novática
CEPIS UPGRADE Revista de la Asociación de Técnicos de Informática

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software). *Novática* co-edita asimismo **UP**GRADE, revista digital de **CEPIS** (*Council of European Professional Informatics Societies*), en lengua inglesa, y es miembro fundador de **UP**ENET (**UP**GRADE *E**uropean** **NET**work*).

<http://www.ati.es/novatica/>
<http://www.ati.es/reicis/>
<http://www.upgrade-cepis.org/>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y es representante de España en **IFIP** (*International Federation for Information Processing*); tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*), así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2, ASTIC, RITSI** e **Hispalinux**, junto a la que participa en **ProInnova**.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn./fax 963330392 <secreval@ati.es>
Administración y Redacción ATI Cataluña
Via Laietana 46, ppal. 1ª, 08003 Barcelona
Tlfn.934125235; fax.934127713 <secregen@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Andalucía <secreand@ati.es>
Redacción ATI Galicia <secregal@ati.es>
Suscripción y Ventas <http://www.ati.es/novatica/interes.html>, ATI Cataluña, ATI Madrid
Publicidad
Padilla 66, 3º, dcha., 28006 Madrid
Tlnf.914029391; fax.913093685 <novatica@ati.es>
Imprenta: Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: La mirada circular - Concha Arias Pérez / © ATI
Diseño: Fernando Agresta / © ATI 2003

Nº 208, noviembre-diciembre 2010, año XXXVI

sumario

Monografía del próximo número: "*Internet de las cosas*"

Orinoco Framework: publicacion, composicion y ejecucion de Servicios Web en ambientes GRID

Keysis Kiss[1], Eduardo Blanco[2], Yudith Cardinale[2]
[1]Escuela de Computación, Facultad de Ciencias, Universidad Central de Venezuela; [2]Departamento de Computación y Tecnología de la Información, Universidad Simón Bolívar, Venezuela

<kkiss@gmail.com>,
<{eduardo,yudith}@ldc.usb.ve>

Este artículo ha sido seleccionado de entre las mejores ponencias presentadas en la XXXVI Conferencia Latinoamericana de Informática (XXXVI CLEI), evento anual promovido por el Centro Latinoamericano de Estudios en Informática (CLEI), entidad en la cual ATI es el representante de España. Para más información ver la página 5 de este mismo número.

Resumen: *la gran cantidad de servicios existentes en plataformas Grid y la necesidad de satisfacer requerimientos complejos, requieren de la integración de los recursos de distintas organizaciones. Esta integración implica procesos de registro de servicios, usando lenguajes para la descripción semántica de funcionalidades, y de procesos de descubrimiento, composición y ejecución semiautomáticos. Este artículo presenta Orinoco, una infraestructura para ambientes Grid que integra las facilidades de publicación, descubrimiento, composición y ejecución de Servicios Web (SWs). Orinoco permite transformar aplicaciones Java en SWs, generando las descripciones semánticas correspondientes. También asiste a los usuarios resolviendo requerimientos que impliquen la coordinación de servicios. Ante un requerimiento, Orinoco genera un plan de ejecución que al ser evaluado dará respuesta al requerimiento. Se demuestran las funcionalidades de Orinoco con un caso de estudio y se presentan resultados de rendimiento sobre gLite, un middleware de Grid.*

Palabras clave*: composición de Servicios Web, computación Grid, planes de ejecución, registro y publicación de Servicios Web, Web Semántica.*

1. Introduccion

El crecimiento de la computación orientada a servicios ha permitido la creación de aplicaciones integradas por componentes que interactúan entre sí [1]. La idea de explotar recursos de diferentes instituciones para resolver problemas científicos y de negocios está promoviendo la creación de nuevos modelos de negocios para proveedores de servicios en ambientes distribuidos como el *Grid* [2]. Por un lado, dado el auge de los Servicios Web (*SWs*), las organizaciones buscan presentar sus funcionalidades a través de esta tecnología. No obstante, conforme aumenta la cantidad de servicios publicados en diferentes localidades geográficas e implementados en distintos lenguajes, se hace más difícil la tarea de encontrar servicios que satisfagan los requerimientos de los usuarios [3]. Por otro lado, la Web Semántica define y enlaza datos en la Web de forma que puedan ser entendidos por un computador para visualizarlos, automatizar tareas, integrar y reutilizar datos entre aplicaciones [4]. De la unión de estas dos tecnologías surgen los Servicios Web Semánticos, que extienden las tecnologías de los *SWs* con ontologías que facilitan la selección, integración e invocación dinámica de servicios. Una de las iniciativas más exitosas, orientada a la descripción semántica de servicios, es *OWL-S* [5]. Ésta propone una ontología para la descripción de la semántica de servicios; sin embargo, esto no ha dado lugar a una implementación de las plataformas y motores necesarios para la ejecución de los servicios.

Compartir e intercambiar Servicios Web Semánticos involucra varios procesos: publicación, registro, descubrimiento, composición y evaluación. Primero se debe implementar el servicio y describirlo de acuerdo a sus capacidades, que pueden estar expresadas a través de palabras claves, funcionalidades, parámetros de entrada/salida, entre otros. Luego el servicio, junto con su descripción, debe ser publicado en un servidor disponible para la comunidad con la que se quiera compartir. Una vez publicados, los servicios requieren ser descubiertos por los usuarios que los necesiten. Para requerimientos complejos estos servicios podrían ser coordinados para poder generar un plan de ejecución que al ser evaluado produzca la respuesta solicitada. La evaluación del plan implica invocar cada servicio que lo compone en el orden apropiado y con los parámetros adecuados.

Este trabajo presenta *Orinoco Framework*[1] una infraestructura que integra las facilidades de publicación, registro, composición y ejecución de *SWs* en plataformas *Grid*. Orinoco permite transformar aplicaciones JAVA en *SWs* y generar información semántica asociada al servicio y sus parámetros, creando un repositorio de datos de Servicios Web Semánticos. Orinoco resuelve los requerimientos de usuarios que impliquen la interacción coordinada de diversos servicios, generando como resultado un plan de ejecución. Al evaluar todos los servicios en un plan, Orinoco logra dar respuesta a la consulta del usuario. Se demuestran las funcionalidades de Orinoco a través de un caso de estudio y se presentan resultados de rendimiento sobre *gLite* [6], un *middleware* de *Grid*.

2. Trabajos relacionados

Las tareas de publicación, descubrimiento, composición y ejecución de *SWs* requieren de técnicas, herramientas y enfoques particulares para llevarlas a cabo de manera sencilla y de ser posible semiautomática. Esta sección presenta una revisión de trabajos relacionados con cada una de estas tareas.

Los estándares definidos para los *SWs* [7] señalan cómo se deben publicar para que sean accedidos. Establecen cómo se debe realizar la invocación y cómo los resultados deben ser enviados. Ninguno de estos estándares define la forma en la que deben implementarse los servicios, es decir, los desarrolladores están en libertad de elegir el lenguaje de programación que deseen utilizar. La invocación de *SWs*, así como la transferencia de datos, está basada principalmente en los siguientes estándares: SOAP, WSDL, UDDI [8]. Su utilización conjunta permite a las aplicaciones interactuar siguiendo un modelo débilmente acoplado que es independiente de las plataformas subyacentes.

La Web Semántica usa lenguajes, como el lenguaje estructurado XML (*Extensible Markup Language*) y el lenguaje RDF (*Resource Description Framework*), para dotar a los recursos de un significado. Esto permite a los computadores poder procesar a un nivel conceptual la información y los procesos, logrando que esta información pueda ser integrada y reutilizada en forma automática. Siguiendo esta misma idea, la *Grid* Semántica considera describir semánticamente al *Grid* en función de sus recursos: los datos y los procesos disponibles. En este sentido se han realizado algunos trabajos que consideran el uso de tecnologías de la Web Semántica dentro de la infraestructura *Grid*, en sistemas multi-agentes, y a través de Servicios Web Semánticos [9][10][11][12][13][14].

Una vez que un SW es desarrollado, es importante registrarlo para que pueda ser descubierto y usado. La publicación de un Servicio

Web Semántico implica registrar el SW junto con una descripción semántica de sus capacidades. CODE [15] es una herramienta que apoya el proceso de publicación de Servicios Web Semánticos. Se implementa como un *plugin* para Eclipse (ambiente de desarrollo integrado - IDE). CODE provee un editor para el desarrollo de las descripciones semánticas usando la ontología OWL-S y permite la publicación y registro de los *SWs* usando el estándar UDDI. El Agente Publicador de Orinoco es inspirado en CODE, pero considera más facilidades para la generación automática de *wrappers* y archivos de configuración.

El descubrimiento de servicios se refiere a identificar, en base a la descripción de sus capacidades, los *SWs* que satisfagan los requerimientos de una consulta. Los primeros algoritmos semánticos de descubrimiento de servicios se basan en la información publicada en el perfil del servicio para identificar coincidencias de funcionalidad en términos de entradas, salidas, pre-condiciones y efectos [16][17][18]. La desventaja principal de estos algoritmos está en la limitación para describir las funcionalidades y las estructuras de control usadas en la implementación del servicio; por lo que es posible seleccionar uno incorrectamente (dos servicios pueden tener las mismas entradas y salidas pero con funcionalidades totalmente diferentes). En [19][20], se presentan propuestas de descubrimiento de servicios que consideran la información del perfil del servicio y la información especificada en el modelo de proceso. Sin embargo, estas propuestas no son capaces de identificar una solución que requiera la participación de más de un servicio.

La composición de *SWs* implica un mecanismo que permita identificar un conjunto de dos o más servicios que puedan cooperar entre sí para resolver requerimientos que van más allá del alcance de sus capacidades individuales [21][22][23]. En este sentido, han sido propuestos diferentes enfoques o simplificaciones del problema que usan técnicas de diferentes áreas: inteligencia artificial (IA) [24][25], algoritmos de recorrido en grafos [26][27][28][29], entre otros.

En el *Grid*, existen algunas propuestas que estudian el problema de optimización, ya sea en el propio proceso de composición o para mantener el costo estimado de evaluación de un plan generado [9][30][31][32][33]. ARGUGRID [9] proporciona un nuevo modelo para un ambiente de desarrollo en el *Grid* a un nivel semántico. Los agentes se asocian a los solicitantes de servicios/recursos y a los proveedores de servicios/recursos dentro de una arquitectura orientada a servicios. La tecnología de argumentación se utiliza para apoyar la toma de decisión racional así como la negociación, entre los agentes, requeridos para facilitar la composición transparente de servicios/recursos en planes ejecutables [34].

Orinoco *Framework* integra, en una única herramienta, diferentes enfoques y tecnologías para asistir en todas las etapas de composición y ejecución de *SWs*.

3. Arquitectura de Orinoco Framework

Orinoco *Framework* está conformado por 4 agentes que ejecutan funcionalidades específicas: Publicador, Registrador OWL-S, Compositor y Evaluador. En la **figura 1** se presenta una visión completa de la arquitectura de Orinoco *Framework* y la interacción entre sus agentes. Cuando un usuario envía al Agente Publicador una solicitud para publicar una aplicación, Orinoco ejecuta esencialmente los siguientes pasos: 1) transforma la aplicación en SW y lo publica en uno de los servidores disponibles; 2) genera la descripción semántica (OWL-S) del *SW* y la publica en el Agente Registrador OWL-S. De esta manera los *SWs* son publicados y registrados para que estén disponibles tanto para otros usuarios como para los Agentes Compositor y Evaluador.

Un usuario que requiera satisfacer un requerimiento complejo puede usar Orinoco enviando una consulta al Agente Compositor quien debe seleccionar los *SWs* adecuados para generar un plan que dé respuesta al requerimiento. El plan generado por el Agente Compositor es entregado al usuario. Luego, el usuario puede solicitar su evaluación por medio del Agente Evaluador. A continuación se explica en detalle la arquitectura y funcionamiento de cada agente.

3.1. Agentes Publicador y Registrador OWL-S

El Agente Publicador es un SW encargado de trasformar una aplicación Java en un SW y de generar su respectiva definición semántica usando el formalismo OWL-S. El Agente Registrador OWL-S administra dichas descripciones semánticas permitiendo que los otros agentes puedan descubrir los servicios disponibles a través de sus capacidades. El Agente Publicador recibe una clase Java que contiene los métodos que se desea publicar y como resultado el *SW* es publicado en los

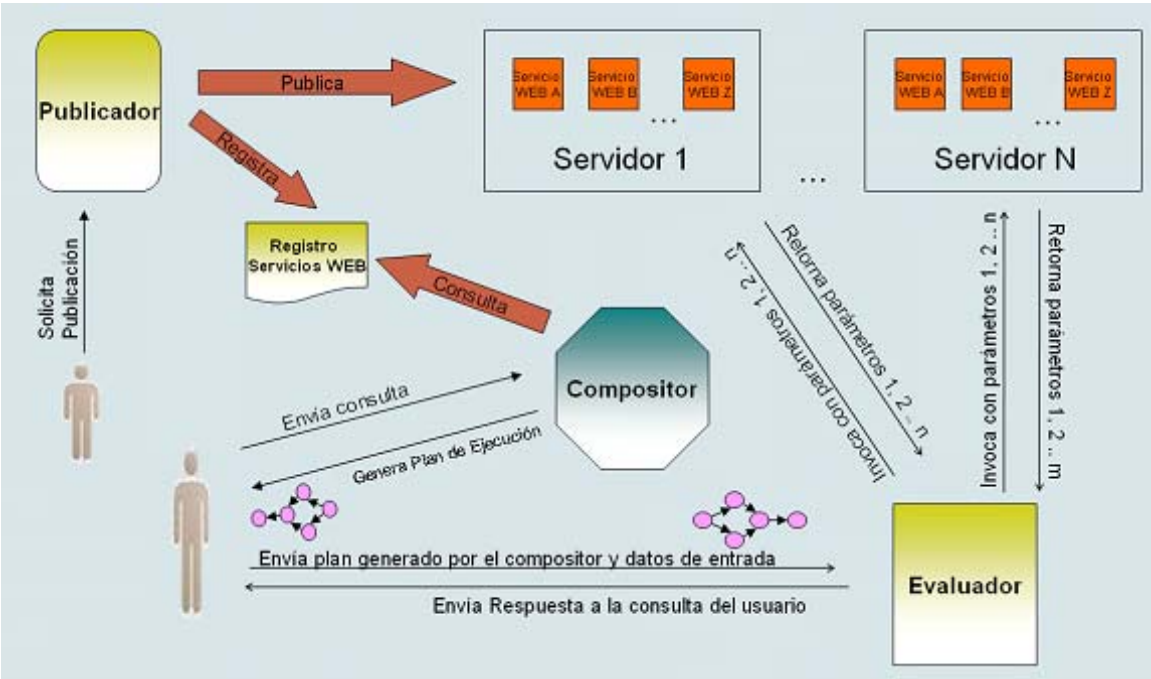

Figura 1. Arquitectura de Orinoco Framework.

servidores *Sun Java System Application Server* seleccionados y luego registrado en el Agente Registrador OWL-S. La interacción del usuario con el Agente Publicador, se realiza a través de una interfaz gráfica (GUI) que facilita las tareas de publicación. En general las funcionalidades que ofrece el Agente Publicador son:

- Inspecciona el *bytecode* de las librerías que el usuario desea publicar con el fin de extraer las clases contenidas, sus métodos y parámetros para generar automáticamente los envoltorios (*wrappers*) que permitirán a la aplicación comportarse como *SW*.
- Permite la carga de conceptos ontológicos indicando las respectivas URLs.
- Asiste al usuario en el despliegue de los *SWs* en los servidores de aplicaciones.
- Asiste en la definición de las capacidades de los servicios usando los conceptos de las ontologías especificadas.
- Genera automáticamente los archivos necesarios para la publicación, incluyendo el archivo .jdl (que representa una tarea computacional, un ejecutable o un *script*) que el usuario enviará al *Grid* para invocar el servicio.

El Agente Publicador crea los *wrappers* para las aplicaciones, a partir de las clases y métodos indicados por el usuario. Los *wrappers* encapsulan las llamadas a los métodos de las aplicaciones Java para que sean accedidos como SW. Luego, empaqueta la aplicación y la publica remotamente en un conjunto de servidores de aplicación. Para la generación de los *wrappers*, se implementó un motor de enlace utilizando JAX para la recepción y envío de los mensajes SOAP usados para la comunicación con el servicio.

Cuando se publica el SW se crea el archivo .wsdl. Adicionalmente, se generan las descripciones semánticas y ontológicas de los servicios a partir del archivo .wsdl previamente generado y de los conceptos indicados por el usuario. Tales descripciones semánticas detallan las propiedades del servicio como su nombre, su descripción e información de contacto sobre los autores, publicadores, etc. Los servicios son descritos funcionalmente en base a las entradas, salidas, pre-condiciones y efectos. También se especifica cómo se interactúa con el servicio llegando a describir para ello el protocolo que lo implementa (RPC, SOAP, CORBA, HTTPFORM, etc.), el formato de los mensajes, serialización, transporte y direccionamiento (máquina y puerto) para ejecutarlo. Las descripciones semánticas se almacenan en un archivo .owls que se publica en el servidor y se registra en el Agente Registrador OWL-S. Como el archivo .wsdl es un XML que describe SW (independiente de la implementación) es posible generar descripciones semánticas de SW ya publicados y que no estén implementados en Java para luego ser registrados en el Registrador OWL-S. Los Agentes Compositor y Evaluador interactúan con el Agente Registrador OWL-S para actualizar la lista de las definiciones OWL-S que éste administra. Se utilizó la OWLS-API [35] que proporciona los mecanismos necesarios para manipular y generar los archivos *OWL-S* a partir de archivos .wsdl.

3.2. Agente Compositor

El Agente Compositor es un SW encargado de seleccionar, a partir de una consulta, un conjunto de servicios y coordinarlos para resolver requerimientos complejos. La consulta contiene los conceptos de entrada y los de salida: los que el usuario puede proveer y los que espera obtener. El Agente Compositor retorna el plan que será evaluado por el Agente Evaluador para resolver el requerimiento.

Una vez que una consulta es recibida, el Agente Compositor interroga al Agente Registrador OWL-S para seleccionar los servicios que podrían ser útiles para responder la consulta y luego debe coordinarlos en forma apropiada para generar la respuesta esperada. Actualmente existen dos estrategias de composición implementadas en el Agente Compositor: *PT-SAM* y *DP-BF* [27][28]. Ambas estrategias extienden SAM [26] y, para construir planes de ejecución eficientes, definen meta-heurísticas, con el fin de considerar simultáneamente requerimientos funcionales (la función que debe realizar la composición) y restricciones no funcionales (valores permitidos para un conjunto de parámetros de *QoS*). Estas estrategias usan un modelo de costo que "agrega" los parámetros de *QoS* para guiar la búsqueda hacia el espacio de composiciones que mejor cumplan con el criterio no-funcional, pero usan técnicas diferentes para seleccionar y coordinar los servicios: 1) *DP-BF:* utiliza una variación de programación dinámica, y 2) *PT-SAM*: utiliza un algoritmo de desdoblamiento usado para redes de Petri. En [27][28] se presentan los detalles de estas estrategias de composición.

Un plan se representa como un grafo bipartito (es decir que contiene dos tipos de nodos, nodos tipo dato y nodos tipo servicio) sin

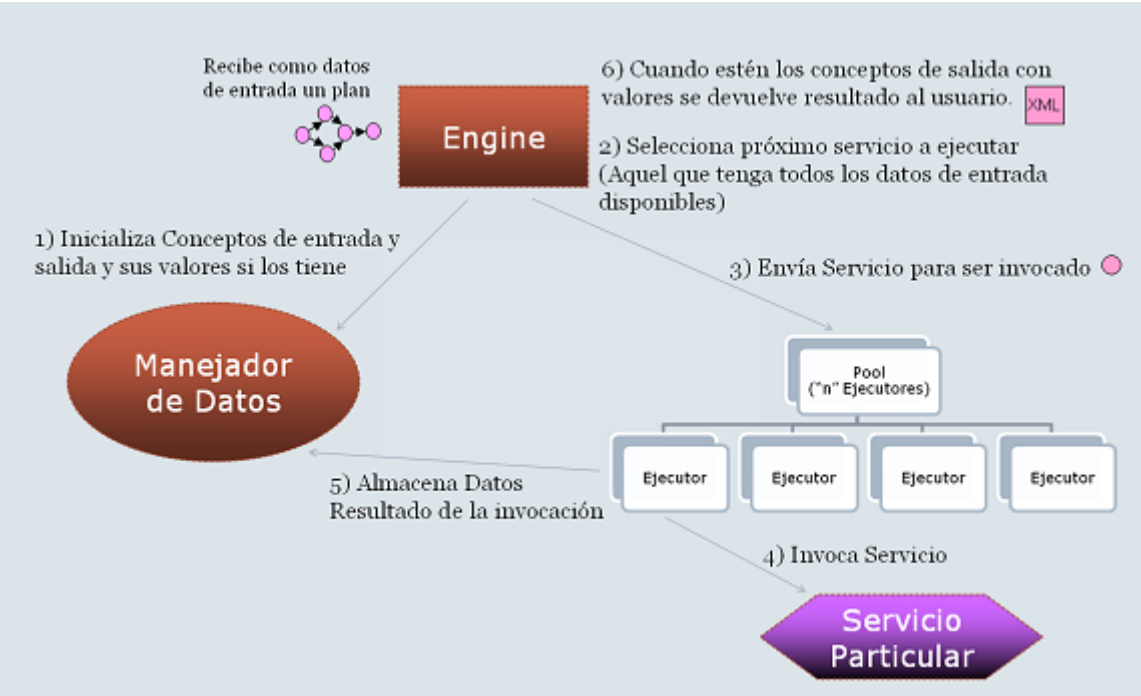

Figura 2. Interacción de los componentes del Servicio de Evaluación.

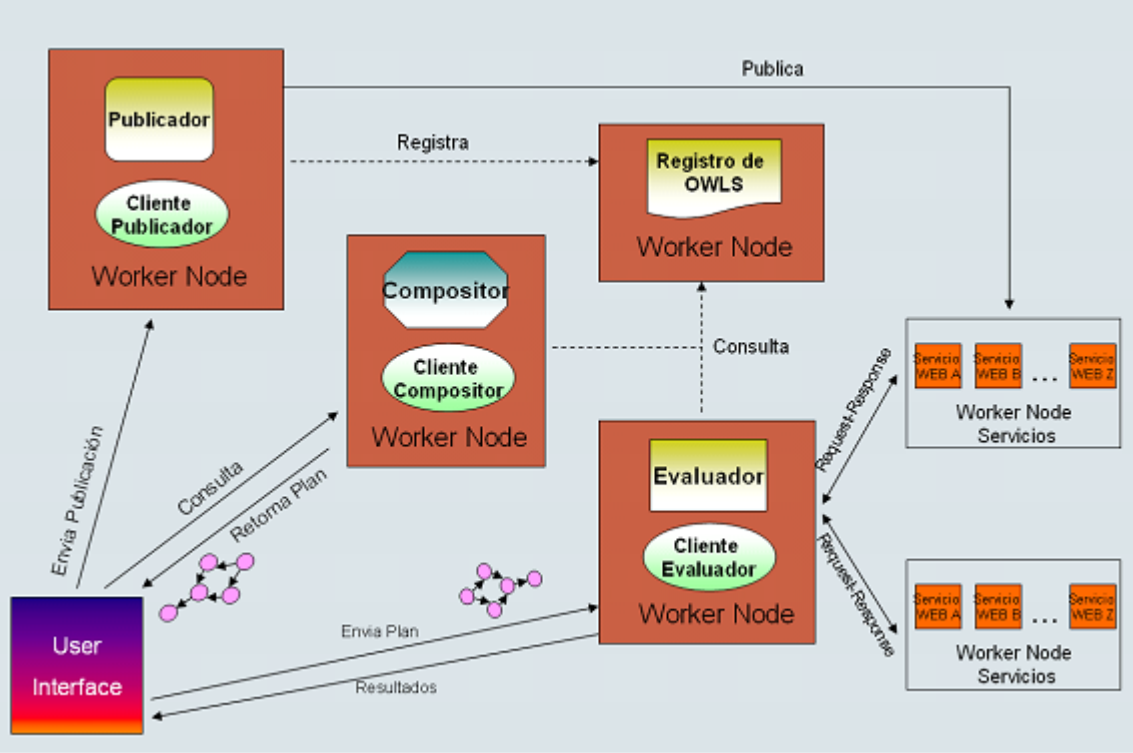

Figura 3. Arquitectura de Orinoco en el Grid.

ciclos que representa las dependencias entre los servicios y los conceptos de la ontología.

3.3. Agente Evaluador

El Agente Evaluador es el SW encargado de ejecutar los planes generados por el Agente Compositor. Este servicio debe ser invocado con un plan, un conjunto de instanciaciones de los conceptos de entrada y los conceptos de salida. El resultado es la respuesta a la consulta realizada por el usuario, es decir, los valores de los conceptos de salida que contiene el plan.

Al ser evaluado un plan, se almacenan en el manejador de datos todos los conceptos involucrados en el plan, junto con los valores de entrada (paso 1 de la **figura 2**). Luego se selecciona el próximo servicio a ser ejecutado (alguno de los que tenga todos los datos de entradas disponibles, paso 2) y se pasa al *Pool* de *Threads* para que lo invoque (paso 3); si no hay *threads* disponibles, se encola hasta que se libere alguno. El *Pool* de *Threads* permite ejecutar concurrentemente varios servicios independientes. Luego de invocado un SW (paso 4), se almacenan los datos de salida en el manejador de datos (paso 5), y así sucesivamente hasta llegar a la condición de tener todos los conceptos de salida solicitados en la consulta del usuario, con sus respectivos valores.

4. Orinoco Framework en gLite

Las tecnologías *Grid* permiten compartir contenidos, poder de cálculo y capacidad de almacenamiento. El *middleware gLite* [6] es el empleado actualmente por muchos proyectos y es en sí una plataforma Orientada a Servicios para proveer los procesos necesarios para la gestión de computación distribuida y recursos de almacenamiento, así como los servicios relacionados con la seguridad, auditoría e información [36][37]. A continuación se describen los servicios más significativos de *gLite* para Orinoco:

- UI (*User Interface*): Servidor de interfaz de usuario. Proporciona funcionalidad de acceso al *Grid* ya sea por línea de comando, web o el API provisto.
- WMS (*Workload Management Service*): Servicio de administración de carga de trabajos. Se encarga de manejar el envío de trabajos al *Grid*.
- CE (*Computing Element*) Recurso de cómputo final. Distribuye los trabajos en los diferentes nodos (*Worker Nodes*) que tenga registrado.
- WN (*Worker Nodes*): Son las máquinas donde los trabajos son realmente ejecutados. Están enlazados con el CE, quien envía los trabajos.

La integración de Orinoco Framework a la plataforma de *gLite*, se realizó siguiendo la modalidad *Share Area* [38]. El uso de este método requiere que los administradores creen un área común en la cual se puede pre-instalar aplicaciones. Estas podrán luego ser utilizadas por las Organizaciones Virtuales (OV) autorizadas. Únicamente los administradores tienen privilegios para realizar modificaciones en este espacio. Al estar pre-instaladas las aplicaciones, se reduce el *overhead* producido por la continua descarga e instalación de paquetes de software requeridos; sin embargo, se requiere la coordinación de las OV y la planificación para realizar las instalaciones en múltiples sitios. Es importante resaltar que es el manejador de aplicaciones de las OV el responsable de instalar, probar y mantener el software colocado en este espacio.

Hay varios pasos necesarios para la instalación y uso de aplicaciones usando *Share Area*: 1) el manejador de software instala las aplicaciones en los sitios seleccionados; 2) luego, crea una etiqueta y la asocia al nuevo software; y 3) los usuarios ejecutan los trabajos identificando a través de la etiqueta el software que desean usar.

Siguiendo los lineamientos de esta modalidad, los Agentes de Orinoco se instalaron en los *Worker Nodes* y su invocación se realiza a través de clientes *standalone* instalados directamente en los *Worker Nodes* (invocación local). De esta manera el usuario sólo debe enviar los parámetros necesarios para la ejecución e indicar en el archivo .jdl el servicio que desee ejecutar (Publicación, Composición, Registrador *OWL-S* y Evaluación). El usuario canaliza las peticiones a través del UI y *gLite* selecciona el recurso más apropiado para responder a la solicitud. En la **figura 3** se muestra la integración de Orinoco a *gLite*.

Al ser solicitado alguno de los servicios de Orinoco por algún trabajo que llega al WMS de gLite, éste ubica un CE (*Computing Element*) donde esté instalado el servicio solicitado. Una vez en el WN (*Worker Node*), se ejecuta el cliente Java que invoca al SW especificado (Publicador, Compositor o Evaluador).

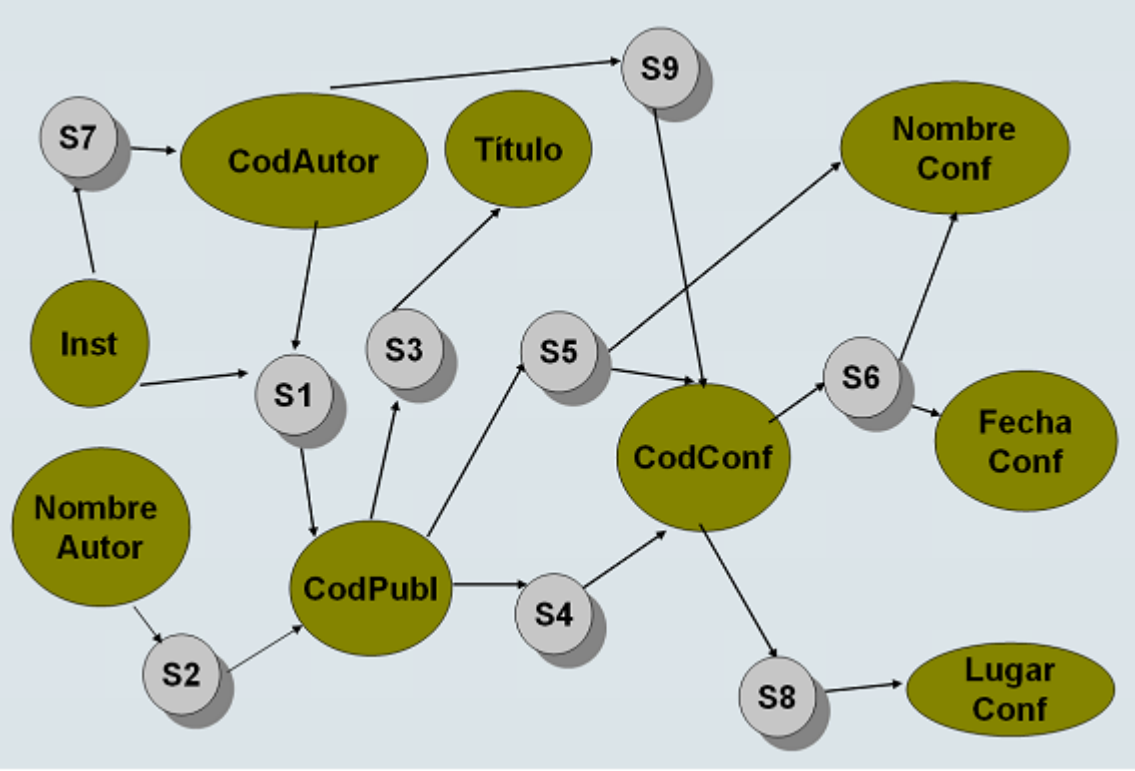

Figura 4. Grafo de dependencias: interrelación entre las aplicaciones.

5. Caso de estudio y resultados de rendimiento

En esta sección se muestran las funcionalidades de Orinoco a través de un ejemplo ilustrativo y se presenta una evaluación de desempeño del Agente Evaluador.

Todas las pruebas se realizaron en la versión de Orinoco integrada a *gLite*. Orinoco se desarrolló utilizando el lenguaje de programación Java 1.6 y el OWL-S API. La plataforma utilizada para las pruebas fue un *cluster* con *Scientific Linux* 4.6 de 25 nodos (24 de cómputo sin disco y un *front-end*), *AMD Athlon, dualcore* de 2.4GHz, 1GB de RAM, 120 GB de disco y una red de interconexión *Myrinet full duplex* de 2Gb/s de ancho de banda.

5.1. Caso de estudio

Consideremos un conjunto de aplicaciones en Java descritas en función de sus parámetros de entrada y los de salida (Ver **tabla 1**).

Estas descripciones inducen una interrelación entre las aplicaciones como se muestra en la **figura 4**. Suponga que un usuario del *Grid* tiene el siguiente requerimiento: "*Los Nombres y Fechas de Conferencias en las que haya participado algún investigador afiliado a la Universidad Simón Bolívar*" (Q={Input={*Inst*}, Output={*NombreConf, FechaConf*}}).

S1(CodAutor,Inst)	→	(CodPubl)
S2(NombreAutor)	→	(CodPubl)
S3(CodPubl)	→	(Titulof)
S4(CodPubl)	→	(CodConf)
S5(CodPubl)	→	(CodConf , NombreConf)
S6(CodConf)	→	(NombreConf , FechaConf)
S7(Inst)	→	(CodAutor)
S8(CodConf)	→	(Lugar)
S9(CodAutor)	→	(CodConf)

Tabla 1. Métodos disponibles y sus descripciones.

A continuación se detalla, para este ejemplo, el proceso de Publicación, Composición y Evaluación.

Publicación: A través de la GUI del Agente Publicador, se generan los *wrappers* y descripciones ontológicas de cada una de las 9 aplicaciones de la **tabla 1**, para transformarlas en *SWs*. Esta información está contenida en los archivos Ontologias.bin, Librerias.bin y Servicios.bin que son clases serializadas que contienen las URL's de las ontologías de conceptos, los nombres de las librerías, y los métodos con sus parámetros y conceptos asociados tanto de entrada como de salida. La GUI genera adicionalmente un siguiente archivo jdl (ver **cuadro 1**) para la invocación del servicio efectivo de Publicación sobre *gLite*:

Desde la UI, el usuario envía a *gLite* el jdl que contiene la solicitud de publicación; como resultado se obtienen los servicios publicados en los *Worker Nodes* (*WNs*).

Composición: Una vez registrados los servicios en *gLite*, el usuario puede realizar consultas (ver **cuadro 2**). El siguiente archivo myquery.xml representa la consulta Q. Para resolver la consulta, se debe invocar el servicio de composición en gLite con un archivo .jdl (ver **cuadro 3**).

Como respuesta el usuario obtendrá una posible solución para Q, como el plan de ejecución mostrado en la **figura 5**. El plan de ejecución se representa en un archivo con formato xml.

```
Type = "Job";
JobType = "Normal";
Executable = "/opt/exp_soft/publicador.sh";
StdOutput = "publicador.out";
StdError = "publicador.err"; InputSandbox={"Ontologias.bin","Librerias.bin","Servicios.bin","Servicios.jar"};
OutputSandbox = {"publicador.err","publicador.out"};
Requirements=Member("SWS_PUB",other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

Cuadro 1. Archivo para la invocación del servicio efectivo de Publicación sobre *gLite*.

```
<Query NAME="Q">
  <Inputs NUM="1">
    <Input NUM="1" TYPE="http://www.ldc.usb.ve/Ont/Conceptsowl#Institucion"/>
  </Inputs>
  <Outputs  NUM="2">
    <Output NUM="0" TYPE="http://www.ldc.usb.ve/Ont/Concepts.owl#Investigador"/>
    <Output NUM="0" TYPE="http://www.ldc.usb.ve/Ont/Concepts.owl#FechaPublicacion"/>
  </Outputs>
</Query>
```

Cuadro 2. Archivo myquery.xml representando la consulta Q.

```
Type = "Job";
JobType = "Normal";
Executable = "/opt/exp_soft/compositor.sh";
Arguments = "DP-BF myquery.xml";
StdOutput = "compositor.out";
StdError  =  "compositor.err";
InputSandbox={"myquery.xml"};
OutputSandbox = {"compositor.err","compositor.out"};
Requirements=Member("SWS_COMP", other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

Cuadro 3. Archivo .jdl mediante el que se invoca el servicio de composición en gLite.

```
<Query NAME="Q">
  <Inputs NUM="1">
    <Input NUM="1" TYPE="http://www.ldc.usb.ve/Ont/Concepts.owl#Institucion">
      <Value VAL="USB"/>
    </Input>
  </Inputs>
  <Outputs NUM="2">
    <Output NUM="0" TYPE="http://www.ldc.usb.ve/Ont/Concepts.owl#Investigador">
      <Value VAL="Pedro Perez"/>
      <Value VAL="Gabriel Ramirez"/>
    </Output>
    <Output NUM="0"
TYPE="http://www.ldc.usb.ve/Ont/Concepts.owl#FechaPublicacion">
      <Value VAL="2008"/>
      <Value VAL="2010"/>
    </Output>
  </Outputs>
</Query>
```

Cuadro 4. Valores de los atributos de salida de la consulta Q.

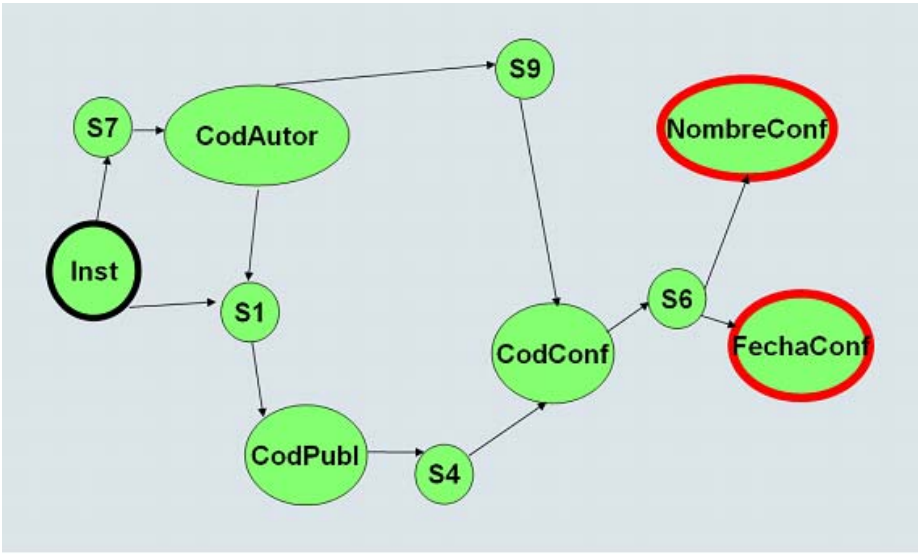

Figura 5. Una posible solución para Q.

Evaluación: El servicio de Evaluación se invoca con el archivo plan.xml, que contiene el plan de ejecución devuelto por el Compositor, y con un archivo similar a myquery.xml con etiquetas adicionales para indicar los valores de los atributos de entrada. Para este ejemplo, se agrega la línea *<VALUE VAL = "USB">*, que instancia el atributo de entrada Institución. El resultado son los valores de los atributos de salida de Q que podemos observar en el **cuadro 4.**

5.2. Pruebas de desempeño del evaluador

Los experimentos para las pruebas de desempeño consistieron en variar el tamaño del *Pool* de *Threads* y observar cómo se aprovecha la concurrencia que éste provee. Se definió un conjunto de 10 servicios *dummy* con diferentes tiempos de ejecución. Se realizaron diferentes consultas, obteniendo del Compositor planes de ejecución de diferentes tamaños: de 2 a 8 *SWs*. Para cada tamaño de plan, se generaron 6 planes de ejecución diferentes (48 planes en total). Los 48 planes fueron evaluados por el Evaluador variando el nivel máximo permitido de concurrencia.

Con el fin de reducir los tiempos de evaluación, uno de los objetivos del Evaluador es el de incorporar mecanismos para evaluar, en forma concurrente, los servicios de los planes que así lo permitan. En la **figura 6** se presenta el efecto, en el desempeño de los planes, de aumentar el nivel de concurrencia permitido en el agente Evaluador. Como lo muestran las líneas de tendencia de la **figura 6** los tiempos de ejecución disminuyen a medida que se aumenta el tamaño del *Pool*, debido a que se aprovecha la concurrencia en aquellos planes que lo permitan. A pesar de existir una tendencia que muestra que a mayor tamaño del *Pool de Threads* los tiempos de ejecución son menores, el tamaño y tipo (amplio o profundo) del plan también influye. No siempre el Evaluador podrá aplicar concurrencia, en estas pruebas se combinaron tanto planes que lo permitían (representados por planes amplios) y planes que no lo permiten (planes profundos) ya que hay dependencia de datos.

6. Conclusiones y trabajo futuro

En este trabajo presentamos Orinoco *Framework*, una infraestructura que permite la Publicación, Composición y Ejecución de *SWs* para la resolución de requerimientos de usuarios en ambientes *Grid*. Contar con un *framework* como Orinoco, que semi-automatiza las tareas de Publicación, Composición y Evaluación de Servicios Web Semánticos minimiza los errores comunes cuando se realizan en forma manual. Orinoco beneficia a usuarios sin conocimientos profundos en el área de computación que experimentan con requerimientos complejos, así como a científicos y organizaciones que desean compartir sus servicios.

Al integrar Orinoco a un *Grid* se aprovechan las ventajas que brindan estas plataformas: agrupación de usuarios con intereses comunes, seguridad, confiabilidad, disponibilidad y acceso a recursos, entre otros. Los resultados demuestran que es una herramienta útil que facilita las tareas de Publicación, Composición y Evaluación, pudiendo reutilizar Servicios Web desarrollados por otros. Además de que el proceso de evaluación es efectivo y aprovecha la concurrencia.

Actualmente estamos trabajando en extender la implementación de gLite para colocar los servicios de Publicación, Composición y Evaluación como servicios propios del *Grid*, creando tipos de trabajo especiales para cada servicio. De esta forma serían "Servicios *Embeded in gLite*", mejorándose el rendimiento y uso de los recursos.

Referencias

[1] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, 2005. ISBN: 0131858580.

[2] B. Vassiliadis, K. Giotopoulos, K. Votis, S. Sioutas, N. Bogonikolos, S. Likothanassis. Application Service Provision through the Grid: Business models and Architectures. *Proc. of the Int. Conference on Information Technology: Coding and Computing (ITCC'04)*, Volume 2, Washington, DC, USA (2004) 367.

[3] J. Rao, X. Su. A survey of automated web services composition methods. *Proc. of the First International Workshop in Semantic Web Services and Web Process Composition. Volume 3387*, Springer, LNCS, 2004 pp. 43-54.

[4] N. Shadbolt, T. Berners-Lee, W. Hall. The Semantic Web Revisited. *IEEE Intelligent Systems 21(3)* ,2006, pp. 96-101.

[5] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, K. Sycara. Bringing Semantics to Web Services: The OWL-S Approach. *Proceedings of The First Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*.

[6] gLite. *Lightweight Middleware for Grid Computing* (2006), <http://glite.web.cern.ch/glite/>.

[7] World Wide Web Consortium. W3C 2010, <http://www.w3c.es/>.

[8] M.P. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. *Fourth International Conference on Web Information Systems Engineering (WISE'03)*, 2003.

[9] Argugrid. Programme, E.C.S.F.: ARGUGRID (2006) <http://www.argugrid.eu/>.

[10] T. Guan, E. Zaluska, D.D. Roure. An automatic service discovery mechanism to support pervasive devices accessing the semantic grid. *International Journal of Automatic Computing 1(1)*, 2009, pp. 34-49.

[11] L. Chen, N. Shadbolt, C. Goble, F. Tao, C. Puleston, S. Cox. Semantics-assisted problem solving on the semantic grid. *Journal of Computational Intelligence, special issue 21(2)*, 2005.

[12] S. Majithia, D.W. Walker, W.A. Gray. Automated Composition of Semantic Grid Services. *Proceedings of AHM, 2004*.

[13] J. Jiang, S. Zhang, J.H. Schlichter, G. Yang. Workflow managment in the grid era: A goal-driven approach based on process patterns. *Multiagent and Grid Systems 5(3)*, 2009, pp. 325-343.

[14] S. Bromuri, V. Urovi, M. Morge, F. Toni, K. Stathis. A multi-agent system for service discovery, selection and negotiation. *Proc. of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.

[15] N. Srinivansa, M. Paolucci, K. Sycara. *CODE: A Development Enviroment for OWL-S Web Services*. Technical Report CMU-RI-TR-05-48, Robotics Institute, Pittsburg, PA, 2005.

[16] M. Paolucci, T. Kawamura, T.R. Payne, K.P. Sycara. Semantic matching of web services capabilities. *International Semantic Web Conference*, 2002 pp. 333-347.

[17] M. Klusch, B. Fries, K. Sycara. OWLS-MX: A Hibryd Semantic Web Service Matchmaker for OWL-S Services. *Journal in Web Semantic 7(2)*, 2009, pp. 121-133.

[18] K. Zamanifar, A. Zohali, N. Nematbakhsh.

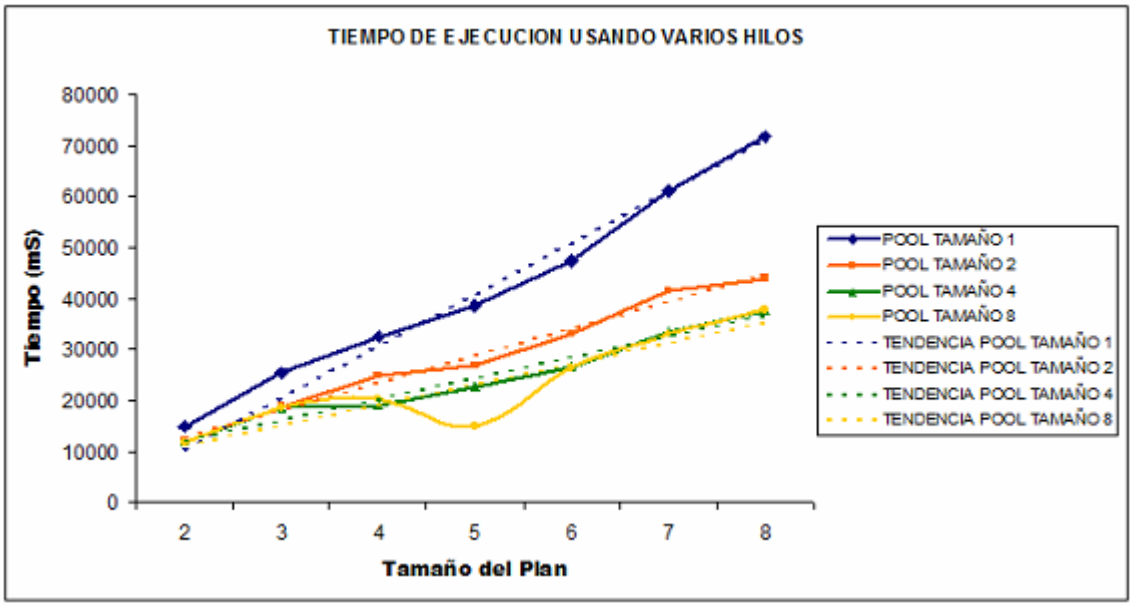

Figura 6. Evaluación de planes variando el tamaño del *Pool de Threads*.

Matching model for semantic services discovery. *Proc. of the International Conference on Advance Information Networking and Applications Workshops*, Bradford, United Kingdom, 2009, pp. 50-54.

[19] S. Bansal, J.M. Vidal. Matchmaking of Web Services-Based on the DAML-S Service Model. *II Internat. Joint Conf. on Autonomous Agents and Multiagent Systems*, 2003, pp. 926-927.

[20] P. Fu, S. Liu, H. Yang, L. Gu. Matching algorithm of web services based on semantic distance. *Proc.of the International Workshop on Information Security and Application (IWISA)*, Qingdao, China, 2009.

[21] Y.L. Chi, H.M. Lee. A formal modeling platform for composing web services. *Expert Systems with Applications: An Int. Journal 34(2)*, 2008, pp. 1500-1507.

[22] J. Hoffmann, J.S. Ingo Weber, T. Kaczmarek, A. Ankolekar. Combining Scalability and Expressivity in the Automatic Composition of Semantic Web Services. *ICWE, 2008*, pp. 98-107.

[23] M.C. Jaeger, G. Muhl, S. Golze. QoS-aware composition of web services: An evaluation of selection algorithms. *LNCS 3760 (2005)*, pp. 646-661.

[24] D. Thakker, T. Osman, D. Al-Dabass. Knowledgeintensive semantic web services composition. *Proceedings of Tenth International Conference on Computer Modeling and Simulation (UKSIM 2008)*, Los Alamitos, CA, USA, IEEE Computer Society, pp. 673-678.

[25] C. Hogg, U. Kuter, H. Muñoz-Avila. Learning Hierarchical Task Networks for Nondeterministic Planning Domains. *Proceedings of the 21st Internat. Joint Conference on Artificial Intelligence (IJCAI09)*, Pasadena, California, USA, 2009.

[26] A. Brogi, S. Corfini, R. Popescu. Semantics-based composition-oriented discovery of web services. *ACM Transactions on Internet Technology 8(4)*, 2008, pp. 1-39.

[27] E. Blanco. Techniques to Produce Optimal Web Service Compositions in The Semantic Grid. *On-line. CEUR Workshop Proceedings*, Tenerife, España (2008) 6–10 Advisors: Yudith Cardinale and Maria Esther Vidal.

[28] E. Blanco, Y. Cardinale, M.E. Vidal. Aggregating Functional and Non-Functional Properties to Identify Service Compositions. *IGI BOOK (53) (2010)*. Aceptado para ser publicado en 2011.

[29] Y. Cardinale, J.E. Haddad, M. Manouvrier, M. Rukoz. Web services selection for transactional composition. *Proc. of the International Conference on Computational Science (ICCS)*, Amsterdam, The Netherlands, 2010.

[30] K. Lee, R. Sakellariou, N.W. Paton, A.A.A. Fernandes. Workflow Adaptation as an Autonomic Computing Problem. *WORKS '07: Proceedings of the 2nd workshop on Workflows in support of largescale science*, New York, NY, USA, ACM, 2007, pp. 29-34.

[31] R. Sakellariou, H. Zhao. A Low-Cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems. *Sci. Program. 12(4)*, 2004, pp. 253-262.

[32] L. Zeng, A.H.H. Ngu, B. Benatallah, R.M. Podorozhny, H. Lei. Dynamic composition and optimization of web services. *Distributed and Parallel Databases 24(13)*, 2008, pp. 45-72.

[33] K. Wolstencroft, P. Alper, D. Hull, C. Wroe, P. Lord, R. Stevens, C. Goble. The myGrid ontology: bioinformatics service discovery. *Int. J. Bioinformatics Research and Applications 3(3)*, 2007, pp. 303-325.

[34] P. Torroni, M. Gavanelli, F. Chesani. Argumentation in the Semantic Web. *IEEE Intelligent Systems 22(6)*, 2007, pp. 66-74.

[35] The Mindswap Group. *OWLS: Semantic Markup for Web Services API*, 2004. <http://www.mindswap.org/2004/owl-s/api/>.

[36] gLite. *gLite Middleware*, 2010. <http://glite.web.cern.ch/glite/>.

[37] D. Scardaci, G. Scuderi. A Secure Stora Service for the gLite Middleware. *Proc. of the Third Internat. Symp. on Information Assurance and Security*, 2007.

[38] EGEE. EGEE-II: User Information Group (UIG). EGEE Software Installation, 2010. <http://egee-uig.web.cern.ch/egee-uig/>.

Nota del editor

[1] "Orinoco Framework es un *framework* web ligero de pila completa escrito en PHP5. Se basa en la arquitectura MVC (*Model-View-Controller*) e implementa el patrón de diseño Model 2. Su infraestructura ayuda a los desarrolladores a crear aplicaciones limpias y mantenibles" (traducción libre de <http://code.google.com/p/orinoco-framework/>).