de realizar una adaptación incluye el costo de obtener información, el costo de cambiar un proveedor y de recomputar el workflow o cambiar de plan. El uso de VOC permite evaluar el costo de cambiar un workflow cuando suceden cambios en el entorno y permite evaluar si es efectivo hacer la adaptación o si es mas costoso que no hacerla.

El framewok presentado donde se generan múltiples workflows en cada etapa ha demostrado aumentar la eficiencia del sistema A-SWCE reduciendo el número de callbacks a la etapa física y aún mas el número de llamadas a la etapa lógica. El sistema puede funcionar de una variedad de formas:

- Con consultas continuas sobre los valores de calidad a todas las instancias: ha demostrado ser mejor cuando el costo de adaptación es insignificante.

- Con VOC: permite decidir en ejecución si hay que consultar los valores de calidad a las instancias. Permite consultar inteligentemente estimando la relación entre costo de adaptación y ganancia esperada. Es ideal cuando el costo de adaptación se incrementa. Además aumenta la estabilidad del sistema reduciendo la cantidad de cambios.

- Con consultas aleatorias: en ejecución el sistema decide si hay que consultar los valores de calidad a las instancias con una probabilidad de 0,5. Si se decide consultar toma el valor de QoS de las instancias, consulta el valor de QoS del nuevo workflow y los usa para seleccionar el mejor workflow a ejecutar. Esta alternativa es la menos estable ya que las consultas aleatorias se realizan en un número incremental.

En [87] se propone la plataforma eFlow para especificar, representar y monitorear la composición de servicios. Los servicios compuestos son modelados como procesos, soporta un lenguaje simple y a la vez poderoso de composición de servicios, manejo de eventos y excepciones, manejo de transacciones y control de cuestiones de seguridad.

El sistema eFlow, representado en la Figura 4.18, soporta la definición y representación de servicios compuestos adaptivos y dinámicos, a través de la especificación de procesos que se pueden configurar automáticamente en tiempo de ejecución de acuerdo a los servicios disponibles y a las necesidades del cliente. Esto permite una gran flexibilidad para modificar las instancias de los servicios.

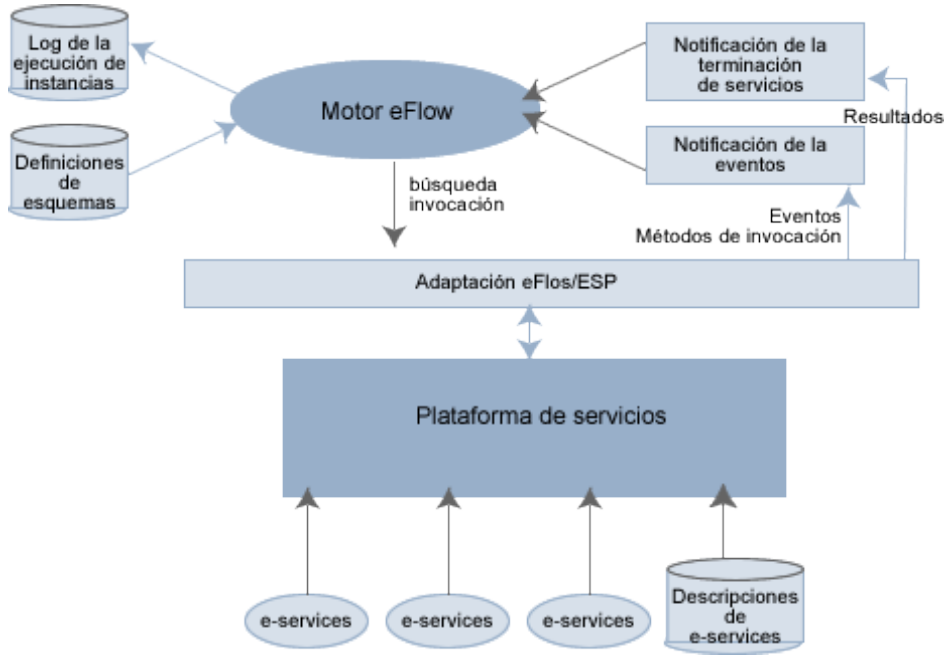

Fig. 4.18 Motor de eFlow para procesar eventos y notificaciones.

Un servicio compuesto es descrito como un esquema de procesos modelado como un grafo el cual define el orden de ejecución entre los nodos del proceso, definiendo el flujo de invocaciones entre los servicios. Los arcos representan que un nodo puede ser ejecutado solo si el predicado del arco es verdadero. Varias diferentes instancias de un proceso pueden derivarse de un esquema.

Para que sea posible la adaptación de los procesos eFlow provee diferentes conceptos:

- *Descubrimiento dinámico de servicios*, el cual se realiza a través de *reglas de selección de servicios*.

- *Selección de la conversación dinámicamente*, los flujos de invocación de métodos se deberían realizar en tiempo de ejecución. Para lograr esto eFlow tiene un repositorio de conversaciones, las cuales están asociadas a uno o mas servicios.

- *Nodos Multiservicio*, permiten la activación múltiple y en paralelo de un mismo servicio. Algo importante en este tipo de nodo es la especificación de cuando un nodo multiservicio es completado y por lo tanto puede pasar el control al siguiente nodo. Esto puede suceder

cuando todas las instancias del servicio han sido completadas o puede darse el caso en que no sea necesario esperar a que todas ellas sean completadas.

- *Creación de nodos de servicios dinámicamente,* los requerimientos de los clientes son variados, por lo que el mismo servicio compuesto puede no ser adecuado para cada uno de ellos. La posibilidad de crear servicios dinámicamente permite satisfacer las necesidades de cada usuario. Se define el concepto de nodo de servicio genérico, que a diferencia de los nodos comunes, no instancian servicios específicos estáticamente sino que ofrecen parámetros que se instancian en tiempo de ejecución y serán usados cuando el nodo sea activado, en ese momento se realiza la instanciación de los servicios.

Puede darse el caso en el que los esquemas de procesos deben ser modificados o que haga falta realizar ciertas acciones mientras un esquema de procesos es ejecutado. Estas modificaciones pueden responder a situaciones inesperadas, incorporación de nuevas reglas o políticas del negocio, mejorar procesos, corregir errores o deficiencias. Se distinguen dos tipos de modificaciones:

1. Cambios ad hoc: es un tipo de modificación que se aplica a la instancia de procesos de ejecución, típicamente para manejar situaciones inesperadas. Hay dos tipos de cambios ad hoc:

- Modificaciones al esquema de procesos. Los usuarios autorizados pueden modificar varios aspectos del esquema: estructura de flujos, definición del servicio, nodos de eventos, datos del proceso, etc.

- Modificaciones al estado de la instancia de procesos. Esto puede incluir modificar el valor de las variables, iniciar el rollback de una región del proceso, terminar el proceso, etc.

2. Cambios en masa: son modificaciones que se aplican a un conjunto de instancias de procesos que se ejecutan. Esto se da por ejemplo cuando se actualiza la versión de un proceso, lo cual puede modificar las restricciones del servicio compuesto por lo que es

preferible hacer la modificación en masa. Las modificaciones se realizan especificando uno o mas esquemas destino al cual serán migrados los esquemas.

El sistema también incorpora reglas de consistencia para evitar errores en tiempo de ejecución como consecuencia de la introducción de modificaciones, y reglas de autorización que garantizan que solo los usuarios autorizados pueden introducir modificaciones.

La colaboración de los procesos determina cuando, como y por quien es realizada cada una de las actividades. La Colaboración de los procesos incluye el manejo de la colaboración y la coordinación de las actividades que son realizadas por los participantes.

En la propuesta presentada en [115] se ha desarrollado una infraestructura para manejar la colaboración de los procesos, llamada Infraestructura de Administración de la Colaboración (CMI). Esto fue motivado por aplicaciones donde la tecnología de workflow y groupware no era suficiente. Para reducir el impacto causado por la colaboración de los participantes, el CMI debe permitir el aumento de los procesos, permitir que expertos y coordinadores tomen decisiones, proveer ventanas de oportunidades para decidir la opción a tomar. Para lograr esto el CMI cuenta con el Modelo de Administración de la Colaboración (CMM) y un sistema que lo implementa. El CMM combina las primitivas existentes de workflow y groupware y provee nuevas primitivas que principalmente son primitivas de datos estándar y de control de flujo para coordinar a los participantes y para automatizar la colaboración de los procesos. También se definen primitivas para especificar templates de procesos y su extensión, lo cual es realizado dinámicamente. En esta propuesta se pretende reducir la necesidad de intervención por parte de los expertos del dominio.

4.5 Resumen

Actualmente la Web es el medio para que las organizaciones brinden servicios y para que los clientes encuentren los servicios que necesitan para satisfacer sus requerimientos. Los servicios Web son aplicaciones capaces de llevar a cabo sus propias actividades para implementar procesos de negocios, por lo cual requieren poder conectarse con otros servicios para poder realizar

mayores transacciones de negocio. En este contexto han nacido diversos estándares, plataformas, lenguajes y modelos para llevar a cabo la descripción y descubrimiento de servicios Web.

Los investigadores han propuesto diferentes enfoques para llevar a cabo la implementación de la composición. Por un lado la definición del concepto de componente [115]. Un servicio componente representa una aplicación modularizada que facilita el reuso, extensión, especialización y herencia. Además un componente encapsula la lógica del negocio y se comunica a través de una interfaz bien definida a través de mensajes y operaciones.

Por otro lado los investigadores han identificado los problemas y cuestiones abiertas en relación con la composición de servicios, desde las deficiencias en la especificación hecha por el cliente, completitud de la solicitud y obserbabilidad parcial, el tipo de composición, manual o automática, la última es mas flexible pero la dificultad está en la forma de hacer la selección adecuada de los servicios para lo cual se deben tener en cuenta la sintaxis y semántica de los servicios; y por último la arquitectura que soporte la orquestación de los servicios que integran la composición.

Varios autores han propuesto diferentes enfoques para realizar la composición automática de servicios usando diferentes técnicas. En primer lugar hemos analizado las características de la composición usando Máquinas de Estados Finitos, en este contexto hemos definido los conceptos de compatibilidad, sustitutabilidad y conversación. En segundo lugar hemos estudiado las propuestas usando métodos de Planning, donde se usa lógica de primer orden para representar las precondiciones y efectos de las operaciones, así como el estado del mundo. Dentro de Planning nos hemos abocado a Cálculo de situación, Model checking (con el que se relaciona el lenguaje BPEL y se consideran las acciones no determinísticas, la observabilidad parcial del mundo y se define el concepto de Metas extendidas), y HTN (donde dos conceptos fundamentales son las acciones y el estado del mundo). Finalmente hemos estudiado las propuestas que usan workflow para llevar a cabo la composición, en este contexto es fundamental el concepto de procesos. Varios autores han identificado la necesidad de la adaptación dinámica de workflow para responder a los

cambios del entorno durante la ejecución de los mismos, hemos estudiado diferentes propuestas donde destacamos los conceptos de Volatilidad de los datos, generación de múltiples workflow para realizar la recuperación de menor costo, y hemos estudiado una plataforma de composición llamada eFlow.

En el siguiente capítulo nos abocaremos al estudio de las plataformas que hacen posible la implementación de estas propuestas.

Capítulo 5:

Implementación de Servicios Web Semánticos

En este Capítulo se describen las principales características que deben tener las plataformas para composición, ejecución y orquestación de los servicios Web. También se describe la tecnología de agentes, las plataformas de implementación y ejecución de agentes, y finalmente la relación de estas tecnologías con la implementación de los servicios Web.

Para finalizar el Capítulo se propone una aplicación de agentes y se realiza el diseño de la aplicación.

5.1 Introducción

El sueño de una Web semántica de servicios requiere, para ser alcanzado, la colaboración sobre la Web. La colaboración implica la interacción en forma cooperativa de los individuos en pos de alcanzar objetivos complejos. Se requiere para esto tecnologías que soporten la automatización de la colaboración sobre la Web semántica. La combinación de la tecnología de agentes, ontologías y servicios Web semánticos, prometen brindar el soporte para lograrlo.

Como hemos descrito en los capítulos previos, los servicios Web son componentes de software independientes de la plataforma que son publicados por los proveedores mediante lenguajes y estándares. Los servicios Web pueden ser localizados e invocados a través de protocolos predefinidos y pueden ser compuestos para satisfacer requerimientos de los usuarios cuando éstos no pueden ser alcanzados mediante un solo servicio.

Distintas tecnologías han surgido para resolver las cuestiones relacionadas con la

composición de servicios Web, y en especial para posibilitar que ésta sea realizada en forma automática. Los agentes o sistemas multiagentes son una de las tecnologías que mas se destacan para implementar los servicios Web.

En este capítulo se analizarán en primer lugar las tecnologías que han surgido como respuesta a la tendencia de computación distribuida. Luego se analizan las plataformas de agentes mas destacas y se propone un sistema de agentes para brindar un conjunto de servicios en particular.

5.2 Tecnologías

La evolución de los modelos para la implementación de sistemas de software ha sido guiada principalmente por la tendencia de adopción de un modelo de computación distribuida sobre la red, lo cual ha implicado el aumento de sitios para brindar recursos de datos y procesamiento.

Varias tecnologías han sido estudiadas y se han realizado diversas propuestas tales como el enfoque de comunicación a través de blackboard, utilizando objetos distribuídos y a través de agentes.

El enfoque de comunicación en el que se utiliza un **blackboard,** consiste en que múltiples procesos se comuniquen a través de la lectura y escritura de tuplas en un almacenamiento de datos global. Cada proceso puede acceder a los datos que le interesan y realizar sus cómputos de acuerdo a ello.

Las arquitecturas basadas en este enfoque, tales como FLiPSiDE Schwartz y LINDA Gelernter, proveen una solución flexible al problema de comunicación entre procesos distribuidos pero no se provee ningún tipo de control. Este enfoque no ha evolucionado y no se han realizado nuevas propuestas.

En cuando a la **tecnología de Objetos Distribuidos** podemos destacar que los lenguajes

orientados a objetos tales como C++ y JAVA, permiten mayor reusabilidad y modularidad del código gracias a las siguientes características: *encapsulación*, la creación de librerías de interfaces reduce la dependencia entre los algoritmos y con las estructuras de datos, permitiendo hacer modificaciones en la estructura interna de un algoritmo sin necesidad de modificar el código que usa dicha librería; *herencia*, permite extender rutinas de la librería; y *polimorfismo*, permite que un código pueda trabajar con un número arbitrario de tipos de datos. Las tecnologías de objetos distribuidos (DOOP) tales como CORBA OMG y DCOM Microsoft, permiten crear programas cuyos componentes pueden ser ejecutados en cualquier máquina. Esto es transparente al usuario gracias a un mecanismo que registra las descripciones de los objetos remotos. En CORBA el registro es llamado Object Request Broker (ORB) y se encarga de encontrar un objeto que pueda implementar la solicitud del cliente, de pasarle los parámetros, invocar el método correspondiente y devolver los resultados.

Esta tecnología tiene algunas desventajas, por un lado las interacciones entre los objetos son estáticamente definidas en el código, lo cual es una restricción al querer ser usado desde otra aplicación. Por otro lado el estilo de comunicación RPC (Remote Procedure Call) es un esquema de comunicación sincrónico que no es el mas adecuado para las características de un entorno distribuido.

La tecnología que aún sigue evolucionando es la de **Agentes**. Los agentes son entidades de software que tienen un conjunto de características:

- Autonomía: pueden trabajar sin la intervención del usuario y tienen cierto control sobre sus acciones y estado interno, además pueden responder a los cambios en el entono y su comportamiento está determinado por los objetivos que persiguen que en general no es determinístico y se define a través de un conjunto de acciones que serán realizadas según las precondiciones establecidas.

- Inteligencia: un agente puede razonar y decidir reaccionar tras un evento, elegir qué acción ejecutar para alcanzar un objetivo, el agente puede adaptarse a cambios en el entorno a

través de técnicas de aprendizaje.

- Interacción: los agentes tienen responsabilidades que combinadas pueden resolver problemas mayores, así un sistema multiagente es flexible, escalable, tolerante a fallos, hace una mejor administración de los recursos, etc.

- Sociable: un agente puede dialogar con otra entidad para interactuar, hacer la delegación de tareas, cooperar para alcanzar ciertos objetivos, coordinarse, realizar acuerdos entre partes implicadas, etc.

- Movilidad: los agentes pueden migrar de un nodo a otro en la red, preservando su estado.

Este paradigma tiene algunas características que lo favorecen frente a la tecnología de objetos distribuidos, por un lado permiten considerar el ancho de banda disponible para determinar la mejor forma de realizar una tarea, por ejemplo en algunos casos es mas eficiente realizar una consulta sobre los datos (el agente migra) que enviar los datos a un programa; por otro lado permiten realizar tareas en paralelo por varios agentes para llevar a cabo una tarea.

Como desventaja podemos identificar que los agentes deben especificar como interactúan con el entorno. Además los agentes deben estar escritos en lenguajes de programación que sean soportados por el entorno en que se ejecutan, mientras que otras tecnologías distribuidas pueden soportar componentes escritos en diversos lenguajes.

Este paradigma requiere de cuatro componentes: mecanismo de transporte, debe permitir el intercambio de mensajes en forma asincrónica; protocolo de interacción, debe permitir varios tipos de comunicación; lenguaje de contenido y vocabulario compartido, lo cual puede implementarse con la definición de ontologías.

En [116] se presenta un framework para la construcción de sistemas multiagentes llamado Open Agent Architecture (OAA). En este framework existe un *agente de interfaz de usuario* y varios *agentes de aplicación* y *meta-agentes*, donde las relaciones entre ellos son coordinadas por

un *agente facilitador*. En algunos sistemas el facilitador también provee un almacenamiento de datos global para los agentes clientes, lo cual es similar al estilo de interacción blackboard. Pueden existir mútliples facilitadores que se encarguen de ciertos grupos de clientes. Los agentes de aplicación son un tipo de agentes especialistas en un tipo particular y pueden ser independientes del dominio, tal como un agente de procesamiento de lenguaje natural, o agentes de recuperación de datos como por ejemplo agentes de planificación de viajes. Los Meta-agentes asisten al facilitador para coordinar las actividades de otros agentes, pueden usar aplicaciones y conocimiento específico del dominio, reglas de razonamiento, algoritmos de aprendizaje, etc. Los agentes de interfaz de usuario están constituidos por pequeños agentes para manejar la interacción con el usuario.

La cooperación entre los agente se realiza a través de mensajes expresados en el lenguaje ICL (Interagent Communication Language). Los participantes de la comunicación son: el facilitador que provee la descripción de los servicios registrados, lo solicitantes de los servicios que envían sus requerimientos al facilitador y esperan la respuesta desde éste, y por último los facilitadores que coordinan los servicios que pueden satisfacer las metas.

El componente principal del ICL es el evento, el cual contiene un tipo, un conjunto de parámetros y un contenido. Cada agente en el sistema define y publica un conjunto de capacidades en ICL, describiendo los servicios que provee, lo cual define la interfaz del agente que será usada por el facilitador para comunicarse con él.

Este enfoque tiene la desventaja de que el facilitador constituye un cuello de botella en el canal de comunicación a la vez que constituye un punto de falla crítico.

Como se destaca en [119] una característica de los servicios es que son cajas negras desde el punto de vista de los clientes. Dentro de las cajas negras pueden haber alternativas y decisiones, como elegir una fecha o un título, pero ellas no alteran el comportamiento de los agentes. Desde esta perspectiva un servicio es un módulo de comportamiento que encapsula su estado y provee primitivas para dar información y acciones para solicitar datos.

Ya que los servicios Web son módulos de comportamiento un servicio Web compuesto se define como la arbitración de dicho comportamiento, a través de la selección de acciones.

En este libro los autores proponen considerar un servicio como un agente en si mismo, que en forma autónoma podrá cumplir ciertas metas, o como parte de un agente, el cual será invocado por otro agente a través de la Web para alcanzar metas de mayor nivel.

5.3 Plataformas de Agentes

Una plataforma de agentes debe implementar los servicios y brindar la infraestructura básica para posibilitar el desarrollo del ciclo de vida de los agentes así como su movilidad, puede brindar servicios de páginas blancas y amarillas, debe soportar el transporte y traducción de mensajes, debe implementar cuestiones de seguridad, debe soportar la planificación de múltiples tareas de agentes, etc.

La orquestación de los servicios que integran una composición puede realizarse de diversas maneras: como los sistemas peer-to-peer, donde los servicios interactúan entre ellos y con el cliente sin ningún control centralizado; o en forma centralizada, donde existe un mediador que realiza la orquestación.

JADE (*Java Agent DEvelopment Framework*) es una plataforma de software para el desarrollo de agentes que está implementada en Java.

En este entorno se crean múltiples contenedores para los agentes, un conjunto de contenedores constituye una plataforma. Cada plataforma debe tener un contenedor principal que tiene dos agentes especiales denominados AMS y DF. El **AMS** (Agent Management System) es el agente que controla la plataforma, siendo el único que puede crear y destruir a otros agentes, destruir contenedores y parar la plataforma. El **DF** (Directory Facilitator) es un agente que proporciona un directorio que registra los agentes que están disponibles en la plataforma.

Los agentes pueden migrar o clonarse a si mismos hacia otros host de la plataforma. El

ciclo de vida de los agentes puede controlarse a través de una interfaz gráfica que a su vez permite realizar tareas de debugging. La arquitectura de comunicación ofrece una forma de mensajería flexible y transparente a los agentes usando protocolos de transporte de mensajes que responden a los estándares FIPA.

Cougaar (***Cognitive Agent Architecture***) [72] es otra plataforma de software para el desarrollo de agentes implementada en Java. Pertenece a un proyecto de investigación DARPA. Cougaar no es solo una arquitectura, sino que es una metodología para el diseño y construcción de aplicaciones distribuidas.

Este enfoque permite considerar problemas de diferentes dominios realizando la descomposición funcional de tareas complejas y la integración de aplicaciones y fuentes de datos distribuidas. Además, considera la dinamicidad de los distintos dominios siendo capaz de generar y mantener el plan de ejecución dinámicamente así como la ejecución de aplicaciones paralelas con una comunicación de bajo acoplamiento y bajo consumo de ancho de banda.

Cougaar puede describirse como un motor de workflow a gran escala basado en componentes y agentes distribuidos, donde los agentes cooperan para resolver un problema comunicándose entre si a través de un protocolo de pasaje de mensajes asincrónico. La composición de los agentes se realiza según el aspecto funcional y se modifica dinámicamente mientras cambian los parámetros, restricciones y el entorno de ejecución. En esta arquitectura se definen dos elementos principales: sociedad y comunidad. Una sociedad consiste de un número de comunidades, cada una de las cuales contiene agentes y puede contener sub-comunidades. Una sociedad o comunidad refleja la estructura y comportamiento de una sociedad o comunidad real en término de su comportamiento.

April Agent Platform (**AAP**) es una plataforma de agentes que responde a FIPA[2], diseñada para ser una solución poderosa y liviana para el desarrollo de sistemas basados en agentes. Está escrita en el lenguajes de programación April y el InterAgent Communication

[2] Foundation for Intelligent Physical Agents (FIPA) desarrolla estándares para el dominio de agentes

System (IMC), y provee muchas características para acelerar el desarrollo de agentes y plataformas de agentes.

La herramienta **FIPA-OS** [122] se basa en componentes para permitir el desarrollo rápido de agentes que responden a las especificaciones FIPA. Este modelo soporta Agent shell, soporte de comunicación entre agentes en diferentes niveles, manejo de mensajería y comunicación y configuración dinámica de la plataforma.

Un Agent Shell puede producir agentes que cuentan con el soporte para comunicarse. Básicamente estos agentes son implementados como clases de java . Hay dos tipos de Agent Shell, en el primero las clases son derivadas de una clase base y cuentan con el soporte de transporte de mensajes, recuperación de mensajes y transferencia y almacenamiento de mensajes. El otro tipo de shell soporta mensajes ACL. Además los agentes pueden ser desarrollados independientemente, y no residir dentro de la plataforma, pero pueden comunicarse con los agentes residentes a través de un API de transporte.

Para entender un mensaje ACL este debe ser procesado según su posición temporal dentro de una secuencia de mensajes entre dos o mas agentes. Para esto se debe conocer el tipo de comunicación, la estructura del contenido y finalmente su semántica. Los cuatro componentes principales de ACL son: conversación o interacción de mensajes, mensaje ACL, estructura del contenido y semántica.

La plataforma **Grasshopper** (desarrollada por GMD FOKUS y IKV++ GmbH) [123] adopta los estándares de OMG MASIF[3] y FIPA, y permite desarrollar desde agentes móviles a sistemas multi-agentes estáticos que se comunican mediante Agent Communication Language (ACL) para resolver problemas en forma distribuida.

Grasshopper implementa un *Distributed Agent Environment* (DAE), el cual está compuesto por regiones, lugares, agencias y diferentes tipos de agentes. Un *Lugar* provee un agrupamiento

[3] Object Managment Group's *Mobile Agent System Interoperability Facility* (OMG MASIF) es el primer estándar de la industria de agentes móviles.

lógico de funcionalidad dentro de una agencia. El concepto de *región* facilita la administración de componentes distribuidos (agencias, lugares y agentes). Tanto las *agencias* como sus *lugares* pueden estar asociados con *regiones* específicas, lo cual es registrado en el registro de regiones. Todos los agentes están alojados en las *agencias*, las cuales son registradas automáticamente dentro del registro. Si un agente se mueve a otra localización el registro se actualiza en forma automática.

En esta plataforma se identifican dos tipos de agentes: móviles y estacionarios. El entorno actual donde se ejecutan ambos tipos de agentes es una *agencia*, en cada host hay al menos una agencia la cual consiste de dos partes: el corazón y uno o mas lugares. El corazón representa la funcionalidad mínima requerida para soportar la ejecución de los agentes y provee los siguientes servicios: comunicación (permite la localización y comunicación transparente entre agentes, lugares y otras entidades), registro (cada agencia debe conocer todos los agentes y los lugares donde están), administración de servicios (controlar los agentes y lugares de una agencia), seguridad y persistencia (a través de un medio persistente se pueden almacenar agentes y lugares).

Zeus es un sistema de agentes Open Source implementado en Java y desarrollado por los laboratorios BT (British Telecom). Puede considerarse como una herramienta para la construcción de aplicaciones multi-agentes colaborativos. Esta plataforma provee el soporte para implementar la funcionalidad de los agentes y para realizar tareas sofisticadas como planning y scheduling de las acciones de los agentes.

El soporte de comunicación de esta plataforma responde a los estándares de transporte de mensajes de FIPA ACL y y como mecanismo de entrega sockets TCP/IP. Esta plataforma provee un entorno visual para construir agentes y dirigir el comportamiento de los mismos.

El enfoque de planning y scheduling usado representa las metas y las acciones usando descripciones que incluyen los recursos requeridos y las precondiciones que sedeben cumplir.

Otras plataformas que han sido propuestas son Comtec Agent Platform, JACK Intelligent

Agents, JAS (Java Agent Services API) y LEAP.

En [121] se define un MAS (Sistema Multi-Agentes) como un modelo descentralizado, distribuido y abierto a diferentes sistemas y entornos. Un MAS está integrado por un conjunto de agentes que trabajan juntos a través de una red para resolver problemas, sus ventajas principales son: descentralización, reuso de componentes, trabajo cooperativo, flexibilidad y simplicidad.

Los cuatro componentes de esta arquitectura son: Agentes, Workspaces, Repositorios, y Agoras. La definición de *agente* es similar a lo que hemos mencionado previamente en esta tesis, en resumen es una pieza de software autónoma que interactúa con otros agentes y/o con el usuario. El *workspace* es el lugar donde están los archivos y herramientas que son usados por los agentes, y en donde se desarrolla la interacción entre los agentes y los usuarios. Un *Agora* es un lugar donde los agentes se encuentran e interactúan y donde intercambian información y negocian. El principal objetivo de un agora es proveer el soporte de cooperación para los agentes. Además un agora ayuda a manejar entornos heterogéneos. En esta arquitectura un *repositorio* es un servidor de información y puede ser accedido por herramientas y agentes. La información que contiene incluye políticas e información de contexto, y permite a los agentes razonar sobre ella.

En [130] RETSINA MAS se define como un sistema multi-agentes que se basa en la idea de que opera en un mundo abierto, es decir que el entorno en que operan los agentes es abierto y dinámico, las capacidades y ubicación de los agentes es incierto así como la topología y el entorno de la red. También se asume que hay un grado de replicación funcional de forma que si un agente falla otro puede sustituirlo. RETSINA adopta la visión de sociedades multi-agentes, donde los agentes conforman estructuras sociales. La infraestructura del MAS constituye el sistema donde los servicios y componentes viven, se comunican e interactúan.

5.4 Aplicación: Plataforma de Agentes para brindar un servicio de reservación de pasajes de colectivo

5.4.1 Especificación del servicio de Reservación de Pasajes

Nuestro ejemplo consiste en un servicio de reservación de pasajes para trasladarse dentro de las provincias de Río Negro y Neuquén, llamamos al servicio RePaCo el cual es implementado combinando los servicios correspondientes a las empresas que realizan viajes dentro de las provincias antes mencionadas y un servicio de búsqueda de una ruta (cubierta por una o mas empresas) que permita ir desde un lugar a otro.

El servicio RePaCo debe consultar el servicio de las empresas de colectivo, el cual puede brindar información y efectuar reservas, y al servicio de búsqueda para combinar los viajes que sean necesarios para satisfacer el requerimiento del usuario.

Para hacer esto describimos las interacciones esperadas entre el servicio RePaCo y otros actores. En el caso de las empresas de Transporte las interacciones son definidas en término de que el servicio requerido sea aceptado. En el caso del Usuario, describimos las interacciones en término de los requerimientos que el usuario pueda enviar a RePaCo. En el caso del servicio de Búsqueda debe interactuar con el de Transporte para determinar la combinación de colectivos que permiten viajar de una ciudad a otra. En consecuencia el servicio RePaCo debería interactuar con tres actores: Transporte (conoce los servicios que brindan las empresas de transporte de colectivos), Búsqueda (cuando no es posible ir directamente de una ciudad a otra puede determinar si hay alguna combinación de colectivos que lo haga posible) y Usuario. A continuación se describen brevemente los actores que intervienen:

El servicio de Transporte acepta los requerimientos para proveer información sobre un viaje (origen-destino) dado y, si el trayecto es cubierto por una empresa provee información sobre el costo, horario de salida desde origen y llegada al destino, el tipo de coche y el tipo de servicio que brinda. Este servicio también acepta requerimientos para reservar un pasaje, en este caso

retorna la lista de asientos disponibles en el colectivo elegido. La oferta puede ser aceptada (con la elección de un asiento) o rechazada por el servicio externo que ha invocado al servicio de Transporte.

El servicio de Búsqueda será invocado cuando el servicio de Transporte de una respuesta negativa, lo que significa que ninguna empresa tiene viajes directos desde origen a destino. Este servicio se encarga de verificar si existe alguna combinación de colectivos (que pueden pertenecer a diferentes empresas) para ir de una ciudad a otra. Este servicio se debe comunicar con el servicio de Transporte para confeccionar un grafo de los viajes provistos por las empresas y buscar en él un camino (el mas corto).

El Usuario envía un requerimiento para obtener un pasaje para ir desde una ciudad a otra y espera una respuesta negativa si esto no es posible, o una oferta indicando el precio y demás datos del servicio. El usuario puede aceptar o rechazar la oferta, una interacción típica entre el usuario, el servicio RePaCo y la empresa del transporte sería como el siguiente:

- El usuario consulta a RePaCo por un viaje que desea hacer, indicando origen y destino;
- RePaCo le pregunta a las empresas de transporte los datos del servicio: costo, hora de salida y llegada, tipo de coche y de servicio;
- RePaCo provee al usuario una oferta con el costo y demás información;
- El usuario envía la confirmación a RePaCo y éste solicita la lista de asientos disponibles.
- RePaCo provee al usuario la lista de asientos entre los cuales el usuario elegirá uno.
- El usuario envía el asiento seleccionado a RePaCo.
- RePaCo envía el asiento al servicio de Transporte para que se efectúe la reserva.

Este es un caso básico y se deberían considerar otras interacciones, por ejemplo para el caso que una sola empresa de colectivos no pueda satisfacer el requerimiento (en tal caso entra en

juego el servicio de Búsqueda), o que el usuario rechace la oferta final. La Figura 5.1 describe los flujos de datos entre los servicios intervinientes.

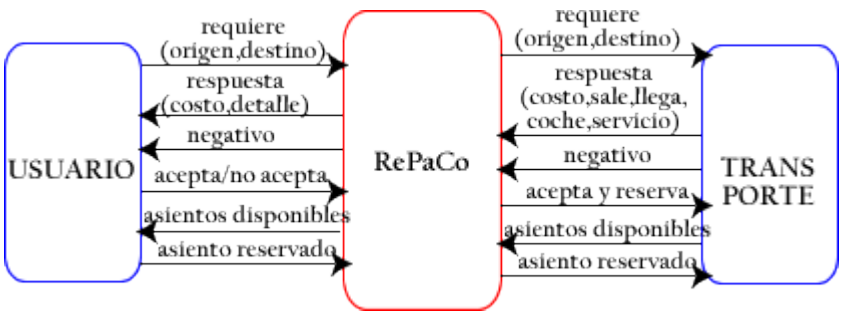

Fig 5.1. Diagrama de Flujos de Datos para un caso básico

5.4.2 Implementación

Usuario: una página Web en la cual la persona puede ingresar los datos, ver respuestas, aceptar hacer la reserva, selección de asiento, etc.

RePaCo: un agente que reciba la solicitud del Usuario y se comunique con el servicio de Transporte para saber cuales de ellas cubren el trayecto en cuestión. En caso de que ninguna empresa cubra el trayecto total se debe comunicar con el servicio que se encarga de buscar la combinación de trayectos que permitan llegar desde origen a destino. La información completa es proporcionada al usuario: el costo (suma de los costos si se requiere viajar con mas de una empresa), hora de salida y llegada, tipo de coche (en cada tramo) y tipo de servicio (en cada tramo). Finalmente si el usuario acepta la oferta se comunica con las empresas para obtener la lista de asientos disponible y le brinda al usuario esta información para que haga su elección. Finalmente concreta la reserva de los asientos seleccionados.

Transporte: Un agente que conoce de cada empresa los trayectos que recorre, el tipo de coche y servicio que brinda, los horarios de salida y llegada a cada ciudad y el costo.

Búsqueda: Un agente que se comunica con el agente de Transporte para obtener la

información de los viajes que brinda cada empresa de transporte. Con esta información arma un grafo sobre el cual podrá determinar el camino mas corto para ir desde la ciudad origen al destino.

5.4.3 Agentes

Página Web para poner en funcionamiento la Plataforma de Agentes

A través de una página Web se invoca a un servlet que permite levantar la plataforma de agentes. En la siguiente página que muestra el usuario podrá ingresar su solicitud.

Página Web que hace de nexo con el Usuario

A través de una página Web el usuario puede ingresar datos y ver resultados, se implementa a través de un servlet que se comunica con el agente RePaCo.

Agente que implementa el servicio RePaCo

Este es uno de los agentes principales de la plataforma, es el que recibe la solicitud del usuario consultando la posibilidad de viajar de una ciudad a otra. Este agente debe comunicarse con el agente que conoce los servicios ofrecidos por las empresas de transporte de colectivo y en caso de obtener respuestas positivas puede darle la respuesta al usuario; en caso de no encontrar un viaje directo debe comunicarse con el agente que conoce los caminos existentes (el agente de búsqueda) para saber si es posible darle una solución al usuario.

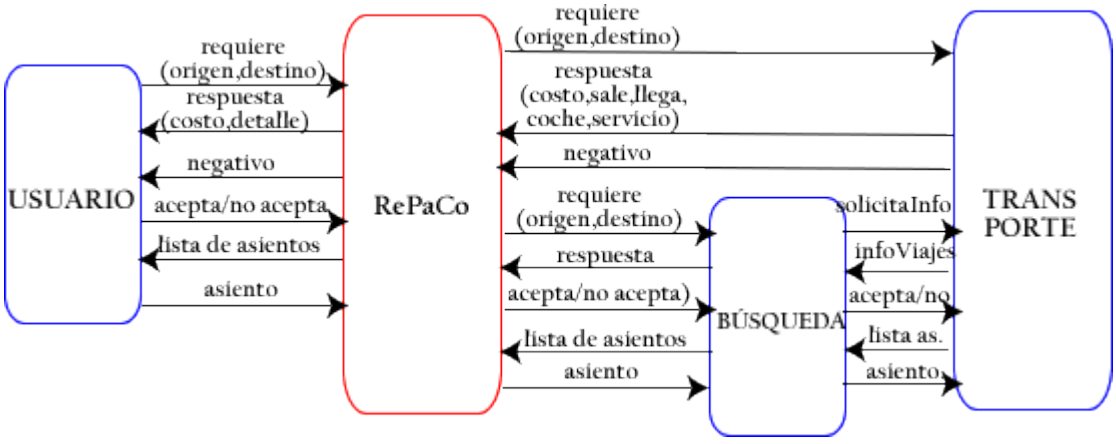

Fig 5.2: Diagrama de Flujos para el caso en que interviene el agente de búsqueda

Agente que implementa el servicio de las Empresas de Transporte

Este servicio brinda información sobre los viajes que realizan las diferentes empresas para lo cual usa la ontología ViajeColectivo.owl.

También puede recibir solicitudes para reservar asientos en un colectivo.

Este agente es invocado desde el agente RePaCo y desde el agente de Búsqueda.

Agente que implementa el servicio de búsqueda del camino mas corto de una ciudad a otra

Este servicio brinda información sobre los viajes que realizan las diferentes empresas para lo cual usa la ontología caminos.owl, para saber si existen caminos recorridos por las diferentes empresas para llegar de una ciudad a otra.

Este agente utiliza grafos para representar la información y realizar la búsqueda del camino mas corto de una ciudad a otra (el camino mas corto cubierto por las diferentes empresas de colectivos). El agente debe generar el grafo al inicializarse a partir de la ontología de caminos donde los nodos representarán a las ciudades y los arcos los caminos (para los cuales hay al menos una empresa de colectivos que los recorre) entre ellas, los cuales serán rotulados con la distancia. Luego el agente responderá a solicitudes para hallar el camino mas corto entre dos ciudades.

5.4.4 Ontologías

Una ontología es una descripción explicita y formal de conceptos en un dominio de discurso (clases (a veces llamadas conceptos)), propiedades de cada concepto describiendo varias características y atributos del concepto (slots (a veces llamados roles o propiedades)), y restricciones sobre los slots (facetas (algunas veces llamados restricciones de rol)). Una ontología junto con un conjunto de individuos de clases constituye una base de conocimiento.

Los pasos sugeridos para desarrollar una ontología son:

Paso 1. Determinar el domino y alcance de la ontología

Paso 2. Considerar la reutilización de ontologías existentes

Paso 3. Enumerar términos importantes para la ontología

Paso 4. Definir las clases y la jerarquía de clases

Paso 5. Definir las propiedades de las clases: slots

Paso 6. Definir las facetas de los slots

Paso 7. Crear instancias

Ontología para la implementación de un servicio de reservación de pasajes

Paso 1. Determinar el domino y alcance de la ontología

- ¿Cuál es el dominio que la ontología cubrirá?

  El dominio es el de reservación de pasajes terrestres (colectivo) en el área provincial.

- ¿Para qué usaremos la ontología?

  Esta ontología será usada para el desarrollo de un servicio de búsqueda y reservación de pasajes para viajar desde una ciudad a otra.

- ¿Para qué tipos de preguntas la información en la ontología debería proveer respuestas?

  Preguntas referidas a la existencia de viajes desde una ciudad a otra, la disponibilidad de asientos, el horario de partida y llegada de un micro a una ciudad, el tipo de servicio (coche cama, semi-cama o comun, con o sin refrigerio).

- ¿Quien usara y mantendrá la ontología?

  El servicio RePaCo.

Paso 2. Considerar la reutilización de ontologías existentes

No usé ontologías existentes.

Paso 3. Enumerar términos importantes para la ontología

Viaje	Ciudad	Salida	Llegada	Servicio
Coche-cama	Semi-cama	Común	Desayuno	Almuerzo
Cena	Café	Video	Azafata	Escalas
Abajo	Arriba	Calefacción	Aire acondicionado	Baño
Asientos	Disponibilidad	Completo	Combinación	Tiempo de espera

Empresa	Trayectoria	Chofer	Co-chofer	Número coche
Vianda	Individual	De a dos	Origen	Destino

Paso 4. Definir las clases y la jerarquía de clases

- Asientos
  - Asientos abajo
    - De a dos
    - Individual
  - Asientos arriba
    - De a dos
    - Individual
- Servicios
  - Coche-cama
    - con vianda
    - sin vianda
  - Semi-cama
    - con vianda
    - sin vianda
  - Común
    - con vianda
    - sin vianda
- Coches
- Trayectos
- Viaje

Paso 5. Definir las propiedades de las clases: slots

- Asientos (número, piso, individual?, libre?)
- Coche (número, cant_asientos-arriba, cant_asientos_abajo, tipo_servicio)
- Servicio (refigerio?, azafata?, video?, toilet?, calefacción?, aire_acondicionado?)
- Trayecto (origen, destino, salida, llegada)
- Viaje (origen, destino, salida, llegada, coche, servicio)
- Ciudad (nombre)
- Empresa (nombre)

Paso 6. Definir las facetas de los slots

- Asientos
  - Número: entero
  - Piso: enumeración: abajo – arriba (tipo Symbol)
  - individual?: booleano
  - libre?: booleano
- Coches
  - Número: entero
  - cant_asientos-arriba: entero

    - cant_asientos_abajo: entero
    - tipo_servicio: enumeración: coche-cama - semi-cama – común
    - asiento: instance: asientos → múltiple cardinalidad, min=30, max=50
- Servicios
    - Nombre: string
    - Refigerio: booleano
    - Azafata: booleano
    - Video: booleano
    - Toilet: booleano
- Trayectos
    - Origen: instance: ciudad
    - Destino: instance: ciudad
    - Salida: String
    - Llegada: String
    - Coche: instance: coche
    - Servicio: instance: servicio
    - Empresa: instance: empresa
    - Precio: número Float
- Viajes
    - Origen: instance: ciudad
    - Destino: instance: ciudad
    - Trayecto: instance: trayecto → múltiple cardinalidad, mín=1
- Ciudades
    - Nombre: String
- Empresas
    - Nombre: String

Paso 7. Crear instancias

Por ejemplo si el usuario desea viajar de neuquén a Villa la Agostura las siguientes instancias estarían implicadas:

Empresas:

Algarrobal

Tuc

Coches:

Coche-común

Coche-cama

Servicios:

Simple:

Refigerio: no
Azafata: no
Video: no
Toilet: no
Calefacción: no
aire_acondicionado: no

Completo:
Refigerio: si
Azafata: si
Video: si
Toilet: si
Calefacción: si
aire_acondicionado: si

Ciudades:

Bariloche
Villa la Angostura
Neuquén

Instancias de trayectos:

Trayecto Neuquén-Bariloche:
Origen: Neuquén
Destino: Bariloche
Salida: 6hs
Llegada: 23,15hs
Coche: Coche-cama
Servicio: Azafata
Empresa: Tuc
Precio: $75

Trayecto Bariloche-Villa la Angostura:
Origen: Bariloche
Destino: Villa la Angostura
Salida: 8hs
Llegada: 10hs
Coche: Coche-común
Servicio: Simple
Empresa: Algarrobal
Precio: $25

Instancia de Viajes:

Viaje Neuquén-Villa la Angostura:

Origen: Neuquén

Destino: Villa la Angostura

Trayecto: Neuquén-Bariloche y Bariloche-Villa la Angostura

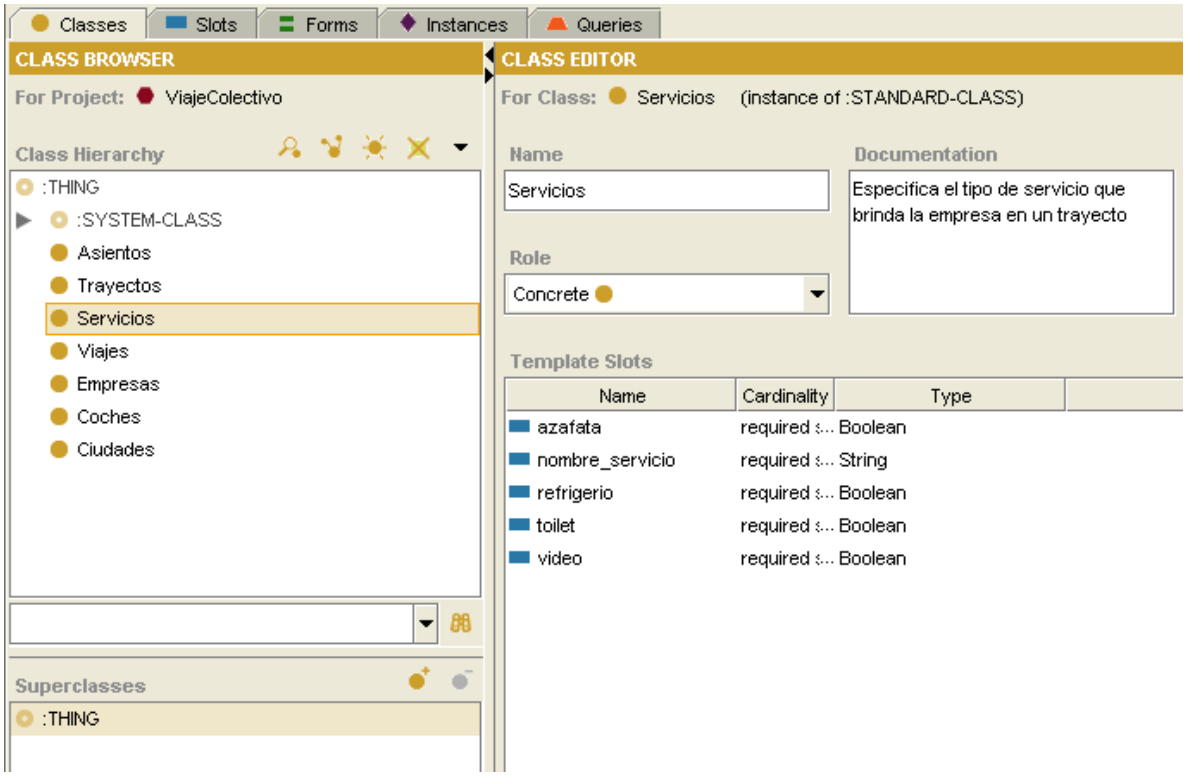

Fig 5.3: interfaz de Protegé con la jerarquía de clases del proyecto ViajeColectivo

Fig 5.4: interfaz de Protegé con instancias del proyecto ViajeColectivo

Ontología de caminos que unen ciudades

Paso 1. Determinar el domino y alcance de la ontología

- ¿Cuál es el dominio que la ontología cubrirá?

  El dominio es el de reservación de pasajes terrestres (colectivo) en el área provincial.

- ¿Para qué usaremos la ontología?

  Esta ontología será usada para saber si es posible llegar de una ciudad a otra aunque para ello haya que hacer escalas y transbordos en ciudades intermedias.

- ¿Para qué tipos de preguntas la información en la ontología debería proveer respuestas?

  Preguntas referidas a la posibilidad de viajar desde una ciudad a otra para lo cual no hay viajes directos. Esta ontología permitirá establecer la/s combinación/es de viajes que hagan posible ir de una ciudad a otra, la disponibilidad de asientos, el horario de partida y llegada de un micro a una ciudad, el tipo de servicio (coche cama, semi-cama o común, con o sin refrigerio).

- ¿Quien usara y mantendrá la ontología?

  El servicio de Búsqueda.

Paso 2. Considerar la reutilización de ontologías existentes

No use en este caso.

Paso 3. Enumerar términos importantes para la ontología

Viaje	Ciudad	vecina
Ruta	Distancia	Camino

Paso 4. Definir las clases y la jerarquía de clases

- Ciudades
- Vecinas
- Viaje
- Camino

Paso 5. Definir las propiedades de las clases: slots

- Ciudades (nombre)
- Caminos (numero, ciudad1, ciudad2, distancia)

Paso 6. Definir las facetas de los slots

- Ciudades
    - Nombre: String
- Caminos
    - Vecinas: instance: ciudad. Cardinalidad exactamente2.
    - Distancia: float

Paso 7. Crear instancias

Por ejemplo si el usuario desea viajar de neuquén a Villa la Agostura las siguientes instancias estarían implicadas:

Ciudades:

Bariloche
Villa la Angostura
Neuquén
Plottier
Senillosa
Piedra del Águila
Plaza Huincul

Caminos:

Vecinos: Neuquén - Plottier Distancia: 40	Vecinos: Plottier - Senillosa Distancia: 24	Vecinos: Senillosa - Plaza Huincul Distancia: 70
Vecinos: Plaza Huincul - Picún Leufú Distancia: 90	Vecinos: Senillosa - Picún Leufú Distancia: 100	Vecinos: Picún Leufú - Piedra del Águila Distancia: 92
Vecinos: Piedra del Águila - SC de Bariloche Distancia: 199	Vecinos: SC de Bariloche - Villa la Angostura Distancia: 83	

5.5 Resumen

En este Capítulo hemos descrito las características de las tecnologías que hacen posible la implementación de los servicios Web Semánticos, tanto los agentes como las plataformas de agentes que permiten realizar la composición, ejecución y orquestación delos servicios Web.

Los agentes son entidades de software que tienen un conjunto de características: autonomía, inteligencia, interacción, sociables y movilidad. Estas características hacen que sean adecuados para realizar la implementación de servicios Web, los cuales se ejecutan en entornos dinámicos, a través de la interacción en forma automática con otros servicios que pueden encontrarse distribuidos en la red y cuya disponibilidad, y otras características, pueden variar en el tiempo.

Las plataformas de agentes permiten implementar los servicios a través de agentes y brindan la infraestructura básica para posibilitar el desarrollo del ciclo de vida de los agentes así como su movilidad, brindan servicios de páginas blancas y amarillas, el soporte de transporte y traducción de mensajes, implementan cuestiones de seguridad, permiten realizar la orquestación de los servicios, etc.

Hemos especificado también un sistema de agentes que implementa la reservación de pasajes de colectivo desde una ciudad a otra considerando la posibilidad de cubrir todo el trayecto mediante diferentes empresas. Se ha realizado el diseño del sistema para lo cual se han definido los agentes que implementan cada servicio, se ha diseñado la ontología a utilizar en este sistema utilizando el editor de ontologías *Protégé*. La plataforma de agentes **JADE (*Java Agent DEvelopment Framework*)** fue instalada para realizar la implementación de los agentes. Se ha comenzado a estudiar dicho entorno y se preve la implementación del sistema de agentes para el futuro.

En el capítulo siguiente se realizará un análisis y comparación de las tecnologías para realizar la composición de servicios Web previamente descritas.

Capítulo 6:

Conclusiones

6.1 Estado del Arte

Dada la heterogeneidad de los sistemas existentes, tanto en las plataformas como los lenguajes usados, la integración de los mismos es una misión complicada. Los servicios Web son una tecnología que posibilitaría el desarrollo de complejas aplicaciones construidas sobre la base de componentes de software disponibles en la Web, pertenecientes a diferentes empresas y ejecutándose en plataformas heterogéneas. Por ello, uno de los temas a los que se han abocado los investigadores es la composición de los servicios Web, para posibilitar el reuso del software existente así como la integración de los mismos. En esta tesis se han estudiado las estrategias, tecnologías y plataformas existentes para realizar la composición de servicios. Al finalizar este capítulo se analizan las áreas que continúan inexploradas y que guían las actividades de investigación de los expertos.

El modelo básico de Servicios Web consiste de tres tipos de entidades: el proveedor de servicios, el registro y el cliente que consume los servicios. El proveedor de servicios crea un servicio y lo ofrece, haciéndolo conocer a través del registro en el cual debe publicar la descripción en algún lenguaje basado en XML. El consumidor del servicio encuentra su descripción en el registro y la información para ponerse en contacto con el proveedor e invocar el servicio Web. Los tres pasos básicos son publicar, ligar e invocar. Para que sea posible la comunicación entre aplicaciones que se ejecutan en diferentes plataformas es necesario contar con estándares para realizar cada uno de estos pasos. Como se ha estudiado en los capítulos previos dichos estándares están constituidos básicamente por:

- Web Service Description Language WSDL, que usa el formato XML para describir los servicios Web (parámetros de entrada y salida, tipos de dato y protocolos de transporte tales como HTTP, etc).

- Universal Description Discovery and Integration standard UDDI, es el registro donde se publica la información sobre el proveedor de servicios y los servicios ofrecidos.

- Simple Object Access Protocol SOAP, es un protocolo usado para el intercambio de información XML entre las entidades.

Esta infraestructura puede ser suficiente para un escenario simple donde solo hay interacciones entre un cliente y un servicio Web. Pero cuando la implementación de un servicio Web requiere la combinación de otros servicios existentes surge la necesidad de replantear este esquema. En un servicio compuesto la lógica del negocio es implementada por varios servicios. El uso de lenguajes de programación convencionales y las plataformas heterogéneas en que conviven los servicios Web no son suficientes para realizar la composición de servicios Web, se requiere un middleware que provea la abstracción e infraestructura necesarias. Un middleware debe proveer mínimamente un lenguaje y modelo de composición, un entorno de desarrollo con una interfaz gráfica para incorporar y sacar componentes y un entorno de ejecución para ejecutar la lógica del negocio.

La composición de servicios presenta varios problemas que han sido analizados en esta tesis, en especial cuando se pretende automatizar esta tarea:

- Especificación ambigua y observabilidad parcial. El modelo de interacción del usuario puede variar en el grado de completitud y observabilidad y ser diferente al modelo de interacción del servicio compuesto ya que el cliente no sabe cómo está constituido el servicio compuesto, lo cual dificulta la tarea de encontrar un servicio Web que satisfaga los requerimientos.

- Especificación ambigua o incompleta de los servicios Web (falta marcado semántico), la

incorporación de semántica y el uso de ontologías permiten superar este problema. También hay otras propuestas como las técnicas de aprendizaje, INDIGO es un sistema de planificación que puede realizar el marcado semántico de los servicios[147].

- Soporte de Cambios. El entorno cambiante afecta la correctitud de la composición diseñada, hay algunas propuestas como las técnicas de adaptación de workflow para afrontar este problema. También si la interfaz de un servicio cambia ésto no debe afectar el servicio compuesto; una solución es construir una capa de funcionalidad sobre la pila de estándares de composición, como se describe en [93].

- Acuerdo entre las partes que interactúan. Por un lado los sistemas heterogéneos donde se ejecutan e interactúan los servicios imponen la necesidad de plantear estándares que permitan superar las incompatibilidades. Además debe existir un acuerdo entre el servicio compuesto y los servicios individuales, éstos últimos pueden especificar políticas con relación a la participación en un servicio compuesto [19]. También es necesario realizar la verificación de la composición para comprobar la sanidad y correctitud de la misma.

6.2 Cuestiones que se deben atender en la Composición de Servicios Web

La composición de servicios Web debe atender algunas cuestiones que se enumeran a continuación.

1) Coordinación

Cuando se componen servicios se requiere la coordinación de la ejecución de las operaciones. Los protocolos usados en el modelo inicial, tales como WSDL no son suficientes, ya que por ejemplo no se puede expresar el orden en que deben ejecutarse las operaciones ni expresar qué información debe ser obtenida antes de realizar una operación dada. Tampoco es posible expresar si una operación debe ser realizada dentro de una transacción. WSCI es un estándar que provee nuevas construcciones para expresar el comportamiento de un servicio en el contexto de un

proceso dado, el cliente que invoca el servicio sabe cómo se debe interactuar con el servicio.

Otros estándares surgidos para atender esta cuestión son WS-Coordination de IBM y OASIS [139][141] y y WS-CF de Sun y ORACLE [140] y http://www.infoworld.com/d/developer-world/sun-oracle-others-propose-transaction-specificaton-062 [28].

La principal capacidad del WS-Coordination [139] es la posibilidad de registrar un servicio como integrante de una función específica de una clase de dominio.

WS-Context provee un modelo de sesión para el entorno de los servicios web. Los mensajes SOAP que son procesados dentro del alcance de una actividad contienen un header de contexto que permite identificar claramente la actividad. WS-Coordination extiende este modelo con protocolos para definir el contexto de registración extendiéndolo para registrar el end-point de registración. La registración en el contexto de una actividad agrega información sobre el grupo de actividades, lo cual puede ser de utilidad para coordinar señales.

La coordinación es un requerimiento presente en varios aspectos de las aplicaciones distribuidas, por ejemplo en workflow, transacciones atómicas, replicación, seguridad, actividades business-to-business, etc.

Este framework está integrado por tres servicios componentes[141]: un servicio de Activación, con una operación que le permite a una aplicación crear una instancia o contexto de coordinación; un servicio de Registración, con una operación que le permite a una aplicación registrar protocolos de coordinación; y un conjunto de tipos de protocolos de coordinación. Estos elementos se ver diagramados en la Figura 6.1.

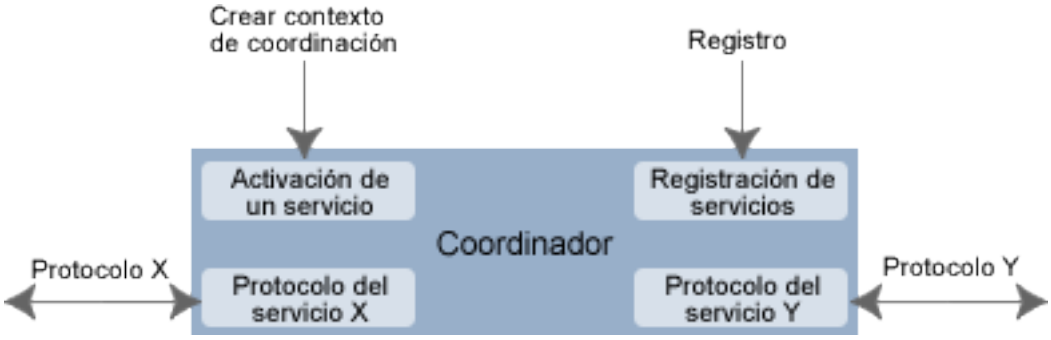

Fig.6.1 Elementos que integran el framework de coordinación.

Las aplicaciones crean el contexto de coordinación de una actividad usando el servicio de activación que luego puede ser enviado a otras aplicaciones. Una aplicación que recibe un contexto puede usar el servicio de registración de la actividad original o el impuesto por el coordinador. El contexto de coordinación sirve para pasar información entre los servicios que son coordinados.

2) Transacción

Las transacciones son un concepto fundamental para la construcción de aplicaciones distribuidas confiables, definiendo un mecanismo para asegurar que los participantes en una aplicación respetan mutuamente un acuerdo. Cuando las operaciones deben ser ejecutadas dentro de transacciones deberían asegurarse que se cumplen los requisitos de atomicidad, consistencia, integridad y durabilidad. Algunos estándares que han surgido para atender esta cuestión son WS-Transaction [141][142] [23,24, 25] propuesto por IBM y WS-TXM [30] propuesto por Sun.

La especificación de WS-Transaction está integrada por [141]: *Protocolos para transacciones atómicas*, que manejan actividades de corta duración. El Completion protocol es usado por una aplicación para decirle al coordinador si una transacción fue completada (commit) o abortada. El protocolo de dos fases (2PC) define como múltiples participantes acuerdan sobre la terminación de las transacciones: en la primer fase todo el trabajo es completado, siendo exitoso el resultado de una actividad si todas las operaciones fueron realizadas o no exitoso si no todas las actividades pudieron ser realizadas, lo ocurrido es informado a otros usuarios en la segunda fase. *Protocolos para transacciones de negocios*, manejan actividades largas, por lo que se requieren

mecanismos para resolver fallas y mecanismos de compensación de forma de que las actividades que han sido completadas exitosamente no sean afectadas.

El contexto de coordinación puede tener un atributo Expires, que especifica el momento máximo en que la transacción puede terminar, en caso de no cumplirlo se puede decidir hacer un roll back de la transacción.

3) Contexto

Una actividad representa la ejecución e interacción de un conjunto de servicios Web, estas interacciones se relacionan a través del contexto, y se define así un agrupamiento conceptual de servicios que cooperan para realizar una tarea. El contexto es la forma concreta en el cual se efectúa dicho agrupamiento, y es representado por la información usada por los servicios que debe ser actualizada durante la ejecución. Un estándar para realizar esta especificación es WS-Context [144] [31] que integra los estándares WS-CAF propuestos por Sun.

El elemento principal de la especificación del WS-Context es la estructura del contexto, la cual define el modelo organizacional de la información del contexto, a través de estructuras anidadas (relación padre-hijo) y mecanismos para pasar información del contexto. Se define el *Contexto del Servicio* para manejar los contextos de las actividades, define el alcance de una actividad y cómo la información sobre ella puede ser referenciada y propagada en entornos distribuidos. Un contexto es identificado por una URI y puede ser referenciado por múltiples servicios Web que están asociados a la misma actividad. El Contexto del Servicio mantiene un repositorio con todos los contextos compartidos, los cuales pueden ser propagados dentro de mensajes.

4) Conversación

Modelar las conversaciones facilita el descubrimiento y ligadura dinámica de servicios, validación de la composición y la generación de templates de composición. Algunos estándares creados para apuntar a esta cuestión son WSCL(Web Services Conversation Language) [10] y

WSCI (Web Service Choreography Interface) [128]. También están relacionados con este tópico el WS-Coordination y WS-Transaction ya que proponen las conversaciones específicas que pueden usarse para coordinar la interacción entre las partes.

6.3 Enfoques

Los investigadores han conducido sus trabajos a través de varias dimensiones, a continuación presentamos un análisis de los enfoques que se han propuesto para atender las cuestiones relacionadas con la composición de servicios Web:

- Composición estática vs. composición dinámica: considera el momento en que es realizada la composición. Algunos enfoques son: composición dirigida por modelos, composición dirigida por reglas del negocio, y composición declarativa.

  En la composición estática los servicios son elegidos y ligados en tiempo de diseño, Microsoft Biztalk y Bea WebLogic son algunos motores de composición de este tipo. Si un servicio es reemplazado por otro o deja de funcionar se debe ligar al nuevo servicio y algunos casos puede requerir redefinir los procesos del negocio. La composición dinámica permite la adaptación a los cambios de este tipo, un motor que entra en esta categoría es eFlow y StarWSCoP.

- Composición manual vs. automática: en el segundo caso entran en juego las tecnologías de la Web Semántica. La Web Semántica provee la información necesaria para determinar los servicios que mejor se adecuan para alcanzar una solicitud. Y a través de diversos enfoques tales como Procesos de negocios e Inteligencia artificial se puede llevar a cabo la composición automática. Las técnicas de Inteligencia Artificial han resultado adecuadas para realizar la tarea ya que es posible razonar, realizar inferencias, adaptar la composición diseñada a cambios en el entorno, etc. También el uso de workflow y las técnicas de adaptación presentadas en varias propuestas como en [44].

  Los principales problemas para implementar la composición automática de servicios son la selección y la interoperación de los mismos, donde se debe enfocar en las características

sintácticas y semánticas de los servicios y ser transparente al usuario. Para poder realizar esto las ontologías constituyen una conceptualización compartida y capturan la semántica de los servicios Web. Algunos sistemas propuestos que realizan composición dinámica son por ejemplo Colombo y StarWSCoP.

- Ejecución centralizada vs. ejecución descentralizada: cuando se trata de un servicio compuesto es necesario controlar la ejecución de los servicios implicados. Esto puede ser hecho en forma centralizada o descentralizada. La ejecución centralizada es similar al paradigma cliente-servidor donde el servidor controla la ejecución de los componentes, tal como lo hace la plataforma eFlow [87]. En la ejecución distribuida cada host tiene un coordinador que controla la ejecución de los servicios locales y debe colaborar con otros coordinadores para asegurar la ejecución ordenada y correcta de los servicios. La plataforma SELF-SERV [145] realiza este tipo de ejecución.

6.4 Comparación de los enfoques

Dado que los servicios Web están en auge en la actualidad por su capacidad para integrar aplicaciones dentro y entre empresas, dada la posibilidad de realizar esta integración en entornos y plataformas heterogéneas, el bajo acoplamiento de los componentes (servicios) y la posibilidad de adaptarse a cambios en el entorno, por ejemplo cambios en los parámetros de calidad, es que los investigadores han apuntado sus esfuerzos para mejorar las tecnologías y propuestas existentes. Una de las cuestiones que se han investigado y aún continúa en estudio es el de la composición automática y dinámica de los servicios Web. En el capítulo 3 hemos explicado en forma general las tecnologías sobre las que se basan muchas de estas propuestas, tales como las técnicas de planning y workflow, y en el capítulo 4 hemos estudiado muchas de dichas propuestas. En este apartado pretendemos realizar una comparación que nos permita vislumbrar todo el espectro de alternativas, analizarlas y compararlas, para poder hacer un análisis y reflexión sobre cada una de ellas en comparación con el resto.

Para realizar esta comparación debemos considerar varios espectros que ya hemos

analizado en este capítulo: coordinación, transacción, contexto, conversación, composición estática/dinámica, composición manual/automática, y finalmente composición centralizada/descentralizada.

Enfoques	**Representación del estado del mundo - contexto**	**Representación de la conversación - orquestación**	**Consideraciones**	**En ejecución**	**Ejemplos – referencias - estándares**
Máquinas de estados finitos	Estados	Colas de mensajes	Observador Clases mensajes	Mediador	Colombo[59] [98] [104] [105] [106]
Métodos de Planning	Situaciones	Acciones Precondiciones y efectos	FO-SMDP		Haley[45]
Cálculo de Situación	Situaciones Fluentes	Acciones	Observabilidad parcial	Servicios de búsqueda de información	[99]
Model Checking	Estados	Acciones	No determinismo Observabilidad parcial	Considera el contexto para ejecutar una acción Heurísticas	BPEL4WS EaGLe [64] [66]
Hierarchical Task-Network	Estados: átomos verdaderos	Acciones o Tareas	Descomposición de tareas	SHOP2 planifica el orden exacto de ejecución de las operaciones	Strips [65] SHOP2 [67
Workflow		Actividad de trabajo Reglas	Control y monitoreo de los procesos		AZTEC[107]

Tabla 6.1: Comparación de los enfoques de composición

En la clasificación realizada previamente podemos analizar la utilidad de los diferentes enfoques en las etapas de la composición: desarrollo del plan y ejecución del plan. Las técnicas que usan máquinas de estados finitos para diseñar la composición no se involucran con la etapa de ejecución de la misma, por lo cual son deficientes en el entorno de los servicios Web, donde se requiere una gran dinamicidad para poder adaptar la composición realizada en respuesta a cambios

del entorno, de los requisitos, de la calidad, de la disponibilidad de los servicios, etc.

Las técnicas de planificación y las de workflow permiten considerar ambas etapas, teniendo en cuenta que a la hora de ejecutar el servicio compuesto puede ser necesario adquirir información (consideración de la observabilidad parcial, uso de servicios de búsqueda de información, etc) y se considera también el comportamiento no determinístico de los servicios. Para que estas cuestiones puedanser consideradas se debe poder modificar el plan de composición a medida que se ejecuta. En el análisis que realizamos a continuación se efectúan las comparaciones entre las mismas.

	Características	**Composición estática/ dinámica**	**Composición manual/ automática**	**Ejecución centralizada/ descentralizada.**	**Adaptación**
Máquinas de estados finitos					
Colombo [59]	Estado del mundo Considera no determinismo, almacenamiento local, restricciones de integridad Colas para cada *tipo* de mensaje	Composición dinámica usando la información de cada servicio y del propio mediador	Logra automatización con técnicas basadas en Lógica proposicional dinámica	Mediador, se considera el valor de los parámetros según el almacenamiento local	--
Métodos de Planning					
Haley[45]	Enfoque jerárquico Lógica de primer orden para representar precondiciones y efectos	La composición es dinámica usando el conocimiento y las probabilidades	Calcula la probabilidad con que ocurren las acciones Crea una base de conocimiento Determina en forma automática el servicio	Se realiza en forma centralizada	--
Calculo de situación y **Golog**[99]	Usa estados, acciones, Usa calculo de	Los servicios web son compilados como cajas negras	Técnica de planning basada en operador	--	--

	situación para representar precondiciones y efectos Golog para representar acciones complejas				
Servicios Web **BPEL4WS**[64]	Usa model checking Considera no determinismo observabilidad parcial metas extendidas	Se realiza en forma dinámica	Composición automática Usa el estado	Se realiza en forma centralizada	Detecta señales externas Obtención de información del dominio
SHOP2 [67]	Enfoque jerárquico Puede ejecutar procesos en DAML-S	La composición es dinámica usando el razonamiento y las fuentes de información	Realiza la composición automática Utiliza las precondiciones y hace inferencia y razonamiento Integra fuentes de información externa	Se ejecutan las tareas en el orden exacto en que son planificadas	--
Métodos de Workflow					
AZTEC [107]	Diagramas de flujo Eventos Repositorio que representa el contexto	Selección y ensamble de servicios dinámicamente mientras se ejecutan	Esquemas de procesos usando HTN Selección y ensamble de servicios automáticamente Uso de políticas y un componente de administración	Motor de ejecución dirigido por eventos	Schema Management, permite modificar los esquemas durante la ejecución
Ejecución adaptiva en un modelo orientado a servicios [113]	Grafo (actividades como nodos y flujos de control y de datos como arcos) Repositorio que guarda información sobre servicios	--	--	--	Selección de servicios que cubran parte del workflow, consideraciones de QoS, mecanismo de pesos.
A-WSCE[42]	Chequeo de la	Creación de	--	El Runtime	Adaptación

	calidad en forma aleatoria Impacto de adaptación Estadísticas sobre volatilidad de los parámetros Múltiples workflows (VOC)	templates (etapa lógica) manualmente o usando planning Selección de un grupo de templates maximizando los valores de QoS		Mannager ejecuta el workflow chequeando los valores de QoS.	incremental Cambiar una instancia usando el mismo template (etapa física) Cambiar el template cuando no hay instancias disponibles (etapa lógica)
eFlow[87]	Los servicios se modelan como procesos de negocios, y son representados como grafos Considera transacciones	Reglas de selección de servicios para descubrimiento dinámico de servicios Los servicios se instancian en forma dinámica (en ejecución)	El usuario puede manualmente hacer cambios	Los procesos de negocios se configuran automáticamente Reglas de consistencia y de autorización	Responde a eventos (modificación dinámica de procesos) Descubrimiento dinámico de servicios Selección de la conversación dinámicamente Nodos multiservicio y genéricos

Tabla 6.2: Comparación de las técnicas de composición

Hemos analizado algunas de las propuestas mas relevantes en las dos técnicas mas utilizadas para realizar la composición de los servicios Web, técnicas que en general han evolucionado en un primer nivel para automatizar esta tarea y luego en un segundo lugar para considerar y adaptar la composición realizada a los cambios en el entorno.

Analizando el cuadro presentado podemos darnos cuenta que todas las propuestas que se basan en workflow para sintetizar la composición, tienen las características de ser dinámicas, automáticas y adaptables. Cada una de ellas con sus mecanismos propios. En cambio las técnicas que se basan en planificación en general no han evolucionado a tal punto, excepto la propuesta en la que se definen los servicios como Servicios Web en **BPEL4WS**[64] brinda estas características, las otras podrían evolucionar de la siguiente manera:

Haley podría adaptarse para considerar los cambios producidos en el entorno. De esta manera si se considerase el VOC se podría actualizar la composición realizada. Además según la propuesta en [150] la composición podría realizarse en forma adaptativa: si se considera la Volatilidad de la Información (VOC). Algo similar ocurre con la propuesta llamada SHOP2 . En [151] se introduce otra mejora a la propuesta llamada VOC, en la cual se supone que los proveedores pueden garantizar el valor de los parámetros por una cantidad de tiempo, por lo cual mientras el valor de los mismos no haya expirado no es necesario volver a consultarlos. Esta propuesta es llamada "V*alue of changed information with expiraton times"*.

El planificador Strips [65] se basa en un enfoque jerárquico: realiza la descomposición de tareas. Este tipo de planificadores asume que conoce el estado del mundo a través de un conjunto de átomos. La construcción del plan para llegar a la meta se realiza a través de heurísticas, y una función de transición que considera las precondiciones que deben cumplirse para poder ejecutar una acción. La suposición del mundo cerrado que hace Strips no es adecuada para el entorno dinámico y sumamente cambiante en el que se ejecutan los servicios Web, por este motivo esta propuesta ha evolucionado en otras que consideran esta cuestión.

En [152] se presenta una propuesta en la que se extiende el lenguaje Golog para considerar las preferencias de los usuarios al hacer la composición. La idea es que la composición de servicios sea mas flexible, para lo cual se crean templates de composición que luego son instanciados con las preferencias de los usuarios, las cuales serian especificadas en un lenguaje de primer orden. De esta manera tanto el sistema SHOP2 como la propuesta presentada en [99] podrían ser mas dinámica, aunque deberían incorporar otras características para poder adaptarse a los cambios en el entorno, para lo cual se debería analizar la posibilidad de utilizar el VOC definido en otras propuestas.

Las propuestas basadas en máquinas de estados finitos apuntan en general a cuestiones relacionadas con la compatibilidad de los servicios al realizar la composición. En dichas propuestas se confecciona un sistema de estados en el que los servicios se comunican a través del

pasaje de mensajes, contando con un almacenamiento local de los parámetros y en general un mediador que realiza la orquestación de los servicios, el cual cuenta con un almacenamiento local y usando técnicas de lógica proposicional puede decidir en forma automática y dinámica los servicios que interactúa. Una propuesta que ha sido ampliamente difundida y tomada como base para otras propuestas es Colombo, el cual debería ser considerado para incorporar características de adaptabilidad: podría incorporarse el uso del VOC por parte del mediador para decidir la composición mas adecuada a medida que ésta se ejecuta.

6.5 Estándares

Debido a que los servicios Web constituyen un sistema de software diseñado para soportar la interacción entre máquinas sobre una red, donde cada servicio puede ser implementado en diferentes lenguajes y sobre diferentes tipos de redes de computadoras, la adopción de estándares es fundamental para lograr la interoperabilidad de los servicios. The Organization for the Advancement of Structured Information Standards y el World Wide Web Consortium son los responsables de establecer los estándares y la arquitectura que lo soporte. También la Web Services Interoperability Organization (WS-I) se aboca a la integración de los estándares que garanticen y mejoren la interoperabilidad de los servicios web.

En este trabajo hemos analizado algunos de los estándares que integran la pila de protocolos de los servicios Web los cuales permiten describir, encontrar e invocar servicios así como la comunicación de los servicios entre si. Los estándares para la descripción de los servicios son WSDL (describe la interfaz de un servicio) y Web Service Policy (describe un servicio en término de políticas). Además se han definido lenguajes para la descripción semántica de los servicios: OWL-S, WSMF, WSMO, WSML, WSDL-S y SAWSDL. Y lenguajes para la especificación de vocabularios (ontologías): RDFS, DAML, DAML+OIL, DAML-S o OWL, SWRL.

Finalmente los lenguajes para la especificación de la composición propuestos son: BPEL4WS, WSFL, DAML-S, WSCI, WSCL, BPML, PDDL, BPSS.

El análisis y descripción realizada de cada uno de estos estándares, y especialmente la clasificación propuesta de acuerdo a su funcionalidad, son especialmente útiles para los investigadores que deben comenzar a trabajar en el ámbito de servicios Web y Web semántica ya que resuelve un análisis que debería ser realizado para ubicarse en este contexto.

6.6 Arquitecturas y Plataformas para Composición de Servicios

Existen plataformas de Servicios Web que responden el modelo simple de proveedores, clientes y el registro de servicios, tales como HP e-speak o Sun Jini, que permiten el desarrollo y entrega de servicios seguros a negocios y clientes. Los proveedores pueden registrar las descripciones de sus servicios y controlar la ejecución de los mismos, y los clientes pueden descubrir e invocar los servicios.

Otras plataformas proveen algunas características que permiten realizar la composición de servicios, modificar la misma en tiempo de ejecución, observar la ejecución, etc.

La plataforma de composición de servicios **StarWSCoP** (Star Web Services Composition Platform) se enfoca en la composición dinámica de los servicios, En esta plataforma WSDL es mejorado con atributos de QoS, tales como tiempo, costo y confiabilidad, para permitir la composición dinámica de los servicios. Además se incorpora el uso de ontologías al UDDI.

Algunas plataformas que utilizan **workflow** para realizar la composición de servicios son: **Mentor-lite**, es un sistema de worflow liviano que puede estar distribuido; **WIDE,** a través del manejo de transacciones aumenta la flexibilidad y permite mejorar la semántica de los workflow de procesos y soporte de reglas; **EVE,** usa un modelo basado en un Broker y Servicios para mejorar la definición de diversos aspectos del sistema de workflow; **AZTEC,** es un sistema de workflow donde las sesiones de los servicios son vistas como objetos de sesión que generan eventos y llamadas a funciones; **eFlow**, es una plataforma para especificar, representar y monitorear la composición de servicios.

Además se han realizado varias propuestas como las *técnicas de adaptación de wokflow* [44] que se basa en el monitoreo de los cambios en el entorno, en [42] se estudia el impacto de los cambios ocurridos sobre el workflow y se considera la posibilidad de adaptación mientras se ejecuta el workflow a través la generación de múltiples workflows y el uso de parámetros de QoS y la definición del Valor de la Información Cambiada (VOC)asociado a cada workflow.

Otras plataformas se basan en el uso de las **técnicas de planificación**, donde se supone que cada servicio puede ser especificado en términos de sus precondiciones, efectos y restricciones, por ejemplo para reflejar la lógica del negocio. Dentro de las técnicas de planificación podemos nombrar: Calculo de situación, es un método para modelar sistemas dinámicos basado en lógica de primer orden, modela en forma explícita el hecho de que distintas situaciones se puedan dar en el tiempo; Planning basado en reglas, en este método se usan reglas de composición sintácticas y semánticas, el sistema **SWORD** utiliza esta técnica; Model Checking, se utilizan grafos que representan el estado actual del problema y se analiza el comportamiento de las soluciones parciales con respecto a las metas, el servicio y los requerimientos son las entradas del algoritmo de model checking y el plan es la salida; HTN, las tareas son el concepto central, el método se basa en la descomposición de las tareas sucesivamente hasta obtener tareas atómicas (o primitivas) las cuales pueden ser ejecutadas directamente por la invocación de algunas operaciones atómicas, la información semántica puede ser incorporada a través de construcciones como *efectos*, *condiciones*, y *restricciones y críticos,* el sistema **SHOP2** utiliza esta técnica.

Por último las **Máquinas de estado finito** son una alternativa a las redes de petri que permiten describir los procesos componentes y los canales de interconexión, pero solo ciertos protocolos pueden ser descritos, por ejemplo no puede describirse un protocolo que permite un número arbitrario de mensajes; Colombo es un sistema que utiliza esta técnica para representar la composición.

La gran cantidad de propuestas similares, cada una usando técnicas diferentes, resolviendo ciertos problemas previamente identificados pero dejando otros abiertos, implican que los

investigadores deban analizar y hacer un gran esfuerzo para identificar las similitudes y diferencias, así como las posibles ventajas y desventajas. Por este motivo consideramos de gran utilidad las comparaciones y análisis realizados en este capítulo ya que pueden servir como base para otros trabajos de investigación.

6.7 Resultados obtenidos

A partir del trabajo de investigación realizado en torno a los servicios Web y Web semántica y en particular sobre el tópico composición de servicios Web, se han producido una serie de trabajos de investigación que han sido publicados en workshop y congresos nacionales:

- "Una plataforma de Servicios Web". Ana Alonso de Armiño, Pablo Rubén Fillotrani. Publicado en XI Workshop De Investigadores De Ciencias De La Computación. San Juan. 2009

- "Incorporación de Semántica en plataformas para e-learning". Lidia Marina López. Ana Alonso de Armiño. Presentado en Jornadas de difusión científica de la Facultad de Economía y Administración. Neuquén. 2009

- "Incorporación de Semántica en plataformas para e-learning". Lidia Marina López. Ana Alonso de Armiño. Publicado en III Congreso de Tecnología en Educación y Educación en Tecnología. Bahía Blanca. 2008

- "Clasificación de los Lenguajes definidos en torno a Servicios Web y Web Semántica". Ana Alonso de Armiño, Pablo Rubén Fillotrani. Publicado en X Workshop De Investigadores De Ciencias De La Computación. General Pico 2008.

6.8 Conclusiones y Trabajo Futuro

El desarrollo de estándares en el ámbito de los Servicios Web ha sido fundamental para la rápida evolución y asentamiento de esta tecnología. La esencia de los servicios Web, que permite que dos entidades de software sean capaces de interactuar en forma automática,

independientemente de las plataformas en las que se ejecuten cada uno de ellos, es lo que hace atractivo y ha incentivado a la investigación de diversas comunidades científicas y académicas así como organizaciones y compañías de software. Es así que el paradigma Orientado a Servicios se sigue afianzando día a día con las propuestas y tecnologías que surgen del trabajo de grupos dedicados a profundizar en las diferentes cuestiones que aún permanecen abiertas. Es el caso de la composición de servicios Web, área que continúa en el foco de atención de los investigadores que siguen refinando y ampliando las propuestas con el fin de mejorar la capacidad de este paradigma para lograr el grado de automatización requerida.

En esta tesis hemos estudiado los estándares y tecnologías surgidos en torno a los servicios Web, y hemos analizado y comparado las técnicas de composición. Creemos que este trabajo puede ser utilizado por los investigadores como una documentación que les permita interpretar el contexto actual de los servicios Web, así como la evolución de los mismos, y en especial en torno a la composición de servicios Web semánticos. Es decir que puede servir como puntapie inicial para otros trabajos de investigación. Como trabajos futuros se pretende:

Estudiar el uso de servicios Web desde dispositivos móviles

El ámbito mas común de investigación, y al que nos hemos enfocado en esta tesis, es el de computación distribuida en una red de computadoras. Actualmente hay diferentes tecnologías que han evolucionado y se han afianzado, es en estos ámbitos a donde se abren nuevos horizontes que deben ser atendidos. Los servicios Web pueden estar disponibles a través de diferentes canales, por lo que pueden ser accedidos usando diferentes dispositivos como PCs, palmtops, teléfonos celulares o sets de TV, y a través de diferentes tecnologías de red y protocolos. el objetivo es proveer el mismo servicio a través de la Web, SMS o call centers. El Multichannel Adaptive Information Systems project MAIS propuesto en [154] es una plataforma, metodología y herramienta de diseño para la construcción de sistemas distribuidos basados en servicios electrónicos. Además se modela el sistema separando los niveles de aplicación y tecnológicos.

Las arquitecturas MAIS incluyen en su conjunto de información, además de información

sobre la red, los dispositivos y los proveedores de servicios, la descripción del contexto, la cual es usada tanto por los proveedores como por los consumidores de servicios. El administrador del contexto está ligado a una plataforma de interacción a través de un canal adaptivo, en el cual se envían los requerimientos de QoS a la plataforma de composición de servicios. Clientes con características heterogéneas y el número creciente de nuevos servicios requieren que puedan ser accedidos desde cualquier dispositivo en forma similar. Se define el término contexto como una clase de información. En [148] hay varios tipos de información de contexto como: l*ocalización*, que contiene información sobre la localización del consumidor como país, hora local, zona horaria, etc., también puede incluir información semántica como que el consumidor se encuentra en ese momento en su trabajo; *consumidor*, contiene información sobre quien invoca el servicio, como nombre, dirección de email, preferencias, etc.; *cliente*, contiene información sobre el cliente consumidor como hardware y software que usa.

Estudiar la integración de servicios Web provistos por dispositivos móviles

El acceso a los datos por sistemas móviles a través de servicios Web tiene la ventaja de ocultar la naturaleza heterogénea de los datos y proveer interfaces bien definidas para el acceso a los mismos. En este contexto el descubrimiento de servicios juega un rol critico ya que se deben considerar parámetros como la ubicación del servicio, capacidades de almacenamiento del dispositivo (hosting) y tipos de los resultados retornados. En [149] se proponen formas de incorporar el contexto en los mecanismos de búsqueda, confeccionando luego un índice basado en el contexto para mejorar la eficiencia del mecanismo.

Para desarrollar un directorio de servicios que considere el contexto los autores introducen el modelo llamado Multidimensional OEM (MOEM) que puede contener información para mostrar diferentes facetas de un contexto. El directorio de servicios es representado usando un grafo multidimensional OEM, en el cual se modelan los servicios como nodos atómicos que se ubican en las hojas. También hay otro tipo de nodos, los nodos de contexto y dos tipos de arcos, los arcos de contexto que son rotulados con información del contexto, y los arcos de entidades que

representan relaciones entre las entidades. En el grafo se realiza un algoritmo de búsqueda primero a lo ancho tomando como entrada una representación sintáctica del contexto, la cual es comparada con el contexto representado por cada arco y se analiza la intersección de dicha comparación. Los nodos atómicos encontrados en la búsqueda, los cuales representan servicios, son incluidos en los resultados.

En este trabajo también se propone realizar algún tipo de predicción sobre la disponibilidad de los servicios, en términos de tiempo y espacio geográfico, por ejemplo cuando y donde se espera que un servicio esté disponible en el futuro. Esto puede lograrse usando el estado actual o una secuencia de estados del dispositivo en donde está el servicio, analizando el comportamiento pasado del usuario es posible descubrir patrones que permitan realizar predicciones sobre la disponibilidad de los servicios en los dispositivos.

Estudiar la evolución de las técnicas de composición de servicios Web

La composición de servicios Web es un tópico relevante por lo que existen un gran número de propuestas que usando diferentes técnicas pretenden afrontar el problema. En esta tesis hemos identificado algunos de los problemas que afectan la automatización de la composición de servicios Web, hemos descrito algunas de las propuestas que pretenden resolver dichos problemas y hemos comparado los diferentes enfoques.

Como conclusión podemos decir que la elección de las alternativas para implementar los servicios Web y para llevar a cabo la composición, dependerán de las preferencias de cada desarrollador-usuario, ya que hay varias alternativas que han demostrado ser efectivas para realizar una implementación.

La tendencia de trabajo desde dispositivos móviles agrega otro camino a investigar, se pretende trabajar en ese sentido para investigar las posibilidades de lograr la integración e interoperación de las aplicaciones que se ejecutan en entornos heterogéneos, desde diferentes lugares y dispositivos, en un contexto que es dinámico y debe ser considerado para lograr la

correcta interoperación de las partes.

Apéndices

Apéndice A

Grafos

Los árboles pueden ser considerados como una generalización del concepto de lista porque permiten que un elemento tenga más de un sucesor. Los grafos aparecen como una extensión del concepto de árbol, ya que en este nuevo tipo de estructuras cada elemento puede tener, además de más de un sucesor, varios elementos predecesores. Esta propiedad hace a los grafos las estructuras más adecuadas para representar situaciones donde la relación entre los elementos es completamente arbitraria, como pueden ser mapas de rutas y caminos.

Los grafos se pueden clasificar en diferentes tipos dependiendo de cómo se defina la relación entre los elementos: podemos encontrar grafos dirigidos o no dirigidos y etiquetados o no etiquetados. También se pueden combinar ambas categorías. Formalmente, un grafo *G* consiste en dos conjuntos finitos *N* y *A*. *N* es el conjunto de elementos del grafo, también denominados *vértices* o *nodos*. *A* es el conjunto de *arcos*, que son las conexiones que se encargan de relacionar los nodos para formar el grafo. Los arcos también son llamados *aristas* o *líneas*.

Los nodos suelen usarse para representar objetos y los arcos para representar la relación entre ellos. Por ejemplo, los nodos pueden representar ciudades y los arcos la existencia de caminos que las comunican.

Cada arco queda definido por un par de elementos n_1, n_2 pertenecientes a *N* a los que conecta. Aunque habitualmente los elementos son distintos, permitiremos que sean el mismo nodo ($n_1 = n_2$). Representaremos gráficamente un arco como una línea que une los dos nodos asociados.

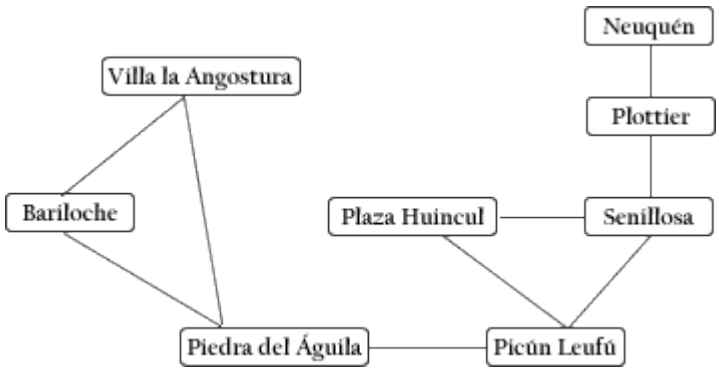

Fig A.1: grafo que representa las rutas entre ciudades

Se dice que dos nodos son *adyacentes* o *vecinos* si hay un arco que los conecta. Los nodos adyacentes pueden ser representados por pares (a, b).

Un *camino* es una secuencia de nodos n_1, n_2, ..., n_m tal que $\forall i$, $1 \leq i \leq (m-1)$, cada par de nodos (n_i, n_{i+1}) son adyacentes. Se dice que un camino es *simple* si cada uno de sus nodos, excepto tal vez el primero y el último, aparece sólo una vez en la secuencia.

La *longitud* de un camino es el número de arcos de ese camino. Se puede considerar como caso especial un nodo por sí mismo como un camino de longitud 0.

Un grafo no dirigido es un grafo donde los arcos conectan a los nodos en ambos sentidos.

Un nodo *N* se dice alcanzable desde un nodo *M* si y sólo si existe un camino desde *M* hasta *N*. Más formalmente, un nodo *N* se dice alcanzable desde un nodo *M* si:

(1) *N* y *M* son el mismo nodo, o

(2) *N* es alcanzable desde algún nodo que sea sucesor de *M*.

Para cada nodo de un grafo existe un conjunto de nodos alcanzables desde ese nodo, denominado *conjunto alcanzable*.

Un nodo *N* se dice *directamente alcanzable* desde un nodo *M* si y sólo si son adyacentes y *N* es el sucesor de *M*.

En ciertos casos es necesario asociar información a los arcos del grafo. Esto se puede lograr mediante una etiqueta que contenga cualquier información útil relativa al arco, como el nombre, peso, costo o un valor de cualquier tipo de datos dado. En este caso hablamos de *grafos etiquetados*. Esta etiqueta podría significar la distancia que hay entre dos ciudades.

Especificaciones del tipo de dato Grafo:

La especificación de un t*ipo de dato para implementar grafos no dirigidos debe incluir:*

- *1. Constructores:*
- *- **crear un grafo vacío,** operación constante que devuelve un grafo vacío (sin nodos ni arcos).*
- *- **añadir un nodo**, que devuelve un grafo con todos los nodos y arcos del grafo inicial junto con el nuevo nodo si no estaba ya o el grafo original si el nodo ya estaba incluido en el grafo y*
- *- **añadir una arista**, que devuelve un grafo con todos los nodos y arcos del grafo inicial y un arco nuevo entre dos nodos del grafo o bien el grafo original si ese arco ya existía. Esta operación tiene como precondición que ambos nodos pertenezcan al grafo.*
- *- **devolver un grafo sin un determinado nodo**, que devuelve un grafo con los nodos del grafo inicial excepto el que se borra y sin los arcos que contenían al nodo borrado o el grafo inicial si el nodo no pertenecía al grafo.*
- *- **o sin una determinada arista**, que devuelve el grafo que re*sulta de eliminar la arista indicada si existe o el grafo original si no existe.
- 2. Funciones selectoras:

- comprobar si un grafo **es vacío**,

- comprobar si un nodo **pertenece** al grafo, y

- si dos nodos **son adyacentes**.

Definición de las operaciones:

tipo Grafo
dominios Grafo, Elemento, BOOLEAN
generadores
Crear : → Grafo
Añadir_Nodo : Grafo × Elemento → Grafo
Añadir_Arista : Grafo × Elemento × Elemento → Grafo
constructores
Borrar_Nodo : Grafo × Elemento → Grafo
Borrar_Arista : Grafo × Elemento × Elemento → Grafo
selectores
Es_Vacio : Grafo → BOOLEAN
Contiene : Grafo × Elemento → BOOLEAN
Son_Adyacentes: Grafo × Elemento × Elemento → BOOLEAN
Nodo: Grafo → Elemento
Arista: Grafo → Elemento × Elemento
auxiliares
Hay_Arista : Grafo → BOOLEAN

En esta especificación establecemos como precondición de la operación Añadir_Arista que ambos nodos pertenezcan al grafo.

Búsqueda del camino de menor costo

La búsqueda de caminos de menor peso tiene sentido cuando hablamos de grafos etiquetados. El método general para resolver este problema es el llamado algoritmo de **Dijkstra**. En este algoritmo cada nodo se marca a su paso con una distancia provisional para cada nodo. Esta

distancia resulta ser la longitud del camino más corto desde el origen a N_i usando como nodos intermedios sólo nodos marcados. En CAMINO se almacena el último nodo que ha ocasionado un cambio en el valor de DISTANCIA.

En cada etapa, el algoritmo selecciona entre los nodos no visitados aquel que tiene el camino más corto al origen. El primer nodo seleccionado es el nodo origen, que tiene una camino de peso 0. Los demás tienen un camino de peso inicial ∞.

Una vez seleccionado el nodo, comprueba si se puede mejorar la distancia a sus nodos adyacentes no marcados. Si el peso del camino hasta el nodo seleccionado más el peso del arco hasta el nodo adyacente es menor que el peso del camino provisional hasta el nodo adyacente, modificamos su distancia, adaptándola a la del nuevo camino. El campo CAMINO también se actualizará a este nuevo nodo. El algoritmo acaba una vez que se hayan visitado todos los nodos.

Bibliografía

[1] "Resource Description Framework(RDF) Schema Specification 1.0". http://www.w3.org/TR/2000/CR-rdf-schema-20000327

[2] "About the DAML Language". http://www.daml.org/index.html

[3] "RDF Primer". W3C Recommendation 10 February 2004.

[4] "SWRL: A Semantic Web Rule Language Combining OWL and RuleML". W3C Member Submission 21 May 2004. http://www.w3.org/Submission/SWRL/

[5] D. Fensel. Vrije. "The Web Service Modeling Framework WSMF". Universiteit Amsterdam (VU). C. Bussler. Oracle Corporation.

[6] "Web Service Modeling Ontology (WSMO)". W3C Member Submission 3 June 2005

[7] "Web Service Semantics - WSDL-S". http://www.w3.org/Submission/WSDL-S/

[8] Semantic Annotations for WSDL and XML Schema. W3C Candidate Recommendation 26 January 2007

[9] Akhil Sahai, Carol Thomposn, William Vambenepe. "Specifying and constraining W.S. behaviour through policies". In W3C Workshop on Constraints and Capabilities for Web Services. 2004. http://www.w3.org/2004/08/ws-cc/hp-20040908

[10] "Web Services Conversation Language(WSCL) 1.0". http://www.w3.org/TR/wscl10/

[11] "OWL-S: Semantic Markup for Web Services". http://www.w3.org/Submission/OWL-S

[12] "DAML+OIL (March 2001)". Reference Description. W3C Note 18 December 2001.

[13] "Web Service Modeling Language (WSML)". W3C Member Submission 3 June 2005.

[14] "Web Services Description Language (WSDL)". http://www.w3.org/TR/wsdl20

[15] "Introduction to XML Schema". http://www.w3schools.com/schema/schema_intro.asp

[16] "Web Services Flow Language (WSFL)".http://xml.coverpages.org/wsfl.html

[17] "OWL Web Ontology Language Overview". http://www.w3.org/TR/owl-features/

[18] Jinghai Rao, Xiaomeng Su. "A Survey of Automated Web Service Composition Methods". In First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), ser. Lecture Notes in Computer Science, vol. 3387. Springer, 2004, pp. 43–54.

[19] B. Medjahed, A. Bouguettaya, A.K. Elmagarmid. "Composing Web Services on the Semantic Web". VLDB J., vol. 12, no. 4, pp. 333-351, 2003.

[20]http://www.w3.org/2005/04/FSWS/Submissions/4/BPT-PositioningPaperForW3CWorkshop05.html

[21] F. Giunchiglia, and P. Traverso. "Planning as Model Checking". IRST, Instituto per la Ricerca Scientifica e Tecnologica. Trento.

[22] K. Erol, J. Hendler, and D. Nau. "Semantics for Hierarchical Task Network Planning". 1994.

[23] S. R. Ponnekanti and A. Fox. "SWORD: A Developer Toolkit for Web Service Composition". In Proceedings of the 11th World Wide Web Conference, Honolulu, HI, USA, 2002.

[24] D. McDermott. "Estimated-regression Planning for Interactions with Web Services". In Proceedings of the 6th International Conference on AI Planning and Scheduling, Toulouse, France, 2002. AAAI Press.

[25] Tim Berners-Lee, James Hendler and Ora Lassila, "The Semantic Web", Scientific American, May 2001.

[26] Ian Horrocks And Peter F. Patel-Schneider. "Three Theses of Representation in the Semantic Web". Conference WWW 2003, mayo 20 -24 2003, Budapest.Hungary.p39-47.

[27] Ivan Herman, W3C. "Introduction to the Semantic Web".

[28] "GRDDL (**G**leaning **R**esource **D**escriptions from **D**ialects of **L**anguages)." http://www.w3.org/TR/2006/WD-grddl-primer-20061002/

[29] "POWDER (Protocol for Web Description Resources)." http://www.w3.org/TR/2009/REC-powder-dr-20090901/

[30] "SPARQL". http://www.w3.org/TR/rdf-sparql-query/

[31] "EARL". http://www.w3.org/TR/EARL10-Guide/

[32] S. McIlraith, T. Son. "Adapting Golog for Composition of Semantic Web Services". McGrawHill. 2002.

[33] H. Levesque, R. Reiter. "GOLOG: A Logic Programming Language for Dynamic Domains". Journal of Logic Programming, 31(1-3):59–84, April-June 1997.

[34] J. McCarthy, P. Hayes. 'Some Philosophical Problems from the Standpoint of Artificial Intelligence". Machine Intelligence, Vol. 4. 1997.

[35] "SOAP". http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

[36] A. Ankolenkar, M. Burstein. "DAML-S: Semantic Markup for Web Services". In The First International Semantic Web Conference (ISWC), Sardinia (Italy). 2003.

[37] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic composition of transition based semantic web services with messaging. In Proc. of VLDB 2005.

[38] Jhoachim. Peer. "A PDDL Based Tool for Automatic Web Services Composition". eds.: PPSWR. Volume 3208 of Lecture Notes in Computer Science., Springer (2004) 149–163.

[39] "WS-Addressing". http://www.w3.org/Submission/ws-addressing/#_Toc77464316

[40] "WS-Addredding". http://www.w3.org/standards/techs/wsaddr

[41] "OWL". http://www.w3.org/TR/2009/REC-skos-reference-20090818/

[42] G.Chafle, P.Doshi, J.Harney, S.Mittal, B.Srivastava. "Improved Adaptation of Web Service Composition using Value of Changed Information". In Proceeding of

IEEE International Conference on Web Services, Salt Lake City, UT, 2007. ICWS 2007.

[43] V.Agarwal, G.Chafle, K.Dasgupta, A.Kumar, S.Mittal, B.Srivastava. "Synthy: A System for end to end composition of Web services". J. Web Semantics, vol3:4, 2005.

[44] T.C.Au, U.Kuter, D.S.Nau. "Web Service composition with volatile information". En International semantic Web Conference, pages 52–66, 2005.

[45] H.Zhao, P.Doshi. "Haley: A Hierarchical Framework for Logical Composition of Web Services". International Conference on Web Services (ICWS), July, 2007.

[46] G.Chafle, K.Dasgupta, A.Kumar, S.Mittal, B.Srivastava. "Adaptationin Web service composition and execution". En International Conference on Web services (ICWS) 2006.

[47] P.Doshi, R.Goodwin, R.Akkiraju, K.Verma. "Dynamic workflow composition using markov decision processes". Jounal of Web Service Research (JWSR) 2005.

[48] S.Kambhampati. "Refinement search as Unitying framework for Analyzing Planning Algorithms". 1994.

[49] D.Wu, B.Parsia, E.Sirin. "Automating Daml-s Web services composition using shop2". En ISWC, 2003.

[50] H.Zhao, P.Doshi. "A hierarchical framework for composing nested Web processes". En ICSOC, 2006.

[51] M. Pistone, A.Marconi, P.Bertoli, P.Traverso. "Automated composition of Web services by planning at the knowledge level". En IJCAI, 2005.

[52] Hong Qing Yu, Stephan Reiff-Marganiec. "Semantic Web Services Composition via Planning as Model Checking". Technical Report CS-06-003, University of Leicester.

[53] K. Erol, J. Hendler, and D. Nau. "Semantics for Hierarchical Task Network Planning". 1994.

[54] R. Rodríguez, F. Sánchez, J. Conejero. "Modelando Procesos de Negocio Web desde una Perspectiva Orientada a Aspectos". Software Engineering Group, Universidad de Extremadura, 2002.

[55] R. Eliane, K. Kevin."Inteligencia Artificial". Segunda edición. McGrawHill Revista Tradumática. Noviembre 2003. Madrid 1994.

[56] D.Berardi, D.Calvanese, G.De Giacomo, R. Hull, M.Mecella. "Towards Automatic Web Service Discovery and Composition in a Context with Semantics, Messages, and Internal Process Flow".

[57] S. Narayanan and S. McIlraith. "Simulation, Verification and Automated Composition of Web Services". In Proceedings of the 11th International World Wide Web Conference (WWW 2002), pages 77 – 88. ACM Press, 2002.

[58] D. Berardi, F. Cheikh, G. De Giacomo, and F. Patrizi. Automatic service composition via simulation. Int. J. Found. Comput. Sci., 19(2):429–451, 2008.

[59] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. "Automatic composition of Web services in Colombo". In Proc. of 13th Itallian Symp. on Advanced Database Systems, June 2005.

[60] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. "Automatic Composition of e-Services that Export their Behavior". In Proceedings of the 1st International Conference on Service Oriented Computing (ICSOC 2003), volume 2910 of LNCS, pages 43–58. Springer, 2003.

[61] T. Bultan, X. Fu, R. Hull, and J. Su. "Conversation Specification: A New Approach to Design and Analysis of E-Service Composition". In Proceedings of the 12th International World Wide Web Conference (WWW 2003), pages 403–410. ACM, 2003.

[62] J. de Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, M. Kifer, B. K¨ nig-Ries, J. Kopecky, R. Lara, E. Oren, A. Polleres, J. Scicluna, and M. Stollberg. "Web Service Modeling Ontology (WSMO)". Technical report, DERI, 2005.

[63] S. W. S. L. working group. Swsl home page. http://www.daml.org/services/swsl/, 2005.

[64] M. Pistore F. Barbon, P. Bertoli, D. Shaparau, P. Traverso. "Planning and Monitoring Web Service Composition". In The 11th International Conference on Artificial Intelligence, Methodologies, Systems, and Applications (AIMSA), pages 106–115, 2004.

[65] Kutluhan Erol, James Hendler, Dana S. Nau. "Semantics for Hierarchical Task-Network Planning". Technical report CS-TR-3239, UMIACS-TR-94-31, Computer Science Dept., University of Maryland, March 1994.

[66] P Bertoli, A. Cimatti, M. Pistore, M. Roveri, P. Traverso. "MBP: a Model Based Planner". In Proceeding of ICAI-2001 workshop on Planning under Uncertainty and Incomplete Information, Seattle, WA, 2001, pp. 93–97.

[67] Dan Wu, Evren Sirin, James Hendler, Dana Nau, Bijan Parsia. "Automatic DAML-S Web Services Composition Using SHOP2". In Proceedings of 2nd International Semantic Web Conference (ISWC2003), Sanibel Island, Florida, (2003).

[68] Alberto Finzi, Fiora Pirri, Ray Reiter. "Open World Planning in the Situation Calculus".

[69] Jingao Rao. "Semantic Web Service Composition Via Logic-Based Program Synthesis". Ph.D Thesis. Department or computer and information science. Norweigan University of science and technology. 2003.

[70] M. Matskin, O. J. Kirkeluten, S. B. Krossnes. "Infrastructure for Agents, Muddlti-Agents, and Scalable Multi-Agent Systems. Support in Multi-Agent Systems", pages 28-40. Springer Verlag, 2001.

[71] U. Kuter, E. Sirin. "Information Gathering During Planning for Web Services Composition". American Association for Artificial Inteligence. 2004.

[72] "**Cougaar** (*Cognitive Agent Architecture*)". http://cougaar.org/

[73] "Semantic Web". http://www.w3.org/standards/semanticWeb/

[74] "RIF Rule Integration Format". http://www.w3.org/standards/techs/rif#w3c_all

[75] "Web Services Architecture". http://www.w3.org/TR/ws-arch/

[76] Fielding, Roy Thomas. "Architectural Styles and the Design of Network-based Software Architectures". Doctoral dissertation, University of California, Irvine, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[77] Sheila McHraith, Tran Cao Son, Honglei Zeng. "Semantic Web Services".

[78] "Web Services Choreography Description Language Version 1.0." http://www.w3.org/TR/ws-cdl-10/

[79] "OASIS Web Services Business Process Execution Language (WSBPEL)". http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[80] "WS-Choreography Definition Language (WS-CDL)". http://www.ebpml.org/ws_-_cdl.htm

[81] Ruoyan Zhang, I. Budak Arpinar, Boanerges Aleman-Meza. "Automatic Composition of Semantic Web Services". In ICWS 38-41. 2003.

[82] Amit Sheth, Marek Rusinkiewicz. "On Transactional Workflows". IEEE Data Engineering Bulletin.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.3180&rep=rep1&type=url&i=2

[83] Margie Virdell. "Business processes and workflow in the Web services world". IBM Developer Relations.
http://www.ibm.com/developerworks/Webservices/library/ws-work.html

[84] X. Dong, A. Halevy, J. Madhavan, E. Nemes, J. Zhang, Similarity. "Search for Web Services", Proceedings of the 30th International Conference on Very Large Data Bases (VLDB 2004), Morgan Kaufmann, 2004,

[85] Junhao Wen, Zhuo Jiang, Liyun Tu, Pan He. "A Task-Oriented Web Service Discovery Algorithm Using Semantic Similarity for Adaptive Service Composition".

[86] Richard Hull, Michael Benedikt, Vassilis Christophides, Jianwen Su. "E-Services: A Look Behind the Curtain". In Proceedings of PODS'03.

[87] Fabio Casati, Ming-Chien Shan. "Dynamic and adaptive composition of e-services". In The 12th international conference on advanced information systems engineering (CAiSE 00) Pages: 143 – 163. 2001.

[88] G. Shegalov, M. Gillmann, G. Weikum, "XML-enabled Workflow Management for e-Services across Heterogeneous Platforms". The VLDB Journal - The International Journal on Very Large Data Bases. Volume 10 , Issue 1(August 2001). Pages: 91 - 103.

[89] Jeanine Weissenfels, Michael Gillmann, Olivier Roth, German Shegalov, Wolfgang Wonner. "The Mentor-lite Prototype: A Light-Weight Workflow Management System". 16th Int'l Conference on Data Engineering, pp. 685-686, 2000.

[90] Massimo Mecella, Francesco Parisi Presicce, Barbara Pernici. "Modeling E -service Orchestration through Petri Nets". In Proceedings of the 3rd VLDB International Workshop on Technologies for e-Services (VLDB-TES 2002), Hong Kong, Hong Kong SAR.

[91] Marie-Christine Fauvet, Marlon Dumas , Boualem Benatallah , and Hye-Young Paik. "Peer-to-Peer Traced Execution of Composite Services". Lecture Notes In Computer Science; Vol. 2193 Proceedings of the Second International Workshop on Technologies for E-Services. Pages: 103 – 117. 2001.

[92] KAARTHIK SIVASHANMUGAM. "Framework for Semantic Web Process Composition".

[93] Santhosh Kumaran, Prabir Nandi. "Conversation Support for Web Services". http://www-106.ibm.com/developerworks/Webservices/library/ws-conver/

[94] M Gillmann, W Wonner, G Weikum. "Workflow Management with Service Quality Guarantees". International Conference on Management of Data. Proceedings of the 2002 ACM SIGMOD international conference on Management of data. Madison, Wisconsin SESSION: Research sessions: potpourri. Pages: 228 – 239. 2002.

[95] Jian Yang and Mike. P. Papazoglou. "Service Components for Managing the Life-cycle of Service Composition".

[96] Snehal Thakkar, José Luis Ambite and Craig A. Knoblock. "A Data Integration Approach to Automatically Composing and Optimizing Web Services".

[97] S. McIlraith and T.C. Son, "Adapting Golog for Composition of Semantic Web Services", Proceedings of the 8th International Conferences on Principles of Knowledge Representation and Reasoning (KR 2002), Morgan Kaufmann, 2002, pp. 482 – 493.

[98] T. Bultan, X. Fu, R. Hull, and J. Su, "Conversation Specification: A New Approach to Design and Analysis of E-Service Composition", Proceedings of the 12th International World Wide Web Conference (WWW 2003), ACM, 2003, pp. 403–410.

[99] S. McIlraith and R. Fadel. "Planning with complex actions". In Proceedings of the Ninth International Workshop on Non-Monotonic Reasoning (NMR'02), pages 356–364, Toulouse, France, April 19-21 2002.

[100] Ali Arsanjani, Ph.D.. "Service-oriented modeling and architecture *How to identify, specify, and realize services for your SO"*. *http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/*

[101] Web services .Bouna Sall, Chris Dacombe, Zhijian Pan . "Developing next-generation converged applications with SIP and asynchronous". http://www.ibm.com/developerworks/websphere/library/techarticles/1003_sall/1003_sall.html

[102] Case Study: Web 2.0 SOA Scenario. http://www.redbooks.ibm.com/abstracts/redp4555.html?Open

[103] F. Cheikh, G. De Giacomo, and M. Mecella. Automatic web services composition in trustaware communities. In Proc. of SWS 2006.

[104] D. Brand and P. Zafiropulo. "On communicating finite-state machines". Journal of the ACM, 30(2):323–342, 1983.

[105] Lucas Bordeaux, Gwen Sala¨n, Daniela Berardi, and Massimo Mecella. "When are Two Web Services Compatible?". In M.-C. Shan, U. Dayal, and M. Hsu, editors, Proceedings of the 5th International Workshop on Technologies for E-Services (TES'04), Toronto, Canada, volume 3324 of Lecture Notes in Computer Science, pages 15–28. SpringerVerlag, Berlin, 2004.

[106] Richard Hull, Jianwen Su. "Tools for Composite Web Services: A Short Overview". SIGMOD Record, 34(2):86–95, 2005.

[107] Vassilis Christophides, Richard Hull, Gregory Karvounarakis, Akhil Kumar, Geliang Tong, Ming Xiong. "Beyond Discrete E-services: Composing Session-oriented Services in Telecommunications". Proceedings of the 2nd VLDB International Workshop on Technologies for e-Services (VLDB-TES 2001), LNCS, vol. 2193, Springer, 2001, pp. 58 – 73.

[108] Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Muehl. "QoS Aggregation for Web Service Composition using Workflow Patterns".

[109] W.M.P. van der Aalst, K.M. van Hee, and G.J.Houben. "Modelling workflow management systems with high-level petri nets". Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms, pages 31–50, 1994.

[110] S Ceri, P Grefen, G Sánchez. "WIDE-A Distributed Architecture *for* Workflow Management".

[111] Event-based distributed workflow execution with EVE. A Geppert, D Tombros - Proceedings of Middleware, 1998 – Citeseer

[112] Rule-based dynamic modification of workflows in a medical domain. R Muller, E Rahm - Proc. Datenbanksysteme in Bro, …, 1999 - db15.infomatik.uni-leipzig.de

[113] J Klingemann, J Wasch, "Adaptive Outsourcing in Cross-Organizational Workflows". K Aberer - 2001 - por*tal.acm.org*

[114] KM Chandy, A Rifkin. "Systematic Composition of Objects in Distributed Internet Applications: Processes".

[115] D. Georgakopoulos, H. Schuster, D. Baker, and A. Cichocki. "Managing Escalation of Collaboration Processes in Crisis Mitigation Situations". Proceedings of ICDE2000, San Diego, CA, USA, 2000.

[116] D. L. Martin, A. J. Cheyer, and D. B. Moran. "The open agent architecture: A framework for building distributed software systems.". Applied Artificial Intelligence, January-March 1999.

[117] S.K. Shrivastava , L. Bellissard , D. Féliot , M. Herrmann , N. De Palma , S.M. Wheater1. "A Workflow and Agent based Platform for Service Provisioning".

[118] Donald Judge, Brian Odgers, John Shepherdson, Zhan Cui. "Agent Enhanced Workflow".

[119] Capítulo 5.3 del LIBRO *Web Inteligence,* Ning Zhong, Jiming Liu, Yiyu Yao.

http://books.google.com.ar/books?hl=es&lr=&id=9ovNrElSkKEC&oi=fnd&pg=PA37&dq=+Agent-Based+Composition+Services+in+DAML-S&ots=0aVfnicZxJ&sig=HD8Gsmmo4D6j8Dhm0-E7TzwqCAA#v=onepage&q=&f=false

[120] U Küster, M Stern, B König-Ries. "A classification of issues and approaches in automatic service composition". Intl. Workshop WESC, 2005.

[121] AI Wang, R Conradi, C Liu. "A multi-agent architecture for cooperative software engineering". In The Eleventh International Conference on Software Engineering and Knowledge Engineering (SEKE'99), pp. 1-22, Kaiserslautern, Germany.

[122] FIPA-OS. http://fipa-os.sourceforge.net/information.htm

[123] Thomas Magedanz, Christoph Bäumer, Markus Breugst, Sang Choy. "Grasshopper - A Universal Agent Platform Based on OMG MASIF and FIPA Standards". IKV ++ Technologies. 2000

[124] "Herramienta de descubrimiento de servicios DySCO". http://msdn.microsoft.com/es-es/library/cy2a3ybs(VS.80).aspx

[125] Jini. http://www.jini.org/wiki/Main_Page

[126] Endrei M. "Patterns: Service-Oriented Architecture and Web Services". Redbook, SG24-6303- 00, April 2004. http://publib.boulder.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg246303.html?

Open

[127] Julian Robichaux, "Practical Web Services in IBM Lotus Domino 7: What are Web services and why are they important?". http://www.ibm.com/developerworks/lotus/library/web-services1/

[128] BEA, Intalio, SAP, and Sun, "Web Service Choreography Interface (WSCI) 1.0", W3C Document. http://www.w3.org/TR/wsci/, 2002.

[129] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, Dana Nau. "HTN Planning for Web Service Composition Using SHOP2". Web Semantics, Elsevier, Volume 1, Issue 4, October 2004, Pages 377-396.

[130] K. Sycara, J.A. Giampapa, B.K. Langley, and M. Paolucci, "The RETSINA MAS, a Case Study," Software Engineering for Large-Scale Multi-Agent Systems: Research Issues and Practical Applications, Alessandro Garcia, Carlos Lucena, Franco Zambonelli, Andrea Omici, Jaelson Castro, ed., Springer-Verlag, Berlin Heidelberg, Vol. LNCS 2603, July, 2003, pp. 232--250.

[131] G. De Giacomo and S. Sardina. "Automatic synthesis of new behaviors from a library of available behaviors". In Proc. of IJCAI 2007.

[132] S. Sardina, F. Patrizi, and G. De Giacomo. "Behavior composition in the presence of failure". In Proc. of KR 2008.

[133] S. Sardina, F. Patrizi, and G. De Giacomo. "Automaticsynthesis of a global behavior from multiple distributed behaviors". In Proc. of AAAI 2007.

[134] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. "Synthesis of underspecified composite e-Services based on automated reasoning". In Proc. of ICSOC 2004.

[135] Sanjiva Weerawarana, Francisco Curbera. "Business Process with BPEL4WS: Understanding BPEL4WS, Part 1". http://www.ibm.com/developerworks/webservices/library/ws-bpelcol1/index.html

[136] Ajamu Wesley. "WSFL in action, Part 1". http://www.ibm.com/developerworks/webservices/library/ws-wsfl1/

[137] "Business Process Modeling Language". http://xml.coverpages.org/BPML-2002.pdf

[138] "Business Process Specification Schema". http://www.service-architecture.com/web-

services/articles/business_process_specification_schema_bpss.html

[139] Web Service Cordination Framework (WS-CF). http://www.oasis-open.org%2Fcommittees%2Fdownload.php%2F10889%2FWSCF-Working-12-22.pdf&rct=j&q=Web+Services+Coordination+Framework+(WS-+CF)&ei=v1jES4PdEYuQuAe8zdWbDw&usg=AFQjCNEi-T4gOoIEhm60y6ZfdrkCEottFw

[140] Mark M. Davydov. "Managing State in Service-Oriented Architecture". http://www.oracle.com/technology/pub/articles/davydov_soa.html

[141] Web Service Transaction specifications, IBM. http://www.ibm.com/developerworks/library/specification/ws-tx/

[142] Freund, Tom & Tony Story: Transactions in the world of Web Services, Part 1. An overview of WS-Transaction and WS-Coordination. In: http://www-106.ibm.com/developerworks/webservices/library/ws-wstx1/

[143] Freund, Tom & Tony Story: Transactions in the world of Web Services, Part 2. An overview of WS-Transaction and WS-Coordination. In: http://www-106.ibm.com/developerworks/webservices/library/ws-wstx2/

[144] Web Services Context Specification (WS-Context) Version 1.0. OASIS Standard. http://docs.oasis-open.org/ws-caf/ws-context/v1.0/wsctx.html

[145] Sheng, Quan Z., Boualem Benatallah, Marlon Dumas & Eileen Oi-Yan Mak: "SELF-SERV – A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment". Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

[146] Schahram Dustdar, and Wolfgang Schreiner. "A survey on web services composition". International Journal of Web and Grid Services, v.1 n.1, p.1-30, August 2005 .

[147] J.G. Plaza, J. Guzmán Luna, A. Ledesma Castilloa. "Sistema multiagente para la composición de servicios web semánticos". Universidad Nacional de Colombia, Medellín.

[148] Keidl, Markus & Alfons Kemper: A Framework for Context-Aware Adaptable Web-Services. E. Bertino et al. (Eds.): EDBT 2004, LNCS 2992, pp. 826-829, 2004. Springer-Verlag Berlin Heidelberg 2004

[149] D oulkeridis, Christos, Efstratios Valavanis & Michalis Vazirgiannis: Towards A Context-Aware Service Directory. B. Benatallah and M.-C. Shan

(Eds.): TES 2003, LNCS 2819, pp. 54-65, 2003. Springer-Verlag Berlin Heidelberg 2003.

[150] John Harney, Prashant Doshi. "Selective Querying for Adapting Hierarchical Web Service Compositions Using Aggregate Volatility". IEEE International Conference on Web Services (IWCS), pages 43–50, 2009.

[151] John Harney, Prashant Doshi. "Speeding up Adaptation of Web service Compositions using Expiration Times". In WWW, pages 1023–1032, 2007.

[152] Shirin Sohrabi, Nataliya Prokoshyna, and Sheila A. McIlraith. "Web Service Composition via the Customization of Golog Programs with User Preferences". Conceptual Modeling:Foundations and Applications, pages 319–332, 2009.

[153] Sharon Paradesi, Prashant Doshi and Sonu Swaika, "Toward Integrating Social Trust in Web Service Compositions", short paper, AAAI Spring Symposium on Social Semantic Web, 2009

[154] L. Baresi, D. Bianchini, V. De Antonellis, M.G. Fugini, B. Pernici, P. Plebani. "Context-aware Composition of E-Services". In Technologies for E-Services, 4th International Workshop, TES 2003, Berlin, Germany, September 8, 2003, Proceedings, edited by B. Benatallah and M. C. Shan: Springer.