Combinando acceso local y global a Ontologías en Sistemas Multiagentes

Héctor G. CEBALLOS y **Ramón F. BRENA**
Centro de Sistemas Inteligentes, Instituto Tecnológico de Monterrey
Monterrey, Nuevo León 64849, Mexico

RESUMEN

A medida que el conocimiento Ontológico cobra más importancia en sistemas basados en agentes, su manejo se vuelve crucial para aplicaciones exitosas. Pero colocar todas las capacidades para el manejo de ontologías en cada uno de los agentes del sistema puede volverlo demasiado pesado. Proponemos un enfoque híbrido en el cual parte de la ontología es manejada localmente, usando un "componente cliente", y el resto del conocimiento sobre la ontología es manejado por un "agente de ontologías", el cual es accesado por otros agentes en el sistema a través de su componente cliente. Proponemos métodos específicos para representar, almacenar, consultar y traducir ontologías para su uso efectivo en el contexto del sistema "JITIK", el cual es un sistema multiagentes para distribución de conocimiento e información. Asímismo reportamos la implementación de un prototipo operativo de nuestra propuesta.

Palabras clave: Ontologías, Agentes, Sistemas Multiagentes, Distribución de Conocimiento, Semantic Web.

1 INTRODUCCIÓN

Es ampliamente aceptado que la comunicación es un requerimiento absoluto para la mayoría de las aplicaciones de sistemas multiagentes. Aún cuando la comunicación a nivel sintáctico no está completamente resuelta, lo más retador ahora es tomar en cuenta el *significado* de los mensajes de los agentes. Este es uno de los aspectos cruciales que tenemos que tratar para construir aplicaciones reales basadas en agentes[3].

El término *ontología* se refiere a la definición de los significados de términos usados en comunicaciones entre agentes[4]. Las ontologías permiten definir conceptos y sus relaciones, propiedades, operaciones, y todo ello en forma estructurada. Estándares abiertos como DAML+OIL[12], o más recientemente OWL[14], permiten publicar conocimiento ontológico en una manera entendible tanto por humanos como por máquinas.

Pero aún teniendo una representación estándar, hay que decidir dónde poner cada pieza de conocimiento a representar. Algunos esfuerzos como el proyecto Cyc[16] sugieren construir un enorme repositorio centralizado de conocimiento enciclopédico. Otros consideran esto impráctico en términos de desempeño y robustez, y prefieren enfoques descentralizados[6]. Pero manejar ontologías distribuídas genera nuevos problemas como: 1) dónde poner o de dónde tomar las piezas de conocimiento, esto es, cómo distribuir el conocimiento; 2) cómo mantener algún grado de coherencia entre las diferentes piezas -o inclusive versiones- de conocimiento ontológico.

El método que presentaremos en este trabajo está diseñado para ser usado en el contexto del proyecto JITIK[5]. JITIK -que significa *Just-In-Time Information and Knowledge*- es un sistema basado en multiagentes para distribuir piezas de conocimiento entre los miembros de una organización grande o distribuida, por esto soporta la función de administración de conocimiento. A pesar de que nuestra propuesta de manejo de ontologías fue primariamente orientada para su aplicación en el sistema JITIK, lo cual impone una arquitectura así como otras restricciones, es aplicable en principio a un amplio rango de sistemas basados en agentes.

Nuestro enfoque

En este trabajo proponemos una solución a uno de los aspectos de esta problemática: proponemos un método para combinar ontologías centralizadas con ontologías distruibidas. Consideramos un repositorio central encapsulado en un "agente de ontologias", (AO) que provee respuestas a preguntas acerca de la ontología a otros agentes en el sistema. Pero además de eso, dotamos a cada agente en el sistema con un "componente cliente de la ontología" (CCO), el cual le proporciona las capacidades básicas para manejo de ontologías. Este esquema funciona de la siguiente manera:

- Los agentes "regulares" inician con un subconjunto de la ontología común, la cual es cargada al arranque desde un recurso en internet. Los agentes regulares utilizan sus ontologías locales, manejadas por el CCO, mientras su conocimiento local sea suficiente para la actividad del agente.
- Cuando más conocimiento es requerido -por ejemplo, un término desconocido llega desde otro agente- el CCO consulta al AO, y recibe un anexo a la ontología básica, que le permite al agente continuar trabajando. El CCO almacena localmente la adición a la ontología para poderla usar en el futuro.

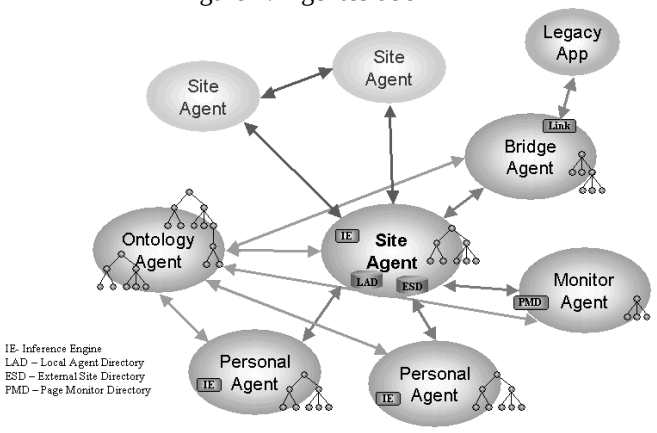
Esta solución simplifica alguna de las complejidades inherentes de la distribución de conocimiento, porque:

1. No hay riesgo de incoherencia -cada pieza de conocimiento viene a final de cuentas de una ontología común-ya sea desde la ontología inicial o como resultado de una consulta al AO.
2. No hay decisiones *a priori* acerca de cómo distribuir o especializar el conocimiento, porque el conocimiento local se va especializando automáticamente a medida que la operación del agente procede.

Hacemos notar, sin embargo, que este enfoque no es adecuado para todo tipo de situaciones. En particular, en ambientes abiertos y dinamicos como el internet, donde concurren ontologias distintas y hasta contradictorias, no se puede sostener la hipotesis de una ontologia global y coherente. Sin embargo, en ambientes tales como una empresa en particular, nuestra hipotesis de una ontologia global es perfectamente razonable.

En la sección 2 detallamos nuestro método. La sección 3 describe un prototipo funcional. Los resultados de los experimentos se presentan en la sección 4. En la sección 5 presentamos algunos trabajos relacionados, y finalmente en la sección 6 damos algunas conclusiones.

Figure 1: Agentes de JITIK



2 SOLUCIÓN PROPUESTA PARA MANEJO DE ONTOLOGÍAS EN JITIK

En la figura 1 mostramos la arquitectura de JITIK, compuesta por varios tipos de agentes, como el *agente de Sitio*, quien tiene a su cargo la distribución de información a distintos *agentes personales*, los cuales interactúan con un usuario final; también hay *agentes bridge* para interactuar con programas tradicionales (aplicaciones heredadas). Los agentes de sitio son el corazón de un "grupo" compuesto por un agente de sitio y varios agentes personales atendidos por el primero. En una organización, los grupos pueden estar asociados a departamentos, divisiones, etc., dependiendo del tamaño de estos. Las redes pueden ser hechas conectando varios agentes de sitio. Las organizaciones distribuidas como compañías multinacionales pueden tener una red de muchos agentes de sitio conectados. También hay *agentes de ontología*, los cuales discutiremos a continuación.

Desde el punto de vista de manejo de ontologías, podemos clasificar los agentes en dos categorías: *agentes de ontologías*, y agentes "*regulares*", que son todos los otros agentes. Algunas veces llamaremos agentes "clientes" a los agentes regulares, porque actúan como un cliente respecto al agente de ontologías. El AO y los clientes corresponden a los roles "iniciador" y "respondedor" en metodologías estándar de diseño de agentes como [7].

Los agentes regulares accesan las ontologías a través del CCO previamente introducido, y el CCO eventualmente se comunica con el AO. La arquitectura tanto del agente de ontologías como del agente cliente se muestra en la figura 2. Ahí se muestra al AO en el lado izquierdo, con algunos componentes internos que se discuten posteriormente, y un agente cliente en el lado derecho, con el Componente Cliente de la Ontología (CCO). Pero antes de discutir los detalles internos del AO y del CCO, debemos tratar algunos aspectos esenciales de manejo de ontologías que impactarán en gran medida el diseño del AO y del CCO. Principalmente, tenemos que presentar nuestras opciones de codificación de ontologías, de formatos y de almacenamientos, lenguajes y métodos de consulta.

Agente de ontologías y Agentes clientes

Los agentes clientes tratan de satisfacer sus necesidades de conocimiento ontológico usando el conocimiento del CCO. Si es necesario, el CCO hace una consulta al AO e interpreta y usa la respuesta, y eventualmente la incorpora al conocimiento local.

Agente de ontologías

El AO encapsula la funcionalidad para jugar el rol de proveedor de conocimiento, almacenar la ontología convenientemente codificada, traducir, interpretar y ejecutar las consultas recibidas, y luego traducir los resultados a un formato entendible por los agentes clientes. Esta separación de formatos provee una capa de independencia, de manera que la representación de la ontología pueda ser cambiada en el AO sin impactar a los agentes clientes.

Agente cliente

Los agentes clientes acceden las definiciones de la ontología a través de su CCO. Al arranque cargan una *ontología base*, la cual es evidentemente dependiente de la aplicación, y tratan de usarla mientras sea suficiente para su operación. En el sistema JADE[1], las ontologías son necesarias para propósitos de validación de mensajes en primera instancia. Cada término en las conversaciones de los agentes debe ser validado contra una definición en la ontología. Por esto, normalmente la ontología base contendrá definiciones de términos comunes y poco especializados. El tamaño de la ontología base debe hallar un punto de equilibrio entre eficiencia en espacio -lo que lleva a una ontología inicial pequeña- y eficiencia en tiempo -lo que lleva a maximizar la cobertura del conocimiento local de manera que se minimizen las consultas remotas.

Representación de la ontología

Quisimos representar la información ontológica desde un punto de vista abstracto, esto es, independientemente de un estándar específico para representación de ontologías. Esta es la razón por la cual se definen algunas categorías ontológicas, denominadas *metaontología*. Posteriormente, cuando se construya el prototipo, estas categorías se mapean a las categorías

Figure 2: Arquitectura para manejo de ontologías

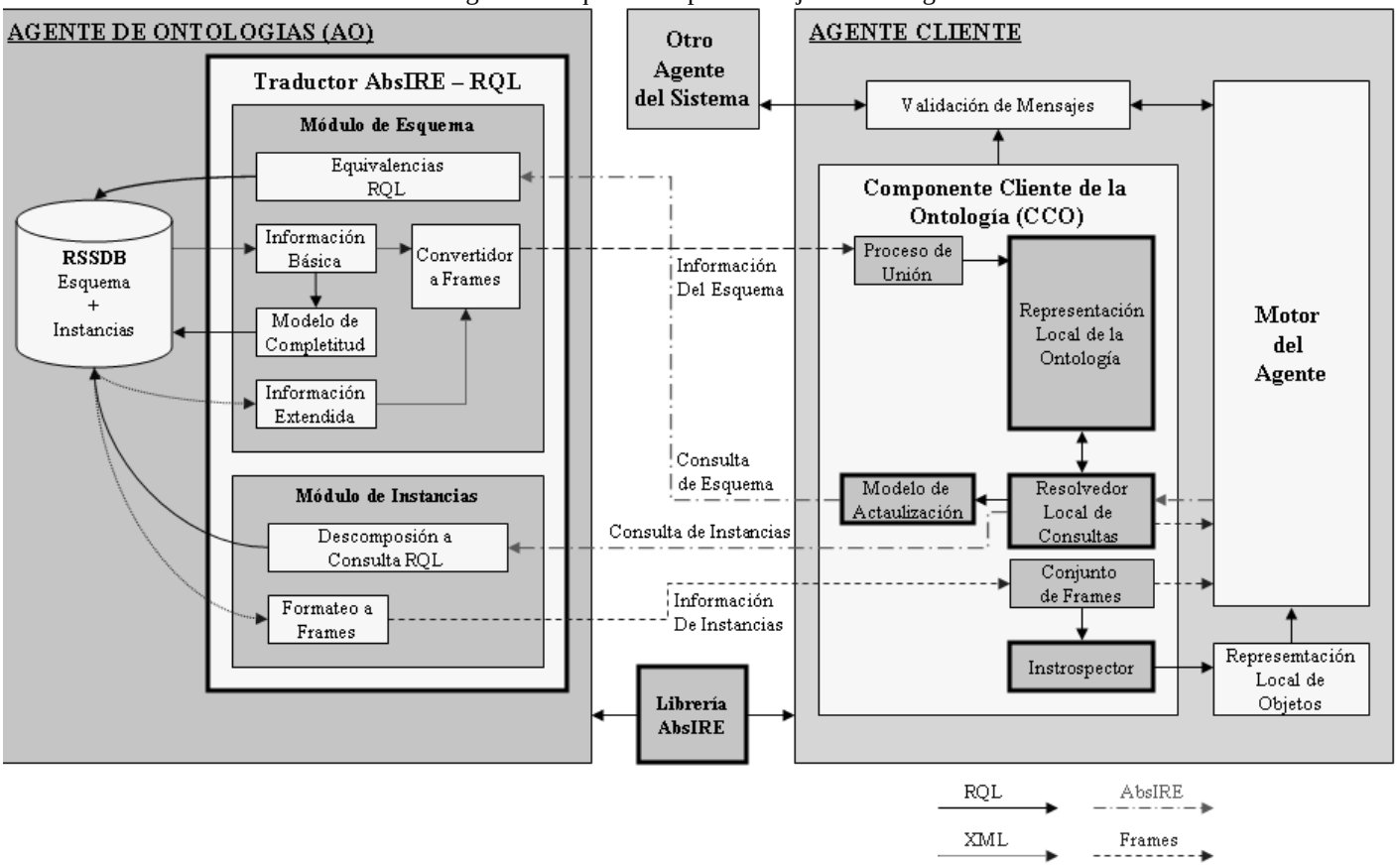


Table 1: Principales conceptos de la metaontología

Concepto	Atributos
Clase	Nombre Propiedades SubclaseDe SuperclaseDe
Propiedad	Nombre Rango Dominio
Instance	Nombre Clase Valores

disponibles en un estándar específico, como OWL[14]. Las categorías principales de la metaontología están representadas en la Tabla 1.

Un ejemplo de información acerca de la administración de una universidad codificada en esta metaontología se vería de la siguiente manera:

```
Clase Nombre: Persona
    SubclaseDe: Entidad
    SuperclaseDe:
        Trabajador,
        Estudiante
    Propiedades:
        nombre,
        correo,
        telefono,
        ...
Clase Nombre: Trabajador
    SubclaseDe: Persona
    SupercalseDe: Staff, Profesor
    Propiedades:
        salario,
        ...
Clase Nombre: Profesor
    SubclaseDe: Persona
    SuperclaseDe:
        ProfAsociado,
        ProfTiempoCompleto,
        ProfVisitante
    Propiedades:
        enseña,
        asesora,
        ...
Propiedad Nombre: Salario
    Rango: numero
```

Mecanismo de Consulta

A continuación describiremos el mecanismo de consulta en general; detalles del mecanismo se presentan únicamente en la descripción del prototipo (sección 3). Consiste de tres elementos: el lenguaje de consulta, el motor de consultas y el formato de respuesta.

Respecto al lenguaje de consulta, estábamos interesados en lenguajes capaces de expresar consultas que contengan conjunciones de términos de la forma:

$$\forall x \; Attr(x) \otimes k$$

donde Attr es un atributo (una fórmula lógica con predicados tomados de los atributos en la metaontología), la cual debe relacionarse a una constante dada k por medio de una relación $\otimes$ - como la igualdad, por ejemplo. Un ejemplo simple de un consulta abstracta podría ser, para el ejemplo de la universidad, preguntar por las instancias de la clase *Profesor*: [1]

$$\forall x \; (Class.Name(x) = 'Professor').Instances$$

Entre estos lenguajes podemos encontrar a RQL [15], que a pesar de estar orientado a RDF, su sintaxis es similar a SQL, por lo que la codificación de consultas no es difícil. Otras opciones serán presentadas en la sección del prototipo (sección 3).

El *motor de consultas* es responsable de resolver las consultas hechas a la ontología. Su desempeño será uno de los factores más críticos en el desempeño global del AO, dado que estará respondiendo constantemente preguntas provenientes de otros agentes. Un ejemplo de motor de consulta es RQL sobre RSSDB de RDFSuite[9]. Jena cuenta con un motor de consulta (RDQL), además de métodos de consulta sobre la ontología.

Las respuestas a las consultas se codifican en un *formato de respuesta*. Una vez que el cliente reciba una respuesta del AO, puede procesar su información. Este procesamiento-decodificación puede ser costoso tanto para el agente cliente como para el AO si no se eligen formatos adecuados.

Entre los formatos de respuesta disponibles encontramos RDF sobre XML, y los formatos de frames provistos por el soporte para ontologías de JADE. RSSDB arroja respuestas en XML, así que hay que hacer la traducción a frames de JADE, ya sea en el servidor o en el cliente. Consideramos preferible hacer la traducción en el lado del servidor (el AO), porque de esta forma el proceso se vuelve transparente para los agentes clientes, y así un reemplazo de tecnología en el AO no tiene que ser notado en los agentes clientes.

El Componente Cliente de la Ontología (CCO)

El acceso local a la ontología se encapsula en el CCO el cual es incorporado en el agente cliente. Al arranque del agente, el CCO es responsable de obtener -normalmente de una dirección en internet- una ontología base. Este mecanismo es suficientemente general para ser personalizado de manera que diferentes tipos de agentes carguen diferentes ontologías base, a pesar de que no hicimos esto en nuestro prototipo.

Para subsanar las limitaciones de la ontología base, el CCO es responsable de accesar al AO para extender su conocimiento sobre la ontología, a través de mecanismos de consulta que hemos descrito. Los resultados de la consulta son incorporados por el CCO a la ontología local, extendiendo automáticamente la ontología conforme sea necesario.

En este modelo, la existencia del AO es transparente al agente cliente, dado que este último dirige todas sus consultas al CCO, éste toma a su cargo el proceso completo hasta

[1] No daremos la sintaxis detallada para estas consultas abstractas por lo evidente del ejemplo.

que llega una respuesta al agente -ya sea que venga de una consulta al CCO o de una consulta del CCO al AO.

Como se puede ver en el diagrama de la figura 2. El CCO tiene los siguiente elementos:

- *Representación Local de la Ontología*. Permite almacenar un subconjunto de la ontología y soporta consultas locales.
- *Resolvedor Local de Consultas*. Sirve de interfaz entre el agente mismo y la vista de la ontología. Provee métodos usados por el agente para consultar acerca de los esquemas o instancias en la ontología.
- *Validación de Mensajes*. Dado que el CCO contiene definiciones de términos de la ontología base y de las consultas al AO, permite validar mensajes en términos de una ontología, como es requerido por la plataforma JADE.
- *Contenedor de Esquemas y de Instancias*. Deseamos que la información de esquemas esté separado de la información de instancias por razones de desempeño, particularmente cuando involucra un número grande de instancias. La información de instancias puede ser accesada ya sea directamente desde el agente cliente o exportando a una clase Java a través del uso del *Introspector*.
- Mecanismo de Agregación. El nuevo conocimiento proveniente del AO como respuesta a una consulta es incorporada en la vista local. Por supuesto, el uso imprudente de esta característica puede degradar el desempeño del CCO.

3 PROTOTIPO

Hemos desarrollado una implementación algo simplificada de las ideas presentadas aquí. Las simplificaciones que introdujimos fueron las siguientes:

- A pesar de que trabajamos con el paquete RDFSuite, y de hecho construimos un prototipo muy básico con esta tecnología, sólo soportó ontologías RDF. Así que teníamos que desarrollar la traducción mencionada de estructuras JADE a RQL. Decidimos únicamente usar el paquete Jena en su lugar, el cual en algún punto de nuestro proyecto incorporó almacenamiento persistente, volviéndose así una solución muy atractiva. Descartamos RDFSuite por lo mismo.
- El acceso a las ontologías en los agentes clientes y en el AO era idéntico, ambos basados en una clase *ClientOntology* que desarrollamos, la cual utiliza características del paquete Jena. ClientOntology es implementado tanto en el CCO como en el AO.
- El CCO no redirecciona automáticamente las consultas al AO. En lugar de eso, el agente cliente especifica a cuál componente enviar la consulta, ya sea al CCO o al AO.
- No se hace distinción entre instancias y esquemas para propósitos de almacenamiento.

Formato de Ontología

Utilizamos particularmente RDF[13] y DAML+OIL[12] como lenguaje de representación. RDF, con su extensión RDFS, contiene todos los elementos mínimos necesarios para representar nuestras ontologías, pero DAML+OIL provee mejores características ontológicas que nos permitirían extender nuestro sistema en un futuro.

Buscamos un software intermedio que encapsulara el acceso a la ontología, de forma que cualquier cambio en el formato permaneciera local al software y transparente para el resto del sistema. Se eligió Jena Toolkit [8] para esta tarea. Jena está conformado de librerías JAVA para manejo de ontologías RDF y DAML+OIL.

Estructura del prototipo

Un componente encapsulado en la clase xont.ClientOntology constituye el núcleo del prototipo. Dicho componente aparece tanto en el agente cliente como en el AO. Sus elementos principales son: 1) la ontología DAML es almacenada utilizando Jena Toolkit; 2) hay un Resolvedor de Consultas, el cual traduce las consultas de frames JADE a llamadas de métodos Jena; 3) en el lado del AO los resultados son ensamblados en la forma de frames JADE, para ser enviados al agente cliente.

En el agente cliente *se pueden encontrar los Conceptos Dinámicos JADE* que constituyen las estructuras ontológicas JADE que almacenan la ontología DAML leída por medio de Jena. La ontología en el CCO tiene un formato doble, por una parte se almacena en estructuras de Jena (DAML+OIL) y por otra en estructuras de JADE (frames). Ambas versiones están sincronizadas y tienen usos distintos: la primera se utiliza para hacer consultas, mientras que la segunda permite que JADE valide los mensajes entre agentes.

Un problema técnico para construir las ontologías JADE en paralelo es que los conceptos JADE no pueden pertenecer a varias clases al mismo tiempo. Para resolver esta incompatibilidad, introdujimos las *instancias mixtas*, que son instancias definidas en DAML+OIL con varios tipos, y que al momento de traducirse se definen como instancias de una *clase compuesta*. Esto está detallado en [2].

Durante la inicialización, el AO carga varias piezas de ontologías de una lista de URLs, en la cual se incluye la ontología base, entre otras. Los agentes clientes sólo cargan la ontología base al inicio.

Cuando un agente cliente necesita hacer una consulta, tiene dos opciones: ya sea que pregunte a su CCO, o al AO. Se deja a consideración, en este prototipo, que el agente cliente decida a cuál ontología es mejor preguntar. Como regla general, sería buena idea preguntar al CCO primero, en orden de reducir el tráfico en la red.

Las consultas se codifican como objetos de la clase AbsIRE de JADE, los cuales son construidos por medio de código Java en el agente cliente. Después son eventualmente enviados al AO usando mensajes en FIPA ACL, y luego son interpretados y resueltos conforme van llegando. Luego los métodos de Jena se hacen cargo de la consulta. Cuando el CCO es consultado, el preoceso es exactamente el mismo, excepto que no se envía mensaje ACL para hacer la pregunta.

La respuesta se obtiene haciendo múltiples llamadas a las métodos de Jena y se traduce de nuevo a frames JADE por

medio del CCO. Tanto si se pregunta al AO o al CCO, se utiliza un mensaje ACL como vehículo para recibir la respuesta; de esta manera se tiene un método único de recepción de respuestas.

Resolución de Consultas

En el prototipo, las consultas consisten de lo siguiente:

- Un *cuantificador*, el cual indica si queremos todos los resultados o sólo el número de incidencias.
- Una *variable*, que especifica el tipo de dato buscado.
- Un *operador de consulta*.

Los operadores de consulta están definidos de forma que su evaluación se realiza en dos pasos: primero, se especifican las características de los objetos, y segundo, se especifica el elemento buscado. Durante el primer paso, Jena extrae una lista de esquemas DAML+OIL que satisfacen la especificación dada, y en el segundo paso se construyen los resultados.

Por ejemplo, asumamos que deseamos conocer cuáles propiedades están definidas en la clase "Trabajador". Usaremos el cuantificador ALL, así que las propiedades mismas, y no sólo el número de ellas, es devuelto. Ahora definimos la variable "x" de tipo *CLASS_PROPS*, la cual almacena una lista de propiedades definidas en una clase.

Finalmente, el operador DescWhere es introducido, usando como parámetros un filtro y la estructura de los resultados. En el ejemplo a continuación, el filtro es un nombre de clase (Trabajador), y la estructura del resultado usa la variable "x" para almacenar la respuesta. La consulta en nuestro ejemplo podría ser como sigue:

```
(ALL
  :VARIABLE (Variable
    :VALUETYPE CLASS_PROPS
    :NAME x)
  :PROPOSITION (DESCWHERE
    :DESC (CLASSDESCRIPTOR
       :CLASS_PROPS
          (Variable
            :VALUETYPE CLASS_PROPS
            :NAME x))
    :WHERE (CLASSDESCRIPTOR
       :CLASS_NAME Trabajador)))
```

Usando nuestra ontología ejemplo, el resultado obtenido sería el siguiente:

```
(RESULTS :RESULTS_SET (DESCLIST
  #0 (CLASSDESCRIPTOR
    :CLASS_NAME Trabajador
    :CLASS_PROPS (PROPLIST
      #0 (PROPERTYDESCRIPTOR
            :PROP_NAME id)
      #1 (PROPERTYDESCRIPTOR
            :PROP_NAME puesto)
      #2 (PROPERTYDESCRIPTOR
            :PROP_NAME telefono)
      #3 (PROPERTYDESCRIPTOR
            :PROP_NAME oficina)
      #4 (PROPERTYDESCRIPTOR
            :PROP_NAME correo)
      #5 (PROPERTYDESCRIPTOR
            :PROP_NAME intereses)
      #6 (PROPERTYDESCRIPTOR
            :PROP_NAME nombre)))))
```

Podemos ver que la clase Trabajador agrupa en el resultado la lista de propiedades por las cuales preguntamos. Incluimos el nombre de la clase de manera que la respuesta este autocontenida, y pueda ser incorporada en la ontología local del agente cliente de manera significativa. Nótese que la lista de propiedades devuelta en este ejemplo incluye no sólo las propiedades directas de Trabajador, sino también aquellas definidas en sus superclases también.

Se deja al programador la tarea de interpretar y usar los resultados dados por las características ontológicas en nuestro prototipo. El sistema únicamente lleva a cabo automáticamente la unión de las respuestas que llegan con la ontología local, como se discute en la siguiente subsección.

Adaptación del soporte de JADE para ontologías

A partir de la versión 2.5, JADE incorpora un soporte para manejo de ontologías. Usando estas características construimos el CCO que proporciona a los agentes clientes acceso inmediato a la parte local de la ontología.

Para hacer esto, fue necesario redefinir la clase "Ontology", la cual encapsula la definición de ontología, así como la implementación de métodos de acceso para consultarla.

En JADE, los operadores de consulta pueden ser definidos usando *predicados* (PredicateSchema) y *cuantificadores* (AbsIRE). La metaontología está definida en términos de *conceptos* (AbsConcept) y *agregados* (AbsAggregate).

Otra característica de JADE es la clase *Introspector*, que permite hacer la traducción automática entre objetos Java y cadenas de texto utilizadas en mensajes entre agentes; estas últimas codificadas en *frames*.

Extensibilidad de la Ontología Local

En el prototipo logramos una integración básica CCO-AO, de manera que los resultados de las consultas son enviados al CCO, el cual los reenvía al agente, y adicionalmente incorpora estos resultados a la ontología local. Estamos aprovechando los mecanismos de Jena para unir ontologías. Cuando una respuesta llega del AO, en vez de llegar directamente al agente cliente pasa a través del CCO, permitiendo incorporar tales resultados como una extensión de la ontología base.

Como discutiremos posteriormente, la extensibilidad del CCO debe estar limitada de alguna manera, pues un crecimiento arbitrario podría sobrecargar al CCO o al menos hacerlo similar en tamaño al AO.

Caché de Consultas

Experimentamos con el uso de un *caché de consultas*, usado en tal manera que cuando un consulta entrante es idéntica a una en el caché, la respuesta es tomada inmediatamente, en vez de recalcularla. Esta es una idea muy común en sistemas computacionales, pero su desempeño, como se muestra en la siguiente sección, depende en gran medida de la distribución estadística de frecuencia de las consultas.

4 EXPERIMENTOS Y RESULTADOS

Diseñamos y llevamos a cabo experimentos enfocados a asegurar que cada posible consulta pueda ser resuelta por nuestro sistema, y esa traducción funcione adecuadamente. Asumimos que el software sobre el cual estamos construyendo

(JADE, Jena) trabaja correctamente. Llevamos a cabo una metodología de pruebas formal, ordenando primero todas las posibles consultas en una secuencia lineal, y luego tomamos aleatoriamente algunas de las consultas, hasta que un tamaño de muestra es alcanzado. Los detalles de nuestro métodos de prueba se reportan en [2].

El principal resultado de nuestros experimentos fue que el 100% de la muestra de 15 consultas fue resuelta correctamente con diferentes parámetros. Un número mayor de pruebas se consideró innecesaria, debido al 100% de los éxitos, y al alto nivel de redundancia que se hizo evidente a medida que se incrementaba la complejidad de las consultas.

A continuación, necesitamos medir el desempeño de nuestra propuesta híbrida. Consideramos una medida de *costo*, definida como el producto del tiempo y espacio computacional, como el indicador principal de eficiencia; mientras más bajo, mayor eficiencia.

Los siguientes experimentos fueron realizados por medio de un modelo matemático, el cual es descrito brevemente a continuación. Por supuesto que esta decisión tiene evidentes ventajas y desventajas comparada con experimentos del mundo real. Por una parte, los resultados son mas fáciles de explicar, así como sujetos a comprobación. Por la otra, el modelo teórico simplifica la realidad en muchas formas.

Introducimos a continuación alguna notación para estos experimentos. Notará una gran cantidad de simplificaciones; asumimos, por ejemplo, un tamaño igual en las definiciones ontológicas, tiempos iguales para la resolución de cualquier consulta, etc.

Conjuntos

- O: el conjunto de elementos ontológicos en el sistema.
- O_L: conjunto de elementos ontológicos almacenados localmente en el CCO.

Constantes

- N_a: número de agentes clientes en el sistema.
- s_E: espacio ocupado por un elemento ontológico.
- s_O: espacio ocupado por la ontología completa. $s_O = |O| \cdot s_E$.
- s_L: espacio ocupado por la ontología almacenada localmente, $s_L = R_L \cdot s_O$, ver la definición de R_L más adelante.
- t_L: tiempo requerido para resolver una consulta localmente, ya sea por un agente cliente o en el AO.
- t_T: tiempo requerido por una pregunta y su respuesta para ir al AO y volver al agente cliente.
- t_q: tiempo requerido para resolver una consulta q:

$$t_q = \begin{cases} t_L \text{ si } a \in O_L \\ t_L + t_T \text{ si } a \notin O_L \end{cases}$$

donde a es la respuesta para q.

Proporciones

- R_L: proporción de la ontología almacenada localmente, $R_L = |O|/|O_L|$.
- R_T: proporción del tiempo local y remoto para resolver consultas, $R_T = t_L/(t_L + t_T)$.

Probabilidades

- Pl_q: Probabilidad de que una consulta q pueda ser resuelta usando la ontología local.

Escenarios Extremos

El desempeño de los enfoques que estamos considerando (centralizado, distribuido, híbrido) son muy dependientes de la distribución de probabilidad de las consultas. De hecho, los enfoques distribuidos son mejores con respecto al tiempo cuando la mayoría de las consultas pueden ser resueltas localmente, pero tienden a ocupar más espacio. Esta es la razón por la cual estamos tomando como medida principal de desempeño el *costo total*, C, definido como el producto del tiempo promedio (tomado para resolver una consulta) y el espacio (tamaño total de las ontologías almacenadas).

Consideremos ahora dos casos extremos:

1. Pl_q uniforme, esto es, $Pl_q = R_L$;
2. Pl_q = 1 para cualquier q.

En el primer escenario los costos totales son:

- *Enfoque Centralizado*: El tiempo promedio es $t_L \cdot kN_a^2 + t_T$, donde k es una constante que representa el factor de degradación del desempeño en el AO cuando el número de clientes crece. El espacio total es s_O, y $C = (t_L \cdot kN_a^2 + t_T) \cdot s_O$. Este enfoque no se ve afectado por la distribución de probabilidad.
- *Enfoque Distribuido*: el tiempo promedio es t_L, asumiendo que los agentes trabajan en paralelo. El espacio total es $N_a \cdot s_O$ (no hay AO), y el costo es $C = t_L \cdot N_a \cdot s_O$.
- *Enfoque Híbrido*: el tiempo promedio es un punto medio entre los otros dos enfoques:

$$R_L \cdot t_L + (1 - R_L) \cdot (t_L \cdot kN_a^2 + t_T)$$

El espacio total es $s_O \cdot (1 + R_L \cdot N_a)$, y el costo es

$$[R_L \cdot t_L + (1 - R_L) \cdot (t_L \cdot kN_a^2 + t_T)] \cdot [s_O \cdot (1 + R_L \cdot N_a)]$$

Table 2: Costos para n agentes, primer escenario.

n	*Central.*	*Distrib.*	*Híbrido*
1	2000	1000	2090
10	101000	10000	182000
100	10001000	100000	99011000

La Tabla 2 muestra valores numéricos usando los parámetros: $|O| = 1000$, $s_E = 1$, $R_L = 0.1$ (para el híbrido), $t_L = 1$, $t_T = 2$, $k = 1$, y variando el número de agentes. Los parámetros específicos no son críticos para los resultados, como verificamos en experimentos no reportados aquí. Como se puede observar en la tabla, los valores de desempeño del enfoque híbrido son los peores de los tres enfoques, porque sufre de pobres tiempos de resolución del enfoque centralizado el 90 por ciento de las veces, y sufre también de una penalización por espacio en almacenamiento local.

Ahora tomemos el segundo caso, donde cada consulta puede ser resuelta localmente en el enfoque híbrido. Esto no parece ser realista pues significa que la parte de la ontología no almacenada localmente es inútil, pero representa un límite para aquellos casos donde las consultas remotas son muy raras.

Para el enfoque centralizado y distribuido los costos son idénticos al caso anterior. Para el enfoque híbrido los costos totales son:

El tiempo promedio es t_L. El espacio total es

$$s_O \cdot R_L \cdot N_a + s_O \cdot (1 - R_L)$$

y el costo es $C = t_L \cdot [s_O \cdot R_L \cdot N_a + s_O \cdot (1 - R_L)]$.

Table 3: Costos para n agentes, segundo escenario.

n	*Central.*	*Distrib.*	*Híbrido*
1	2000	1000	1100
10	101000	10000	2000
100	10001000	100000	11000

En la tabla 3 podemos ver ahora que el enfoque híbrido claramente supera el desempeño tanto del enfoque centralizado como del distribuido por mucho. Esto no debe sorprendernos, pues el segundo escenario es la situación ideal para el enfoque híbrido. El enfoque centralizado esta altamente penalizado por el costo de acceso, mientras que el distribuido está penalizado por altos costos de almacenamiento.

Otras distribuciones de probabilidad son simplemente puntos intermedios entre los dos casos extremos que hemos considerado, y por esto los resultados deben ser interpolados. Mientras más concentrada esté la probabilidad en el almacenamiento local, más ventajoso es el enfoque híbrido.

También probamos el uso de un caché de consultas, y los resultados fueron muy similares a los anteriormente descritos. La única diferencia fue que los enfoques distribuido e híbrido redujeron el tiempo de resolución de consultas de t_L a 0 (asumimos que el costo de acceder al caché fuera cero).

5 TRABAJOS RELACIONADOS

En el proyecto KAON [20] el énfasis es en reusar ontologías y en propagar los cambios a ontologías distribuidas. Se lleva un registro de las "URI" de las ontologías conocidas para el sistema en unos "servidores de ontologías", los cuales toman a su cargo administrar la evolución de las ontologías (inclusion, actualización, cambios, propagación, etc.). Cada servidor de ontologías provee un servicio de interrogación a su comunidad de agentes. En el servidor de ontologías se guarda una copia local de la ontología original, tomada inicialmente del URI dado, y actualizado en curso de funcionamiento, para propósitos de interrogación.

Nuestro AO, en cambio, utiliza ontologías DAML+OIL. En nuestro enfoque, es el CCO quien actualiza su ontología, no el servidor (esto es, el AO). En el lado negativo, nosotros no hemos tomado en cuenta aun el aspecto de la evolución de las ontologías, principalmente porque esto acarrea problemas de mantener la coherencia y detectar cuando se ha perdido esta.

En el proyecto COMMA [18], como en JITIK, se tiene una ontología global que es propagada al conjunto de agentes conocidos. Cada agente recibe una copia completa de la ontología, y puede resolver consultas por si mismo. COMMA usa RDF como el lenguaje de codificación de ontologías. Incluye un API especializado en el acceso a ontologías por los agentes del sistema, y hay un "archivista de ontologías" que administra el acceso. Obviamente este enfoque se ubica en el extremo centralizado del espectro.

En el sistema FRODO [19] se consideran los roles de "proveedor" y "consumidor" de ontologías. En los proveedores se concentra la provisión de servicios de ontologías, incluyendo su mantenimiento. Los consumidores simplemente usan las ontologías para ejecutar sus aplicaciones. En FRODO hay dos niveles de distribución de ontologías: interno a una comunidad de agentes y externo o "intersistemas". En JITIK tenemos solamente un nivel de distribución de ontologías, pues aun no hemos considerado la comunicación entre agentes de ontologías de dos grupos de agentes distintos, cada uno alrededor de un agente de sitio. En FRODO se definen tres categorías respecto a las capacidades de manejo de ontologías: *uso* de ontologías, *evolución* de ontologías y *socialización* de ontologías. En el *uso* de ontologías se encuentra la formulación de consultas y su solución. En JITIK consideramos el uso y la socialización de las ontologías, pero no la evolución.

6 DISCUSIÓN

La aplicabilidad de nuestro enfoque, como lo sugieren los experimentos presentados, es altamente dependiente de la distribución de probabilidad de las consultas con respecto a si pueden o no ser resueltas usando la parte local de la ontología. Debe haber una clara tendencia a resolver la mayoría de las consultas usando sólo la parte local de la ontología (o las consultas en caché) para que el enfoque híbrido sea provechoso. Está por verse si este es el caso en aplicaciones del mundo real (ver sección de trabajos futuros).

Ahora bien, aparte del prototipo, la arquitectura propuesta y el prototipo exploran opciones tecnológicas ligeramente diferentes, dando de esta forma un rango de posibles soluciones para sistemas específicos. La arquitectura conceptual ilustrada en la figura 2 usa almacenamiento persistente explícito, así como una separación entre esquema e instancias. Esto puede ser preferible sobre esquemas más homogéneos como Jena en el caso de un número de instancias extremadamente grande, debido a que podemos aprovechar consultas más eficientes a bases de datos, en lugar de mecanismos especializados de inferencia sobre ontologías. Nuestro prototipo no usa formas de almacenamiento persistente, a pesar de que Jena ha ofrecido recientemente soporte a persistencia. De esta forma, incorporar persistencia es cuestión de actualizar la versión de Jena. A pesar de ello pensamos que la persistencia no es esencial para el CCO en el lado del agente cliente; el cliente podría cargar la ontología base como se hace en el prototipo, y obtener definiciones adicionales del almacenamiento persistente en el AO como se explicó arriba.

7 CONCLUSIONES

Hemos presentado una arquitectura y un prototipo que resuelve el problema de manejo de ontologías para el sistema JITIK, y que puede ser aplicado a otros sistemas multiagente también.

El principal requerimiento para aplicar nuestra arquitectura es que debe haber una ontología común, la cual es en un principio aceptada por todos en el sistema, pero que no es completamente conocida por cada agente en el sistema. Asi, proponemos una forma de compartir el conocimiento de la ontología común residente en el Agente de Ontologías, pero evitar los cuellos de botella que podrían resultar de un manejo de ontologías centralizado. Por esto, hemos incorporado a todos los agentes en el sistema un Componente Cliente de la Ontología, el cual es capaz de resolver localmente parte de la consultas ontológicas.

Se reporta un prototipo, el cual implementa los elementos básicos de nuestra arquitectura, haciendo uso extensivo de Jena Toolkit. Se desarrolló una librería (xont) que encapsula toda la funcionalidad adicional requerida para cargar y consultar ontologías DAML+OIL desde JADE.

Presentamos una serie de experimentos usando un modelo matemático simple, mostrando que en ciertas condiciones - despreciando la distribución de probabilidad de las consultas - nuestro enfoque híbrido tiene costos espacio-temporales más bajos comparados con los enfoques centralizado y distribuidos puros.

Pensamos que un enfoque híbrido introduce la posibilidad de afinar el compromiso entre acceso central y distribuido a la ontología, básicamente variando el tamaño de la ontología local. En un extremo, una ontología vacía en el CCO es equivalente a una solución centralizada, mientras que una ontología en el CCO idéntica a la del AO equivale a una solución completamente descentralizada. Cualquier solución intermedia es posible en principio.

Trabajo Futuro

Para optimizar la cantidad de conocimiento ontológico distribuido en los agentes clientes vislumbramos dos áreas de oportunidad: la conformación de la ontología base y la cantidad de conocimiento ontológico incorporado a la ontología local.

El primer punto es crítico en sistemas donde la ontología general es demasiado grande. En ellos se puede genera una serie de ontologías base sobre clusters de términos aplicando un método de poda como el descrito en [21]. Análisis estadísticos en el CCO permitirían solicitar una ontología base afín.

En segundo lugar, para evitar la sobrecarga del CCO se puede mantener un "caché" de las definiciones más frecuentemente usadas, reemplazando eventualmente las menos usadas.

Por último, es importante probar nuestro método en un amplio rango de escenarios reales multiagente de conocimiento intensivo, de forma que el ajuste fino global-local que mencionamos antes pueda ser puesto en práctica.

REFERENCES

[1] Bellfemine, Fabio; et alt. **JADE Administrator's Guide.** TILAB S.p.a. 2002. `http://sharon.cselt.it/projects/jade/`

[2] Ceballos, H., **Manejo de Ontologías en Sistemas Multiagentes por medio de un Agente de Ontologías aplicado a JITIK**, master thesis, Monterrey Tech, June 2003.

[3] H. S. Nwana & D. T. Ndumu, "A Perspective on Software Agents Research", **The Knowledge Engineering Review**, Vol 14, No 2, pp 1-18, 1999.

[4] M. Wooldridge, **Introduction to MultiAgent Systems**, John Wiley and Sons in March 2002.

[5] R. Brena, J.L. Aguirre, A.C. Treviño, "Just-in-Time Knowledge Flow for Distributed Organizations using agents technology", **Knowledge Technologies 2001 Conference**, Austin, Texas, 4-7 March 2001, `http://www2.gca.org/knowledgetechnologies/2001/proceedings/index.asp`.

[6] F. Weichhardt, C. Fillies, R. Smith, **The Semantic Web is the Database: Decentralised Modeling with central Coordination**, 2002, `http://www.semtalk.com/pub/spain2.htm`.

[7] Jaron Collis, Divine Ndumu. **The Zeus Agent Building Toolkit. The Role Modelling Guide.** British Telecommunications plc. 1999

[8] HP Labs. **Jena Semantic Web Toolkit - Data Sheet** `http://www.hpl.hp.com/semweb/jena-datasheet.htm`.

[9] Alexaki, Sofia; Athanis, Nikos; Vassilis, Christophides; Karvounarakis, Greg; Maganaraki, Aimillia; Plexousakis, Dimitris. "The ICS-FORTH RDFSuite: High level scalable Tools for the Semantic Web". Poster Session of the **Eleventh International World Wide Web Conference (WWW'02)**. Honolulu, Hawaii, USA, May 8, 2002.

[10] Caire, Giovanni. **JADE Tutorial. Application-Defined Content Languages and Ontologies.** TILab S.p.a. Grecia. 2002. `http://sharon.cselt.it/projects/jade/`

[11] Kifer, Michael; Lausen, Georg. "F-logic: a higher-order language for reasoning about objects, inheritance, and scheme". **SIGMOD RECORD**, Vol: 18, No: 6, June 1990, pp. 134-146 . `citeseer.nj.nec.com/kifer90flogic.html`

[12] Conolly, Dan; et alt. **DAML+OIL (March 2001) Reference Description.** W3C Note 18 Dec 2001. `http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218`

[13] Lassila, Ora; R. Swick, Ralph. **Resource Description Framework (RDF) Model and Syntax Definition Specification.** W3C Recommendation, Feb 1999. `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222`

[14] Mike Dean, Guus Schreiber, et. alt. **OWL Web Ontology Language Reference.** W3C Working Draft, 31 March 2003. http://www.w3.org/TR/2003/WD-owl-ref-20030331/

[15] G. Karvounarakis¸ S. Alexaki¸ V. Christophides¸ D. Plexousakis¸ M. Scholl¸ "RQL: A Declarative Query Language for RDF"¸ In **The 11th Intl. World Wide Web Conference (WWW2002)**¸ citeseer.nj.nec.com/556066.html.

[16] Lenat, D., "Context dependence of representations in CYC". **Colloque ICO'93**, Montreal, Canada, May 1993.

[17] Bechhofer, S.¸ Horrocks, I.¸ Goble, C.¸ Stevens, R., "OilEd: A Reason-able Ontology Editor for the Semantic Web", **24th German / 9th Austrian Conference on Artificial Intelligence**, 2001.

[18] Gandon, F., "Agents handling annotation distribution in a corporate semantic web", **Web Intelligence and Agent Systems**. Volume 1, Number 1. 2003. OIS Press.

[19] van Elst, L., Abecker, A. "Domain Ontology Agents in Distributed Organizational Memories", **Knowledge Management and Organizational Memories**, Rose Dieng-Kuntz and Nada Matta (eds.), Kluwer Academic Publishers, Julio 2002.

[20] Maedche, A., Motik B., Stojanovic L., Studer R., Volz R. "An Infrastructure for Searching, Reusing and Evolving Distributed Ontologies", **WWW2003 proceedings**, ACM, Budapest, Hungria, Mayo 20-24, 2003.

[21] Conesa, J., Olivé, A., "A General Method for Pruning OWL Ontologies". **CoopIS/DOA/ODBASE. Lecture Notes in Computer Science**. Vol 3291. Springer Verlag. 2004. pp 981-998.