Un Modelo de Planificación Incremental para Servicios Web Semánticos

An Incremental Planner Model for Semantic Web Services

Jaime Alberto Guzmán Luna, MSc., Demetrio Arturo Ovalle Carranza PhD.
GIDIA: Grupo de Investigación y Desarrollo en Inteligencia Artificial
Escuela de Ingeniería de Sistemas, Universidad Nacional de Colombia Sede Medellín
{jaguzman, dovalle}@unal.edu.co

Recibido para revisión 10 de Septiembre de 2007, Aceptado 30 de Noviembre de 2007, Versión final 10 de Diciembre de 2007

***Resumen*—Este documento presenta las características principales del planificador de Inteligencia Artificial que hace parte de la propuesta del sistema INDIGO, un sistema de planificación para la composición de servicios Web OWL-S, el cual permite construir de manera incremental en la Web semántica un plan de composición de servicios Web OWL-S que pueda ser ejecutado de manera concurrente durante su composición. Las características presentadas aquí por el planificador permiten atacar problemas asociados al proceso de composición como son la observación parcial de entorno, las restricciones impuestas por los usuarios sobre el tiempo que el sistema puede emplear deliberando antes de actuar y el comportamiento altamente dinámico que presenta la Web.**

Palabras Clave— Planificación Incremental, Composición de Servicios, Web Semántica, Ontología, Búsqueda Heurística.

***Abstract*—This paper shows the main characteristics of Artificial intelligence planner which is part of INDIGO system's proposal, a planning system for OWL-S Web services composition. This system allows constructing on incremental way within Semantic Web an OWL-S Web services composition plan which can be executed on concurrent way during its composition. The characteristics described here by the planner allow face associated problems related to composition process such as the partial observation of environment, the restrictions imposed by the users about the time that the system could use to deliberate before acting and the highly dynamic behavior presenting on the Web environment.**

***Keywords*—Incremental Planning, Services Composition, Semantic Web, Ontology, Heuristic Search.**

I. INTRODUCTION

La tecnología de los servicios Web presenta la ventaja de permitir de una manera lo suficientemente simple la conformación de servicios complejos a partir de servicios más simples. Este caso se extiende a los Servicios Web Semánticos (SWS)[15], tales como los especificados en OWLS[9], WSMO[17] ó WSDL-S[16], donde la composición en tiempo de diseño ha sido ampliamente estudiada y hoy en día existen diferentes herramientas disponibles que llevan a cabo esta tarea apoyados en diferentes planificadores de Inteligencia Artificial [4].

Los planificadores utilizados en dichas soluciones se desempeñan adecuadamente en la solución de problemas de composición donde: a) se dispone de un conocimiento detallado y completo del dominio, b) no existe ninguna restricción de tiempo para su solución y c) el mundo es determinístico, por lo que se considera el mismo durante la planificación y la ejecución. Sin embargo, se ha identificado claramente que las anteriores condiciones en general no se cumplen en el contexto real de la Web ya que: a) no se puede esperar que se tenga toda la información relevante en la base de conocimiento local del sistema, por lo cual el planificador al tener información incompleta necesitará recolectar alguna información con el fin de resolver el problema de composición, b) en la Web para la mayoría de los usuarios existen limitaciones sobre el tiempo que el sistema puede emplear deliberando antes de actuar, y c) La Web es un mundo altamente dinámico lo que hace que el efecto que produce un servicio no siempre sea conocido ni predecible. Entonces la pregunta que surge es, ¿como se pueden enfrentar los anteriores problemas de composición mediante un planificador?

Para este propósito, en este documento se hace una propuesta general del sistema INDIGO, en el cual se propone un planificador que implementa las características básicas de los planificadores *anytime* (los cuales son muy útiles cuando el tiempo disponible para su computación es limitado) bajo un

enfoque de planificación heurística diseñado para trabajar en ambientes dinámicos, como ocurre en el contexto de la Web, donde existen restricciones de tiempo y no se tiene un conocimiento completo del dominio, con el fin de utilizarlo para la composición de servicios Web semánticos. En INDIGO adicionalmente se propone una herramienta de conversión que traslada las descripciones de servicios Web OWL-S 1.1 a sus correspondiente descripciones en el Lenguaje de Definición de Dominios de Planificación (PDDL) [5], lenguaje estándar centrado en la acción, el cual desde su creación ha supuesto un punto de referencia como lenguaje de modelado para la inmensa mayoría de los planificadores. 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.

El presente documento está estructurado como sigue: En la sección 2 se proporciona una visión general de la arquitectura del sistema INDIGO, seguido por una breve descripción del modulo convertidor de OWL-S a PDDL en la sección 3. El corazón del Sistema INDIGO, el planificador INDIGO, es presentado y comparado con otros trabajos en las secciones 4 y 5 respectivamente. Las conclusiones y trabajo futuro son presentadas en la sección 6.

II. VISION GENERAL DE INDIGO

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 de la Web y una consulta de planificación expresadas 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.

Esta propuesta se basa en el contexto de los sistemas de planificación y ejecución concurrente [14] para tratar los diferentes problemas asociados con la composición de servicios Web semánticos que fueron relacionados en la sección anterior. Los sistemas que siguen esta aproximación solapan los procesos de planificación y ejecución. Estos sistemas ofrecen diversas ventajas frente a los sistemas de planificación y ejecución alternada utilizados en algunas propuestas para llevar a cabo la composición de servicios Web semánticos, ya que ofrecen una mayor reactividad a los cambios del entorno y un menor tiempo total de planificación

Basándose en lo anterior, la arquitectura de INDIGO consta de un Convertidor de especificaciones OWL-S a PDDL, el planificador para llevar a cabo una planificación incremental de los servicios Web y el ejecutor para inicializar el problema de planificación y ejecutar de manera concurrente cada una de las acciones que son planteadas por el planificador, como se detalla en la Figura 1.

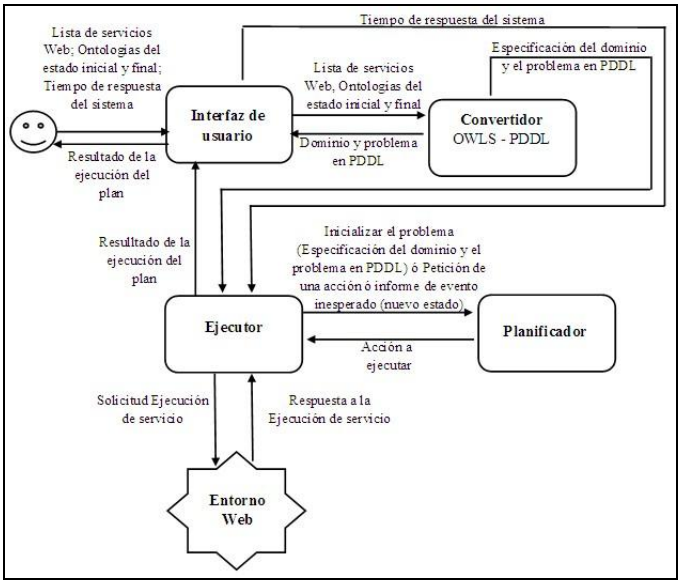

Figura 1. Arquitectura del Sistema INDIGO

INDIGO primero convierte las ontologías del dominio y de la consulta, junto con las descripciones de los servicios implementadas respectivamente en OWL y OWL-S, a las descripciones equivalentes del problema y el dominio en el lenguaje PDDL usando un convertidor de especificaciones. La descripción del dominio contiene la definición de todos los tipos, predicados y acciones, mientras la descripción del problema incluye todos los objetos, el estado inicial, y el estado objetivo. Ambas descripciones son usadas por el planificador de Inteligencia Artificial de INDIGO para crear un plan de composición de manera incremental el cual cuando sea ejecutado concurrentemente logre solucionar la consulta dada. Es así, como un operador PDDL del dominio de planificación corresponde a un perfil del servicio (*service profile*) en OWL-S ya que tanto el operador y el perfil, en esencia, describen un patrón o plantilla de cómo una acción o una instancia de un servicio Web debería ser vista.

En general el comportamiento funcional que se propone en INDIGO luego de pasar los datos de entrada del sistema por el convertidor de especificaciones, es el detallado en la Figura 2, donde el ejecutor inicia el proceso de planificación enviándole las especificaciones del dominio, el estado y los objetivos en el formato PDDL, para lo cual el sistema de planificación realiza el cálculo de la primera acción (a_0) y luego de un tiempo de espera (que es asignado por el usuario) el ejecutor realiza una petición de esta primera acción al planificador, quien le entrega la especificación de la misma. Luego de esto, el ejecutor lleva a cabo la ejecución de esta acción a_0, para lo cual toma un tiempo en dicha ejecución, el cual es aprovechado por el planificador para calcular la segunda acción del plan. Cuando el ejecutor termina de

ejecutar la primera acción solicita al planificador la segunda acción y se repite el proceso hasta que se alcancen todos los objetivos solicitados. En caso de que falle una de las acciones durante su ejecución, el ejecutor informa al planificador de este evento indicándole el nuevo estado luego de la falla, a lo que el planificador realiza un nuevo cálculo de dicha acción mientras que el ejecutor espera, y luego del respectivo tiempo, el ejecutor solicita al planificador la acción correspondiente, quien le retorna la especificación de la nueva acción.

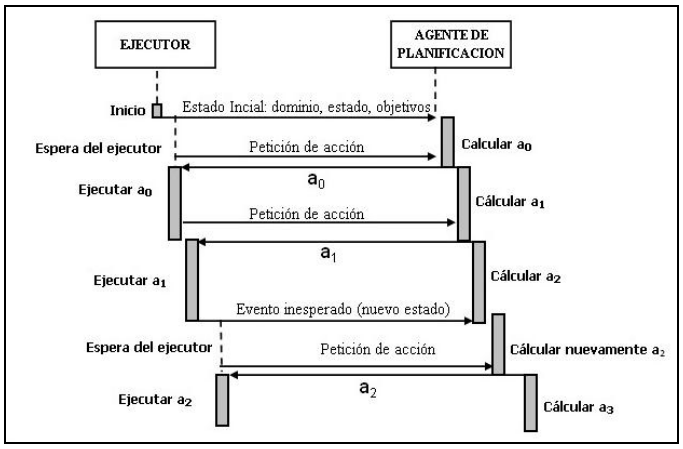

Figura 2. Diagrama de secuencias del funcionamiento de INDIGO

III. EL CONVERTIDOR DE OWLS A PDDL

Con el fin de obtener la definición del dominio PDDL, Este módulo realiza la conversión de las clases (*OWL Class*) y propiedades (*OWL Property*) incluidas en las ontologías del estado inicial y final a los tipos (*PDDL Type*) y predicados (*PDDL Predicate*) respectivamente del dominio en el PDDL. Es así como, por cada clase (*OWL Class*) que aparezca en la ontología inicial y final escrita en OWL se incluirá un tipo (*PDDL Type*) en el dominio del PDDL, mientras que por cada propiedad (*OWL Property*) que aparezca representada en la ontología inicial y final se creará un predicado (*PDDL Predicate*) en la descripción del dominio. Una vez realizado lo anterior, el convertidor lleva a cabo un mapeo de las descripciones de los servicios OWL-S a las acciones en el PDDL. Para ello, por cada servicio Web se crea una acción en el dominio del PDDL. La conformación de la estructura interna de cada acción PDDL se lleva a cabo como se describe a continuación. Todo parámetro de una precondición del servicio (*hasPrecondition parameter*) puede ser directamente transformado a una precondición de la acción mediante el uso de predicados. Lo mismo se realiza para el caso de cualquier parámetro de un efecto del servicio (*hasEffect parameter*). En el caso de la conversión de un parámetro de Salida (*hasOutput parameter*) de un servicio OWL-S particular, es decir, la información que el servicio ofrece al mundo, su traslación al PDDL es un poco más compleja. El problema es que la condición *hasEffect* del servicio describe explícitamente como el estado del mundo cambiará mientras este no es el caso para el valor de un parámetro de salida (*hasOutput parameter*), aunque este puede implícitamente influenciar el proceso de planificación de la composición. Sin embargo, el PDDL no permite describir un conocimiento no físico, tal como por ejemplo como la lista de sillas disponibles de un vuelo la cual es producida como salida de un servicio.

Este problema puede ser resuelto al realizar un mapeo del parámetro de salida "*X*" (instancia *output X* del servicio) a un tipo especial del parámetro *hasEffect* del servicio. En particular, toda variable de salida *A* es descrita en y adicionada al estado del mundo de planificación por medio de un nuevo tipo de predicado que adiciona un efecto en el PDDL el cual se llamará "*agentKnows(X)*". De manera similar, en el caso de cualquier parámetro de entrada "*Y*" (*hasInput parameter*) del perfil del servicio OWL-S, este se correlaciona con un parámetro del mismo nombre en la acción PDDL y se le adiciona como precondición un predicado del tipo especial "*agentKnows(Y)*". Teniendo en cuenta lo anterior, INDIGO puede utilizar la descripción de un servicio durante su proceso de planificación solamente, si el predicado adicional de la precondición "*agentKnows*(*Y*)" en el conocimiento disponible sobre el dato de entrada del servicio es satisfecho de tal forma que *X*= *Y* se cumple. De lo contrario, la ejecución del servicio fallaría ya que al chequear las precondiciones del servicio se puede encontrar que ellas no son satisfechas en el estado actual del mundo.

Para obtener la definición del problema en el PDDL, el convertidor por cada instancia, tanto de una clase, como de las propiedades de la Ontología OWL inicial genera un objeto y un predicado PDDL, que hace parte de la descripción del estado inicial del problema. Así mismo, por cada instancia tanto de una clase, como de las propiedades de la Ontología OWL final, el convertidor genera un objeto PDDL y un predicado que hace parte de la descripción del estado objetivo del problema (consulta de planificación).

IV. EL PLANIFICADOR DE INDIGO

El planificador de INDIGO, está diseñado para trabajar en entornos dinámicos, buscando generar planes de forma incremental y totalmente deliberativa dentro de una arquitectura que permita llevar acabo la construcción y ejecución de los planes concurrentemente, para abordar problemas potenciales propios de la Web, tales como la falta de información completa sobre el entorno, la aparición de situaciones inesperadas y la existencia de restricciones en los tiempos de respuesta del planificador. En primer lugar, el planificador toma decisiones a partir de una descripción incompleta del entorno, consiguiendo lograr primero aquellos objetivos que según la información disponible son alcanzables y programando "acciones de sensado" mediante la ejecución de servicios de información cuyos efectos sean del tipo "*agentKnows(X)*" que permitan obtener la información necesaria para lograr el resto de objetivos. En segundo lugar,

el algoritmo de planificación trata de encontrar en cada ciclo de ejecución únicamente la siguiente acción a ejecutar, reduciendo el esfuerzo de planificación (ya que no se desgasta en calcular un plan completo donde cualquier cambio en el entorno o en los objetivos puede invalidar tal plan) y evitando tener que tomar decisiones sobre acciones lejanas en el tiempo. Por último, el tiempo de deliberación del planificador, puede ser controlado, así el planificador dispone en cualquier momento de una repuesta (acción a ejecutar) que se va refinando, mientras exista tiempo disponible.

A. Visión General del algoritmo de planificación

En general un problema de planificación $P = (O; I; G)$ es una tripleta donde O es el conjunto de operadores, I es el estado inicial y G los objetivos a alcanzar. Para su solución, el algoritmo del planificador de INDIGO consta de cuatro pasos (ver Figura 3):

Un primer paso, que es el Preproceso, consiste principalmente en una instanciación de los operadores y de los predicados proporcionados en la especificación del dominio junto a la instanciación de la especificación del problema de planificación.

El segundo paso del algoritmo corresponde al Grafo Relajado de Planificación, el cual esta relacionado con la generación de la heurística necesaria para la construcción de los planes.

El tercer paso, se relaciona directamente con la Generación de los Planes Mono-Objetivos, donde el algoritmo calcula de forma concurrente un plan, para cada sub-objetivo del problema y finalmente se lleva a cabo.

El cuarto paso, que consiste en la Ordenación de los Planes, con el cual se busca seleccionar cual será la primera acción a ejecutarse entre todas aquellas que conforman cada uno de los planes. La especificación de dicha acción se envía al ejecutor para su ejecución.

Este proceso se repite continuamente, hasta que el usuario decide detener la ejecución del planificador, o si el planificador logra alcanzar todos los objetivos.

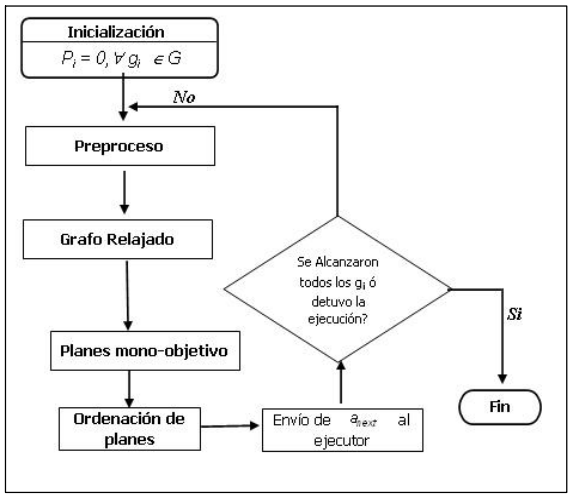

Figura 3. Esquema general del algoritmo de planificación en INDIGO. El ejecutor interacciona con el planificador para solicitarle acciones y notificarle la información adquirida del entorno

B. Etapa de preproceso

En esta etapa se procesa y organiza información sobre el problema y el dominio los cuales se encuentran especificados mediante el lenguaje PDDL en los archivos generados por el conversor de OWL-S a PDDL. Esta etapa se divide en cuatro tareas:

Verificación del dominio y del problema PDDL, mediante el uso de un parser se verifica que la especificación del dominio y del problema PDDL sean sintácticamente correctos. El planificador de INDIGO permite el manejo de literales al igual que el manejo de variables numéricas, también el manejo de precondiciones y efectos conjuntivos y el uso de la negación en las precondiciones y efectos. Así mismo, incluye la habilidad para expresar una estructura de tipos para los objetos en un dominio permitiendo expresar el tipo de los parámetros que aparecen en las acciones y restringiendo los tipos de argumentos en los predicados.

Instanciación de acciones y literales, para generar todas las acciones posibles, se sustituyen los parámetros de los operadores por valores concretos, del mismo modo, para generar los literales, se instancian los parámetros de los predicados. Esta etapa puede realizarse más de una vez dado que en la ejecución pueden aparecer nuevos objetos o lo que es lo mismo, se requiera asignar nuevos valores a los parámetros de los operadores.

Análisis de alcanzabilidad, en la etapa anterior se generan algunas acciones no alcanzables, es decir, que no existe un plan que pueda conducir a un estado en el que se satisfagan todas las precondiciones de la acción. Esta comprobación tiene un alto costo computacional, por ello, cuando se relaja el problema se ignoran las precondiciones numéricas y negadas de las acciones. Este análisis genera un ahorro de tiempo considerable, sin embargo, durante el proceso de ejecución alguna de las acciones ignoradas puede convertirse en alcanzable si el entorno de la Web evoluciona de forma inesperada, razón por la cual la tarea se puede volver a repetir, si el ejecutor así lo notifica.

Conformación de la estructura de datos optimizada. Toda la información del dominio y del problema (literales, acciones, objetivos...) se almacenan en estructuras de datos, con el fin de obtener nueva información.

En INDIGO, las dos primeras etapas están unificadas de manera que solo se instancien, aquellas acciones que corresponden a acciones alcanzables. Inicialmente se compara los literales de la situación inicial con las precondiciones de los operadores, cuando estos coinciden, se almacena el valor correspondiente. Así, se obtiene un conjunto de valores posibles para cada parámetro del operador. Probando luego todas estas combinaciones, se obtienen un conjunto de

acciones donde algunas son alcanzables, siendo los efectos de estas últimas los que se añaden a la lista de literales. El proceso termina cuando no es posible instanciar ninguna nueva acción.

C. Etapa del grafo relajado de planificación

El Grafo Relajado de Planificación (RPG), proporciona la información heurística necesaria para la construcción de los planes [1] [7]. El RPG, es un grafo dirigido y nivelado que consta de dos tipos de nodos, literales y acciones, y dos tipos de aristas, aristas de precondiciones y de efectos. El primer nivel L_0, corresponde al nivel de todos los literales ciertos en el estado actual S_0. Los niveles de acciones A_t contienen todas las acciones aplicables al nivel anterior de literales L_t; el nivel de literales se extiende L_{t+1} con la inclusión de los efectos positivos de dichas acciones. La expansión del RPG termina cuando se alcanza un nivel de literales que contiene todos los objetivos del problema, o cuando ya no es posible aplicar ninguna nueva acción, como señala el algoritmo descrito en la Figura 4.

```
t = 0; L0 = S0                                  // Inicialización
while ∃gi 2 G/gi ∉ Lt do                        // Nueva expansión
  At = {a ∈ A/Lprec+(a) ∈Lt}                    // Nivel de acción
  Lt+1 = Lt ∪Leff+(a), ∀a ∈ At                  // Nivel de literales
  if Lt+1 = Lt then fail endif
  t = t + 1
endwhile
```

Figura 4. Expansión tradicional del RPG.

La expansión tradicional del RPG presenta, sin embargo, varios inconvenientes, ya que no tiene en cuenta características importantes del problema de composición de servicios Web, como son la incertidumbre del estado actual de la Web, los objetivos numéricos y las acciones de sensado. A continuación se describen las extensiones que se han planteado para incluir estas características en el grafo relajado.

1) Representación del conocimiento parcial del estado actual de la Web.

Para manejar el conocimiento parcial sobre el estado actual de la Web, se emplea una extensión definida como lógica trivaluada (verdadero, falso y desconocido), la cual contrario a muchos planificadores, permite dudar del valor de verdad de un literal que no se haya incluido explícitamente en la descripción de un estado. Tal literal es descrito como desconocido, así, los niveles de literales se denominaran niveles proposicionales, ya que estos contendrán proposiciones lógicas (no literales). Igualmente, el primer nivel L_0, tendrá todas las proposiciones lógicas que se satisfacen en S_0. Los niveles de acciones A_t contiene todas las acciones, cuyas precondiciones positivas y negativas se encuentran en el nivel L_t; el siguiente nivel L_{t+1}, extiende L_t con los efectos positivos y negativos de las acciones A_t. Por lo anterior, un nivel proposicional puede contener dos proposiciones que representen un mismo literal, uno en forma positiva y otro en forma negativa. Este algoritmo del RPG modificado se presenta en la Figura 5. De esta forma, en t pasos de tiempo, podemos conseguir que una proposición P_i, sea verdadera o falsa con lo cual, L_t contendrá entonces las propociones P_i, como, $\neg P_i$.

```
t = 0;                  // Incialización con restricciones de tiempo
Lt = {pi ∈ PL/satisfy (pi, S0) = ν }
while ∃gi ∈ G / gi ∉ Lt do                  // Nueva expansión
  At = {a ∈ A / Lprec(a) ∈ Lt }             // Nivel de acción
  Lt+1 = Lt ∪ Leff(a), ∀ a ∈ At             // Nivel Proposicional
  if Lt+1 = Lt Then fail endif
   t = t + 1
endwhile
```

Figura 5. Algoritmo de expansión del RPG considerando la incertidumbre en los valores lógicos de las proposiciones.

2) Manejo de funciones numéricas

Durante la composición de servicios es necesario trabajar con problemas de planificación numéricos por lo que es necesario también considerar como se va a relajar la parte numérica del problema, es decir las precondiciones numéricas de las acciones, y el criterio de optimización. Respecto a las precondiciones y efectos numéricos, se sigue la aproximación del planificador Sapa [2], el cual los ignora en el problema simplificado, lo que permite trabajar de forma más sencilla el problema de planificación relajado (RPG). Dado que durante este proceso de composición de servicios surgen varios planes alternativos de solución, es necesario considerar una función de optimización o métrica del problema con el fin de seleccionar la mejor alternativa posible. Para tal fin se ha modificado el algoritmo de expansión de RPG para asociar a los diferentes niveles del grafo, los costos estimados de acuerdo con la métrica del problema que el planificador debe tratar de optimizar (minimizar). La propuesta en INDIGO, consiste en calcular el costo de las acciones en el estado actual S_0. La principal diferencia con el RPG tradicional, es que los niveles del grafo no representan el número de acciones aplicadas sino, los costos relativos a la métrica del problema (m). El RPG expande por orden de costo todas las posibles acciones hasta alcanzar los objetivos. Por ello, para calcular los niveles del grafo se debe estimar el costo de ejecutar una acción a:

$$cost(a) = evalue(m, result(a, S_0, 0))\text{-}evalue(m, S_0) + \varepsilon \quad \textbf{(Ecuación 1)}$$
$$If\ cost(a) < \varepsilon\ then\ cost(a) = \varepsilon\ end\ if$$

El costo de una acción, como se detalla en la Ecuación 1, se calcula como el incremento en el valor de la métrica causado por la ejecución de a en S_0, sin la presencia de factores externos que alteran el entorno. La constante ε, representa que todas las acciones incluso aquellas cuya ejecución no afecta el valor de la métrica tienen un costo; el valor de ε es

un número muy pequeño para evitar afectar significativamente el cálculo.

Finalmente, se comprueba que el costo de la acción sea positivo (de otro modo desordena el grafo) para las acciones que decrementan el valor de la métrica, se les asigna el costo mínimo de la constante ε. La expansión del grafo se realiza en orden incremental de costo, en el que cada nivel de acción A_j, de una forma estimada, cuestan más de alcanzar que las de los niveles anteriores A_i,($i< j$). Tras las anteriores consideraciones una primera versión del algoritmo de expansión del RPG de la que hace uso el planificador es el mostrado en la Figura 6.

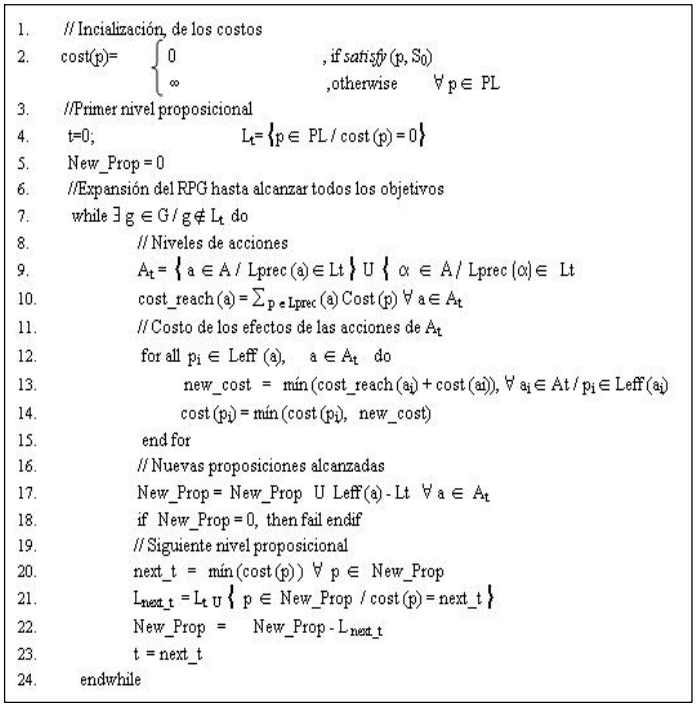
1. // Incialización, de los costos
2. cost(p)= { 0 , if *satisfy* (p, S_0) ; ∞ , otherwise } ∀ p ∈ PL
3. //Primer nivel proposicional
4. t=0; $L_t = \{p \in PL / cost(p) = 0\}$
5. New_Prop = 0
6. //Expansión del RPG hasta alcanzar todos los objetivos
7. while ∃ g ∈ G / g ∉ L_t do
8. // Niveles de acciones
9. $A_t = \{ a \in A / Lprec(a) \in Lt \} \cup \{ \alpha \in A / Lprec(\alpha) \in Lt$
10. cost_reach (a) = $\sum_{p \in Lprec(a)}$ Cost (p) ∀ a ∈ A_t
11. // Costo de los efectos de las acciones de A_t
12. for all $p_i \in$ Leff (a), a ∈ A_t do
13. new_cost = mín (cost_reach (a_j) + cost (a_i)), ∀ $a_i \in At / p_i \in Leff(a_i)$
14. cost (p_i) = mín (cost (p_i), new_cost)
15. end for
16. // Nuevas proposiciones alcanzadas
17. New_Prop = New_Prop U Leff(a) - Lt ∀ a ∈ A_t
18. if New_Prop = 0, then fail endif
19. // Siguiente nivel proposicional
20. next_t = mín (cost (p)) ∀ p ∈ New_Prop
21. $L_{next\_t} = L_t \cup \{ p \in New\_Prop / cost(p) = next\_t \}$
22. New_Prop = New_Prop - L_{next_t}
23. t = next_t
24. endwhile

Figura 6. Algoritmo de expansión del RPG considerando la incertidumbre en los valores lógicos de las proposiciones

Por último, un aspecto importante a tener en cuenta en este punto, es que en INDIGO el cálculo de los costos en el RPG, se realiza sobre un estado de partida determinado, por lo que cualquier cambio en dicho estado disminuye la calidad de las estimaciones, lo cual requiere recalcular un nuevo RPG, cada vez que se produzca un cambio de estado (planificación incremental).

3) Manejo de las acciones de sensado

Uno de los principales objetivos del planificador en INDIGO, es trabajar con información incompleta del entorno, lo cual requiere de mecanismos de sensado, que en el contexto de los servicios Web de INDIGO representan servicios que brindan información y tienen como efecto el predicado "*agentKnows(X)*". Para tener en cuenta este aspecto, se modificó la versión del algoritmo de expansión del RPG mostrado en la Figura 6. Para realizar esto, es necesario distinguir las acciones normales de las acciones de sensado, para lo cual se utilizará la nomenclatura, *a* y α respectivamente.

Así, el primer paso para calcular un RPG que tenga en cuenta las acciones de sensado es realizar una primera expansión mediante el algoritmo presentado en la Figura 4. Este algoritmo ya incluye las acciones de sensado en el grafo, pero no añade sus efectos en los niveles proposicionales. De este modo, el RPG obtenido solo incluirá aquellas proposiciones que sean alcanzables mediante la ejecución de acciones normales, es decir sin recurrir a las acciones de sensado.

Si en esta primera expansión se alcanzan todos los objetivos proposicionales del problema, entonces no es necesario adquirir nueva información a través de las acciones de sensado. En caso contrario, es necesario incluir los posibles efectos de las acciones de sensado en el grafo. Para realizar esto se lleva a cabo una segunda expansión del RPG mediante el algoritmo detallado en la Figura 7. El objetivo de esta segunda expansión es el de obtener, en un orden incremental de costo, toda la información necesaria para satisfacer los objetivos.

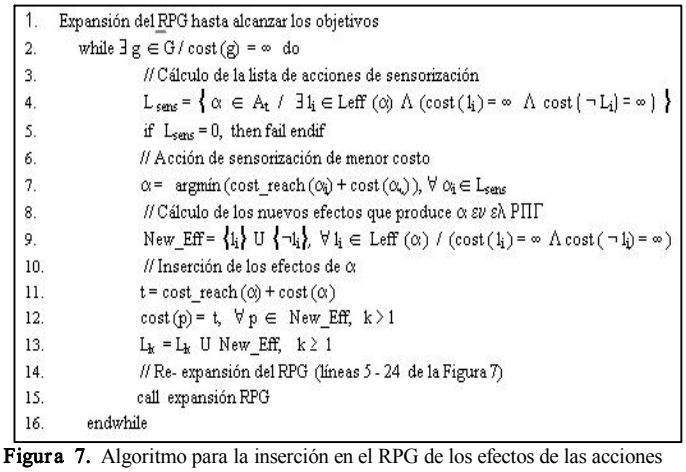
1. Expansión del RPG hasta alcanzar los objetivos
2. while ∃ g ∈ G / cost (g) = ∞ do
3. // Cálculo de la lista de acciones de sensorización
4. $L_{sens} = \{ \alpha \in A_t / \exists l_i \in Leff(\alpha) \wedge (cost(l_i) = \infty \wedge cost(\neg L_i) = \infty) \}$
5. if L_{sens} = 0, then fail endif
6. // Acción de sensorización de menor costo
7. α = argmín (cost_reach (α_i) + cost (α_i)), ∀ $\alpha_i \in L_{sens}$
8. // Cálculo de los nuevos efectos que produce α εν εΛ ΡΠΓ
9. New_Eff = $\{l_i\} \cup \{\neg l_i\}$, ∀ $l_i \in$ Leff (α) / (cost (l_i) = ∞ Λ cost ($\neg l_i$) = ∞)
10. // Inserción de los efectos de α
11. t = cost_reach (α) + cost (α)
12. cost (p) = t, ∀ p ∈ New_Eff, k > 1
13. $L_k = L_k \cup$ New_Eff, k ≥ 1
14. // Re- expansión del RPG (líneas 5 - 24 de la Figura 7)
15. call expansión RPG
16. endwhile

Figura 7. Algoritmo para la inserción en el RPG de los efectos de las acciones de sensado

En esta segunda expansión, dado que los efectos de una acción de sensado son un conjunto de literales cuyo valor de verdad se desconoce, si $l \in L$ es un efecto de una acción de sensado α, en el grafo se deberá incluir tanto a *l* como a $\neg l$. En primer lugar se obtiene una lista con todas las acciones de sensado insertadas en la primera expansión L_{senst} (proposiciones cuyo valor de verdad es desconocido), los efectos de estas acciones de sensado se van añadiendo por orden de costo, por lo que en cada iteración se seleccionará de la acción de sensado α cuyos efectos tengan menor costo. Todos los efectos que puede producir α de se insertan en el nivel correspondiente L_t (donde *t* es el costo de los efectos de α), sin embargo L_t, no es el último nivel del grafo por lo que sus efectos se propagan a niveles de mayor costo. Finalmente el RPG vuelve a expandirse para insertar las acciones y proposiciones que ahora si son alcanzables. Si después de este proceso, algún objetivo del problema resulta inalcanzable, se asume que no existe solución.

D. Etapa del cálculo de planes mono-objetivos

Con el fin de que el planificador de INDIGO sea capaz de trabajar con planes incompletos para atender de manera incremental las solicitudes de actividades por parte del ejecutor en una arquitectura de planificación y ejecución concurrente, en este planificador se usa la técnica de descomposición de objetivos, donde el algoritmo calcula de forma concurrente un plan P_i por separado para cada uno de los objetivos proposicionales del problema $gi \in G$. El cálculo de cada plan P_i se realiza de forma incremental: se construye un plan inicial, posiblemente incompleto (plan esqueleto) y se va refinando con el tiempo, esto permite la interrupción del proceso en cualquier instante y le da al planificador un comportamiento anytime; el refinamiento del plan P_i, termina cuando se alcanza un plan válido, suponiendo que no existen situaciones inesperadas ($\delta=0$).

Los planes P_i se construyen de forma regresiva, sin tener en cuenta las precondiciones numéricas de las acciones. Los planes iniciales tienen la siguiente característica: En el cálculo del plan se garantiza que la primera acción sea ejecutable en el estado actual ($exec(first(P_i), S_0) = V$), por lo que podría ser una posible respuesta del planificador. De hecho, la primera acción que el planificador devolverá al ejecutor, corresponderá con la primera acción de uno o más planes P_i.

Los pasos para el cálculo de cada plan mono-objetivo P_i para alcanzar un objetivo proposicional g_i son: se parte inicialmente de un plan vacío y de un conjunto de sub-objetivos proposicionales *SG*, que contiene únicamente el objetivo que se pretende alcanzar (g_i). Tras esta inicialización, el primer paso es seleccionar una proposición p del conjunto de sub-objetivos *SG* que de acuerdo al RPG calculado tiene asociado un mayor costo, ya que esta permite obtener planes más sencillos de reparar. Una vez, que se escoge el siguiente sub-objetivo a resolver *p*, se escoge una acción que permita alcanzar *p*, es decir que p se encuentre en sus efectos. Esta acción se inserta al principio del plan P_i. Entre todo el conjunto de acciones que producen *p*, se selecciona la mejor evaluada, de acuerdo a un conjunto de criterios asociados a una función de evaluación el cual calcula la bondad de incluir cada acción en el plan P_i.

El proceso anterior continúa de forma iterativa con lo cual el plan P_i se va construyendo regresivamente. Durante este proceso, el conjunto de sub-objetivos *SG* se sustituye entonces por el conjunto de precondiciones proposicionales de cada acción, excepto por aquellas que ya se satisfacen en el estado actual S_0. En el momento en que *SG* queda vacío, termina la construcción del plan inicial.

Luego del cálculo de cada P_i, se procede a una fase de refinamiento la cual se realiza mientras el ejecutor no requiera de una acción o reporte una situación inesperada. Esta fase se compone de dos etapas: la validación del plan P_i y la selección de un conflicto y su posterior solución. Esta fase se repite tantas veces hasta que un plan sea válido.

En la validación se verifica la validez de la secuencia de acciones, o lo que es lo mismo comprobar si P_i es un plan válido y alcanza el objetivo. El problema surge en el hecho de que muchas veces un sub-plan P_i, puede resultar no válido, ya sea porque no alcanza el objetivo g_i o porque alguna precondición de una de sus acciones no ha sido satisfecha. En ambos casos existen dos posibles causas que son necesario diferenciar: a) existe una acción que falla (a_{fail}) por que tiene precondiciones proposicionales que no se satisfacen y b) existe una acción que falla (a_{fail}) que tiene únicamente precondiciones numéricas no satisfechas.

Para proceder a explicar sus respectivas soluciones se supondrá que P_i tiene una acción ficticia a_{n+1}, sin efectos y con la proposición g_i como su única precondición. Adicionalmente se define S_j como el estado sobre el cual se aplica cada una de las acciones a_j del plan P_i (ver Figura 8a).

Si P_i, no es válido, se selecciona una precondición no satisfecha de la primera acción no ejecutable a_{fail} y se procede según el caso. En el primer caso, es decir cuando una acción a_{fail} con precondiciones proposicionales que no se satisfacen, ocurre, se escoge de todas estas precondiciones, aquella con el mayor costo (de acuerdo con el RPG) y se lleva a cabo el siguiente proceso:

(a) alcanzar un estado en el que se satisfaga la proposición que falla (p_{fail}). Para ello se calcula un nuevo plan P_{ij} desde cada estado anterior a a_{fail} que satisfaga p_{fail} (ver Figura 8b);

(b) desde los estados alcanzados que satisfacen p_{fail} se intenta alcanzar un estado anterior a a_{fail} (ver Figura 8c);

Y finalmente, se selecciona la mejor alternativa estudiando el número de precondiciones no satisfechas presentes en cada una de las alternativas. Al igual que el número de precondiciones no satisfecha, se escoge la alternativa de menor costo, es decir aquella en la que la suma de los costos de las acciones es menor.

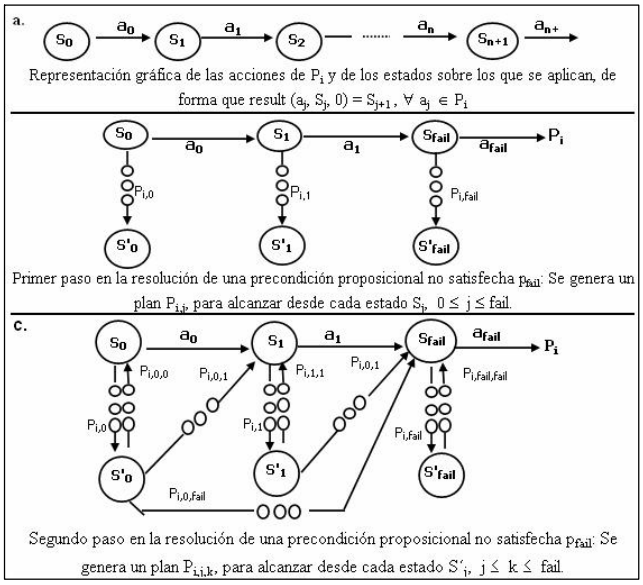

Figura 8. Representación de las acciones de Pi y de los estados en la fase de refinamiento

En el segundo caso, es decir si a_{fail} tiene solo precondiciones numéricas no satisfechas, la solución es un tanto compleja, dado que no es fácil descubrir que acciones permiten alcanzar un estado en el que dicha acción se cumpla y puede hacer falta calcular toda la secuencia de acciones que modifique los valores de las variables para que la restricción numérica se cumpla. Esto requiere incluir un proceso de exploración y búsqueda para explorar las distintas alternativas (hasta ahora no requerido). En INDIGO esta exploración se aprovecha no solo para resolver las precondiciones numéricas de a_{fail}, sino también para encontrar un P_i válido; La búsqueda para encontrar un plan P_i válido se realiza en un espacio de estados siguiendo la estrategia de búsqueda del Primero el Mejor [12]. El estado de partida se corresponde con el estado actual, S_o, y los sucesores (o hijos) que corresponden a un nodo n, son los estados resultantes de aplicar las acciones ejecutables sobre n. el objetivo de esta búsqueda se alcanza cuando se encuentra un plan válido que consigue g_i.

E. Ordenación de planes

En esta etapa es posible encontrar varios planes P_i que no sean totalmente ejecutables para cada uno de sus respectivos objetivos g_i del problema debido a la interacción existente entre todos los planes P_i. Cuando el ejecutor requiere de una acción, la etapa de refinamiento se detiene. La acción que se devuelve al ejecutor es la primera acción de uno ó más de los planes P_i. En esta etapa con el fin de encontrar cual plan debe ser ejecutado primero, se establecen unos criterios para excluir aquellos planes que no deben iniciar su ejecución aún:

Inversas: Un plan es excluido cuando comienza con una acción que es inversa a la última acción ejecutada.

Compartir secuencias de acciones: si se consideran dos planes P_i ($\{a_{i0}, a_{i1}, \ldots, a_{in}\}$) y P_j ($\{a_{j0}, a_{j1}, \ldots, a_{jm}\}$), un plan P_i es ordenado antes que P_j, si P_i incluye la primera acción del plan P_j ($a_{ik} = a_{j0}$), y la secuencia de acciones $\{a_{i0}, \ldots, a_{ik}, a_{j1}, \ldots, a_{jn}\}$ es ejecutable. En otras palabras, el plan P_i puede empezar su ejecución sin afectar la ejecución de P_j.

Conflictos no flexibles: Suponiendo que se den dos planes P_i y P_j, donde ambos tienen una acción que necesita y borra un literal l. Esto es del tipo orden no flexible, ya que no es posible ordenar estas acciones a menos que exista una acción adicional que restaure l, y que sea insertada entre ellas. Si esta acción adicional existe en solo uno de estos planes (por ejemplo P_j), entonces P_j es ordenado antes que P_i y por lo tanto P_i es ordenado después de P_j.

Intercambios de la primera acción: Si la primera acción de un plan P_j puede ser remplazada por la primera acción de un plan P_i, sin causar conflictos, entonces P_i es ordenado antes que P_j.

Conflictos flexibles: Suponiendo que la primera acción P_i produce el literal l, y l no se suprime a lo largo del plan. Si la primera acción del plan P_j necesita y también borra el literal l, entonces P_i se ordena antes que P_j.

Después de aplicar los respectivos procesos que evalúan los anteriores criterios, el planificador selecciona uno de los planes que no ha sido excluido. La primera acción a_{next} del plan seleccionado es enviada al ejecutor. El planificador entonces actualiza sus creencias con los efectos de a_{next}. El resto de los planes son descartados y un nuevo plan inicial es calculado nuevamente. El proceso de planificación se repite hasta que todos los objetivos son alcanzados.

V. Trabajos Relacionados

Una serie de trabajos orientados a la composición de servicios Web utilizando diferentes técnicas de la planificación IA existen en la actualidad. Entre los más destacados se encuentran los siguientes.

Un planificador para la composición de servicios DAML-S basado en la lógica, se desarrollo en la UMBC, USA [13]. Este planificador usa servicios bajo un estilo STRIPS para componer un plan, dándole el objetivo y un conjunto de servicios. Este se implementa en JESS (Java Expert System Shell), y usa un conjunto de reglas para trasladar las descripciones DAML-S a servicios atómicos en operaciones de planificación. Este trabajo se diferencia con INDIGO por que realiza la planificación *off-line*, es decir realiza el proceso de planificación de la composición del servicio sin tener en cuenta su ejecución, retornando solamente el plan de la composición. A diferencia, en INDIGO se busca tener en cuenta lo que pasa durante la ejecución del plan a medida que se realiza la planificación de dicha composición. Adicionalmente, en el trabajo de la UMBC, es necesario tener conocido todas las características del entorno antes de realizar el proceso de planificación, mientas que INDIGO permite insertar información adicional del entorno durante la planificación. Por último, en este trabajo no se tiene en cuenta las restricciones de tiempo, las cuales si se consideran en INDIGO.

Una de las aplicaciones más conocidas para llevar a cabo la composición de servicios es la planteada por la Universidad de Maryland, USA [18], en la cual, utilizando una técnica de planificación HTN [3] mediante el uso del planificador SHOP2 [8], se lleva a cabo la composición de servicios Web descritos en el OWL-S. Los autores proveen la correspondencia entre la semántica del SHOP2 y la semántica del cálculo de situaciones del modelo de procesos del OWL-S. EL SHOP2 genera planes correctos y completos sobre un conjunto de descripciones OWL-S y tratan la salida de un servicio como efectos que cambian, ya sea el estado del conocimiento del agente de planificación o el estado del mundo. Durante la planificación, los servicios Web no son

ejecutados, evitando afectar el estado del mundo. Este trabajo contrasta con INDIGO, dado en este último, si se realiza la ejecución de servicios Web en tiempo de planificación, actualizando constantemente la base de conocimiento del agente de planificación. Otra diferencia que presenta INDIGO, es su característica anytime, con lo cual, permite tener restricciones de tiempo en la búsqueda de su solución, mientras que el SHOP2 no permite esta característica.

Otro de los trabajos en los que se aplica las técnicas de planificación es el presentado por la universidad de St. Gallen [11], en el cual se utiliza una técnica de Planificación de Orden Parcial, utilizando el planificador VHPOP [19] para llevar a cabo la composición de servicios Web. En este utilizan su propio lenguaje de marcado semántico de los servicios Web llamado SESMA [10] tanto para especificar los servicios a utilizar como para especificar el problema de composición, especificaciones que se trasladan a sus respectivas representaciones PDDL, con el fin de utilizar el planificador. En este trabajo se enfrentan los problemas de la información incompleta mediante el uso planes condicionales que son valorados luego durante la etapa de ejecución insertando en el sistema los elementos de información obtenidos al ejecutar los respectivos servicios de información. En este trabajo se tiene en cuenta que el mundo es dinámico para lo cual se verifica que en el tiempo de ejecución se lleve a cabo correctamente el plan inicialmente planteado, y en caso de que falle, se realiza un proceso de re-planificación. Este trabajo tiene sus semejanzas con INDIGO en que verifican que el plan desarrollado si sea correcto corrigiéndolo en caso de que falle, pero se diferencian en la forma en que enfrentan este problema ya que en este trabajo se realiza primero el plan y luego se ejecuta, mientras que en INDIGO se lleva a cabo de manera concurrente el plan y su respectiva ejecución, corrigiendo cada acción que falle inmediatamente se ha calculado, con lo cual los tiempos de planificación y ejecución pueden disminuir. INDIGO se asemeja a este trabajo, en cuanto a que ambos enfrentan el problema del conocimiento incompleto del entorno de la Web.

VI. Conclusiones y trabajo futuro

En este trabajo de ha presentado un sistema de planificación para la composición de servicios Web semánticos especificados en OWL-S, llamado INDIGO, el cual permite de manera concurrente la composición y ejecución de servicios Web al utilizar un convertidor de especificaciones OWL-S a PDDL y un planificador incremental que combina técnicas de planificación anytime con técnicas de planificación heurísticas.

De manera detallada se presentaron las diferentes características del corazón del sistema INDIGO, que es su planificador. En este planificador se plantean algunas características, no muy comunes en los planificadores en Inteligencia Artificial, que le permiten trabajar en el entorno de la Web, dado que soporta acciones de sensado, realiza la planificación con restricciones de tiempo, hace uso de funciones numéricas, considera métricas para optimizar un problema, maneja la incertidumbre del entorno de la Web relacionada con el conocimiento parcial de su entorno y su comportamiento altamente dinámico.

Actualmente, se realiza una versión inicial en JAVA de este sistema propuesto. Paralelamente se está implementando un ambiente de prueba en el dominio del Satélite extraído de la competición de la planificación IA [6], el cual consta de servicios Web reales y sus respectivas marcaciones en OWL-S.

VII. Reconocimientos

El presente trabajo está apoyado parcialmente el proyecto de la Convocatoria Nacional – 20101006096, “Modelo Multi-Agente de Planificación Bajo Conocimiento Incompleto para la Composición Automática de Servicios Web” asociado al grupo de investigación Sistemas Inteligentes Web “SINTELWEB” y el proyecto de investigación de la Tesis de Doctorado “Modelo Multi-Agente de Planificación y Ejecución Concurrente para la Composición de Servicios Web Semánticos en Entornos Parcialmente Observables”, auspiciada por Colciencias, Universidad Nacional de Colombia, Sede Medellín y el Banco Mundial, enmarcado en el programa de apoyo a la comunidad científica nacional en programas de Doctorado 2004.

REFERENCIAS

[1] A. Blum y M. Furst, Fast planning through planning graph analysis. Artificial Intelligence, 1997, 90:281-300.

[2] M. Do y S. Kambhampati. Sapa : A multiobjective metric temporal planner. Journal of Artificial Intelligence Research (JAIR), 2003, 20:155-194.

[3] K. Erol, J. Hendler., y D. Nau Semantics for Hierarchical task network planning, 1994.

[4] M. Ghallab, D. Nau y P. Traverso. Automated Task Planning. Theory and Practice. Morgan Kaufmann, 2004.

[5] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld y D. Wilkins. PDDL—The Planning Domain Definition Language, 1998.

[6] International Planning Competition IPC. http://planning.cis.strath.ac.uk /competition/2002.

[7] J. Hoffman. Local search topology in planning benchmarks: A theorical analysis. Proceedings of the 6th. International Conference on Artificial Intelligence Planning and Scheduling (AIPS), 2002, pp. 379-387,

[8] D. Nau, H. Muñoz Avila, Y. Cao, A. Lotem, and S. Mitchell. Total-order planning with partially ordered subtasks. In proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, 2004.

[9] OWL-S. Semantic Markup for Web Services. http://www.w3.org/Submission/OWL-S/. última visita septiembre de 2007.

[10] J. Peer, Semantic Service Markup with SESMA. Language Specification, version 0.8, http://elektra.mcm.unisg.ch/pbwsc/docs/sesma 0.8.pdf , 2004.

[11] J. Peer, A POP-based Replanning Agent for Automatic Web Service Composition, Second European Semantic Web Conference (ESWC'05), 2005

[12] Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Upper Saddle River, NJ. 1995.

[13] M Sheshagiri, M. Desjardins y T. Finin. A planner for composing services described in DAML-S. Proceedings of AAMAS. Workshop on web service and agent-based engineering.2003.
[14] R. Washintong. Incremental planning for truly integrated planning and reaction. Proceedings of the Scandinavian Conference on AI (SCAI), 1995, 28: 305-316.
[15] Web services. HP, Web Services Concepts: a Technical Overview, HP Document. http://www.bluestone.com/downloads/pdf/web services tech_overview.pdf, 2001.
[16] WSDL-S. Web Service Semantics WSDL-S, http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.pdf. Última visita febrero de 2007.
[17] WSMO. http://www.wsmo.org/2004/d2/v1.0/. Última visita septiembre de 2007.
[18] D. Wu, B Parsia, E. Sirin, J Hendler y D. Nau. Automating DAML-S web services composition using SHOP2. Proceedings of the 2nd International Semantic Web Conference (ISWC2003), Sanibel Island , Florida USA, 2003, pp. 20-23
[19] H. L. Younes, y R. Simmons. VHPOP: Versatile Heuristic Partial Order Planner. Journal of Artificial Intelligence Research, 2003.

Jaime Alberto Guzmán Luna. Ingeniero Civil (1990) y Magíster en Ingeniería de Sistemas (1997) de la Universidad Nacional de Colombia, sede Medellín. Es especialista en Comunicación Educativa de la Universidad de Pamplona, Colombia en el año 2000 y a la fecha adelanta estudios de doctorado en Ingeniería en la Universidad Nacional de Colombia, sede Medellín.

Actualmente es Profesor Asistente en la Universidad Nacional de Colombia, sede Medellín donde es el director del grupo de Investigación SINTELWEB: Sistemas Inteligentes en la Web, registrado en Colciencias. Imparte los cursos de Objetos y Sistemas de Recuperación de Información en la Web en el programa de Ingeniería de Sistemas y en la Maestría en Ingeniería de Sistemas. En años anteriores el se ha desempeñado como docente en otras universidades de Colombia como son la Universidad Distrital Francisco José de Caldas, en Bogotá y en la Universidad de Pamplona, Pamplona Norte de Santander. El es autor de varios artículos relacionados con el tema de la recuperación de Información en la Web y los servicios Web semánticos.

Demetrio Arturo Ovalle Carranza. Profesor Asociado, Universidad Nacional de Colombia sede Medellín. Director de la Escuela de Ingeniería de Sistemas de la Universidad Nacional de Colombia – Sede Medellín. Director del GIDIA: Grupo de Investigación y Desarrollo en Inteligencia Artificial, Categoría A de Colciencias. Ingeniero de Sistemas y Computación, Universidad de los Andes, Bogotá, Colombia (1984). Magíster en Informática del Institut National Polytechnique de Grenoble, Francia (1987). Doctor en Informática de la Université Joseph Fourier, Francia (1991).

El área de énfasis de su investigación es Inteligencia Artificial, más específicamente Sistemas Híbridos Inteligentes integrando Redes Neuronales, Sistemas de Lógica Difusa y Sistemas Multi-Agente aplicados a la Simulación de los Mercados de Energía y a la Detección de Fallas en Líneas de Transmisión. Otros tópicos de investigación que trabaja actualmente son: Inteligencia Artificial en Educación, Sistemas Tutoriales Inteligentes, Sistemas basados en CBR (Case-Based Reasoning) y Técnicas de Planificación Inteligente aplicadas a la Construcción de Sistemas de Composición de Servicios Web.