PROYECTO DE GRADO

COMPOSICIÓN DINÁMICA DE SERVICIOS WEB SEMÁNTICOS BASADA EN PLANIFICACIÓN

Andrés González, Germán Pereyra, Pablo Silva
Universidad de la República – Montevideo, Uruguay

Tutores: Regina Motz, Alberto Pardo

RESUMEN

En este trabajo se estudia la composición automática de servicios web semánticos mediante la transformación del problema a uno de planificación. Dado un problema de composición el mismo se aborda en dos fases. La primera fase consiste en generar un problema de planificación a partir de la información semántica de la composición que se desea generar. Para ello se trabaja con el estándar OWL-S de especificación de servicios web semánticos. La segunda fase se focaliza en la resolución de dicho problema de planificación.

Para esto se realizó el relevamiento y estudio de diferentes planificadores con el objetivo de seleccionar los más adecuados para la composición de servicios web.

A partir del análisis de tecnologías vinculadas a la planificación automática, se construyó un prototipo que permite resolver problemas de composición de servicios web haciendo uso de la información semántica disponible y utilizando distintos planificadores. Dicho prototipo cuenta con una interfaz que brinda al usuario la posibilidad de visualizar los resultados obtenidos de forma gráfica.

CONTENIDO

1. INTRODUCCIÓN

En la actualidad, los servicios web juegan un rol sumamente importante en la Web ya que suplen la creciente necesidad de interoperabilidad, integración y colaboración entre sistemas y aplicaciones heterogéneas que exponen sus funcionalidades a través de una red. Las tecnologías de servicios web proporcionan una manera de comunicarse e interactuar con sistemas de información a través de una interfaz estándar, independiente de la implementación interna. En muchos casos, a medida que los requisitos de los usuarios demandan funcionalidades más complejas, los mismos no pueden ser satisfechos por un solo servicio web atómico. Este problema puede ser resuelto mediante la composición de servicios web, o sea, la combinación de forma apropiada de ciertos servicios web atómicos con el fin de lograr un objetivo complejo.

Sin embargo, a medida que el número de servicios web atómicos disponibles aumenta, la tarea de composición de manera manual se torna significativamente dificultosa, consume demasiado tiempo y es propensa a errores, resultando ineficiente e ineficaz. Por lo tanto, la capacidad para automatizar el proceso de composición de servicios web resulta esencial.

La automatización del descubrimiento, invocación, composición e interoperación de servicios web se ve significativamente simplificada por la existencia de información semántica en la descripción de los servicios web atómicos. Dicha información representa el conocimiento que describe los aspectos semánticos y pragmáticos de los servicios. Estas descripciones adicionales permiten el uso de computadoras para razonar acerca de los servicios publicados. La incorporación de semántica en la descripción de servicios web es facilitada a través de un número de lenguajes estándar como OWL-S [4] y SAWSDL [2], dando lugar al concepto de servicios web semánticos, los cuales son definidos, evolucionan y operan en la Web Semántica [3]. Sin la presencia de la información semántica, se requeriría un alto grado de participación humana para componer servicios web de manera significativa y no basada exclusivamente en similitudes sintácticas circunstanciales. La existencia de información semántica facilita considerablemente la composición utilizando técnicas inteligentes, tales como la planificación.

Con el objetivo de apreciar el problema desde un punto de vista más concreto, consideremos el siguiente simple ejemplo práctico: se tiene un sitio dentro del contexto del comercio electrónico. Este sitio opera como cualquier otro sitio de esta índole (por ejemplo, Amazon.com). Ofrece distintos productos a la venta, y permite a los usuarios del mismo la búsqueda y compra de los productos que deseen mediante tarjeta de crédito o pago electrónico, siendo luego las compras realizadas entregadas a domicilio.

Para esto el sitio cuenta con los siguientes servicios web semánticos involucrados al realizar una compra:

- *searchProductService*: Este servicio recibe como entrada la descripción de un producto (atributo que es único) y retorna toda la información del producto con esa descripción.

- *authorizeService*: Este servicio recibe como entradas un producto y un número de tarjeta de crédito, autorizando con la empresa de crédito la compra del producto y retornando la autorización necesaria para realizar la compra.

- *payService*: Este servicio recibe como entrada una autorización de compra y un producto, realizando el pago del producto con la tarjeta correspondiente a la autorización y retornando como salida una orden de compra.

- *deliveryService*: Este servicio recibe como parámetro de entrada la dirección del domicilio de un usuario, retornando la información de entrega correspondiente.

- *finalizeBuyService*: Este servicio recibe como entradas una orden y la información de entrega y retorna una factura de la compra realizada, finalizando de esta manera el proceso de compra.

Para realizar el proceso de una compra en su totalidad en el contexto especificado, el usuario debe seleccionar el producto e ingresar su número de tarjeta de crédito y dirección, y el sistema que existe por detrás del sitio debe ejecutar uno por uno cada uno de los servicios web indicados anteriormente en el orden planteado. De esta manera, en este caso la composición de servicios a realizar sería la siguiente:

((*searchProduct . authorize . payService*) | *delivery*) . *finalizeBuy*

Primero, se ejecutan los servicios correspondientes a la búsqueda del producto, la autorización y pago de la compra, y la generación de la información de entrega (notar que los servicios involucrados en la compra y el servicio correspondiente a la entrega pueden llegar a ser ejecutados en paralelo). Luego, la información generada al ejecutar los servicios anteriores es pasada al servicio *finalizeBuy* para de esta manera completar el proceso de compra.

La lógica para la ejecución de esta composición de servicios debe estar programada en el sistema del sitio. En caso de ser modificados los servicios web o en caso de que cambie el proceso de compra, estos programas también deben ser modificados. Si se contara con una manera de realizar la composición de servicios web de manera automática, a partir de un conjunto de datos iniciales y una meta compleja a alcanzar, no sería necesaria la modificación de forma manual de los programas en caso de cambios. Con tal herramienta, para el caso del ejemplo, al querer realizar una compra (meta), el usuario ingresa los datos correspondientes (datos iniciales) y el sistema, solo a partir de esta información, determinaría que combinación de los servicios web disponibles debe ejecutar y en qué orden debe hacerlo para realizar el proceso de compra en su totalidad.

Distintos abordajes existen para la resolución del problema planteado. Uno de ellos se basa en la aplicación de técnicas de planificación automática para su resolución. Las herramientas para sostener este enfoque, y el aporte que puede brindar el aprovechamiento de la información semántica disponible a la hora de resolver el problema de planificación es lo que se estudia y aborda en este trabajo.

Básicamente, la planificación automática involucra la selección de ciertas acciones adecuadas y su ordenamiento en una secuencia apropiada para el alcance de cierta meta. El problema de composición de servicios web puede ser visto bajo esta perspectiva. Como primer paso, esto implica la traducción del problema de composición de servicios web a términos de un problema de planificación. Esta traducción se basa en la observación de la existencia de

ciertas correspondencias entre ambos dominios, que dadas las correspondencias adecuadas, permiten del problema de forma efectiva. Tales correspondencias incluyen los servicios web disponibles que pueden ser combinados para formular combinaciones significativas, los cuales pueden ser mapeados al dominio de planificación, y los requerimientos del usuario sobre el servicio compuesto deseado, los cuales pueden ser vistos como un problema de planificación bajo este dominio.

Bajo este enfoque, se desarrolla una herramienta que implementa la composición, realizando las traducciones y mapeos necesarios y que permite la conexión de distintos planificadores que implementan distintos algoritmos de planificación para la resolución del problema, presentando las soluciones obtenidas mediante una interfaz gráfica.

El resto de este documento está organizado de la siguiente manera: en el capítulo 2 presentan los conceptos preliminares necesarios para la comprensión del trabajo, en el capítulo 3 se discuten trabajos relacionados con la composición basada en planificación, en el capítulo 4 se describen y comparan las distintas herramientas de planificación estudiadas, en el capítulo 5 se detalla la implementación realizada, en el capítulo 6 se presentan algunos casos de estudio y por último, en el capítulo 7 se exponen las conclusiones obtenidas y posibles trabajos futuros.

2. CONCEPTOS PRELIMINARES

2.1. SERVICIOS WEB

Los servicios web permiten que los sistemas de información puedan interoperar mediante el intercambio de mensajes en la web. En este escenario existen proveedores que ofrecen diferentes servicios que adoptan la forma de métodos que pueden ser invocados por sus consumidores en la web.

Los servicios web facilitan la interoperabilidad entre aplicaciones de dos formas principales:

- Usando un formato de mensaje para el intercambio de información que es independiente de los lenguajes y plataformas de cada uno de los sistemas de información que intercambian estos mensajes.

- Empleando la web como medio de comunicación, la cual tiene un alcance, tanto en extensión geográfica, como en número de usuarios, superior a cualquier otro medio.

Los servicios web enriquecen la propia web, ya que permiten que ésta pueda ofrecer información dinámica.

Una vez que se dispone del medio de comunicación, la web, y un mecanismo como los servicios web, para que los sistemas de información puedan intercambiar información, estos servicios se pueden combinar para realizar tareas más complejas, lo cual requiere a su vez el desarrollo de una arquitectura estándar que facilite y dé soporte a esta complejidad.

En la actualidad existen estándares con un uso bastante extendido para el uso de los servicios web, en el resto de este trabajo se presenta una introducción a ellos, y se analiza su potencial para proporcionar el mayor grado de interoperabilidad posible, no sólo desde el punto de vista tecnológico, sino también desde el punto de vista semántico, para permitir una automatización sobre la web en la resolución de problemas.

2.2. WSDL (WEB SERVICE DEFINITION LANGUAGE)

Los servicios web son mayormente definidos y descritos mediante un lenguaje basado en XML llamado Web Service Definition Language (WSDL). Es el estándar aceptado y recomendado por la W3C Consortium [4].

WSDL describe un servicio web especificando los métodos que éste ofrece. Estos métodos se describen de forma abstracta. Para el uso de los servicios hay que emplear algún protocolo de comunicaciones, el cual no está impuesto por WSDL. El uso de los servicios se materializa mediante el intercambio de mensajes que contienen información de tipo procedimental. El formato de estos mensajes tampoco viene determinado por WSDL. Esta separación entre la especificación de los servicios y la implementación de éstos es uno de los aspectos más relevantes de WSDL. En la Figura 1 se detallan las secciones de un documento WDSL.

Sección	Descripción
types	Permite definir tipos empleando algún sistema de tipos, como XML Schema (XSD).
message	Una descripción de los datos intercambiados cuando se hace uso de los servicios.
portType	Un conjunto de operaciones abstractas ofrecidas por uno o más puntos finales (*endpoint*).
binding	Un protocolo de comunicaciones y un formato de mensajes concretos utilizados por un tipo de puerto.
service	Funcionalidad ofrecida por el servicio

Figura 1: Secciones de un documento WDSL

WSDL es el lenguaje más habitual para describir servicios web, su gran acierto ha sido permitir una descripción abstracta de servicios, sin ligarlos a una implementación concreta, lo que ofrece las siguientes ventajas:

- Permite el uso de cualquier implementación.
- Es posible reutilizar la definición de estos servicios con diferentes implementaciones y escenarios
- Facilita la interoperabilidad cuando existen implementaciones diferentes de los mismo servicios

A pesar de las ventajas anteriores, desde el punto de vista semántico, WSDL carece de recursos para describir aquellos aspectos de los servicios web que vayan más allá de su interfaz o la implementación concreta que se usará. Los aspectos semánticos de los servicios, que principalmente permitirían describir el propio servicio, y las condiciones que se tienen que dar para que el servicio pueda ser prestado, quedan fuera del alcance de WSDL.

El dejar fuera de su ámbito los aspectos semánticos, por un lado hace que WSDL sea más sencillo de entender y usar, lo que ha permitido su rápida adopción.

Por otro lado hay que decir que usar WSDL requiere tener que describir los propios servicios mediante otros documentos, los cuales suelen ser textuales, y sin seguir ningún estándar, lo que provoca que en muchos casos estos documentos sean incompletos y/o ambiguos.

Esta forma de describir la semántica tiene otro inconveniente, el de impedir que la localización y el uso de los servicios pueda realizarse de forma automatizada. Para poder automatizar estas tareas es necesario utilizar lenguajes que permitan describir los aspectos semánticos de los servicios web.

2.3. ONTOLOGÍA

Las ontologías representan de manera formal y consensuada especificaciones de conceptos, que proveen un conocimiento compartido y común del dominio el cual es procesable por máquinas e interoperable a través de agentes (organizaciones, individuos y software).

Las ontologías utilizan lenguajes formales que describen de forma precisa los conceptos de un determinado dominio y las relaciones existentes entre esos conceptos. Estos lenguajes formales ofrecen un vocabulario básico que sirve para la descripción de cualquier dominio mediante ontologías.

Una ontología se compone principalmente de los siguientes componentes:

- Conceptos: Elementos básicos de las tareas del dominio. Se suelen organizar en taxonomías.
- Instancias: Son elementos específicos de los conceptos.
- Relaciones: Expresan relaciones entre los conceptos del dominio.
- Funciones: Métodos que pueden ser invocados en la instancia específica de un concepto.
- Axiomas: Sentencias del modelo que son siempre ciertas. Hechos.

Las ontologías recomendadas por el consorcio W3C para describir servicios web se basan en Lógica Descriptiva y proveen así una herramienta formal para representar y razonar con los servicios web semánticos.

2.4. SERVICIOS WEB SEMÁNTICOS

Los servicios web semánticos son servicios web que cuentan con meta-información que permite facilitar su búsqueda y composición de forma automática.

La incorporación de semántica a estos servicios facilita la automatización del proceso de descubrimiento, composición, invocación, interoperabilidad y ejecución de los mismos. Algunos de los mecanismos más importantes para la incorporación de semántica a la descripción de los servicios, se fundamentan en el uso de ontologías, tales como OWL-S o WSMO.

Algunas de las principales características en el uso de los Servicios Web Semánticos son las siguientes:

- Publicación de servicios en un registro semántico.
- Descubrimiento automático en función de la petición y la descripción semántica publicada en el servicio.
- Selección de servicios en caso de conflicto entre varios que satisfagan la petición.
- Composición automática de servicios, de modo que el resultado de varios servicios formen la solución a la petición del usuario.
- Invocación automática del servicio, validando los parámetros del servicio.

2.5. OWL-S

Como el contenido de la web se ha multiplicado y el número de servicios aumenta día a día, el descubrimiento, la integración y la ejecución de los recursos se están convirtiendo en una tarea excesivamente abrumadora si se hace manualmente. Si este proceso es automatizado, las aplicaciones pueden extraer, analizar y comprender el contenido de forma inequívoca, lo cual no es posible a menos que establezca una norma homogénea utilizada en toda la web. Es por esto que se iniciaron proyectos para el desarrollo de lenguajes que ayuden a agentes autónomos a entender la web no solo sintácticamente sino también semánticamente. Uno de estos lenguajes es OWL-S.

OWL-S, que se ha desarrollado específicamente para los servicios alojados en la web, es una ontología para describir las funcionalidades de los servicios. Los objetivos principales de este lenguaje son:

- Automatizar el descubrimiento de servicios
- Automatizar la invocación de servicio
- Automatizar la composición de servicios y por lo tanto la interoperabilidad.

2.5.1. ESTRUCTURA DE OWL-S

OWL-S ofrece un modelo para las definiciones de servicio. Cada servicio que se define en OWL-S debe declarar *ServiceProfile*, *ServiceGrounding* y *ServiceModel*. Estas tres vistas diferentes miran desde un aspecto distinto el mismo servicio. *ServiceProfile* describe el servicio semánticamente para el descubrimiento. *ServiceGrounding* explica la interfaz del servicio para la interoperación y *ServiceModel* especifica la estructura interna del servicio para la composición. Este modelo combinado se muestra en la Figura 2.

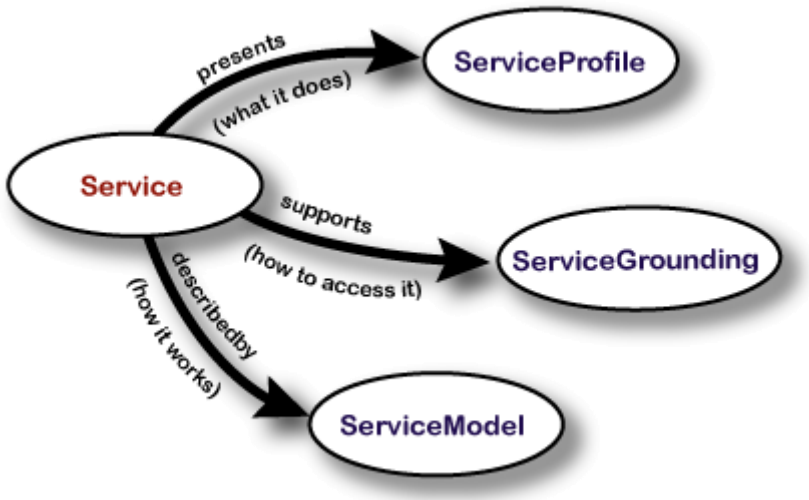

Figura 2: estructura de owl-s [1]

SERVICE PROFILE

ServiceProfile contiene toda la información necesaria para el descubrimiento del servicio. La función del servicio se detalla en este elemento de modelado y ayuda a los solicitantes de servicios a localizar el servicio. Los atributos del perfil de servicio se muestran en la Figura 3.

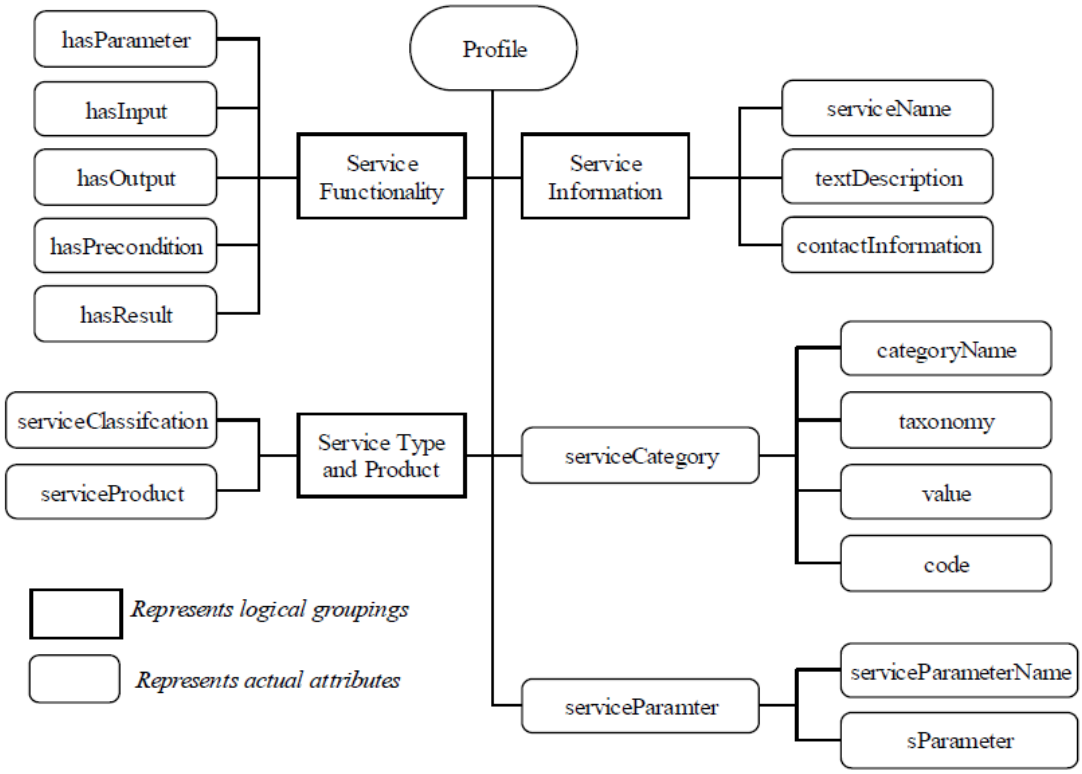

Figura 3: estructura de ServiceProfile [9]

Los perfiles tienen como fin especificar los propósitos de los servicios. Los atributos de perfil podrían ser agrupados en categorías de acuerdo con sus propósitos.

En la categoría *Service Information*, se lista información general sobre el servicio. Se muestra el nombre del servicio, un texto con la descripción y la persona de contacto del proveedor de servicio. Los datos que se muestran están en formato libre, por lo que su interpretación debe ser realizada por personas.

Service Functionality describe la interfaz del servicio con el fin de dar información sobre sus entradas, salidas, efectos y precondiciones. Todos los atributos listados aquí, si son propiedad del servicio, podrían estar vinculados con las instancias de *Service Model Process* para obtener una descripción detallada, pero no hay ninguna restricción dictada por OWL-S. *Service Functionality* ofrece una sintaxis de alto nivel para describir cómo el servicio web va a interactuar con el mundo exterior.

ServiceParameter es una lista de propiedades que podría facilitarse como información adicional para el servicio. Dentro de la lista, hay tuplas de nombres y referencias de ontologías OWL. El atributo *serviceCategory* se utiliza para dar una referencia a la información categorizada o taxonomías que se utilizan junto con el dominio de servicio, pero fuera del ámbito de OWL-S. Este atributo proporciona un punto de extensión de tales enlaces externos o interoperabilidad con las categorizaciones ya preparadas. Finalmente, agrupados como *Service Type and Product* hay dos atributos más del *Service Profile*. El primer atributo *serviceClassification* guarda una referencia a la ontología OWL del negocio de dominio, y *serviceProduct* guarda una referencia a la ontología OWL del producto.

SERVICE MODEL

Service Model describe la estructura interna de los servicios. Cada servicio tiene un proceso para representar su composición interna. Un proceso especifica cómo los clientes utilizan el servicio. En él se describen las entradas, salidas, resultados, condiciones previas y los efectos del servicio. Las precondiciones y post condiciones del servicio junto con las entradas previstas y los posibles resultados se describen en detalle a la solicitante del servicio por el proveedor de servicios. La estructura del proceso se presenta en la Figura 4.

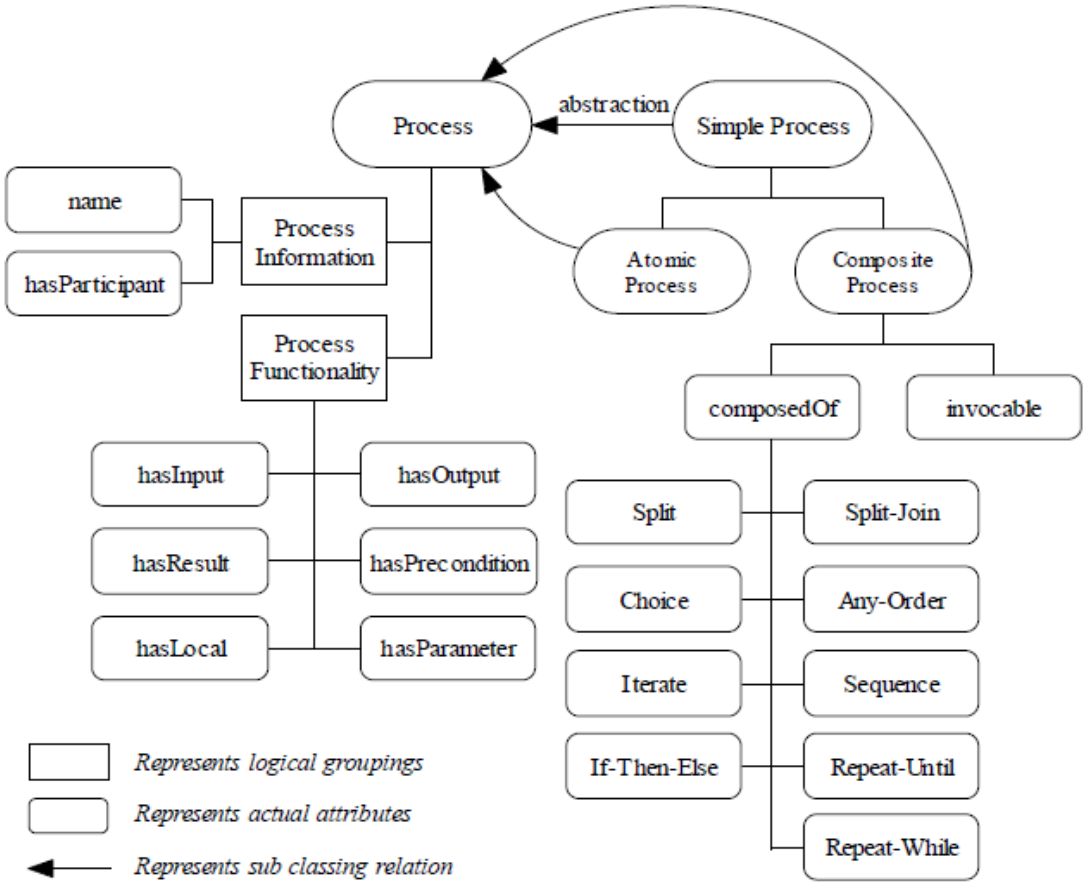

Figura 4: estructura de ServiceModel [9]

En el grupo de atributos *Process Information* se especifican el nombre y los participantes en el proceso. Normalmente los participantes por defecto son los clientes y el servidor que aloja el servicio. Si existen otros participantes se enumeran en este parámetro. El grupo *Process Functionality*, al igual que el grupo correspondiente en *Profile*, detalla la función del servicio. En este caso se da la interfaz exacta del servicio. Los *inputs* son los datos que se le dan al servicio para procesar. Los *outputs* son información recuperada por el servicio como un importante subproducto de su ejecución. Las precondiciones son las expresiones lógicas de requisitos previos que tienen que ser verdad para invocar los servicios. Los resultados de los servicios se utilizan para determinar el éxito y los efectos del servicio.

Los procesos atómicos representan servicios que se ejecutan directamente de acuerdo a sus propiedades asociadas. Los procesos compuestos representan un conjunto de procesos atómicos o compuestos que son conectados uno a otro con

ciertas reglas de control. *Split, Split-Join, Sequence, Choice, If-Then-Else, Iterate, Repeat-While, Repeat-Until* y *Any-Order* se usan para definir el flujo de la composición.

Finalmente, los procesos simples son abstracciones o vistas simplificadas de los procesos atómicos y compuestos. Estos no son ejecutables y no están vinculados con el *grounding*, sino que se perciben como un proceso de un solo paso, al igual que el proceso atómico, lo que facilita las tareas de planificación y razonamiento sobre los procesos.

SERVICE GROUNDING

Service Grounding describe los detalles de acceso al servicio que son: el protocolo utilizado, el formato del mensaje, el transporte y el tratamiento de los servicios. Es un mapeo de la especificación abstracta del servicio a una definición concreta. Se utiliza WSDL para referenciar las descripciones abstractas con definiciones concretas. *Binding* en WSDL sirve exactamente como *grounding* en OWL-S.

Los procesos atómicos, que son las entidades ejecutables de OWL-S, están ligados a las operaciones WSDL para especificar concretamente la interfaz del servicio web abstracto. La Figura 5 resume las correspondencias entre los dos lenguajes.

OWL-S	WSDL
Atomic Process	Operation
Atomic Process with input and then output	Request-response operation
Atomic Process with input only	One-way operation
Atomic Process with output only	Notification operation
Atomic Process with output and then input	Solicit-response operation
Atomic Process inputs	Operation input message
Atomic Process outputs	Operation output message
Atomic Process input and output types	Abstract types

Figura 5: correspondencias owl-s y wsdl [9]

La clase *WsdlGrounding* se utiliza para vincular los procesos a su descripción WSDL asociada dentro de OWL-S. Contiene una lista de instancias de *WsdlAtomicProcessGrounding* que conectan los procesos atómicos a las operaciones de servicio en WSDL. La estructura se visualiza en la Figura 6.

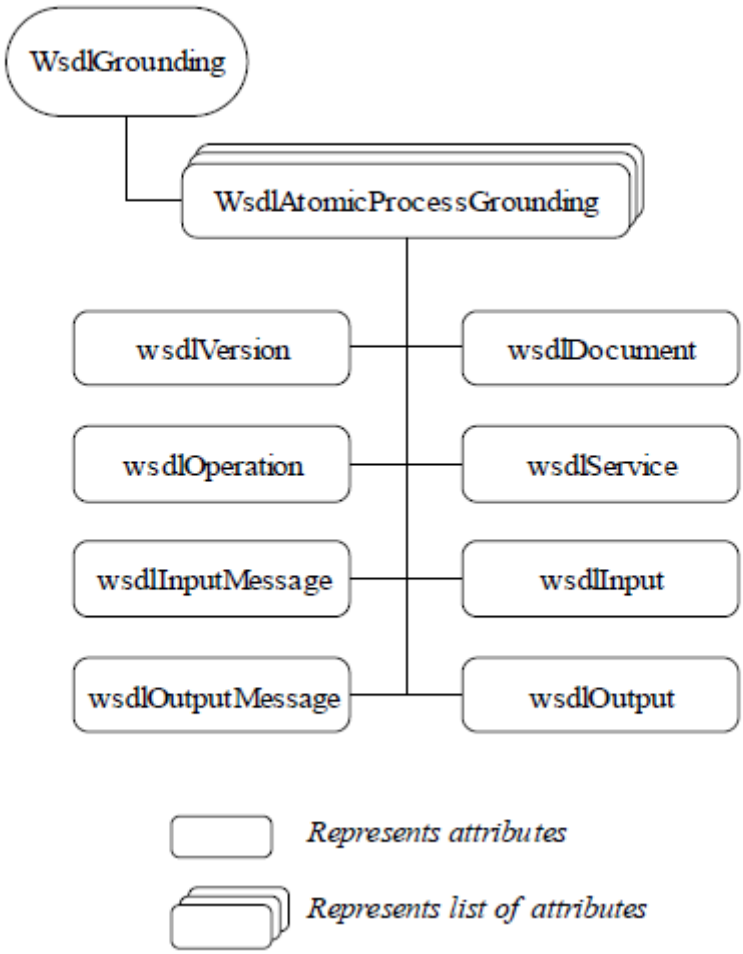

Figura 6: estructura de WsdlGrounding [9]

La instancia *WsdlAutomicProcessGrounding* contiene información de referencia para su correspondiente WSDL *grounding*. La versión en uso de WSDL y el documento WSDL son referenciados usando este atributo. La operación WSDL asociada se da en el atributo *wsdlOperation*. Finalmente el resto de los atributos hacen referencia a las entradas y salidas de la operación WSDL, y a los mensajes de entrada y salida de la operación.

2.6. PLANIFICACIÓN AUTOMÁTICA

Los métodos de planificación automática se centran en la selección de un conjunto de acciones adecuadas y su ordenamiento en una secuencia apropiada para cumplir con cierto objetivo. En general, un problema y domino de planificación clásico se puede modelar de acuerdo a la notación STRIPS (*Stanford Research Institute Planning System*) [5] como una tupla $\langle S, S_0, G, A, \Gamma \rangle$ donde:

- S es el conjunto de todos los posibles estados del mundo.
- $S_0 \subset S$ denota el estado inicial del mundo.

- $G \subset S$ denota el estado objetivo del mundo que el sistema de planificación intenta alcanzar.

- A es el conjunto de acciones disponibles que el planificador puede realizar para alcanzar un estado deseado.

- La relación de traducción $\Gamma \subseteq S \times A \times S$ define las precondiciones y efectos de la ejecución de cada acción.

El conjunto A contiene todas las acciones que pueden ser utilizadas para modificar estados. Cada acción A_i tiene asociada tres listas de hechos dentro de Γ que contienen las precondiciones de A_i, los hechos que son agregados y los hechos que son borrados del estado del mundo luego de aplicada la acción, denominadas como $prec(A_i)$, $add(A_i)$ y $del(A_i)$ respectivamente.

Las siguientes condiciones se cumplen para los estados en la notación STRIPS:

- Una acción A_i es aplicable a un estado S_i si $prec(A_i) \subseteq S_i$.

- Si A_i es aplicado a S_i, el estado obtenido S' es calculado como $S' = S - del(A_i) \cup add(A_i)$.

- La solución a un problema de planificación (plan P) es una secuencia ordenada de acciones $P = A_1, A_2, \ldots, A_n$, que si son aplicadas a S_0 generan un estado S' tal que $S' \supseteq G$.

2.7. PDDL

PDDL [6] (Planning Domain Definition Language) surge como un intento de estandarizar los lenguajes de planificación de inteligencia artificial. Fue desarrollado en 1998 para hacer posible la 'International Planning Competition' (IPC) y ha evolucionado en cada competición. Este lenguaje está influenciado principalmente por STRIPS, sin embargo, se extiende a contener la expresividad de muchos otros lenguajes de planificación como ADL [7] para incorporar las proposiciones lógicas y UMCP [8] para incorporar acciones. Su sintaxis es muy similar al lenguaje de programación LISP.

2.7.1. PARTES DEL LENGUAJE

PDDL consta de dos partes, una para especificar el dominio y otra para especificar el problema. La definición de dominio contiene toda la información sobre el espacio del problema. Por ejemplo, las acciones, los axiomas y los predicados se definen en esta parte. La definición del problema contiene el estado inicial del dominio y el estado de la meta que se pretende alcanzar. Estas partes de PDDL se muestran en la figura 7, donde (a) corresponde a la definición del dominio y (b) a la definición del problema.

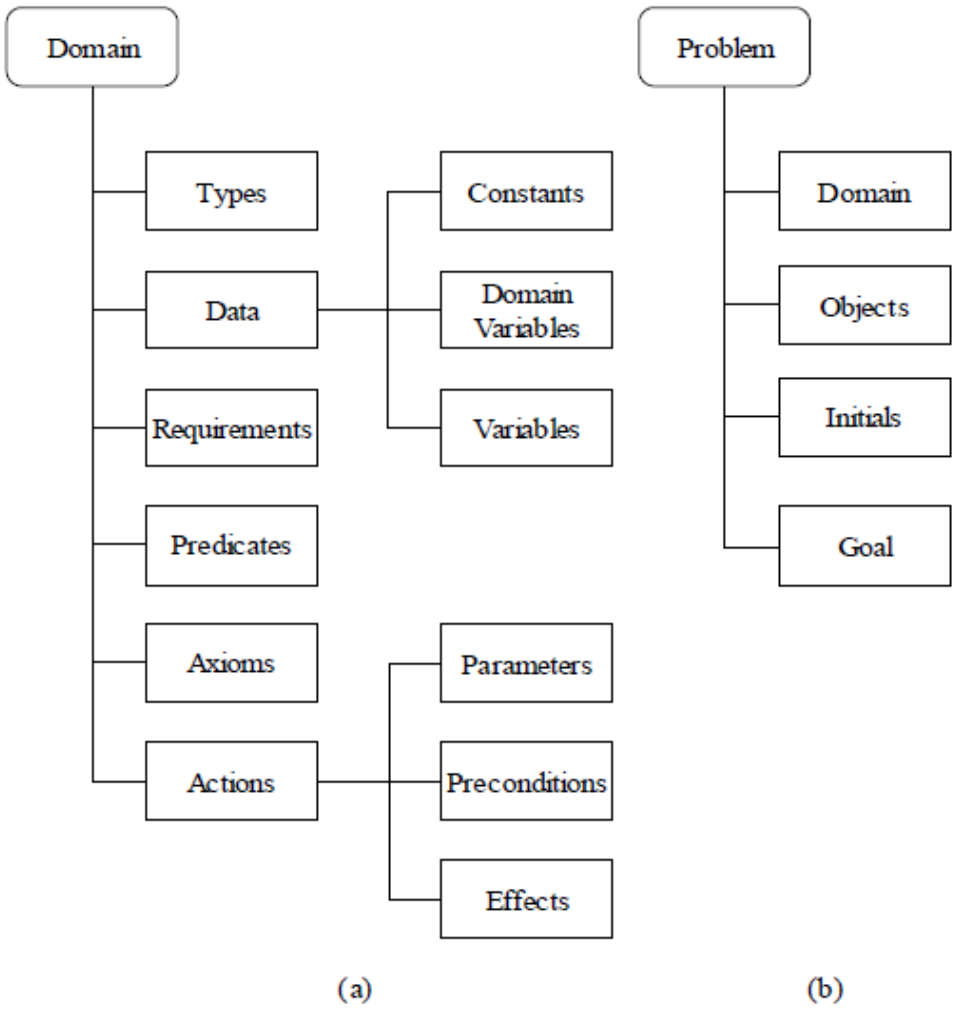

Figura 7: estructura del lenguaje pddl [9]

PDDL DOMAIN

En el dominio se describe el mundo en el cual se va a resolver el problema. Capta todos los aspectos del mundo que utilizarán los mecanismos de planificación. Durante la fase de resolución de problemas, los estados sucesivos del mundo son generados por las reglas definidas en esta parte.

Requirements

Los requerimientos se describen como un conjunto de banderas que especifican los campos que el dominio necesita cubrir con el fin de definir el espacio del problema.

Los requerimientos más comúnmente utilizados son:

:strips
El subconjunto más básico de PDDL, que consiste de STRIPS solamente.

:equality
El dominio tiene definido un predicado de igualdad.

:typing
El dominio utiliza types (ver Types a continuación).

:adl
El dominio utiliza algunas o todas las ADL [7].

Types

En PDDL todos los objetos y las variables tienen un dominio. Todos los dominios admiten tipos de objetos y números, sin embargo, estos pueden extenderse para expresar variables complejas.

PDDL soporta la herencia de objetos y tipos derivados que pueden ser usados en lugar de los tipos base como en los lenguajes tradicionales orientados a objetos. PDDL también admite específicamente el tipo *fluent*, un término que permite que un valor cambie como resultado de realizar una acción.

Data

Hay diferentes tipos de definición de datos en el lenguaje PDDL. El primero de ellos son las constantes. Definen los valores simbólicos de los tipos especificados y no cambian su valor. Son visibles en todo el alcance del dominio. Las variables de dominio, al igual que las constantes, tienen alcance a nivel de todo el dominio, pero su valor puede ser modificado. Son como las variables globales en los lenguajes de programación tradicionales. Es también posible definir variables locales dentro de las acciones, axiomas o predicados de una manera similar a los de los lenguajes de programación y su alcance es el lugar en que se definen.

Predicates

Los predicados se utilizan para atribuir propiedades a un objeto o definir una relación entre varios objetos.

Actions

Las acciones son las entidades más importantes de un dominio de planificación. Son las que desencadenan las transiciones de estado del dominio. Con el fin de alcanzar el objetivo de un problema de planificación, las acciones se utilizan para controlar la sucesión de situaciones. Si el objetivo es satisfacible entonces el planificador da como salida una secuencia de acciones que produce el resultado deseado.

Los parámetros de las acciones pueden ser utilizados como input durante la ejecución de una acción, como chequeo de precondiciones de la acción o se pueden utilizar para producir efectos relacionados con estas precondiciones. Las precondiciones de una acción constituyen un conjunto de afirmaciones lógicas que deben satisfacerse antes de ejecutar la acción. Los efectos son las consecuencias de la ejecución de una acción. Estos causan la transición al estado actual modificando el valor de las variables ("*fluents*") o agregando nuevos predicados en los objetos.

En la Figura 8 se muestra un ejemplo de PDDL Domain [9] para ilustrar las declaraciones explicadas en esta sección.

```
(define (domain carry-balls)
  (:requirements :strips :typing :adl :domain-axioms)

  (:types (robbot gripper ball - object room - location))

  (:predicates (at-robby ?r - robot)
               (at ?b - ball ?r - room)
               (free ?g - gripper)
               (carry ?o - ball ?g - gripper)
               (has ?r - robot ?g - gripper))

  (:action move
    :parameters (?fr - room ?to - room)
    :precondition (at-robby ?fr)
    :effect (and (at-robby ?to) (not (at-robby ?fr))))

  (:action pick
    :parameters (?b - ball ?rm - room
                 ?rb - robot ?g - gripper)
    :precondition (and (available ?rb) (at ?b ?rm)
                       (at-robby ?rm))
    :effect (and (carry ?b ?g) (not (at ?b ?rm))
            (not (free ?g))))

  (:action drop
    :parameters (?b - ball ?r - room ?g - gripper)
    :precondition (and (carry ?b ?g) (at-robby ?r))
    :effect (and (at ?b ?r) (free ?g)
                 (not (carry ?b ?g))))

  (:axiom
    :vars (?r - robot ?g - gripper)
    :context (exists (?g)
              (and (has r g) (free g)))
    :implies (availabe ?r)))
```

Figura 8: ejemplo de dominio PDDL [9]

En este dominio, algunas acciones se definen para que un robot lleve un número de pelotas a un destino final. Las acciones "pick" y "drop" se utilizan para llevar la pelota y "*move*" cambia la posición del robot. Por último, el axioma indica que el robot sólo puede llevar una pelota si allí sale un agarre ("*gripper*") libre en él.

PDDL PROBLEM

PDDL Problem define el objetivo del problema de planificación en cuestión. El estado final del dominio se describe en la declaración "*goal*". La cuantificación y los operadores lógicos pueden utilizarse con el fin de especificar el objetivo. El estado inicial del dominio se define en la declaración "*init*". Los planificadores deben encontrar un conjunto parcialmente ordenado de acciones que den lugar al estado objetivo desde el estado inicial. En la Figura 9, se muestra un ejemplo de problema PDDL [9] para el ejemplo de dominio dado anteriormente.

```
(define (problem carry-all-balls)
  (:domain carry-balls)

  (:init (room rooma) (room roomb)
         (ball ball1) (ball ball2)
         (gripper left) (gripper right)

         (at-robby rooma) (free left) (free right)
         (at ball1 rooma) (at ball2 rooma))

  (:goal (and (at ball1 roomb) (at ball1 roomb))))
```

Figura 9: ejemplo de problema PDDL [9]

En este problema se indica que el objetivo del robot es llevar las bolas de una habitación a otra.

3. COMPOSICIÓN BASADA EN PLANIFICACIÓN

El uso de técnicas de planificación para resolver el problema de la composición de servicios web semánticos ha sido estudiado anteriormente utilizando distintos enfoques. Aunque existen similitudes entre todas las propuestas, las mismas difieren en varios aspectos claves, como por ejemplo, la performance, uso de la información semántica, planificadores utilizados, calidad de la solución, etc. A continuación se presenta un breve resumen de algunas de estas propuestas.

El planificador de composición de servicios OWLS-XPlan [10] utiliza las descripciones semánticas de los servicios web atómicos en OWL-S para derivar los dominios y problemas de planificación, e invoca a un módulo de planificación llamado XPlan para generar los servicios compuestos. OWLS-XPlan consiste de varios módulos para el preprocesamiento de información y planificación. Recibe como entrada un conjunto de servicios OWL-S disponibles, una descripción del dominio que consiste de las ontologías OWL relevantes y una consulta de planificación, y devuelve una secuencia de servicios compuestos que satisfacen la consulta objetivo. Para este propósito primero convierte la ontología del dominio y las descripciones de servicios en OWL y OWL-S, respectivamente, a descripciones equivalentes de problema y dominio en PDDL. La descripción del dominio contiene la definición de todos los tipos, predicados y acciones, mientras que la descripción del problema incluye todos los objetos, el estado inicial y el estado objetivo. Ambas descripciones son utilizadas luego por el planificador Xplan para crear un plan (que representa un web service compuesto) en PDDL que resuelve el problema dado en el dominio y estado inicial actuales. En el caso de esta propuesta, aunque el sistema importa las descripciones semánticas, la información semántica proporcionada por las ontologías de dominio no es utilizada, por lo que el módulo de planificación requiere una coincidencia exacta entre las entradas y salidas de los servicios.

El framework PORSCE II [11] tiene como objetivo la composición de servicios web mediante el empleo de planificación, haciendo hincapié en el uso de la información semántica asociada a los servicios. Su contribución se centra en la utilización eficaz de la información semántica presente en la descripción OWL-S de los servicios web para mejorar el proceso de composición facilitando composiciones aproximadas cuando sea necesario, obtenidas a través de planificación. Como primer paso, se traduce el problema de composición de servicios en OWL-S a uno de planificación en PDDL. En este punto, de acuerdo a preferencias del usuario, el proceso de traducción puede tomar en cuenta la semántica resultante del análisis semántico del dominio. Si este es el caso, los conceptos semánticamente equivalentes o relevantes también se incluyen, con el propósito de manejar casos en los cuales no se encuentren planes exactos. Como resultado del proceso de transformación se obtiene un problema de planificación completamente formulado que incorpora toda la información semántica necesaria. Luego, el problema de planificación es exportado a PDDL y se invocan sistemas de planificación externos para obtener planes que conforman la descripción del servicio compuesto deseado. Cada plan se evalúa en términos de medidas estadísticas y de precisión. Finalmente, el sistema incluye una interfaz gráfica que permite la visualización y modificación de los planes generados.

El sistema INDIGO [12] implementa un planificador con un enfoque de planificación heurística diseñado para trabajar en ambientes dinámicos, donde

existen restricciones de tiempo y no se tiene un conocimiento completo del dominio. En INDIGO adicionalmente se propone una herramienta de conversión que traduce las descripciones OWL-S a sus correspondientes descripciones en PDDL. Estas descripciones PDDL, son usadas por el planificador de INDIGO como datos de entrada para planificar una composición de servicios web que alcance un objetivo dado. La propuesta de INDIGO consiste de varios módulos para preprocesar la información, planificar y ejecutar de manera incremental una composición de servicios web. Este toma como entrada: un tiempo de respuesta del sistema, un conjunto de servicios OWL-S disponibles, una descripción parcial del dominio y una consulta de planificación expresada mediante ontologías OWL. El sistema retorna al usuario el resultado correspondiente a la ejecución de una secuencia de un plan de servicios compuesto que satisface el objetivo consultado.

Rodriguez-Mier et al. [13] presentan un algoritmo A* que resuelve el problema del matching semántico de estructuras de mensajes de entrada-salida para la composición de servicios web. Dado un pedido, un grafo de dependencias entre servicios es generado dinámicamente a partir de un subconjunto de los servicios originales de un repositorio externo. Luego, utilizando el algoritmo de búsqueda A*, se encuentra una composición mínima que satisfaga el pedido del usuario. Además, con el objetivo de mejorar la performance, se aplican un conjunto de técnicas dinámicas de optimización sobre el proceso de búsqueda que reducen servicios inútiles o equivalentes, minimizando la cantidad de servicios y maximizando la paralelización.

De los enfoques y herramientas estudiadas, el framework PORSCE II cuenta con la gran ventaja de la extensa explotación que realiza de la información semántica disponible de los servicios web a la hora de resolver el problema de planificación. Esto representa una gran ventaja con respecto a las otras propuestas debido a que permite la obtención de mejores soluciones, y en caso de ser necesario, soluciones aproximadas. Al tener en cuenta esta información al formular y resolver el problema de planificación permite la explotación de las relaciones de equivalencias y jerarquía que puedan existir al momento de formular el servicio compuesto, a diferencia de los otros enfoques, en donde los tipos de las entradas y salidas deben corresponderse de forma exacta. Otra ventaja, es que a diferencia de las otras propuestas, la herramienta PORSCE II no necesita ningún conocimiento previo sobre el dominio para resolver el problema, solo con las descripciones en OWL-S de los servicios web atómicos y las especificaciones de las ontologías correspondientes es suficiente. De entre todos los trabajos relevados, este fue del cual se encontró más información disponible sobre su implementación y funcionamiento. En cambio, en los otros trabajos, en donde la escasa o poco clara documentación fue un gran obstáculo al momento de analizar y entender detalladamente las propuestas realizadas, e incluso en algunos casos hacer funcionar las demos o prototipos disponibles. Por estas razones fue que se decidió utilizar la propuesta realizada por la herramienta PORSCE II como base para este trabajo.

4. HERRAMIENTAS DE PLANIFICACIÓN

En este capítulo se brinda una breve descripción de una serie de planificadores que se analizaron y se presenta una comparativa para argumentar cuales fueron elegidos para el problema que se desea resolver.

4.1. PLANIFICADORES

JPlan

Jplan [14] es una implementación realizada en java del algoritmo de planificación *Graphplan.* Este algoritmo de planificación es independiente del dominio y fue desarrollado en el año 1995. *Graphplan* toma como entrada un problema de planificación expresado en notación *STRIPS* y devuelve como salida en caso de ser posible una secuencia de operaciones que permite alcanzar el objetivo deseado. *Jplan* fue desarrollado en el año 2004 y permite experimentar con el planificador así como implementar variantes del Graphplan. También da la posibilidad de integrar las capacidades de planificación de esta herramienta con otros proyectos.

PDDL4J

Pddl4j (*Pddl for Java*) [15] es una librería de código abierto que facilita la implementación en Java de planificadores basados en el lenguaje de planificación PDDL. Esta librería contiene una parser que es capaz de identificar los distintos elementos de un problema definido en el lenguaje PDDL y posee además una implementación del algoritmo Graphplan.

LPG-td

LPG-td [16] es una versión más reciente del planificador LGP que fue desarrollado en el año 2003. Esta nueva versión (desarrollada en el año 2004) realiza mejoras y extiende la anterior para incorporar el manejo de la mayoría de las características de la versión 2.2 de PDDL. El algoritmo utilizado por el planificador LPG se basa en una búsqueda local y puede usar múltiples heurísticas que saquen provecho de la estructura del grafo de planificación utilizado por el algoritmo.

JavaGP

JavaGP [17] se enmarca en un proyecto denominado "Embeddable Graphplan Implementations" cuyo objetivo es proveer implementaciones en varios lenguajes del algoritmo Graphplan, de forma que estás puedan ser integradas a otros proyectos. Hasta el momento el proyecto además de contar con JavaGP que implementa en Java el algoritmo anteriormente mencionado, cuenta también con Emplan que realiza una implementación en el lenguaje C++. JavaGP es capaz de resolver problemas de planificación expresados en lenguaje STRIP así como problemas expresados en PDDL.

PDDL-driven GraphPlan implementation

PDDL-driven GraphPlan implementation [18] es una implementación realizada en Java del algoritmo de planificación GraphPlan. La última versión de esta implementación fue desarrollada en el 2008 y si bien contiene mejoras con respecto a la versión anterior solo es útil para resolver problemas que pueden

expresarse utilizando un subconjunto reducido de elementos del lenguaje PDDL, lo que acota la cantidad de problemas resolubles con esta herramienta.

4.2. SELECCIÓN DE LOS PLANIFICADORES A UTILIZAR

A la hora de seleccionar las herramientas a utilizar para resolver el problema de la composición de servicios basada en planificación ponderamos varios aspectos entre los que se destacan la compatibilidad de la herramienta con los estándares a utilizar, la performance de la misma, y la posibilidad de modificar, extender y adaptar la herramienta de forma de poder incluirla en nuestro proyecto.

En primera instancia se probaron los planificadores de forma independiente (sin vincularlos directamente con el problema de nuestro proyecto), utilizando los ejemplos que incluían cada uno de ellos e intercambiando los ejemplos disponibles, lo cual nos posibilitó sacar algunas conclusiones iniciales.

Esta primera etapa nos sirvió para descartar la utilización de *PDDL-driven GraphPlan implementation,* dado que era capaz de resolver un grupo muy reducido de problemas de planificación, al no manejar todas las características de PDDL.

También pudimos observar que si bien todos los planificadores deberían ser capaces de resolver los ejemplos adjuntos a otros planificadores, en algunos casos esto no era así. Esto se debía principalmente a que no eran capaces de parsear los ejemplos porque no manejaban exactamente la misma versión del lenguaje PDDL o porque no admitían todas las características de dicho lenguaje.

Una apreciación muy importante que pudimos realizar en esta etapa fue que no todos los planificadores daban la misma solución al mismo problema. Algunos planificadores, por ejemplo, siempre daban una solución secuencial a los problemas, donde una tarea siempre debía realizarse a continuación de otra. En cambio, otros planificadores daban soluciones que presentaban paralelismo (ejecución de tareas en forma simultánea), en los casos en que el problema en cuestión lo permitía. Nos planteamos entonces como meta que nuestro proyecto incluyera al menos un planificador que produjera soluciones secuenciales y otro que fuera capaz de generar soluciones con paralelismo.

Pasada esta primer etapa los cuatro planificadores que quedaban en carrera resolvían todos una gran cantidad de problemas de forma acertada y con una performance más que correcta, optamos por decir cuáles de ellos utilizar en una etapa posterior de nuestro trabajo. En una segunda etapa de consideración los principales factores que se tuvieron en cuenta fueron las posibilidades de integración de los planificadores con el software que estábamos construyendo, la capacidad de estos para resolver los ejemplos que consideramos fundamentales para la exposición de nuestro proyecto, y que los planificadores aportaran distintas características entre sí.

Teniendo en cuenta estos aspectos decidimos utilizar los planificadores *JPLan*, *LPG-td* y *PDDL4J*. Mientras que *JPlan* genera siempre soluciones secuenciales, los otros dos planificadores tienen la capacidad de dar soluciones con paralelismo, por lo que cumplimos la consigna anteriormente planteada. Si bien podríamos haber incluido un solo planificador que manejara soluciones no secuenciales nos tomamos el trabajo de incluir dos de ellos ya que a pesar de la similitud en la forma de las soluciones, estos no resolvían todos los problemas de igual forma por lo que nos pareció que era una contribución importante a nuestro trabajo.

Finalmente terminamos descartando la inclusión del planificador *JavaGP* dado que tres planificadores ya nos parecía un número suficiente. Además el esfuerzo de incluir este planificador no se justifica considerando que no realiza ningún aporte significativo. Incluso *JavaGP* mantiene muchas similitudes con PDDL4J, ya que utiliza las librerías desarrolladas en PDDL4J así como los ejemplos de prueba que se adjuntan en ese planificador.

En el siguiente cuadro comparativo se ilustran a modo de resumen, algunas de las características de los planificadores estudiados.

Planificador	Lenguaje de implementación	Capaz de producir soluciones con paralelismo	Capaz de resolver una variedad de problemas aceptable	Incluido en el prototipo
JPlan	Java	NO	SI	SI
PDDL4J	Java	SI	SI	SI
LPG-td	Java	SI	SI	SI
JavaGP	Java / C++	SI	SI	NO
PDDL-driven GraphPlan implementation	Java	NO	NO	NO

Figura 10. Cuadro comparativo de planificadores

5. IMPLEMENTACIÓN

En este capítulo se describe la implementación de un prototipo que permite integrar varios planificadores en la búsqueda de composiciones de servicios web.

El prototipo se desarrolló como una aplicación web en Java, utilizando la versión Java SE 6u25. La aplicación corre sobre el servidor web Tomcat 7 y se utiliza XAMPP como servidor apache para el manejo de ontologías.

El código fuente del proyecto puede ser descargado en la siguiente dirección: http://sourceforge.net/projects/composicionwssemanticos/

5.1. ARQUITECTURA DEL SISTEMA

La arquitectura del sistema desarrollado en este proyecto se puede dividir en tres capas fundamentales: la vista del usuario, la capa lógica, y los planificadores. La vista del usuario está diseñada de forma que el usuario del sistema ingrese los datos para la composición (esto es, las descripciones semánticas de servicios en OWL-S, las ontologías, los parámetros, etc), y defina el problema a resolver. En la capa lógica se analiza el problema ingresado y se utilizan los servicios semánticos para generar los archivos PDDL que posteriormente serán ejecutados por los planificadores. Finalmente se ejecutan los planificadores seleccionados previamente por el usuario utilizando como entrada los archivos .pddl generados anteriormente y se analiza la salida.

Alguno de los puntos claves que presenta el sistema son:

- Traducción de las descripciones semánticas de los servicios web en OWL-S (atómicas o compuestas) en términos de los operadores de planificación.
- Interacción con el usuario con el fin de adquirir sus preferencias respecto de la composición de servicios deseada mediante métricas para la relajación semántica.
- Exportar el problema de la composición de servicios web como una planificación PDDL con dominio y problema.
- Adquisición de soluciones mediante la invocación de los planificadores externos.
- La flexibilidad en la elección del planificador, ya que se puede utilizar cualquier sistema de planificación compatible con PDDL.
- Visualización de la solución mediante un grafo interactivo.

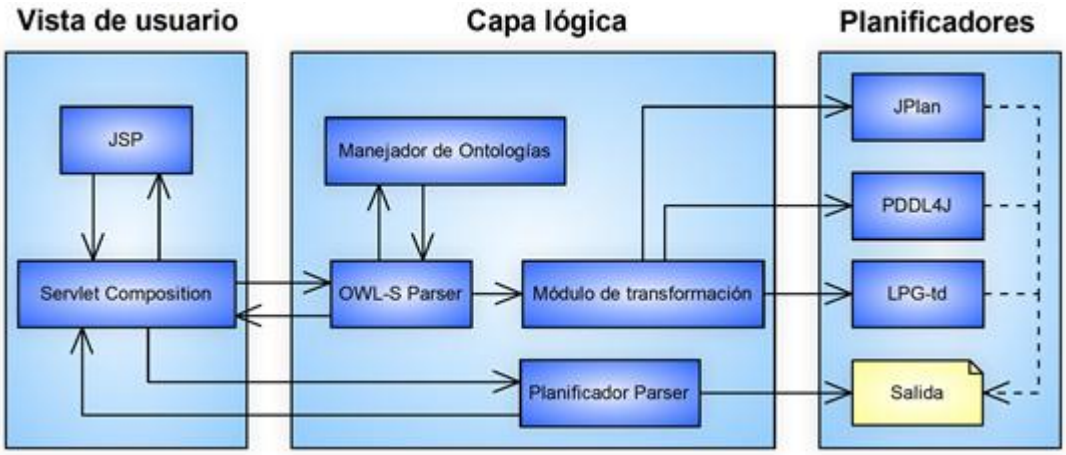

Figura 11: arquitectura del sistema

Dentro de la capa lógica se reutilizaron partes de código del framework PORSCE II (presentado en el Capítulo 4) [11]. La utilización y adaptación de parte de este código simplificó enormemente el proceso de transformación de servicios OWL-S en acciones PDDL. Se crearon dos módulos reutilizando el código de este framework, el OWL-S Parser y el Manejador de Ontologías (MO). El OWL-Parser es responsable del parseo del conjunto de profiles de los servicios web pasados por el usuario y de determinar a qué ontologías pertenecen los conceptos que aparecen en las descripciones de estos servicios. El MO maneja las ontologías relacionadas con el problema pasadas como parámetro por el usuario (las mismas deben ser desplegadas en un servidor Apache al momento de ejecución). Este modulo es el encargado de descubrir las similitudes semánticas existentes entre conceptos dentro de las ontologías. A partir de estos módulos se modificó el Módulo de Transformación de forma de tomar las acciones generadas y pasarlas a un problema en lenguaje PDDL. En esta capa también se agregó un módulo que procesa los resultados de los planificadores (Planificador Parser) para luego mostrarlos en la vista de usuario.

En la vista de usuario se creó un manejador de datos (*servletComposition*) de forma de llevar los datos ingresados por el usuario con el formato adecuado a los módulos de la capa lógica. Este manejador es quién recibe los resultados obtenidos por los planificadores y los procesa para generar la visualización de los planes al usuario.

5.2. PROCESO DE TRANSFORMACIÓN

El proceso de transformación implementado incluye la traducción del problema de composición de servicios web a un problema de planificación y el posible enriquecimiento del mismo con la información semántica disponible. El proceso comienza en el parser OWL-S, el cual parsea las descripciones OWL-S de los servicios atómicos disponibles y se los pasa al Módulo de Transformación. El Módulo de Transformación es responsable de un gran número de operaciones, incluyendo la traducción de las descripciones de los servicios web recibidas desde el parser OWL-S a operadores de planificación y la mejora de ellos con conceptos semánticamente similares obtenidos por el Manejador de Ontologías (MO). Además, interactúa con el usuario para poder plantear el problema de planificación deseado, y exporta tanto el dominio como el problema de planificación a formato PDDL.

Traducción de OWL-S a PDDL

Un problema de planificación es definido como una tupla <*I,A,G*>, tal que I es el estado inicial, A es el conjunto de acciones que pueden ser usadas para modificar estados, y G es el conjunto de metas. Cada acción A_i, tiene tres listas de hechos que contienen las precondiciones de A_i, los hechos que son agregados al estado y los hechos que son borrados del estado luego de ejecutada A_i, denominadas $prec(A_i)$, $efec(A_i)$ y $del(A_i)$ respectivamente.

El primer paso en el proceso de traducción genera el dominio de planificación mediante la traducción de las descripciones en OWL-S de cada servicio web disponible WSD_i, a una acción de planificación en PDDL A_i como se ilustra en la Figura 12.

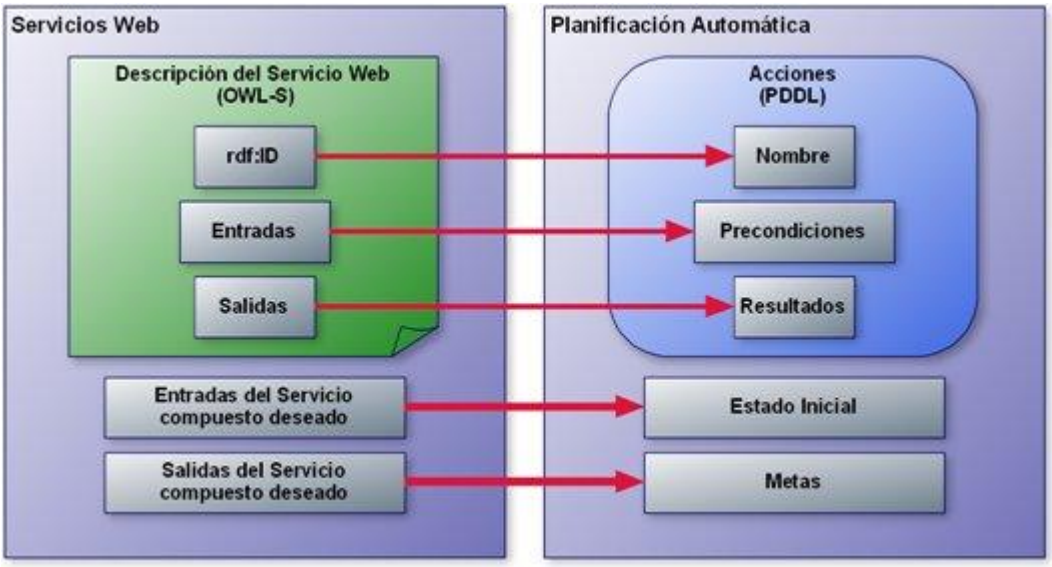

Figura 12: traducción owl-s a pddl

Más específicamente:

- El nombre de cada acción es tomado del campo *rdf:ID* de la descripción del servicio: $nombre(A_i) = WSD_i.ID$
- Las precondiciones de cada acción se obtienen a partir de las definiciones de los inputs de cada servicio: $prec(A_i) \equiv \bigcup_{k=1}^{n} WSA_i.hasInput_k$
- Los efectos de cada acción se obtienen a partir de las definiciones de los outputs de cada servicio: $efec(A_i) \equiv \bigcup_{k=1}^{m} WSA_i.hasOutput_k$
- La lista de los hechos que son borrados se mantiene vacía ya que queda por fuera del alcance de la solución propuesta considerar servicios que tengan efectos negativos sobre el dominio. Queda como trabajo futuro incluir este tipo de servicios en la solución.

Esta transformación puede ser aplicada tanto a servicios web atómicos como compuestos descritos en OWL-S, mientras las salidas del servicio compuesto sean

totalmente deterministas, sin importar como los servicios estén implementados por dentro. En la Figura 13 se presenta un ejemplo de transformación de OWL-s a PDDL, donde se subraya el mapeo presentado anteriormente. La descripción del servicio web del ejemplo recibe como entradas una autorización de pago y un producto, y devuelve como salida la orden generada al realizar el pago. La adición de un parámetro auxiliar en cada predicado PDDL que representa un concepto de entrada o salida es necesaria debido a la restricción de ciertos planificadores que no aceptan predicados sin argumentos. Esto no afecta de ninguna manera la composición resultante.

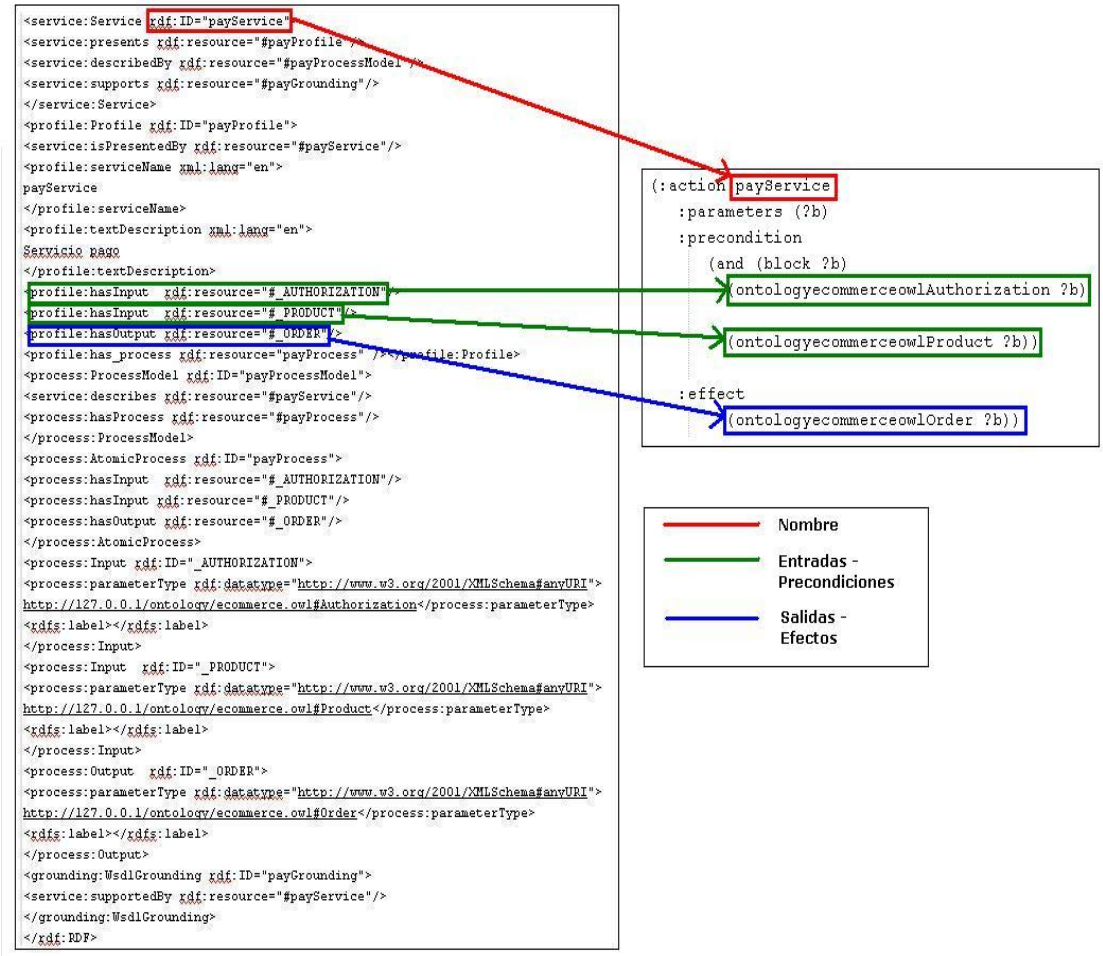

Figura 13: ejemplo de transformación

Luego de la creación del dominio de planificación, el siguiente paso es la generación del problema de planificación correspondiente basado en los requerimientos que el usuario tiene sobre el servicio compuesto. El usuario selecciona del conjunto de conceptos disponibles en la ontología los que desea pasarle al servicio compuesto como entradas, y el conjunto de salidas que desea que devuelva (metas). Más formalmente, la solución empleada en este paso es la siguiente: Sea IC el conjunto de conceptos que el usuario desea pasarle al servicio compuesto y GC el conjunto de salidas deseadas. Si O es el conjunto de todos los conceptos existentes en la ontología, entonces $IC \subseteq O$, $GC \subseteq O$ y $IC \cap GC \equiv \emptyset$. Con las entradas que el usuario desea proveer se formula el estado inicial del problema de planificación, mientras que con las salidas deseadas del

servicio compuesto se formula el objetivo: $I = IC$ y $G = GC$. Tanto el conjunto de entradas como el de salidas son provistos por el usuario.

Análisis semántico

La fase de análisis semántico, que le sigue al proceso de transformación descripto en la sección anterior, permite al sistema la explotación y aprovechamiento de la información semántica disponible. Este paso es implementado en el MO. Durante la traducción, el MO es utilizado de manera extensiva para llevar a cabo la relajación semántica, que es útil en los casos en los cuales no sea posible encontrar un plan en el cual las entradas se correspondan de manera exacta con las salidas. El MO encuentra equivalencias semánticas entre conceptos y otra información semántica relevante al problema actual, para de esta manera, lograr la obtención de planes aproximados.

Dos conceptos de una ontología son considerados semánticamente similares si y solo si existe una relación de jerarquía entre ellos y su distancia semántica no excede cierto valor umbral definido por el usuario.

En lo que se refiere a la relación jerárquica, se utilizan cuatro filtros jerárquicos para su definición entre dos conceptos de la ontología A y B:

- *equivalente(A,B)*: Los dos conceptos tienen el mismo identificador (URI) o deben ser equivalentes en términos de equivalencia de clases OWL, o sea $A = B \vee A \equiv B$.
- *subclase(A,B)*: El concepto A es subclase del concepto B, o sea $A \sqsubseteq B$.
- *superclase(A,B)*: El concepto A es superclase del concepto B, o sea $B \sqsubseteq A$.
- *hermano(A,B)*: Ambos conceptos no tienen que tener una relación jerárquica, ni ser disjuntos; si no que deben tener una superclase en común *S*, tal que $A \sqsubseteq S \wedge B \sqsubseteq S$.

La distancia semántica entre dos conceptos de la ontología es calculada de la siguiente manera: se establece una distancia entre vértices que se basa en la observación que en la estructura de una ontología jerárquica en forma de árbol, cuanto más lejos se encuentren dos conceptos en el árbol, menos relación tienen a nivel semántico. De esta manera, se calcula la distancia semántica entre dos conceptos en término del número de vértices en el camino de menor distancia en el árbol de la ontología. Un vértice existe entre dos conceptos A y B si A es subclase directa de B.

Enriquecimiento Semántico

Luego de completada la traducción y el análisis semántico, es posible utilizar la información sobre relaciones semánticas obtenida a fin de enriquecer el problema de planificación. Este paso es esencial, ya que la representación del problema de composición de servicios como un problema de planificación se ve sumamente beneficiada si el sistema de planificación considera las similitudes semánticas existentes entre conceptos sintácticamente diferentes a la hora de formular los planes.

La implementación de este paso involucra el enriquecimiento de las descripciones del dominio y el problema con toda la información semántica obtenida y dejar que el planificador maneje el problema como un problema de planificación clásico. Esto tiene dos grandes ventajas: por un lado, es posible utilizar cualquier planificador que utilice PDDL como base, ya que el enriquecimiento semántico aplicado sobre el dominio es transparente para el planificador; y por otro lado, se minimizan las interacciones entre el planificador y el MO, mejorando de esta manera la performance.

En la fase de pre-procesamiento, antes de realizar la planificación, el sistema utiliza el MO con el fin de adquirir todos los conceptos semánticamente relevantes tanto como para los hechos del estado inicial como para las salidas de los operadores, obtenidos por el proceso de análisis semántico descripto anteriormente. El enriquecimiento se basa en las siguientes reglas:

- Los conceptos presentes en el estado inicial original, junto con los conceptos semánticamente equivalentes y similares, forman un nuevo conjunto de hechos denominado el estado inicial mejorado (EIM).

- Las metas del problema no cambian.

- Se construye el conjunto de operadores mejorado (COM), que es obtenido modificando las descripciones de cada operador, mientras que se conserva el tamaño original del conjunto. Más concretamente, las precondiciones de cada operador son las mismas, mientras que el conjunto de efectos de cada operador se mejora mediante la inclusión de todos los conceptos semánticamente equivalentes y similares para los conceptos en el conjunto inicial.

A modo de ejemplo, supongamos que el estado inicial es el siguiente:

EI = {tarjetaDebito(X), fechas, motel}

Por otro lado, se cuenta con los siguientes operadores:

ActivarTarjeta: prec={tarjetaCredito(X), noHabilitada(X)}
efectos={habilitada(X)}

ReservaHotel: prec={fechas, hotel}
efectos={reserva}

El MO encuentra las siguientes equivalencias semánticas:

tarjetaDebito ≈ tarjetaCredito

motel ≈ hotel

activo ≈ habilitado

El preprocesador modifica la definición del problema de la siguiente manera:

EIM = {tarjetaDebito(X), fechas, motel, tarjetaCredito(X), hotel}

COM : ActivarTarjeta: prec={tarjetaCredito(X), noHabilitada(X)}
efectos={habilitada(X), activa(X)}

ReservaHotel: prec={fechas, hotel}
efectos={reserva}

Este nuevo problema obtenido, es escrito en PDDL y es pasado al sistema de planificación con el fin de adquirir una solución. Al codificar la información semántica de esta manera se logra que la misma sea transparente para los planificadores, por lo que estos son capaces de resolver el problema como cualquier otro problema de planificación clásico.

5.3. OBTENCIÓN DE SOLUCIONES

Dado que el proceso de transformación resulta en la exportación tanto del dominio como del problema de planificación a PDDL, cualquier sistema de planificación externo compatible con PDDL puede ser utilizado a la hora de obtener resolución de la composición. Esta es una característica de gran importancia debido a que le da al sistema la posibilidad de incorporar nuevos planificadores y de esta manera mantenerse al día con los avances en la investigación sobre planificación. Como se mencionó anteriormente, se seleccionaron tres planificadores para ser incorporados al sistema: JPlan, que es una implementación open-source de GraphPlan, LPG-td y PPDL4J. Los tres planificadores probaron ser sumamente rápidos y capaces de manejar un gran número de operadores, lo cual es de suma importancia ya que esto permite que el sistema sea escalable y pueda ser utilizado para resolver problemas con un gran número de servicios web. Luego de concluido el proceso de planificación Jplan retorna el plan en un formato propio, compuesto de una lista secuencial de acciones. Por otro lado, LPG-td, exporta el plan a un formato compatible con PDDL. En este caso el plan puede ser no secuencial, sino estructurado en niveles. Las acciones en un mismo nivel pueden ser ejecutadas en cualquier orden, pero todas las acciones de un cierto nivel deben ser completadas antes de que se ejecuten las acciones del nivel siguiente. Esto es representa una gran ventaja sobre los demás planificadores ya que permitiría la ejecución en paralelo de los servicios perteneciente a un mismo nivel.

Luego de ejecutados los planificadores, los planes obtenidos son visualizados.

5.4. VISUALIZACIÓN DE LOS PLANES

El visualizador permite una mejor comprensión del servicio compuesto obtenido al proporcionar una representación grafica del mismo. El servicio compuesto es representado como un grafo donde los nodos corresponden a los servicios atómicos que lo componen. Además se muestran las entradas y salidas de cada servicio web incluido en el grafo.

5.5. PROBLEMAS TÉCNICOS

Los principales inconvenientes surgidos durante la implementación del prototipo se dieron al intentar reutilizar parte de un sistema ya existente, el PORSCE II, e integrar nuevas funcionalidades para agregar otros planificadores.

Para el parseo y traducción de los servicios semánticos en acciones y posteriormente en problemas planteados en PDDL, hubo que modificar parte de la traducción que realizaba el PORSCE II ya que el resultado generado no se correspondía con los estándares PDDL, generando errores al intentar integrar nuevos planificadores al proyecto. El costo de estos cambios fue bastante significativo puesto que la implementación del PORSCE II no es modularizada.

Otro de los problemas que surgieron fue para la representación gráfica del resultado obtenido. Se muestra el resultado mediante una visualización interactiva, de forma que el usuario pueda comprender el proceso de composición e interactuar con los nodos de los servicios viendo la información de cada uno. Para esto se optó por desarrollar un applet de Java, usando la librería JGraphX [19]. Esta librería permite generar nodos enlazados, donde cada nodo representa un servicio semántico. En cada nodo se agregan las entradas y salidas correspondientes al servicio y se enlaza de acuerdo al orden de la composición. Si bien esta librería permite ese comportamiento, para algunas funcionalidades brindadas encontramos que no se comportaba como se especifica.

6. CASOS DE PRUEBA

Los casos de prueba fueron definidos con el objetivo de evaluar las funcionalidades principales del prototipo, estas son, la composición de servicios semánticos utilizando distintos planificadores y la posibilidad de utilizar distintos niveles semánticos. Se definieron dos casos de prueba principales, el primero apuntando a la composición de servicios sin el uso de niveles semánticos buscando las distintas soluciones planteadas por los diferentes planificadores. El segundo caso aplica lo niveles semánticos de equivalencia, superclase, subclase y hermanos explicados previamente en la sección 5.2 (Análisis Semántico).

6.1. TIENDA DE COMERCIO ELECTRÓNICO

Este caso de prueba fue el que sirvió como punto de partida para la implementación del prototipo. Trata de una tienda de comercio electrónico donde se cuenta con servicios para la búsqueda del producto mediante una descripción (*searchProductService*), la autorización del número de tarjeta de crédito para la compra de un producto (*authorizeService*), el pago del producto con la tarjeta (*payService*), la entrega del producto a la dirección de domicilio del usuario (*deliveryService*), y la finalización de la compra (*finalizeBuyService*).

Para ejecutar el caso de prueba se selecciona como datos iniciales a la descripción del producto (hasDescription), la dirección de domicilio del usuario (Address) y el número de tarjeta de crédito (CreditCard). Para el resultado que se quiere obtener se selecciona la factura de la compra realizada (Invoice). En este caso de prueba no se aplica ninguno de los niveles semánticos.

El resultado de correr este ejemplo con los tres planificadores fue el siguiente:

- **JPlan**

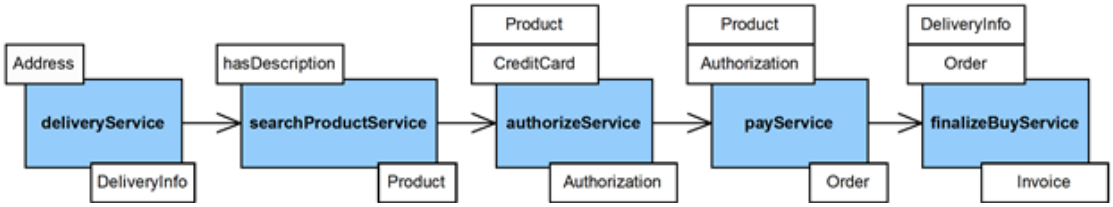

Figura 14: resultado de ejecución de JPlan

Con este planificador vemos que el resultado es secuencial, es decir, cada servicio debe ser ejecutado a continuación del otro. En esta solución en primera instancia se ejecuta el servicio de entrega del producto, pidiendo como dato de entrada la dirección seleccionada por el usuario y retornando la información de entrega. Esta será utilizada al ejecutarse el servicio de finalización de la compra. Luego se ejecuta la búsqueda del producto tomando como entrada la descripción ingresada y retornando el producto y a continuación se ejecuta la autorización de la tarjeta de crédito para la compra del producto obtenido en el paso anterior. Con la autorización obtenida y el producto se ejecuta el servicio de pago, que retorna como resultado la orden de compra. Por último está el servicio de finalización de la compra, donde se recibe la información de entrega (obtenida en la primera ejecución) y la orden de compra, retornando la factura de la compra realizada.

- **Pddl4j y LPG-td**

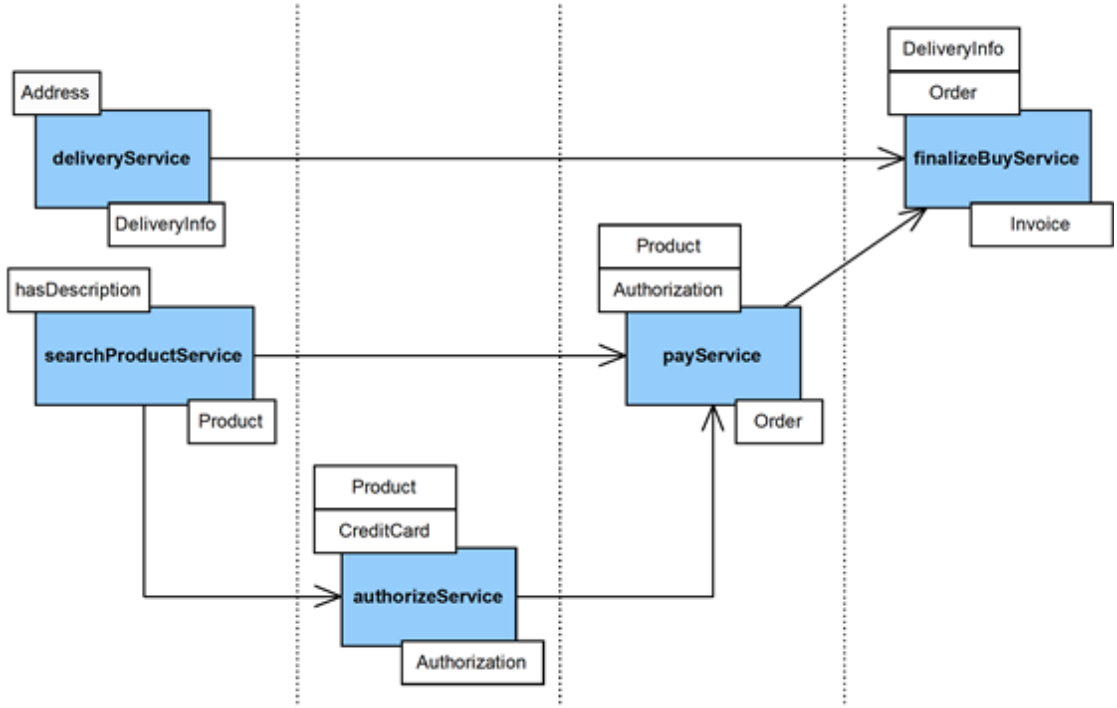

Figura 15: resultado de ejecución de Pddl4j y LPG-td

El resultado para estos dos planificadores es el mismo, pero difieren con el de JPlan pues en este caso las ejecuciones pueden ser en paralelo. Esto ocurre con las primeras dos ejecuciones, tanto para el servicio de búsqueda como para el servicio de entrega, los datos de entrada no dependen de ninguna otra salida, por lo tanto pueden ejecutarse en paralelo. Luego de este primer paso se ejecuta el resto de los servicios de igual forma que en el caso anterior.

6.2. BOOKSTORE

Este caso de prueba trata de un comercio electrónico de libros. A diferencia del caso anterior en este se aplican niveles semánticos para la ejecución y se compara con la ejecución sin aplicar los mismos.

El bookstore cuenta con los siguientes servicios web semánticos involucrados al realizar una compra:

- *BookToPublisherService*: Este servicio recibe como entrada la información de un libro y un autor y retorna la información de la editorial correspondiente.

- *CreditCardChargeService*: Este servicio recibe como entradas la información de la orden de compra y un número de tarjeta de crédito, retornando el pago para realizar la compra.

- *CustomsCostService*: Este servicio recibe como entrada la información de la orden de compra y la información de la editorial, y retorna el costo de aduana para esa orden.

- *ElectronicOrderService*: Este servicio recibe como entrada el tipo de pago electrónico y retorna la orden de compra.

- *PublisherElectronicOrderService*: Este servicio recibe como entrada la información de la editorial y retorna la información de la orden de compra.

- ShippingService: Este servicio recibe como entrada la información de la dirección de domicilio del usuario y los datos de la compra y retorna la fecha de entrega.

- WaysOfOrderService: Este servicio recibe como entrada la información de la editorial del libro y retorna el tipo pago electrónico.

Para ejecutar el caso de prueba se selecciona como datos iniciales al autor del libro (Author), la información del libro a comprar (Book), el número de tarjeta de crédito (creditcard) y la dirección de envío (Address). Para el resultado que se quiere obtener se selecciona la factura de la compra realizada (payment), la fecha de envío de la compra (ShippingDate) y el costo de aduana para la orden (CustomsCost).

En primera instancia se ejecuta la composición sin aplicar ninguno de los niveles semánticos dando como resultado:

- **JPlan**

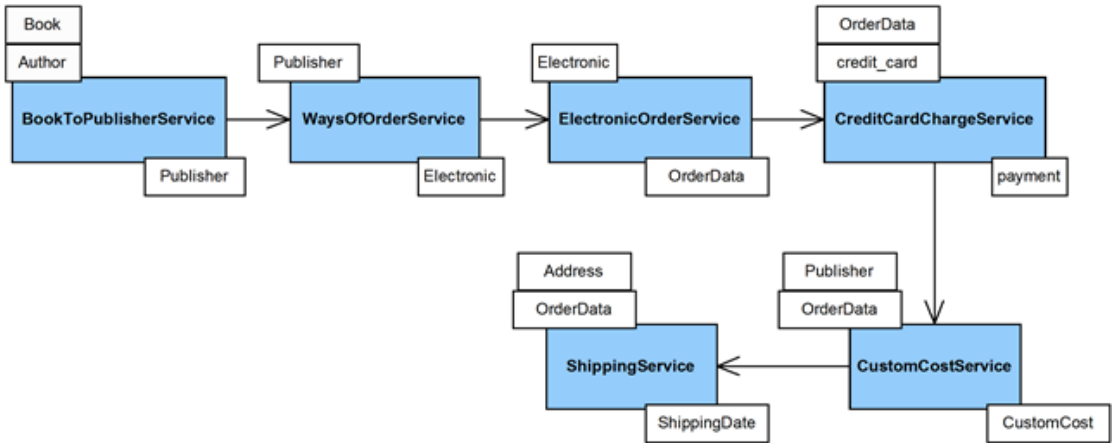

Figura 16: resultado de ejecución de JPlan sin aplicar niveles semánticos

Al igual que en el ejemplo anterior vemos que la ejecución del caso de prueba con este planificador retorna un resultado secuencial donde cada servicio se ejecuta después del otro. En primera instancia se busca la editorial (Publisher) usando los datos del libro y autor (Book y Author) ingresados por el usuario. Con el resultado de este se ejecutan los servicios WaysOfOrderService y ElectronicOrderService para obtener los datos de la compra. A continuación se utilizan esos datos para obtener el pago (payement, previo ingreso de los datos de la tarjeta de crédito), los costos de aduana (CustomCost) y finalmente la fecha de envío (ShippingDate).

- **Pddl4j y LGP-td**

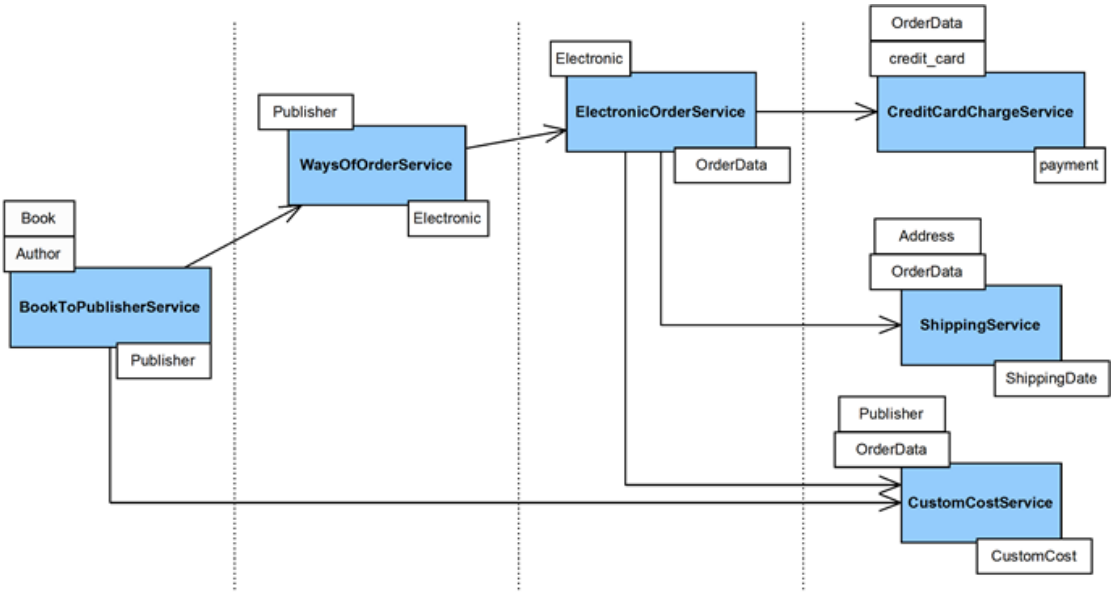

Figura 17: resultado de ejecución de Pddl4j y LPG-td sin aplicar niveles semánticos

Con este resultado vemos que utilizando estos planificadores se pueden correr los últimos tres servicios que ejecuta el JPlan de forma paralela. El flujo de datos es el mismo que con el planificador anterior con la ventaja que se puede ejecutar todo en 4 pasos, a diferencia de 6 como con el JPlan.

En segunda instancia se aplican todos los niveles semánticos con valor en 1 dando como resultado:

- **JPlan**

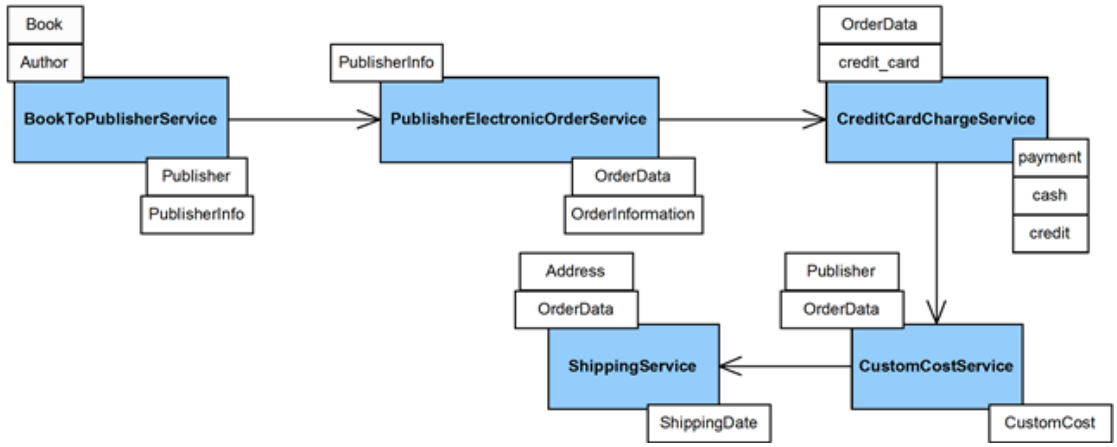

Figura 18: resultado de ejecución de JPlan aplicando niveles semánticos

En el resultado se ve como a diferencia de la primera ejecución sin niveles semánticos, en este caso luego de la búsqueda de la editorial se devuelve como resultado PublisherInfo, que permite ejecutar el servicio PublisherElectronicOrderService para obtener la información de la compra en un solo paso, acortando el proceso respecto de la primera ejecución. El flujo de datos termina ejecutando los otros servicios del mismo modo que el ejemplo anterior. Esto permite que el proceso con este planificador se realice en 5 pasos en lugar de los 6 que eran necesarios sin utilizar los niveles semánticos.

- **Pddl4j y LGP-td**

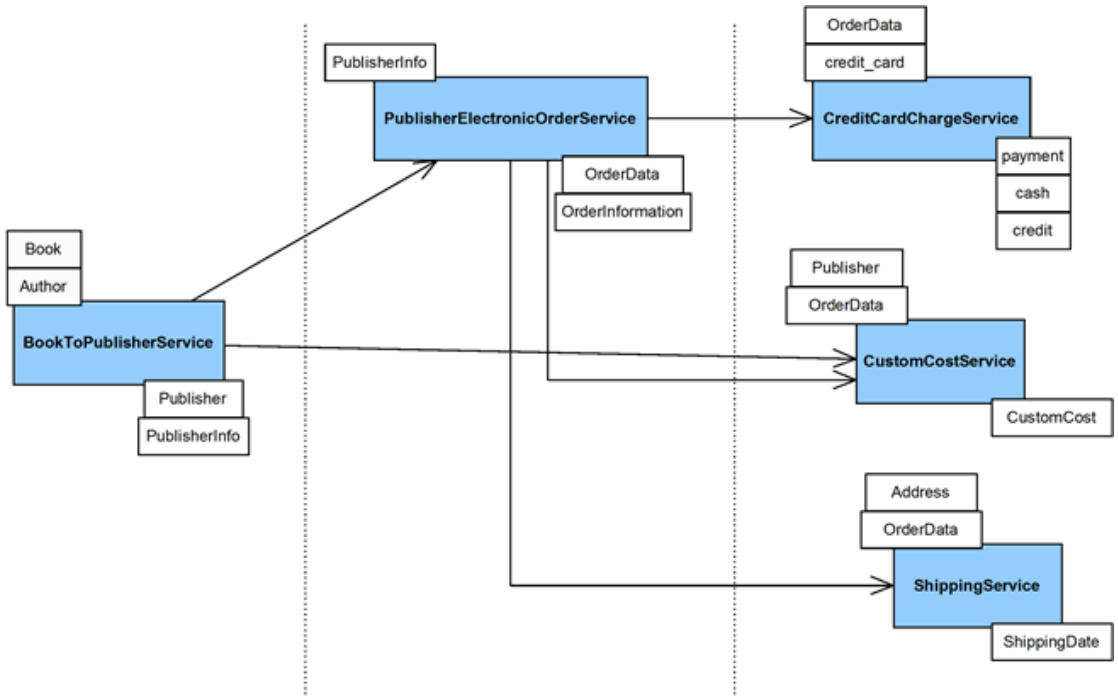

Figura 19: resultado de ejecución de Pddl4j y LPG-td aplicando niveles semánticos

Al igual que con el JPlan, en este caso el proceso se reduce en un paso con respecto a la ejecución sin niveles semánticos. En primera instancia se realiza la búsqueda, luego se obtiene la información de la compra y finalmente se ejecutan en forma paralela los servicios para obtener el pago, el costo de aduana y la fecha de envío.

Las diferencias que se presentan entre los dos ejemplos de este caso de prueba están dadas por la utilización de los niveles semánticos. Esto permite ampliar el dominio posibilitando que el servicio PublisherElectronicOrderService acepte como entrada una entidad del tipo Publisher y no solo del tipo PublisherInfo, al considerarlo semánticamente equivalentes (ya que una es subclase de la otra). Esto permite soluciones aproximadas cuando no hay soluciones exactas posibles o reducir el número de servicios ejecutados para un mismo objetivo.

7. CONCLUSIONES, LIMITACIONES Y TRABAJOS FUTUROS

7.1. CONCLUSIONES

El objetivo principal del proyecto era construir una plataforma para la composición dinámica de servicios web semánticos utilizando planificadores, y a su vez estudiar las herramientas ya existentes para este tipo de problemas. Se logró implementar un prototipo que cubre los siguientes aspectos:

- Traducción de las descripciones semánticas de los servicios web OWL-S (atómicas o compuestas) en los operadores de planificación.

- Interacción con el usuario con el fin de adquirir sus preferencias respecto de la composición de servicios deseada mediante métricas para la relajación semántica.

- Exportar el problema de la composición de servicios web como una planificación PDDL con dominio y problema.

- Adquisición de soluciones mediante la invocación de los planificadores externos.

- La flexibilidad en la elección del planificador, ya que se puede utilizar cualquier sistema de planificación compatible con PDDL.

- Visualización de la solución mediante un grafo interactivo.

Un aspecto que consideramos negativo fue el tiempo invertido en el estudio de las herramientas de composición ya existentes. Si bien esto era interesante y necesario como punto de partida, consumió más tiempo de lo esperado debido a que algunas de las herramientas disponibles no contaban con las funcionalidades requeridas para nuestro proyecto y aquellas que si servían no contaban con buena documentación o código claro. Como punto positivo de dicho análisis es que se deja documentado para proyectos similares o trabajos que se realicen sobre este mismo.

Consideramos que el prototipo se puede tomar como base para terminar el objetivo general de construir una herramienta de composición de servicios web que además de dicha composición determine las ventajas y desventajas del uso de cada uno de los planificadores.

Desde el punto de vista académico el proyecto fue de gran valor para nosotros ya que nos aportó conocimientos sobre temas que no son comúnmente tratados como los servicios web semánticos y que actualmente están siendo fuertemente estudiados.

7.2. LIMITACIONES

La principal limitación que posee la herramienta desarrollada es que por tratarse de un prototipo fue testeada con un número acotado de casos de prueba. La idea inicial era que la herramienta fuera capaz de resolver el problema planteado en un principio, que estaba enmarcado en un escenario de comercio electrónico. Luego de que corroboramos que era capaz de resolver dicho problema, realizamos pruebas utilizando otros ejemplos de complejidad relativamente simple, para los cuales la herramienta se comportó de forma acertada. Sin embargo no se garantiza que sea capaz de resolver problemas de una gran complejidad.

7.3. TRABAJOS FUTUROS

En esta sección se comentan aquellas características o funcionalidades que no fueron implementadas en el prototipo realizado, ya sea por restricciones de tiempo, o porque quedaban por fuera del alcance del proyecto. Es importante mencionar que por más de que algunas características hayan quedado al margen del prototipo, de todas formas se realizó un estudio de las mismas analizado el impacto de la inclusión de estas en el trabajo realizado.

Los elementos que quedaron por fuera del prototipo podrían clasificarse en tres grupos que podríamos denominar como extensiones, mejoras y alternativas. Las extensiones comprenden aquellos elementos que agregan alguna funcionalidad o característica que el trabajo realizado no posee. Las mejoras serían aquellos aspectos que de incluirse en el prototipo podrían enriquecer o perfeccionar alguna funcionalidad ya existente. Por último, las alternativas hacen referencia a determinados aspectos que si bien fueron abordados o desarrollados de una forma particular, podrían haber sido encarados de otra manera distinta, dado que no había una única forma y hubo que optar por alguna de ellas.

Dado que el alcance de nuestro proyecto se limitaba a la composición semántica de servicios web, utilizando únicamente el prototipo desarrollado no es posible recrear exactamente lo que sería un escenario real de interoperabilidad en la Web Semántica. En un escenario real, dado cierta tarea u objetivo a alcanzar (por medio de la utilización de servicios web), primero es necesario efectuar la tarea de descubrimiento de los servicios web disponibles, posteriormente se debe determinar (si es posible) una composición de servicios que permita cumplir el objetivo, y finalmente realizar la ejecución de los servicios siguiendo la composición determinada anteriormente, para alcanzar el objetivo deseado. Por lo tanto nuestro trabajo podría extenderse abarcando también las tareas de descubrimiento y ejecución de los servicios web. Visto desde otro punto de vista podría decirse también que aquellas herramientas capaces de descubrir y/o ejecutar servicios web de forma automatizada, podrían extenderse para generar composiciones de los servicios, incluyendo un prototipo como el desarrollado en nuestro trabajo.

Otra extensión que podría aplicar a nuestro proyecto sería la inclusión de más planificadores, lo cual podría ser útil en los casos en los que se desea obtener un gran número de composiciones distintas para realizar un mismo objetivo, o cuando se debe afrontar un objetivo de una complejidad tan grande, que es necesario probar con varios planificadores con distintas heurísticas hasta hallar alguno capaz de resolver el problema planteado.

En cuanto a las alternativas que se plantearon en este proyecto, podría decirse que la disyuntiva fundamental radicaba en si optábamos por implementar todos los componentes del prototipo, o si en cambio nos decidíamos por utilizar herramientas ya desarrolladas. Como se pudo apreciar se utilizaron varias herramientas previamente implementadas como los planificadores, por el hecho de que no nos parecía adecuado invertir tiempo en desarrollar planificadores (u otros componentes), si ya había varios que funcionaban correctamente. Sin embargo podríamos haber desarrollado un planificador específico para nuestro proyecto.

De la mano con la alternativa anterior, también nos planteamos si utilizaríamos lenguajes o mecanismos propios para resolver o plantear los problemas que se nos presentaban, o utilizaríamos los estándares ya establecidos. Si bien optamos por utilizar los estándares existentes, como por ejemplo PDDL, nos topamos con varias herramientas que utilizan sus propios lenguajes y mecanismos. El hecho de utilizar notaciones propias nos hubiera permitido por ejemplo implementar algunos componentes de forma más sencilla sin preocuparnos por ciertas restricciones que implica la utilización de estándares.

En relación a las mejoras que pueden hacerse en nuestro prototipo, una de las más importantes consistiría en contemplar la existencia de ciertos Servicios Web que actualmente nuestro trabajo no contempla. Esta omisión se presenta ya que al momento de la Traducción de OWL-S a PDDL no se considera la existencia de servicios que tengan efectos negativos sobre el dominio, o sea servicios que al realizar la traducción a PDDL deriven en acciones cuya ejecución implique eliminar hechos de un estado.

También podrían realizarse algunas mejoras en la interfaz de usuario, como por ejemplo permitir realizar algún tipo de manipulación adicional al grafo que presenta las soluciones encontradas (además de las que ya dispone), de forma de poder visualizar y entender las soluciones encontradas de forma más clara aún.

8. BIBLIOGRAFÍA

1. **OWL-S: Semantic Markup for Web Services.** [Online] 2004. [Citado: 08/08/2014.] http://www.daml.org/services/owl-s/1.1/overview/.

2. **SAWSDL: Semantic Annotations for WDSL Working Group.** [Online] 2007. [Citado: 08/08/2014.] http://www.w3.org/2002/ws/sawsdl/.

3. **Semantic Web.** [Online] [Citado: 08/08/2014.] http://semanticweb.org.

4. **World Wide Web Consortium (W3C).** [Online] [Citado: 08/08/2014.] http://www.w3.org/.

5. Nilsson R., Fikes N. **STRIPS: a new approach to the application of theorem proving to problem solving**. Artificial Intelligence, vol. 2 (3-4), pp. 189-208, 1971.

6. The AIPS-98 Planning Competion Commitee. **The Planning Domain Definition Language Version 1.2**. Technical Report CVC TR-98-003/DCS TR-1165 (New Haven, CT: Yale Center for Computational Vision and Control), 1998.

7. Pednault E. **ADL. Exploring the Middle Ground Between STRIPS and the Situation Calculus**. Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning, pp. 324-332, 1989.

8. Erol K., Hendler J., Nau D. **UMCP: A sound and complete procedure for hierarchical task network planning (poster)**. Proceedings of the Second International Conference on Artificial Intelligence Planning Systems, pp. 249-254, 1994.

9. Aydın O., Kesim N., Cicekli Ilyas C. **Automated web service composition with the event calculus**. 8th International Workshop, ESAW 2007, Athens, Greece, October 22-24, 2007, Revised Selected Papers, Lecture Notes in Computer Science, vol. 4995, pp 142-157, 2008.

10. Klusch M., Gerber A. **Semantic Web Service Composition Planning with OWLS-XPlan**. Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web, 2005.

11. Hatzi, O., Meditskos, G., Vrakas, D., Basiliades, N., Anagnostopoulos, D., Vlahavas, I. **PORSCE II: Using Planning for Semantic Web Service Composition**. In: Proc. of the ICKEPS2009 in Conjuction with ICAPS 2009, Thessaloniki, Greece (2009)

12. Guzmán Luna J.A., Ovalle Carranza D.A**. Un Modelo de Planificación Incremental para Servicios Web Semánticos**. Revista Avances en Sistemas e Informática, vol. 4, núm. 3, pp. 141-150, Colombia, 2007.

13. Rodriguez-Mier P., Mucientes M., Lama M**. Automatic web service composition with a heuristic-based search algorithm**. IEEE International Conference on Web Services (ICWS), pp. 81-88, 2011.

14. **JPlan: Java GraphPlan Implementation.** [Online] [Citado: 19/06/2014.] http://jplan.sourceforge.net.

15. **Pddl4j (Pddl for Java).** [Online] [Citado: 19/06/2014.] http://sourceforge.net/projects/pdd4j/.

16. **LPG-td.** [Online] [Citado: 19/06/2014.] http://zeus.ing.unibs.it/lpg/.

17. **JavaGP.** [Online] [Citado: 19/06/2014.] http://emplan.sourceforge.net/.

18. **PDDL-driven GraphPlan implementation.** [Online] [Citado: 19/06/2014.] http://www.zeynsaigol.com/software/graphplanner.html.

19. **JGraphX - Open Source Java Component.** [Online] [Citado: 16/04/2014.] http:// www.jgraph.com/

ANEXOS

A. MANUAL DE USO

Antes de ejecutar la aplicación, las ontologías necesarias deben ser desplegadas en un servidor Apache y los descriptores OWL-S de los servicios semánticos involucrados deben ser situados en la misma máquina desde donde se ejecutará la aplicación.

En la primera pantalla, se deben indicar la ruta en la máquina donde se encuentran los descriptores de los servicios (en el ejemplo "C:_eCommerce") y la ruta en el servidor Apache donde están desplegadas las ontologías requeridas (en el ejemplo "http://127.0.0.1/ontology"). Luego de presionando el botón "Seguir" el sistema empezará con el proceso de parseo de las ontologías, lo que puede llevar unos segundos.

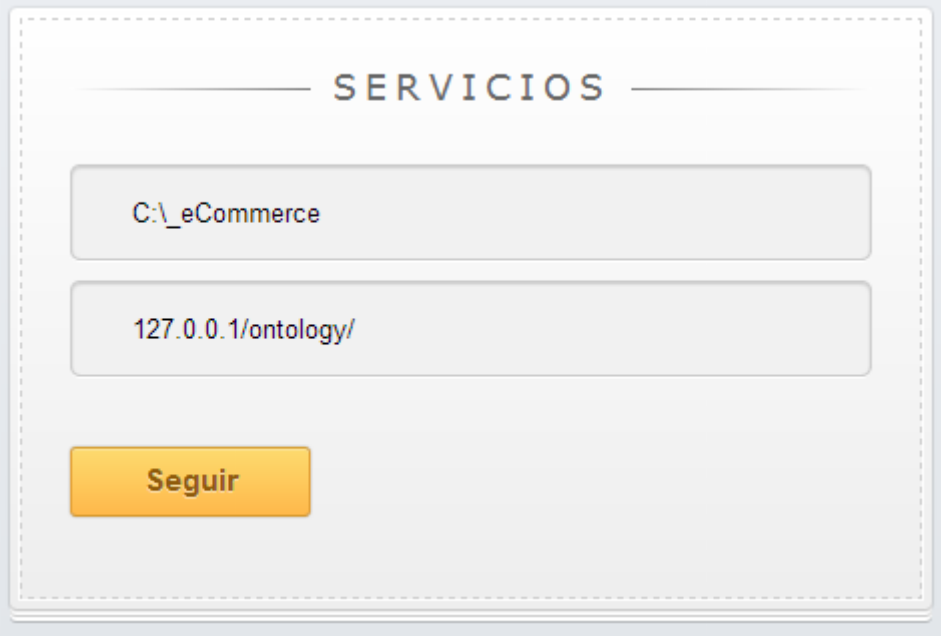

Figura 20: vista para el ingreso de los servicios

En la siguiente pantalla, luego de parseadas las ontologías, se debe plantear el problema que se quiere resolver y como se desea que sea resuelto. Para esto se deben indicar las siguientes tres cosas:

- En el primer cuadro se plantea el problema que se quiere resolver a partir el estado inicial y las metas que se quieren alcanzar. En primer lugar, se debe formular el estado inicial del problema, seleccionando de la lista de todas las entradas de los servicios disponibles cuales son las que el usuario ingresa (En el ejemplo "#CreditCard", "#DeliveryInfo" y "#hasDescription"). Estas serán las entradas del servicio compuesto que se desea obtener. Por otro lado, se plantea la meta que se desea obtener, indicando en la lista de todas las salidas de lo servicios disponibles, cuales son las salidas que el usuario desea obtener (En el ejemplo "#Invoice"). Estas serán las salidas del servicio compuesto.

- En el segundo cuadro se indican cuales de los planificadores disponibles se utilizarán para obtener la resolución del problema (JPlan, Lpgtd y Pddl4j).

- En el tercer cuadro se deben indicar las opciones referidas a la información semántica disponible que se desea tener en cuenta a la hora de resolver el problema. Aquí se indica si se desea que se tenga en cuenta la relación de equivalencia entre conceptos y que umbrales de profundidad para las relaciones de padre, hijo y hermano deben ser considerados. En la sección 6.2 se explica detalladamente el significado de estas opciones.

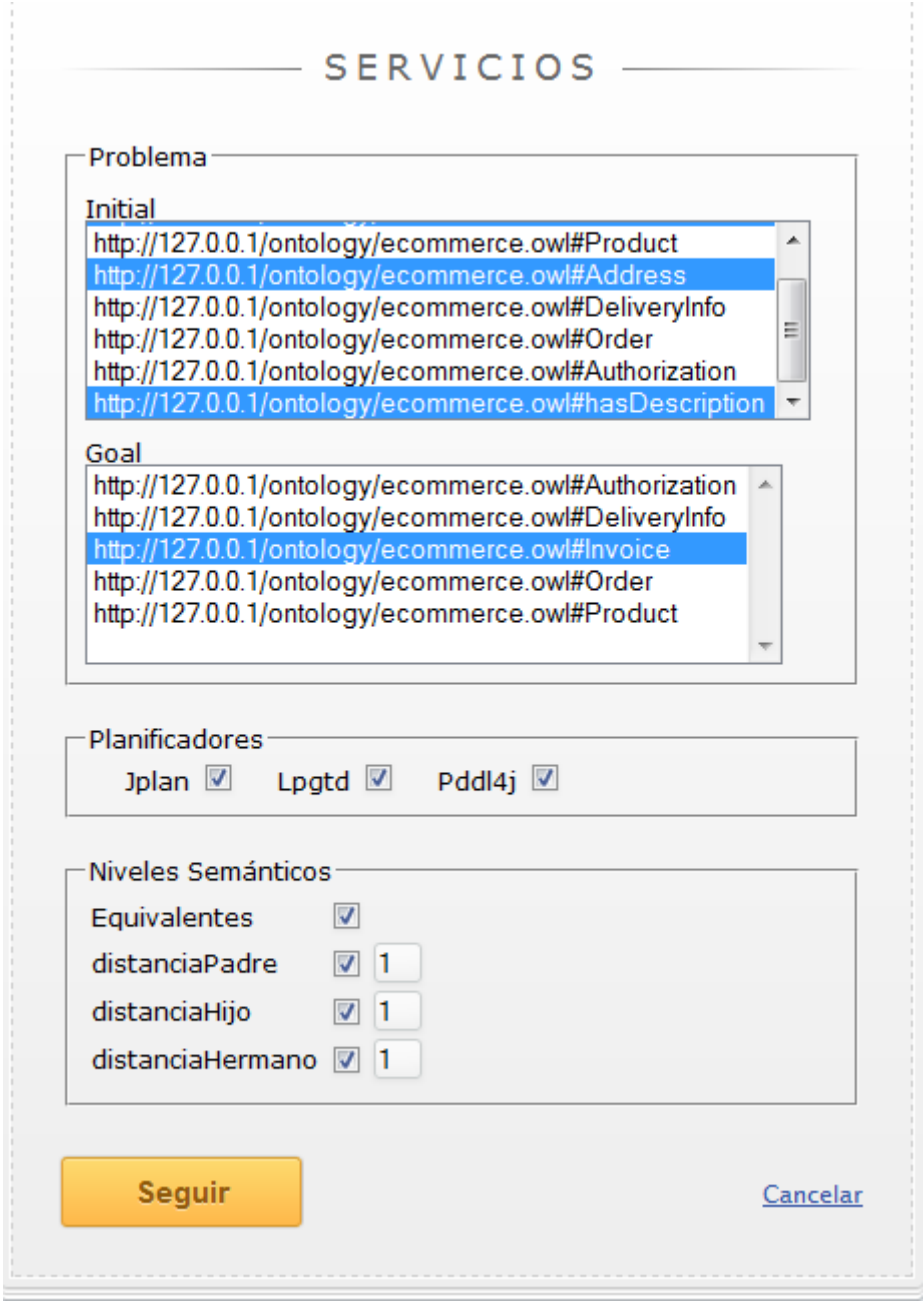

Figura 21: vista para definir el problema

Finalmente, se despliegan los planes obtenidos como solución al problema por cada uno de los planificadores, representando de forma gráfica lo servicios

compuestos construidos. En esta pantalla el usuario tiene la opción de desplegar u ocultar los inputs y los outputs de cada uno de los servicios atómicos involucrados en las soluciones resultantes.

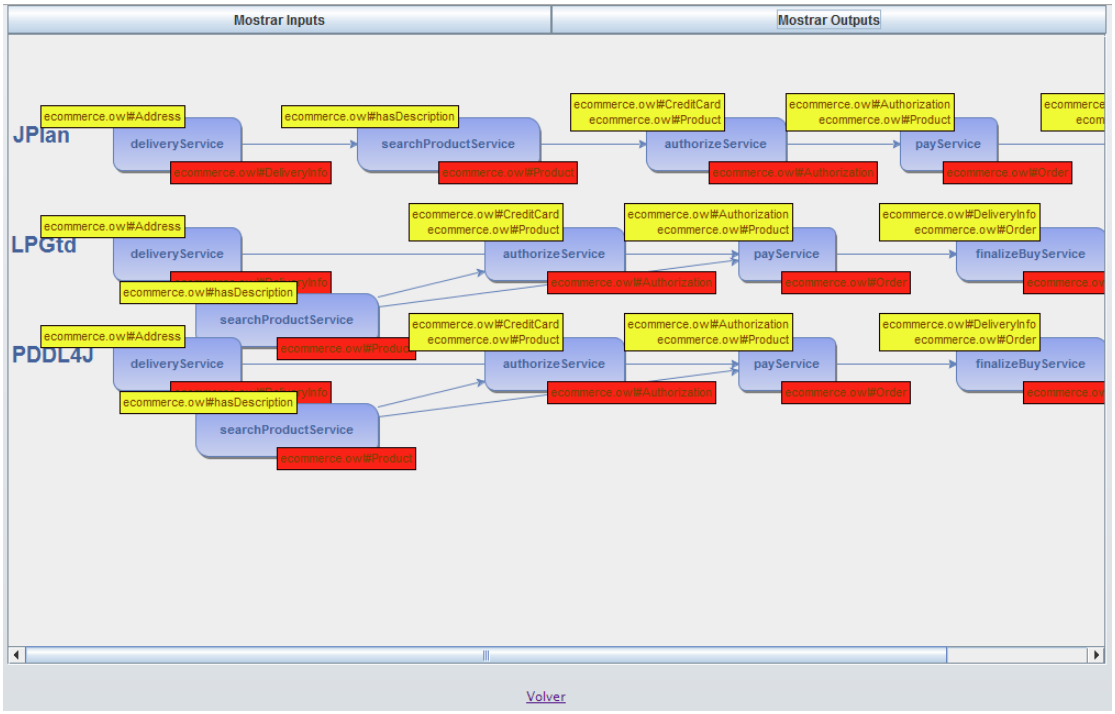

Figura 22: vista de los planes obtenidos

B. GLOSARIO

ADL (Action description language)

Es un lenguaje de planificación automática, particularmente diseñado para robots. Si bien se considera una mejora de STRIPS tiene varías diferencias sustanciales con este lenguaje, ya que ADL sigue el paradigma de mundo abierto (considera como desconocida y no como falsa cualquier cosa que no ocurra en las condiciones) y permite además la utilización de disyunciones y literales negativos.

Ontología

Es un esquema conceptual que representa de manera formal y consensuada especificaciones de conceptos, que proveen un conocimiento compartido y común de un dominio. Las ontologías utilizan lenguajes formales para describir de forma precisa tanto los conceptos de un determinado dominio como las relaciones existentes entre esos conceptos, por lo que brindan información semántica procesable por aplicaciones.

OWL (Web Ontology Language)

Es un lenguaje de marcado utilizado para publicar y compartir datos usando ontologías en la Web Semántica. Es capaz de representar conocimiento complejo acerca de entidades, permite agruparlas y describir las relaciones entre estás. Está diseñado para ser utilizado por aquellas aplicaciones que necesitan procesar contenido y consta de tres sub-lenguajes (OWL Lite, OWL DL, and OWL Full) con distintos niveles de expresividad.

OWL-S

Es una ontología basada en OWL (Web Ontology Language) utilizada para describir servicios web semánticos. Está escrita en XML y sigue principalmente las definiciones del lenguaje RDF (Resource Definition Framework). Los objetivos principales de OWL-S son permitir la automatización del descubrimiento, de la invocación y de la composición de servicios web para lograr la interoperabilidad.

PDDL (Planning Domain Definition Language)

Es el lenguaje de planificación automática más utilizado actualmente. Su sintaxis es muy similar al lenguaje de programación LISP y está fuertemente influenciado por el lenguaje STRIPS. Contiene además la expresividad de muchos otros lenguajes de planificación como ADL (para incorporar las proposiciones) lógicas y UMCP (para incorporar acciones).

RDF (*Resource Description Framework*)

Es una familia de especificaciones de la W3C originalmente diseñada como un modelo de datos para metadatos. Es utilizada como un método general para la descripción conceptual o modelado de la información que se implementa en recursos Web. RDF está pensado para ser leído y entendido por computadoras y está escrito en XML.

SAWSDL (Semantic Annotations for WSDL and XML Schema)

Es una recomendación del W3C que permite añadir anotaciones semánticas a los componentes del lenguaje WSDL para cooperar con la clasificación, descubrimiento, composición e invocación de servicios Web. SAWSDL proporciona mecanismos mediante los cuales los conceptos de los modelos semánticos definidos dentro o fuera del documento WSDL, pueden ser referenciados dentro de los componentes de WSDL como anotaciones. Esta semántica expresada en lenguajes formales contribuye a eliminar la ambigüedad en la descripción de los servicios Web durante el descubrimiento automático y la composición de los mismos.

Servicio web

Conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web. Estos servicios proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, que interactúan entre sí para presentar información dinámica al usuario.

Servicio Web Semántico

Son servicios web que cuentan con meta-información que permite facilitar la automatización del proceso de descubrimiento, composición e invocación, así como también la interoperabilidad y ejecución de los mismos. Algunos de los mecanismos más importantes para la incorporación de semántica a la descripción de los servicios web, están basados en el uso de ontologías, tales como OWL-S y WSMO.

STRIPS (Stanford Research Institute Problem Solver)

Es un generador de planes automatizado o planificador desarrollado a principios de la década de 1970. El término STRIPS también hace referencia al lenguaje formal para modelar las entradas de este planificador. El lenguaje STRIPS es la base para la mayoría de los lenguajes utilizados para expresar problemas de planificación automática.

UMCP

Es un sistema de planificación de tareas basado en el algoritmo HTN (Hierarchical task network) e implementado en el lenguaje LISP. La idea del algoritmo de planificación automática HTN es crear un plan por descomposición de tareas en subtareas, hasta lograr primitivas que pueden ser ejecutadas directamente. La dependencia entre las acciones se proporciona en forma de red y la descomposición se aplica en cumplimiento de precondiciones según una jerarquía.

W3C (World Wide Web Consortium)

Es un consorcio internacional que produce recomendaciones para la World Wide Web, donde trabajan conjuntamente las organizaciones miembro, el personal del consorcio y el público en general. La misión del W3C es guiar la Web hacia su máximo potencial, para lo cual esta organización ha desarrollado y

desarrolla una gran cantidad de estándares vinculados al manejo de información en la Web.

Web semantic

La web semántica es una propuesta de W3C (World Wide Web Consortium) que implica la creación y utilización de tecnologías para publicar datos en la web, de forma que estos sean legibles por aplicaciones informáticas Se basa en la idea de añadir metadatos semánticos y ontológicos a la Web, que describan el contenido, el significado y la relación de los datos de manera formal. El objetivo principal de es mejorar Internet ampliando la interoperabilidad entre los sistemas.

WSDL (Web Services Description Language)

Es un lenguaje basado en XML utilizado para describir servicios Web. WSDL describe la interfaz pública de los servicios Web, explicitando la forma de comunicación, lo que incluye los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen de manera abstracta y se ligan posteriormente al protocolo concreto de red y al formato del mensaje.

WSMO (Web Service Modeling Ontology)

Es un modelo conceptual que sirve para representar los aspectos relevantes de los Servicios Web semánticos. Provee un mecanismo basado en ontologías y un lenguaje formal para describir semánticamente los Servicios Web con el objetivo de facilitar la automatización del manejo de los mismos y la interoperabilidad entre ellos.

XML (Etensible Markup Language)

Es un lenguaje de marcas desarrollado por W3C utilizado para almacenar datos en forma legible. Deriva del lenguaje SGML y permite definir la gramática de lenguajes específicos para estructurar documentos grandes. Fue diseñado principalmente para soportar publicaciones electrónicas de gran escala, y es utilizado para el intercambio de datos en la Web (y otros ámbitos). También es especialmente útil cuando varias aplicaciones se deben comunicar entre sí o integrar información.

XML Schema

Es un lenguaje que sirve para describir la estructura, las restricciones y la semántica de los contenidos de los documentos XML de forma precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML. Este lenguaje, desarrollado por W3C, permite brindar una percepción del tipo de documento con un nivel alto de abstracción, e imponer ciertas reglas a las aplicaciones que utilizan documentos XML.