Incorporación de un Agente que Razona en Base a Casos en JITK

TESIS

MAESTRO EN CIENCIAS
EN SISTEMAS INTELIGENTES

INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY

Por:
Ing. Omar Montaño Rivas

DICIEMBRE DE 2003

Incorporación de un Agente que Razona en Base a Casos en JITIK

por

Ing. Omar Montaño Rivas

Tesis

Presentada al Programa de Graduados en Electrónica, Computación, Información y Comunicaciones

como requisito parcial para obtener el grado académico de

Maestro en Ciencias

especialidad en

Sistemas Inteligentes

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Diciembre de 2003

Incorporación de un Agente que Razona en Base a Casos en JITIK

por

Ing. Omar Montaño Rivas

Tesis

Presentada al Programa de Graduados en Electrónica, Computación, Información y Comunicaciones

como requisito parcial para obtener el grado académico de

Maestro en Ciencias

especialidad en

Sistemas Inteligentes

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Diciembre de 2003

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

División de Electrónica, Computación, Información y Comunicaciones

Programa de Graduados

Los miembros del comité de tesis recomendamos que la presente tesis de Omar Montaño Rivas sea aceptada como requisito parcial para obtener el grado académico de **Maestro en Ciencias**, especialidad en:

Sistemas Inteligentes

Comité de tesis:

Dr. José Luis Aguirre Cervantes
Asesor de la tesis

Dr. Ramón Felipe Brena Pinero
Sinodal

Dr. Leonardo Garrido Luna
Sinodal

Dr. David Garza Salazar
Director del Programa de Graduados

Diciembre de 2003

A mi querida madre y a mi admirable hermano.

Reconocimientos

A mi familia por su amor incondicional.
Agradezco a mi asesor, el Dr. José Luis, por su guía en la realización de esta tesis.
Agradezco a mis sinodales, el Dr. Brena y al Dr. Leonardo, por sus valiosas aportaciones y orientación enriquecedora.
Igualmente, agradezco a todos aquellos con quien conviví por su amistad y companerismo.

OMAR MONTAÑO RIVAS

Instituto Tecnológico y de Estudios Superiores de Monterrey
Mayo 2003

Incorporación de un Agente que Razona en Base a Casos en JITIK

Omar Montaño Rivas, M.C.
Instituto Tecnológico y de Estudios Superiores de Monterrey, 2003

Asesor de la tesis: Dr. José Luis Aguirre Cervantes

En este documento se presenta la propuesta de tesis para obtener el grado de Maestro en Ciencias en Sistemas Inteligentes en el Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey. Esta propuesta tiene como objetivo aplicar la tecnología del *Razonamiento Basado en Casos* en un *agente* incorporado al sistema *JITIK* ("Just-in-time" Information and Knowledge) que es un sistema que utiliza tecnologías de *Agentes Inteligentes* para apoyar la difusión de información y conocimientos relevantes a personas que se encuentran en una organización distribuida.

"El *Razonamiento Basado en Casos* es una tecnología, relativamente nueva, de inteligencia artificial para la construcción de sistemas basados en el conocimiento reutilizando experiencias (casos)" [16]. Este nuevo paradigma ha venido emergiendo rápidamente y lo demuestra el gran número de proyectos y aplicaciones que hasta ahora se han desarrollado. La incorporación de este agente permitiría mejorar el manejo y distribución del conocimiento o información a un conjunto de usuarios, en la medida que se reutilicen "experiencias" pasadas para la solución de nuevos problemas en una organización.

En el documento primeramente se introduce a la problemática en torno a la administración y distribución de experiencias y conocimiento en organizaciones distribuidas. Asimismo, se describen los conceptos más importantes que envuelven este trabajo como por ejemplo: agentes inteligentes, sistemas multiagentes, razonamiento basado en casos. Se presentan algunos trabajos relacionados con los conceptos anteriores y además se dan todos los detalles concernientes a la construcción del *agente RBC* así como también su integración dentro de JITIK.

Finalmente se describen las pruebas realizadas al *agente RBC* que nos aseguran el funcionamiento adecuado y un rendimiento aceptable. Las conclusiones de este trabajo son presentadas en el capítulo 5.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Antecedentes

Las tecnologías de los *Agentes Inteligentes* y los *Sistemas Multiagentes* son áreas de investigación muy activa en los años recientes. La razón de ello es que los sistemas basados en agentes proponen interesantes oportunidades para crear sistemas computacionales más flexibles y robustos que los sistemas tradicionales. Estas tecnologías surgidas de la interacción de varias disciplinas, como la Inteligencia Artificial, los Sistemas Distribuidos de Software, y hasta las Ciencias Sociales, se aplican muy naturalmente a organizaciones humanas distribuidas geográfica o lógicamente. Es por esto que el grupo de investigación en AI del *Centro de Sistemas Inteligentes* (CSI) del Campus Monterrey del *ITESM*, se propuso llevar adelante un proyecto de amplitud, orientado a la aplicación innovadora de tecnologías de AI en organizaciones distribuidas. Este proyecto es conocido actualmente con el nombre *JITIK* ("Just-in-time" Information and Knowledge) y es dirigido por el Dr. José Luis Aguirre y el Dr. Ramón Brena del CIA. La primera presentación descriptiva del proyecto fue en 1998, llamada en ese entonces *CORREA* [21] (COordinación de Recursos de Educación e Investigación mediante Agentes). En 1999 se le cambió el nombre a *RICA* [20] (Redes de Información y Conocimiento mediante Agentes) para enfatizar el aspecto de la interconexión de redes de usuarios y servicios, que no aparecían en forma explícita en el nombre CORREA. Finalmente en 2001 se optó por llamar al proyecto *JITIK* [22] para remarcar que el conocimiento adecuado es conectado con la persona adecuada en el momento adecuado.

Actualmente la mayoría de las organizaciones se administran con el apoyo de una base de datos que manejan archivos con gigabytes de información y realizan millones de transacciones diarias. Resulta de gran importancia descubrir y sacar provecho a esos grandes volúmenes de información. A esta información clave usualmente se le llama *conocimiento* [25]. Sin embargo, el manejo del conocimiento que se lleva a cabo en la mayoría de las organizaciones es deficiente. Las principales deficiencias en este manejo son:

- Las personas pierden mucho tiempo buscando información
- El conocimiento esta disponible solo en la cabeza de pocas personas

- Información valiosa se encuentra olvidada en pilas de documentos y datos
- Son cometidos repetidamente errores costosos debido a que no se toman en cuenta experiencias previas
- Retraso y calidad subóptima de producto resulta de un insuficiente flujo de información

El manejo del *conocimiento* como recurso en una organización requiere capturar y retener información útil y mantenerla disponible en una forma utilizable cuando se necesite en el futuro. Este proceso es complicado por dificultades en la obtención y representación del conocimiento, en el acceso de información relevante, y en adaptar lecciones previas a nuevas situaciones. Diferentes tecnologías ofrecen distintas alternativas para resolver estos problemas, y una de estas es el *Razonamiento Basado en Casos* (RBC). El *Razonamiento Basado en Casos* ha sido estudiado, de manera creciente, como una técnica para el manejo del conocimiento para ayudar en la recuperación y adaptación de casos pasados.

"Los sistemas basados en casos operan en forma diferente a los ya bien conocidos sistemas expertos basados en reglas. Mientras que estos últimos sistemas toman una especificación de entrada y obtienen una solución encadenando un conjunto de reglas, los sistemas de *Razonamiento Basado en Casos* toman un caso como entrada y buscan en su memoria por uno ya existente que encaje con el caso dado. Si se encuentra un estado que encaja perfectamente con la entrada, se llega directamente a la solución. Si no es así, se obtiene un caso similar y se sigue por un proceso de adaptación, generando una solución y un nuevo caso que puede ser aprendido por el sistema" [16].

Si hacemos una comparativa de la tecnología de *Razonamiento Basado en Casos* y los *Sistemas Expertos* basados en reglas (esta comparativa no es el objetivo de la investigación), estos últimos tienen ciertos puntos débiles: el cuello de botella de la adquisición del conocimiento, la falta de memoria, la falta de aprendizaje y la falta de robustez [16].

"Los sistemas de Razonamiento Basados en Casos prometen resolver estos problemas de manera limpia y natural, de una forma semejante al razonamiento humano. Los procesos de adaptación y aprendizaje son punto clave en la arquitectura de estos sistemas" [16]. Es precisamente debido a la creciente importancia de los sistemas de *Razonamiento Basado en Casos* y a su rápida proliferación y aceptación, que se hace necesario emplearlos a problemas que hemos venido enfrentando en años pasados como lo es la distribución, manejo, y clasificación de la información y/o conocimiento.

1.1. Definición del problema

El valor de una organización no es dado sólo por sus bienes tangibles o físicos, sino también por el conocimiento contenido en su personal y en su organización interna y externa.

El aprendizaje en organizaciones se centra en almacenar las lecciones aprendidas en algún repositorio de información, de manera que estas puedan ser recuperadas y usadas cuando se necesiten. Actualmente la inclusión de estas "experiencias" a las bases de datos se le denomina como *memoria de una organización.*

La *memoria de una organización* puede ser definida como "una representación explícita, no material, y persistente del conocimiento e información de una organización" [12]. Cualquier pieza de conocimiento e información que contribuya al funcionamiento de una organización debería ser almacenada en la *memoria.* Conocimiento sobre productos, procesos de producción, clientes, estrategias de mercadeo, resultados financieros, lecciones aprendidas, planes estratégicos y metas, entre otros, podrían formar parte de la *memoria de una organización.*

El Departamento de Energía de los Estados Unidos analiza sus lecciones aprendidas combinándolas con conocimiento en la *memoria* y después, las envían a los trabajadores para los que las lecciones aprendidas puedan ser relevantes. De hecho, JITIK nació de la idea de proporcionar la pieza de conocimiento correcto a la persona adecuada y en el momento preciso. Un ejemplo claro de como ayudaría la memoria en el ITESM puede ser que un catedrático del Centro de Sistemas Inteligentes (CSI) imparte un curso rediseñado en el Campus Monterrey utilizando algún esquema de trabajo innovador. Al final del curso se da cuenta que la mayoría de sus alumnos terminan con calificaciones bajas y pocos conocimientos. Esta información o caso es de suma importancia para evitar futuras fallas dentro o fuera del Campus.

Existen tres tipos de esquemas de representación de conocimiento para ser usados en la *memoria de la organización,* en donde los principales son: Model-Based Reasoning (estructuras de conceptos y métodos de inferencia), Rule-Based Reasoning (conocimiento organizado dentro de reglas condicionales if-then), y Case-Based Reasoning (captura conocimiento directamente de experiencias).

En la actualidad la mayoría de las organizaciones se administran con el apoyo de una base de datos que manejan archivos con gigabytes de información y realizan millones de transacciones diariamente. Estos archivos por lo general tienen reportes, artículos, manuales, patentes, fotografías, imágenes, video, sonido, software, entre otros; pero pocas veces contienen información sobre "experiencias" o casos que ayuden a solventar problemas futuros. *Son cometidos repetidamente errores costosos en organizaciones debido a que no se toman en cuenta experiencias previas.* El hecho de que el paradigma del *Razonamiento Basado en Casos* (Case-Based Reasoning) tiene la cualidad de capturar conocimiento directamente de experiencias, sugiere una disminución o

incluso la eliminación de los errores debidos a que no se toman en cuenta experiencias pasadas.

Recientemente se ha investigado mucho el área del Razonamiento Basado en Casos (RBC) para solucionar estos tipos de problemas, como lo demuestran los numerosos artículos, libros y aplicaciones en el área. Pero no se ha implementado un agente que Razone en Base a Casos con la finalidad de conectar una "herramienta de conocimiento" (Razonamiento Basado en Casos) con una "herramienta de almacenamiento" (Repositorios o Librerías de Casos[1]) para enfrentar los problemas descritos anteriormente asociados con el flujo de conocimiento en una organización.

La incorporación de un agente que Razone en Base a Casos a *JITIK* permitiría incorporar servicios para el manejo del conocimiento contenido en repositorios de casos.

1.2. Objetivos

1.2.1. Objetivos Generales

Conectar una "herramienta de conocimiento" (Razonamiento Basado en Casos), con una "herramienta de almacenamiento" (Repositorio de Casos) para desarrollar un agente que utilice la tecnología del *Razonamiento Basado en Casos* y de esta manera proporcionar un servicio de flujo de conocimiento contenido en una librería de casos a *JITIK*.

1.2.2. Objetivos Particulares

- Diseñar y implementar un *Sistema RBC* que se encargue de la fase de recuperación [2] dentro del paradigma del *Razonamiento Basado en casos*. Del mismo modo, diseñar un esquema que permita una futura inclusión de módulos que se encarguen de las fases de modificación (adaptación, evaluación, y reparación[3]) y aprendizaje (almacenamiento) según el paradigma del *Razonamiento Basado en Casos*.

- Diseñar, implementar, e integrar el *agente RBC* así como también las comunicaciones con el *agente de sitio* y el *agente de ontologías*, de manera que pueda utilizar el *Sistema RBC* mencionado anteriormente para proporcionar los siguientes servicios a *JITIK*:

  1. Procesar consultas realizadas por el *agente de sitio* a la librería de casos utilizando algún mecanismo de indexación.

[1]En esta investigación se le llamará indistintamente Repositorio de Casos o Librería de Casos a una fuente de información que use el Modelo de Datos Relacional (ver apéndice B)

[2]La descripción de la fase de acceso a casos es explicada en la sección 2.6.1 página 14.

[3]La fase de adaptación, evaluación, y reparación es explicada en las secciones 2.6.3 a la 2.6.5

2. Procesar consultas realizadas por el *agente de sitio* utilizando un mecanismo que mapée un perfil de intereses de un *agente personal* a casos dentro de la librería de casos. Este perfil de intereses estará contenido dentro de una ontología administrada por el *agente de ontologías* y será accedida utilizando el lenguaje de consulta especificado en [5].
3. Monitorear cualquier modificación que se llevase a cabo en la librería de casos y comunicarlo al *agente de sitio.*

1.2.3. Alcances

Como resultado del desarrollo de la presente investigación se obtendrá como producto un agente incorporado a la arquitectura de JITIK que cumpla con los estándares de plataformas de desarrollo de sistemas multiagentes vigentes y estables, y que utilice la tecnología de *Razonamiento Basado en Casos* para proporcionar los servicios mencionados en la sección anterior (1.2.2). Dicho agente será sometido a pruebas y evaluaciones para asegurar el correcto funcionamiento en un ambiente simulado.

1.3. Hipótesis

Este trabajo pretende conectar la "herramienta de conocimiento" del *Razonamiento Basado en Casos*, con una "herramienta de almacenamiento" (*Repositorio de Casos*) mediante un agente inteligente incorporado a la arquitectura de *JITIK*.

Las "preguntas metodológicas", que tienen como finalidad el guiar nuestra investigación, son planteadas a continuación:

1. ¿Qué arquitecturas del Manejo y Distribución de la Información existen actualmente?
2. ¿Alguna de estas arquitecturas maneja el paradigma del *Razonamiento Basado en Casos*?
3. ¿Cómo se han venido construyendo los sistemas RBC?
4. ¿Cómo podemos incorporar un agente que Razone en Base a Casos a *JITIK*?
5. ¿Cómo podríamos diseñar un Repositorio de Casos?
6. ¿Cómo podría monitorear este agente al Repositorio de Casos de manera que comunique a *JITIK* todas las modificaciones realizadas?
7. ¿Cómo podría llevarse a cabo el mecanismo de extracción de conocimiento o información relevante al repositorio de casos?

Capítulo 2

Marco teórico

En este capítulo se da una breve reseña de los conceptos más importantes en que se desenvuelve la presente tesis así como también algunos trabajos relacionados con la misma. Primeramente se dará una noción sobre los agentes inteligentes y sistemas multiagentes, después se describirá al *Sistema JITIK* así como también el sistema de software en donde fue programado (JADE), y por último se explicará el paradigma del *Razonamiento Basado en Casos.* Asimismo, al final del capítulo se presentan algunos trabajos relacionados con el Razonamiento Basado en Casos, el manejo de la información y conocimiento, y/o la tecnología de agentes inteligentes y sistemas multiagentes.

2.1. Tecnología de Agentes Inteligentes y Sistemas Multiagentes

Un agente es todo aquello que puede considerarse que *percibe* su ambiente mediante *sensores* y que *responde* o *actúa* en tal ambiente por medio de *efectores* [23]. Los agentes humanos tienen ojos, oídos, y otros órganos que les sirven de sensores, así como piernas, manos, boca y otras partes de su cuerpo que les sirven de efectores. En el casos de un agente de software, sus percepciones y acciones vienen a ser cadenas de bits codificados.

Además de cumplir con la definición anterior, un agente inteligente debe cumplir con ciertos requisitos extra, como la reactividad (que el agente pueda responder a los estímulos a tiempo para dar una respuesta efectiva), proactividad (el agente presenta comportamiento dirigido a metas y toma la iniciativa para lograr sus objetivos), y sociabilidad (los agentes inteligentes son capaces de interactuar con otros agentes para satisfacer sus objetivos).

La tecnología de sistemas basados en agentes ha generado gran emoción en años recientes por prometer un nuevo paradigma para la conceptualización, diseño e implementación de sistemas de software. Estos sistemas de software pueden operar en sistemas abiertos como internet, lo cual es particularmente atractivo. Actualmente, la gran mayoría de los sistemas basados en agentes consisten en un único agente. Sin embargo, mientras la tecnología evoluciona enfrentándose a aplicaciones cada vez más

complejas, la necesidad de sistemas que consisten en múltiples agentes que interactúen unos con otros se está haciendo aparente.

La capacidad de un agente inteligente está limitada por su conocimiento, sus recursos de cómputo, y su perspectiva. Esta limitada racionalidad es una de las razones fundamentales para la creación de organizaciones que resuelvan problemas. Las herramientas más poderosas para el manejo de problemas complejos son la modularidad y abstracción. Los *Sistemas Multiagentes* ofrecen modularidad. Si el dominio de un problema es complejo, largo e impredecible, entonces la única manera de ser enfrentado es desarrollar componentes modulares con funcionalidad específica (agentes) que se especializan en resolver un problema particular. Cuando surgen problemas interdependientes, los agentes en el sistema deben cooperar entre si, y esta es la base de los *Sistemas Multiagentes* [9].

2.2. Sistema JITIK

Muchas grandes organizaciones distribuidas sufren de deficiencias en la circulación del conocimiento, ya que frecuentemente éste es creado en un lugar y necesitado (a menudo no usado) en otro. Una buena difusión de piezas de información relevante y conocimiento, es muy importante en organizaciones amplias y/o distribuidas. *JITIK* nace de la necesidad de dar apoyo o respaldo a la circulación del conocimiento conectando el conocimiento adecuado a la persona adecuada en el momento adecuado [22].

JITIK hace uso de la tecnología de *Agentes Inteligentes* y los *Sistemas Multiagentes* para monitorear la información de manera autónoma. De esta manera los usuarios delegan tareas a los agentes y estos se encargan de realizarla y de reportar resultados al usuario. En *JITIK* el conocimiento es difundido a los usuarios tan rápido éste esté disponible y sea relevante a estos usuarios. La figura 2.1 muestra la arquitectura de *JITIK*.

Para la realización de los servicios de *JITIK*, se consideró que era necesario que el sistema siguiera trabajando aún en ausencia del usuario y hasta de la computadora personal de éste, pues varios de los servicios incluyen el monitoreo continuo de eventos de interés, tales como la modificación de páginas Web y otros. Desde el punto de vista arquitectural esto implica que se necesita tener en operación continua el sistema, lo cual motivó a la decisión de ubicar los procesos de *JITIK* en "servidores" computacionales, esto es, máquinas de poder mediano a grande que no se apagan o desconectan nunca (o casi nunca).

Así, los usuarios individuales se conectan a un servidor de *JITIK* por medio de su computadora personal, la cual corre un software de interfaz con el software en el servidor, y al establecer comunicación, reporta al usuario eventos de interés y le permite manipular sus datos de registro en el sistema, tales como sus áreas académicas de interés.

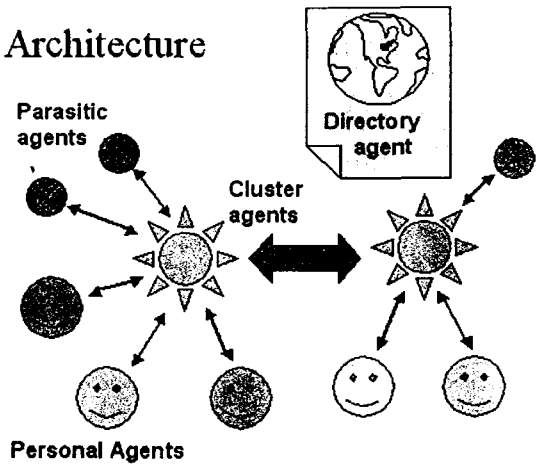


Figura 2.1: Arquitectura de JITIK

Este modelo se puede ver en la figura 2.2 y muestra la distribución de los diferentes agentes que conforman *JITIK*.

El agente *"Site"* es el principal o central y puede encontrarse caracterizado en un departamento o área, por lo que puede haber más de un agente *Site* ya sea de forma local o remota. El agente *"Ontology"* puede ser uno por varios agentes de *Site* y contiene la jerarquización de la información y de los usuarios (conocimiento sobre la organización, las áreas de interés, los miembros de la organización junto con sus perfiles, etc.). Los *"Directory agents"* ayudan a encontrar a otros agentes y dan una gran flexibilidad a la (re)configuración del sistema. Los *"Personal agents"* están encargados de filtrar la información que le hace llegar el *agente de sitio* para determinar si se la presenta a su usuario. El agente *"Bridge"* es el encargado de la integración de procesos internos de la organización. Esta integración se hace por medio de un *"Link"* y puede haber n agentes *"Bridge"* por sitio. Los *"Parasitic agents"* permiten la obtención de información de otros sistemas. Estos agentes están "atados" a programas convencionales.

En cuanto a su implementación, el *Sistema JITIK* utiliza tecnología de información muy avanzada. Además de utilizar las tecnologías modernas de internet, *JITIK* hace uso de métodos surgidos de la Inteligencia Artificial, tales como sistemas expertos, inferencia automática, entre otros. En la actualidad con una versión del sistema programada en JADE, que es un software orientado a agentes programado en java. Además del *agente de sitio* [6], el *agente de ontologías* [5] y una primera versión del *agente personal*, ya ha sido implementado el agente monitor web.

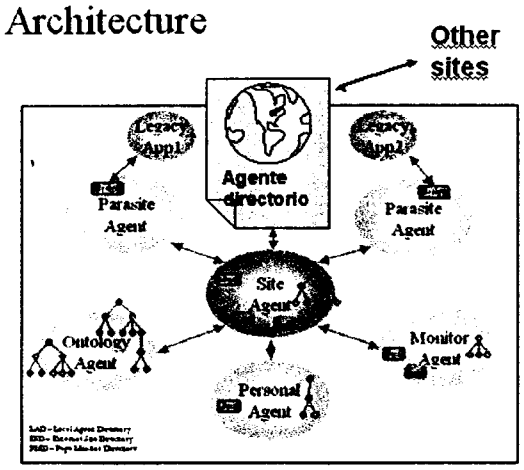


Figura 2.2: Arquitectura detallada de JITIK

2.3. JADE (Java Agent DEvelopment Framework)

JADE (Java Agent DEvelopment Framework) [7] es un software hecho completamente en Java. *JADE* simplifica la implementación de sistemas multiagente usando las especificaciones de *FIPA* (Foundation for Intelligent Physical Agents) por medio de un conjunto de herramientas que soportan el depuramiento. La plataforma de agente puede ser distribuida a través de máquinas (que no necesitan tener el mismo sistema operativo) y la configuración puede ser controlada remotamente. La configuración puede ser incluso cambiada en tiempo de ejecución moviendo agentes de una máquina a otra, cuando sea requerido. El único requerimiento del sistema es el "Java Runtime" versión 1.2.

La plataforma de agentes de JADE aparte de cumplir con las especificaciones de FIPA, incluye todos los componentes básicos para administrar la plataforma, esto es el RMA (Remote Monitoring Agent), AMS (Agent Management Service) y el DF (Directory Facilitator). Toda la comunicación de agente es realizada a través de la traducción de mensajes, donde FIPA ACL es el lenguaje para representar mensajes. La plataforma de agentes puede distribuirse en varios "hosts" o servidores. Sólo una aplicación de Java, y por lo tanto una Máquina Virtual de Java, es ejecutada en cada "host".

Cada agente de JADE está compuesto por un hilo de ejecución dentro de la máquina virtual, pues los agentes frecuentemente necesitan ejecutar tareas paralelas. Además de la solución "multihilo" de java, JADE soporta también la programación de comportamientos cooperativos (clase de java `Behaviour`), donde jade organiza esas

tareas de forma ligera y eficiente. La figura 2.3 muestra en formato UML (Unified Modeling Language) los comportamientos que proporciona JADE así como también la descripción de los mismos.

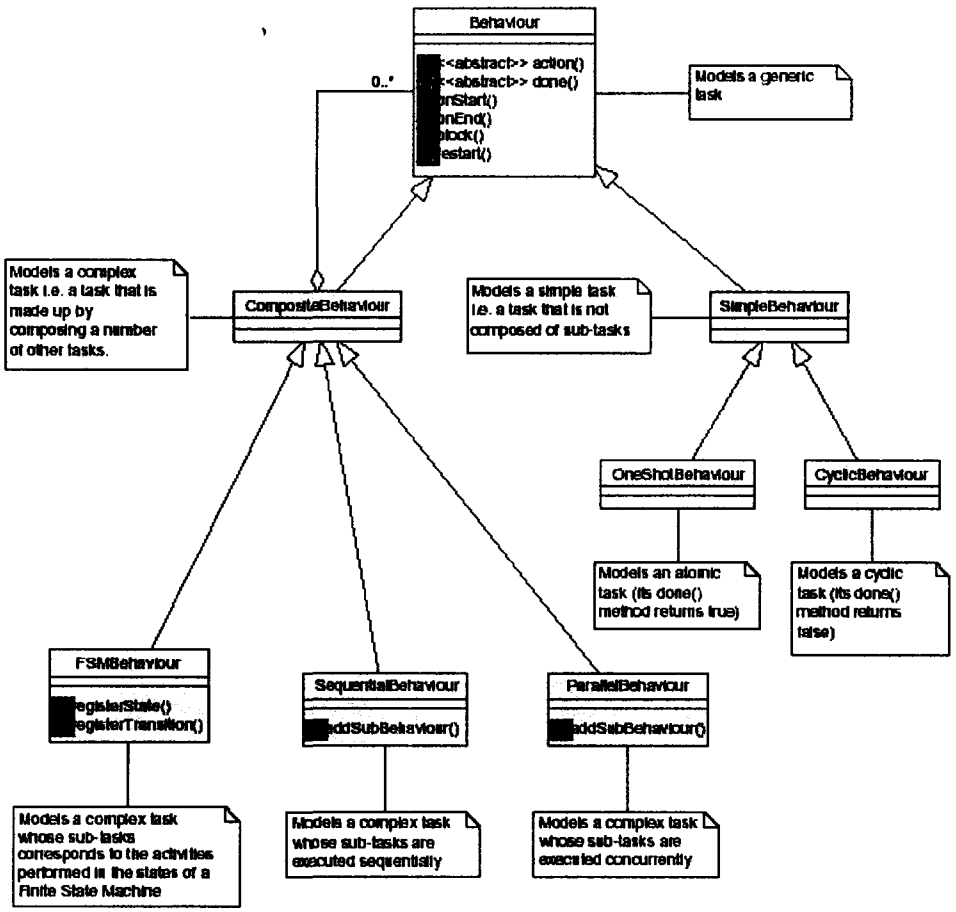


Figura 2.3: Comportamientos de JADE [8]

2.4. Razonamiento Basado en Casos

El razonamiento es comúnmente modelado como un proceso que desarrolla conclusiones encadenando reglas generales. El Razonamiento Basado en Casos (RBC) toma un punto de vista muy distinto. En el RBC, nuevas soluciones son generadas recuperando los casos más relevantes de la memoria y adaptándolos a las nuevas situaciones y no

encadenando reglas generales [13].

El razonamiento basado en casos es un paradigma general para razonar por medio de la experiencia. El RBC supone un modelo de memoria para la representación, indexación, y organización de casos pasados; también da por hecho un modelo de procesamiento para la recuperación y modificación de antiguos casos así como también de la asimilación de los nuevos [14].

El enfoque del RBC está basado en dos principios acerca de la naturaleza del mundo. El primer principio es que el mundo es regular: problemas similares tienen soluciones similares. Consecuentemente, soluciones para problemas previos son un punto de inicio muy útil para la solución de nuevos problemas. El segundo principio es que los tipos de problemas que un agente encuentra tienden a repetirse. Y por tanto, problemas futuros son muy similares con los problemas recientes o vigentes. Cuando los dos principios son válidos, vale la pena recordar y rehusar razonamientos previos (CBR es una estrategia efectiva de razonamiento) [13].

2.5. Estructura de un Caso

Los casos pueden tener formas y tamaños muy variados. Por ejemplo, los casos pueden representar situaciones que evolucionan con el tiempo (como en el diseño de un edificio o darle seguimiento a un paciente de ceguera severa), pueden representar una instantánea (al escoger un tipo de ventana particular para un edificio o recordar una regla de un juicio jurídico), o incluso, pueden representar un lapso de tiempo de tamaño variable en medio de los dos extremos [14]. También pueden representar un episodio en donde se resolvió un problema (como en los casos médicos y arquitectónicos), etc.

Lo que es común para todos los casos es que representan una situación o experiencia pasada. Esa situación, cuando es recordada, forma un contexto en que el conocimiento basado en el caso es aplicable. Cuando una situación similar surge, esas decisiones y el conocimiento que viene dentro de éstas, dan un buen punto de partida para interpretar la nueva situación.

La definición de un caso proveniente de la información anterior es: *Un caso es una porción contextualizada de conocimiento representando una experiencia que enseña una lección fundamental para lograr la meta del razonador.*

Las tres partes más importantes que todo caso debe tener son [27]:

1. **Descripción del problema/situación.** El estado del mundo cuando el episodio almacenado en el caso ocurrió. Describe circunstancias previas que son relevantes. Esas circunstancias pueden incluir las metas del razonador, las tareas a realizar, o las características del problema/situación.

2. **Solución.** El estado o la solución encontrada al problema especificado en la descripción del problema. Le dice al razonador cómo fue resuelto un problema en una instancia en particular.

3. **Salida.** El estado del mundo resultante cuando la solución fue llevada a cabo.

2.6. Proceso del Razonamiento Basado en Casos

El razonar, bajo el tradicional punto de vista de la inteligencia artificial y de la psicología cognoscitiva, es un proceso para recordar operadores abstractos y hacer composiciones sobre estos operadores [14]. El razonamiento basado en casos toma una vista alternativa. En vez de ver el razonamiento como un proceso principalmente de composición, el RBC ve al razonamiento como un proceso de recuerdo de una situación o un conjunto de situaciones para así, poder hacer inferencias comparando la nueva situación con las previas. Como se pudo mostrar en la sección anterior los casos tienen básicamente dos finalidades: los casos proveen sugerencias de soluciones a problemas y los casos proporcionan un contexto para entender o evaluar una situación. La figura 2.4 muestra el proceso del razonamiento basado en casos. Este proceso es descrito brevemente a continuación.

1. Un problema o caso nuevo es analizado y representado adecuadamente para que el sistema RBC pueda recuperar casos previos que son relevantes. La meta es recuperar casos útiles que tengan el potencial de proveer una solución al nuevo problema en mano.

2. Una vez que los casos relevantes son recuperados, estos son *"clasificados"* u *"ordenados"* (basándose en algún conocimiento de similitud) y el mejor subconjunto, o casos más prominentes, son regresados.

3. Muy frecuente, un caso previo no encaja exactamente con el caso nuevo; por tanto es necesario hacer cambios a una antigua solución para ajustarla a la nueva. El proceso de hacer estos cambios (adaptación) puede variar desde hacer pequeñas sustituciones, hasta hacer cambios estructurales. Qué adaptación, y cómo esto debe ser hecho, depende en que dominio se encuentre el conocimiento.

4. La solución inicial al nuevo caso es propuesta.

5. La solución propuesta es probada, evaluada y posiblemente mejorada por realimentación.

6. El nuevo caso es actualizado dentro de la librería de casos para futuro uso. Agregando nuevas situaciones o casos dentro de la librería de casos, el sistema está realmente llevando un proceso de aprendizaje.

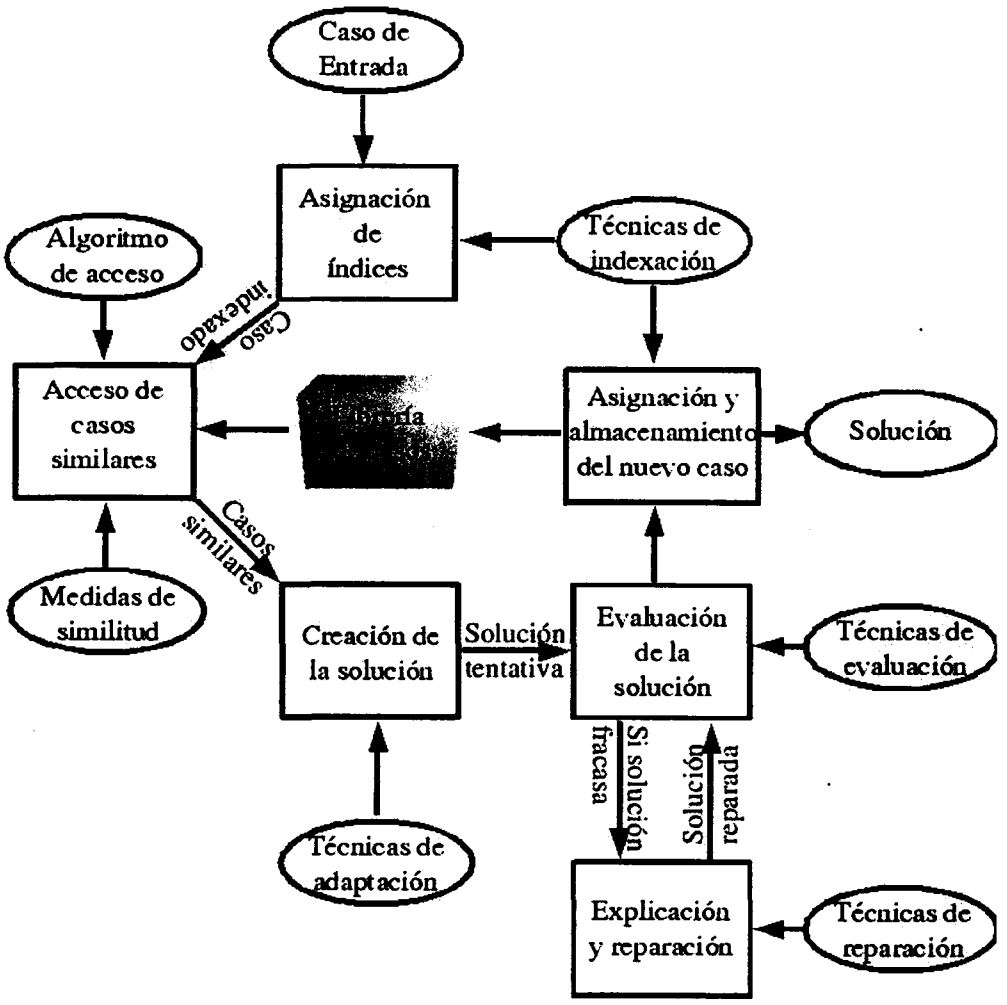


Figura 2.4: Proceso del paradigma del razonamiento basado en casos [16].

2.6.1. Acceso a Casos

El acceso a casos es uno de los más importantes procedimientos dentro del razonamiento basado en casos. El recuerdo es el proceso de acceder a un caso o a un conjunto de casos de la memoria. En general este consiste de dos subpasos [13]:

1. **Acceso de casos previos.** La meta de este paso es el recuperar de la memoria "buenos" casos. Los casos buenos son aquellos que tienen el potencial de hacer predicciones relevantes acerca del nuevo caso. El acceso es llevado a cabo utilizando índices dentro de la librería de casos.

2. **Selección del mejor subconjunto.** Este paso selecciona los casos más relevantes o casos a razonar de aquellos seleccionados en el paso anterior. Algunas veces es apropiado seleccionar un solo mejor caso; y algunas otras un conjunto pequeño de casos es necesario.

Existen algunos problemas que deben considerarse para hacer el acceso a los casos. Primero, necesitamos darle a la computadora el significado de que un caso es aplicable a otro nuevo. Este problema se denomina como *evaluación de similitud.* Una manera de tratar a este problema es el usar más que solo la representación del caso para la comparación. Los casos pueden ser comparados a un nivel más abstracto de representación. Para lograr esto se necesita descubrir cual de todas las maneras de representación abstracta de un caso es la adecuada para comparación. Normalmente se llama a esto el problema de *indexación de vocabulario* [14].

Algunas veces se sabe tan poco de una nueva situación que hace muy pequeño el nivel de comparación que se pueda hacer con otras situaciones. Algunas veces lo que se sabe está en una forma muy poco refinada como para formar una comparación, y algunas características adicionales se necesitan derivar. A esto se le llama el problema de *evaluación de situación.* Por ejemplo, en la predicción de quien ganará una batalla, la proporción o razón de la fuerza de defensa con la fuerza de ataque se puede predecir, pero ninguna de las dos por si solas se puede calcular.

Otro aspecto a considerarse es el *algoritmo de recuperación.* ¿Cómo podemos buscar casos apropiados dentro de una gran librería de casos de manera eficiente? Una manera eficiente de solucionar este problema es el encontrar una forma de representar índices de casos que puedan conducirnos sobre los casos más parecidos de manera eficiente en una máquina paralela. El supuesto aquí es que en una máquina paralela, el cuello de botella se encuentra en el *algoritmo de similitud* y no en el *algoritmo de búsqueda*

Juntos, todos esos problemas conforman el *problema de indexación.* El problema de indexación es el problema de recuperar casos aplicables a tiempos apropiados (a pesar de todos los problemas descritos anteriormente). En general, este problema ha sido tratado

como una problema de asignación de etiquetas, llamadas índices, a casos de los cuales se puedan hacer inferencias útiles. Los índices actúan como índices a libros en una librería. Los índices de un caso son combinaciones de sus descriptores importantes, aquellos que lo distinguen de otros casos. Es decir, como nosotros normalmente usamos una carta bibliográfica en una librería para dirigirnos directamente a lo que estamos buscando o a lo que necesitamos. Los algoritmos de recuperación utilizan estos índices para obtener o acceder a los casos más parecidos a satisfacer las necesidades del razonador. Cuando se agrega un caso a una librería de casos, se le asigna un índice, como se le asigna índices a un libro que acaba de llegar a una librería.

2.6.2. Creación de la solución

En el siguiente paso, porciones relevantes del caso o conjunto de casos son extraídas para formar la solución del nuevo caso [16]. Si se está usando al RBC para resolver problemas, entonces se usan las soluciones previas para formar una nueva para el nuevo caso. Si se está usando al RBC para interpretación o evaluación de situaciones, entonces los casos recuperados o accesados son particionados en función de la interpretación o solución que predicen y, basado en eso, asignar una interpretación inicial a la nueva situación.

Existen algunos puntos que deben tomarse en cuenta en la construcción de la solución. Por ejemplo, como seleccionar las porciones importantes o significativas de casos previos. Un caso pasado puede ser arbitrariamente largo y no es deseable el considerar porciones con ninguna o poca relevancia al nuevo caso [3]. Una posible solución a este punto es que la meta del razonador determine que partes del caso previo puedan ser de importancia. Es decir, ya que el razonador seleccionó anticipadamente las partes del caso previo, lo más lógico es hacer que el razonador tome como punto de partida estas partes para la solución del caso nuevo. Por tanto, si el razonador está tratando de derivar una solución, éste se enfocará en soluciones previas. Si el razonador está tratando de interpretar una situación, la clasificación en los casos previos serán el punto de partida a evaluar.

2.6.3. Adaptación

En el RBC enfocado a resolver problemas, las soluciones pasadas son usadas para resolver nuevas situaciones. Ya que es poco frecuente que las situaciones previas son idénticas a las nuevas, las soluciones deben ser adaptadas para satisfacer las nuevas situaciones. En este paso, la solución obtenida en el paso anterior es adaptada para corresponder a la nueva situación [16].

La adaptación puede ser tan simple como la sustitución de un componente de la solución anterior o puede ser tan complicada como la modificación de la estruc-

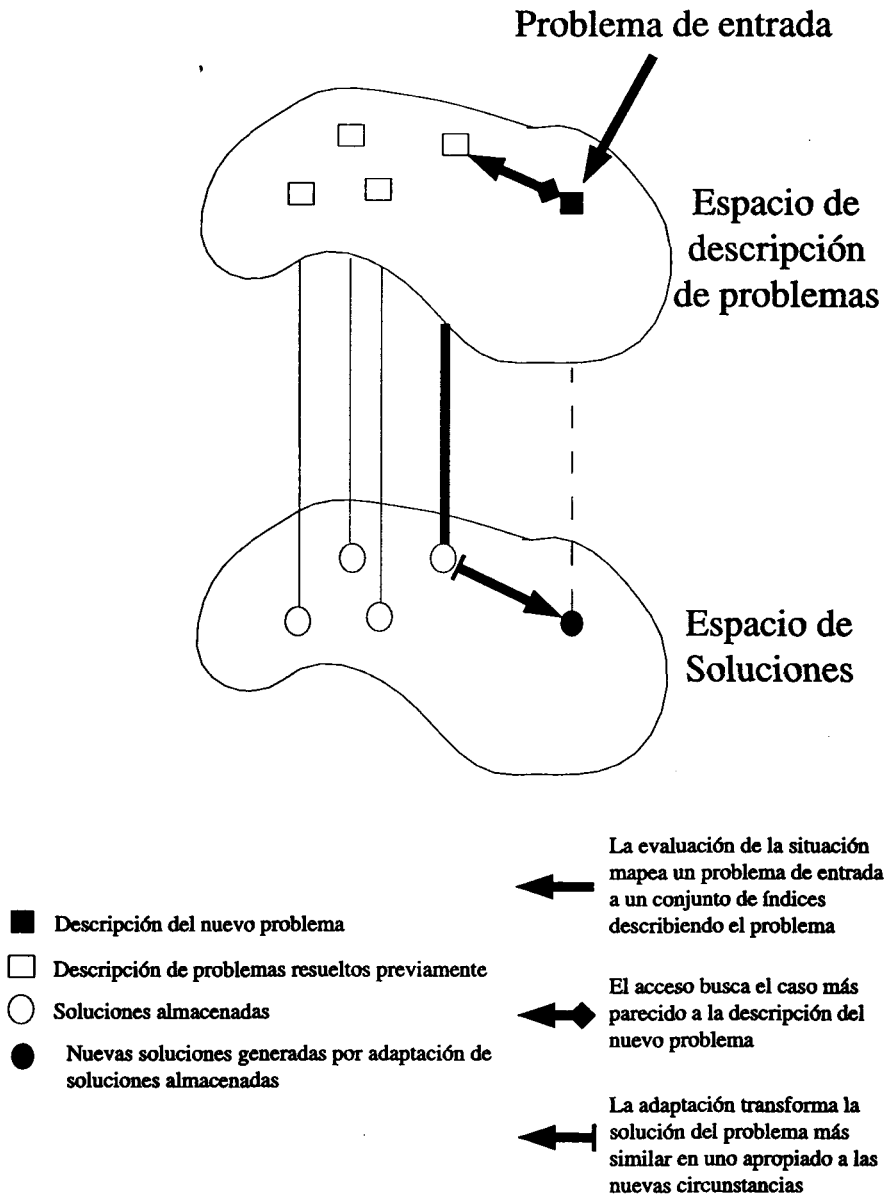


Figura 2.5: Esquema de como el RBC basado en resolución de problemas genera una nueva solución.

tura general de la solución anterior. La adaptación puede realizar distintas acciones: algunas cosas pueden insertarse en las soluciones anteriores, algunas otras pueden ser eliminadas, algún atributo puede ser sustituido por otro o alguna parte de la solución anterior puede ser transformada [13, 28].

Pueden encontrarse diez métodos de adaptación descritos en libros de razonamiento basado en casos [13].

- Métodos de sustitución.
  - Reinstanciación
  - Ajuste paramétrico
  - Búsqueda local
  - Consulta a memoria
  - Búsqueda especializada
  - Sustitución basada en casos
- Métodos de transformación
  - Transformación de sentido común
  - Reparación de modelo guiado
- Adaptación y reparación de propósito especial
- Derivativo

La finalidad de esta investigación no es la implementación de esta parte del razonamiento basado en casos, por esta razón no se darán los detalles de estos métodos.

2.6.4. Evaluación de la solución

La solución creada en la etapa de adaptación es una solución tentativa. Es necesario ahora discriminar que tan acertada es la solución anterior para poder identificar las debilidades, así como también las cualidades [16].

Precisamente esta es la tarea de la evaluación de la solución y es definitivamente uno de los pasos más cruciales si el dominio del sistema es uno en el cual no se pueden correr riesgos. "Para la ejecución de este proceso se han desarrollado diferentes ideas: el probar la solución con contraejemplo hipotéticos o reales, o bien, usando la solución como índice para buscar en memoria otro caso en el que se haya usado esa solución y se haya fracasado en circunstancias similares" [16].

2.6.5. Explicación, reparación y prueba

Para no repetir los mismos errores encontrados en la etapa de evaluación, si la solución fracasa, es importante obtener y analizar información de retroalimentación. El sistema debe explicar todo resultado inesperado para tratar de reparar la solución, y de esta forma evaluar la solución nuevamente [16].

Son usadas muchas técnicas para tratar de encontrar que parte de la solución falló, como por ejemplo, técnicas de Machine Learning o incluso usar el mismo RBC [16].

2.6.6. Asignación de índices y almacenamiento del nuevo caso

Se procede a entregar la solución al mundo real una vez creada y evaluada. Esta nueva solución puede generar un nuevo caso el cual no solo se forma de la solución creada, sino del caso o problema original, las justificaciones y explicaciones, y la solución encontrada. A este nuevo caso se le pueden asignar índices los cuales se utilizan para el proceso de almacenaje en el repositorio de casos [16].

La asignación de índices o el dónde guardar el nuevo caso es un problema relevante y se puede llevar a cabo indexando el caso por fallas o problemas que se encontraron, para no obtenerlos nuevamente [16].

2.7. Trabajo previo en Sistemas Basados en Casos

¿Cómo se han venido construyendo los sistemas de RBC? La respuesta a esta pregunta es la que se explora en este capítulo. En la primera parte de esta sección se hace una introducción a lo que es la construcción de sistemas basados en casos. Ya con los antecedentes de las secciones anteriores, ahora se procede a la descripción de algunos de los trabajos que se han realizado en la aplicación del razonamiento basado en casos.

En las siguientes secciones se consideran diversos y diferentes sistemas que de alguna manera u otra han contribuido al crecimiento de esta nueva área de investigación, considerando especialmente los trabajos más recientes y los que de alguna forma se relacionan con esta investigación.

2.7.1. Construcción de los sistemas basados en casos

Las razones por las que se puede estar interesado en un sistema basado en casos son: una base de datos en la que se pueda realizar acceso a casos que solo encajen parcialmente con la especificación de entrada; para sugerir respuestas a problemas; dar soporte en la toma de decisiones, resolver problemas de diagnóstico, planeación, programación, entre otras [16].

Se pueden considerar a grandes razgos dos tipos de sistemas basados en casos: los totalmente automatizados, y los parcialmente. Los primeros son los que resuelven los problemas en su totalidad respecto al proceso de adaptación. Mientras que los segundos, esto no se resuelve de manera autónoma, es decir, sin la intervención humana. En estos sistemas el usuario y la máquina trabajan de manera conjunta para resolver el problema. Mientras que el usuario se encarga de la fase de adaptación y de las soluciones difíciles, la máquina se encarga de las partes sencillas [16].

Los dos tipos de sistemas de RBC son casos extremos. La gran mayoría de estos sistemas se encuentran entre estos extremos unicamente variando el grado de intervención humana en la toma de decisiones [16].

2.7.2. Toma de decisiones en ingeniería redundante

Aveces se presentan problemas no muy frecuentes para los que no se tienen una solución determinada, cuando éstos se resuelven sin utilizar la experiencia de problemas similares, comúnmente se le nombra como problema de ingeniería redundante. Normalmente, estos problemas ocurren por ciertas razones; porque los problemas son solucionados por diferentes personas, porque los problemas ocurren distanciados en tiempo y espacio, por rotación de personal [16]. Dichos problemas no tienen una solución estandar.

El largo proceso de construcción de barcos es un buen ejemplo para problemas de IR. La solución para este tipo de problema consiste en la extracción de conocimiento de múltiples expertos, el conocimiento relevante debe ser expresado de manera uniforme y accesible a cualquier experto. A finales de los ochentas y principios de los noventas se inició el desarrollo de un sistema RBC para solucionar este problema [16].

Mediante listas de pares de atributos/valores pueden representarse los casos, mismos que pueden ser numéricos o simbólicos; si éstos son numéricos cada caso puede ser visto como un punto en un espacio n-dimensional, y la distancia euclidiana puede ser usada como medida de similitud. En caso de que los atributos sean simbólicos, entonces se puede medir la similitud simplemente contando el número de atributos que pueden ser iguales; esto no asegura la similitud real de los casos y para solucionar esto se aplicó lo siguiente [16]:

- "Introducir reglas de determinación que almacenan información relevante entre conjuntos de atributos y posibles problemas." [16]
- "Manejar la similitud como una función del propósito. El conocimiento acerca del propósito de la comparación de dos casos enfoca la atención hacia los atributos relevantes. Por supuesto esta idea puede ser combinada con las reglas de determinación, siendo estas reglas las que almacenan la información relevante entre propósitos y conjuntos de atributos." [16]

El mantenimiento y actualización de la base de casos se realizó directamente por los usuarios de forma automática durante la utilización [16].

En este trabajo se muestra cómo se aplicó un mecanismo basado en similitud para la tarea de la recuperación de casos similares para atributos de tipo numérico y simbólico (cadenas). Aunque existen otros mecanismos como por ejemplo los basados en *inducción* (árboles de decisión) son un poco más complicados de implementar y tienen la gran desventaja de no poder ser usados para atributos continuos. Aparte, cuando un atributo no es conocido entonces no puede ser recuperado ningún caso.

2.7.3. Indexación para la explicación de fallas

Simpre es necesario explicar una falla que se presente con el objetivo de diagnosticar acertadamente la causa del problema y si es posible una futura recurrencia de la misma [16].

El programa ACCEPTER tiene la capacidad de entender historias. Este programa usa el razonamiento basado en casos para generar las explicaciones de las fallas adaptando explicaciones de circunstancias previas que sean similares [16].

Fue desarrollado un vocabulario para implementar una recuperación de casos efectiva a fin de caracterizar anomalías. De esta manera los casos son almacenados en memoria utilizando el vocabulario como medio de indexación.

El proceso de acceso a casos previos se lleva a cabo en tres pasos:

1. Se detecta la anomalía.

2. Al *caracterizador* se le especifica la descripción de la anomalía y regresa como resultado una caracterización de la anomalía bajo el vocabulario de anomalías.

3. Se utiliza como índice la caracterización de la anomalía para acceder a la explicación de la base de explicaciones.

Se observan varios aspectos al presentarse fallas similares: la situación en sí misma, fallas similares producen caracterizaciones similares. Dichos aspectos se pueden observar en las categorías y subcategorías del vocabulario de ACCEPTER [16].

Una vez que se determina cierta categoría de la anomalía mediante el caracterizador, es usada la categoría para las explicaciones relacionadas con la misma caracterización. Cada una de estas caracterizaciones se asocia con una estructura para describir las fallas por categoría. Los slots de la estructura de caracterización son llenados mediante reglas de especificación [16].

En el momento que se desea la recuperación o acceso de alguna explicación solamente se consideran aquellas que estén dentro de la misma categoría o subcategoría caracterizada. La similitud en los valores de los atributos o slots de las estructuras son

usados para discriminar en las similitudes de las mismas estructuras de caracterización de fallas [16].

Cabe notar que no se garantiza que las explicaciones recuperadas sean las apropiadas, aún considerando que el procedimiento de explicaciones toma en cuenta la similitud de las fallas o anomalías con la situación actual. Se debe buscar una explicación lo más parecida posible cuando no se encuentre una explicación acertada y de ser así se puede sugerir qué características son relevantes para realizar una comparativa parcial [16].

En conclusión, la construcción de explicaciones a partir de la nada sería un problema intratable. Afortunadamente, el razonamiento basado en casos reutiliza explicaciones almacenadas en memoria haciendo el problema más tratable. Sin embargo, queda el problema de cómo guiar el proceso de búsqueda. Para esto las categorías dan ciertos criterios de cuáles explicaciones pueden ser más relevantes, restringiendo así el espacio de búsqueda. Después, la asociación de las categorías con estructuras de caracterización particulares da mayor, guía, ya que los slots en la estructura identifican características que son probablemente importantes.

2.7.4. Selección de tareas en la planeación analógica

La planeación analógica sugiere algunos pasos principales para llevarse a cabo: la selección, el mapeo, y la inferencia. Primeramente se tiene un caso o problema que requiere de una solución. Entonces el planeador busca un caso con características equiparables con el problema en cuestión, y es usado un mecanismo de adaptación que interpola el caso seleccionado al nuevo problema. Y finalmente, se utiliza esa adaptación para inferir el plan para la solución del problema original [3, 28, 16].

"El sistema ANAGRAM (ANAlogical GRAph Match) utiliza un algoritmo para comparar grafos. Cada grafo representa un plan determinado. El sistema acepta como entrada dos subgrafos: uno de descripción inicial y otro de especificación de la meta, entonces el programa busca en la base de casos por el grafo que mejor encaje con los dos subgrafos de entrada. Una vez hecha esa selección, el sistema procede a realizar el mapeo del grafo encontrado para generar la solución (el plan). En caso de que el plan generado no sea exitoso o que varios casos similares se encuentren, entonces el sistema intenta mezclar varios casos que son similares entre sí. Con esto se logra un caso virtual que elimina anomalías y hace generalizaciones de varios casos para tratar de generar un plan exitoso" [16].

Uno de los objetivos de esta tesis, es el de mapear una ontología (un grafo) a un conjunto de casos. Aunque ANAGRAM es un algoritmo para comparar grafos utilizando diferentes operadores entre cada nodo. Es posible retomar la idea en que se basa el algoritmo ANAGRAM para la extracción de conocimiento de una ontología de dominio específico. Por ejemplo, se puede considerar el realizar una exploración dentro del grafo ontológico utilizando diferentes operadores con la finalidad de extraer los índices

necesarios para la etapa de recuperación dentro del proceso del RBC.

2.7.5. Experience-Sharing Architecture: Un caso de estudio para el control de calidad de software en NEC corporation

El sistema SQUAD es un prototipo basado en casos desarrollado y sustentado usando el "CASE_METHOD"[1] [14]. Es un sistema de soporte en control de calidad de software basado en casos que tiene la finalidad de dar soporte a "*the experience sharing architecture*" (ESA), un nuevo concepto en sistemas de información corporativos puesto a prueba en NEC corporation. En la versión actual, el sistema SQUAD no soporta una fase de adaptación. SQUAD maneja solo la fase de recuperación por dos razones primarias. Primeramente, la fase de recuperación de casos es suficiente para la mayoría de tareas de soporte. Segundo, el dominio es tan complicado y amorfo que cualquier esquema de adaptación requeriría elevados costos de desarrollo. Además, el comportamiento del sistema sería inestable ya que no se entiende la naturaleza del dominio.

La principal característica de SQUAD es el uso de una RDBMS comercial para almacenar y manejar casos. Aunque se investigaron muchas consolas (Shell) RBC comerciales ninguna de ellas soportaba la recuperación basada en similitud en una RDBMS. El sistema RBC comercial usado fue CARET (RDBMS-based Case Retrieval Shell). En este sistema RBC, cada caso es representado como un registro de una tabla en una base de datos relacional. El uso de una RDBMS ofrece muchas ventajas como la seguridad de los datos, independencia de los datos, estandarización de los datos, e integridad de los datos (ver apéndice B).

Ya que el uso de una RDBMS fue el requerimiento mínimo para incorporar sistemas RBC como parte de un sistema de información en corporaciones a gran escala, CARET fue la herramienta escogida para este propósito. CARET genera expresiones SQL apropiadas para llevar a cabo la recuperación de casos basada en similitud sobre bases de datos comerciales como ORACLE. Existen dos restricciones en el uso de una RDBMS para sistemas basados en casos. Primero, el lenguaje SQL no soporta la recuperación basada en similitud, consecuentemente un mecanismo para llevar a cabo esta tarea debe ser definido. Segundo, los casos tienen que ser representados como un registro plano de relaciones n-arias. Las RDBMS no incluyen mecanismo para soportar esquemas de indexación complejas vistas en la mayoría de las investigaciones de RBC. Sin embargo, esta restricción no es necesariamente un factor limitante para la representación de los casos. Además se ha visto que complejos esquemas de indexación

[1]CASE_METHOD es una metodología para la construcción y mantenimiento de sistemas RBC a gran escala basados en el paradigma ESA (Experience-Sharing Architecture).

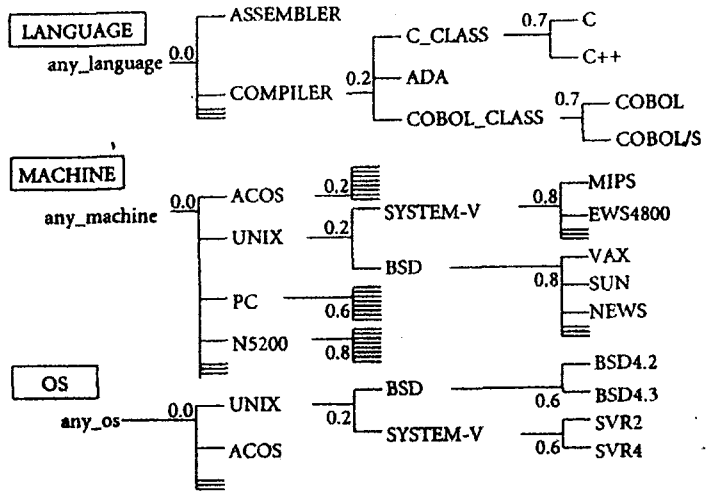


Figura 2.6: Ejemplo de una jerarquía abstracta.

son demasiado difíciles de mantener, particularmente por ingenieros ordinarios y aparte entorpecen la velocidad de recuperación en computadoras paralelas por la enorme comunicación entre los procesadores para buscar conjuntamente a través de las complejas estructuras de indexación. Un registro plano es más adecuado para búsquedas paralelas por la naturaleza paralela del calculo de similitud.

Para poder crear un conjunto de expresiones SQL en la recuperación basada en similitud CARET utiliza jerarquías abstractas, como las mostradas en la figura 2.6, para generar un conjunto de valores vecinos al valor especificado por el usuario. Las medidas de distancia entre los valores y el valor representante de la importancia del atributo (peso del atributo) son usados para asignar valores de similitud a cada expresión SQL. Como resultado, un conjunto de expresiones SQL (asignando a cada una un valor de similitud) son producidas. Si por ejemplo un usuario especifica `ADA` como *language* y `VAX` como *machine*, las expresiones SQL mostradas en la tabla 2.7.5 serán generadas y enviadas a la RDBMS.

CARET usa la técnica de recuperación del vecino más cercano (Nearest Neighbor Retrieval) en lugar de métodos basados en indexación que han sido estudiados en la mayoría de las investigaciones en el RBC [Cain 1991, Cook 1991, Kolodner 1988]. La similitud ente el caso de consulta (query) Q y el caso C en la base de casos $S(Q, C)$ es la suma con pesos de la similitud de cada atributo.

rank	*Similarity*	*SQL Specification (only WHERE clause is shown)*
1	1.0	(language = ada) and (machine = vax);
2	0.89	(language = ada) and (machine in (sun, news, ...));
4	0.54	(language = ada) and (machine in (mips, ews4800, ...));
3	0.66	(language in (c, c++, cobol, cobol/s)) and (machine = vax);
4	0.54	(language in (c, c++, cobol, cobol/s)) and (machine in (sun...));
6	0.2	(language in (c, c++, cobol,...)) and (machine in (mips,...));

Cuadro 2.1: Resultados SQL.

$$S(Q,C) = \frac{\sum_{i=1}^{n} W_i \times s(Q_i, C_i))}{\sum_{i=1}^{n} W_i} \tag{2.1}$$

donde W_i es el peso del atributo, $s(Q_i, C_i)$ es la similitud entre el valor del atributo i-ésimo del caso de consulta Q y el caso C en la base de datos relacional.

Las implementaciones tradicionales computan el valor de similitud para todos los registros, y ordenan todos los registros basados en sus similitudes. Sin embargo, esta es una tarea que consume tiempo linealmente creciente respecto al número de casos en la base de casos (C: *cardinalidad* de la base de datos) y el número de atributos definidos (D: *grado* de la base de datos). Esto resulta en una complejidad en tiempo de $O(C \times D)$. *Aplicaciones con esta estrategia de implementación para una RDBMS serían muy lentas a menos que se combine con algún mecanismo de indexación para no computar la similitud de todo caso dentro de la base de datos*[2].

La similitud entre valores es definido usando las jerarquías abstractas mostradas en la figura 2.6. Está definido cada atributo en una tabla dentro de la base de datos, como `language` , `machine`, y `OS`. En este ejemplo la similitud entre `C` y `C++` es 0.7. De esta forma la similitud entre la consulta de entrada y cada caso dentro de la base de casos es calculada de acuerdo a los valores de similitud de cada atributo dentro de la jerarquía abstracta. Para poder hacer esto CARET tiene antes que crear el conjunto de los valores vecinos (NVS's) utilizando la jerarquía abstracta como es mostrado en la figura 2.7, para generar un conjunto de valores de vecindad a los especificados por el usuario. Por ejemplo, supongamos que el usuario especificó `BSD4.2` en la jerarquía mostrada en la figura 8. `BSD4.2` es un elemento vecino de primer orden (1NVS). `BSD` y `BSD4.3` son dos elementos vecinos de segundo orden (2NVS), y `UNIX`, `System-V`, `SVR2`

[2]Esta es la principal inspiración en la creación de los índices de partición y similitud (PI y SI) dentro de esta investigación mencionados en la sección 3.4

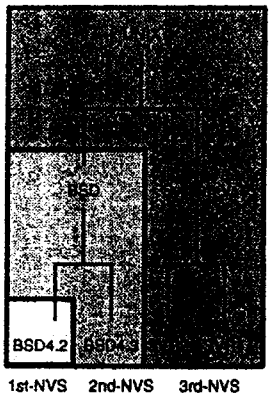


Figura 2.7: Conjuntos de valores vecinos.

and (language(ada), machine(vax));
and (language(ada), machine(or (sun, news, ...)));
and (language(ada), machine(or (mips, ews4800, ...)));
and (language(or (c, c++, cobol, cobol/s)), machine(vax));
and (language(or (c, c++, cobol, cobol/s)), machine(or (sun...)));
and (language(or (c, c++, cobol,...)), machine(or (mips,...)));

Cuadro 2.2: Combinaciones creadas.

y SVR4 son elementos vecinos de tercer orden (3NVS). Esos conjuntos son creados por cada atributo.

Supongamos que el usuario especificó VAX para el atributo *machine* y ADA para el atributo *language* en la jerarquía de la figura 8. La tabla 2.7.5 muestra los conjuntos NVS.

Todas las posibles combinaciones de valores vecinos del conjunto vecino de orden n-ésimo, son mostradas en la tabla 2.7.5. El siguiente paso es asignar un valor de similitud entre los valores especificados por el usuario y los valores de cada combinación. El cálculo es similar al de la ecuación 2.1 excepto que no todos los atributos son involucrados. El algoritmo de CARET no computa atributos no especificados por el usuario. Si el usuario especificó o no un atributo, esto es mostrado en la matriz de máscara M, que es una matriz unidimensional cuyo tamaño es igual al grado de la base de casos. El elemento i-ésimo de la matriz será 1 si el usuario especificó un valor para el atributo i-ésimo, de lo contrario M_i será 0. La formula para calcular la similitud es la siguiente:

Set	Language	Machine
1NVS	ADA	VAX
2NVS	ADA	SUN, NEWS, ...
3NVS	ADA	MIPS, EWS4800...

Cuadro 2.3: NVS's para ADA y VAX.

$$S(Q,F) = \frac{\sum_{i=1}^{n} M_i \times W_i \times s(Q_i, F_i))}{\sum_{i=1}^{n} M_i \times W_i} \tag{2.2}$$

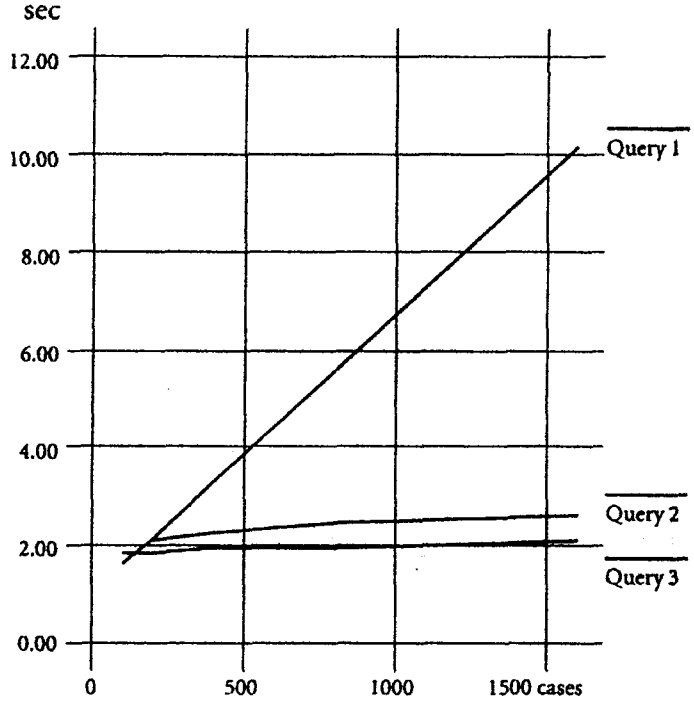
donde F es una combinación de NVS's y F_i es el i-ésimo atributo de la combinación. Se debe notar que la similitud es calculada entre el atributo especificado por el usuario y la combinación de NVS's que son las expresiones SQL. Por ejemplo la similitud de la combinación, (["C"], ["MIPS,..."]) al caso de consulta es 0.2. Esto es porque solo el atributo *language* y *machine* están involucrados (el usuario especificó sólo esos atributos), y cuyos pesos son 0.3 y 0.4 respectivamente, y la similitud entre `ADA` y `C` es de 0.2 y la de `VAX` y `MIPS, ...` es de 0.2.

$$S(Q,F) = \frac{0{,}2 \times 0{,}3 + 0{,}2 \times 0{,}4}{0{,}3 + 0{,}4} = 0{,}2$$

Cuando existen demasiadas combinaciones, el crear todas las combinaciones y enviar todas las consultas SQL a la base de datos es ineficiente y costoso. Métodos para limitar el número de expresiones SQL a ser creadas son necesarios en una aplicación real. Dos métodos fueron implementados para solucionar este problema. El primero, es el de enviar la expresión SQL que tiene mayor similitud y contar el número de casos recuperados, deteniéndose cuando se tengan los N casos más parecidos. El segundo método es el de especificar un cota inferior o valor umbral en el cual solo se obtengan expresiones SQL que tengan mayor similitud que ese valor.

2.7.6. Un Sistema Multiagente Basado en Casos para la "Navegación del Conocimiento"

El Laboratorio de Inteligencia Artificial en la Universidad de Chicago ha empezado el desarrollo de un nuevo sistema de agentes de software diseñado para manejar el desbordante flujo de datos llamado informalmente "information superhighway". Esta metodología toma su primacía de tecnología basada en casos puesto que la construcción



Factor	Query-1	Query-	Query-3
Longitud del Query	1	2	3
Profundidad del árbol &3	2 X 3	2 X 2 X 2	
Ancho del árbol	16	16+12	8+8+9
Número de expresiones SQL	3	6	4
Cases Matched	158+4+199	0+0+11+0+0+94	2+0+0+0

Figura 2.8: Tiempos de respuesta de tres consultas de usuario.

de este sistema da énfasis al uso de ejemplos encima de consultas explícitas o preguntas a manera de método de comunicación con el usuario [14].

Se propuso la construcción de tres sistemas: sistema de búsqueda (llamado sistema FIND-ME), organizadores de tareas basados en preferencias (llamado BUTLERS), y agentes de "internet news group" (llamado CORRESPONDENTS). Los tres tipos de agentes están diseñados a ayudar al usuario a navegar a través de un espacio de información y también encontrar o construir respuestas apropiadas a las necesidades del usuario.

Dos características distinguen estos sistemas. La primera es que estos sistemas se derivan de las ideas o principios que propone el paradigma del razonamiento basado en casos ya que utilizan el acceso y adaptación de casos como modelo principal. La segunda es que ellos usan archivos existentes y bases de datos como recursos de casos a ser minados en lugar de crear nuevas bases de conocimiento independientemente del usuario.

Agentes FIND-ME

El sistema FIND-ME está diseñado para permitirle a un usuario navegar a través de un conjunto de posibles soluciones o productos que encajaron sus necesidades. Los proyectos FIND-ME son guiados por tres conceptos principales: *interfaz metafórica, razonamiento a través de ejemplos*, y el soporte de *búsqueda no jerárquica.* Una de las ideas principales de los sistemas FIND-ME es el desarrollo de interfaces de usuario que se unen metafóricamente con artefactos conocidos. Así como el ambiente Macintosh hace uso del "desk-top metaphor", se han construido interfaces que son análogas a artefactos existentes que ayudan búsqueda dentro de un dominio. Esto les permite a los usuarios tener predicciones fuertes sobre los efectos de sus propias acciones y reduce la carga explicativa en los sistemas. La meta global es proporcionar a los usuarios un sentido de donde ellos están en el espacio del dominio y cómo ellos llegaron allí. El objetivo es proteger al usuario del hecho que él o ella están investigando a través de un espacio de carácter multi-dimensional. En cambio se le presenta al usuario un artefacto familiar, como una revista o pasillo de tienda de video, que tienen el rasgo de ser dinámico. Esto significa que los usuarios pueden hojear un ambiente que se queda estable con respecto a lo que ellos ya han visto, pero es dinámico en que los próximos artículos que son sugerencias que el sistema cree las necesidades del usuario.

La segunda principal aportación a la navegación de conocimiento es que nosotros podemos usar las diferencias entre un ejemplo presentado y el objetivo de un usuario para formular una descripción del objetivo, que después es usado para acceder una base de conocimiento de futuros ejemplos (un proceso iterativo de exploración del espacio de ejemplos).

Actualmente existen dos implementaciones de estos sistemas `CAR NAVIGATOR` (per-

mite a los usuarios el explorar el dominio de los autos nuevos), y VIDEO NAVIGATOR (permite a los usuarios el explorar el dominio de los video).

Agentes BUTLERS

El segundo tipo de agentes se diseñan para ser especialistas a tareas específicas que pueden hacerse usando información en-línea. La idea es construir un agente específico para encargarse de la tarea de buscar cosas en repositorios de datos en-líneas. BUTLERS son intermediarios de información y son aún más activos que los sistemas FIND-ME; ellos acumulan activamente información relevante a sus tareas de fuentes diversas. Por ejemplo, se propuso el desarrollo de un BUTLER restaurantero que usa conocimiento de preferencias (sabores) personales, lugares acostumbrados para comer, con quién se acostumbra comer, horarios, etc. El agente BUTLER usará información de las guías en-líneas como la ZAGAT (guía de restaurantes), su propio mapa de ciudad, y su horario personal para deducir el tiempo y lugar que satisfacen sus necesidades.

Agentes CORRESPONDETS

En la actualidad, el Internet y los grupos de noticias eran áreas absolutamente públicas en las que los usuarios podrían anunciar mensajes. Esto ha evolucionado con la suma de moderadores, archivos, bases de datos moderados, y archivos de las "frequently-asked-questions". La meta en la construcción de agentes CORRESPONDENTS es continuar este desarrollo agregando agentes inteligentes que utilicen la experiencia acumulada por usuarios.

Los CORRESPONDENTS son sistemas de recuperación basados en casos que son como los sistemas FIND-ME, pero totalmente automatizados. Los CORRESPONDENTS leen peticiones o descripciones de problemas, y construyen querys con los cuales buscar una librería de casos de posibles respuestas o soluciones, y de esta manera enviar la respuesta que encontraron. Un agente de este tipo que se propuso es el FAQFINDER, un sistema "question-aswering" automatizado que usa los documentos de "preguntas más frecuentes" en diversos USENETS de grupos de noticias. FAQFINDER tomará una consulta de usuario sobre un tópico, tratará de encontrar el archivo de "preguntas más frecuentes" con mayor posibilidad de producir la solución, y después busca la pregunta más parecida dentro del archivo y regresa la solución encontrada.

El trabajo hecho en este tipo de sistemas ha demostrado que la estrategia del razonamiento basado en casos es la manera natural para ayudar a usuarios en búsquedas complejas y en dominios pobremente estructurados. Estos sistemas trabajan mejor cuando los usuarios pueden fácilmente evaluar ejemplos, pero tienen dificultades integrando los atributos o características que son de interés.

Capítulo 3

Integración del Agente que Razona en Base a Casos a JITIK

Este capitulo está dividido en tres partes importantes dentro de la construcción del *agente RBC*. La primera parte explica algunos puntos importantes en la integración del *agente RBC* dentro de JITIK. La segunda parte describe al diseño del *agente RBC*, así como también, la forma en que va a ser integrado el agente con el sistema JITIK. La tercera parte mostrará algunos aspectos importantes en la implementación o realización del *agente RBC* como por ejemplo: lenguaje de programación a ser utilizado, tipo de repositorio de casos y software para construcción de agentes (ej. JADE, ZEUS, entre otros).

Por último, en la sección 3.8 se describirán las adaptaciones efectuadas al *agente de sitio* para que pueda proveer los servicios ofrecidos por el *agente RBC*.

3.1. Integración del agente RBC

Antes de empezar el diseño del agente RBC es necesario mencionar algunos puntos importantes. Primeramente, para poder integrar al *agente RBC* dentro de JITIK es necesario especificar los agentes de JITIK con los cuales el *agente RBC* va a interactuar (la siguiente sección dará una descripción más detallada de los roles de estos agente, así como también, los roles del *agente RBC* de manera que se pueda proponer un esquema de integración más concreto).

Como se menciona en la sección 2.2 el *agente de sitio* es el intermediario (broker) entre los agentes personales y los servicios ofrecidos por el sistema JITIK, es decir, este agente de alguna manera u otra debe ser un intermediario necesario entre los servicios que proporciona el *agente RBC* (independientemente de los servicios que el *agente RBC* ofrezca) y los *agentes personales*. Esto sugiere que uno de los agentes con los cuales va a interactuar el *agente RBC* es el *agente de sitio*. Aunque sabemos que el *agente RBC* no va a interactuar directamente con los *agentes personales*, sí lo va a hacer indirectamente. Supongamos que un agente personal se "registró" dentro del sistema JITIK, este nuevo "registro" podría sugerirle al *agente de sitio* el consultar al *agente*

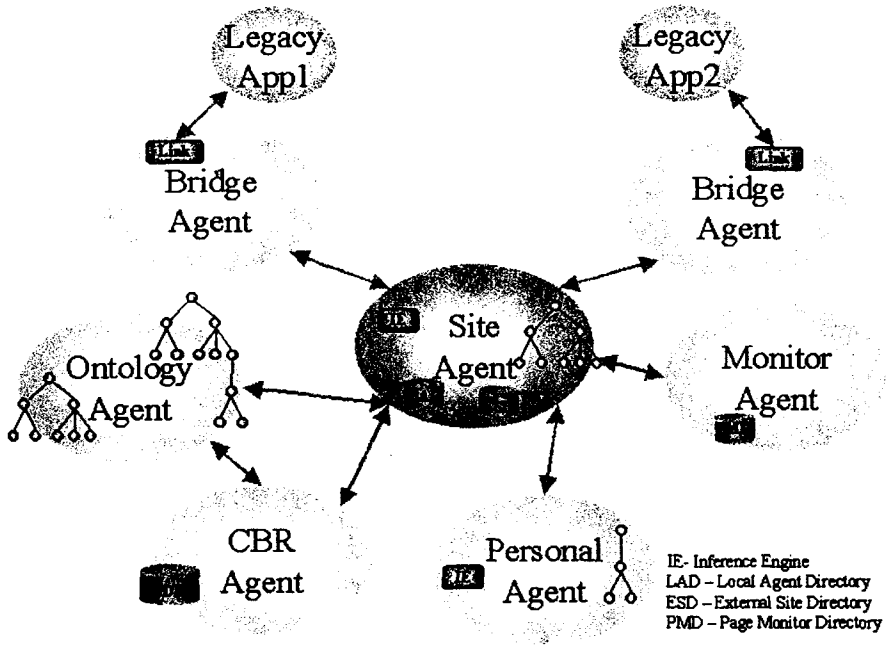


Figura 3.1: Integración del agente RBC con el sistema JITIK.

RBC para buscar casos (dentro de la librería de casos) que pudieran ser relevantes a este nuevo usuario. Otro ejemplo sería el que un usuario cambiara su perfil de intereses, y por tanto, el *agente de sitio* consultará al *agente RBC* para una nueva búsqueda de casos que sean aptos para este nuevo perfil.

Por otro lado, sabemos de antemano que el modelo o arquitectura del agente que se va a diseñar deberá conectar ontologías de dominio específico con una flexible adquisición de casos de dominio independiente, y esto sugiere una continua comunicación con el agente encargado de las ontologías en JITIK (*agente de ontologías*). Este agente contiene toda la información ontológica del sistema JITIK, y en cierta manera, posee el conocimiento necesario para describir el perfil de intereses de todo usuario dentro de JITIK. La figura 3.1 muestra la integración del sistema JITIK y el *agente RBC*.

La figura 3.2 muestra la arquitectura interna del *agente RBC*. La capa de comunicación se encarga de la recepción y envío de mensajes ACL. Los mensajes entrantes son filtrados por los servicios de mapeo y son procesados en la capa de procesamiento. El motor RBC se encarga del razonamiento basado en casos consultando a la memoria (librería de casos). El gui realiza cambios a la estructura de la memoria y estos cambios son reportados por el servicio de monitoreo a librería.

Los servicios que el *agente RBC* dará dentro del sistema JITIK pueden ser clasificados en dos tipos desde el punto de vista del *agente personal*: servicios automáticos,

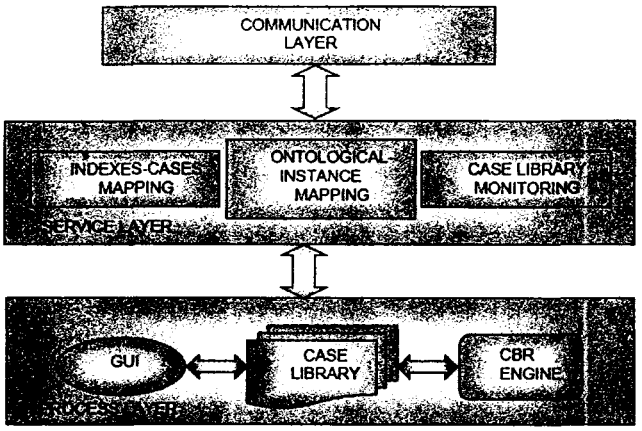


Figura 3.2: Arquitectura del agente RBC.

son aquellos que *agente personal* recibe de forma automática sin necesidad de realizar una petición al *agente de sitio*; y servicios explícitos, son aquellos que el *agente personal* puede solicitar al *agente de sitio* en cualquier momento, y al contrario de los servicios automáticos, es necesario que el *agente personal* solicite explícitamente este servicio.

El *agente RBC* ofrece los siguientes servicios dentro del sistema *JITIK*, mostrados en la figura 3.2:

1. **Servicio de Mapeo de índices a casos.** Se le pide al agente RBC, a través de la capa de comunicación, el mapeo de índices a casos dentro de la librería de casos (capa de memoria, ver figura 3.2) y el agente responde con uno o más casos. Este servicio explícito es iniciado cuando un *agente personal* solicita al *agente de sitio* el mapeo de índices a casos. La figura 3.3 muestra la solicitud de este servicio.

2. **Mapeo de instancia ontológica a casos.** Se le pide al agente RBC, a través de la capa de comunicación, el mapeo de un concepto ontológico instanciado a un conjunto de casos dentro de la librería de casos y el agente responde con uno o más casos. Este servicio automático es iniciado cuando el *agente de sitio* detecta el registro de un nuevo agente personal. La figura 3.4 muestra la solicitud de este servicio.

3. **Servicio de Monitoreo de Modificación en Librería de Casos.** En caso de que el agente RBC detectara la adición de un caso, la eliminación de un caso o la modificación de un caso existente entonces comunicará esta acción a el *agente de sitio* (la edición de la librería de casos se hace a través de la GUI). Este es otro

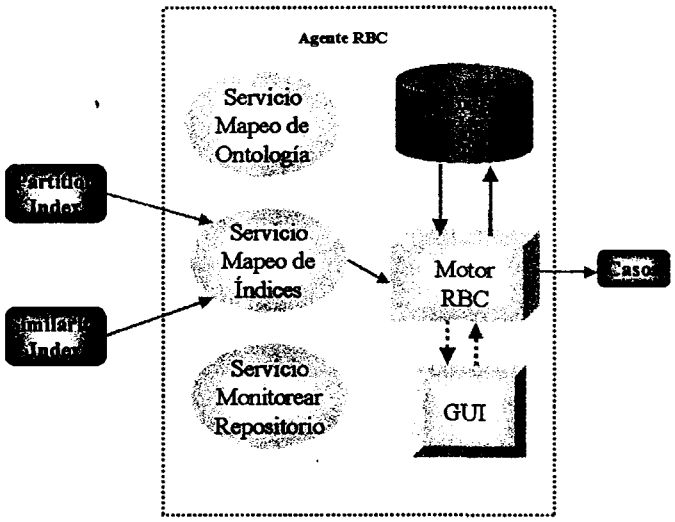


Figura 3.3: Mapeo de índices a casos.

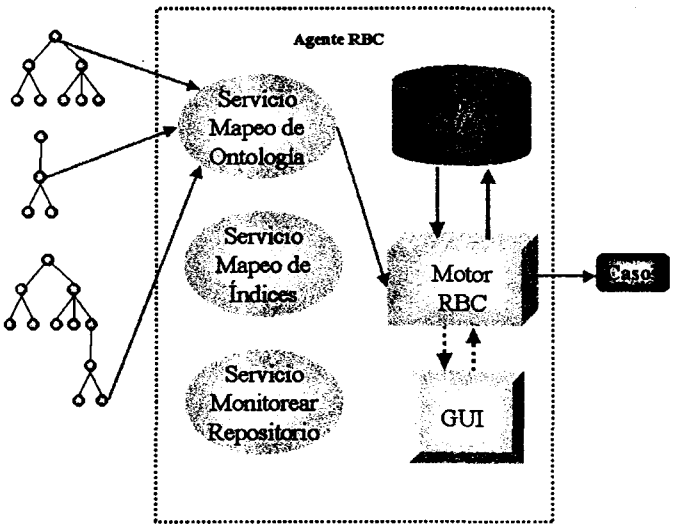


Figura 3.4: Mapeo de una instancia ontológica a casos.

servicio automático proporcionado por el *agente RBC*. La figura 3.5 muestra la solicitud de este servicio.

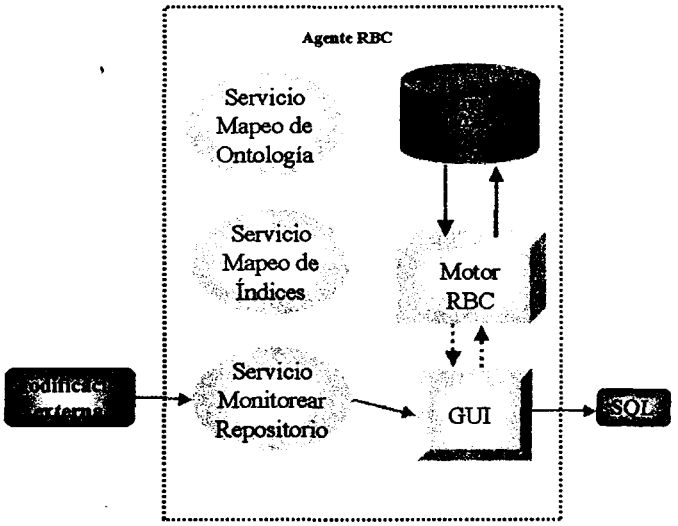


Figura 3.5: Sistema de monitoreo a la librería de casos del *agente RBC*.

El análisis descrito a continuación estará basado en la metodología usada por ZEUS para el diseño de agentes, y en particular, se utilizará la plantilla para diseño de agente dentro del campo del manejo de la información. Aunque no se considerarán los roles de los *agentes personales*, se tomarán en cuenta indirectamente a través del rol del *agente de sitio*.

3.2. Diseño del agente RBC

Ahora que se conocen los dos agentes que van a interactuar con el *agente RBC* (*agente de sitio* y *agente de ontologías*) es necesario describir los roles que cada uno de estos agentes tienen dentro del sistema JITIK.

Primeramente empezaremos por describir el rol del *agente RBC* dentro del JITIK. Vamos a considerar el *agente RBC* como una entidad que mantiene un repositorio de información (librería de casos) y que tiene acceso exclusivo a este repositorio [11]. El *agente de sitio* requiere de la información poseída por el *agente RBC* que puede obtener por medio de envío de mensajes (mensajes ACL). Por otra parte, el *agente RBC* necesitará información que controla el *agente de ontologías* y la extracción de esta información se hará por medio de mensajes (mensajes ACL). Este modelo asume

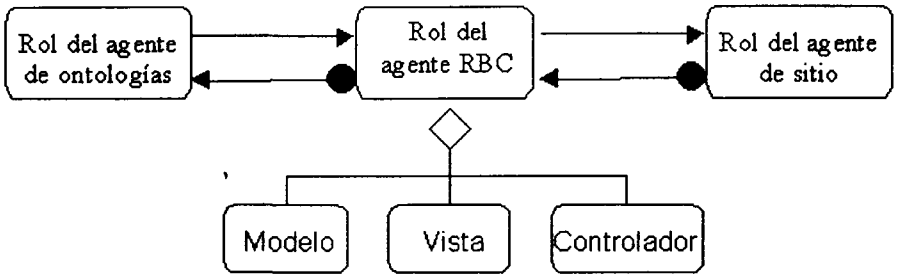


Figura 3.6: Diagrama del modelo de roles.

que existe un sólo *agente RBC* con un repositorio de información, sin embargo pueden haber muchos *agentes RBC* que pueden tener acceso a otros repositorios de datos.

El *agente RBC* contiene tres roles: el modelo es el responsable de manejar la información almacenada, la vista es responsable de mostrar o visualizar la información, y el controlador sirve como el intermediario entre la información almacenada y el *agente de sitio*; El modelo también realiza consultas al *agente de ontologías.*

Por otra parte están los roles de los agentes de sitio y ontología. El *agente de sitio* tiene que monitorear el registro de *agentes personales* dentro de JITIK. Una vez que el *agente de sitio* detecte el registro de un *agente personal*, solicitará al *agente RBC* el servicio de mapeo de instancia ontológica a casos, en donde la instancia ontológica pertenecerá al perfil del nuevo usuario registrado. Otra de las tareas que realizará el *agente de sitio* es la de informar los cambios realizados a la librería de casos. Cuando el *agente RBC* de aviso de una modificación a la librería de casos, el *agente de sitio* comunicará esta acción a todos los *agentes personales* de forma automática[1]. Por último, el *agente de sitio* se encargará de intermediar el servicio de mapeo de índices a casos proporcionado por el *agente RBC.* El *agente de ontologías* tiene como única responsabilidad recibir peticiones de consulta a la ontología y reportar resultados. El diagrama completo de los roles es mostrado en la figura 3.6

La figura 3.6 muestra que el rol del *agente RBC* está formado por tres sub-roles, mientras que ninguna suposición es hecha de la estructura de los roles del *agente de sitio* y el *agente de ontologías.* También muestra que uno o más *agentes de sitio* pueden solicitar los servicios del *agente RBC.* Este modelo de rol contiene cuatro conjuntos de interacciones, y esto es mostrado en la figura 3.7; hay una interacción entre el controlador del rol del *agente RBC* y el rol del *agente de sitio* que involucra la emisión y respuesta a consultas. Las respuestas a consultas son formuladas consultando al modelo (Sistema RBC). El rol de la vista también puede consultar, e incluso modificar, al modelo. Estas

[1]Aunque el sistema JITIK promueve la difusión de información *adecuada* y *oportuna*, el direccionamiento de la información contenida en los cambios efectuados a la librería de casos a los *agentes personales* se deja como trabajo futuro.

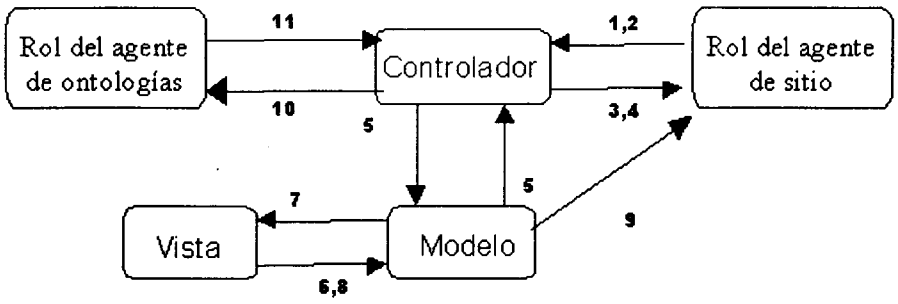


Figura 3.7: Interacciones entre los distintos agentes.

modificaciones son informadas al *agente de sitio* directamente a través del modelo. Por otra parte, el controlador también establece comunicación con el rol del *agente de ontologías*. La explicación de cada colaboración es explicada en la tabla 3.1.

Las figuras 3.8 a la 3.12 muestran una descripción más detallada de los roles mencionados hasta ahora.

3.3. Especificaciones del agente RBC

Este trabajo, como se mencionó en la sección de hipótesis, pretende mostrar si es posible conectar una "herramienta de conocimiento", como lo es el *Razonamiento Basado en Casos*, con una "herramienta de almacenamiento" (sistemas de archivo, DBMS, etc.) dentro del sistema *JITIK* y si es así, como hacer esta conexión.

El modelo o arquitectura del agente que se va a diseñar deberá conectar ontologías de dominio específico con una flexible adquisición de casos de dominio independiente (servicio de mapeo instancia ontológica casos) , mapeará índices a casos dentro del la librería de casos y también deberá monitorear la fuente de datos (librería de casos) para dar aviso sobre cambios que ésta pueda tener.

Antes de empezar la especificación de estos servicios es necesario analizar las especificaciones del *sistema RBC* (rol del modelo) para determinar qué componentes se necesitan tener para poder brindar estos servicios de manera adecuada.

3.4. Especificaciones del Razonador Basado en Casos

Esta sección describe las especificaciones para el rol del modelo, cuyas tareas y responsabilidades así como también la interacción con los otros roles están descritos en

	Colaboración	Explicación
1	Solicitud de mapeo índices-casos	Se le pide al agente RBC el mapeo de índices a casos dentro de la librería de casos y éste responde con uno o más casos
2	Solicitud de mapeo ontología-casos	Se le pide al agente RBC el mapeo de un concepto ontológico instanciado a un conjunto de casos dentro de la librería de casos y éste responde con uno o más casos
3	Respuesta de mapeo índices-casos	El agente RBC informa el resultado obtenido del mapeo de índices a casos
4	Respuesta de mapeo ontología-casos	El agente RBC informa el resultado del mapeo de un concepto ontológico instanciado a un conjunto de casos
5	Recuperación de casos	El Modelo o Sistema RBC es consultado por el controlador haciendo uso de índices, para la recuperación de casos
6	Consulta al Modelo	El Modelo o Sistema RBC es accesado para obtener información del contenido del repositorio de información
7	Respuesta de consulta al Modelo	El Modelo o Sistema RBC provee información sobre el contenido del repositorio
8	Actualización del Modelo	El repositorio es modificado a través del Modelo o Sistema RBC
9	Comunicación de Actualización al Modelo	El agente de sitio es informado acerca de una modificación en la librería de casos mediante un mensaje ACL
10	Consulta a ontologías	El agente RBC consulta al agente de ontologías
11	Respuesta de ontologías	El agente de ontologías responde al agente RBC

Cuadro 3.1: Colaboración entre los distintos agentes (estas colaboraciones aparecen como números encerrados entre corchetes en las figuras 3.9 a la 3.12 dentro del campo responsabilidades).

CONTROLADOR	
Modelo de Rol: Modelo de los agente de sitio-RBC-ontologías.	
Conexión con otros roles: Contenida por el rol del agente RBC.	
Descripción: Este rol sirve como suministrador de información y por tanto es un rol escencial del agente RBC. Este rol maneja peticiones de información hechas por el agente de sitio, adaptándolas para poder consultar al modelo (Sistema RBC) y proveer una respuesta. También controla el sistema de consultas al agente de ontologías.	
Responsabilidades:	**Colaboradores:**
[1, 3] Estar atento a peticiones de mapeo índices-casos y emitir una respuesta con la información.	$\Leftarrow \Rightarrow$ agente de sitio
[2, 4] Estar atento a peticiones de mapeo otología-casos y emitir una respuesta con la información.	$\Leftarrow \Rightarrow$ agente de sitio
[10, 11] Consultar al agente de ontologías y estar atento a su respuesta.	$\Leftarrow \Rightarrow$ agente de ontologías
Interfaces externas:	
Consultar al modelo en respuesta a una petición del agente de sitio	
Prerrequisito:	
Ninguno	

Figura 3.8: Rol del Controlador.

MODELO	
Modelo de Rol: Modelo de los agente de sitio-RBC-ontologías.	
Conexión con otros roles: Contenida por el rol del agente RBC.	
Descripción: Este rol es el cerebro del agente RBC pues contiene al sistema RBC que le permite al agente utilizar el paradigma del razonamiento basado en casos. Este rol maneja la información de la librería de casos. Proporciona un sistema de extracción de casos usando índices y tambien permite la consulta y modificación de la información contenida en el repositorio.	
Responsabilidades:	**Colaboradores:**
[6, 7] Proveer información en demanda y permitir la modificación de la información almacenada en el repositorio.	$\Leftarrow \Rightarrow$ Vista
[5] Acceso a casos por medio de un sistema de indexación.	$\Leftarrow \Rightarrow$ Controlador
[9] Aviso de actualización del modelo.	$\Rightarrow$ agente de sitio
Interfaces externas:	
Utilizar el paradigma del razonamiento basado en casos y encapsular información de la librería de casos	
Prerrequisitos:	
Ninguno	

Figura 3.9: Rol del Modelo.

VISTA	
Modelo de Rol: Modelo de los agente de sitio-RBC-ontologías.	
Conexión con otros roles: Contenida por el rol del agente RBC.	
Descripción: Este rol es opcional; este rol es usado solamente cuando se requiere del servicio de modificación al repositorio [6,7] asi como también del monitoreo del mismo [9].	
Responsabilidades:	**Colaboradores:**
[6, 7] Proveer información en demanda y permitir la modificación de la información almacenada en el repositorio.	⇐ ⇒ Modelo
Interfaces externas:	
Visualizar el contenido del modelo y realizar cambios en la información contenida dentro del repositorio a través del modelo.	
Prerrequisitos:	
Ninguno	

Figura 3.10: Rol de la vista.

ROL DEL AGENTE DE SITIO	
Modelo de Rol: Modelo de los agente de sitio-RBC-ontologías.	
Descripción: Intermediario entre el agente RBC y los agentes personales.	
Responsabilidades:	**Colaboradores:**
[1, 3] Hacer y recibir consultas de mapeo índices-casos.	⇐ ⇒ Controller
[2, 4] Hacer y recibir consultas de mapeo ontología-casos.	⇐ ⇒ Controller
Interfaces externas:	
Solicita un mapeo de ontología-casos y/o índices-casos en el momento que un usuario se da de alta dentro del sistema JITIK o cambia su perfil de intereses. Ser intermediario ante la solicitud de mapeo de índices a casos. Comunicar las modificaciones en la librería de casos a los agente personales.	
Prerrequisitos:	
Ninguno	

Figura 3.11: Rol del agente de sitio.

ROL DEL AGENTE DE ONTOLOGÍAS	
Modelo de Rol: Modelo de los agente de sitio-RBC-ontologías.	
Descripción: Proporciona información ontológica a los agentes de JITIK.	
Responsabilidades:	**Colaboradores:**
[10, 11] Recibir consultas sobre la ontología y elaborar respuestas a las mismas.	⇐ ⇒ Controller
Interfaces externas:	
Carga ontologías en distintos formatos como por ejemplo DAML+OIL. Realiza consultas sobre la taxonomía de la ontología y reporta resultados.	
Prerrequisitos:	
Ninguno	

Figura 3.12: Rol del agente de ontologías.

la sección 3.2.

Antes de empezar a describir el *sistema RBC* es necesario mencionar algunos puntos importantes. El *sistema RBC* tendrá dos cosas en común con la consola (shell) comercial CARET (ver sección 2.7.5). Primero, el *sistema RBC* leerá una fuente de información de casos que soporte el modelo relacional de datos [19] y no precisamente una base de datos ya que pueden escogerse como fuente de datos: archivos de texto, flujos sobre sockets en internet, sistema de archivos, entre otros. Segundo, el *sistema RBC* contará con un sistema de recuperación basado en similitud (índices de similitud) pero a diferencia de CARET no será necesario tener que especificar una *jerarquía abstracta* como la vista en la sección 2.7.5; además poseerá un sistema de indexación basado en el modelo relacional de datos para no tener que procesar cada registro dentro de la fuente de datos (índices de partición).

Como se pudo apreciar en el capítulo 2.7, cada problema, aplicación y dominio distintos dan diversas oportunidades para aplicar el paradigma del razonamiento basado en casos. En los trabajos aquí presentados se han utilizado técnicas diferentes para la indexación y selección (que son las que interesan en esta investigación) de casos, siempre buscando sean las más apropiadas para la aplicación y dominios particulares de cada problema. Así pues, las técnicas aquí descritas se han utilizado en problemas tales como diagnóstico, explicación de fallas, planeación, diseño, etc. El objetivo aquí es el desarrollar un sistema RBC lo suficientemente genérico para poder utilizarlo sobre distintos dominios y en la solución de problemas tan variados como los vistos en la sección 2.7.

La figura 3.13 muestra el diagrama *use case* del *Razonador RBC*. En este diagrama

se pueden distinguir cuatro roles diferentes que todo razonador, independientemente del contexto en que es empleado, debe tener. Debe quedar claro que la parte de modificación y la parte de almacenamiento (aprendizaje), no van a ser implementadas; ya que no son necesarias para los fines de este trabajo. Sin embargo, se implementarán clases abstractas de java de estas partes del RBC que podrán ser utilizadas en un futuro para una aplicación en particular.

La figura 3.14 muestra las partes más importantes del *sistema RBC*, mismas que son descritas a continuación:

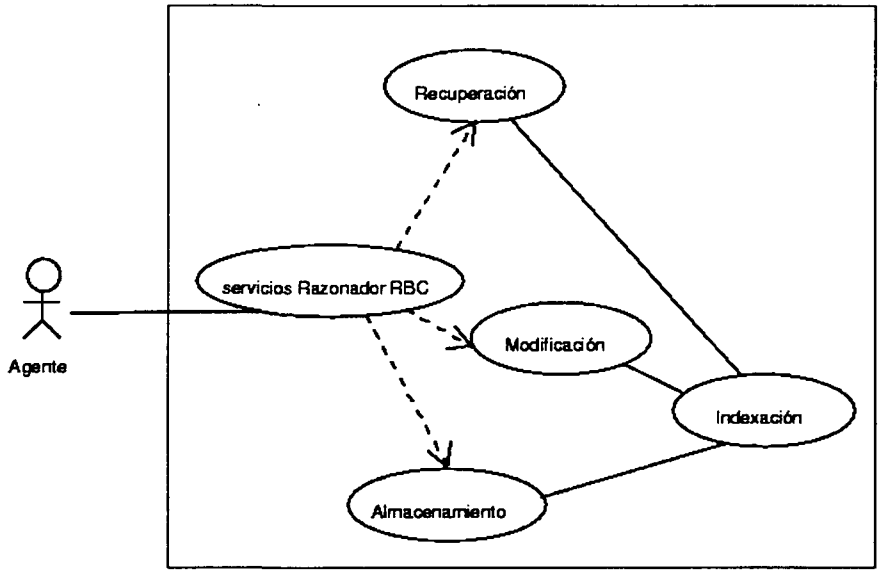


Figura 3.13: Diagrama de casos de uso del sistema RBC

1. **Indexación.** El propósito de la indexación es el de elaborar la descripción del caso, usando técnicas de indexación, para calcular o asignar índices de forma que éste pueda ser recuperado o almacenado. Para lograr esto, el *sistema RBC* utiliza dos índices distintos: *índices de partición* e *índices de similitud* (*PI* y *SI* por su nombre en inglés). Los *PI* contienen información necesaria para extraer un conjunto de casos de la librería de casos con la finalidad de no procesar todos los casos dentro del repositorio (ver sección 2.7.5). Normalmente estos índices contienen operadores que comúnmente son aplicados en teoría de conjuntos como unión, intersección, complemento, etc. Una buena elección de formato para estos índices podría ser el lenguaje estándar y ya establecido SQL (Structured Query Language) [19]. Un ejemplo de un índice de partición puede ser `SELECT`

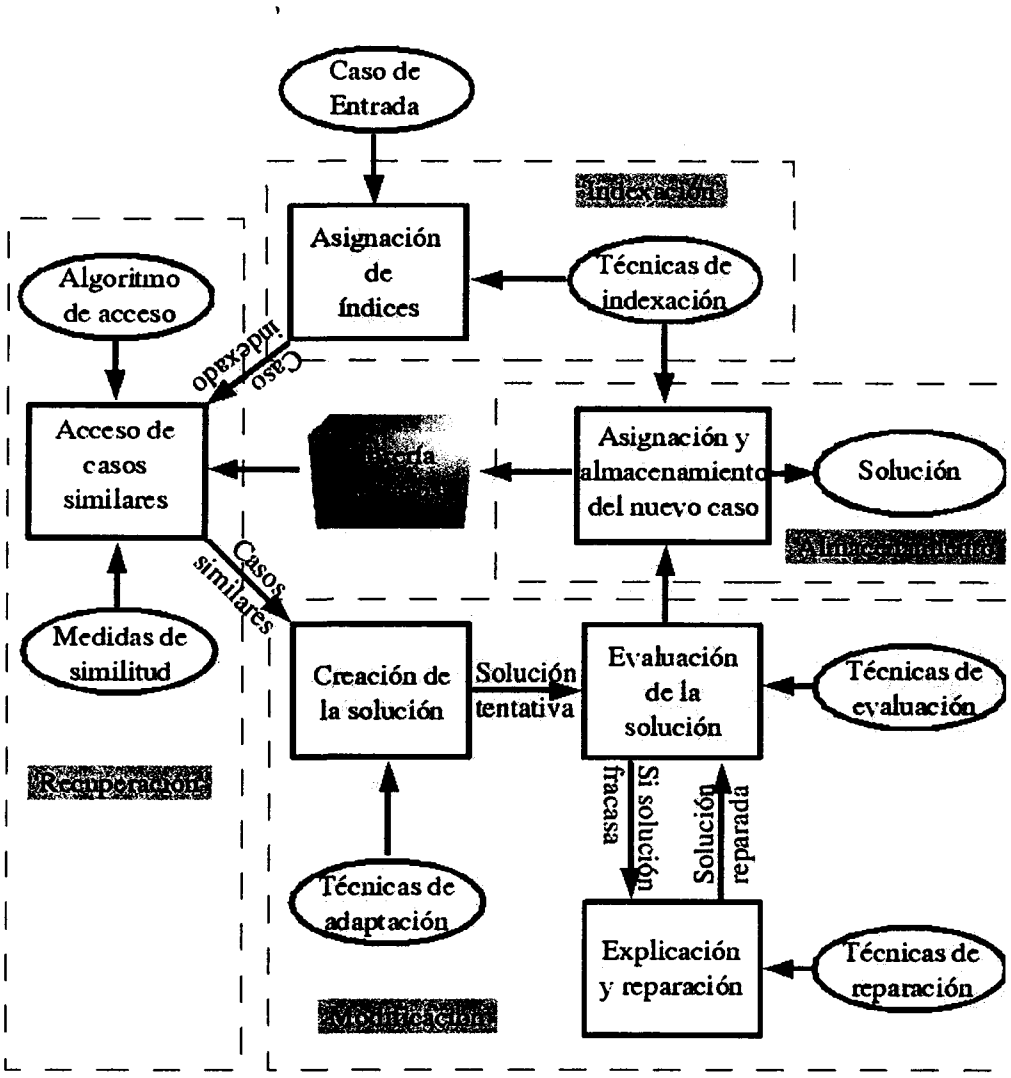


Figura 3.14: Módulos del sistema RBC dentro del algoritmo básico del razonamiento basado en casos mostrado en la figura 2.4.

`Profesores.intereses from Profesores` (suponiendo que se utilice el lenguaje SQL). Los *SI* contienen información necesaria para calificar al conjunto de casos extraídos de la librería de casos por los *PI*. Estos índices le dan información importante al *agente RBC* para que pueda decidir que tan buenos o malos son los casos extraídos y dan una descripción del problema o situación que se va a resolver. Normalmente estos índices contienen operadores binarios como por ejemplo: *atributo OP valor*. Un ejemplo de un índice de similitud puede ser `intereses` $\sim$ `Artificial Intelligence`, en donde el operador $\sim$ indique que se buscan áreas de interés afines o similares a la inteligencia artificial.

2. **Recuperación.** Es aquí donde se acceden casos similares usando algoritmos de acceso y las medidas de similitud para su posterior modificación. Se puede pensar de una librería de casos como un tipo especial de base de datos. Como en las bases de datos, esta librería almacena un gran número de registros y uno tiene que ser capaz de recuperar registros apropiados en una cantidad razonable de tiempo aún cuando los casos almacenados sean miles o cientos de miles. Las medidas de similitud u operadores de los índices de similitud (SI), son descritas en las tablas 3.2, 3.3, 3.4 y 3.5 y muestran los tipos de atributos manejados así como también las medidas de distancia implementadas para cada tipo de atributo. La ecuación 3.1 computa la similitud entre los casos Q y C. Las tablas 3.7 y 3.8 muestra ejemplos sobre consultas realizadas a la base de casos de la tabla 3.6 sobre atributos de tipo fecha, numérico, lógico y cadena.

3. **Modificación.** En la *modificación* es creada la solución, evaluada y si es necesario explicada y reparada usando técnicas de adaptación, técnicas de evaluación y técnicas de reparación. Estas técnicas no van a ser implementadas pero se diseñarán métodos abstractos para cada una de ellas.

4. **Almacenamiento.** Asignación y almacenamiento de la solución. Una vez creada y evaluada la solución, se procede a entregar la solución. Además, se puede crear un nuevo caso. Este caso se forma no solo de la solución encontrada, sino del caso o problema original, la solución encontrada, y las justificaciones y explicaciones (si las hay). Al nuevo caso se le asignan índices y se almacena en la base de casos. Esta fase del aprendizaje RBC no es implementada, pero se elaborará una clase abstracta que encapsule esta funcionalidad.

$$S(Q,C) = 1 - \sqrt{\frac{\sum_{i=1}^{n} d(Q_i, C_i)^2}{\sum_{i=1}^{n} W_i^2}} \tag{3.1}$$

Operador	Medida de distancia
$\sim$	$d(x,y) = (1 - \frac{\sqrt{(x-y)^2}}{X_{max} - X_{min}})w$
$!\sim$	$d(x,y) = (\frac{\sqrt{(x-y)^2}}{X_{max} - X_{min}})w$
%	no definido
!%	no definido

Cuadro 3.2: Operadores definidos para el atributo de tipo numérico (`Number`). Donde x es el valor numérico (obtenido por el método `doubleValue()`) del atributo en el caso de consulta (query), y es el valor numérico (obtenido por el método `doubleValue()`) en el caso dentro de la base de casos , w es el peso del atributo (importancia), y X_{max} y X_{min} son el máximo y mínimo valor numérico (obtenido por el método `doubleValue()`) respectivamente encontrados en la librería de casos.

Operador	Medida de distancia
$\sim$	$d(x,y) = (1 - \frac{\sqrt{(x-y)^2}}{X_{max} - X_{min}})w$
$!\sim$	$d(x,y) = (\frac{\sqrt{(x-y)^2}}{X_{max} - X_{min}})w$
%	no definido
!%	no definido

Cuadro 3.3: Operadores definidos para el atributo de tipo fecha (`Date`). Donde x es el valor de tiempo (obtenido por el método `getTime()`) del atributo en el caso de consulta (query), y es el valor de tiempo (obtenido por el método `getTime()`) en el caso dentro de la base de casos ,w es el peso del atributo (importancia), y X_{max} y X_{min} son el máximo y mínimo valor de tiempo (obtenido por el método `getTime()`) respectivamente encontrados en la librería de casos.

Operador	Medida de distancia
$\sim$	$d(x,y) = \begin{cases} 0 & \text{si } x = y \\ w & \text{si } x \neq y \end{cases}$
!$\sim$	$d(x,y) = \begin{cases} w & \text{si } x = y \\ 0 & \text{si } x \neq y \end{cases}$
%	no definido
!%	no definido

Cuadro 3.4: Operadores definidos para el atributo de tipo lógico (`Boolean`). Donde x es el valor lógico (`true` o `false`) del atributo en el caso de consulta (query), y es en valor lógico (`true` o `false`) en el caso dentro de la base de casos y w es el peso del atributo (importancia).

Operador	Medida de distancia
$\sim$	$d(x,y) = (1 - lcs(x,y)) \times w$
!$\sim$	$d(x,y) = lcs(x,y) \times w$
%	$d(x,y) = \begin{cases} 1{,}0 - \frac{mult}{m/n} & \text{si } m > n \\ 1{,}0 - \frac{mult}{n/m} & \text{si } m \leq n \end{cases}$
!%	$d(x,y) = \begin{cases} \frac{mult}{m/n} & \text{si } m > n \\ \frac{mult}{n/m} & \text{si } m \leq n \end{cases}$

Cuadro 3.5: Operadores definidos para el atributo de tipo cadena (`String`). Donde x es la cadena del atributo en el caso de consulta (query), y es la cadena en el caso dentro de la base de casos, w es el peso del atributo (importancia), $lcs(x,y)$ es la distancia de las dos cadenas usando el algoritmo de subsecuencias comunes más largas (*Longest Common Subsequences*, ver sección 4.2.1) , *mult* es el número de veces que la cadena x es encontrada dentro de la cadena y o viceversa (dependiendo de qué cadena sea más larga), m es la longitud de la cadena x, y n es la longitud de la cadena y (este operador dice la proporción en que están combinadas las cadenas de entrada y es muy usado en el análisis de textos, ver sección 4.2.1).

No. de caso	*fecha*	*numero*	*logico*	*cadena*
1	19/06/94 05:34:23 p.m.	1	false	Hola Mundo
2	19/06/95 05:34:23 p.m.	2	false	Hola
3	19/06/94 06:34:23 p.m.	3	true	Mundo

Cuadro 3.6: Ejemplo de una base de casos utilizada por las tablas 3.7 y 3.8.

Similitud	*No. de caso*	*fecha ~ 1994-06-19 17:34:23*
1.0	1	19/06/94 05:34:23 p.m.
0.99	3	19/06/94 06:34:23 p.m.
0.0	2	19/06/95 05:34:23 p.m.
Similitud	*No. de caso*	*fecha !~ 1994-06-19 17:34:23*
1.0	2	19/06/95 05:34:23 p.m.
1.14E-4	3	19/06/94 06:34:23 p.m.
0.0	1	19/06/94 05:34:23 p.m.
Similitud	*No. de caso*	*numero ~ 3*
1.0	3	3
0.5	2	2
0.0	1	1
Similitud	*No. de caso*	*numero !~ 3*
1.0	1	1
0.5	2	2
0.0	3	3
Similitud	*No. de caso*	*logico ~ true*
1.0	3	true
0.0	1	false
0.0	2	false
Similitud	*No. de caso*	*logico !~ false*
1.0	3	true
0.0	1	false
0.0	2	false

Cuadro 3.7: Ejemplo de consultas utilizando el índice de similitud sobre atributos de tipo fecha, numérico, y lógico en la base de casos de la tabla 3.6.

Similitud	*No. de caso*	*cadena* $\sim$ *Hola Mundo*
1.0	1	Hola Mundo
0.5	3	Mundo
0.4	2	Hola
Similitud	*No. de caso*	*cadena* $!\sim$ *Hola Mundo*
0.6	2	Hola
0.5	3	Mundo
0.0	1	Hola Mundo
Similitud	*No. de caso*	*cadena % Hola*
1.0	2	Hola
0.4	1	Hola Mundo
0.0	3	Mundo
Similitud	*No. de caso*	*cadena !% Hola*
1.0	3	Mundo
0.6	1	Hola Mundo
0.0	2	Hola

Cuadro 3.8: Ejemplo de consultas utilizando el índice de similitud sobre atributos de tipo cadena en la base de casos de la tabla 3.6.

3.5. Especificaciones del servicio de mapeo de índices a casos

En la sección 2.6.1, se explicó la necesidad de utilizar algún método de indexación de casos para un rápido acceso a ellos. Este servicio que el *agente RBC* brinda está íntimamente relacionado con el mecanismo de indexación que es implementado por el módulo de indexación dentro del *sistema RBC*. Este servicio es iniciado por el *agente de sitio* al enviar un mensaje ACL (tipo `REQUEST`) al *agente RBC* en respuesta a una petición de un *agente personal* (recordar que este servicio no es automático). Este mensaje tendrá la información de los índices necesarios para que el *Sistema RBC* con que cuenta el *agente RBC* sea capaz de aplicar la técnica del razonamiento basado en casos (índices de partición e índices de similitud). Una vez aplicado el razonamiento basado en casos, el *agente RBC* responderá con las soluciones encontradas dentro del proceso de recuperación, adaptación, evaluación-reparación y almacenamiento mediante un mensaje ACL de tipo `INFORM`. Este es el servicio básico del *agente RBC*.

Este servicio estará disponible siempre y cuando la configuración de la librería de casos sea correcta. En la figura A.6 se muestra como configurar a la librería de casos. Se debe hacer notar que el *agente de sitio* no activa este servicio de manera automática, como es el caso en el mapeo de una instancia ontológica a casos que se activa cuando un *agente personal* se registra. Este servicio debe ser activado directamente por el *agente personal*.

3.6. Especificaciones del servicio de mapeo de instancia ontológica a casos

Este servicio es posiblemente el más útil y poderoso que el *agente RBC* brinda, ya que este servicio conecta una ontología de dominio específico con uno o más casos adaptados y evaluados provenientes de una fuente de información convencional como un sistema de archivos, una base de datos, un archivo de texto, etc.

Este servicio es iniciado cuando el *agente de sitio* envía un mensaje ACL de tipo `REQUEST` al *agente RBC* (ver sección 3.8) con el nombre de la instancia a mapear[2] y el *agente RBC* responde con una descripción de las acciones realizadas dentro del proceso de recuperación, modificación y aprendizaje, así como también con las soluciones encontradas. Este servicio estará disponible siempre y cuando la configuración de la librería de casos y la ruta a la ontología estén especificadas correctamente. En la figura A.6 se muestra como configurar a la librería de casos y a la ontología.

[2]Esta instancia puede representar al área de interés del usuario, su puesto en el organigrama, o cualquier otra entidad que describa la información que es relevante para el usuario.

Supongamos que el *agente RBC* maneja una librería de casos configurada adecuadamente, al igual que una ontología válida para el *agente de ontologías*. Dado que el *sistema RBC* que maneja el *agente RBC* mapea índices (PI y SI) a casos, ahora la pregunta es: ¿Cómo extraerá el *agente RBC* los índices de partición y similitud a partir de una ontología? Esta pregunta será contestada durante las siguientes tres secciones (secciones 3.6.1, 3.6.2 y 3.6.3) haciendo uso de la teoría de grafos[3].

3.6.1. Mecanismo de búsqueda de índices

El *agente RBC* debe ser capaz de extraer el conocimiento necesario dentro de la ontología para poder inferir los índices con los cuales puede acceder a la librería de casos. Para esto, el agente "navega" a través de la ontología consultando cada nodo en un orden que es determinado por el tipo de nodos por el cual visita.

Más formalmente, sea $G = \{V, E\}$ un digrafo [4] donde $V = \{v_1, v_2, \ldots, v_i\}$ el conjunto de todas las instancias (nodos) de la ontología y $E = \{e_1, e_2, \ldots, e_i\}$ el conjunto de todos los predicados[4] (arcos) de la ontología. Supongamos que el agente está visitando o consultando la instancia (nodo) v, entonces la secuencia de búsqueda estará determinada por cada uno de los tres tipos de instancias diferentes que v puede tener y que el *agente RBC* puede distinguir:

1. **Expansión de la búsqueda en una instancia dirigida por rango.** Sea v una instancia (nodo) dirigida por rango expandida al tiempo t y sea $E_i = \{e_1, \ldots, e_n\}$ el conjunto de todos los predicados (arcos) cuyo dominio sea v, entonces el *agente RBC* expandirá su búsqueda en cada elemento que sea rango de E_i a un tiempo $t' > t$ y en un orden no determinado [5](ver tabla 3.9).

2. **Expansión de la búsqueda en una instancia dirigida por dominio.** Sea v una instancia (nodo) dirigida por dominio expandida al tiempo t y sea $E_i = \{e_1, \ldots, e_n\}$ el conjunto de todos los predicados (arcos) cuyo rango sea v, entonces el *agente RBC* expandirá su búsqueda en cada elemento que sea dominio de E_i a un tiempo $t' > t$ y en un orden no determinado (ver tabla 3.9).

3. **Expansión de la búsqueda en una instancia dirigida por jerarquía.** Sea v una instancia (nodo) dirigida por jerarquía expandida al tiempo t y sea $Sub_i = \{v_1, \ldots, v_n\}$ el conjunto de todas las instancias que tengan la clase[6] de v como superclase, y sea $Sup_i = \{w_1, \ldots, w_n\}$ el conjunto de todas las instancias cuya clase sea heredada por v, entonces el *agente RBC* expandirá su búsqueda en cada

[3]Una ontología puede ser bien descrita mediante un grafo.

[4]$e_j = \{v_k, v_l\}$ donde v_k es el dominio de e_j y v_l es el rango.

[5]El comportamiento no determinista es a consecuencia que el *agente RBC* no puede saber el orden en el cual el *agente de ontologías* dará los resultados de los predicados, instancias o clases.

[6]Cada instancia pertenece a una o más clases.

elemento de Sub_i a un tiempo $t' > t$ en un orden no determinado si es que la configuración de la instancia v (ver tabla 3.9) así lo indica, de caso contrario expandirá su búsqueda en cada elemento de Sup_i a un tiempo $t' > t$ en un orden no determinado.

3.6.2. Criterios de paro para la búsqueda

Supongamos que al tiempo t el *agente RBC* visita una instancia v que fue sucesora de alguna instancia w que fue visitada a un tiempo $t' < t$, entonces el *agente RBC* no expandirá su búsqueda a partir del nodo v si y solo si se dan cualquiera de las siguientes condiciones:

- Si $v = w$, o en otras palabras, si v fue visitada anteriormente a un tiempo $t' < t$.
- Si la clase a la que pertenece v es una clase primitiva[7].

3.6.3. Mecanismo de extracción de índices

Ya que se ha propuesto el mecanismo de búsqueda de índices y su respectivo criterio de paro, ahora solo falta describir el mecanismo de extracción de índices.

Se van a extraer dos tipos de índices:

1. **Extracción de los índices de partición (*PI* por su nombre en inglés).** Los *PI* van a ser extraídos de la ontología por el *agente RBC* mediante todos los predicados cuyo nombre sea `cbrPartitionIndex` o `cbrInheritPartitionIndex` (para heredar el índice a instancias expandidas a través de esta instancia). Es decir, sea v la instancia (nodo) que el *agente RBC* está consultando actualmente y sea e un predicado (arco) de nombre `cbrPartitionIndex/cbrInheritPartitionIndex` cuyo dominio sea v y su rango sea w, entonces el *agente RBC* considerará como un *PI* al valor de w. Si más de un predicado es encontrado entonces el *agente RBC* hará una operación de unión sobre los conjuntos recuperados por cada *PI*. En caso de que el *agente RBC* no encontrase alguno de estos predicados entonces ningún caso ($\{\emptyset\}$) es recuperado de la librería de casos.

2. **Extracción de los índices de similitud (*SI* por su nombre en inglés).** Los *SI* son extraídos de la ontología por el *agente RBC* mediante todos los predicados cuyo nombre sea `cbrSimilarityIndex` o `cbrInheritSimilarityIndex` (para heredar el índice a instancias expandidas a través de esta instancia). Es decir, sea v la instancia (nodo) que el *agente RBC* está consultando actualmente y sea e un

[7]La descripción ontológica del formato DAML+OIL considera como clases primitivas al `string`, `entero`, `flotante`, entre otros.

predicado (arco) de nombre `cbrSimilartyIndex/cbrInheritSimilarityIndex` cuyo dominio sea v y su rango sea w, entonces el *agente RBC* considerará como un *SI* al valor de w. Se pueden tener tantos *SI* como sean necesarios y esto normalmente tiene como consecuencia que la descripción del problema o situación que se quiere resolver sea más precisa.

```
Retrieve-Cases(O:Ontology, u:ontological-instance)
N:list-of-ontological-instances
R:set-of-cases
R ← ∅
ENQUEUE(N,u)
while N ≠ ∅ do
    v ← DEQUEUE(N)
    if v is-range-instance then
        for each w ∈ f(O,v) do
            if w is-non-terminal then
                ENQUEUE(N,w)
    else if v is-domain-instance then
        for each w ∈ h(O,v) do
            if w is-non-terminal then
                ENQUEUE(N,w)
    else if v is-hierarchy-instance then
        for each w ∈ g(O,v) do
            if w is-non-terminal then
                ENQUEUE(N,w)
    R ← R ∪ cbr(O,v)
end
```

El algoritmo mostrado anteriormente representa el proceso de mapeo de instancia ontológica a casos. El proceso empieza con una ontología O, la instancia ontológica u a ser consultada y con la inicialización de los conjuntos N y R (conjunto de instancias ontológicas a ser procesadas y conjunto de casos de respuesta respectivamente). Una vez inicializados los conjuntos N y R comienza el ciclo de búsqueda. Este ciclo de búsqueda se realiza mientras $N \neq \{\varnothing\}$, es decir, mientras el conjunto N no sea conjunto vacío. En caso de que $N = \{\varnothing\}$ entonces termina la búsqueda y se regresa el conjunto de casos R. Dentro del ciclo, primeramente se toma un elemento v dentro del conjunto N ($v \in N$). Después se procede a calcular los nodos o instancias sucesoras de v. Estos sucesores w son tomados por las funciones $f(O,v)$, $g(O,v)$, ó $h(O,v)$ de acuerdo al tipo de instancia al cual pertenezca v (instancia dirigida por rango, por jerarquía, o por dominio respectivamente), y después son agregados al conjunto de nodos a procesar N, siempre y cuando no sean un nodo terminal de acuerdo a los criterios de paro

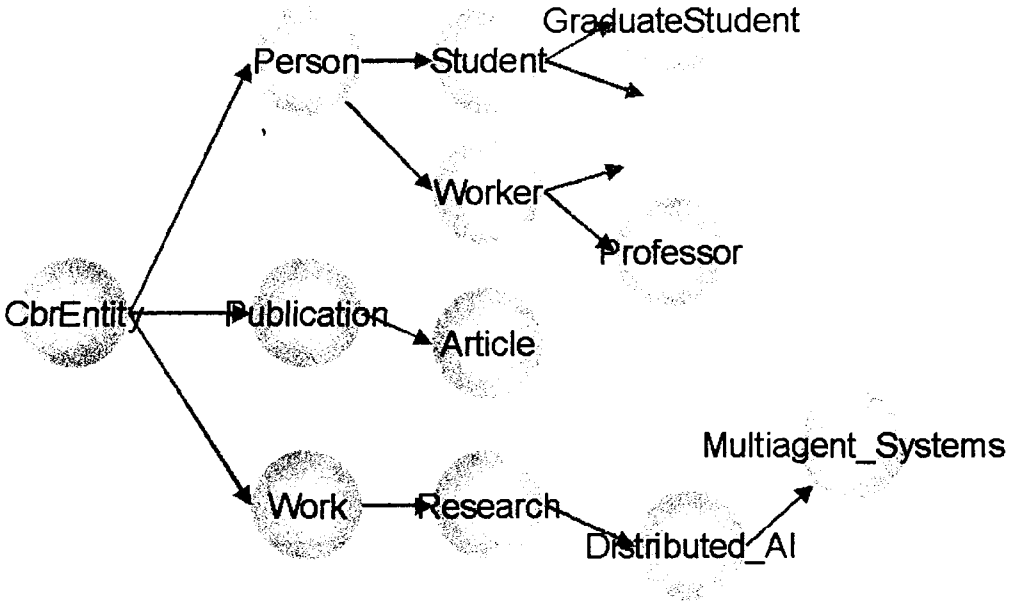


Figura 3.15: Ejemplo de una taxonomía ontológica.

descritos anteriormente. Después la función $cbr(O, v)$ extrae los índices de partición y de similitud de v (usando los predicados `cbrPartitionIndex` y `cbrSimilarityIndex`) así como también los índices de partición y similitud heredados por los progenitores de v (usando los predicados `cbrInheritPartitionIndex` y `cbrInheritSimilarityIndex`) para posteriormente usarlos dentro del mecanismo de indexación del *sistema RBC* y recuperar los casos asociados a v. Ya que fueron calculados los casos "mapeados" de v, estos son agregados al conjunto R y v es extraído del conjunto N para que nuevamente se realice el ciclo de búsqueda.

La figura 3.16 muestra un grupo de instancias ontológicas, cuya descripción taxonómica es mostrada por la figura 3.15, con sus respectivos predicados. Supongamos que al *agente RBC* se le solicitó un mapeo de la instancia ontológica `Professor_Jose_L_Aguirre` a un conjunto de casos. Inicialmente el *agente RBC* solicita al *agente de ontologías* todos los predicados en cuyo dominio se encuentre la instancia `Professor_Jose_L_Aguirre`. El *agente de ontologías* le contesta que `publicationInterests` es el único predicado y que el rango de ese predicado es `Article_1`. En este punto, el *agente RBC* trata de extraer casos del modelo (Sistema RBC) consultándolo con ningún índice hasta el momento (no hay PI ni SI). Puesto que `Professor_Jose_L_Aguirre` es una instancia dirigida por rango (el *agente RBC* siempre considera a la primer instancia como dirigida por rango a menos que esta instancia contenga un predicado `cbrConfig` que

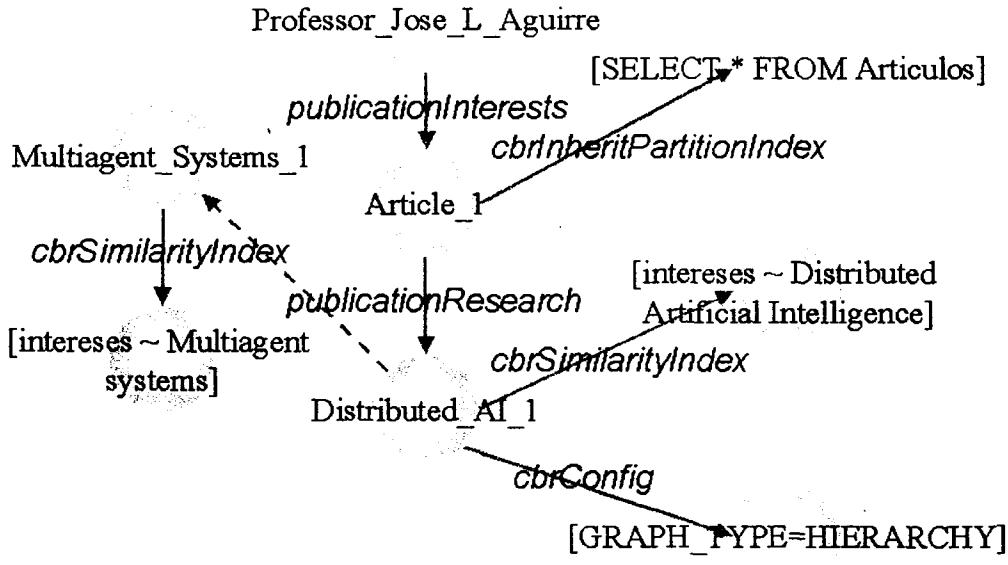


Figura 3.16: Ejemplo de una ontología instanciada de acuerdo a la taxonomía de la figura 3.15.

diga que es dirigida por dominio o jerárquica), el *agente RBC* consulta ahora a la instancia `Article_1`. El *agente de ontologías* responde con todos los predicados cuyo dominio sea `Article_1`, que en este caso son: `cbrInheritPartitionIndex` y `publicationResearch`. El rango de `cbrInheritPartitionIndex` es `[SELECT * FROM Articulos]` (los corchetes significan que es una instancia de tipo primitivo y en este caso de tipo `String`) y por tanto el agente ya tiene un índice de partición (PI) que va a ser heredado por todas las instancias a partir de `Article_1` (el predicado `cbrPartitionIndex` agrega el PI solamente a la instancia actual, y este valor no es heredado por instancias posteriormente visitadas), sin embargo, aún no tiene ningún índice de similitud (SI). El *agente RBC* trata de extraer nuevamente casos del modelo pero ahora con `PI=SELECT * FROM Articulos` y ningún `SI` (si no hay índices de similitud, entonces ningún caso es recuperado por el *sistema RBC*). Como el rango del predicado `publicationResearch` es `Distributed_AI_1`, entonces el *agente RBC* consulta esta instancia preguntando al *agente de ontologías* todos los predicados cuyo dominio sea `Distributed_AI_1` y el *agente de ontologías* responde con: `cbrSimilarityIndex` y `cbrConfig`. El rango de `cbrSimilarityIndex` es `[intereses ~ Distributed Artificial Inteligence]` y por tanto el *agente RBC* ya tiene un índice de similitud (SI) que va a ser utilizado en esta instancia solamente (el predicado `cbrInheritSimilarityIndex` hereda el índice de similitud a todas las instancias visitadas posteriormente). Ahora el *agente RBC* con-

Nombre	Valores	Explicación
GRAPH_TYPE	RANGE, DOMAIN, HIERARCHY, STATIC	Especifica el tipo de instancia. Por defecto es RANGE.
NODE_TYPE	ASCENDING, DESCENDING	Si la instancia es jerárquica, este parámetro especifica una búsqueda por superclases o subclases. Por defecto es DESCENDING.
NUM_RESULTS	n	n es el máximo número de casos a ser recuperados por cada instancia. Por defecto es 20.

Cuadro 3.9: Configuración ontológica.

sulta el modelo con `PI=SELECT * FROM Articulos` y `SI=intereses` $\sim$ `Distributed Artificial Inteligence`. El rango de `cbrConfig` es `[GRAPH_TYPE=HIERARCHY]` y esto le dice al *agente RBC* que sucesivas expansiones de nodos a partir de `Distributed_AI_1` van a ser de tipo jerárquico. Como `Distributed_AI_1` es de tipo jerárquico el *agente RBC* pregunta al *agente de ontologías* todas las instancias cuya clase sea una subclase de la clase a la que pertenece `Distributed_AI_1`. El *agente de ontologías* responde con: `Multiagent_Systems_1`. El *agente RBC* pregunta nuevamente al *agente de ontologías* todos los predicados cuyo dominio sea `Multiagent_Systems_1` y el *agente de ontologías* responde con: `cbrSimilarityIndex`. El predicado `cbrSimilarityIndex` tiene como rango el valor de `[intereses` $\sim$ `Multiagent systems]`, entonces el *agente RBC* agrega el índice de similitud dentro de la búsqueda local. Ahora el *agente RBC* consulta al modelo con `PI=SELECT * FROM Articulos` y `SI=intereses` $\sim$ `Multiagent systems`. Como `Multiagent_Systems_1` es una instancia dirigida por jerarquía, el *agente RBC* pregunta al *agente de ontologías* todas las instancias cuya clase sea una subclase de la clase a la que pertenece `Multiagent_Systems_1` y el *agente de ontologías* responde que no existe ninguna instancia con esas características. Es en este momento que termina la búsqueda en la ontología. La tabla 3.9 muestra los valores que puede tomar el rango del predicado `cbrConfig`.

3.7. Especificaciones del servicio de monitoreo / modificación de librería de casos

La construcción de un mecanismo de monitoreo puede resultar muy complicado ya que la implementación es muy dependiente de la plataforma con que se esté trabajando. Por ejemplo, el control de acceso a recursos es distinto en un sistema `UNIX` que en uno en

Windows. El tratar de llevar un control de cambios en un sistema de archivos puede ser muy ineficaz e incierto. Es por esta razón que se decidió el realizar todos los cambios a la librería de casos a través del *agente RBC* por medio de una interfaz gráfica de usuario (GUI, por sus siglas en inglés). De esta manera, si se le pide al agente RBC que haga los cambios a la librería de casos, éste comunicará al *agente de sitio* sin necesidad de estar monitoreando la fuente de datos. La interfaz implementada ofrece al administrador de *JITIK* la capacidad de agregar nuevos casos al repositorio o bien, eliminar o modificar uno ya existente. Además, le da un control total sobre la estructura de los casos puesto que implementa funciones de agregar o quitar atributos a la estructura de casos.

Este servicio es activado en el momento que se elige el mostrar al *agente RBC* con GUI dentro de la configuración de arranque. Es importante que se indique también dentro de la configuración, la dirección del agente que va a recibir los mensajes de modificación enviados por el *agente RBC* ya que sin este parámetro el *agente RBC* enviará mensajes a todos los *agentes de sitio* registrados dentro del sistema JITIK. Cuando el *agente RBC* detecte alguna acción de modificación por parte de la vista, informará el cambio mandando un mensaje ACL (de tipo INFORM) en donde el contenido del mensaje contendrá el comando SQL utilizado. El lenguaje SQL es muy adecuado para la descripción de estos cambios a la librería de casos (los comandos SQL más comunes son UPDATE, CREATE, DROP, ver apéndice B).

A continuación se listan todas las modificaciones que se pueden realizar a través del GUI del *agente RBC*:

1. Agregar una nueva base de casos. El GUI del *agente RBC* preguntará por el nombre de la base de casos a crear y después preguntará por los tipos de atributos que esta base de casos tendrá.

2. Borrar una base de casos. El GUI del *agente RBC* preguntará por el nombre de la base de casos a borrar.

3. Agregar un nuevo atributo a una base de casos existente. El GUI del *agente RBC* mostrará los tipos de atributos soportados por el repositorio.

4. Borrar un atributo de una base de casos existente. El GUI del *agente RBC* preguntará por el nombre del atributo a eliminar.

5. Agregar un nuevo caso a una base de casos existente. El GUI del *agente RBC* preguntará por los valores de los atributos del caso a insertar.

6. Borrar un caso existente de una base de casos. El GUI del *agente RBC* borrará el caso seleccionado.

7. Modificar un caso existente. Se el valor de un atributo en un caso dentro de la base de casos.

3.8. Adaptaciones al agente de sitio

Para que el *agente de sitio* pueda dar el servicio que otorga el *agente RBC* se implementó un comportamiento que otorga las siguientes funcionalidades:

1. Solicitud automática de mapeo de ontología a casos en el momento en que un *agente personal* se registra dentro del sistema JITIK. Una vez que el *agente de sitio* detecta el registro de un *agente personal* le solicita al *agente RBC* el mapeo de una instancia ontológica a casos; el nombre de la instancia ontológica a ser mapeada será tomada de la dirección local (sin la parte alta) del *agente personal* dentro del sistema JITIK, por ejemplo, si el *agente personal* se llama `Jose_L_Aguirre@host:1099/JADE` el nombre de la instancia a ser mapeada será `Jose_L_Aguirre`.

2. Solicitud explícita de mapeo de ontología a casos en el momento que un *agente personal* lo solicite. Esta funcionalidad es un agregado del servicio anterior ya que un *agente personal* puede solicitarlo en cualquier momento bajo alguna situación importante, como por ejemplo, el cambio del perfil de intereses del *agente personal* o cambio de la ontología.

3. Solicitud explícita de mapeo de índices a casos en el momento que el *agente personal* lo solicite. Aunque el mapeo de índices a casos es un servicio de más "bajo nivel" que el servicio de mapeo ontológico (puesto que el *agente personal* solicitante debe conocer el sistema de indexación de casos manejado por el *agente RBC* y además la estructura del repositorio de casos), la importancia de este servicio se hace evidente en aplicaciones en que se requiere una utilización muy específica de casos (ver sección 4.2.1).

4. Aviso a los *agente personales* sobre modificaciones realizadas a la librería de casos por el *agente RBC*. Es importante aclarar que es difícil de saber e implementar a qué *agente personal* (usuario JITIK) le podría interesar alguna modificación de la librería de casos y es por eso que el *agente de sitio* le notifica a todos los *agente personales* de JITIK. Por ejemplo, supongamos que a través del `GUI` del *agente RBC* se agrega una nueva base de casos. Esta modificación de la librería de casos es notificada por el *agente RBC* al *agente de sitio*, pero ¿a qué agente personal le podría interesar la creación de una nueva base de casos? Tal vez este problema tenga varias soluciones pero el diseño e implementación de las mismas podrían alargar esta investigación y se decidió no realizar esta tarea.

3.9. Implementación

Esta sección presenta una síntesis del apéndice A. El lenguaje de programación a ser usado es java [10] en conjunto con las librerías de JADE ya que el sistema JITIK está implementado usando estas librerías.

La implementación del *agente RBC* va a ser dividida en cuatro partes: la primera parte de la implementación será dedicada al *sistema RBC* o modelo, la segunda parte al controlador, la tercera parte a la vista, y por último, la cuarta parte será destinada a las adaptaciones del *agente de sitio.*

La figura 3.17 muestra el diagrama de clases del *agente RBC.* Se puede observar que la clase de java `CbrAgent` usa distintas clases para la tarea del modelo (`CbrEngine`), tarea del controlador (`CbrIndexMappingService` y `CbrOntologyMappingService`), y tarea de la vista (`CbrAgentGui`), así como también algunas otras clases de apoyo.

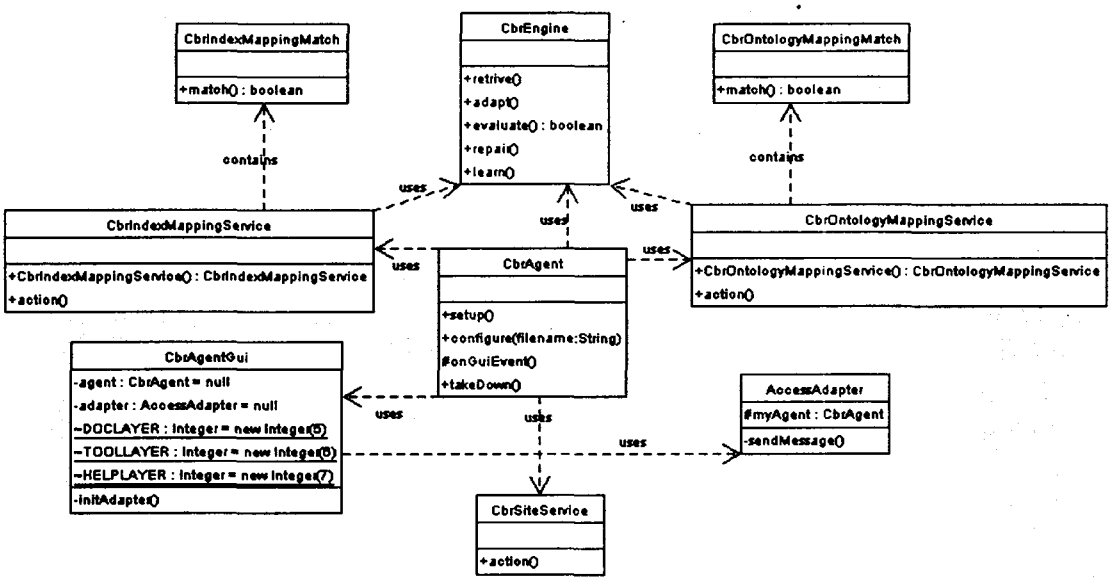


Figura 3.17: Diagrama de clases del agente RBC.

3.10. Implementación del Sistema RBC

1. **CbrEngine.** Esta clase se encarga de controlar al modulo de indexación (clase `CbrIndexer`), recuperación (clase `CbrRetriever`), modificación (clase `CbrModifier`) y almacenamiento (clase `CbrStorer`). Esta clase representa la implementación del rol Modelo. Los métodos principales de esta clase son: `retrieve()`, `adapt()`, `evaluate()`, `repair()` y `learn()`.

2. **CbrIndexer.** Esta clase se encarga de la indexación utilizando dos índices distintos: *índices de partición* e *índices de similitud* (*PI* y *SI* por su nombre en inglés). Los métodos más importantes de esta clase son: `createCase()` y `parseSimilarityIndex()`.

3. **CbrRetriever.** Esta clase de java accesa casos similares usando algoritmos de acceso y las medidas de similitud para su posterior modificación. Los métodos principales de esta clase son: `retrieve()` y `retrieveAlgoritmh()`.

4. **CbrCase.** Esta es la clase abstracta que representa cualquier tipo de caso en cualquier librería de casos. Esta es una clase que debe ser heredada e instanciada para su uso posterior. El método principal de esta clase es: `computeSimilarity()`, que tiene como finalidad el comparar que tan similar es un caso con otro (medidas de similitud).

5. **CbrTrait.** Esta clase abstracta representa o describe un atributo de un caso. Esta clase al igual que `CbrCase` debe ser heredada e instanciada para su posterior uso. El método principal de esta clase es `getSimilarity()` que tiene como finalidad el comparar dos atributos homónimos de dos casos distintos.

3.11. Implementación del controlador

El controlador esta formado principalmente por dos clases de java que se encargan de realizar los servicios de mapeo de índices a casos y de ontología a casos. Estas clases son: `CbrIndexMappingService` y `CbrOntologyMappingService`. En la figura 3.17 se pueden observar estas clases dentro del diagrama UML de clases del *agente RBC*. A continuación se describen estas dos clases de java, así como también sus principales métodos.

1. **CbrIndexMappingService.** Esta clase de java se encarga de recibir las peticiones del *agente de sitio* que solicitan el mapeo de índices a casos. `CbrIndexMappingService` extiende la funcionalidad de la clase de JADE `CyclicBehaviour` y está activa mientras dure la vida del *agente RBC* y aparte esté configurado adecuadamente el *sistema RBC* (ver figura A.6). Esta clase recibe todos los mensajes ACL de tipo `REQUEST` que sean enviados por la clase de java `CbrIndexMappingRequest`, misma que utiliza el *agente de sitio* para realizar la petición. El filtro para los mensajes con las características mencionadas anteriormente es realizado por la clase de java `CbrIndexMappingMatch`. Cuando es recibido este tipo de mensajes por la clase `CbrIndexMappingService`, es agregado un comportamiento llamado `CbrIndexMappingAction` que extiende la funcionalidad de una clase de JADE `OneShotBehaviour`. La clase `CbrIndexMappingAction` realiza la consulta

directamente a la clase CbrEngine (modelo) y regresa los resultados mediante un mensaje ACL de tipo INFORM al *agente de sitio* solicitante.

2. **CbrOntologyMappingService.** Esta clase de java se encarga de recibir las peticiones del *agente de sitio* que solicitan el mapeo de una instancia ontológica a casos. CbrOntologyMappingService extiende la funcionalidad de la clase de JADE CyclicBehaviour. Esta clase recibe todos los mensajes ACL de tipo REQUEST que sean enviados por la clase de java CbrOntologyMappingRequest, misma que utiliza el *agente de sitio* para realizar la petición. El filtro para los mensajes con las características mencionadas anteriormente es realizado por la clase de java CbrOntologyMappingMatch. Cuando es recibido este tipo de mensajes por la clase CbrOntologyMappingService, es agregado un comportamiento llamado CbrOntologyMappingAction que extiende la funcionalidad de una clase de JADE CyclicBehaviour. La clase CbrOntologyMappingAction es un comportamiento cíclico que esta activo mientras se esté realizando el proceso de búsqueda de índices en la ontología y crea tantas instancias de la clase de java CbrOntologyInstance como instancias ontológicas consulte. El proceso de búsqueda de índices, los criterios de paro y los mecanismos de extracción de índices son implementados por la clase de java CbrOntologyInstance (ver secciones 3.6.1 a la 3.6.3). Una vez realizado el proceso de extracción de índices se envía un mensaje ACL de tipo INFORM con los resultados encontrados al *agente de sitio* solicitante.

3.12. Implementación de la vista

La vista esta implementada por una serie de clases de java manejadas por la clase CbrAgentGui. CbrAgentGui es una clase que extiende la clase de java JFrame. La clase más importante manejada por la clase CbrAgentGui es CaseLibraryDocumentFrame. Esta clase realiza consultas SQL92 a la librería de casos, y muestra la información en una tabla (JTable). Todas las operaciones de modificación descritas en la sección 3.7 son realizadas por este frame interno mediante la clase de java AccessAdapter que es muy parecida a la clase CbrIndexer. En el momento en que se realiza alguna modificación a la librería de casos, la clase AccessAdapter da aviso al *agente de sitio* mediante un mensaje ACL de tipo INFORM.

3.13. Implementación de las adaptaciones al agente de sitio

Todas las adaptaciones descritas en la sección 3.8 son implementadas por la clase de java CbrSiteService que extiende de la clase de JADE CyclicBehaviour.

Capítulo 4

Pruebas de funcionamiento y evaluación de desempeño

Las pruebas realizadas están divididas en dos partes. La primera parte de las pruebas mostrará el funcionamiento adecuado del *sistema RBC* o modelo. La segunda parte probará el controlador y la vista del *agente RBC*.

4.1. Pruebas al *Sistema RBC*

En esta parte se probarán todos los operadores ($\sim$, $!\sim$, %, !%) implementados sobre los cuatro tipos de atributos (numérico, fecha, lógico, y cadena) considerados en las especificaciones de la sección 3.4. También se mostrarán los distintos tiempos de respuesta a consultas realizadas en bases de casos de prueba con los cuatro diferentes atributos.

Para probar todos los operadores implementados sobre los distintos tipos de atributos se creó la base de casos mostrada en la tabla 3.6 de la sección 3.4. Esta base de casos es consultada por el modelo utilizando cada operador definido sobre los cuatro tipos de atributos. Los resultados de las consultas son mostradas en las tablas 3.7 y 3.8 de la sección 3.4. En estas tablas se observa que todas las métricas implementadas son correctas.

Para evaluar el desempeño del *Sistema RBC* fue creada una librería de casos (Base de Datos, ver apéndice B) con cuatro bases de casos (tablas o relaciones), es decir, una base de casos por cada tipo de atributo manejado por el modelo. Cada base de casos contiene el mismo número de casos (cardinalidad de la relación, ver apéndice B) y el mismo número de atributos (grado de la relación, ver apéndice B). La única diferencia entre estas bases de casos serán los tipos de atributos que contienen. Cada base de casos tiene atributos de tipo numérico, fecha, lógico (boolean) y cadena (cadena de caracteres o bytes) respectivamente.

En las figuras 4.1, 4.2, 4.3, y 4.4 se muestran los distintos tiempos de respuesta para consultas con una cardinalidad (casos) de 500, 1000 y 1500 instancias y un grado (atributos) de orden 5, 10 y 15 bases de casos con atributos de tipo numérico, fecha,

lógicos, y cadena respectivamente. El *sistema RBC* o modelo se comportó de manera estable, y los resultados de similitud entre casos fueron los esperados de acuerdo a la ecuación 3.1 mostrada en la sección 3.4.

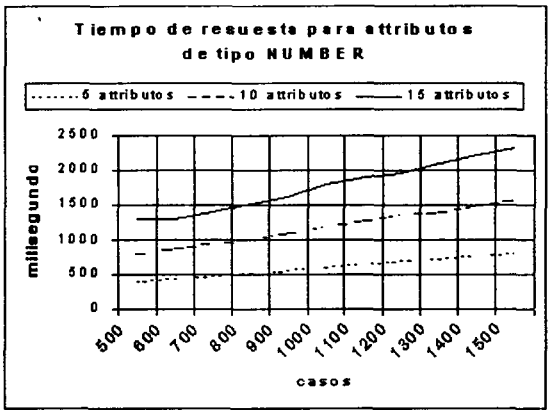


Figura 4.1: Tiempos de respuesta para atributos de tipo numérico. Estos tipos numéricos son representados dentro de la máquina virtual de java como objetos `java.lang.Number`.

4.2. Pruebas al controlador y la vista

Se van a presentar dos casos de estudio del *sistema JITIK* en conjunto con el *agente RBC* los cuales mostrarán el correcto funcionamiento del controlador. El primer caso de estudio prueba el funcionamiento adecuado del servicio de mapeo índices-casos. Este caso de estudio muestra los beneficios de contar con un sistema multi-agente enfocado únicamente a la distribución de conocimiento, y muy particularmente, a encontrar patrones en cadenas de aminoácidos y proteínas usando los operadores $\sim$ y % descritos en el capítulo anterior. El segundo caso de estudio es acerca del Centro de Sistemas Inteligentes (en el ITESM) y tiene como finalidad el comprobar el funcionamiento adecuado del servicio mapeo ontología-casos. Este caso de estudio muestra la manera en que alumnos y profesores del CSI pueden obtener información y conocimiento de acuerdo a su perfil de intereses especificado dentro de una ontología (ver apéndice C). Cabe mencionar que las pruebas al controlador están enfocadas a verificar las especificaciones descritas en las secciones 3.5, 3.6, y 3.7. Dado que el *sistema RBC* traduce de forma automática el tipo de atributo implícito dentro del índice de similitud y esta traducción es comprobada en la evaluación del modelo; no es necesario el probar con cada tipo de atributo puesto que para el controlador no existe ninguna diferencia en la representación

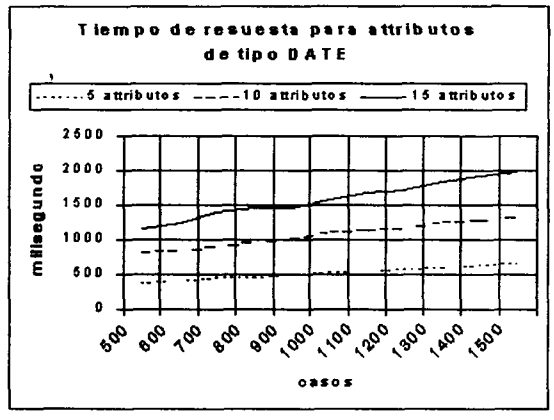


Figura 4.2: Tiempos de respuesta para atributos de tipo fecha. Estos tipos son representados dentro de la máquina virtual de java como objetos `java.util.Date`.

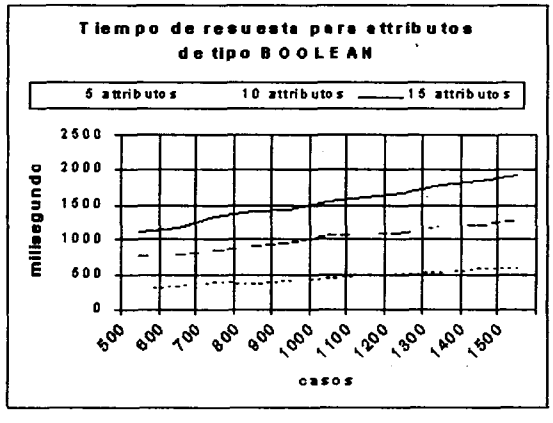


Figura 4.3: Tiempos de respuesta para atributos de tipo lógico (`true`, `false`). Estos tipos lógicos son representados dentro de la máquina virtual de java como objetos `java.lang.Boolean`.

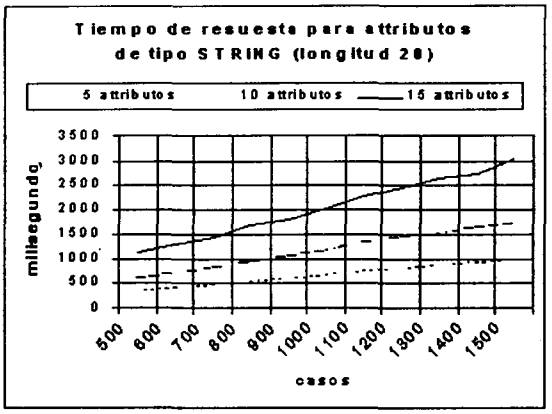


Figura 4.4: Tiempos de respuesta para atributos de tipo cadena (arreglos de bytes o caracteres). Estos tipos booleanos son representados dentro de la máquina virtual de java como objetos `java.lang.String`.

del índice de similitud. Por último, para la vista, se probarán cada una de las especificaciones descritas en la sección 3.7 de manera que se asegure un funcionamiento correcto. El servicio de monitoreo de modificación al repositorio de casos se levantará sobre una librería de casos de prueba cuya información no tiene relevancia en la evaluación del servicio.

4.2.1. Caso de estudio del servicio de mapeo índices-casos: Sistema de análisis de secuencia sobre DNA y códigos genéticos

Actualmente existen grandes bases de datos con secuencias de DNA de organismos como virus, bacterias, plantas, animales, entre otros, así como también secuencias de proteínas naturales o sintéticas. Estas bases de datos están disponibles a través de internet para laboratorios especializados en biogenética o para cualquier persona interesada en el tema. Este caso de uso del sistema JITIK (en colaboración del agente RBC) tiene como meta el usar un esquema de sistemas multi-agente (*sistema JITIK*) para una adecuada distribución y flujo de información (conocimiento) de estas bases de datos, y de esta manera, completar algunas de las tareas que son llevadas a cabo en la mayoría de los laboratorios genéticos como son: buscar secuencias de DNA sobre varios organismos (casos) observando patrones que sugieran estructuras regulatorias dentro de la hélice de DNA (secuencia activa) y calcular concentraciones de diversos aminoácidos en secuencias de prueba (algunas veces aleatorias).

Simulación y descripción de prueba

El *agente RBC* y el *agente de sitio* se colocan en una computadora A en un contenedor principal. La librería de casos (información genética) será local al contenedor principal. El *agente personal* 1 se encuentra en un contenedor secundario 1 en una computadora B remota al contenedor principal. El *agente personal* 2 se encuentra en un contenedor secundario 2 en la misma computadora remota B al contenedor principal. Esta prueba se realizará sobre un sistema operativo Windows 98 (en las dos computadoras) y ACCESS como librería de casos. La base de datos de ACCESS contiene 42364 Kb (aproximadamente 40 MB) de información genética. El *agente personal* `client1` solicita un mapeo de índices a casos al *agente de sitio*, en donde el índice de similitud informa que se requieren todas las secuencias de DNA que se parezcan (subsecuencia común más larga, operador: $\sim$) a una secuencia objetivo (fue tomado como objetivo la secuencia de un virus) y los índices de partición sugieren una búsqueda en las bases de casos: Virus, Bacteria, Drosophila, Fungi, Invertebrate, Plant, Plasmid, Rodent, y Vertebrate. El *agente personal* `client2` solicita un mapeo de índices a casos al *agente de sitio*, en donde el índice de similitud informa que se requiere un análisis de concentración[1] (operador: %) de una secuencia objetivo (fue tomado como objetivo la secuencia de un virus) y el índice de partición sugieren una búsqueda dentro de la base de casos: DNADictionarySecOne.

Resultados

Dentro de la respuesta obtenida por el *agente personal* `client1` se puede observar claramente que existe más similitud en los casos dentro de la base de casos `Virus` que en los casos dentro de las otras bases de casos. Se obtuvo una similitud de 1.0 con la secuencia de DNA del virus de consulta mientras que las otras secuencias de virus obtuvieron similitudes en el rango de 0.58 a 0.7. Los casos contenidos en las otras bases de casos estuvieron dentro del rango 0.22 a 0.59.

La respuesta obtenida por el *agente personal* `client2` muestra las concentraciones de las distintas secuencias de aminoácidos contenidos en la base de casos `DNADictionarySecOne`.

[1]Todas las secuencias de DNA están formadas por pequeños trozos o pequeñas secuencias de aminoácidos ya establecidas, por ejemplo: `AAA` es la secuencia de la licina, `AAC` es la secuencia de la serina, etc. En este ejemplo el operador % nos dice las concentraciones de estos aminoácidos contenidos en la base de casos `DNADictionarySecOne`.

4.2.2. Caso de estudio del servicio de mapeo ontología-casos: Centro de Sistemas Inteligentes

Como se mencionó en la sección 3.6.1, el controlador debe reconocer tres tipos de instancias ontológicas: *instancias dirigidas por rango*, *instancias dirigidas por dominio*, e *instancias dirigidas por jerarquía*. La idea para esta prueba es verificar que el *agente RBC* consulte las instancias ontológicas en el orden descrito dentro de las especificaciones de la sección 3.6 de acuerdo al tipo de instancia visitada. La verificación de los casos recuperados no es necesaria en esta etapa de pruebas puesto que esa parte ya fue considerada dentro de las pruebas al *sistema RBC* o modelo.

Este caso de estudio describe algunos procesos de conocimiento en que puede ser útil el *sistema JITIK* en conjunto con el *agente RBC* dentro del Centro de Sistemas Inteligentes (CSI) en el ITESM. Algunas de las tareas principales son la localización eficiente de todo tipo de documentos como lo son los manuales, diccionarios, libros, artículos, tesis, entre otros; así como también la asignación de asesores (profesores) a los alumnos y viceversa. Todo nuevo estudiante de postgrado del CSI tiene sus propios intereses de investigación y se ve en la necesidad de escoger un asesor de tesis que tenga intereses afines a los de él. El problema aquí es que el alumno no conoce a los profesores y mucho menos sus intereses. Esto también puede ser observado desde el punto de vista del profesor, puesto que cualquier catedrático puede desear el encontrar algún alumno sin asesor que tenga intereses semejantes a los de él. Es aquí donde nos encontramos con la necesidad de crear un sistema que maneje una ontología sobre intereses personales, alumnos, profesores, artículos, etc. que nos asesore en este tipo de cuestiones. Para la realización de este caso de estudio se optó por el uso de una ontología en formato `DAML+OIL` en conjunto con el *agente de ontologías*. La librería de casos está formada por cuatro bases de casos. Una contiene información sobre las personas que laboran en el CSI (profesores asociados, profesores asistentes, profesores titulares, profesores adscritos, personal administrativo, asistentes, etc.). Otra base de casos está formada por estudiantes de postgrado de la maestría en ciencias con especialidad en Sistemas Inteligentes. La tercera base de casos contiene tesis anteriores de estudiantes egresados. La cuarta base de casos está formada por artículos escritos por alumnos y profesores del CSI. Todas las bases de casos tienen en común un atributo llamado `intereses`. Y los valores que puede tomar este atributo están determinados dentro de la ontología `cs1_0.daml`. Las figuras 4.5 y 4.6 muestran a grandes rasgos la taxonomía de la colección de categorías declaradas dentro de la ontología `cs1_0.daml`.

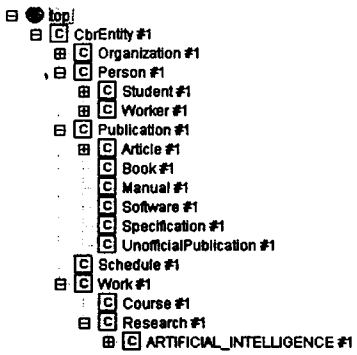


Figura 4.5: Colección de categorías declaradas en la ontología usada para el caso de estudio del Centro de Sistemas Inteligentes.

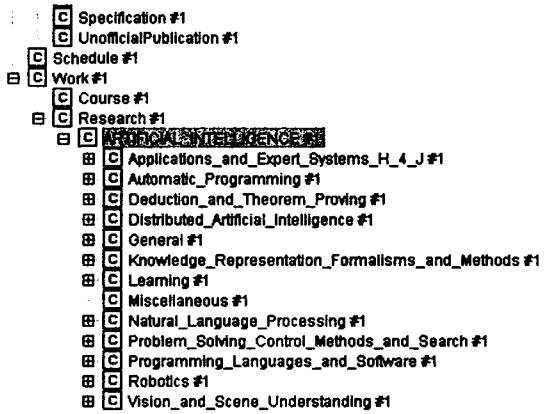


Figura 4.6: Taxonomía de los intereses del Centro de Sistemas Inteligentes. Esta taxonomía es la usada por la ACM (Association for Computing Machinery) para la clasificación del área "Inteligencia Artificial" (ver apéndice C).

Simulación y descripción de prueba

El *agente RBC* y el *agente de sitio* se colocan en una computadora A en un contenedor principal. La librería de casos (profesores, alumnos, artículos y tesis) será local al contenedor principal. El agente personal 1 se encuentra en un contenedor secundario 1 en una computadora B remota al contenedor principal. El agente personal 2 se encuentra en un contenedor secundario 2 en la misma computadora B remota al contenedor principal.

El *agente personal* 1 envía una petición de mapeo de la instancia ontológica `GraduateStudent_Omar_Montano` y el orden de las consultas a las instancias de la respuesta esperada es mostrada en el diagrama de la figura 4.7. Esta consulta tiene como finalidad el probar las instancias de tipo: *rango*, y *jerárquico* en su modalidad ascendente. Por otra parte, el *agente personal* 2 envía una petición de mapeo de la instancia ontológica `FullProfessor_Jose_L_Aguirre` y el orden de las consultas a las instancias de la respuesta esperada es mostrada en el diagrama de la figura 4.8. Esta consulta tiene como finalidad el probar las instancias de tipo: *rango*, *dominio*, y *jerárquico* en su modalidad descendente.

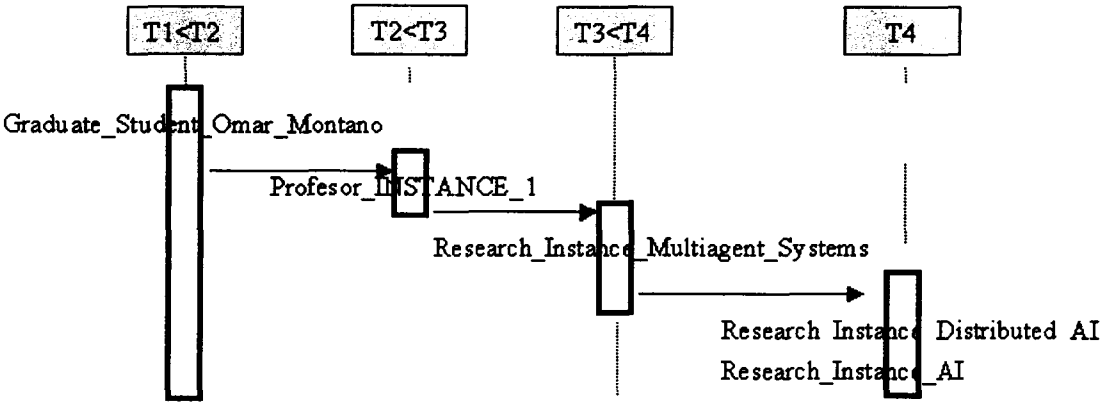


Figura 4.7: Diagrama de secuencia para la consulta de la instancia `GraduateStudent_Omar_Montano`.

Recorrido ontológico para la instancia `Graduate_Student_Omar_Montano`

La figura 4.9 muestra las instancias consultadas por el *agente RBC*. La secuencia fue la siguiente: `Graduate_Student_Omar_Montano`, `Professor_INSTANCE_1`, `Research_Instance_Multiagent_Systems`, `Research_Instance_Distributed_AI`, y `Research_Instance_AI`.

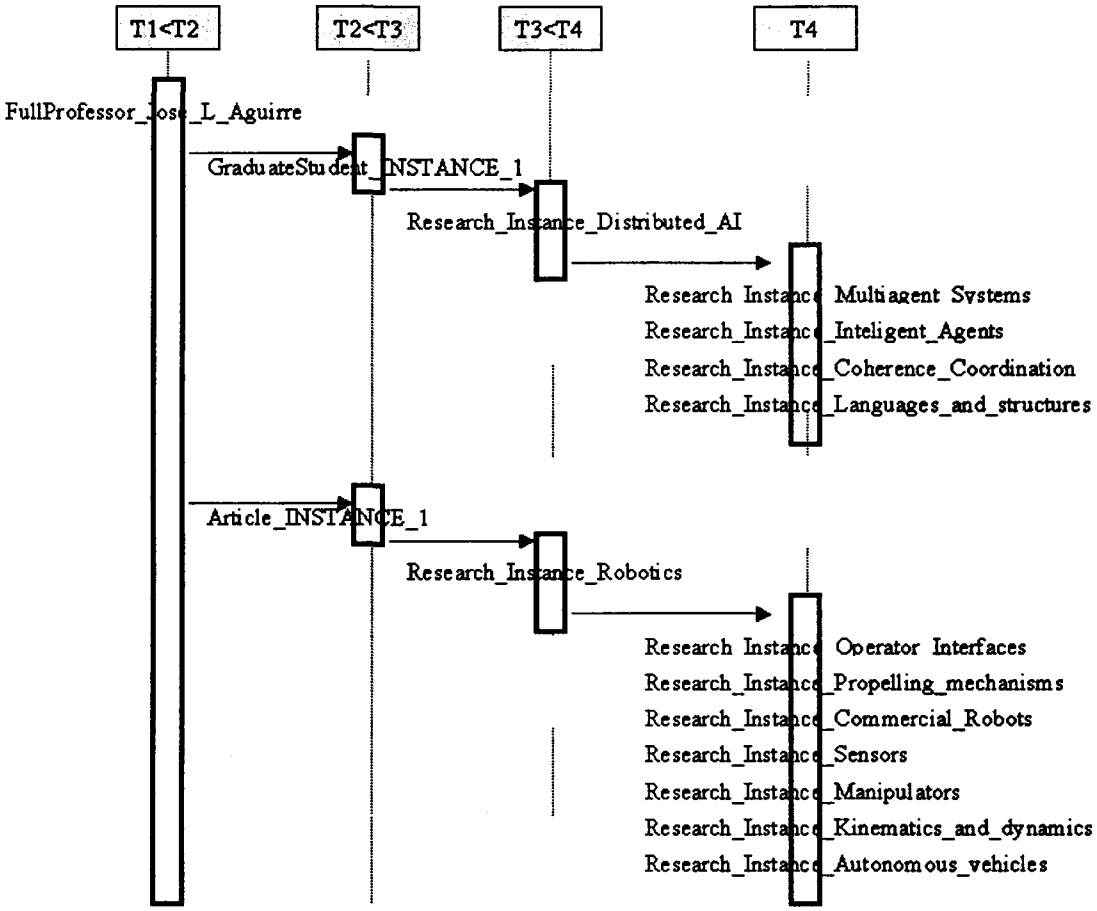


Figura 4.8: Diagrama de secuencia para la consulta de la instancia `FullProfessor_Jose_L_Aguirre`.

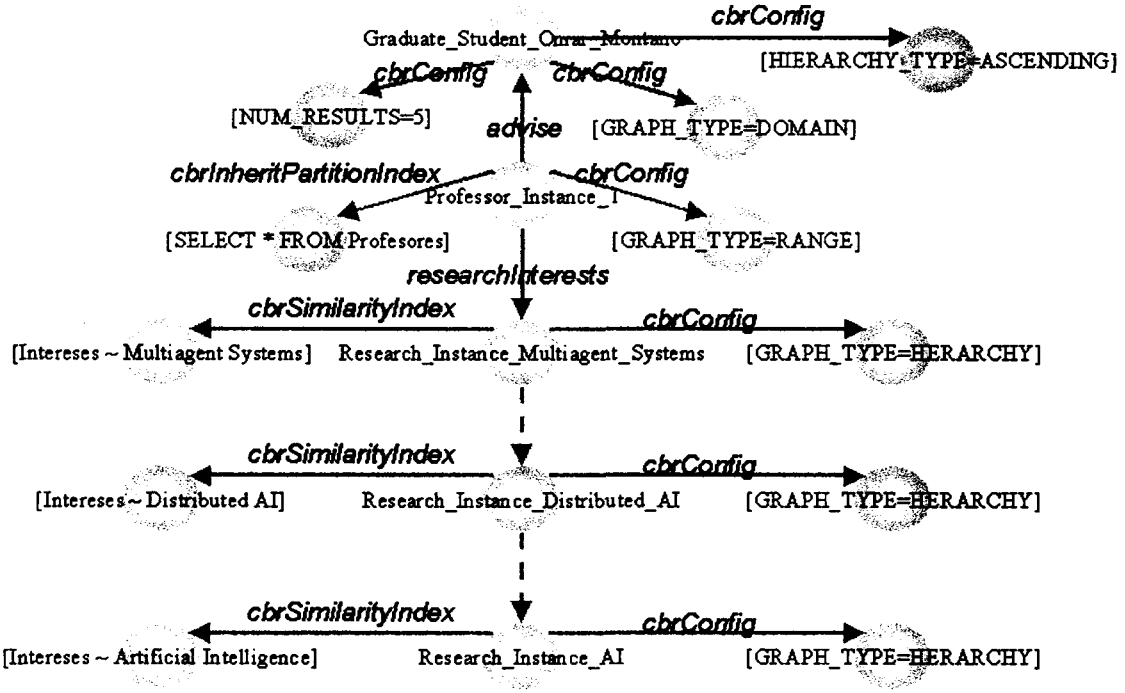


Figura 4.9: Instancias consultadas por el *agente RBC* en la petición del *agente personal* `Graduate_Student_Omar_Montano_Rivas`.

Recorrido ontológico para la instancia FullProfessor_Jose_L_Aguirre

La figura 4.10 muestra las instancias consultadas por el *agente RBC*. La secuencia fue la siguiente: FullProfessor_Jose_L_Aguirre, GraduateStudent_Instance_1, Article_Instance_1, Research_Instance_Distributed_AI, Research_Instance_Robotics, Research_Instance_Intelligent_Agents, Research_Instance_Languages_and_structures, Research_Instance_Multiagent_Systems, Research_Instance__Coherence_Coordination, Research_Instance__Autonomous_Vehicles, Research_Instance__Sensors, Research_Instance__Manipulators, Research_Instance__Propelling_mechanisms, Research_Instance__Operator_interfaces, Research_Instance__Workcell_organization_and_planning, Research_Instance__Commercial_Robots, y Research_Instance__Kinematics_and_dynamics.

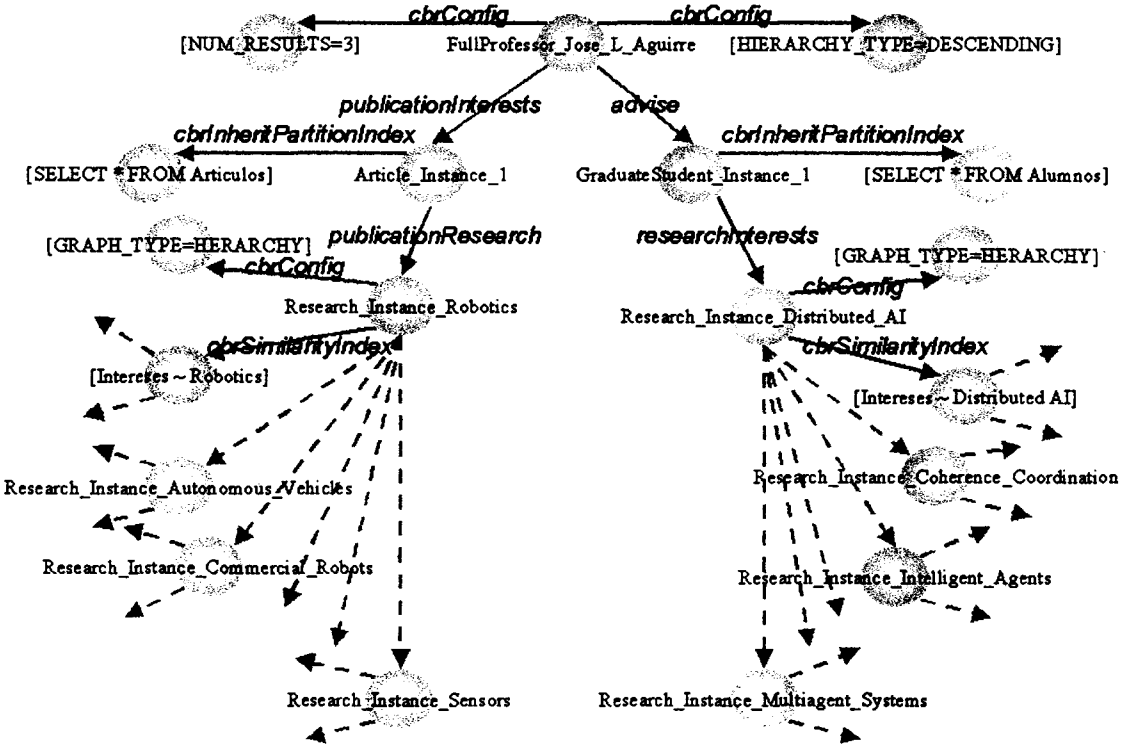


Figura 4.10: Instancias consultadas por el *agente RBC* en la petición del *agente personal* Full_Professor_Jose_L_Aguirre.

Especificación	Observaciones
Agregar base de casos	Es creada una nueva tabla dentro de la base de casos.
Borrar base de casos	Es borrada una nueva tabla dentro de la base de casos. Cuando la tabla está siendo usada por otra aplicación entonces no puede ser eliminada.
Agregar atributo	Es agregado un nuevo atributo dentro de la tabla.
Borrar atributo	Es borrado un atributo dentro de la tabla. Cuando la tabla está siendo usada por otra aplicación entonces no puede ser eliminado ningún atributo de esta tabla.
Agregar caso	Es agregada una fila (caso) dentro de la tabla.
Borrar caso	Es eliminada una fila (caso) de la tabla. Cuando la tabla está siendo usada por otra aplicación entonces no puede ser eliminado ninguna fila de esta tabla.
Modificar caso	Es modificado el valor de una columna (atributo) en una fila (caso) de la tabla. Si el controlador de la base de datos determina que la columna correspondiente al atributo a ser modificado no puede ser buscado entonces, no se modifica el valor del atributo. Esto sucede cuando la función `ResultSet.isSearchable(int col)` (la variable `col` es el número de la columna a ser evaluada) regresa un valor de `false`.

Cuadro 4.1: Resultados y observaciones para las modificaciones realizadas por la vista.

4.2.3. Pruebas a la vista

En la tabla 4.1 se muestran los resultados obtenidos al realizar cada una de las especificaciones descritas en la sección 3.7. Todas las modificaciones fueron notificadas al *agente de sitio* mediante un mensaje ACL de tipo `INFORM` cuyo contenido presentaba la instrucción SQL92 utilizada. Por ejemplo, el contenido del mensaje ACL enviado por el *agente RBC* al agente de sitio cuando se agregaba una base de casos, siempre contenía una instrucción SQL92 con la siguiente estructura: `CREATE TABLE tableName (attributes...)`; en donde `tableName` es el nombre de la tabla o relación creada y `(attributes...)` es la definición de todos los atributos contenidos por esta relación.

4.3. Análisis de resultados

Los resultados obtenidos en las pruebas al modelo, controlador y vista fueron satisfactorios de acuerdo a las especificaciones descritas en las secciones 3.5, 3.6, y 3.7. Estos resultados garantizan un funcionamiento adecuado del *agente RBC*.

Se probaron todos los servicios con todas las posibles combinaciones dentro del esquema de especificaciones, e incluso se realizaron consultas sobre atributos no existentes y el *agente RBC* simplemente no los toma en cuenta. Es decir, se probó cada posible acción que el *agente RBC* pueda tomar y cada posible consulta que se pueda realizar. Y por último, el *agente RBC* es perfectamente escalable puesto que puede trabajar tanto con librerías de casos de algunos casos como con librerías de casos con miles de casos (ver figuras 4.1 a 4.4) requiriendo de los mismos recursos de la máquina virtual de java.

Todas las pruebas realizadas tanto al *Sistema RBC* o modelo (paquete `xcbr`) como al controlador y la vista (paquete `xcbr.agent`) fueron ejecutadas en una computadora IBM con un procesador pentium III a 650 Mhz, memoria RAM de 128 MB (sincronizada a 100 Mhz) y velocidad de reloj de la tarjeta madre de 100 Mhz. Todas las pruebas se realizaron tanto en el sistema operativo linux (Open Source, distribución Mandrake 9.0) como en Windows 98 (a excepción de las pruebas a la vista que se realizaron en Windows 98) y se usó como repositorio de casos a mySQL versión 3.23.53 y a ACCESS.

4.4. Trabajos relacionados

Como se pudo observar en la sección 2.7 existen un gran número de sistemas RBC pero pocos son de propósito general. El sistema `SQUAD` es un ejemplo de estos sistemas RBC de propósito general que tiene la ventaja de manejar casos almacenados en una base de datos convencional. Una desventaja de esta herramienta en relación al sistema propuesto es que solo usa atributos de tipo numérico sin considerar que es necesario el definir toda una jerarquía abstracta para la resolución de las consultas. Otra gran desventaja es que el mecanismo de resolución es combinatorio en el número de instancias de atributos, y a consecuencia de eso, el sistema `SQUAD` no puede ser usado para manejar casos con atributos continuos e incluso en casos con atributos discretos que puedan tomar muchos valores.

De todos los sistemas RBC, el único que tuvo el enfoque de los sistemas multiagentes fue el del laboratorio de IA de la universidad de Chicago. Los agentes `FIND-ME`, `BUTLERS`, y `CORRESPONDENTS` están diseñados para ayudar a un usuario a "navegar" a través de un espacio de información. Dado que JITIK es un sistema que se enfoca hacia la "difusión" más que a la "navegación", el *agente RBC* propuesto no tiene características equiparables con los tres agentes anteriores. La única característica similar con el *agente RBC* es que se utiliza el paradigma del razonamiento basado en casos. Sin embargo, es necesario implementar un agente `FIND-ME`, `BUTLER`, o `CORRESPONDENT` para cada aplicación diferente en contraste con el *agente RBC*, que puede ser usado para una gran variedad de aplicaciones distintas.

Capítulo 5

Conclusiones

Para finalizar, en este capítulo se presentan las conclusiones relacionadas con este trabajo. Primero se describen las características más relevantes del *agente RBC*. Después se mencionan las aportaciones que se desprenden de esta tesis así como también algunas de las posibles extensiones a considerar como trabajo futuro. Y por último, se hacen algunos comentarios finales.

5.1. Características del Agente RBC

El *agente RBC* tiene un comportamiento dirigido a metas descritas dentro de las especificaciones de la sección 3.3. Todas estas especificaciones están bien definidas, integradas, e implementadas dentro de las acciones del *agente RBC* en JITIK. Como fue observado en el capítulo 4, el *agente RBC* responde a las peticiones o consultas de manera rápida y apropiada. La eficiencia en los tiempos de respuesta del *sistema RBC* se comporta de manera aproximadamente lineal conforme se incrementa la cardinalidad de la base de casos consultada, sin importar el tipo de atributo o los valores que éste pueda tener, cosa que no sucede con la consola comercial CARET. Además, la comunicación con los agentes de *sitio* y *ontologías* está perfectamente adaptada a las necesidades del *agente RBC*. Todo esto muestra que los requisitos de la reactividad, proactividad, y sociabilidad que tiene todo agente inteligente [24] son características bien establecidas del *agente RBC*.

Para el *agente RBC*, se ha hecho notar que la definición de los dos tipos de índices mencionados antes, los PI y los SI, facilita la busqueda de casos. Los PI permiten construir una consulta general, y los SI permiten refinarla.

El *sistema RBC* construido, contiene un conjunto de operadores ($\sim$, $!\sim$, %, !%) sobre los cuales guiar el proceso del razonamiento basado en casos bajo el esquema de cuatro tipos de atributos (numéricos, fecha, lógicos, y cadenas) utilizando una serie de métricas y algoritmos. Cada una de estas métricas son las que comúnmente se usan en las aplicaciones de RBC.

El uso de una base de datos tradicional como fuente de casos es uno de los fac-

tores más importantes dentro de la implementación del *sistema RBC* puesto que fueron resueltos problemas como seguridad, velocidad de selección y almacenamiento, y estandarización de los datos. Además, el uso de un RDBMS ofrece otras ventajas, como la independencia de datos, e integridad en los datos. Esas características son proporcionadas por toda RDBMS por más básica que sea su funcionalidad.

La motivación detrás del uso de un RDBMS está contenida en dos características principales. Primero, toda organización distribuida maneja su información y conocimiento dentro de bases de datos relacionales, y en la mayoría de las veces, esta información y conocimiento necesita permanecer dentro de un control muy riguroso de seguridad, privacidad e integridad. Segundo, un número significativo de bases de datos, comerciales y libres, están disponibles para el manejo de información en las organizaciones. La aplicación de técnicas de RBC provee a los sistemas de información de las organizaciones nuevas características interesantes y útiles, puesto que los RDBMS actuales solo proveen capacidades de consulta primitivas (como lo son las consultas exactas).

En resumen, el *agente RBC* diseñado e implementado en esta tesis, explota la idea de agentes inteligentes así como también el paradigma del razonamiento basado en casos de manera natural y genérica haciendo uso de tecnologías que aseguran la privacidad, seguridad, e integridad de la información y conocimiento almacenado por una organización.

Como producto se obtiene un agente configurable y escalable, adecuado a la arquitectura que el sistema JITIK necesita.

5.2. Aportaciones

Sin duda alguna, el servicio más importante que el *agente RBC* aporta al sistema *JITIK* es el de mapeo de instancia ontológica a casos (controlador, ver sección 3.2). Este servicio consta de un mecanismo de extracción de conocimiento de una ontología para la elaboración de un esquema de búsqueda de casos dentro de repositorios (ver sección 3.6). Con este servicio, el *agente RBC* hace posible la conexión entre el perfil de intereses de los usuarios con los casos almacenados en la librería de casos.

Consideramos que el manejo de una ontología para encontrar casos relevantes puede tener muchas ventajas que además de utilizar relaciones de especialización se introducen los conceptos de instancias dirigidas por rango y por dominio. Primeramente, es posible especificar de manera declarativa el conocimiento que el *agente RBC* necesita para construir las consultas; segundo, nos podemos beneficiar de la estructura jerárquica de la ontología para tomar ventaja de su herencia inherente, por tanto se pueden describir índices relevantes en muchos niveles.

Fue definido, implementado y probado con muchos casos un algoritmo original para el recorrido a través de la ontología. Se revisó bibliografía e información en la

internet y no se encontró ningún mecanismo de extracción de índices a partir de una ontología para después ser usados en el proceso de selección dentro del paradigma del RBC.

Las integraciones mencionadas en el documento representan además los primeros servicios de un agente de *JITIK* que claramente explotan fuentes de conocimiento usando tecnologías de IA. Esto lo consideramos de suma importancia si uno realmente quiere hablar de flujo de conocimiento en un sistema de manejo de conocimiento (Knowledge Management).

Otra aportación al *sistema JITIK* fue la construcción del *sistema RBC* (modelo, ver sección 3.2). Cabe señalar que la implementación del *sistema RBC* se debió a tres razones principales:

1. No existen herramientas para el RBC creadas en java que sean gratuitas.
2. La única herramienta gratuita llamada `Selection Engine` [1] es extremadamente ineficiente[1] y contiene muchos errores de implementación[2], sin mencionar que no existe una documentación para el uso de la herramienta.
3. Velocidad de selección y almacenamiento, seguridad, integridad y estandarización de los datos, entre otras.

5.3. Trabajo futuro

Existen varios trabajos que se pueden desprender a partir de este trabajo realizado. Estos posibles trabajos se pueden catalogar en dos grandes tipos: por un lado, los trabajos de desarrollo; y por el otro, los de investigación.

Dentro de los trabajos de desarrollo, se puede considerar la programación de los módulos de modificación y almacenamiento (`CbrModifier` y `CbrStorer`) para las tareas de adaptación, evaluación, reparación y aprendizaje. Estas partes no son muy fáciles de implementar de manera genérica ya que son muy dependientes del dominio de aplicación, aunque se podrían programar para una aplicación en particular como por ejemplo planeación. Se pueden encontrar buenas referencias de esto en [13, 17, 15, 26]. También se puede trabajar en la definición de operadores, agregando un mayor número de estos o definiendo nuevas métricas.

La interfaz gráfica del agente fue diseñada utilizando el modelo de múltiples documentos, perteneciendo cada documento a tres distintas capas: la capa de documento (manejo de la librería de casos), la capa de herramienta (manejo de la configuración),

[1] Antes de procesar los casos, estos son cargados a la memoria principal de la máquina virtual de java. Los repositorios de casos son archivos de texto.

[2] En algunas partes del programa se discrimina la sintaxis en nombres de los atributos y en otras partes no.

y la capa de ayuda (manejo de la ayuda). Esto podría sugerir la creación de nuevos tipos de documentos, herramientas o ventanas de ayuda para dar una mayor funcionalidad al *agente RBC*. Por ejemplo, se podrían desarrollar herramientas, documentos, y/o ventanas de ayuda para hacer la recolección de los casos más fácil, para hacer la modificación de la librería de casos más amigable, para ayudar a la creación de la solución, para ayudar con la evaluación, para ayudar con la adaptación, para ayudar con la reparación, entre otras.

Ahora bien, por el lado de los trabajos de investigación, se puede considerar como uno de los más importantes el realizar pruebas en contextos organizacionales reales. De esta manera se podrán hacer mediciones para ver que tanto mejora la práctica de la administración del conocimiento y en particular el flujo de conocimiento en una organización con sistemas como *JITIK*.

El elaborar una metodología para la implantación del *sistema JITIK* junto con el *agente RBC* podría ser otro trabajo futuro. Dentro de las tareas a realizar serían: diseño de una arquitectura adecuada para la petición de alguno de los tres servicios del *agente RBC*, la creación del repositorio o librería de casos así como la estructura de las bases de casos, elaboración de una ontología adecuada al dominio de conocimiento (ingeniería del conocimiento), y configuración y seguridad de datos.

Por otro lado, el sistema de "recorrido" ontológico del controlador (ver sección 3.6) es altamente configurable, lo cual proporciona una gran cantidad de opciones al administrador de JITIK acerca de la forma en que la extracción de conocimiento de la ontología debe llevarse a cabo. Sin embargo es posible mejorar este mecanismo diseñando un esquema más elaborado en la manera en que se pudieran generar los índices a partir de la ontología. Por ejemplo, se podría implementar un mecanismo similar al de una máquina de estados en el cual sea posible decidir qué nodos consultar e incluso realizar ciclos sobre un conjunto de nodos. En la actual implementación no es posible consultar más de una vez un mismo nodo del grafo ontológico y la dirección de la búsqueda de índices está controlada por los tipos de nodos o instancias (rango, dominio, y jerárquico) en la ontología.

Otra posible oportunidad de investigación es referente al mantenimiento de la librería de casos. ¿Quién puede actualizar la librería de casos?, ¿cuándo se debería realizar esta operación?, y ¿cómo puede ser administrada en organizaciones distribuidas? Estas son algunas preguntas que nos podríamos hacer cuando nos encontramos en un sistema donde existen múltiples usuarios en diferentes lugares (organizaciones distribuidas). Para que una librería de casos se desempeñe tan bien como la memoria de una organización e incluso llegue a superarla con el transcurso del tiempo en la medida que proporcione un mejor asesoramiento, es necesario que incremente sus experiencias [13]. Pero en un sistema interactivo y dinámico, es difícil realizar esta labor. Se podría estandarizar un protocolo que decida qué casos deberían ser agregados a la librería de casos, cuales deberían ser ignorados y cuales podrían ser compuestos de manera que

sean útiles para los objetivos de la organización.

También se podría investigar la manera de realizar el mapeo de "modificaciones" a la librería de casos a *agentes personales* dentro de JITIK. Actualmente el *agente RBC* notifica al *agente de sitio* todos los cambios realizados a la librería de casos y el *agente de sitio* notifica a todos los *agentes personales* registrados en JITIK. Por otro lado, sería más conveniente que el *agente de sitio* notificara solamente a los *agentes personales* interesados en determinado tipo de modificación a la librería de casos. Obviamente esta no es una tarea trivial en el sentido de que se pueden realizar una infinidad de modificaciones a una librería de casos.

Un último trabajo futuro de investigación se centra en uno de los pasos básicos del RBC: la adaptación de los casos. De hecho, la adaptación es el reto más importante de las actuales investigaciones del RBC ya que la habilidad para resolver problemas nuevos depende de la habilidad de adaptar casos recuperados (adecuarse a nuevas situaciones) y de la habilidad de reparar soluciones que fallaron. La dificultad se origina en el cómo realizar la adaptación. Existen muchas maneras de adaptar un caso, sin embargo, una adaptación efectiva depende de tener conocimiento de las posibles adaptaciones y la manera de seleccionar aquellas que sean apropiadas y efectivas en una situación en particular.

5.4. Comentarios finales

Antes de dar por terminado este trabajo, cabe comentar que la relativa complejidad en la construcción de este agente fue primordialmente por el hecho en que se desarrolló un sistema de RBC lo más genérico posible ya que la construcción de herramientas genéricas para la aplicación del paradigma del razonamiento basado en casos en muchos dominios de conocimiento, no es una tarea fácil de realizar. Además, la implementación de clases abstractas para las tareas de modificación (`CbrModifier`) y aprendizaje (`CbrStorer`) pueden ayudar a que en un futuro se agreguen al *sistema RBC* y de esta forma poder realizar operaciones de adaptación, evaluación, reparación y aprendizaje. Cabe aclarar que los distintos operadores implementados fueron los usados normalmente por las distintas aplicaciones encontradas en la literatura y que el significado que estos puedan tener es dependiente de la aplicación en particular.

Uno de los resultados alcanzados en este trabajo fue el análisis de la literatura en el área del razonamiento basado en casos: se obtuvieron los conceptos y las características básicas que distinguen a los sistemas que utilizan este nuevo paradigma, tales como indexación, selección, adaptación y aprendizaje; además se describieron varios trabajos realizados en esta área. También se estudiaron las tecnologías de los agentes inteligentes y los sistemas multiagentes dentro del proceso de administración del conocimiento.

De toda esta literatura se puede concluir que la tendencia de la inteligencia ar-

tificial es la de formar sistemas computacionales más inteligentes y robustos que nos ayuden en las tareas más críticas, así como también a comprender cómo es que aprendemos, recordamos y razonamos.

Apéndice A

Implementación

Antes de empezar a describir la implementación del *agente RBC* y del *sistema RBC* es necesario mencionar algunos puntos importantes. Primero, tanto el *agente RBC* como el *sistema RBC* serán implementados con el lenguaje de programación Java [2, 18, 10, 29]. El *agente RBC* estará implementado usando como base a las librerías de JADE (Java Agent Development Framework) que es un software para desarrollo de sistemas multi-agente alineado a los estándares de FIPA en agentes inteligentes (ver sección 2.3). La principal razón para el uso de JADE es que JITIK está implementado usando estas librerías.

La implementación del *agente RBC* va a ser dividida en cuatro partes: la primera parte de la implementación será dedicada al *sistema RBC* o modelo, la segunda parte al controlador, la tercera parte a la vista, y por último, la cuarta parte será destinada a las adaptaciones del *agente de sitio.*

La figura A.1 muestra el diagrama de clases del *agente RBC.* Se puede observar que la clase de java `CbrAgent` usa distintas clases para la tarea del modelo (`CbrEngine`), tarea del controlador (`CbrIndexMappingService` y `CbrOntologyMappingService`), y tarea de la vista (`CbrAgentGui`), así como también algunas otras clases de apoyo.

A.1. Implementación del Sistema RBC

En la figura A.2 podemos observar las principales clases del sistema RBC (modelo) en formato UML (Unified Modeling Language). Se hace evidente que la clase de java `CbrEngine` está formada por cuatro clases principales: `CbrIndexer`, se encarga de la indexación de casos; `CbrRetriever`, se encarga de la recuperación de casos; `CbrModifier`, se encarga de la etapa de modificación; `CbrStorer`, se encarga de la fase de aprendizaje. Los únicos métodos que no fueron implementados son: `adapt()`, `evaluate()` y `repair()` de la clase `CbrModifier`; y `learn()` de la clase `CbrStorer`. También se muestra que la clase `CbrIndexer` construye objetos de tipo `CbrCase` que representan casos indexados dentro de la librería de casos. Estos casos (`CbrCase`) contienen una serie de atributos que son implementados dentro de la clase `CbrTrait`. Los

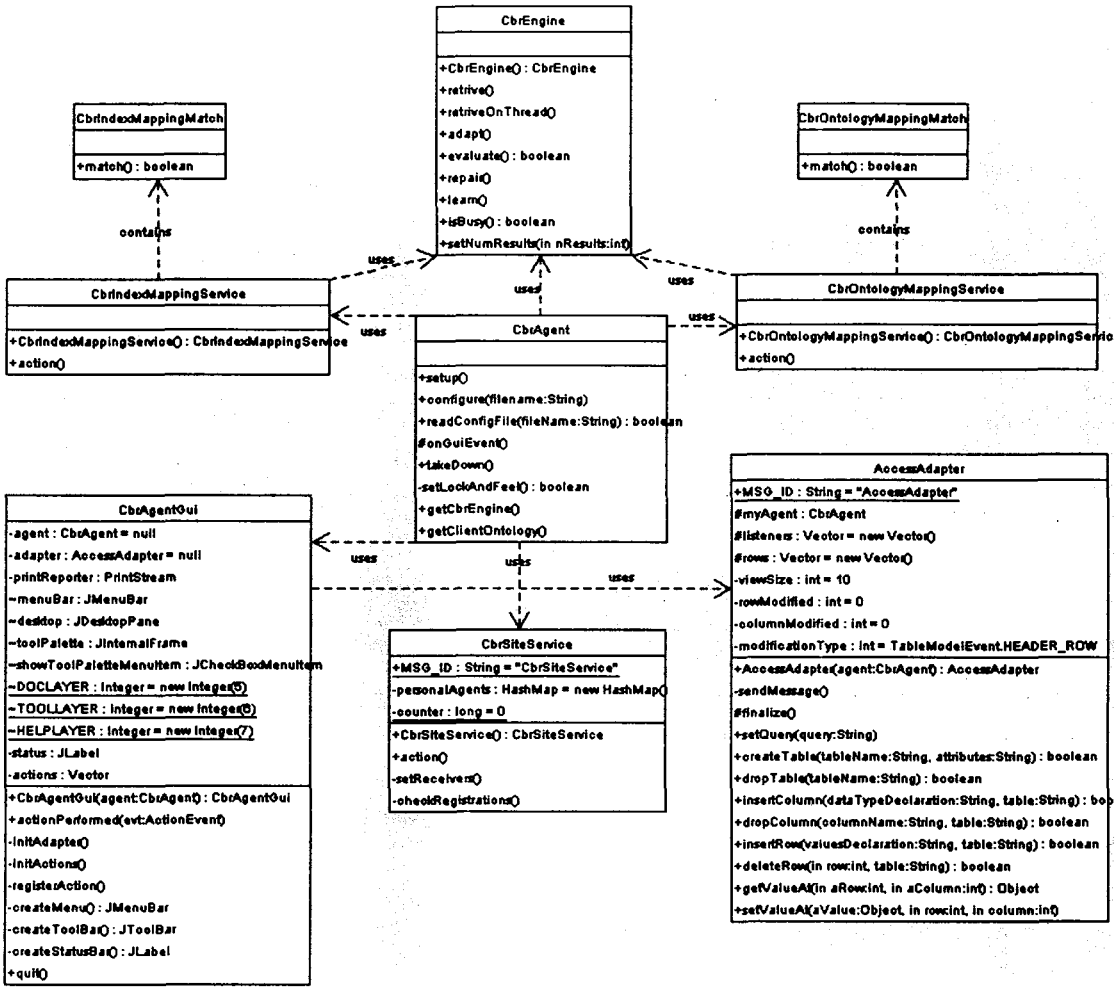


Figura A.1: Diagrama de clases del agente RBC.

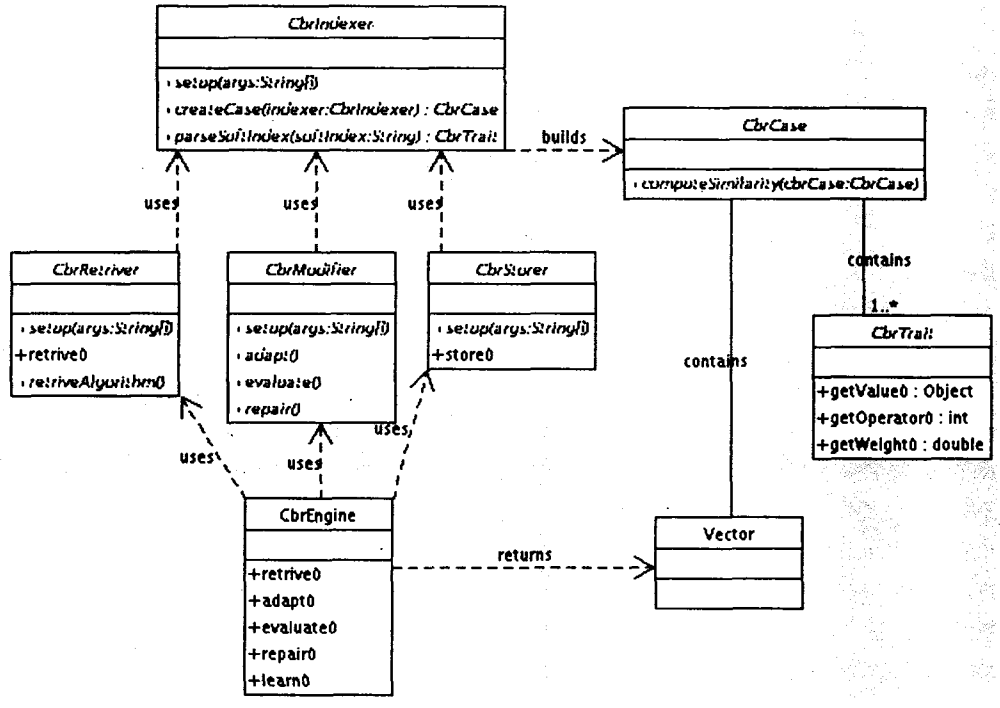


Figura A.2: Diagrama de clase del sistema RBC.

atributos implementados son de tipo numérico, fecha, lógico y cadena. Una vez que la clase `CbrEngine` termina de ejecutar los métodos `retrieve()`, `adapt()`, `evaluate()`, `repair()`, y `learn()` regresa un vector con objetos de tipo `CbrCase`.

1. **CbrEngine.** Esta clase se encarga de controlar al modulo de indexación (clase `CbrIndexer`), recuperación (clase `CbrRetriever`), modificación (clase `CbrModifier`) y almacenamiento (clase `CbrStorer`). Esta clase representa la implementación del rol Modelo. Los métodos principales de esta clase son: `retrieve()`, `adapt()`, `evaluate()`, `repair()` y `learn()`. El método `retrieve()` le ordena al módulo de recuperación que acceda a la librería de casos y recupere todos lo casos que sean similares a una descripción dada, utilizando alguna métrica como medida de similitud. El algoritmo de acceso es programado dentro de la clase `CbrRetriever` y las medidas de similitud son programadas dentro de las clases `CbrCase` y `CbrTrait`. El método `adapt()` hace que el módulo de modificación adapte el o los casos encontrados por el modulo de recuperación utilizando técnicas de adaptación programadas dentro de la clase `CbrModifier`. El método `evaluate()` hace que el módulo de modificación verifique que el o los casos adaptados sean correctos según las técnicas de evaluación programadas dentro de la clase `CbrModifier`. El método `repair()` le ordena al modulo de modificación que repare el o los casos calificados como incorrectos por el método anterior utilizando técnicas de reparación programadas dentro de la clase `CbrModifier`. El método `learn()` le pide al módulo de almacenamiento que indexe las soluciones o casos encontradas para su almacenamiento posterior dentro de la librería de casos. `CbrEngine` también se encarga del manejo de memoria y el control de ejecución cuando los métodos anteriores son llamados dentro de otro proceso o hilo ya que los métodos descritos anteriormente se pueden ejecutar en una tarea distinta a la principal utilizando los métodos `retrieveOnThread()`, `adaptOnThread()`, `evaluateOnThread()`, `repairOnThread()` y `learnOnThread()` respectivamente. Es estrictamente recomendable utilizar esta clase como mediador o controlador de los módulos de indexación, recuperación, modificación y almacenamiento; es decir, no se deberán usar las clases `CbrIndex`, `CbrRetriever`, `CbrModifier` y `CbrStorer` fuera de la clase `CbrEngine`.

2. **CbrIndexer.** Como se mencionó en la sección 3.4, el propósito de la indexación es el de elaborar la descripción del caso, usando técnicas de indexación, para calcular o asignar índices de forma que el caso pueda ser recuperado o almacenado. Para lograr esto, esta clase utiliza dos índices distintos: *índices de partición* e *índices de similitud* (*PI* y *SI* por su nombre en inglés). Esta clase también se encarga del sistema de entrada y salida de la librería de casos. Las clases `CbrRetriever` y `CbrStorer` requieren forzosamente de los servicios de salida y

entrada a la librería de casos respectivamente que la clase `CbrIndexer` proporciona. Dado que el origen de los datos de la librería de casos puede provenir de lugares muy distintos como archivos de texto, flujos dentro de un socket, bases de datos de Oracle o mySQL, entre otros, los distintos métodos de apertura de datos, cierre de datos, entrada de datos y salida de datos deben ser implementados dentro de esta clase. Los métodos más importantes de esta clase son: `createCase()` y `parseSimilarityIndex()`. `createCase()` es un método abstracto que debe ser implementado por toda clase que herede de la clase `CbrIndexer` y tiene como propósito el crear un caso (`CbrCase`) que represente el registro actual dentro de la fuente de datos. Por ejemplo, si la fuente de datos es una base de datos convencional entonces el registro actual podría ser el renglón al que apunta el método de java `ResultSet.currentRow()` [18] dentro de la consulta. Si la fuente de datos es un archivo de texto entonces el registro actual podría ser el recuperado por el método de java `BufferedReader.readLine()`. El método `parseSimilarityIndex()` funciona como un "parseador" de los *SI*. Como fue descrito anteriormente los *SI* son cadenas de caracteres de la forma `attribute OP value` en donde `attribute` es el nombre del atributo o campo del caso, `OP` es el operador y `value` es el valor del atributo. Este método debe ser capaz de leer y entender los diferentes operadores que pudieran estar definidos, y como resultado regresar una clase descriptiva de tipo `CbrTrait`.

3. **CbrRetriever.** Se puede pensar de una librería de casos como un tipo especial de base de datos. Como en las bases de datos, esta librería almacena un gran número de registros y uno tiene que ser capaz de recuperar registros apropiados en una cantidad razonable de tiempo aún cuando los casos almacenados sean miles o cientos de miles. Esta es la tarea de esta clase abstracta definida dentro del conjunto de clases del *sistema RBC*. Es aquí donde se acceden casos similares usando algoritmos de acceso y las medidas de similitud para su posterior modificación. Los métodos principales de esta clase son: `retrieve()` y `retrieveAlgoritmh()`. El método `retrieve()` es el responsable de la recuperación de todos los casos que sean potencialmente útiles para los fines del *sistema RBC* (`CbrEngine`). Este método manda llamar internamente al método abstracto `retrieveAlgoritmh()` en el cual se implementan los algoritmos de acceso y recuperación.

4. **CbrModifier.** En el módulo de *modificación* es creada la solución, evaluada y si es necesario explicada y reparada usando técnicas de adaptación, técnicas de evaluación y técnicas de reparación. Esta clase se encarga de encapsular todas estas tareas que el *sistema RBC* debe proporcionar. Los métodos más importantes de esta clase son: `adapt()`, `evaluate()` y `repair()`. Todos estos métodos son llamados directamente por los métodos del mismo nombre dentro de la clase `CbrEngine`. *Estos métodos no fueron implementados.*

5. **CbrStorer.** Una vez creada y evaluada la solución, se procede a entregar la solución. Además, se puede crear un nuevo caso. Este caso se forma no solo de la solución encontrada, sino del caso o problema original, la solución encontrada, y las justificaciones y explicaciones (si las hay). Al nuevo caso se le asignan índices y se almacena en la base de casos [16, 27]. Estas son las tareas que debe elaborar esta clase dentro del *sistema RBC*. El método más importante de esta clase es `learn()` que es llamado por el método del mismo nombre dentro de la clase `CbrEngine`. *El método `learn()` no fue implementado.*

6. **CbrCase.** Esta es la clase abstracta que representa cualquier tipo de caso en cualquier librería de casos. Esta es una clase que debe ser heredada e instanciada para su uso posterior. El método principal de esta clase es: `computeSimilarity()`, que tiene como finalidad el comparar que tan similar es un caso con otro (medidas de similitud).

7. **CbrTrait.** Esta clase abstracta representa o describe un atributo de un caso. Esta clase al igual que `CbrCase` debe ser heredada e instanciada para su posterior uso. El método principal de esta clase es `getSimilarity()` que tiene como finalidad el comparar dos atributos homónimos de dos casos distintos.

A.2. Implementación del controlador

El controlador esta formado principalmente por dos clases de java que se encargan de realizar los servicios de mapeo de índices a casos y de ontología a casos. Estas clases son: `CbrIndexMappingService` y `CbrOntologyMappingService`. En la figura A.1 se pueden observar estas clases dentro del diagrama UML de clases del *agente RBC*. A continuación se describen estas dos clases de java, así como también sus principales métodos.

1. **CbrIndexMappingService.** Esta clase de java se encarga de recibir las peticiones del *agente de sitio* que solicitan el mapeo de índices a casos. `CbrIndexMappingService` extiende la funcionalidad de la clase de JADE `CyclicBehaviour` y está activa mientras dure la vida del *agente RBC* y aparte esté configurado adecuadamente el *sistema RBC* (ver figura A.6). Esta clase recibe todos los mensajes ACL de tipo `REQUEST` que sean enviados por la clase de java `CbrIndexMappingRequest`, misma que utiliza el *agente de sitio* para realizar la petición. La figura A.3 muestra cómo un *agente personal* puede solicitar el servicio de mapeo de índices a casos; en este ejemplo el *agente personal* le dice al *agente de sitio* que necesita todas las computadoras cuyo procesador sea de "aproximadamente" 650 Mhz, memoria ram de "aproximadamente" 128 Mb, disco duro de "aproximadamente" 20 Gb, y la marca que no sea IBM de preferencia. Se puede observar que

el agente personal solicita que la importancia (peso) sobre el atributo "maker" sea de 3 (cuando no se especifica el parámetro peso "w=" se considera con un peso de 1.0). El parámetro peso puede ser cualquier número real positivo. Si por ejemplo es de suma importancia que el atributo "maker" sea diferente de `IBM` entonces se puede hacer arbitrariamente grande el parámetro peso o como segunda alternativa podría cambiarse el índice de partición a `SELECT * FROM Computadoras WHERE maker != IBM`. El filtro para los mensajes con las características mencionadas anteriormente es realizado por la clase de java `CbrIndexMappingMatch`. Cuando es recibido este tipo de mensajes por la clase `CbrIndexMappingService`, es agregado un comportamiento llamado `CbrIndexMappingAction` que extiende la funcionalidad de una clase de JADE `OneShotBehaviour`. La clase `CbrIndexMappingAction` realiza la consulta directamente a la clase `CbrEngine` (modelo) y regresa los resultados mediante un mensaje ACL de tipo `INFORM` al *agente de sitio* solicitante.

2. **CbrOntologyMappingService.** Esta clase de java se encarga de recibir las peticiones del *agente de sitio* que solicitan el mapeo de una instancia ontológica a casos. `CbrOntologyMappingService` extiende la funcionalidad de la clase de JADE `CyclicBehaviour` y está activa mientras dure la vida del *agente RBC* y aparte esté configurado adecuadamente el *sistema RBC* al igual que la ontología (ver figura A.6). Esta clase recibe todos los mensajes ACL de tipo `REQUEST` que sean enviados por la clase de java `CbrOntologyMappingRequest`, misma que utiliza el *agente de sitio* para realizar la petición. La figura A.4 muestra un ejemplo de uso de la clase `CbrOntologyMappingRequest`. El filtro para los mensajes con las características mencionadas anteriormente es realizado por la clase de java `CbrOntologyMappingMatch`. Cuando es recibido este tipo de mensajes por la clase `CbrOntologyMappingService`, es agregado un comportamiento llamado `CbrOntologyMappingAction` que extiende la funcionalidad de una clase de JADE `CyclicBehaviour`. La clase `CbrOntologyMappingAction` es un comportamiento cíclico que esta activo mientras se esté realizando el proceso de búsqueda de índices en la ontología y crea tantas instancias de la clase de java `CbrOntologyInstance` como instancias ontológicas consulte. El proceso de búsqueda de índices, los criterios de paro y los mecanismos de extracción de índices son implementados por la clase de java `CbrOntologyInstance` (ver secciones 3.6.1 a la 3.6.3). Una vez realizado el proceso de extracción de índices se envía un mensaje ACL de tipo `INFORM` con los resultados encontrados al *agente de sitio* solicitante.

```
..
..
String PI = "SELECT * FROM Computadoras";
String[] SI = {"procesor ~ 650",\\ en Mhz
               "ram ~ 128",\\ en MB
               "hd ~ 20",\\ en GB
               "maker !~ IBM w=3"};
CbrIndexMappingRequest request =
            new CbrIndexMappingRequest(myAgent, PI, SI); \\
myAgent.addBehaviour(request);
..
..
```

Figura A.3: Ejemplo de código fuente para la petición del servicio de mapeo de índices a casos.

```
..
..
String instanceName = "Jose_L_Aguirre";\\
CbrOntologyMappingRequest request =
            new CbrOntologyMappingRequest(myAgent, instanceName); \\
myAgent.addBehaviour(request);
..
..
```

Figura A.4: Ejemplo de código fuente para la petición del servicio de mapeo de instancia ontología a casos.

A.3. Implementación de la vista

La vista esta implementada por una serie de clases de java manejadas por la clase `CbrAgentGui`. `CbrAgentGui` es una clase que extiende la clase de java `JFrame` y maneja un conjunto de frames internos (`JInternalFrame`) clasificados en tres tipos: documento, herramienta, y ayuda. Los frames internos de tipo documento están destinados a mostrar y modificar información que contenga el *agente RBC*, como por ejemplo, el mostrar y modificar información contenida por la librería de casos. Las clases `ConfigurationDocumentFrame`, `TaskReporterDocumentFrame`, y `CaseLibraryDocumentFrame` son frames internos de este tipo y su función está descrita a continuación. Los frames internos de tipo herramienta están destinados a usar herramientas externas al *agente RBC*, como por ejemplo depuradores de agentes o incluso depuradores de java. Aún no hay clases de java que están dentro de esta capa de frames. Los frames internos de tipo ayuda son usados para desplegar ayuda. La clase de java `CbrAgentHelp` despliega la documentación en formato `html` del código del *agente RBC*.

Ahora que se describió un modelo para la implementación del GUI del *agente RBC* van a ser explicadas las tres clases dentro de la capa de documento, ya que estas tres clases son las más importantes en la funcionalidad del GUI.

1. **ConfigurationDocumentFrame** Esta clase de java muestra el archivo de configuración seleccionado en el momento del arranque del agente (ver figura A.6) en donde es posible la modificación de mismo, creación de un nuevo archivo de configuración o abrir un archivo existente.

2. **TaskReporterDocumentFrame** Esta clase de java muestra en texto todas las acciones realizadas por el *agente RBC*, como por ejemplo, la apertura de una librería de casos, las modificaciones realizadas al repositorio, si el *agente RBC* está en estado activo o suspendido, entre otras.

3. **CaseLibraryDocumentFrame** Esta es la clase de java con mayor importancia dentro de la capa documento. Esta clase realiza consultas `SQL92` a la librería de casos, y muestra la información en una tabla (`JTable`). Todas las operaciones de modificación descritas en la sección 3.7 son realizadas por este frame interno mediante la clase de java `AccessAdapter` que es muy parecida a la clase `CbrIndexer`. El método `AccessAdapter.createTable(String tableName, String attributes)` crea una nueva base de casos llamada `tableName` con los atributos `attributes`, el método `AccessAdapter.dropTable(String tableName)` elimina una base de casos llamada `tableName`, el método `AccessAdapter.insertColumn(String dataTypeDeclaration, String table)` agrega un atributo a la base de casos, el método `AccessAdapter.insertRow(String valuesDeclaration, String table)` agrega un caso a la base de casos. En el

```
..
..
CbrSiteService service =
            new CbrSiteService(myAgent); \\
myAgent.addBehaviour(service);
..
..
```

Figura A.5: Ejemplo de código fuente a insertar dentro del método `Agent.setup()` para que un *agente de sitio* pueda intermediar el servicio que el *agente RBC* brinda.

momento en que se realiza alguna modificación a la librería de casos, la clase `AccessAdapter` da aviso al *agente de sitio* mediante un mensaje ACL de tipo `INFORM`.

A.4. Implementación de las adaptaciones al agente de sitio

Todas las adaptaciones descritas en la sección 3.8 son implementadas por la clase de java `CbrSiteService`. La figura A.5 muestra el código a insertar dentro del *agente de sitio* para habilitar todas las adaptaciones realizadas.

```
; <-- this is a comment...
; [Case Library setup]
; Case library definition for index to cases mapping service. if
; CaseLibraryPassword is omitted, then CbrAgent will ask for password
; at running time
CaseLibraryServer=jdbc:mysql://localhost/csidb
CaseLibraryDriver=com.mysql.jdbc.Driver CaseLibraryUser=ODBC
CaseLibraryPassword=

; [XCBR library Setup]
; Main java classes for XCBR library. Here goes the names for:
; CbrIndexer, CbrRetriever, CbrModifier and CbrStorer implementations.
; if you forget to put any of these, CbrAgent will
; die throwing an CbrException
CbrIndexer=xcbr.indexer.NNAIndexer
CbrRetriever=xcbr.retriever.NNARetriever
CbrModifier=xcbr.modifier.DummyModifier
CbrStorer=xcbr.storer.DummyStorer

; Parameters for XCBR library classes. all parameters will be sending
; through setup(String[] args) function of the respective
; implementation class!
CbrIndexerParams=
CbrRetrieverParams=
CbrModifierParams=
CbrStorerParams=

; [Ontology Setup]
; Ontology description (DAML+OIL, RDF, RDFS) for the ontology to cases
; mapping service. if no ontology is defined then this service
; will no be rendering by CBR agent.
Ontology=file://C:/java/OILED3.4/ont/cs1_0.daml

; [GUI (Graphical User Interface) setup]
; GUI definition for case library monitor service.
GUI=true
GUILookAndFeel=true
MonitorAgent=site1@ihatemycomputer:1099/JADE
```

Figura A.6: Ejemplo de un archivo de configuración del *Agente RBC.*

Apéndice B

Librería de casos basada en RDBMS (Relational DataBase Management System)

Lo que se requiere en todo sistema de RBC por simple o complicado que sea es un repositorio de casos. Este repositorio deberá contar con un sistema que maneje y soporte *índices* ya que un adecuado y eficiente manejo de *índices* hacen la diferencia entre un buen sistema de RBC y uno malo. Por esta razón es que se decidió el usar un RDBMS[1] tradicional.

El uso de un RDBMS ofrece muchas ventajas, como la seguridad de los datos, independencia de datos, formato estándar para datos e integridad en los datos. Esas características son proporcionadas por toda RDBMS por más básica que sea su funcionalidad.

La motivación detrás del uso de un RDBMS está contenida en dos características principales. Primero, toda organización distribuida maneja su información y conocimiento dentro de bases de datos relacionales, y en la mayoría de las veces, esta información y conocimiento necesita permanecer dentro de un control de seguridad, privacidad e integridad muy riguroso. Segundo, un número significativo de bases de datos, comerciales y libres, están disponibles para el manejo de información en las organizaciones. La aplicación de técnicas de RBC provee a los sistemas de información de las organizaciones nuevas características interesantes y útiles, puesto que los RDBMS actuales solo proveen capacidades de consulta primitivas (como lo son las consultas exactas).

Además, al esforzarse por convertir esas bases de datos en sistemas de RBC independientes se incurre en redundancia de información, inevitable perdida en el control de seguridad e integridad, y desperdicio de recursos computacionales. Aparte, desde que es creada la base de casos a partir de una RDBMS por algún mecanismo de conversión, la integridad de la base de casos no sería mantenida porque los mecanismos para reflejar automáticamente los cambios en la RDBMS a la base de casos no podría ser instalados por razones técnicas y de seguridad.

Algunas ventajas en el uso de RDBMS para la implementación de la librería de casos en este sistema de RBC en contraste con los sistemas de RBC tradicionales.

[1]Sistema de manejo de Bases de Datos Relacionales

1. **Control de seguridad.** En aplicaciones reales, los casos almacenados incluyen información secreta de una corporación o departamento. Ningún sistema de RBC desarrollado, incorpora alguna medida de seguridad. En la ausencia de control de seguridad, el sistema no puede ser usado en información altamente confidencial donde el máximo valor puede ser explotado.

2. **Escalabilidad.** La eficiencia de una aplicación de RBC depende enormemente del número de casos recolectados. En algunas aplicaciones reales, los casos recolectados pueden crecer drásticamente con el tiempo. Por ejemplo en el sistema SQUAD, alrededor de 3000 casos son agregados a su base de casos cada año. Los dominios reales normalmente son muy complicados y el uso de una indexación compleja necesitaría unos costos significativos de desarrollo y el comportamiento del sistema sería inestable porque los expertos en el dominio no entenderían totalmente la naturaleza de éste. Por tanto, una indexación compleja estaría más allá del control de los ingenieros del sistema.

3. **Velocidad.** A pesar de que varios métodos de indexación y organización de bases de casos han sido investigados, solo pocos le dan atención al costo computacional que estos métodos puedan tener. Un rápido acceso y almacenamiento de casos es una característica indispensable en aplicaciones del mundo real, particularmente para grandes bases de casos.

B.1. El Modelo Relacional

El modelo relacional fue propuesto en 1970. En esa época la mayoría de los sistemas de bases de datos estaban basados en uno de dos modelos de datos antiguos (el modelo jerárquico y el modelo de red); el modelo relacional revolucionó el campo de las bases de datos y reemplazó los otros dos modelos. En la actualidad el modelo relacional es por mucho el dominante de todos los demás modelos existentes, y es el modelo de datos principal de los productos líderes de DBMS, incluyendo DB2 de IBM, Informix, Oracle, Sybase, Access y SQLServer de Microsoft, FoxBase, Paradox, MySQL, entre otros.

El modelo relacional es muy sencillo y elegante; una base de datos es una colección de una o más relaciones, donde cada relación es una tabla con renglones y columnas. La mayor ventaja del modelo relacional sobre los otros modelos es su representación de datos sencilla y la facilidad con que complejas consultas pueden ser expresadas.

B.2. Relaciones

La piedra angular para la representación de datos en el modelo relacional es la relación. Una relación consta de un esquema de relación y una instancia de relación.

sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
50000	Jones	jones@cs	18	3.4
50000	Smith	smith@ee	18	3.2
50000	Smith	smith@math	19	3.8
50000	Madayan	madayan@music	11	1.8
50000	Guldu	guldu@music	12	2.0

Cuadro B.1: Una instancia $S1$ de la relación `Students`

sid	*name*	*login*	*age*	*gpa*
50000	Madayan	madayan@music	11	1.8
50000	Guldu	guldu@music	12	2.0
50000	Smith	smith@ee	18	3.2
50000	Smith	smith@math	19	3.8
50000	Jones	jones@cs	18	3.4
50000	Dave	dave@cs	19	3.3

Cuadro B.2: Una representación alternativa de la instancia $S1$ para la relación `Students`

Intuitivamente, la instancia de relación es una tabla, y el esquema de relación describe las columnas de la tabla. Primero describiremos el esquema de relación y después la instancia de relación. El esquema especifica el nombre de la relación, el nombre de cada campo (o columna, o atributo) y el dominio de cada campo. Un dominio es referido dentro de un esquema de relación por el nombre de dominio, y tiene un conjunto de valores asociados.

Podemos usar como ejemplo la información de un estudiante en base de datos en una universidad.

`Students(`***`sid`***`:string,` ***`name`***`:string, emphlogin:string,` ***`age`***`:integer,`
`gpa``:real)`

Esto dice, por ejemplo, que el campo llamado *sid* tiene un dominio llamado `string`. El conjunto de valores asociados con el dominio `string` es el conjunto de todas las cadenas de caracteres.

Una `instancia` de una relación es un conjunto de tuplas, también llamadas `registros`, en donde cada tupla tiene el mismo número de campos como el esquema de relación. Una instancia de relación puede ser imaginada como una tabla en donde cada tupla son los renglones y todos los renglones tienen el mismo número de

campos. La cardinalidad de una instancia de relación es el número de tuplas en él. El grado de una relación es el número de campos. En la tabla B.2, el grado de la relación es cinco y la cardinalidad de esta instancia es seis.

En una base de datos relacional es una colección de relaciones con distintos nombres. El esquema de la base de datos relacional es el conjunto de esquemas para las relaciones en la base de datos. En esta investigación fue llamada Librería de Casos al esquema de la Base de Datos Relacional y Base de Casos (table) a una instancia de relación dentro de la base de datos.

B.3. Creación y modificación de relaciones usando SQL-92

En SQL-92, hay muchos tipos de datos como `INTEGER`, `DOUBLE`, `VARCHAR`, entre otros. Además pueden ser definidos nuevos tipos con diversas instrucciones SQL-92.

La instrucción `CREATE TABLE` es usada para definir una nueva tabla. Para crear la relación Students, se pudo haber usado la siguiente instrucción:

```
CREATE TABLE Students ( sid    CHAR(20),
                        name   CHAR(20),
                        login  CHAR(20),
                        age    INTEGER,
                        gpa    REAL)
```

Las tuplas son subsecuentemente insertadas usando el comando `INSERT`. Se puede insertar una tupla a la tabla Student con:

```
INSERT
INTO   Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
```

Se pueden borrar tuplas usando el comando `DELETE`. Se pueden borrar todas las tuplas cuyo nombre sea `Smith` a la tabla Student con:

```
DELETE
FROM   Students S
WHERE  S.name = 'Smith'
```

Se pueden modificar valores en renglones existentes usando el comando `UPDATE`. Por ejemplo, podemos incrementar age y decrementar gpa del estudiante con `sid` igual a 53688:

sid	*name*	*login*	*age*	*gpa*
50000	Madayan	madayan@music	11	1.8
50000	Guldu	guldu@music	12	2.0

Cuadro B.3: Estudiantes con age $<$ 18

```
UPDATE Students S
SET    S.age = S.age + 1, S.gpa = S.gpa - 1
WHERE  S.sid = 53688
```

Se puede preguntar a la base de datos acerca del contenido de alguna tabla con el comando SELECT. Por ejemplo podemos recuperar todos los renglones correspondientes a estudiantes que son más jóvenes de 18 años (ver tabla B.3):

```
SELECT *
FROM   Students S
WHERE  S.age < 18
```

Apéndice C

Clasificación del área Inteligencia Artificial de acuerdo a la ACM (Association for Computing Machinery)

I.2 ARTIFICIAL INTELLIGENCE

I.2.0 General
Cognitive simulation
Philosophical foundations

I.2.1 Applications and Expert Systems (H.4, J)
Cartography
Games
Industrial automation
Law
Medicine and science
Natural language interfaces
Office automation

I.2.2 Automatic Programming (D.1.2, F.3.1, F.4.1)
Automatic analysis of algorithms
Program modification
Program synthesis
Program transformation
Program verification

I.2.3 Deduction and Theorem Proving (F.4.1)
Answer/reason extraction
Deduction (e.g., natural, rule-based)
Inference engines (NEW)
Logic programming

Mathematical induction
Metatheory**
Nonmonotonic reasoning and belief revision
Resolution
Uncertainty, ''fuzzy,'' and probabilistic reasoning

I.2.4 Knowledge Representation Formalisms and Methods (F.4.1)
Frames and scripts
Modal logic (NEW)
Predicate logic
Relation systems
Representation languages
Representations (procedural and rule-based)
Semantic networks
Temporal logic (NEW)

I.2.5 Programming Languages and Software (D.3.2)
Expert system tools and techniques

I.2.6 Learning (K.3.2)
Analogies
Concept learning
Connectionism and neural nets
Induction
Knowledge acquisition
Language acquisition
Parameter learning

I.2.7 Natural Language Processing
Discourse
Language generation
Language models
Language parsing and understanding
Machine translation
Speech recognition and synthesis
Text analysis

I.2.8 Problem Solving, Control Methods, and Search (F.2.2)
Backtracking
Control theory (NEW)

Dynamic programming
Graph and tree search strategies
Heuristic methods
Plan execution, formation, and generation
Scheduling (NEW)

I.2.9 Robotics
Autonomous vehicles (NEW)
Commercial robots and applications (NEW)
Kinematics and dynamics (NEW)
Manipulators
Operator interfaces (NEW)
Propelling mechanisms
Sensors
Workcell organization and planning (NEW)

I.2.10 Vision and Scene Understanding (I.4.8, I.5)
3D/stereo scene analysis (NEW)
Architecture and control structures**
Intensity, color, photometry, and thresholding
Modeling and recovery of physical attributes
Motion
Perceptual reasoning
Representations, data structures, and transforms
Shape
Texture
Video analysis (NEW)

I.2.11 Distributed Artificial Intelligence
Coherence and coordination
Intelligent agents (NEW)
Languages and structures
Multiagent systems (NEW)

I.2.m Miscellaneous

Bibliografía

[1] Selection engine home page, 2001. página web: http://ihatebaylor.com/technical/computer/ai/selection_engine/, http://selectionengine.sourceforge.net/.

[2] Java api documentation, 2003. página web: http://java.sun.com/api/index.html.

[3] David W. Aha. Case-based reasoning resources, 2002. página web: http://www.-aic.nrl.navy.mil/aha/research/case-based-reasoning.html.

[4] Bela Bollobas,Sheldon Axler. *Modern Graph Theory.* Springer Verlag, 1998.

[5] Hector Gibrán Ceballos Cancino. Manejo de ontologías en sistemas multiagentes por medio de un agente de ontologías aplicado a jitik. Master's thesis, ITESM, 2003.

[6] Erika Buen Rostro Cruz. Implementación de una plataforma de agentes para just-in-time information and knowledge. Master's thesis, ITESM. Tesis en proceso.

[7] Bellfemine Fabio. Jade administrator's guide, 2002. página web: http://sharon.-cselt.it/projects/jade.

[8] Bellfemine Fabio. Jade programmer's guide, 2002. página web: http://sharon.-cselt.it/projects/jade.

[9] Weiss G. *Multiagent Systems: A Modern Introduction to Distributed Artificial Intelligence.* MIT Press, 1999.

[10] Gary Cornell, Cay S. *Core Java 2*, volume I: Fundamentals. Prentice Hall, 2002.

[11] Jaron Collis, Divine Ndumu. The zeus agent building toolkit. the role modelling guide, 1999. The Information Management Domain.

[12] Jay Liebowitz, Tom Beckman. *Knowledge Organizations: What Every Manager Should Know.* St. Lucie Press, 1998.

[13] Janet Kolodner. *Case-Based Reasoning.* Morgan Kaufmann, 1993.

[14] David B. Leake. *Case-Based Reasoning, Experiences, Lessons and Future Directions.* MIT Press, 1996.

[15] Lean Suan Ong, Arcot Desai Narasimhalu. *The Handbook of Applied Expert Systems, Chapter 11, Case-Based Reasoning.* CRC Press, 1998.

[16] Leonardo Garrido Luna. Diseño de un sistema de asesoramiento basado en casos. Master's thesis, ITESM, 1992. Tesis terminada.

[17] M. V. Nagendra Prasad, Victor R. Lesser, Sussan E. Lander. *Retrieval and Reasoning in Distributed Case Bases.* 1995.

[18] Maydene Fisher, Jon Ellis, Jonathan Bruce, Jonathan Ellis. *JDBC API Tutorial and Reference, Third Edition.* Addison Wesley, 2002.

[19] Raghu Ramakrishnan. *Database Management Systems.* WCB/McGraw-Hill, 1998.

[20] Ramón Brena, José Luis Aguirre. Proyecto correa-rica, 2000. página web: http://-lizt.mty.itesm.mx/RICA2000.

[21] Ramón Brena, José Luis Aguirre. Redes informáticas de conocimiento con agentes, micai, 2000 mexican international congress on artificial intelligence, acapulco méxico, 2000.

[22] Ramón Brena, José Luis Aguirre. Just-in-time information and knowledge: Agent technology for km bussiness process, in technology, economic and social applications of distributed artificial intelligence, 2001 international conference on systems, man and cybernetics, tucson arizona, 2001.

[23] Stuart Russell, Peter Norving. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[24] Katia P. Sycara. *Multiagent Systems: summer.* AI Magazine, 1998.

[25] Uwe M. Borghoff, Remo Pareschi. *Information Technology For Knowledge Management.* Springer Verlag, 1998.

[26] Bain W. *Case-Based Reasoning: A Computer Model of Subjective Assessment.* PhD thesis, 1986.

[27] Ian Watson. *Appling Case-Based Reasoning: Techniques for Enterprise Systems.* Morgan Kaufmann, 1997.

[28] Ian Watson. Ai cbr, 2002. página web: http://www.ai-cbr.org/.

[29] Deitel y Deitel. *Cómo programar en java, 1a Ed.* Prentice Hall Hispanoamericana, 1998.