Dedicatorias

A Dios por estar siempre conmigo.

A mi mamá por enseñarme a salir adelante en la vida a pesar de los obstáculos que esta presenta.

A Rubén, mi esposo, por formar parte de mi proyecto de vida.

A Lilly e Iris, mis hermanas que me han apoyado incondicionalmente.

Agradecimientos

Agradezco al M.C. Rogelio Ferreira Escutia por todo el apoyo brindado en este proyecto, por sus consejos y asesorías. Muchas gracias por la confianza depositada en mí.

Agradezco a mis revisores M.C. Anastacio Antolino Hernández, M.C. José Alfredo Jiménez Murillo y M.C. Felipe Morales López, por sus comentarios y aportaciones en el desarrollo de esta tesis.

Agradezco al Consejo del Sistema Nacional de Institutos Tecnológicos (COSNET) por el financiamiento que me otorgaron para la realización de mis estudios de posgrado.

Agradezco también a la planta docente de la maestría en ciencias de la computación, por siempre apoyarme incondicionalmente.

A todas mis amigas por su apoyo y confianza en los tiempos difíciles y en los tiempos buenos.

A mis amigos de maestría por compartir este etapa profesional conmigo, pasando por esta experiencia juntos como amigos y no como compañeros.

Agradezco a tantas personas que han contribuido en el logro de este proyecto que, aunque no puedo nombrarlos a todos, solo puedo humildemente darles las gracias.

Resumen

La minería de datos es un proceso para el descubrimiento de conocimiento en bases de datos que ayuda, entre otras cosas, a la toma de decisiones en un nivel gerencial. Este proceso comúnmente se realiza de forma centralizada donde la especificación de los parámetros de minería de datos, así como la visualización de resultados, dependen de la presencia física del usuario.

Aprovechando la natural distribución de las bases de datos, es posible la utilización de agentes de software que realicen un proceso de minería de datos en un fragmento de la base de datos. Esta tesis presenta el desarrollo de multiagentes distribuidos utilizando JADE, el cual proporciona un conjunto de librerías y herramientas especialmente diseñadas para el desarrollo de multiagentes.

Los agentes desarrollados utilizan un algoritmo de Minería de Datos Descriptiva para encontrar información útil en red y permiten la visualización de resultados y la configuración de los parámetros de minería de datos a través de Internet o cualquier dispositivo móvil (como celulares y asistentes personales), de tal manera que el proceso de minería de datos es más automático, flexible y rápido que el proceso tradicional.

Abstract

The datamining is a process for the discovery of knowledge in databases that helps, among other things, to decision making in a managemental level. This process commonly it makes of centralized form where the specification of parameters of datamining, as well as the visualization of results, depend on the physical presence of the user.

Taking advantage of the natural distribution the databases, it is possible the use of agents of software who make process of datamining in a fragment of the database. This thesis presents the development of distributed multiagents using JADE, who provides a set of library and tools specially designed for the development of multiagents.

The developed agents use an algorithm of Descriptive DataMining to find information useful in network and they allow visualization of results and the configuration of the parameters of datamining through Internet or any movable device (like cellular phones and personal digital assistants), in such a way that process of datamining is more automatic, flexible and fast that the traditional process.

Índice general

Capítulo 1

Introducción

La minería de datos es un conjunto de técnicas que permiten obtener información de grandes bases de datos que este enfocada a la toma de decisiones, por lo que es aplicada por grandes compañías para establecer estrategias en sus negocios.

Existen herramientas de minería de datos tanto comerciales como de dominio público, las cuales realizan minería de datos descriptiva y/o predictiva. Sin embargo, estas herramientas siguen el proceso tradicional de minería de datos, el cual necesita adaptarse a las necesidades actuales de las empresas como son la distribución del proceso y el acceso vía web desde cualquier dispositivo móvil.

En esta tesis se diseñaron e implementaron multiagentes distribuidos en una red para la realización de minería de datos descriptiva para encontrar patrones de comportamiento en las compras que realizaban los clientes en un centro comercial, y con el conocimiento de éstos patrones puedan tomarse decisiones sobre las estrategias de mercado que deben aplicarse al negocio. Además, la especificación de los parámetros de minería de datos y la visualización de resultados es vía web.

1.1. Antecedentes Históricos

A mediados de los años 1980s, la tecnología de base de datos se caracterizaba por el desarrollo de modelos de datos avanzados (orientados a objetos, deductivos, etc.), sistemas de base de datos

orientados a aplicaciones espaciales, multimedia, bases de datos de conocimiento y científicas entre otras.

En este periodo, la creciente utilización de las bases de datos desemboca en la creación de grandes almacenes de datos (Data Warehousing) para guardar información de varias fuentes heterogéneas.

La tecnología de Data Warehouse incluye el procesamiento de transacciones en línea conocido como OLAP (On-Line Analytical Processing, Procesamiento Analítico en Línea) que a través de la utilización de técnicas de análisis con funcionalidades tales como abstracción, consolidación y agregación que permiten ver la información desde diversos ángulos. Las técnicas OLAP permitieron a los usuario organizar la información de tal manera que se obtuviera un panorama general de la información y en cierta medida ayudara en el proceso de toma de decisiones.

El manejo de las técnicas OLAP debía ser realizado manualmente por un experto, lo que consumía mucho tiempo y dinero, y dependía de la intuición de éste sobre qué información podría ser útil, esto producía errores ya éste no contaba con las herramientas necesarias para poder extraer la información realmente importante para la toma de decisiones.

La incapacidad de obtener, de la inmensa cantidad de datos almacenados, información no trivial para la toma de decisiones que no puede ser obtenida por las técnicas OLAP, propicia el nacimiento de la minería de datos como una herramienta para extraer información útil desde un Data Warehouse. Las técnicas de minería de datos permiten encontrar patrones importantes en los datos lo que contribuye grandemente en las estrategias de negocio [6].

1.2. Estado del Arte

La tendencia de las empresas es tener la mayor cantidad de información útil en bases de datos, por lo cual éstas pueden contener una gran cantidad de registros. La minería de datos se utiliza generalmente sobre bases de datos históricas gigantescas (contienen información de hace semanas, meses, años, etcétera).

El proceso tradicional de minería de datos es el siguiente [6]:

1. Obtención de los datos desde archivos planos, bases de datos y/o Data Warehouse.
2. Los datos inconsistentes, fuera de rango o con ruido son descartados para el análisis.
3. Se identifican atributos similares para su integración.
4. Se seleccionan los datos que son relevantes para el análisis.
5. Si los datos son de diferentes fuentes, los valores de los atributos son unificados.
6. Se realiza el proceso de minería, el usuario del proceso debe especificar el conocimiento que debe ser analizado.
7. Se evalúan los patrones obtenidos del proceso de minería y se determina cuales son de interés y se descartan los demás patrones.
8. Los patrones interesantes se presentan al usuario en la computadora donde se realizó el proceso de minería de datos, por lo que el usuario debe estar en contacto directo con ella para la visualización de resultados (y para la configuración de un nuevo proceso de minería de datos).
9. Si los patrones no representan información útil o no son suficientes para la toma de decisiones se debe reiniciar el proceso de minería con nuevos parámetros (todo esto de forma manual) y volver a evaluar y presentar los patrones obtenidos.

Actualmente, existe una gran cantidad de herramientas para realizar minería de datos. Muchas de las cuales cubren la mayoría de las técnicas de minería [11], y otras sólo algunas. Estas herramientas en la mayoría de los casos no son de dominio público, tales como: Enterprise Miner, Intelligent Miner, Clementine, Bussiness Miner, SQL Análisis Services, SPSS Data Mining, entre otros.

También existen herramientas de dominio público, entre las que realizan el método de asociación de reglas existen: WEKA, DBMiner, ARMiner recientemente Papyrus, este último es el único que realiza minería de datos distribuida sobre clusters y superclusters [12, 5, 10, 15, 7].

En la Tabla 1.1 se muestran las características de las herramientas de dominio público antes mencionadas [12, 5, 10, 15, 7].

Características	DBMiner	ARMiner	WEKA	Papyrus
Distribución del proceso de Minería de Datos	No	No	No	Si
Minería de datos Cliente-Servidor	Si	Si	Si	No
Freeware	Si	Si	Si	Si
Multiplataforma	No	Si	Si	No
Interfaz en Web	No	No	No	No
Interfaz en dispositivos móviles	No	No	No	No
Análisis de datos actualizados en tiempo real	No	No	No	No
Análisis de datos históricos	Si	Si	Si	Si

Tabla 1.1: Características de las herramientas de dominio público para minería de datos.

Las herramientas de minería de datos realizan el proceso tradicional de minería de datos, el cual no ha cambiado para adaptarse a las nuevas necesidades de las empresas que utilizan minería de datos, por lo que este proceso presenta los siguientes problemas:

1. Los resultados obtenidos reflejan relaciones pasadas, dado que son de datos de tiempo atrás.

2. Las herramientas de minería de datos no son de dominio público por lo que se deben de comprar a precios altos.

3. El proceso de minería de datos se desarrolla en forma "manual", es decir el usuario tiene que iniciar el proceso y ver los resultados desde la máquina donde reside el software de minería de datos, lo cual involucra una dependencia total del software de minería hacia el usuario.

Dado los problemas que presenta realizar el proceso de minería de datos sobre una sola gran base de datos se desarrolló el concepto de minería de datos distribuida. Distribuir el proceso de minería de datos es importante ya que disminuye el tiempo de ejecución del proceso, dado que se realiza el análisis en fragmentos de la base de datos, aprovechando que la mayoría de las empresas tienen sus bases de datos fragmentadas en diferentes computadoras o sitios.

Con excepción de Papyrus, las herramientas de minería de datos no distribuyen el proceso de minería de datos en varias computadoras, lo que implica que en un servidor se tiene una gran base de datos histórica (información de hace semanas, meses, años, etc.) y sobre ésta se realiza el proceso de minería de datos.

Desde su nacimiento, la minería de datos no tiene un avance tecnológico significativo hasta que los sistemas de de bases de datos basadas en web hacen su aparición en los años 1990s, es entonces cuando se utiliza minería de datos para obtener relaciones entre la búsqueda de información especificada por el usuario y las páginas web visitadas, a esto se le conoce como Web Mining (Minería Web) [6].

Desde el año 2000 hasta el presente la minería de datos ha evolucionado lentamente y actualmente existen trabajos sobre la creación de un lenguaje de consultas de minería de datos conocido por sus siglas en inglés como DMQL (Data Mining Query Language, Lenguaje de Consultas de Minería de Datos), la utilización de algoritmos de inteligencia artificial y la utilización de agentes para optimizar el proceso de minería de datos [6].

1.3. Objetivo

Diseñar e implementar multiagentes distribuidos que utilizan algoritmos de *Minería de Datos Descriptiva* para encontrar información útil en red y que son programables vía web para permitir la visualización de resultados y la reconfiguración de los parámetros de minería de datos a través de Internet o cualquier dispositivo móvil (como celulares y asistentes personales), de tal manera que el proceso de minería de datos sea más automático, flexible y rápido que el proceso tradicional.

1.4. Justificación

Actualmente, las empresas que invierten en minería de datos son empresas comerciales que cuentan con varias sucursales y desean analizar la información actual (no de hace semanas, ni meses) conjunta de éstas. El proceso de minería de datos debe estar enfocado a usuarios

gerenciales que comúnmente están en movimiento administrando un conjunto de empresas, por lo que el software de minería debe permitirle la configuración de parámetros y puesta en marcha del proceso, así como la visualización de resultados desde cualquier parte en que se encuentre localizado el usuario gerencial.

Otro aspecto importante es la naturaleza de las base de datos de las empresas actuales, la mayoría cuenta con una base de datos relacional distribuida, por lo que si se realizara el proceso tradicional de minería de datos sería necesario enviar por la red los fragmentos de la base de datos distribuida para su integración en una sola computadora donde se realizaría el configuración de los parámetros del proceso de minería de datos y la presentación de los patrones obtenidos. Un nuevo análisis requeriría enviar nuevamente todos los fragmentos de la base de datos lo que consumiría mucho tiempo.

Se optó por la utilización de multiagentes distribuidos para aprovechar la natural distribución de la base de datos y así dividir el proceso de minería de datos en procesos locales de minería que son más rápidos. Los multiagentes distribuidos fueron desarrollados en lenguaje Java y JADE para hacerlos multiplataforma, debido a que generalmente se tienen máquinas con distintos sistemas operativos en una empresa.

Además, se desarrolló una interfaz móvil con el usuario gerencial para permitirle la configuración de los parámetros de minería y la visualización de resultados de manera oportuna y flexible en cuanto a la dependencia del software de minería y el usuario.

En la realización de minería de datos descriptiva se utilizó el algoritmo de Asociación de Reglas [1], enfocado a una aplicación de "análisis de canasta" de una tienda comercial, es decir, se analizan las compras de artículos de los clientes.

1.5. Descripción por Capítulos

El en presente Capítulo I se da una introducción sobre minería de datos y su evolución, así como una revisión sobre algunas herramientas de dominio público para minería de datos y algunas líneas de investigación actuales sobre este tema. Además, se plantea el objetivo del trabajo y su justificación.

El Capítulo II, constituye el marco teórico sobre el que se sustenta este trabajo y se conforma de dos temas principales: minería de datos y agentes de software. La primera parte aborda el concepto y tipos y arquitectura de un sistema de minería de datos, así como una revisión de las características de las diferentes algoritmos existentes para minería de datos. En la segunda parte se describe el concepto, propiedades y tipos de los agentes de software, junto con una revisión de las características de algunas herramientas libres para el desarrollo de agentes; y por último, se realizó la comparación de las herramientas revisadas y la selección de la más adecuada para el este proyecto.

En el Capítulo III se muestra todo lo concerniente al diseño del proyecto; se describe el objetivo del presente trabajo y el diseño de la arquitectura de red y comunicación del sistema de multiagentes distribuidos.

El Capítulo IV consiste en la implementación de las diferentes partes del proyecto. Se muestra el algoritmo de minería de datos descriptiva y la implementación de los diversos agentes con la herramienta de desarrolla JADE.

El Capítulo V consiste en las pruebas realizadas al sistema implementado; dichas pruebas se dividen en tres casos, cada uno con diferentes condiciones iniciales. También se especifican los resultados obtenidos, se hace una comparativa de las características del presente proyecto y de algunas herramientas de minería de datos de dominio público, y por ultimo se describen las ventajas y desventajas del presente trabajo.

Finalmente, en el Capítulo VI se muestran las conclusiones obtenidas del presente proyecto y sus trabajos futuros.

Capítulo 2

Marco Teórico

En este capítulo se presentan los conceptos básicos, tipos y diferentes algoritmos existentes de minería de datos. También se presenta el concepto y propiedades de los agentes de software, así como una recopilación y comparación de las diferentes características de las herramientas de desarrollo de agentes, de las cuales se seleccionó la más adecuada para el proyecto.

2.1. Minería de Datos

La minería de datos es una tecnología reciente, consecuencia del crecimiento de las base de datos, que permite el análisis de información para la toma de decisiones, así como el planteamiento y descubrimiento automático de hechos e hipótesis, ya sean patrones, reglas, grupos, funciones, modelos, secuencias, relaciones, correlaciones, etc.

Esto permite a las empresas la posibilidad de obtener información para la toma de decisiones de sus grandes bases de datos, información que ordinariamente no se obtendría sin utilizar técnicas de minería de datos, haciéndolas más competitivas en el mercado.

2.1.1. Concepto de Minería de Datos

La ***minería de datos*** (o **Data Mining** en inglés) se refiere a la extracción o "minería" de conocimiento de grandes almacenes de información [6].

Por lo tanto, la minería de datos es un proceso que, a través del descubrimiento y cuantificación de relaciones predictivas y descriptivas en los datos, permite transformar la información disponible en conocimiento útil de negocio.

La minería de datos es un paso esencial del proceso de descubrimiento del conocimiento en bases de datos o **KDD (Knowledge Discovery in Databases)**.

El proceso de descubrimiento del conocimiento consiste de los siguientes pasos [6]:

1. **Limpieza de los datos**.- En este paso se eliminan los datos inconsistentes o con ruido.
2. **Integración de los datos**.- Se refiere a la integración de datos provenientes de múltiples fuentes de datos.
3. **Selección de los datos**.- Se obtiene de la base de datos los datos relevantes para el análisis de tareas.
4. **Transformación de los datos**.- En este paso los datos son transformados a una forma adecuada para que puedan utilizarse en un proceso de minería de datos.
5. **Minería de datos**.- Se aplican métodos inteligentes para extraer de los datos patrones de comportamiento de los datos.
6. **Evaluación de patrones**.- Se identifican los patrones interesantes que representan el conocimiento basándose en algunas medidas de interés.
7. **Presentación del conocimiento obtenido**.- Se refiere a la visualización y representación del conocimiento obtenido del análisis de minería de datos al usuario.

2.1.2. Arquitectura de un Sistema de Minería de Datos

La arquitectura típica de un sistema de minería de datos consiste de los siguientes componentes [12]:

1. **Bases de datos, data warehouse y otro almacenes de información**
   Un sistema de minería de datos puede obtener datos de diversas fuentes como bases de

datos, data warehouse y otros tipos de almacenamiento de información; para la unión de estos datos se utilizan técnicas de limpieza e integración sobre los datos.

2. **Servidor de bases de datos o data warehouse**
   El cual es responsable de obtener los datos relevantes basándose en los requisitos de minería de datos especificados por el usuario.

3. **Base del conocimiento**
   Es el conocimiento que será usado para guiar la búsqueda de patrones y posteriormente la evaluación de los mismos.

4. **Software de minería de datos**
   Consiste en un conjunto de módulos funcionales para tareas tales como caracterización, asociación, clasificación, análisis de grupos y análisis de desviación.

5. **Evaluación de patrones**
   Este componente comúnmente emplea medidas de interés e interactúa con los módulos de minería de datos para dirigir la búsqueda hacia los patrones de interés para el usuario. Se pueden utilizar umbrales o límites mínimos de las medidas de interés para filtrar los patrones descubiertos.

6. **Interfaz gráfica con el usuario**
   Este módulo comunica al usuario con el sistema de minería de datos, permitiéndole interactuar con el sistema para especificar una nueva consulta a analizar sobre los datos, proveer información que facilite la búsqueda de patrones y visualizar los resultados obtenidos del proceso de minería de datos.

2.1.3. Tipos de Minería de Datos

Las funciones de minería de datos son usadas para especificar las clases de patrones que serán encontrados por las tareas de minería de datos. En general, las tareas de minería de datos pueden clasificarse en dos categorías:

1. ***Minería Descriptiva***.- Describe los datos en forma concisa y presenta en forma general propiedades interesantes de los datos. Se aplica para descubrir patrones en datos existentes para guiar el proceso de toma de decisiones.

2. ***Minería Predictiva***.- Construye un conjunto de modelos, aplicando inferencia sobre los datos disponibles, haciendo intentos para predecir el comportamiento de nuevos conjuntos de datos.

El tipo de minería de datos a utilizar depende de los patrones que el usuario desee descubrir.

2.1.4. Algoritmos de Minería de Datos

Cada algoritmo de minería de datos especifica diferentes clases de patrones que pueden ser descubiertos, por lo tanto, el tipo de algoritmo que se utilice depende de la aplicación y de los resultados que se esperan obtener, así como del tipo de minería que se desee realizar, debido a que algunos algoritmos se adaptan mejor a determinado tipo de minería de datos.

Los algoritmos de minería de datos son:

1. Descripción de conceptos o clases (caracterización y discriminación)
2. Análisis de asociación (asociación de reglas)
3. Clasificación y predicción
4. Análisis de grupos (agrupamiento)
5. Análisis de regresión

2.1.4.1. Caracterización y Discriminación

En este algoritmo los datos son asociados en clases o conceptos. Describir a los datos como un conceptos o clases individuales nos permite resumir la información en términos precisos. A estas descripciones se les conoce como descripciones de clase o concepto.

Estas descripciones pueden derivarse en:

1. Una *caracterización de los datos*, separando los datos en clases identificadas previamente, llamadas clases objetivo (**target class**).

2. Una *discriminación de los datos*, comparando las clases objetivo con una o varias clases de comparación, llamadas clases de contraste (**contrasting class**).

3. Una combinación de caracterización y discriminación de los datos.

El resultado de la caracterización y la discriminación es un agrupamiento de los datos en clases objetivo que determinan un resumen de las características generales de los datos pertenecientes a ellas. Los datos correspondientes a las clases especificadas por el usuario son normalmente tomados de una base de datos a través de varias técnicas como consultas SQL y operaciones OLAP.

La caracterización y discriminación se utiliza para aplicaciones en las que es necesario separar los datos en categorías que nos permitan determinar las características de un grupo de datos.

2.1.4.2. Asociación de Reglas

El *análisis de asociación* es el descubrimiento de reglas de asociación que muestran las condiciones de relación atributo-valor que pueden ocurrir frecuentemente juntos en un conjunto de datos dado.

Puede entonces relacionarse el valor de un tributo dados el valor de uno o varios atributos, a estas relaciones se les denomina *reglas de asociación*.

Cada regla esta compuesta de la siguiente forma: $X \Rightarrow Y[soporte, confianza]$, que significa "el conjunto de tuplas de una base de datos que satisfaga la condición en X y que por lo tanto satisfaga las condiciones en Y". El soporte y confidencia de una regla son dos medidas (en porcentaje) que reflejan la utilidad y la certeza de la regla descubierta.

El *soporte* de una regla es el porcentaje de transacciones en el conjunto de datos que contienen tanto a X como a Y, es decir $P(X \cup Y)$.

La *confianza* de una regla es el porcentaje de transacciones en el conjunto de datos que contienen X y por lo tanto también contienen Y, es decir $P(Y \mid X)$.

En resumen, el soporte determina el total de transacciones que contienen X o Y mientras que la confianza determina el número de transacciones que contienen a X y a Y.

Típicamente, las reglas de asociación consideradas de interés para la toma de decisiones son las que satisfacen el umbral mínimo de soporte y el umbral mínimo de confianza, denominadas reglas "fuertes"; tales umbrales pueden ser especificados por los usuarios del proceso de minería de datos o por expertos en reglas de asociación.

La minería de reglas de asociación es un proceso de dos pasos:

1. Encontrar todos los conjuntos de datos frecuentes.- cada uno de estos conjuntos ocurrirá con al menos la misma frecuencia que los valores mínimos predeterminados de soporte.
2. Generar reglas de asociación fuertes desde el conjunto de datos frecuentes.- estas reglas deben satisfacer los valores de soporte y confianza mínimos.

El cumplimiento de estas reglas determina patrones de comportamiento determinantes en los datos por lo este algoritmo es muy usado para aplicaciones comerciales en las que es importante determinar el patrón de compras de productos, movimiento de inventarios, etc.

2.1.4.3. Clasificación y Predicción

La *clasificación* es el proceso de encontrar un conjunto de modelos (o funciones) que describen y distinguen clases de datos o conceptos, con el propósito de utilizar los modelos encontrados para predecir la clase de un objeto cuya etiqueta de clase es desconocida.

El modelo derivado esta basado en un análisis de un conjunto de datos de entrenamiento (datos que no han sido clasificados en clases definidas) y puede ser representado de varias formas como reglas IF-THEN, árboles de decisión o redes neuronales.

La clasificación puede ser usada para predecir la clase a la que pertenece un objeto. Sin embargo en muchas aplicaciones los usuarios requieren predecir valores perdidos o que no están definidos como una clase, esto pasa comúnmente cuando los valores son datos numéricos que se desean predecir. Por lo tanto, la clasificación es usada para aplicaciones que deseen sólo predecir a que clase pertenece ciertos datos. Por ejemplo, se puede aplicar la clasificación en los datos de una tienda departamental para determinar clases como cliente frecuente, cliente medio, cliente esporádico y predecir a que clase pertenece un cliente.

2.1.4.4. Análisis de Grupos

El *análisis de grupos* se refiere al análisis de objetos de datos sin consultar una clase definida. En general, las clases definidas no están presentes en los datos de entrenamiento debido a que no se conocen al principio. El análisis de grupos puede ser usados para generar las clases de un conjunto de datos.

En el proceso del análisis, los datos son agrupados basándose en el principio de maximizar las semejanzas entre los datos pertenecientes a una clase y minimizar las semejanzas entre las diferentes clases. Cada grupo obtenido del análisis puede derivarse en una regla de agrupación.

2.1.4.5. Análisis de Regresión

Generalmente, las bases de datos contienen datos que no obedecen a una conducta o modelo en los datos. Estos datos son objetos fuera de rango que la mayoría de los algoritmos de minería de datos los descartaría por considerarlos valores perdidos o excepciones.

Los valores fuera de rango son detectados usando métodos estadísticos que asumen una distribución o modelo de probabilidad para los datos, o usando medidas de distancia donde los datos que se encuentran a una distancia significativa de un grupo son considerados datos fuera de rango.

El análisis de regresión puede usarse para descubrir conductas fraudulentas como por ejemplo para detectar en una cuenta de tarjeta de crédito compras por cantidades extremadamente grandes en comparación con los cambios regulares en la misma cuenta (es decir, cambios en el comportamiento de compras hechas con esa tarjeta). Los valores fuera de rango pueden ser detectados con respecto a la localización, frecuencia y tipo de la compra.

2.1.5. Áreas de Aplicación de Minería de Datos

En la actualidad, existe una gran cantidad de aplicaciones en las que se utiliza la minería de datos, en áreas tales como:

1. *Astronomía*: clustering y clasificación de cuerpos celestes.
2. *Biología molecular*: predicción de substancias cancerígenas, genoma humano, etc.
3. *Aspectos climatológicos*: predicción de tormentas, etc.
4. *Medicina*: caracterización y predicción de enfermedades.
5. *Industria y manufactura*: diagnóstico de fallas.
6. *Mercadotecnia*: identificar clientes susceptibles de responder a ofertas de productos y servicios por correo, selección de sitios de tiendas, etc. Inversión en casas de bolsa y banca: análisis de clientes, aprobación de préstamos, etc.
7. *Detección de fraudes y comportamientos inusuales*: telefónicos, seguros, electricidad, etc.
8. *Análisis de canastas*: análisis de canastas de mercado para mejorar la organización de tiendas.

2.2. Agentes de Software

2.2.1. Concepto de Agente

Según Russell [9], "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".

Los agentes son entidades de software que "viven" en un entorno, del cual pueden obtener información y efectuar acciones que contribuyan al logro de un objetivo o meta [9].

2.2.2. Propiedades de los Agentes

Las propiedades de los agentes son las siguientes [13]:

1. **Autonomía**.- Capacidad de operar sin intervención humana.

2. **Movilidad**.- Habilidad del moverse en forma autónoma.

3. **Adaptabilidad**.- Puede ajustarse a los métodos, hábitos de trabajo y preferencias del usuario.

4. **Colaboración**.- Permite corregir algunos errores causados por el usuario, como la omisión de información, información ambigua, etc.

5. **Capacidad de reacción**.- Pueden reaccionar a los eventos que surgen en su entorno.

2.2.3. Tipos de Agentes

Existen varios tipos de agentes [13] que se diferencian por su nivel de cooperación, su nivel de inteligencia para tomar decisiones en base a su entorno, la localización local o remoto de los agentes y a su capacidad de cambiar de entorno de ser necesario.

Entre los tipos de agentes más comunes podemos nombrar a los agentes autónomos, agentes inteligentes, agentes cooperativos, agentes móviles, agentes distribuidos y multiagentes. Existen otros tipos de agentes que son híbridos de los anteriores.

Agentes autónomos
Son aquellos agentes que habitan en entornos complejos y dinámicos, que sienten y actúan autónomamente en su entorno intentando realizar el conjunto de objetivos marcados.

Agentes inteligentes
Son agentes que pueden llevar a cabo conjuntos de operaciones en representación del usuario o de otro programa, siempre con un alto grado de independencia y cuyas acciones y resultados no difieren demasiado de lo que el usuario obtendría si las realizara él mismo.

Agentes cooperativos
Estos agentes acentúan las características de autonomía y de cooperación con otros agentes con el objeto de realizar tareas efectivas para sus dueños. Los atributos más evolucionados en estos agentes son la autonomía, sociabilidad y auto-actividad.

Agentes móviles
Los agentes móviles son procesos computacionales capaces de navegar por redes WAN, como

Internet, interactuando con equipos, acumulando información en nombre de sus propietarios, y volver después de haber realizado las tareas requeridas por el usuario para informar a este de los resultados.

Agentes distribuidos
Son agentes que se comunican con otros agentes que se encuentran geográficamente distantes para alcanzar un objetivo global.

Multiagentes
Son sistemas integrados por un conjunto de agentes distribuidos con capacidad de coordinar sus actuaciones individuales para conseguir un objetivo global.

Cada agente realiza un determinado proceso y es capaz de comunicarse con otros agentes que residen en otro sitio para cumplir el objetivo global.

Los sistemas multiagentes (SMA) son efectivos cuando el conocimiento para resolver un problema está distribuido en varios sitios.

2.2.4. Herramientas para el Desarrollo de Agentes

Existen herramientas que facilitan en desarrollo de agentes, cada una cuenta con diferentes características que hacen que se adapten mejor a determinada aplicación a desarrollar. Entre estas herramientas se encuentran JADE, JafMas, JATLite y Zeus, de las cuales se tiene una comparativa de sus características [14].

2.2.4.1. Características

Las herramientas de desarrollo de sistemas multiagentes poseen una serie de características que heredan las aplicaciones creadas y que hacen que esas aplicaciones sean más o menos eficientes en la solución de un problema específico. De dichas características, las más importantes son:

Modelo de arquitectura
Desde hace tiempo se está tratando de conseguir una arquitectura estándar que se utilice para todos los sistemas de agentes, aunque por ahora todavía no se ha unificado este criterio y

existen varios modelos que son utilizados por las diferentes herramientas. De esos intentos de estandarizar las arquitecturas, destacan sobre todo los siguientes:

1. **Modelo OMG.-** El grupo OMG (Object Manager Group) ha desarrollado el estándar MASIF (Mobile Agent System Interoperabilities Facility), desarrollo en el que han participado empresas como IBM, General Magic, Crystaliz y GMD-Fokus.

   El estándar MASIF propone el desarrollo de sistemas de agentes cuyos entornos se basan en una composición entre agentes (componentes) y agencias (lugares), entidades que colaboran utilizando patrones y políticas de interacción generales. En este modelo, los agentes se caracterizan por sus capacidades (por ejemplo: planificación), tipos de interacciones (por ejemplo: síncronas o asíncronas), y sobre todo por la movilidad (por ejemplo: estáticos, móviles con estado o sin estado). Por otro lado, las agencias soportan la ejecución concurrente de los agentes y proporcionan seguridad y movilidad, entre otras cosas. Es por tanto un estándar que facilita la creación de sistemas de agentes móviles.

2. **Modelo FIPA.-** El estándar FIPA (Foundation for Intelligent Physical Agents) recoge casi todas las vistas que se tienen de un sistema de agentes (gestión, seguridad, movilidad o comunicación), de tal forma que para cualquiera de esas vistas todos los esfuerzos se hagan en una única dirección. Este estándar se ha convertido en un referente para una gran parte de las herramientas basadas en agentes. Particularmente importante en el estándar FIPA son las especificaciones del Agent Management, del Agent Communication Language (ACL) y de un conjunto de agentes que ayudan a la correcta gestión de los sistemas (DF, AMS, etc.).

3. **Modelo ARPA.-** El Knowledge Sharing Effort (KSE) era un consorcio centrado en el desarrollo de normas para facilitar la compartición y reutilización de bases de conocimientos y sistemas basados en conocimientos, sin embargo en la actualidad prácticamente ha desaparecido y tan solo mantiene activos algunos grupos de investigación independientes. Sin embargo aquella labor ha dado como resultados algunas especificaciones todavía muy extendidas como la del lenguaje KQML que ha sido tomado como estándar de facto para el intercambio de conocimiento por muchas herramientas de desarrollo de sistemas multiagente.

Lenguaje de comunicación entre agentes

Al igual que con los modelos de arquitectura, tampoco existe un lenguaje estándar específico

utilizado por todas las herramientas, y aunque existen algunos más, son los dos siguientes los más utilizados para implementar la comunicación entre los agentes:

1. **KQML** (Knowledge Query and Manipulation Language, Lenguaje de Consulta y Manipulación de Conocimiento) es un lenguaje y un conjunto de protocolos, desarrollado y adoptado por el estándar ARPA, que soporta la identificación, conexión e intercambio de información entre agentes. Es un lenguaje dividido en tres niveles, nivel de contenido, nivel de mensaje y nivel de comunicación. Esto quiere decir que una expresión KQML está formada por un contenido, que es lo que el agente pretende comunicar, un encapsulado en forma de mensaje y finalmente unas directivas de comunicación (identidades, direcciones, etc.).

2. **ACL** (Agent Communication Language, Lenguaje de Comunicación del Agente) ha sido desarrollado por FIPA a partir del KQML, como una evolución de KQML. Los mensajes de ACL están formados por un identificador del tipo de comunicación, que define el significado principal del mensaje (inform, request, agree, etc.), y una secuencia de parámetros, que son un conjunto de parejas clave-valor que permiten asociar a cada mensaje la información necesaria.

Modelo de Comunicación entre Agentes

Se distinguen dos maneras mediante las que se pueden comunicar los agentes, y que pueden ser implementadas o no por las herramientas. Estas dos formas de comunicación son:

1. **Intercambio de mensajes.-** Es el modelo más común de comunicación dentro de una plataforma multiagente, y es implementado por la extensa mayoría de las herramientas. Tiene también dos variantes, el intercambio uno-a-uno, en donde son dos los agentes que intervienen uno enviando y otro recibiendo el mensaje, y el intercambio por multicast o broadcast, que es muy útil cuando una plataforma está dividida en subsistemas y se quiere enviar un mensaje a todos los agentes pertenecientes a ese subsistema.

2. **Comunicación directa.-** Este tipo de comunicación se da entre dos agentes, y no se suele implementar en demasiadas herramientas. Se utiliza cuando se sabe que dos agentes van a tener una colaboración especial entre ellos. El beneficio de este tipo de comunicación es que se evita la transformación de los mensajes a cadenas de bytes por lo tanto

es mucho más rápida. La razón de que no sea utilizada para todas las comunicaciones dentro de un sistema es que mantiene siempre abierta una línea de comunicación entre los agentes participantes, por lo que siempre se están consumiendo recursos aunque no se esté produciendo comunicación, lo que supondrá una sobre saturación del sistema si este posee un número elevado de agentes.

Interfaz de desarrollo y control de la ejecución

No son indispensables, pero sí es muy aconsejable que las herramientas esten provistas con algún tipo de interfaz que facilite a desarrolladores y administradores la implementación y el control de las plataformas de agentes. Existen dos tipos de interfaz según su utilidad que son:

1. **Interfaz de desarrollo.-** cuya existencia permite normalmente que la definición de los agentes, de las interrelaciones entre ellos, de las tareas, etc., en definitiva, la definición del sistema en su conjunto, se realice de forma mucho más rápida y sencilla. Esto lo consiguen mediante la abstracción de las tareas más complejas de implementación.

2. **Interfaz de control.-** Este interfaz será el que utilice el administrador del sistema y se presenta cuando la plataforma comienza su ejecución. Facilita el control (ya sea local o remoto) del ciclo de vida de la plataforma de agentes y de todos los agentes que habitan en ella (creación, eliminación, movilidad, reinicio, etc.).

Lenguaje de desarrollo

Normalmente todas las herramientas están implementadas en un lenguaje orientado a objetos, y este lenguaje en la mayoría de los casos es Java, aunque también las hay que están implementadas en otros lenguajes como C++ o Smalltalk. Sin embargo y pese a la utilización de un solo lenguaje, en algunas herramientas se pueden combinar varios, por ejemplo creando agentes en C++ e integrándolos en plataformas Java.

2.2.4.2. JADE

JADE (Java Agent DEvelopment framework, Ambiente de Desarrollo de Agentes en Java) es un proyecto desarrollado por el CSELT (Centro Studio e Laboratori Telecomunicazione, Centro de Estudio y Laboratorio de Telecomunicaciones), perteneciente al Departamento de Ingeniería

de la Información de la Universidad de Parma, que se lleva a cabo desde el año 1997 y que sigue en contínua evolución. Es una herramienta software totalmente implementada en Java, cuyo objetivo es simplificar el desarrollo de sistemas multiagentes a través de un conjunto de sistemas, servicios y agentes.

Las principales características que ofrece JADE son:

1. Arquitectura: JADE es una de las herramientas que intenta contemplar en sus sistemas los estándares FIPA en su totalidad.
2. Distribución: Freeware.
3. Interfaz: No presenta un interfaz para el desarrollo pero sí para el control de la ejecución del sistema de agentes.
4. Lenguaje de comunicación: Utiliza el lenguaje estándar de FIPA, el ACL.
5. Movilidad: La movilidad es posible dentro de la plataforma JADE.
6. Creación de agentes: JADE permite el registro y eliminación automática de agentes con el AMS (Agent Management System) en cualquier momento de la ejecución ya sea local o remotamente.
7. Lenguaje de programación: JADE está implementada en Java.

Además, JADE proporciona un conjunto de herramientas (agentes) que simplifican la administración de la plataforma de agentes. Actualmente las que están disponibles son las siguientes:

1. El Remote Management Agent (RMA) actúa como consola gráfica para la gestión y control de la plataforma. Es necesaria para iniciar el resto de herramientas.
2. El Dummy Agent es una herramienta de depuración y visualización, en la que es posible componer mensajes ACL y enviarlos, visualizar la lista de todos los mensajes ACL enviados o recibidos, así como la información contenida en dichos mensajes.
3. El Sniffer es un agente capaz de interceptar mensajes ACL y mostrarlos gráficamente utilizando una notación similar a los diagramas de secuencia de UML. Es bastante útil

para depurar las sociedades de agentes mediante la observación de cómo intercambian mensajes ACL.

4. El SocketProxyAgent es un agente simple que actúa como enlace bidireccional entre una plataforma JADE y una conexión TCP/IP, y se utiliza para canalizar los mensajes entrantes y salientes de una plataforma JADE.

5. El DF GUI es interfaz gráfico utilizado junto con el Directory Facilitator de JADE para crear complejas redes de dominios y subdominios. Esta interfaz permite controlar el conocimiento de los DFs y registrar, modificar o buscar dentro de ellos.

2.2.4.3. JafMas

Jafmas (Java Agent-based Framework for Multi-Agent Systems, Ambiente de Desarrollo basado en Java para Sistemas Multiagente) es una herramienta desarrollada por la División de Investigación y Estudios Avanzados de la Universidad de Cincinnati que proporciona una metodología para el desarrollo de sistemas multiagente basados en la comunicación, una arquitectura de agentes y un conjunto de clases escritas en Java para dar soporte a la implementación de estos agentes.

Las principales características que ofrece Jafmas son:

1. Arquitectura: Es una composición de diferentes capas de abstracción.

2. Distribución: Freeware.

3. Interfaz: Jafmas no posee un interfaz gráfico propio para el desarrollo de aplicaciones, pero sí existe una entorno complementario desarrollado también por la Universidad de Cincinnati, que dota a la herramienta de dicho interfaz. Ese entorno se denomina Jive, está integrado en Jafmas, y utiliza las clases de este último para generar el código principal de la aplicación que se está desarrollando, quedando para la implementación el código específico de agentes y tareas. Además también posee un interfaz para el control de la ejecución.

4. Movilidad: Jafmas no ofrece características de movilidad a sus agentes durante su ejecución, aunque permite la creación de estos en "hosts" remotos.

5. Lenguaje de comunicación: Se utiliza el estándar KQML para la comunicación entre agentes.

6. Localización de agentes y recursos: Existe una serie de clases que proporcionan esta función, son las clases RequestedResrcProvider y ReqdResource.

7. Lenguaje de programación: Jafmas está escrita íntegramente en Java.

La arquitectura de Jafmas se compone de varias capas en las que se ubican las diferentes clases de implementación:

1. Las dos capas que forman la base del conjunto, son la del hardware y la del sistema operativo, sobre las que reside el resto del sistema.

2. Las capas que forman el modelo local de los agentes son las tres siguientes, la capa de la infraestructura del agente, y sobre ella la capa de protocolos de comunicación (modos de comunicación utilizados) y la capa lingüística (donde se configurarán los mensajes enviados y recibidos). Jafmas soporta tanto comunicación directa como por broadcast. Las clases que se incluyen en estas capas de comunicación dentro de cada agente son MulticastCom, MulticastGroup y DirectedComImpl.

3. El modelo social del agente es la siguiente capa de la arquitectura, que se construye sobre las capas de comunicación, y proporciona los servicios de coordinación de los agentes. Implementa las clases RequestedResrcProvider, ReqdResource, Conversation, ConvRule y MsgRouter.

4. Por encima de estos niveles reside la capa de la aplicación multiagente que está compuesta por los agentes Jafmas que son implementados mediante la extensión de la clase Agent.

5. Y por último está la capa del interfaz, en donde se proporcionan diferentes tipos de interfaz que ayudan al usuario a interactuar con la aplicación, y que se implementan a través de las clases CreateAgent, ConvOpInterfaz, ConvCanvas, AgentOpInterfaz y AgentCanvas.

2.2.4.4. JATLite

JATLite (Java Agent Template, Lite; Plantilla de Agentes Java, Lite) es un paquete de programas escritos en lenguaje Java que ha sido desarrollado por la Universidad de Stanford y que

habilitan la creación de sistemas de agentes caracterizados principalmente por la robustez de sus comunicaciones.

JATLite es una herramienta de carácter no comercial. La utilización de un AMR (Agent Message Router) que permite a los agentes registrarse en la plataforma utilizando un "login" y un "password", conectarse o desconectarse a Internet automáticamente, o enviar o recibir mensajes por FTP, proporcionando una gran robustez a los desarrollos.

Algunas de las características que ofrece JATLite son las siguientes:

1. Arquitectura: JATLite posee una arquitectura especial dividida en capas.
2. Interfaz: Posee un interfaz para el desarrollo y otro para el control de la ejecución.
3. Movilidad: Permitida y controlada por el AMR.
4. Creación de agentes: Puede ser local o remota.
5. Lenguaje de comunicación: JATLite utiliza el estándar KQML para la comunicación entre sus agentes.
6. Lenguaje de programación: Se utiliza Java, pero también puede integrar código C++.
7. Localización de agentes y recursos: JATLite presenta el AMR Agent Message Router, que funciona básicamente como un servidor de correo electrónico.
8. Persistencia: Proporcionada también por el AMR.
9. Protocolos de comunicación: Soporta diversos estándares como TCP/IP, FTP o POP3.

La arquitectura de JATLite está organizada como una jerarquía de capas especializadas y definidas, lo que facilita la labor a la hora de modificar cualquier aspecto de un sistema. También permite alterar cualquiera de estas capas sin necesidad de modificar el resto. Las capas son:

1. Abtract Layer.- Proporciona una colección de clases abstractas necesarias para la implementación de JATLite.
2. Base Layer.- Proporciona la comunicación básica sobre los protocolos en TCP/IP.

3. KQML Layer.- Permite el almacenamiento y análisis de los mensajes KQML.

4. Router Layer.- Proporciona nombres a los agentes, almacenamiento y direccionamiento para los mensajes. Cuando un agente se desconecta ya sea intencionalmente o por una caída en el sistema, el Router almacena los mensajes entrantes dirigidos a él hasta que el agente se reconecta.

5. Protocol Layer.- Soporta los diversos servicios de Internet como SMTP, FTP, POP3 o HTTP.

2.2.4.5. Zeus

El proyecto Zeus lo ha llevado a cabo BT Labs. (British Telecommunications Laboratories, Laboratorio de Telecomunicaciones Británico) con el objetivo de construir un conjunto de herramientas (**Zeus Agent Building Toolkit**, Herramientas para la Construcción de Agentes Zeus) que facilitarán el desarrollo de nuevas aplicaciones multiagentes, tratando de reunir en Zeus los principios y componentes fundamentales de algunos de los sistemas multiagente ya existentes.

Zeus ha sido desarrollado en Java y está basado en el paradigma de la programación visual, así el proceso de creación y control de agentes está apoyado por un entorno gráfico de ventanas y menús para poder configurar las funcionalidades y características de éstos.

Las principales características que ofrece la herramienta de creación de agentes Zeus son:

1. Interfaz: Zeus implementa un interfaz para el desarrollo y otro para el control de los agentes durante su ejecución.

2. Movilidad: No se permite la movilidad de agentes en tiempo de ejecución.

3. Creación de agentes: Los agentes pueden ser creados en un "host" remoto o local.

4. Lenguaje de comunicación: Zeus utiliza el estándar de FIPA, el ACL para la comunicación entre sus agentes.

5. Lenguaje de programación: Está escrito íntegramente en Java.

6. Localización de agentes y recursos: Se proporciona un Directory Facilitator que realiza esta función.

Zeus está formada por un conjunto de componentes divididos en tres grupos funcionales o librerías:

1. El primer grupo es la librería de componentes. Es una colección de las clases que forman el bloque principal de los agentes. Los contenidos de esta librería comprenden desde la comunicación y coordinación hasta la ontología pasando por la planificación, la interfaz o las estructuras de los agentes.

2. El segundo grupo de componentes es el constructor de agentes Zeus, que es básicamente una herramienta para desarrollar agentes a alto nivel, es decir, una herramienta que oculta al el desarrollador las complejidades de la librería de componentes. Proporciona una serie de editores que permiten a los usuarios crear agentes especificando sus atributos visualmente, editores como el de ontología, el de tareas o el de coordinación. Este grupo constructor tiene dos características importantes:

   - Una metodología para la creación de agentes que guía al desarrollador a través del análisis y el diseño del sistema.
   - Un entorno visual para el desarrollo de agentes, que soporta la metodología mencionada.

3. Por último el tercer grupo es el de "agentes útiles". Estos agentes son los llamados Nameserver y Facilitator, que simplifican el descubrimiento de la ubicación de otros agentes y de nueva información, y el Visualiser que permite observar, analizar y depurar sociedades de agentes Zeus.

2.2.4.6. Comparación de Herramientas para el Desarrollo de Agentes

La Tabla 2.1 muestra una comparación de las características de JADE, JafMas, JATLite y Zeus de acuerdo a un estudio realizado [14] por la Universidad de Coruña en España sobre éstas y otras herramientas para el desarrollo de agentes.

Características	Herramientas			
	JADE	JafMas	JATLite	Zeus
Organización	Universidad de Parma	Universidad de Cincinnati	Universidad de Standford	British Telecom. Labs.
Descripción	Entorno multiagente	Entorno multiagente	Entorno multiagente	Entorno para la creación de agentes
Distribución	Freeware	Freeware	Freeware	Freeware
GUI para desarrollo	No	Si (con Jive)	Si	Si
Lenguaje de comunicación	ACL	KQML	KQML	ACL
Envío de Mensajes Multicast	Si	Si	No	No
Envío de mensajes remotos	Si	Si	Si	Si
Lenguaje de programación	Java	Java	Java	Java
Sistema Operativo	Todos	Todos	Todos	Windows y Solaris
Protocolos de Comunicación	TCP/IP	TCP/IP y UDP/IP	TCP/IP, FTP y SMTP	TCP/IP

Tabla 2.1: Comparación de herramientas para el desarrollo de agentes

Todas las herramientas de la Tabla 2.1 permiten la creación de agentes multiplataforma, ya que están desarrolladas para utilizarse con Java, esta característica era muy importante para el proyecto a desarrollarse debido a que los agentes tenían que ejecutarse bajo cualquier sistema operativo.

Otro rasgo importante en la elección de la herramienta de desarrollo de agentes era el lenguaje de comunicación, se requería que fuera ACL pues es una mejora de KQML y solo JADE y Zeus utilizan el lenguaje de comunicación ACL. Sin embargo Zeus no permite el desarrollo de multiagentes sino de agentes autónomos.

La herramienta de desarrollo de agentes seleccionada fue JADE, puesto que permite la creación de multiagentes multiplataforma que utilizan ACL como lenguaje de comunicación y que además cuenta con una interfaz de monitoreo y control de los multiagentes, por lo que se consideró la herramienta más adecuada para este proyecto.

Capítulo 3

Diseño de Multiagentes para Minería de Datos

En este capítulo se muestra el diseño del proyecto desarrollado, presentando la arquitectura de comunicación y de red.

3.1. Objetivo

Diseñar e implementar multiagentes distribuidos que utilizan algoritmos de *Minería de Datos Descriptiva* para encontrar información útil en red y que son programables vía web para permitir la visualización de resultados y la reconfiguración de los parámetros de minería de datos a través de Internet o cualquier dispositivo móvil (como celulares y asistentes personales), de tal manera que el proceso de minería de datos sea más automático, flexible y rápido que el proceso tradicional.

3.2. Aplicación

Se optó por un análisis de canasta de compras para probar las ventajas de la arquitectura propuesta, ya que el análisis de canasta se aplica en tiendas comerciales donde la base de datos se encuentra distribuida, por lo que puede efectuarse este análisis sobre cada fragmento

y posteriormente unir los resultados obtenidos.

El *análisis de canasta* de compras toma su nombre del análisis que se hace de los artículos que un comprador lleva en el momento de la compra en una tienda comercial, en busca de afinidades para mejorar las ofertas o también para desarrollar las promociones adecuadas [6].

Sin embargo, este análisis se puede realizar en cualquier situación en donde se requiera identificar el tipo de productos o servicios que una persona consume o utiliza. A partir de la identificación de las afinidades se pueden identificar las oportunidades para venta cruzada (venta de artículos de manera conjunta), o para elevar el valor promedio de compra.

Para un negocio comercial, los beneficios de un análisis de canasta son:

- Conocimiento del patrón de consumo de sus clientes
- Identificar oportunidades para la venta cruzada
- Desarrollo de ofertas más atractivas
- Ubicación de los productos dentro de la tienda
- Disminución en los costos de inventarios

Existen varios algoritmos para la realización de minería de datos descriptiva [6] y que fueron descritos en el Capítulo II, de los cuales se seleccionó el algoritmo de asociación de reglas [1, 6], debido a que se adapta mejor al análisis de canasta según [6] que los otros algoritmos (que forman grupos o clases) por la facilidad de encontrar en los datos reglas para identificar los patrones de comportamiento de las compras de los clientes. Este algoritmo se detalla en el siguiente Capítulo.

3.3. Arquitectura de Comunicación

La arquitectura de comunicación se refiere al ambiente de trabajo en el cual se desempeñarán los multiagentes del sistema.

Los agentes son capaces de realizar el proceso de minería de manera automática, es decir, arrancar el proceso a determinado tiempo de acuerdo con parámetros establecidos (que pueden ser modificados vía web). El ambiente de trabajo entre los agentes se especifica en la Figura 3.1 como una red de computadoras con n sitios sobre las cuales se encuentran bases de datos grandes con una estructura fija y son dinámicas en el sentido de que su contenido va variando en tiempo real (debido a la utilización propia de la base de datos como inserciones, eliminaciones y modificaciones), existirán n agentes de procesamiento que se encarguen de analizar la información minada de cada sitio y que es obtenida por un agente de monitoreo capaz de detectar cambios en la base de datos y realizar la extracción modificación de éstos.

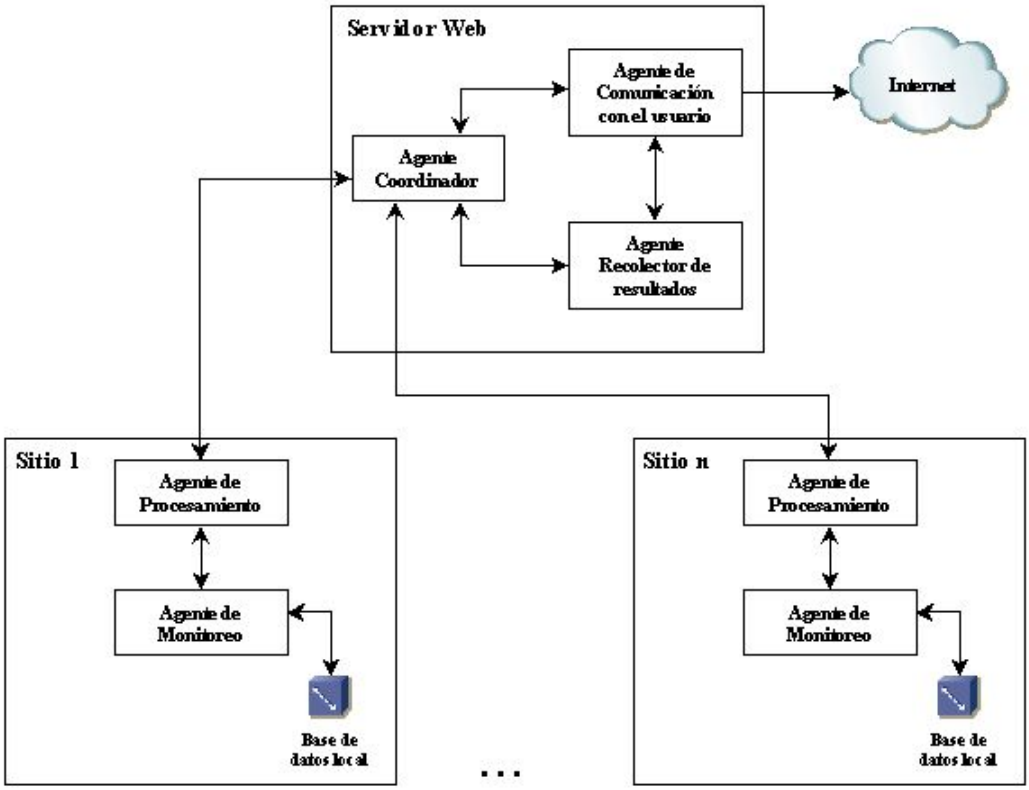

Figura 3.1: Arquitectura de Comunicación

Cada agente de procesamiento debe comunicar sus resultados parciales al agente coordinador el cual los pasa al agente recolector de resultados para que procese el resultado final de minería. El resultado final es enviado al agente de comunicación con el usuario, el cual se encarga de la visualización de los resultados y permite al usuario especificar los parámetros de minería

que desee establecer para un nuevo análisis; estos nuevos parámetros son enviados al agente coordinador para que defina un nuevo proceso de minería de los agentes de procesamiento en los n sitios.

La coordinación del sistema completo está a cargo de el agente coordinador el cual coordina todo el proceso de minería de datos y la comunicación entre los diversos agentes. Al dividir el trabajo entre los agentes existentes en cada sitio el proceso de minería de datos se agiliza. En resumen, las funciones que cada agente debe realizar se ilustran en la Tabla 3.1.

Agente	Funciones
Agente de monitoreo	Verifica constantemente si hubo cambios significativos en la base de datos local del sitio para reportarlos al agente de procesamiento.
Agente de procesamiento	Con los datos obtenidos de la base de datos, aplica un proceso de minería de datos descriptiva para generar un resultado, el cual es enviado al agente recolector de resultados.
Agente recolector de resultados	Recolecta los resultados de los n sitios para generar un resultado general, el cual es enviado al agente de comunicación según lo solicite éste. Para recibir todos los resultados, el agente recolector solicita al agente coordinador que establezca la comunicación con los n sitios y le envíe los resultados.
Agente de comunicación con el usuario	Establece la comunicación con el usuario vía web para enviarle a éste los resultados del proceso de minería de datos descriptiva, además de permitirle especificar los nuevos parámetros de minería de datos para realizar un nuevo análisis, los nuevos parámetros son enviados al agente coordinador para que este los comunique a los agentes de procesamiento de cada sitio.
Agente coordinador	Permite la comunicación entre los diferentes agentes del sistema, por lo que es el responsable de comunicar a los n sitios los cambios en los parámetros del proceso de minería de datos y de coordinar la recopilación de resultados de cada sitio.

Tabla 3.1: Funciones generales de los distintos agentes del sistema

3.4. Arquitectura de Red

La arquitectura de red se refiere al entorno en que se desempeña el sistema, para este proyecto, lo más importante es la interacción del usuario con el sistema.

Obviamente, se requiere una interfaz para la comunicación usuario-agentes que permita programar a los agentes de acuerdo a las necesidades concretas del usuario: especificar los parámetros de minería de datos, patrones que deseen encontrarse, visualización del conocimiento descubierto, entre otras cosas.

La importancia de esta interfaz de comunicación es que permita al usuario comunicarse con el agente principal desde cualquier parte donde éste se encuentre, puesto que la minería de datos se enfoca a usuarios gerenciales que comúnmente están en movimiento administrando un conjunto de empresas, por lo que el software de minería debe permitirle la configuración de parámetros y puesta en marcha del proceso, así como la visualización de resultados desde cualquier parte en que se encuentre localizado el usuario gerencial.

Por lo tanto, la interfaz se debía desarrollar en HTML (HyperText Markup Language, Lenguaje de Marcas de Hipertexto) para la página web y en WML (Wireless Markup Language, Lenguaje Inalámbrico de Marcas) para los dispositivos móviles, de forma que el agente coordinador puede ser programado dinámicamente y mostrar resultados a través de páginas web y dispositivos móviles como celulares y asistentes personales (PDA). La Figura 3.2 muestra esta arquitectura de comunicación con el agente coordinador.

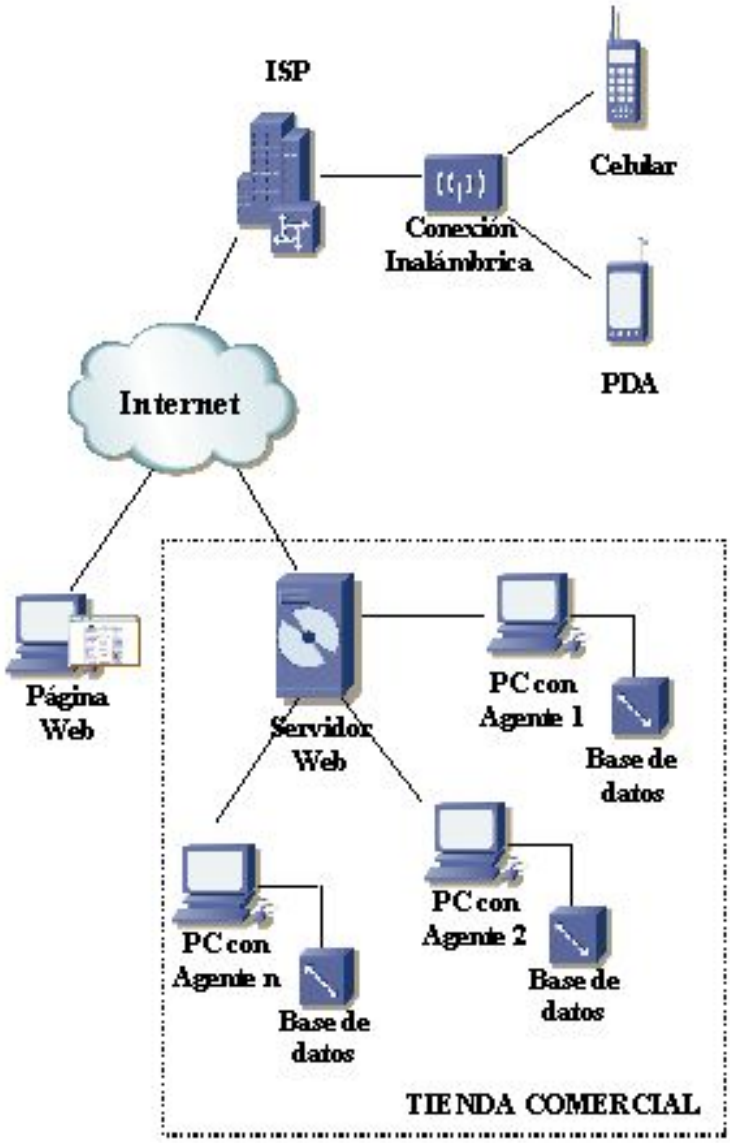

Figura 3.2: Arquitectura de Red

Capítulo 4

Implementación de Multiagentes para Minería de Datos

En este capítulo se muestra todo lo concerniente al desarrollo de los multiagentes distribuidos: se describe el algoritmo de Asociación de Reglas para la implementación de minería de datos descriptiva, el desarrollo de los multiagentes con JADE y finalmente la interfaz del usuario que comprende la realización de páginas web en HTML y WML para dispositivos móviles.

4.1. Algoritmo de Minería de Datos Descriptiva

La aplicación implementada es un *análisis de canasta* de compras, que es un análisis de los artículos que compra en conjunto un cliente en una tienda comercial, para determinar patrones en el consumo de los artículos y así desarrollar estrategias de mercado adecuadas [6]. Este tipo de análisis entra en la categoría de minería de datos descriptiva, pues el objetivo es encontrar patrones que describan el comportamiento de las compras de los clientes.

Existen varios algoritmos para la realización de minería de datos descriptiva [6], de los cuales se seleccionó e implementó el algoritmo de asociación de reglas [1, 6], debido a que se adapta mejor al análisis de canasta según [6] que los otros algoritmos (que forman grupos o clases) por la facilidad de encontrar en los datos reglas para identificar los patrones de comportamiento de las compras de los clientes. Por ejemplo, una regla de asociación [6] Articulo_X $\Rightarrow$ Articulo_Y [30 % confianza, 2 % soporte] indica que "el 30 % de las transacciones que contienen el artículo

X también contienen el artículo Y; el 2 % de todas las transacciones contienen ambos artículos".

4.1.1. Algoritmo de Asociación de Reglas

Este algoritmo genera **reglas de asociación**, que son una forma de representar la relación de un conjunto de artículos junto con otro conjunto de artículos; esta relación tiene dos parámetros que son confianza y soporte. Una regla de asociación [6] tiene la siguiente forma Articulo_$X \Rightarrow$ Articulo_Y [% confianza, % soporte] y representa que "el %confianza de las transacciones que contienen el artículo X también contienen el artículo Y; el %soporte de todas las transacciones contienen ambos artículos".

Para cada **regla de asociación** existen dos conceptos conocidos como el *soporte* y la *confianza*. El soporte se define como la probabilidad de que un registro satisfaga tanto a X como a Y, y esta dado por la Ecuación 4.1. La confianza se define en la Ecuación 4.2 como la probabilidad de que un registro satisfaga a Y dado que satisface a X. Para el ejemplo anterior, 30 % es el nivel de confianza, y el 2 % es el soporte de la regla. Por lo tanto, la regla $X \Rightarrow Y$ tiene el soporte s en el conjunto de transacciones D si el s % de las transacciones en D contienen $X \cup Y$, y la regla $X \Rightarrow Y$ en el conjunto de transacciones D tiene confianza c si el c % de las transacciones en D que contienen X también contienen Y.

$$Soporte = \frac{total\_tickets\_bd * min\_soporte}{100} \tag{4.1}$$

$$confianza = \frac{Soporte(l)}{Soporte(s)} * 100 \tag{4.2}$$

Para la implementación del la asociación de reglas se utilizó el algoritmo Apriori, el cual encuentra y/o se basa en un conjunto de artículos descubiertos de forma frecuente en la base de datos. El seudocódigo de la Función Apriori [6] se muestra en la Figura 4.1.

Donde D es la base de datos del análisis y *min_soporte* es el umbral mínimo de soporte que debe cumplir un conjunto frecuente (CF). Un *conjunto frecuente* es un conjunto de elementos (en este caso son identificadores de artículos) que cumplen con el mínimo soporte establecido

$Datos\,de\,Entrada : D,\ min\_soporte$
$Proceso :$
(1) $L_1 = \text{Encontrar\_1} - esimoCF(D);$
(2) for $(k = 2; L_{k-1} \neq \emptyset; k++)\{$
(3) $C_k = \text{Generar\_Apriori}(L_{k-1}, min\_soporte);$
(4) for each Transaccion $t \in D\,\{$
(5) $C_t = Subconjunto(C_k, t);$
(6) for each Candidato $c \in C_t$
(7) $c.count++;$
(8) $\}$
(9) $L_k = \{\, c \in C_k \,|\, c.count \geq min\_soporte\}$
(10) $\}$
(11) return $L = U_k L_k;$
$Datos\,de\,Salida : L$

Figura 4.1: Función Apriori para la generación de reglas

en el análisis, por lo que se considera que s un conjunto que aparece frecuentemente en la base de datos.

El algoritmo realiza una búsqueda de los conjuntos de datos de tamaño k que cumplen con el mínimo soporte establecido y va descartando los conjuntos que no lo cumplen. El valor de k siempre inicia en 1 (por lo tanto al principio del algoritmo los conjuntos son de tamaño 1, es decir, contienen un elemento) y se incrementa en cada iteración para encontrar conjuntos que cumplan con el soporte, de forma que se van obteniendo conjuntos con un número de elementos k.

Cuando tiene todos los conjuntos frecuentes tamaño k que cumplen con el soporte, genera nuevos conjuntos con ellos pero de tamaño $k+1$ (esto se realiza en la función *Generar_Apriori* mostrada en la Figura 4.2 para nuevamente buscar si cumplen o no con el mínimo soporte.

Al finalizar la ejecución del algoritmo, se obtiene L que es una lista que contiene los conjuntos frecuentes que cumplieron con el mínimo soporte especificado. A partir de esta lista se pueden generar las reglas de asociación.

La función *Generar_Apriori* utiliza la función *Descartado_CF* mostrada en la Figura 4.3 para

Funcion Generar_Apriori(L_{k-1}: (k − 1) − $esimoCF$; $min\_soporte$: minimo soporte)
(1) for each Conjunto $l_1 \in L_{k-1}$
(2) for each Conjunto $l_2 \in L_{k-1}$
(3) if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \ldots \wedge (l_1[k-1] = l_2[k-1])$ then{
(4) $c = l_1 \bowtie l_2$
(5) if Descartado_CF(c, L_{k-1}) then
(6) $borrar\ c$;
(7) else $agregar\ c$ a C_k;
(8) }
(9) return C_k;

Figura 4.2: Función Generar_Apriori

buscar si un subconjunto del conjunto c fue descartado anteriormente y por consecuencia no es un conjunto frecuente pues no cumplirá el soporte mínimo, a esto se le denomina **Propiedad Apriori**.

Funcion Descartado_CF(c: k − esimoCandidato,L_{k-1}: (k − 1) − esimoCF)
(1) for each $(k-1)$ − Subconjunto s de c
(2) if $s \notin L_{k-1}$ then
(3) return $TRUE$;
(4) return $FALSE$;

Figura 4.3: Función Descartado_CF

4.1.2. Generación de Reglas de Asociación

Para la generación de reglas de asociación se toman los conjuntos de L. Para cada conjunto l se generan todos sus subconjuntos no vacíos posibles, de manera que cada subconjunto s genera una regla de la forma $s \Rightarrow (l - s)$ mostrada en la Ecuación 4.3.

$$s \Rightarrow (l - s) \quad if \ \frac{soporte(l)}{soporte(s)} \ \geq min\_confianza \tag{4.3}$$

Para que la regla sea válida debe cumplir con el umbral mínimo de confianza establecido.

4.2. Multiagentes Desarrollados con JADE

En el Apéndice A se detallan los pasos para la instalación de JADE.

4.2.1. Ambiente de Trabajo de los Agentes

En la Figura 4.4 se muestra el ambiente de los agentes desarrollados con JADE, existe una plataforma que se compone de un conjunto de contenedores (instancias de JADE), y de un contenedor principal especial denominado Main container, que se activa al crear la plataforma y al cual todos los demás contenedores (ubicados en diferentes máquinas) se registran tan pronto como son creados.

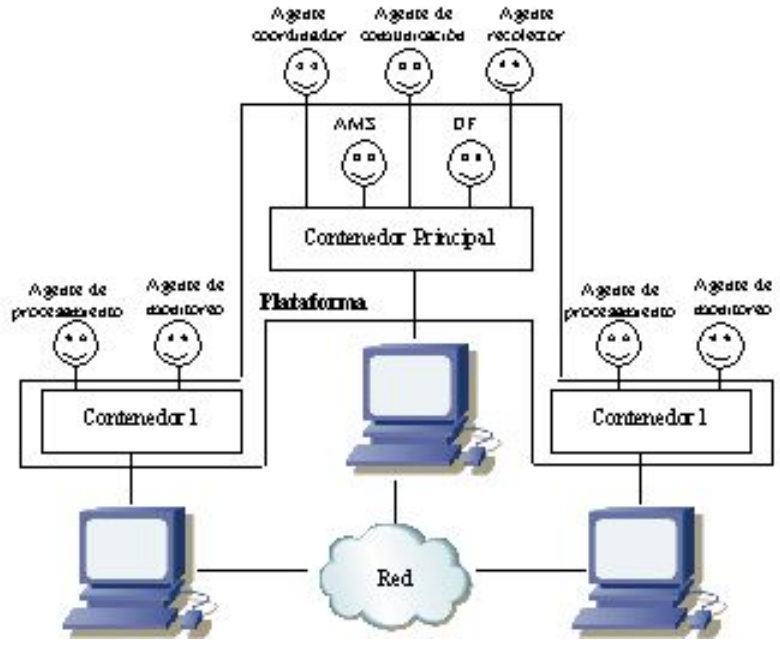

Figura 4.4: Ambiente de los agentes desarrollados con JADE

En el proyecto desarrollado, en el contenedor principal está el agente coordinador, el agente de

comunicación con el usuario y el agente recolector de resultados. En cada contenedor secundario se encuentra un agente de Procesamiento y un agente de monitoreo de la base de datos. Esta distribución de los agentes también puede apreciarse en la Figura 3.1.

Dentro de cada contenedor existen varios agentes, de tal forma que la plataforma de agentes puede estar distribuida entre distintas máquinas (cada una con un contenedor como mínimo) con agentes trabajando entre los contenedores. Es importante la relación entre los contenedores debido a que JADE permite la movilidad y clonación de agentes de un contenedor a otro. Para el soporte de comunicación entre los agentes, JADE implementa una estructura interna [3] que sigue el estándar FIPA [4].

4.2.2. Interacción entre los Agentes

El proceso se inicia ejecutando el *Agente Coordinador* en la máquina donde se encuentra el contenedor principal y las tablas de apoyo del sistema de minería (ver Apéndice C). El *Agente Coordinador* crea en tiempo de ejecución y sin intervención del usuario dos agentes: el *Agente Comunicación* y el *Agente Recolector*, tal como se muestra en la Figura 4.5. Después de crear los dos agentes anteriores queda en un estado de bloqueo, del cual sale sólo cuando llega un mensaje.

El *Agente Comunicación* se encarga de verificar si se han guardado en la tabla **minería** un nuevo proceso, si es así comunica los parámetros del nuevo proceso al *Agente Coordinador* a través de un mensaje **AcMinería**, como se muestra en la Figura 4.6. Cuando el Agente Coordinador recibe este mensaje crea un *Agente_Minero_X* donde X es el **idpm** (identificador del proceso de minería).

El *Agente_Minero_X* se encarga de clonarse a sí mismo en todos los sitios disponibles en la red, una vez terminada su función se elimina a sí mismo.

El *Agente Coordinador* también puede recibir otro tipo de mensajes como se muestra en la Figura 4.6, el mensaje **ClonConfirma**, que es enviado por los sitios donde se clonó el *Agente_Minero_X*, sirve para verificar que la clonación se realizó satisfactoriamente.

En la Figura 4.5 se muestra la interacción (a través de las flechas punteadas) del *Agente Coordinador* con el *Agente Recolector*, el cual verifica constantemente si el tiempo de los procesos

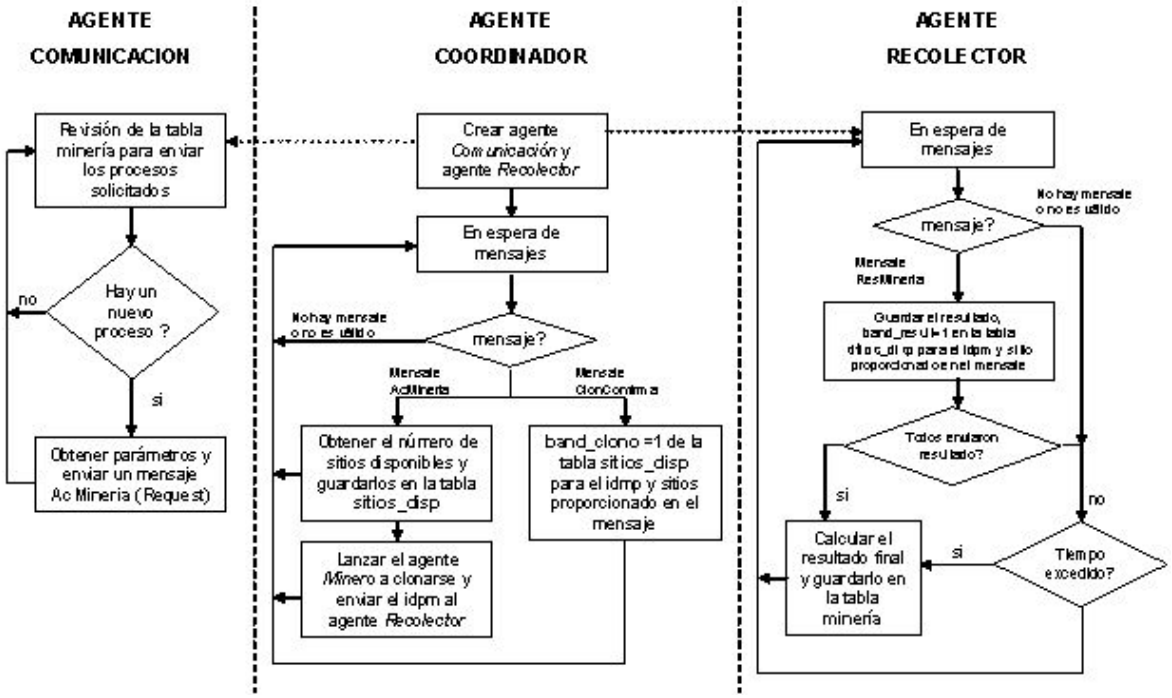

Figura 4.5: Interacción entre el Agente Coordinador y el Agente Recolector

activos de minería se ha excedido en 30 minutos, de ser así calcula el resultado del proceso con los resultados parciales de los sitios que si contestaron en tiempo. También verifica si llega un mensaje **ResMineria** de algún sitio, de ser así guarda este resultado parcial en la tabla **sitios_disp** y verifica si todos los sitios involucrados en el análisis también enviaron sus resultados. Si todos los sitios enviaron sus resultados calcula el resultado final y lo almacena en la tabla **minería**.

Las funciones del Agente Minero y su relación con el Agente Monitor se muestran en la Figura 4.7.

En cada sitio está un *Agente Minero* que fue clonado para la realización de un análisis con parámetros específicos. Cada que se requiere un nuevo análisis se crea un nuevo *Agente Minero*. La primera que realiza este agente es clonarse en todos los sitios disponibles para luego eliminarse a si mismo. Después de clonarse, cada clon debe crear su propio *Agente Monitor* que revisará si hay cambios significativos en el fragmento de la base de datos del análisis que puedan afectar el resultado, de ser así el *Agente Monitor* envía un mensaje de alerta para avisar al *Agente Minero*

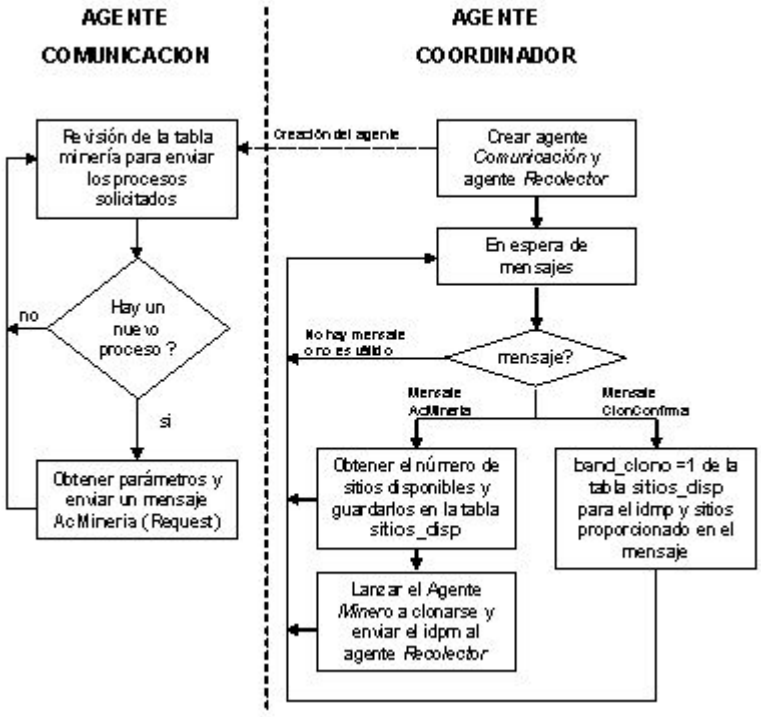

Figura 4.6: Interacción entre el Agente Comunicación y el Agente Coordinador

que debe reiniciar el análisis.

La siguiente función a realizar por cada clon es la **Asociación de Reglas** (algoritmo de minería descriptiva) en cada sitio y de enviar su resultado parcial en un mensaje **ResMineria** al *Agente Coordinador* y entonces eliminarse a sí mismo. Durante la realización de la función de **Asociación de Reglas** el *Agente Minero* revisa si ha llegado un mensaje del Agente Monitor para reiniciar el análisis.

Como se observa, los agentes interactúan a través del envió de mensajes que siguen una ontología específica, la cual se describe en la Sección Y.

4.2.3. Estructura Básica de un Agente

Se tiene una clase `Agent` que proporciona la estructura básica del agente, la funcionalidad éste se define en comportamientos llamados `behaviours`. La estructura básica de un agente se define:

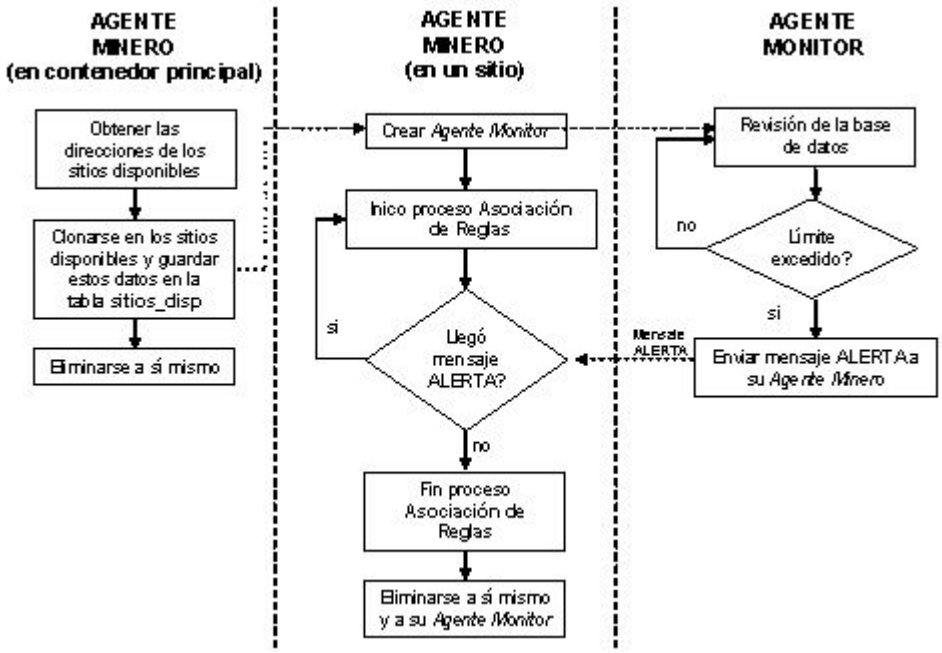

Figura 4.7: Interacción entre el Agente Minero y su Agente Monitor

```
public class MiAgente extends Agent{
  public void setup(){
    //Instrucciones de inicio
  }
  public void takeDown(){
    //Instrucciones de finalización
  }
}
```

En la función `setup()` se definen las funciones iniciales del agente y los comportamientos con los que inicia. Mientras que la función `takeDown()` se utiliza para realizar las tareas de finalización como por ejemplo cerrar una base de datos.

4.2.4. Comportamientos de los Agentes

Existen varios tipos de comportamientos simples o compuestos [3] en JADE que sirven como base para implementar tus propios comportamientos. A continuación se muestran algunos de los comportamientos utilizados en este proyecto:

1. `SimpleBehaviour`.- Es un comportamiento simple en el cual el usuario debe definir dentro del método `onEnd()` la condiciones sobre las cuales el comportamiento debe terminar.
2. `OneShotBehaviour`.- Es un comportamiento simple que se ejecuta sólo una vez y no puede ser bloqueado. Por lo tanto el método `done()` siempre retorna verdadero. La función `done()` retorna verdadero o falso para definir si el comportamiento finalizó o no.
3. `CyclicBehaviour`.- Este comportamiento simple puede ser ejecutado infinitamente, por lo que el método `done()` siempre retorna falso.
4. `FSMBehaviour`.- Es un comportamiento compuesto que se ejecuta de acuerdo a una máquina de estado finito cuyos estados son definidos por el usuario.
5. `SequentialBehaviour`.- Este comportamiento compuesto ejecuta secuencialmente varios subcomportamientos y termina cuando todos los subcomportamientos hayan terminado.
6. `ParallelBehaviour`.- Es un comportamiento compuesto que ejecuta subcomportamientos concurrentemente y termina cuando una condición particular en estos subcomportamientos se cumple.

Cada tipo de comportamiento se diferencia en los elementos que lo componen y de la forma como se ejecuta, termina, activa, etc.

Para la realización del *Agente Minero* se elaboró un comportamiento llamado **Asociaciónde-Reglas** extendido de la clase `Behaviour` que además de realizar el proceso de minería de datos también crea un *Agente Monitor* para monitorear los cambios en la base de datos, en los Apéndices B.4 y B.5 se muestra el código fuente de estos agentes. Para iniciar el comportamiento se utiliza la función `addBehaviour`:

```
public class Mineria extends Agent{
  public void setup(){
    ...
    addBehaviour(
        new AsociaciondeReglas(argumentos)
    );
    ...
  }
  ...
}
```

La definición del comportamiento se realizó de la siguiente manera:

```
class AsociaciondeReglas extends Behaviour {
  public void action(){
    //Instrucciones para minería de datos
  }
  ...
}
```

Se implementó un comportamiento `SequentialBehaviour` en el agente de procesamiento para obtener el nombre de los contenedores activos y enviarles un agente clonado con los parámetros de minería a todas las máquinas conectadas. Para hacer esto, utiliza un `SequentialBehaviour` que se compone de varios subcomportamientos que se ejecutan secuencialmente. Cada subcomportamiento realiza la tarea de clonación del agente.

Para la elaboración del *Agente Coordinador* y del *Agente Recolector* de resultados se utilizó un comportamiento `CyclicBehaviour` pues este comportamiento se repite indefinidamente; en este caso, se determinó que el agente coordinador revise constantemente su cola de mensajes en espera de que llegue una solicitud de un nuevo proceso de minería de datos y el agente recolector que espere la llegada de los resultados de los agentes de procesamiento. El siguiente fragmento de código muestra el uso del comportamiento `CyclicBehaviour` para el Agente Coodinador:

```
addBehaviour(new CyclicBehaviour(this) {
  public void action() {
    //Escucha si llega un mensaje INFORM
     ACLMessage msg =receive (MessageTemplate.
     MatchPerformative(ACLMessage.INFORM));
     if (msg != null) {
      ContentElement p=manager.extractContent(msg);
      ...
     }
  }
  });
  ...
}
```

En los Apéndices B.1 y B.3 se muestra el código fuente del *Agente Coordinador* y del *Agente Recolector* respectivamente.

El *Agente Monitor*, cuya implementación se muestra en el Apéndice B.5, requería que cada cierto tiempo se revisara el tamaño la base de datos local, por lo tanto el comportamiento utilizado fue `TickerBehaviour`. En el ejemplo siguiente cada 2 segundos se ejecuta la función `onTick()`.

```
public class AgenteMonitor extends Agent {
  ...
  addBehaviour(new TickerBehaviour(this,2000) {
    public void onTick() {
            //Instrucciones de monitoreo
//de la base de datos
    }
  });
  ...
}
```

4.2.5. Comunicación entre Agentes

El envío y recepción de mensajes entre los agentes es transparente pues JADE se encarga de elegir la vía más adecuada para transportar ese mensaje, así, si los agentes emisor y receptor están ejecutándose sobre la misma máquina virtual de Java, utilizará los eventos Java, si están en distintas máquinas virtuales, RMI y si están en distintas plataformas, IIOP o HTTP [3].

Los agentes elaborados envían y reciben mensajes utilizando los métodos predefinidos [3]. Por ejemplo, el agente de monitoreo envía un mensaje al agente de procesamiento cuyo contenido es la palabra ALERTA para informarle que hubo cambios significativos en el tamaño de la base de datos, el mensaje se envía de la siguiente manera:

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.setContent(ALERTA);
msg.addReceiver(new AID(agproces, AID.ISLOCALNAME));
myAgent.send(msg);
```

Puede observarse que se define un mensaje del tipo `ACLMessage`, se define el contenido con la función `setContent`, se especifica el agente que va a recibir el mensaje con la función `addReceiver` y finalmente con `send` se envía el mensaje.

4.2.6. Clonación del Agente de Procesamiento

JADE provee un conjunto de funciones para mover o clonar un agente [3]. Para el proyecto se requería que el agente de procesamiento se clonara en todas las máquinas de la plataforma, para que inicie en cada una de ellas su proceso de minería de datos. Se utilizó la función `doClone()` cuyos parámetros son el nombre del contenedor y el nombre del clon. Un agente clonado puede definir tareas antes y después de clonarse con las funciones `beforeClone()` y `afterClone()` respectivamente.

```
public class Mineria extends Agent{
  public void setup(){
```

```
    addBehaviour(new ObtenerLocalidades());
  }
  protected void beforeClone() { }
  protected void afterClone() { }
  ...
}
```

Donde **ObtenerLocalidades** se desarrolló como sigue:

```
SequentialBehaviour sb = new SequentialBehaviour();
for (contador=1; sitios.hasNext(); contador++){
  sb. addSubBehaviour(new OneShotBehaviour(myAgent){
        public void action() {
        ...
         myAgent.doClone(contenedor,nombreclon);
        ...
        }
      });
}
```

4.2.7. Creación de Agentes en Tiempo de Ejecución

Cuando se crea al *Agente Coordinador* este crea sin intervención del usuario y en tiempo de ejecución a dos agentes: el *Agente Recolector* de resultados y al *Agente de Comunicación* con el usuario, esto lo logra utilizando las funciones de `jade.core.Runtime`.

Por ejemplo, para crear al Agente Recolector:

```
Runtime rt = Runtime.instance();
AgentContainer ac = getContainerController();
AgentController agente1=
ac.createNewAgent("AgenteRecolector","minominero.AgenteRecolector",args);
Agente1.start();
```

Se utiliza la función `getContainerController` para definir el contenedor donde va a residir el agente que se desea crear. La función `createNewAgent` se utiliza para especificar el nombre del agente a crearse, la ruta donde se encuentra la clase y sus argumentos (en caso de tenerlos) para finalmente activar al agente con la función `start`.

4.2.8. Monitoreo de los Agentes

Para monitorear este proceso se utilizó la herramienta de administración RMA mostrada en la Figura 4.8, donde se observa gráficamente que al crear al Agente Coordinador este crea automáticamente y sin intervención del usuario dos agentes: el Agente Recolector y el Agente de Comunicación con el usuario y queda en espera de solicitudes de minería de datos.

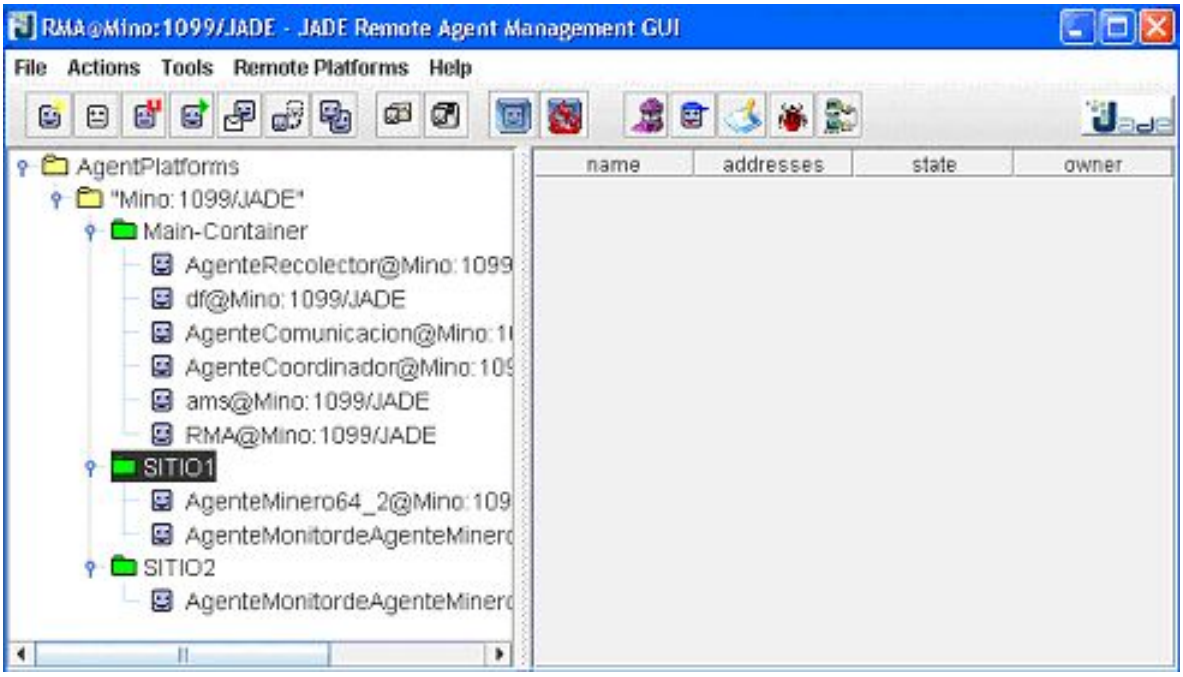

Figura 4.8: Herramienta RMA durante la ejecución del proceso número 64

Al recibir una solicitud el agente de procesamiento (llamado en este caso ***AgenteCoordinador***) crea un agente llamado ***AgenteMineroX***, donde X es el identificador del proceso de minería (AgenteMinero64). El AgenteMinero64 es clonado en todos las máquinas disponibles (sitio1, sitio2, etc.) con el nombre de ***AgenteMineroX_Y***, donde Y es el número del clon, y este a su vez

crea a su propio agente monitor de la base de datos (***AgenteMonitordeAgenteMineroX_Y***). Una vez terminado el proceso el agente AgenteMinero64 se elimina a él mismo y a su agente monitor.

4.3. Interfaz con el Usuario Vía Web

Para la configuración del proceso de minería de datos se desarrolló una interfaz de comunicación vía web con dirección `http://minominero.no-ip.info` y para dispositivos móviles con dirección `http://minominero.no-ip.info\wml`. Se programó en HTML y en WML (para los dispositivos móviles como teléfonos celulares y asistentes personales), de tal forma que el usuario puede:

1. Iniciar un nuevo proceso de minería de datos, especificando los parámetros, artículos de interés y la antigüedad de los datos a analizar.
2. Obtener el resultado del proceso iniciado, así como los sitios involucrados.
3. Obtener los resultados parciales de los n sitios involucrados en el análisis.
4. Monitorear que procesos ya terminaron o están en ejecución.
5. Revisar los parámetros de minería de un proceso.

La interfaz web con el usuario establece una comunicación con los multiagentes del proyecto de la siguiente manera:

1. Cada que un nuevo proceso de minería de datos es configurado, la página utiliza PHP con Mysql para insertar un registro en la tabla minería.
2. El Agente de Comunicación revisa constantemente la tabla minería para encontrar nuevos procesos y de ser así, enviar un mensaje al Agente Coordinador sobre el nuevo proceso de minería de datos.
3. El Agente Coordinador toma de la tabla minería la configuración del proceso y crea el Agente Minero correspondiente.

4. Una vez que el Agente Recolector tiene un resultado de un proceso almacena este resultado en la tabla minería, desde la cual la página web puede obtener los datos y mostrar las reglas al usuario.

Para el desarrollo de la interfaz web se requirió la instalación de PHP y de Apache. La habilitación de la dirección `http://minominero.no-ip.info` se realizó a través del sitio `no-ip.com`.

Tanto la página web en HTML como la página web en WML tienen la misma funcionalidad y opciones. En la Figura 4.9 puede apreciarse el menú principal de cada página web.

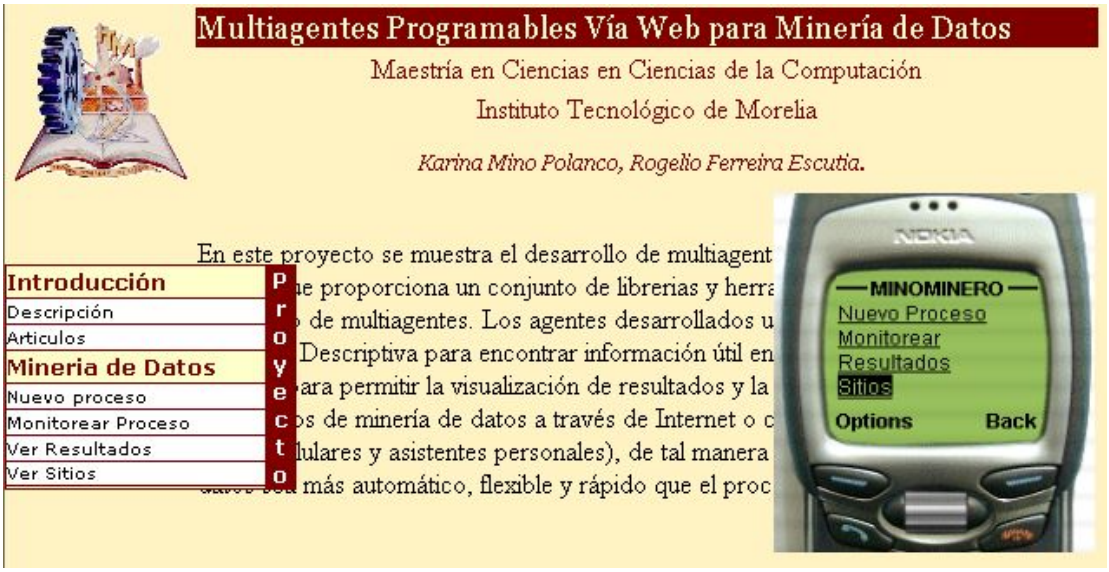

Figura 4.9: Opciones de las páginas web en HTML y en WML

En la Figura 4.10 se muestra la página web accesada desde una computadora personal y desde un teléfono celular en el menú Resultados, donde se muestra el número de identificación del proceso, su estado actual y el nombre y el número de computadoras (hosts) involucradas en el análisis de minería de datos.

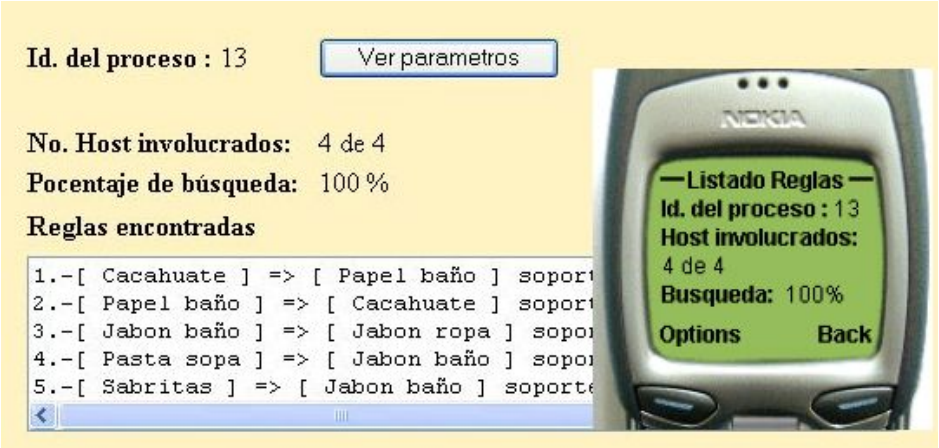

Figura 4.10: Página web para visualizar el resultado de un proceso

Capítulo 5

Pruebas y Resultados

En este Capítulo se muestran las consideraciones iniciales sobre las que se hicieron las pruebas. Se definieron tres casos de estudio, cada uno de ellos tiene establecido un escenario diferente, es decir, un ambiente diferente en el que se desempeña el sistema multiagente desarrollado.

5.1. Consideraciones Iniciales

Para la realización de las pruebas se definieron las siguientes consideraciones de acuerdo a las observaciones realizadas en distintos centros comerciales de Morelia:

- Cada ticket consta de entre diez y cuarenta artículos.
- Se determinó que en una caja entran en promedio seis tickets en diez minutos, lo que nos da un total de 500 tickets por caja al día, ya que una tienda comercial da servicio 13 horas.
- Cada agente debe calcular el soporte de su fragmento de la base de datos de acuerdo a la Ecuación 5.1. Donde *total_tickets_bd* es el número de tickets del fragmento de la base de datos local que se desea analizar y *min_soporte* es el porcentaje que define la porción mínima de apariciones de un conjunto frecuente en el fragmento analizado de la base de datos local; por lo tanto se obtiene como soporte el número mínimo de registros donde debe aparecer un conjunto frecuente.
- Para que una regla sea válida debe calcularse su confianza conforme a la Ecuación 5.2

y que ésta sea mayor o igual al mínimo de confianza establecido por el usuario cuando se define el análisis de minería de datos. La confianza es un porcentaje que representa la porción en que es verdadera la regla de acuerdo al número de apariciones en la base de datos del conjunto l denominado $Soporte(l)$ entre el número de apariciones del conjunto frecuente s denominado $Soporte(s)$.

- Se desarrolló un programa que simulara la entrada de artículos a la base de datos de cada caja de la tienda comercial, de acuerdo a las consideraciones anteriores de cantidad de tickets y de artículos por ticket. Así, cada diez minutos introduce en la base de datos un número aleatorio de tickets (entre 5 y 10) y cada tickets consta de un número aleatorio de artículos (entre 5 y 45 artículos).

$$Soporte = \frac{total\_tickets\_bd * min\_soporte}{100} \tag{5.1}$$

$$confianza = \frac{Soporte(l)}{Soporte(s)} * 100 \ \geq \ min\_confianza \tag{5.2}$$

Para todos los análisis de minería de datos el usuario debe definir los siguientes parámetros:

1. Definir el porcentaje mínimo de soporte y confianza.
2. Especificar si el análisis debe efectuarse sobre todos los artículos, o sobre los artículos que pertenezcan a ciertos departamentos o sobre algunos artículos específicos.
3. Especificar la antigüedad de los datos del análisis de minería de datos, es decir, especificar un intervalo de fechas de los tickets de compras entre las cuales se buscarán los conjuntos frecuentes y por consiguiente las reglas encontradas representarán las relaciones entre los artículos en ese espacio de tiempo.

Una vez definidos los parámetros de minería de datos por el usuario se sigue el proceso siguiente:

1. Cada agente debe calcular el soporte de su fragmento de la base de datos de acuerdo a la Ecuación 5.1.

2. Se genera una lista de artículos $L(1)$ con los id_artículo de los artículos a relacionar, esta lista se forma de los artículos que cumplan las condiciones del usuario respecto a fecha, departamento, artículos específicos, etcétera.

3. En un proceso repetitivo que termina hasta encontrar los últimos conjuntos de artículos que cumplan con el soporte mínimo:

   a) Se obtiene el soporte de cada conjunto de la lista $L(n)$, donde n es el número de la lista (n inicia con valor 1).

   b) Se eliminan los conjuntos que no cumplen con el soporte

   c) Se genera la siguiente lista n mezclando los elementos de los conjuntos de la lista anterior, para formar una lista solo con conjuntos de tamaño n.

   d) Se revisa que un nuevo conjunto generado no tenga un subconjunto que hubiera sido eliminado anteriormente, de acuerdo a la **propiedad Apriori** [6] explicada en la Sección 4.1.1.

   e) Si una lista queda vacía (sin conjuntos) porque ningún conjunto cumplió con el soporte o por la propiedad Apriori se termina el ciclo y con la lista $L(n-1)$ se generan las reglas de asociación. También termina el ciclo si la lista $L(n)$ sólo contiene un único conjunto.

4. Una vez obtenida la lista final, se generan las reglas de asociación de la siguiente manera:

   a) Para cada conjunto l de la lista se generan todos los subconjuntos s posibles no vacíos.

   b) Cada subconjunto s genera una regla de la forma $s \Rightarrow (l-s)$ mostrada en la Ecuación 4.3.

5. Una vez obtenidas las reglas el agente las envía al *Agente Recolector* para que éste genere un resultado final y se almacene en la base de datos para que pueda ser consultado por el usuario posteriormente.

5.2. Caso 1. Base de datos Reducida

Para probar el algoritmo de Asociación de reglas se requería una base de datos pequeña, mostrada en las Tablas 5.1 y 5.2, que permitiera comprobar de forma manual los resultados. Por lo tanto para el primer caso se definió el siguiente escenario.

Escenario:
Número de tickets : 9.
Número de Artículos disponibles: 5.

Id_ticket	Lista de Id_artículos	Fecha del ticket
100	1, 2, 5	12/04/2005
200	2, 4	12/04/2005
300	2, 3	12/04/2005
400	1, 2, 4	27/05/2005
500	1, 3	27/05/2005
600	2, 3	27/05/2005
700	1, 3	27/05/2005
800	1, 2, 3, 5	27/05/2005
900	1, 2, 3	27/05/2005

Tabla 5.1: Tabla detalle_ticket del Escenario del Caso 1

Id_artículo	Nombre del artículo	Departamento
1	Leche	Lácteos
2	Papel	Abarrotes
3	Jabón Baño	Abarrotes
4	Jamón	Salchichonería
5	Detergente	Abarrotes

Tabla 5.2: Tabla artículos del Escenario del Caso 1

5.2.1. Análisis por Artículos

En un análisis por artículos el usuario es el que selecciona desde la interfaz web los artículos de los que desea encontrar reglas.

Configuración del Análisis.- Dado un soporte del 20 % (llamado *min_soporte*) y confianza del 50 % (llamada *min_confianza*) especificados por el usuario, y seleccionados los artículos Leche, Papel, Jabón Baño y Jamón (con Id_articulo 1,2,3,4 respectivamente) por el usuario. El proceso de minería de datos se muestra en la Figura 5.1.

L(1)=

Artículos	Soporte
{1}	6
{2}	7
{3}	6
{4}	2

L(2)=

Artículos	Soporte
{1,2}	4
{1,3}	4
{1,4}	1
{2,3}	4
{2,4}	2
{3,4}	0

L(3)=

Artículos	Soporte
{1,2,3}	2
{1,2,4}	{1,4}
{2,3,4}	{3,4}

Figura 5.1: Análisis de los artículos con id_artículo 1,2,3 y 4

Para iniciar el análisis se calculó el mínimo soporte para esta base de datos de acuerdo a la Ecuación 5.1, calculando $soporte = 9 * 20/100 = 1.8$ y obteniéndose al redondear que el soporte es 2.

Como se muestra en la Figura 5.1, la lista $L(1)$ se forma con los artículos seleccionados por el usuario que cumplen con el mínimo soporte. Al generar la lista $L(2)$ se eliminan los conjuntos {1,4} y {3,4} por no cumplir con el mínimo soporte. Con los conjuntos de la lista $L(2)$ se genera la lista $L(3)$, pero al generarla se eliminan los conjuntos {1,2,4} y {2,3,4} por que sus subconjuntos {1,4} y {3,4} respectivamente, fueron eliminados en un paso anterior, lo que se conoce como **propiedad Apriori** detallada en la Sección 4.1.1.

La lista $L(3)$ queda con un único conjunto {1,2,3} por lo que termina el proceso y se inicia la generación de reglas de asociación. Se toma el conjunto {1,2,3} y se generan todos los subconjuntos posibles no vacíos, que son {1}, {2}, {3}, {1,2}, {1,3} y {2,3}. De acuerdo a la Ecuación 4.3 se obtienen las siguientes reglas de asociación:

Regla 1. – $\{1,2\} \Rightarrow \{3\}\ confianza = 2/4 = 50\,\%$
Regla 2. – $\{1,3\} \Rightarrow \{2\}\ confianza = 2/4 = 50\,\%$
Regla 3. – $\{2,3\} \Rightarrow \{1\}\ confianza = 2/4 = 50\,\%$
Regla 4. – $\{1\} \Rightarrow \{2,3\}\ confianza = 2/6 = 33.33\,\%$
Regla 5. – $\{2\} \Rightarrow \{1,3\}\ confianza = 2/7 = 28.57\,\%$
Regla 6. – $\{3\} \Rightarrow \{1,2\}\ confianza = 2/6 = 33.33\,\%$

Solo las reglas 1, 2 y 3 son válidas para este análisis porque cumplen el mínimo de confianza especificado por el usuario. Estas reglas se almacenan en la base de datos pero se muestran al usuario con el nombre del artículo en lugar de su id_artículo.

Regla 1. – $\{Leche, Papel\} \Rightarrow \{JabonBano\}\ confianza = 2/4 = 50\,\%$
Regla 2. – $\{Leche, JabonBano\} \Rightarrow \{Papel\}\ confianza = 2/4 = 50\,\%$
Regla 3. – $\{Papel, JabonBano\} \Rightarrow \{Leche\}\ confianza = 2/4 = 50\,\%$

Lo que significa para la regla 1 es que cuando se compran juntos Leche y Papel el 50 % de las veces también se compra Jabón de Baño; y este patrón de comportamiento aparece mínimo en el 20 % del total de las compras registradas de la tienda.

5.2.2. Análisis por Antigüedad de los Datos

En un análisis por antigüedad de los datos el usuario define un rango de fechas que los tickets deben cumplir, de manera que solo se toma un fragmento de la base de datos donde buscar, permitiéndole al usuario obtener reglas que representen las relaciones entre los artículos en un espacio de tiempo (por ejemplo una semana específica).

Configuración del Análisis.- Dado un soporte del 20 % (llamado *min_soporte*) y confianza del 50 % (llamada *min_confianza*), seleccionando los artículos Leche, Papel, Jabón Baño y Jamón (con Id_articulo 1,2,3,4 respectivamente) y seleccionando un rango de fechas del 1 de Abril al 31 de Abril del 2005. El proceso de minería de datos se muestra en la Figura 5.2.

Para iniciar el análisis se calculó el mínimo soporte para esta base de datos de acuerdo a la Ecuación 5.1 y tomando en cuenta solo los tickets que cumplen con el rango de fechas, calculando $soporte = 3 * 20/100 = 0.6$ y obteniéndose al redondear que el soporte es 1.

Como se muestra en la Figura 5.2, la lista $L(1)$ se forma con los artículos seleccionados por el usuario que cumplen con el mínimo soporte. Al generar la lista $L(2)$ se eliminan los conjuntos {1,3}, {1,4} y {3,4} por no cumplir con el mínimo soporte (obtenido del conteo de tickets que cumplen con el rango de fechas).

Con los conjuntos de la lista $L(2)$ se genera la lista $L(3)$, pero al generarla se eliminan los conjuntos {1,2,4} y {2,3,4} por que sus subconjuntos {1,4} y {3,4} respectivamente, fueron

L(1)=

Artículos	Soporte
{1}	1
{2}	3
{3}	1
{4}	1

L(2)=

Artículos	Soporte
{1,2}	1
{1,3}	0
{1,4}	0
{2,3}	1
{2,4}	1
{3,4}	0

L(3)=

Artículos	Soporte
{1,2,3}	0
{1,2,4}	{1,4}
{2,3,4}	{3,4}

Figura 5.2: Análisis de los artículos con id_artículo 1,2,3 y 4 con rango de fechas del 1 al 31 de Abril del 2005

eliminados en un paso anterior. El conjunto {1,2,3} también es descartado pues no cumple con el mínimo soporte, así la lista $L(3)$ queda vacía y por lo tanto termina el ciclo y la lista final será $L(2)$.

Con la lista $L(2)$ se inicia la generación de reglas de asociación. Se toman los conjuntos {1,2}, {2,3} y {2,4}, se generan todos los subconjuntos posibles no vacíos de cada conjunto. De acuerdo a la Ecuación 4.3 se obtienen las siguientes reglas de asociación:

Regla 1. – $\{1\} \Rightarrow \{2\}\ confianza = 1/1 = 100\,\%$
Regla 2. – $\{2\} \Rightarrow \{1\}\ confianza = 1/3 = 33.33\,\%$
Regla 3. – $\{2\} \Rightarrow \{3\}\ confianza = 1/3 = 33.33\,\%$
Regla 4. – $\{3\} \Rightarrow \{2\}\ confianza = 1/1 = 100\,\%$
Regla 5. – $\{2\} \Rightarrow \{4\}\ confianza = 1/3 = 33.33\,\%$
Regla 6. – $\{4\} \Rightarrow \{2\}\ confianza = 1/1 = 100\,\%$

Solo las reglas 1, 4 y 6 son válidas para este análisis porque cumplen el mínimo de confianza especificado por el usuario.

Regla 1. – $\{Leche\} \Rightarrow \{Papel\}\ confianza = 1/1 = 100\,\%$
Regla 4. – $\{JabonBano\} \Rightarrow \{Papel\}\ confianza = 1/1 = 100\,\%$
Regla 6. – $\{Jamon\} \Rightarrow \{Papel\}\ confianza = 1/1 = 100\,\%$

Lo que significa para la regla 1 es que cuando se compra Leche el 100 % de las veces también se compra Papel y este patrón de comportamiento aparece mínimo en el 20 % del total de las compras registradas de la tienda entre el 1 al 31 de Abril del 2005.

5.3. Caso 2. Una Computadora con un Contenedor Principal y Varios Sitios

Para probar el comportamiento de los agentes y de el algoritmo de Asociación de reglas se requería una base de datos que contuviera un total aproximado de tickets de una caja de acuerdo a las consideraciones iniciales presentadas al inicio de este capítulo. El escenario definido para este caso se presenta a continuación.

Escenario:
Número de tickets : 1,000.
Número de artículos disponibles: 110.
Número de computadoras: 1 (Windows XP SP1, Celeron a 467 Mhz, 192 Mb en RAM y 19 Gb en disco duro).
Número de sitios en la computadora: 4 (simulando 4 cajas).
Base de datos: La misma en cada máquina.

Dada la dimensión de la base de datos, hacer una comprobación manual sería exhaustiva y poco ilustrativa, determinándose para la comprobación de resultados la realizaron de consultas SQL a la base de datos.

Para los análisis de este caso se simularon cuatro cajas registradoras (CAJA1, CAJA2, CAJA3 Y CAJA4), cada una en diferente sitio o computadora. Todos los agentes y los contenedores iniciados de cada sitio (simulando las cajas) se muestran en la Figura 5.3.

Para iniciar al *Agente Coordinador* y el RMA se ejecutó la siguiente línea:

```
>java jade.Boot -gui  AgenteCoordinador:misprogramas.minominero.AgenteCoordinador
```

Se requiere iniciar un contenedor para cada sitio de la siguiente manera:

```
>java jade.Boot -container -container-name NombreSitio
```

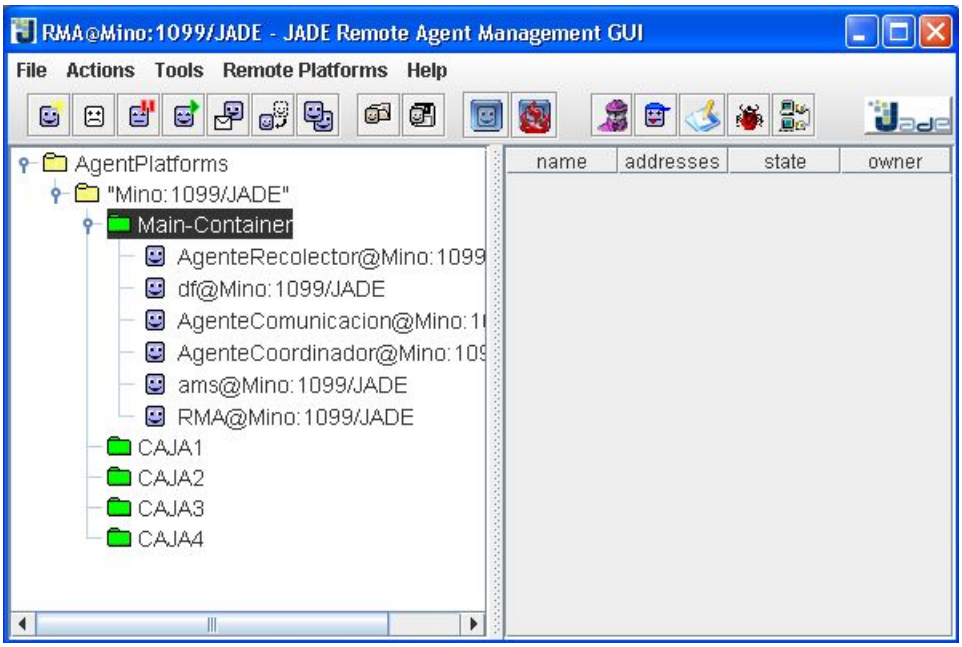

Figura 5.3: RMA del Escenario del Caso 2

5.3.1. Análisis por Departamento

En un análisis por departamento el usuario selecciona desde la interfaz web los departamentos en cuyos artículos se buscaran reglas de asociación.

Configuración del Análisis.- Dado un soporte del 10 % (llamado *min_soporte*) y confianza del 20 % (llamada *min_confianza*), y seleccionando los departamentos Abarrotes, Salchichonería, Lácteos, Frutas y Verduras, como se muestra en la Figura 5.4.

El número de identificación de proceso (**idpm**) fué **5** para este análisis. No se encontraron reglas que cumplan con el mínimo soporte y/o mínima confianza.

Para comprobar lo anterior primero se calcula el soporte de los artículos de los departamentos especificados, de acuerdo a la Ecuación 5.1 $soporte = 1000 * 10/100 = 100$.

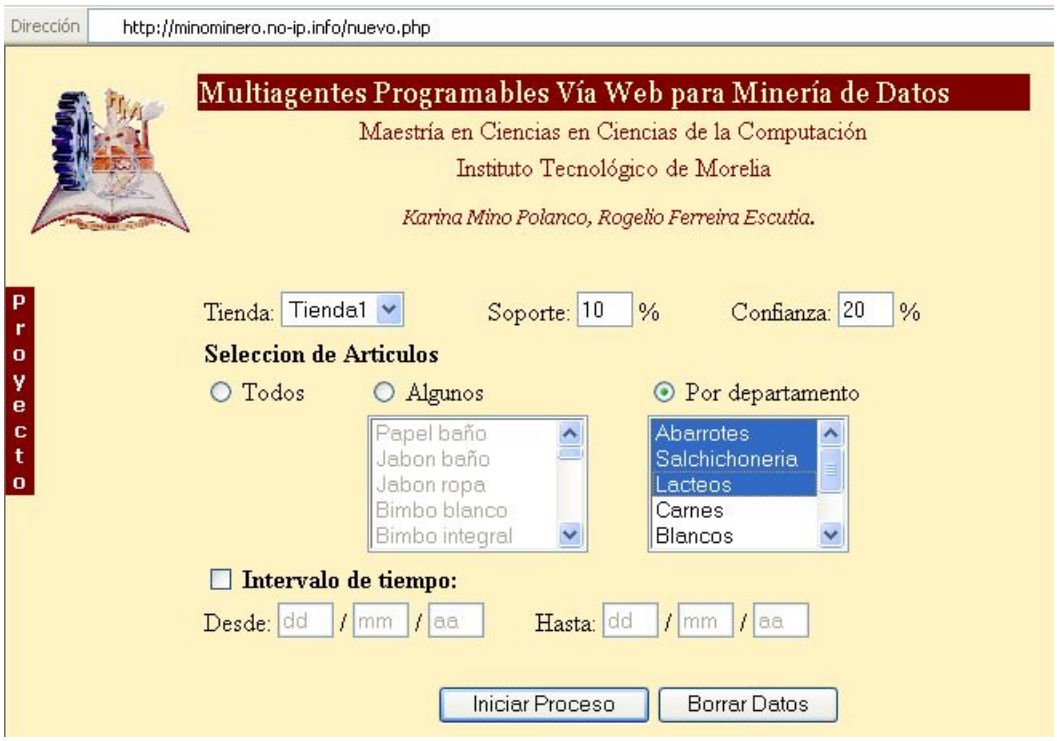

Figura 5.4: Página web para iniciar el proceso de minería de datos por departamentos

Primero obtenemos los id_articulo de los artículos de los departamentos de Abarrotes, Salchichonería, Lácteos, Frutas y Verduras:

```
mysql>select id_articulo from articulo where depto_art IN
('Abarrotes','Salchichoneria','Lacteos','FrutasyVerduras');
```

Al obtener el soporte de los artículos ejecutando la siguiente sentencia SQL (para los artículos del departamento de Abarrotes):

```
mysql>select id_articulo,count(id_ticket) from detalleticket
where id_articulo between 750000000100 and 750000000199;
```

Se encontró que sólo un artículo 750000000201 (con soporte 104) cumplía con el mínimo soporte de 100, mientras que el soporte todos los demás artículos fluctuaba entre 65 y 96.

Dado que un solo artículo cumplió el soporte, no se generó ninguna regla, pues se necesitan mínimo dos artículos. La duración del análisis fué de 40 segundos real y registrado de 1 minuto, esto se debe a que el *Agente Recolector* espera al menos 1 minuto a que el *Agente Minero* se

clone en todos los sitios y los clones alcancen a registrarse con el *Agente Coordinador*.

5.3.2. Análisis por Departamento y Antigüedad de los Datos

Configuración del Análisis.- Dado un soporte del 1 % (llamado *min_soporte*) y confianza del 17 % (llamada *min_confianza*),seleccionando los departamentos Abarrotes, Salchichonería, Lácteos, Frutas y Verduras y con fecha entre el 15 y el 30 de Septiembre del 2005, como se muestra en la Figura 5.5.

Figura 5.5: Página web para iniciar el proceso de minería de datos por departamentos y fecha

El número de identificación de proceso (**idpm**) fué **13** para este análisis.

En la Figura 5.6 se muestra los agentes mineros con sus agentes monitores distribuidos en cada sitio.

Se encontraron las reglas mostradas en la Figura 5.7, que cumplen con el mínimo soporte y la mínima confianza.

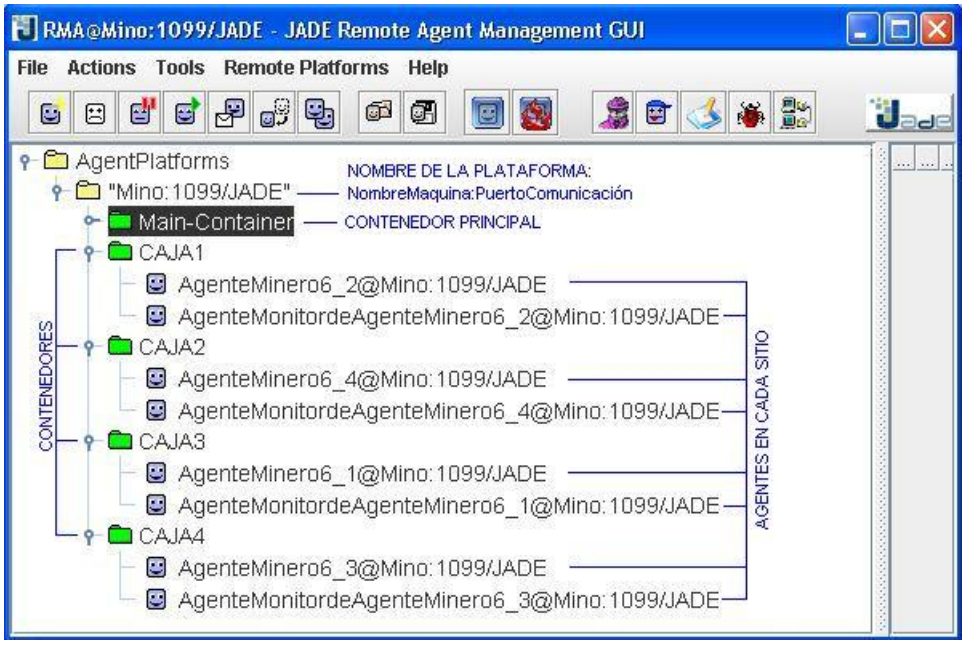

Figura 5.6: RMA para el análisis número 6

Figura 5.7: Resultado del análisis número 13

Para comprobar lo anterior primero se calcula el umbral mínimo de soporte de acuerdo a la Ecuación 5.1 $soporte = 991 * 1/100 = 9.91$ y tomando para el conteo sólo los tickets que tienen fecha entre el 15 y el 30 de Septiembre del 2005 (que son 991 tickets), lo que redondeando da un mínimo soporte de 10 tickets.

Tomamos una de las regla del resultado, por ejemplo:
Regla 10. $-$ {Jabonbano} $\Rightarrow$ {Quesopuerco}$soporte$= 2.02 %$confianza$= 27.03 %

Se obtiene el soporte del artículo Jabón Baño y del artículo Queso puerco con las siguientes sentencias SQL:

```
mysql>select count(id_ticket) from detalleticket where
id_articulo=750000000102 and ticket between 10 and 1000;
 +------------------+
 | count(id_ticket) |
 +------------------+
 |               74 |
 +------------------+
 1 row in set (0.17 sec)

mysql>select count(id_ticket) from detalleticket where
id_articulo=750000000206 and ticket between 10 and 1000;
 +------------------+
 | count(id_ticket) |
 +------------------+
 |               95 |
 +------------------+
 1 row in set (0.17 sec)
```

Para obtener el soporte del conjunto {Jabón baño, Queso puerco}:

```
mysql>select count(id_ticket) as cont from detalleticket
where id_articulo IN(750000000102,750000000206) and
ticket between 10 and 1000
group by id_ticket having cont=2;
```

```
+------------------+
| count(id_ticket) |
+------------------+
|               20 |
+------------------+
1 row in set (0.17 sec)
```

El artículo Jabón baño (750000000102) tiene un soporte de 74, el artículo llamado Queso puerco (750000000206) tiene un soporte de 95 y el conjunto {Jabón baño, Queso puerco} tiene un soporte de 20, entonces calculamos $confianza = \frac{20}{74} * 100 = 27.027\overline{027}$ de acuerdo a la Ecuación 5.2, que nos da el 27.03 % de confianza que fué el obtenido, por lo tanto la regla es válida.

5.4. Caso 3. Dos Computadoras con un Contenedor Principal y Varios Sitios

Para probar el comportamiento de los agentes y de el algoritmo de Asociación de reglas de forma distribuida se definió el siguiente escenario que se representa el ambiente real en el que trabajarían.

Escenario:
Número de tickets : 1,000 en cada máquina.
Número de artículos disponibles: 110.
Número de computadoras: 2

- Computadora **Mino**.- Windows XP SP1, Celeron a 467 Mhz, 192 Mb en RAM y 19 Gb en disco duro
- Computadora **RK**.- Windows 98 SE, Pentium II a 400 Mhz, 64 Mb RAM y 7 Gb en disco duro

Número de sitios en cada computadora: Uno (simulando 2 cajas en total).
Base de datos: Diferente en cada máquina.

Para los análisis de este caso se simularon dos cajas registradoras distribuidas en dos máquinas: En la computadora **Mino** la *CAJA1* y en la computadora **RK** la *CAJA2*, cada una en diferente sitio o computadora como se muestra en la Figura 5.8.

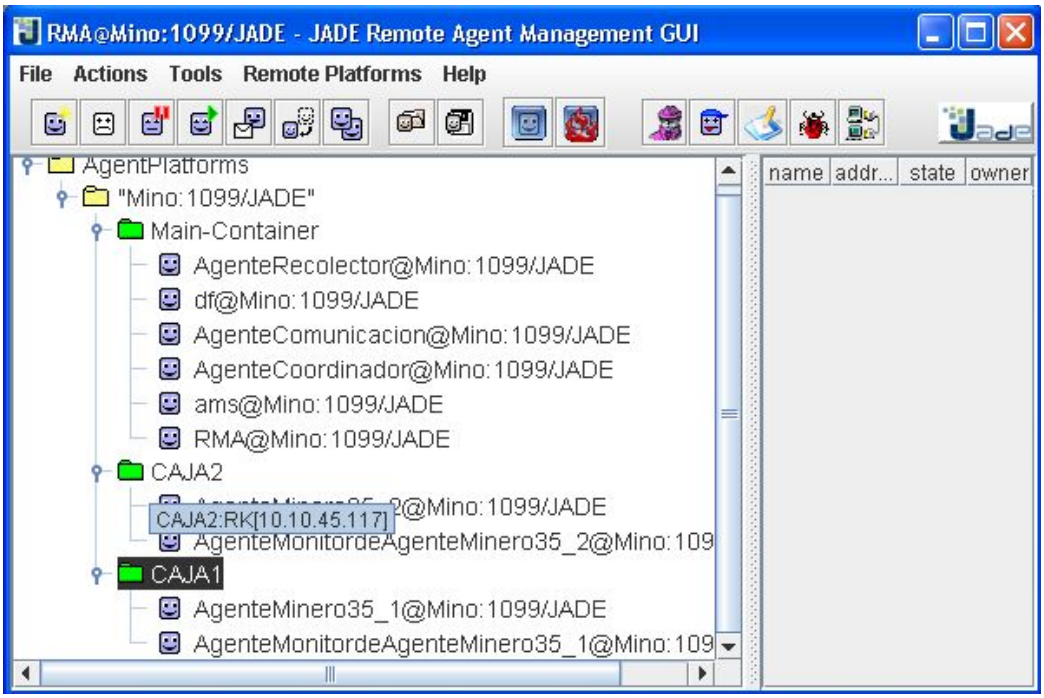

Figura 5.8: RMA del Escenario del Caso 3

Para iniciar al *Agente Coordinador* y el RMA en la máquina *Mino* se ejecutó la siguiente línea:

```
>java jade.Boot -gui  AgenteCoordinador:misprogramas.minominero.AgenteCoordinador
```

Se requiere iniciar el contenedor en la máquina **Mino**:

```
>java jade.Boot -container -container-name CAJA1
```

Para iniciar el otro contenedor en la máquina **RK**:

```
>java jade.Boot -host Mino -container -container-name CAJA2
```

5.4.1. Análisis Completo

Un análisis completo se define como la búsqueda de las relaciones entre todos los artículos de la base de datos considerando todos los tickets registrados. Obviamente, este tipo de análisis

requiere un tiempo de procesamiento mayor pero puede arrojar relaciones que el usuario no tenia contempladas.

Configuración del Análisis.- Dado un soporte del 9 % (llamado *min_soporte*) y confianza del 20 % (llamada *min_confianza*), realizar un análisis completo como se muestra en la Figura 5.9.

Figura 5.9: Página web del celular para iniciar el proceso de minería de datos completo

Para este análisis en ninguno de los sitios se encontraron reglas que cumplieran con el soporte mínimo (que es de 90 tickets para este proceso). En la Figura 5.10 se muestra la reglas que recibió el Agente Coordinador de los agentes de los diferentes sitios.

```
Clonar Minero36  a:CAJA1 Contador: 1
Minero36 iniciara el proceso de clonacion.
Clonar Minero36  a:CAJA2 Contador: 2
Minero36 iniciara el proceso de clonacion.
[AgenteCoordinador] En espera de mensajes
[AgenteCoordinador] Recibi un mensaje INFORM de AgenteMinero36_1
idpm del clon= 36
contenedor del clon= CAJA1
[AgenteCoordinador] En espera de mensajes
No llego ningun mensaje y se continua ESPERANDO
Proceso de Clonacion CONCLUIDO
Minero36 ha terminado
[AgenteCoordinador] En espera de mensajes
[AgenteCoordinador] Recibi un mensaje INFORM de AgenteMinero36_2
idpm del clon= 36
contenedor del clon= CAJA2
[AgenteCoordinador] En espera de mensajes
No llego ningun mensaje y se continua ESPERANDO
[AgenteRecolector] Recibi un mensaje RESMINERIA  de AgenteMinero36_2
ReMineria =IDPM: 36 Sitio: CAJA2 Reglas: [<0 )<0 > soporteab: 0 soportea: 0]  Ta
mbd: 1000
[AgenteRecolector] Recibi un mensaje RESMINERIA  de AgenteMinero36_1
ReMineria =IDPM: 36 Sitio: CAJA1 Reglas: [<0 )<0 > soporteab: 0 soportea: 0]  Ta
mbd: 1000
```

Figura 5.10: Reglas enviadas al Agente Coordinador para el proceso 36

No se obtuvieron reglas en ambos sitios porque el soporte de los conjuntos formados de tamaño dos varia entre 4 y 19, como se muestra en la Figura 5.11, por lo tanto estos conjuntos se eliminan por no cumplir con el mínimo soporte de 90 tickets.

```
Eliminar Conjunto <750000000106,750000000305> y soporte = 17
Eliminar Conjunto <750000000106,750000000306> y soporte = 4
Eliminar Conjunto <750000000106,750000000411> y soporte = 10
Eliminar Conjunto <750000000106,750000000412> y soporte = 13
Eliminar Conjunto <750000000106,750000000413> y soporte = 12
Eliminar Conjunto <750000000106,750000000501> y soporte = 10
Eliminar Conjunto <750000000106,750000000502> y soporte = 13
Eliminar Conjunto <750000000106,750000000503> y soporte = 14
Eliminar Conjunto <750000000106,750000000506> y soporte = 9
Eliminar Conjunto <750000000106,750000000603> y soporte = 11
Eliminar Conjunto <750000000106,750000000604> y soporte = 12
Eliminar Conjunto <750000000106,750000000607> y soporte = 12
Eliminar Conjunto <750000000106,750000000616> y soporte = 15
Eliminar Conjunto <750000000106,750000000619> y soporte = 18
Eliminar Conjunto <750000000110,750000000111> y soporte = 16
Eliminar Conjunto <750000000110,750000000113> y soporte = 12
Eliminar Conjunto <750000000110,750000000115> y soporte = 13
Eliminar Conjunto <750000000110,750000000132> y soporte = 14
Eliminar Conjunto <750000000110,750000000201> y soporte = 16
Eliminar Conjunto <750000000110,750000000206> y soporte = 11
Eliminar Conjunto <750000000110,750000000207> y soporte = 12
Eliminar Conjunto <750000000110,750000000305> y soporte = 12
Eliminar Conjunto <750000000110,750000000306> y soporte = 14
Eliminar Conjunto <750000000110,750000000411> y soporte = 12
Eliminar Conjunto <750000000110,750000000412> y soporte = 12
```

Figura 5.11: Conjuntos eliminados por no cumplir el mínimo soporte para el proceso 36

En ambos sitios se generó una lista final de conjuntos de tamaño 1 que sí cumplían con el mínimo soporte, pero sólo los conjuntos con tamaño mayor a 1 pueden ser candidatos para generar reglas, razón por la cual no se obtuvieron reglas para este proceso. En la Figura 5.12 se muestra el resultado obtenido en el sitio llamado CAJA1 para el proceso número 36, el cual se obtuvo en un tiempo de 1 minuto.

5.4.2. Análisis por Artículos y Antigüedad de los Datos

En ocasiones es mejor realizar un análisis delimitado por ciertos parámetros que un análisis completo que representa mayor tiempo de procesamiento, sobre todo si se desea obtener resultados sobre un conjunto conocido de datos. A continuación se muestra los resultados de un análisis delimitado por artículos específicos y la fecha de los tickets registrados.

```
Cerrando base de datos
Lista final = Lista (
 <7500000000104> y soporte = 93 ,
 <7500000000106> y soporte = 90 ,
 <7500000000110> y soporte = 92 ,
 <7500000000111> y soporte = 93 ,
 <7500000000113> y soporte = 92 ,
 <7500000000115> y soporte = 91 ,
 <7500000000132> y soporte = 91 ,
 <7500000000201> y soporte = 104 ,
 <7500000000206> y soporte = 95 ,
 <7500000000207> y soporte = 96 ,
 <7500000000305> y soporte = 93 ,
 <7500000000306> y soporte = 91 ,
 <7500000000411> y soporte = 96 ,
 <7500000000412> y soporte = 90 ,
 <7500000000413> y soporte = 90 ,
 <7500000000501> y soporte = 90 ,
 <7500000000502> y soporte = 96 ,
 <7500000000503> y soporte = 104 ,
 <7500000000506> y soporte = 90 ,
 <7500000000603> y soporte = 96 ,
 <7500000000604> y soporte = 90 ,
 <7500000000607> y soporte = 94 ,
 <7500000000616> y soporte = 95 ,
 <7500000000619> y soporte = 90 )
Longitud Lista final = 24
LONGITUD de los conjuntos= 1
NO SE GENERO NINGUNA REGLA
RESMINERIA=IDPM: 36 Sitio: CAJA1 Reglas: [<0 )<0 > soporteab: 0 soportea: 0]
mbd: 1000
[AgenteMinero36_1] Creando mensaje INFORM
[AgenteMinero36_1] Se envio mensaje INFORM-RESMINERIA
mensaje de RESULTADOS enviado
Cerrando base de datos
AgenteMinero36_1 ha terminado
AgenteMonitordeAgenteMinero36_1 DEREGISTERED WITH THE DF
```

Figura 5.12: Resultado para el proceso 36 en la CAJA1

Configuración del Análisis.- Dado un soporte del 1 % (llamado *min_soporte*) y confianza del 18 % (llamada *min_confianza*),seleccionando el departamento Abarrotes y la quincena entre el 15 y el 31 de Septiembre del 2005.

Cada sitio (o computadora) contiene una base de datos con la misma cantidad de artículos y de tickets, sin embargo la composición de cada tickets (en cuanto a cantidad y tipo de artículos) es diferente, así como las fechas en que fueron generados éstos tickets, como se muestran en las Figuras 5.13 y 5.14 de la CAJA1 y CAJA2 respectivamente.

Como puede observarse en las Figuras 5.13 y 5.14, el soporte de acuerdo a las fechas es de 9 tickets para el sitio CAJA1 y de 10 tickets para el sitio CAJA2.

Se obtuvieron 45 reglas en la CAJA1, mostradas parcialmente en la Figura 5.15. Para la CAJA2 se obtuvieron solamente 3 reglas mostradas en la Figura 5.16.

```
mysql> select count(id_ticket),fecha from ticket group by fecha;
+------------------+------------+
| count(id_ticket) | fecha      |
+------------------+------------+
|                9 | 2005-09-12 |
|              991 | 2005-09-20 |
+------------------+------------+
2 rows in set (0.03 sec)
```

Figura 5.13: Cantidad de tickets por fecha del sitio CAJA1

```
mysql> select count(id_ticket),fecha from ticket group by fecha;
+------------------+------------+
| count(id_ticket) | fecha      |
+------------------+------------+
|                9 | 2005-09-12 |
|              510 | 2005-09-22 |
|              400 | 2005-09-23 |
|               81 | 2005-10-31 |
+------------------+------------+
4 rows in set (0.05 sec)
```

Figura 5.14: Cantidad de tickets por fecha del sitio CAJA2

```
 [<750000000113 ><750000000117 > soporteab: 17 soportea: 86] [<750000000125 ><75
0000000113 > soporteab: 14 soportea: 75] [<750000000127 ><750000000113 > soporte
ab: 18 soportea: 82] [<750000000113 ><750000000127 > soporteab: 18 soportea: 86]
 [<750000000122 ><750000000116 > soporteab: 16 soportea: 83] [<750000000116 ><75
0000000122 > soporteab: 16 soportea: 85] [<750000000117 ><750000000118 > soporte
ab: 12 soportea: 65] [<750000000117 ><750000000126 > soporteab: 12 soportea: 65]
 [<750000000117 ><750000000127 > soporteab: 12 soportea: 65] [<750000000117 ><75
0000000132 > soporteab: 14 soportea: 65] [<750000000118 ><750000000132 > soporte
ab: 15 soportea: 78] [<750000000122 ><750000000119 > soporteab: 15 soportea: 83]
 [<750000000120 ><750000000121 > soporteab: 14 soportea: 77] [<750000000133 ><75
0000000121 > soporteab: 14 soportea: 74] [<750000000128 ><750000000122 > soporte
ab: 18 soportea: 81] [<750000000122 ><750000000128 > soporteab: 18 soportea: 83]
 [<750000000122 ><750000000132 > soporteab: 16 soportea: 83] [<750000000133 ><75
0000000122 > soporteab: 14 soportea: 74] [<750000000125 ><750000000124 > soporte
ab: 15 soportea: 75] [<750000000131 ><750000000124 > soporteab: 15 soportea: 79]
 [<750000000128 ><750000000126 > soporteab: 15 soportea: 81] [<750000000126 ><75
0000000128 > soporteab: 15 soportea: 78] [<750000000130 ><750000000132 > soporte
ab: 16 soportea: 81]  Tambd: 991
[AgenteMinero38_1] Creando mensaje INFORM
[AgenteMinero38_1] Se envio mensaje INFORM-RESMINERIA
mensaje de RESULTADOS enviado
Cerrando base de datos
AgenteMinero38_1 ha terminado
AgenteMonitordeAgenteMinero38_1 DEREGISTERED WITH THE DF
```

Figura 5.15: Resultado de CAJA1 para el proceso 38

```
Regla no cumplio confianza
Numero binario=101
Conjunto 5 (2): (7500000000103,7500000000129) y soporte = 14
Regla numero 5
(7500000000103,7500000000129) y soporte = 14 => (7500000000108) y soporte = 0
Soporte de l = 9
Soporte de s = 14
Confianza de la regla = 64.28
Soporte de la regla = 0.98
Numero binario=110
Conjunto 6 (2): (7500000000103,7500000000108) y soporte = 15
Regla numero 6
(7500000000103,7500000000108) y soporte = 15 => (7500000000129) y soporte = 0
Soporte de l = 9
Soporte de s = 15
Confianza de la regla = 60.0
Soporte de la regla = 0.98
RESMINERIA=IDPM: 38 Sitio: CAJA2 Reglas: [(7500000000108 7500000000129 )(750000000
103 ) soporteab: 9 soportea: 16] [(7500000000103 7500000000129 )(7500000000108 ) so
porteab: 9 soportea: 14] [(7500000000103 7500000000108 )(7500000000129 ) soporteab:
 9 soportea: 15]  Tambd: 910
[AgenteMinero38_2] Creando mensaje INFORM
[AgenteMinero38_2] Se envio mensaje INFORM-RESMINERIA
mensaje de RESULTADOS enviado
AgenteMinero38_2 ha terminado
Cerrando base de datos
AgenteMonitordeAgenteMinero38_2 DEREGISTERED WITH THE DF
```

Figura 5.16: Resultado de CAJA2 para el proceso 38

La duración completa del análisis fué de 4 minutos con 5 segundos. La comprobación de las reglas encontradas se efectúa de la misma manera que para el Caso 2.

5.4.3. Análisis Ejecutándose Simultáneamente

El usuario puede especificar a través de la interfaz web la ejecución de varios análisis, los cuales se ejecutan simultáneamente en cada sitio o máquina. El objetivo de la presente prueba es mostrar si el tiempo de ejecución de un análisis ya probado se ve afectado si se están ejecutando al mismo tiempo otro análisis.

Configuración del Análisis.- Igual a la prueba anterior, se ejecutan dos análisis iguales que se ejecutan simultáneamente en cada sitio. Los dos análisis tienen, por lo tanto, un soporte del 1 % (llamado *min_soporte*) y confianza del 18 % (llamada *min_confianza*),se ha seleccionado el departamento Abarrotes y la quincena entre el 15 y el 31 de Septiembre del 2005.

Al ejecutar el análisis de la prueba anterior se encontró que el tiempo de ejecución fué de 4 minutos con 5 segundos, sobre un sitio en el que solo se ejecutaba este análisis. Al desarrollar la

presente prueba se encontró que los tiempos de los análisis se vieron afectados por la ejecución simultánea de los mismos. Esto se muestra en la Figura 5.17.

```
mysql> select idpm,hora from restiempo where idpm in (38,42,43);
+------+----------+
| idpm | hora     |
+------+----------+
|   38 | 12:08:06 |
|   42 | 11:02:40 |
|   43 | 11:02:05 |
+------+----------+
3 rows in set (0.00 sec)

mysql> select idpm,hora from mineria where idpm in (38,42,43);
+------+----------+
| idpm | hora     |
+------+----------+
|   38 | 12:04:01 |
|   42 | 10:54:36 |
|   43 | 10:54:43 |
+------+----------+
3 rows in set (0.00 sec)
```

Figura 5.17: Tiempos de inicio y fin de procesamiento de los procesos 38,42 y 43

Como se aprecia, en lugar de que cada análisis se ejecute en 4 minutos con 5 segundos, se ejecutó en 8 minutos con 5 segundos para el proceso 42 y de 7 minutos con 48 segundos para el proceso 43, por lo que se deduce que el tiempo de procesamiento se suma con cada análisis que se ejecuta simultáneamente.

Se realizó otra prueba con configuraciones del análisis diferente, de la siguiente manera:

Configuración del Análisis.- En el análisis con número de proceso de minería (idpm) 41 se tiene un soporte del 1 % y confianza del 18 %,se ha seleccionado el departamento Salchichonería y la quincena entre el 15 y el 31 de Septiembre del 2005. Para el análisis con idpm 40 la configuración de soporte, confianza y de antigüedad de los datos es igual al idpm 41, pero en este análisis el único departamento seleccionado es Abarrotes (este análisis tiene la misma configuración de los análisis con idpm 38, 42 y 43 detallados en la prueba anterior).

Previamente se ejecutó un análisis (proceso 39) igual al proceso 41, ejecutándose este único análisis en menos de un minuto (aunque por especificaciones del proyecto el tiempo oficial de registro es de un minuto). Al ejecutarse simultáneamente el proceso 40 y 41 los tiempos registrados se ven afectados como en la prueba anterior, estos tiempos se muestran en la Figura 5.18.

```
mysql> select idpm,hora from mineria where idpm in (38,39,40,41);
+------+----------+
| idpm | hora     |
+------+----------+
|   38 | 12:04:01 |
|   39 | 10:38:22 |
|   40 | 10:44:00 |
|   41 | 10:44:10 |
+------+----------+
4 rows in set (0.06 sec)

mysql> select idpm,hora from restiempo where idpm in (38,39,40,41);
+------+----------+
| idpm | hora     |
+------+----------+
|   38 | 12:08:06 |
|   39 | 10:39:22 |
|   40 | 10:49:22 |
|   41 | 10:45:10 |
+------+----------+
4 rows in set (0.00 sec)
```

Figura 5.18: Tiempos de inicio y fin de procesamiento de los procesos 38,39,40 y 41

Para el análisis 41, el tiempo fué de un minuto, que es igual al del proceso 39; pero aunque los tiempos son iguales si hubo un incremento en el tiempo de ejecución de este análisis, sólo que por ser un análisis muy sencillo y rápido queda establecido el tiempo mínimo de un minuto.

El impacto en el tiempo se aprecia en el proceso 40 (igual al proceso 38), que fué de 5 minutos con 22 segundos que es mayor a los 4 minutos con 5 segundos que duró el análisis que se ejecutó aisladamente.

5.5. Resultados Obtenidos

Se consideran que en base de las pruebas realizadas los resultados obtenidos son los siguientes:

1. El tiempo mínimo para un análisis es de 1 minuto, aún cuando todos los agentes involucrados en el análisis envíen sus resultados antes de ese tiempo.

2. En el ambiente real del proyecto, en cada sitio solamente existe un contenedor (que denominado CAJA) como en un entorno como el Caso 3. En este tipo de Caso el tiempo de un análisis por departamentos o por artículos varía entre 10 y 20 minutos. Para análisis completos el tiempo varió entre hora y media y dos horas.

3. El tiempo de un análisis que involucra un solo departamento varia entre 4 y 5 minutos cuando en un sitio sólo se esta realizando un análisis.

4. El tiempo de un análisis por departamento o por artículos disminuye un X % al especificar una rango de fechas en que los tickets fueron registrados.

5. Cuando se ejecutan dos análisis simultáneamente en un sitio, el tiempo de ejecución de los análisis se incrementa entre un 25 y 100 %, dependiendo de la complejidad o sencillez de los análisis involucrados.

6. Cuando se realizaron más de dos análisis en un sitio, el tiempo de ejecución de éstos fue aproximadamente la suma de los tiempos de ejecución de cada proceso cuando se ejecutaron aisladamente.

7. Las bases de datos utilizadas tenían el mismo número de tickets y de artículos registrados. Sin embargo, cada ticket variaba en la cantidad de artículos que contenían y esto se vio reflejado en el tiempo de obtención de resultados, puesto que la computadora RK los tickets contenían menos artículos y obtenía los resultados en un tiempo menor a la computadora Mino (que tiene una velocidad mayor).

5.6. Ventajas y Desventajas del Proyecto

A continuación se muestra una comparativa de las características del proyecto desarrollado y de algunas herramientas de dominio público para minería de datos. Esta comparativa sirve de base (junto con los resultados obtenidos del proyecto) para definir las ventajas y desventajas del proyecto desarrollado.

5.6.1. Comparativa con Herramientas de Minería

Algunas herramientas de dominio público existentes, que realizan el método de asociación de reglas, son: WEKA, DBMiner, ARMiner recientemente Papyrus, este último es el único que realiza minería de datos distribuida sobre clusters y superclusters [12, 5, 10, 15, 7].

Se realizó una comparación entre las características del proyecto desarrollado y las herramientas de dominio público antes mencionadas [12, 5, 10, 15, 7] mostrada en la Tabla 5.3.

Características	Proyecto	DBMiner	ARMiner	WEKA	Papyrus
Distribución del proceso de Minería de Datos	Si	No	No	No	Si
Minería de datos Cliente-Servidor	No	Si	Si	Si	No
Freeware	Si	Si	Si	Si	Si
Multiplataforma	Si	No	Si	Si	No
Interfaz en Web	Si	No	No	No	No
Interfaz en dispositivos móviles	Si	No	No	No	No
Análisis de datos actualizados en tiempo real	Si	No	No	No	No
Análisis de datos históricos	Si	Si	Si	Si	Si

Tabla 5.3: Comparación del proyecto desarrollado con herramientas de dominio público para minería de datos.

Como resultado de la comparativa, se encontró que todas las herramientas antes mencionadas (excepto Papyrus) no distribuyen el proceso de minería de datos en varias computadoras, lo que implica que en un servidor tienen una gran base de datos histórica (información de hace semanas, meses, años, etc.) y sobre esta se realiza el proceso de minería de datos. El problema de realizar la minería de datos sobre bases de datos históricas es que los resultados obtenidos reflejan relaciones pasadas, dado que son datos de tiempo atrás. Los resultados de una minería de datos son más relevantes y oportunos para una empresa si se realiza la sobre datos actuales, de esta manera la toma de decisiones se basa sobre información que describe la situación actual de la empresa y no la pasada. Además, distribuir el proceso de minería de datos es importante ya que disminuye el tiempo de ejecución del proceso, dado que se realiza el análisis en fragmentos de la base de datos, aprovechando que la mayoría de las empresas tienen sus bases de datos fragmentadas en diferentes computadoras o sitios.

Otro inconveniente de las herramientas anteriores es que el usuario gerencial tiene que iniciar el proceso y ver los resultados desde la máquina donde reside el software de minería, cuando lo más factible es hacer la solicitud de minería y recibir los resultados donde quiera que éste se encuentre.

El presente proyecto utiliza multiagentes para distribuir el trabajo en diferentes computadoras sobre cualquier sistema operativo, a diferencia de DBMiner, ARMiner y WEKA que realizan un proceso de minería de datos centralizado y de Papyrus que si distribuye el proceso pero no es multiplataforma.

El multiagentes desarrollados pueden mostrar resultados y ser programados desde donde se encuentre el usuario a través de una página web, un celular o un asistente personal, algo que no es posible utilizando DBMiner, ARMiner, WEKA y Papyrus.

5.6.2. Ventajas y Desventajas

De acuerdo a las pruebas realizadas y a la comparativa entre las herramientas de dominio público de minería de datos vista previamente en la Tabla 5.3, se observaron las siguientes ventajas del proyecto:

1. **Rapidez**.- Al efectuar el análisis de minería de datos sobre fragmentos distribuidos de la base de datos, el tiempo de ejecución de un análisis se divide entre el número de sitios o máquinas involucradas, lo que se traduce en un tiempo más corto que el tiempo de ejecución sobre una única base de datos gigantesca.

2. **Interfaz Web con el usuario**.- Es posible definir los parámetros de un proceso de minería de datos y obtener los resultados desde la página web `http://minominero.no-ip.info`.

3. **Interfaz Web para dispositivos móviles**.- También se desarrolló una página web para dispositivos móviles (`http://minominero.no-ip.info\wml`), como celulares y asistentes personales, para especificar los parámetros y la visualización de los resultados de un proceso de minería de datos.

4. **Resultados actuales**.- Los análisis de minería de datos se realizan sobre la base de datos que esta en uso, actualizándose en tiempo real, por lo que los resultados involucran los datos actuales y no datos históricos como otras herramientas de minería de datos vistas en el Estado del Arte.

5. **Multiplataforma**.- Debido a que los multiagentes fueron desarrollados en Java y Jade tienen la capacidad de funcionar sobre cualquier sistema operativo.

6. **Escalabilidad**.- Es muy fácil agregar una máquina o sitio al sistema puesto que el Agente Coordinador se encarga de obtener las direcciones de las máquinas disponibles en la red sin intervención del usuario.

7. **Uso de software libre**.- En el desarrollo del proyecto se utilizó únicamente software libre: java, jade, mysql, apache, php, HTML y WML.

Asimismo, se encontraron las siguientes desventajas:

1. **Sólo realiza minería de datos descriptiva**.- En comparación con otras herramientas libres analizadas en el Estado del Arte, las cuales si realizan minería de datos descriptiva y también predictiva.

2. **No utiliza varios algoritmos de minería descriptiva**.- En otras herramientas se utilizan varios algoritmos de minería de datos descriptiva, de forma que pueda realizar otro tipo de análisis para la solución de distintos tipos de información que se desee obtener.

3. **No hace análisis sobre Data Warehouse**.- Debido a que se requiere un proceso de limpieza e normalización previo para que los resultados puedan utilizarse en un análisis de minería de datos.

Capítulo 6

Conclusiones y Trabajos Futuros

6.1. Conclusiones

Del desarrollo del presente proyecto se concluye:

1. Se logró el objetivo planteado puesto que se diseñaron e implementaron multiagentes distribuidos que utilizan algoritmos de *Minería de Datos Descriptiva* para encontrar información útil en red y que son programables vía web para permitir la visualización de resultados y la reconfiguración de los parámetros de minería de datos a través de Internet o cualquier dispositivo móvil (como celulares y asistentes personales), de tal manera que el proceso de minería de datos es más automático, flexible y rápido que el proceso tradicional.

2. Se probó que la minería de datos descriptiva es muy útil cuando se desea encontrar relaciones entre los datos, relaciones que no se encuentran sin las técnicas de minería de datos.

3. Se implementaron las características más importantes de JADE, concluyéndose que el desarrollo de agentes utilizando JADE permite al programador preocuparse por sólo definir las tareas que desee que realice el agente, ya que JADE es quién maneja y controla los aspectos internos de la comunicación entre los agentes, siendo una herramienta que facilita su creación.

4. JADE proporciona un conjunto de herramientas que facilitan el desarrollo de multiagentes, se probaron sus ventajas en la definición de comportamientos de los agentes, la comunicación entre ellos, su portabilidad, su movilidad y la definición de ontologías propias, pero faltó probar su funcionamiento sobre dispositivos móviles ya que esta característica no es aplicable a este proyecto.

5. El presente proyecto provee una herramienta para minería de datos distribuida de dominio público, donde puede probarse las ventajas que ofrecen los agentes y su programación con una herramienta para el desarrollo de multiagentes.

6.2. Trabajos Futuros

Para trabajos futuros se proponen los siguientes:

- Implementar otros algoritmos de minería descriptiva como son: caracterización y discriminación de conceptos o clases, clasificación, análisis de grupos y análisis de regresión. El implementar otros algoritmos de minería descriptiva proporciona otra forma de visualizar las relaciones entre los datos.

- La implementación de minería de datos predictiva, pues esta se emplea para predecir comportamientos en los datos y por lo mismo es muy útil en la toma de decisiones.

- Añadir un módulo de inteligencia artificial a los agentes, de manera que, de acuerdo a su interacción con el usuario, el agente sea capaz de identificar si un análisis ya se realizó anteriormente, o si es muy similar y el resultado no fue satisfactorio para el usuario, y así evitar la ejecución innecesaria.

- Desarrollar agentes que existan en ambientes móviles, para que desde la interfaz con el usuario el agente pueda realizar una asesoría sobre los parámetros del análisis de minería de datos.

Referencias

[1] Jean-Marc Adamo. *Data Mining for Association Rules and Sequential Patterns : Sequential and Parallel Algorithms*. Springer, 2001.

[2] Fabio Bellifemine and G. Caire. Jade administrator's guide. http://jade.cselt.it, 2003.

[3] Fabio Bellifemine and G. Caire. Jade programmer's guide. http://jade.cselt.it, 2004.

[4] Foundation for Intelligent Physical Agents. Foundation for intelligent physical agents: Specifications. http://www.fipa.org, 2005.

[5] Jiawei Han, Y. Fu, and J. Chiang. *DBMiner: A System por Mining Knowledge in Large Relational Database*. Morgan Kaufmann Publishers, 1997.

[6] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 1a edition, 2001.

[7] J.M. Hernández, J.A. Botia, and A. Gómez. Asistencia personalizada a la minería de datos con agentes inteligentes. *Intelligent Multimedia Information Retrieval in the Net*, 2004.

[8] Telecom Italia Lab. Papers and resources. http://jade.cselt.it, 2005.

[9] Stuart J. Russell and Peter Norving. 1a edition.

[10] L. Grossman S. Bailey and H. Sivakumar. Papyrus: A system por data over local and wide area clusters and superclusters. *IEEE Computer Society Press*, 1999.

[11] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers, 2000.

[12] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers, 2005.

[13] Michael Wooldridge. *Introduction to MultiAgent Systems*. Wiley, 2002. 50 dólares.

[14] Celestino Mosquera y Antonino Santos. Análisis y estudio experimental de herramientas basadas en agentes. *Universidad de Coruña*, 2001.

[15] P. Britos y R. García. Selección de herramientas de explotación de datos. In *X Congreso Argentino de Ciencias de la Computación*, 2005.

Apéndice A

JADE

JADE (Java Agent DEvelopment Framework,Ambiente de Desarrollo de Agentes en Java) es un entorno de trabajo que simplifica la construcción de sistemas multiagente que siguen el estándar FIPA (Foundation for Intelligent Physical Agents, Fundación para Agentes Físicos Inteligentes) [4], así como un conjunto de herramientas para la depuración de las plataformas. JADE está implementado en Java y es un software libre y de código abierto, que ha sido desarrollado por el CSELT (Centro Studi e Laboratori Telecomunicación) del grupo Telecom Italia [8].

Dado que JADE y los agentes que el usuario define para una aplicación específica utilizan el lenguaje de desarrollo Java, la plataforma de agentes tiene una total independencia de los sistemas operativos empleados.

JADE tiene disponibles un conjunto de herramientas con interfaz gráfica para simplificar la administración y monitoreo de los agentes [2]:

1. **RMA** (Remote Management Agent, Agente de Gerencia Remoto) para la gestión de la plataforma. Permite crear, eliminar, suspender o enviar un mensaje a un agente, así como la eliminación de contenedores.

2. **Dummy Agent** para la envío de mensajes y visualización de los mensajes enviados o recibidos a un agentes.

3. **Sniffer Agent** para la visualización de la comunicación entre varios agentes.

4. DF GUI para controlar el conocimiento de los DFs sobre los agentes con sus servicios registrados, con la posibilidad de formar redes complejas con DFs de otras plataformas.

5. **Introspector Agent** para el monitoreo y control del ciclo de vida de un agente, así como su cola de mensajes y de comportamientos (behaviours).

A.1. Instalación y configuración

Primero, descargue JADE versión 3.3 (la última versión hasta la emisión de esta tesis) de la página `http://jade.cselt.it`, e instale normalmente.

Es necesario definir unas variables de entorno para que el compilador de Java encuentre las librerías de JADE. Para Windows XP los pasos para definir las variables de entorno son los siguientes:

- En las propiedades de MiPC accede a *Propiedades avanzadas*.
- Selecciona el botón Variables de entorno.
- En las variables de entorno del usuario (no en las variables del sistema) define una nueva variable cuyo nombre sea *CLASSPATH* con valor:
  `.;C:\ruta_de_jade\lib\jade.jar;C:\ruta_de_jade\lib\jadeTools.jar;`
  `C:\ruta_de_jade\lib\Base64.jar;C:\ruta_de_jade\lib\iiop.jar;`
  `C:\ruta_de_jade\lib\http.jar;%CLASSPATH%`
  sin dejar espacios en blanco.
- Define otra nueva variable con nombre PATH y valor :
  `.;C:\ruta_de_java\bin;C:\ruta_de_jade;%PATH%`.
- Reinicia la máquina.

Para Windows 95/98 los pasos para la configuración de JADE son:

- Ejecutar el comando *msconfig*.

- En el archivo Autoexec.bat selecciona el botón Nuevo.
- Define una nueva variable escribiendo:
  `set CLASSPATH=.;C:\ruta_de_jade\lib\jade.jar;`
  `C:\ruta_de_jade\lib\jadeTools.jar;C:\ruta_de_jade\lib\Base64.jar`
  `;C:\ruta_de_jade\lib\iiop.jar;C:\ruta_de_jade\lib\http.jar;%CLASSPATH%`
  sin dejar espacios en blanco.
- Define otra nueva variable:
  `set PATH=.;C:\ruta_de_java\bin;C:\ruta_de_jade;%PATH%`.
- Reinicia la máquina.

Para comprobar que JADE se ha instalado y configurado correctamente, desde una consola teclea lo siguiente:
`>java jade.Boot -gui`

Este comando abrirá la herramienta RMA de JADE, mostrada en la Figura A.1, donde se puede verificar que se ha iniciado una plataforma con un contenedor principal, dentro del cual se encuentran los tres agentes predefinidos por JADE (DF, AMS y RMA) para la gestión de la plataforma de agentes.

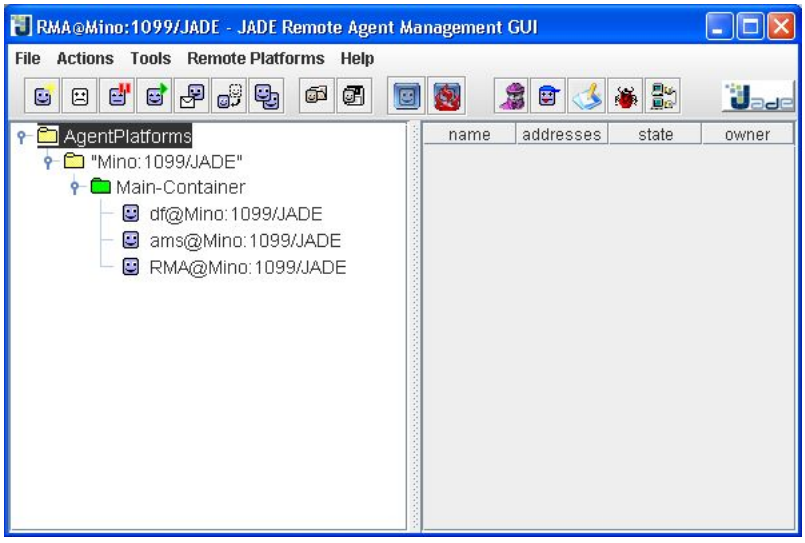

Figura A.1: Herramienta RMA de JADE

Apéndice B

Código Fuente de los Agentes Implementados con JADE

B.1. Agente Coordinador

```
package misprogramas.minominero; import jade.core.Runtime; import
jade.core.Profile; import jade.core.ProfileImpl; import
jade.wrapper.*; import jade.util.leap.*; import jade.proto.*; import
jade.lang.acl.*; import jade.domain.JADEAgentManagement.*; import
jade.domain.mobility.MobilityOntology; import jade.domain.FIPANames;
import jade.content.lang.Codec; import jade.content.lang.sl.SLCodec;
import jade.core.Agent; import jade.core.AID; import
jade.domain.FIPAAgentManagement.*; import jade.domain.DFService;
import jade.domain.FIPAException; import
jade.core.behaviours.CyclicBehaviour; import
jade.lang.acl.ACLMessage; import jade.lang.acl.MessageTemplate;
import jade.core.behaviours.*; import jade.content.*; import
jade.content.onto.*; import jade.content.onto.basic.*; import
jade.content.lang.*; import jade.content.lang.sl.*; import
java.sql.*; import misprogramas.minominero.ontologia.*;

public class AgenteCoordinador extends Agent {
     private ContentManager manager     = (ContentManager)getContentManager();
     private Codec          codec       = new SLCodec(); //LEAPCodec
     private Ontology       ontology    = OntologiaMineria.getInstance();
     public Basedatos basedat = new Basedatos();

     public void setup() {
        try {
            // create the agent descrption of itself
            DFAgentDescription dfd = new DFAgentDescription();
            dfd.setName(getAID());
            // register the description with the DF
            DFService.register(this, dfd);
            System.out.println(getLocalName()+" registrado con el DF");
        }
```

```
catch (FIPAException e) { e.printStackTrace(); }

//Registrar el lenguaje y ontologia-mineria
manager.registerLanguage(codec);
manager.registerOntology(ontology);

try {
    // Toma una instancia en tiempo de ejecucion JADE
    Runtime rt = Runtime.instance();

    // crea un enlace al contenedor local
    final AgentContainer ac = getContainerController();

    //Si se desea enviar el nombre del agente propietario
    final String agentepropietario = this.getLocalName();
    Object args[] = new Object[1];
    args[0] = agentepropietario;

    //Crea el agente Recolector de este agenteMinero
    AgentController agente_r = ac.createNewAgent("AgenteRecolector","misprogramas.minominero.
    AgenteRecolector",args);
    agente_r.start();

    //Crea el agente Comunicacion de este agenteMinero
    AgentController agente_c = ac.createNewAgent("AgenteComunicacion","misprogramas.minominero.
    AgenteComunicacion",args);
    agente_c.start();

    /*Conexion a la base de datos*/
    basedat.ConectarBd("mercado","root","ruben");

    addBehaviour(new CyclicBehaviour(this){
        public void action() {
           System.out.println("[" + myAgent.getLocalName() + "] En espera de mensajes");

           //En un subproceso de mineria se Escucha
           //Escucha si llega un mensaje MINERIA para lanzar un agente minero
           ACLMessage msg = receive();//MessageTemplate.MatchPerformative(ACLMessage.REQUEST));
           try{
            if (msg != null) {
            switch(msg.getPerformative()) {
             case ACLMessage.REQUEST:
                System.out.println("[" + myAgent.getLocalName() + "] Recibi un mensaje REQUEST de "
                +msg.getSender().getLocalName());
                Action a = (Action) manager.extractContent(msg);
                AcMineria proc_rec= (AcMineria) a.getAction();
                if(proc_rec instanceof AcMineria) {
                    //argumentos del agente MineroX
                    final String agentepropietario = myAgent.getLocalName();
                    Object args[] = new Object[10];
                    args[0] = proc_rec.getIdpm();
                    args[1] = proc_rec.getBasedatos();
                    args[2] = proc_rec.getTabla();
                    args[3] = proc_rec.getUsuario();
                    args[4] = proc_rec.getContrasenia();
                    args[5] = proc_rec.getTipocond();
                    args[6] = proc_rec.getCondicion();
                    args[7] = proc_rec.getCondicion2();
                    args[8] = proc_rec.getSoporte();
                    args[9] = proc_rec.getConfianza();

                    // Si el mensaje que llegó fue ACMINERIA entonces LANZA UN AGENTE con elproceso de mineria
                    System.out.println("[" + myAgent.getLocalName() + "] Mensaje tipo ACMINERIA");
```

```
                        try{
                            AgentController agente_m = ac.createNewAgent("Minero"+proc_rec.getIdpm(),
                            "misprogramas.minominero.AgenteMinero",args);
                            agente_m.start();
                        }
                        catch(Exception e) {e.printStackTrace();}
                    }
                    break;
                case ACLMessage.INFORM:
                    ContentElement p = manager.extractContent(msg);
                    if(p instanceof ClonConfirma) {
                        System.out.println("[" + myAgent.getLocalName() + "] Recibi un mensaje INFORM de "
                        +msg.getSender().getLocalName());
                        ClonConfirma confirma_rec = (ClonConfirma)p;
                        System.out.println("idpm del clon= "+confirma_rec.getIdpm());
                        System.out.println("contenedor del clon= "+confirma_rec.getContenedor());
                        //Cambiar la band_clono del sitio en la tabla sitios_disp
                        try{
                            basedat.Actualizar("update sitios_disp SET band_clono=1 where idpm="
                            +confirma_rec.getIdpm()+" and nom_sitio='"+confirma_rec.getContenedor()+"'");
                        }
                        catch(SQLException ex) { System.out.println(ex); }
                    }
                    break;
                default:
                    System.out.println("[" + getLocalName() + "] MENSAJE MAL FORMADO.");
                }//fin case
                }
                else {
                    System.out.println("No llego ningun mensaje y se continua ESPERANDO");
                    block();
                }
                }//fin try
                catch (Codec.CodecException fe) {
                    System.err.println("FIPAException in fill/extract Msgcontent:" + fe.getMessage());
                }
                catch (OntologyException fe) {
                    System.err.println("OntologyException in getRoleName:" + fe.getMessage());
                }
            } //fin action
        });//fin behaviour
    }
    catch(Exception e) {e.printStackTrace();   }
  }//fin setup

  protected void takeDown() {
      // Deregister with the DF
          try {
              DFService.deregister(this);
              System.out.println(getLocalName()+" DEREGISTERED WITH THE DF");
          } catch (FIPAException e) {
              e.printStackTrace();
          }
   }//fin takedown
}//fin de clase
```

B.2. Agente Comunicación

```
package misprogramas.minominero; import jade.util.leap.*; import
jade.proto.*; import jade.lang.acl.*; import
jade.domain.JADEAgentManagement.*; import
jade.domain.mobility.MobilityOntology; import jade.domain.FIPANames;
import jade.content.lang.Codec; import jade.content.lang.sl.SLCodec;
import jade.core.Agent; import jade.core.AID; import
jade.core.behaviours.*; import jade.domain.FIPAAgentManagement.*;
import jade.domain.DFService; import jade.domain.FIPAException;
import jade.core.behaviours.CyclicBehaviour; import
jade.lang.acl.ACLMessage; import jade.lang.acl.MessageTemplate;
import jade.content.*; import jade.content.onto.*; import
jade.content.onto.basic.*; import jade.content.lang.*; import
jade.content.lang.sl.*; import misprogramas.minominero.ontologia.*;
import java.sql.*;

public class AgenteComunicacion extends Agent {
     private ContentManager manager     = (ContentManager)getContentManager();
     private Codec          codec       = new SLCodec(); //LEAPCodec
     private Ontology       ontology    = OntologiaMineria.getInstance();
     final Basedatos basedat = new Basedatos();
     public int ultimo_idpm;
     public int actual_idpm;
     public int tipocond;
     public String condicion;
     public String condicion2;
     public int soporte;
     public int confianza;

   public void setup() {
     try {
            // create the agent descrption of itself
            DFAgentDescription dfd = new DFAgentDescription();
            dfd.setName(getAID());
            // register the description with the DF
            DFService.register(this, dfd);
            System.out.println(getLocalName()+" registrado con el DF");
     }
     catch (FIPAException e) { e.printStackTrace(); }

     //Registrar el lenguaje y ontologia-mineria
     manager.registerLanguage(codec);
     manager.registerOntology(ontology);

     /*Conexion a la base de datos*/
     //basedat.ConectarBd(basededatos,usuario,contrasena);
     basedat.ConectarBd("mercado","root","ruben");

     //obtener el ultimo idp
     try{
         ultimo_idpm=basedat.Pregunta("select MAX(idpm) as res from mineria");
         System.out.println("ultimo idpm de base: "+ultimo_idpm);
     } //fin try
     catch(SQLException ex) { System.out.println(ex); }

     /*Verifica cada 5 segundos la tabla mineria*/
     addBehaviour(new TickerBehaviour(this,5000) {
        public void onTick() {
```

```
        try{
              actual_idpm=basedat.Pregunta("select MAX(idpm) as RES from mineria");
              //System.out.println("actual idpm: "+actual_idpm);
              //System.out.println("ultimo idpm: "+ultimo_idpm);
        } //fin try
        catch(SQLException ex) { System.out.println(ex); }

        //Verifica que se ha añadido un nuevo idpm
        while ( actual_idpm > ultimo_idpm  ){
          System.out.println("Se encontro un nuevo proceso de mineria:"+actual_idpm);

          //Obtencion de los parametros de mineria
          try{
              tipocond=basedat.Pregunta("Select tipocond as RES from mineria where idpm="+actual_idpm);
              condicion=basedat.PreguntaS("Select condicion as RES from mineria where idpm="+actual_idpm);
              condicion2=basedat.PreguntaS("Select condicion2 as RES from mineria where idpm="+actual_idpm);
              soporte=basedat.Pregunta("Select soporte as RES from mineria where idpm="+actual_idpm);
              confianza=basedat.Pregunta("Select confianza as RES from mineria where idpm="+actual_idpm);
          } //fin try

          catch(SQLException ex) { System.out.println(ex); }


          // Preparacion del mensaje REQUEST con contenido ACMINERIA
          //System.out.println( "[" + getLocalName() + "] Creando mensaje ACMINERIA");
          ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
          msg.setSender(getAID());
          msg.addReceiver(new AID("AgenteCoordinador", AID.ISLOCALNAME));
          msg.setLanguage(codec.getName());
          msg.setOntology(ontology.getName());

          //contenido del mensaje
          ultimo_idpm++;
          AcMineria miproc = new AcMineria(ultimo_idpm,"mercado","detalleticket",
          "root","ruben",tipocond,condicion,condicion2,soporte,confianza);
          Action a = new Action(getAID(), miproc);

          try{
              manager.fillContent(msg, a);
              //System.out.println("Mensaje AcMineria= "+msg);
          }
          catch (Codec.CodecException fe) {
              System.err.println("FIPAException in fill/extract Msgcontent:" + fe.getMessage());
          }
          catch (OntologyException fe) {
              System.err.println("OntologyException in getRoleName:" + fe.getMessage());
          }
          send(msg);
          System.out.println( "[" + getLocalName() + "] Se envio mensaje REQUEST-ACMINERIA al AgenteCoordinador");
          //this.doDelete();

     }//fin while
     ultimo_idpm=actual_idpm;
     }//fin onTick
 }); //fin comportamiento Ticker
}//fin setup

protected void takeDown() {
     // Deregister with the DF
          try {
              DFService.deregister(this);
              System.out.println(getLocalName()+" DEREGISTERED WITH THE DF");
          } catch (FIPAException e) {
```

```
            e.printStackTrace();
         }
    }//fin takedown
}//fin de clase
```

B.3. Agente Recolector

```
package misprogramas.minominero; import jade.core.Agent; import
jade.core.AID; import jade.domain.FIPAAgentManagement.*; import
jade.domain.DFService; import jade.domain.FIPAException; import
jade.core.behaviours.CyclicBehaviour; import
jade.lang.acl.ACLMessage; import jade.lang.acl.MessageTemplate;
import jade.core.*; import jade.core.behaviours.*; import
jade.content.*; import jade.content.abs.*; import
jade.content.onto.*; import jade.content.onto.basic.*; import
jade.content.lang.*; import jade.content.lang.sl.*; import
jade.util.leap.List; import jade.util.leap.ArrayList; import
jade.util.leap.Iterator; import java.sql.*; import
java.util.Calendar; import java.util.Date; import
misprogramas.minominero.ontologia.*;

public class AgenteRecolector extends Agent {
    private ContentManager manager     = (ContentManager)getContentManager();
    private Codec          codec       = new SLCodec(); //LEAPCodec
    private Ontology       ontology    = OntologiaMineria.getInstance();
    public Basedatos basedat = new Basedatos();
    public String cad;
    public int actual_idpm;
    public int ultimo_idpm;
    public int no_sitios;
    public List sitios_idpm;

    protected void setup() {
      manager.registerLanguage(codec);
      manager.registerOntology(ontology);
      System.out.println( "[" + getLocalName() + "] En espera de mensajes");

      try {
        // create the agent descrption of itself
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());
        // register the description with the DF
        DFService.register(this, dfd);
        System.out.println(getLocalName()+" registrado con el DF");
    } catch (FIPAException e) {
        e.printStackTrace();
    }

    /*Conexion a la base de datos*/
    basedat.ConectarBd("mercado","root","ruben");
    /*Comportamiento cíclico para recibir mensajes y calcular resultados*/
    addBehaviour(new CyclicBehaviour(this) {
        public void action() {
           //Escucha si llega un mensaje INFORM
           ACLMessage msg = receive(MessageTemplate.MatchPerformative(ACLMessage.INFORM));
           try{
```

```
 /*RECEPCION DE RESULTADOS PARCIALES DE MINERIA DE LOS SITIOS*/
 if (msg != null) {
    ContentElement p = manager.extractContent(msg);
    if(p instanceof ResMineria) {
       System.out.println("[" + getLocalName() + "] Recibi un mensaje RESMINERIA  de "
       +msg.getSender().getLocalName());
       ResMineria res_rec = (ResMineria)p;
       System.out.println("ReMineria ="+res_rec.toString());

       //Almacenar el resultado en la tabla Mineria
       try{
            basedat.Actualizar("update sitios_disp SET resmineria='"+res_rec.toString()
            +"' where idpm="+res_rec.getIdpm()+" and nom_sitio='"+res_rec.getContenedor()+"'");
            basedat.Actualizar("update sitios_disp SET band_result=1 where idpm="+res_rec.getIdpm()
            +" and nom_sitio='"+res_rec.getContenedor()+"'");
            basedat.Actualizar("update sitios_disp SET total_tickets="+res_rec.getTambd()
            +" where idpm="+res_rec.getIdpm()+" and nom_sitio='"+res_rec.getContenedor()+"'");
       }
       catch(SQLException ex) { System.out.println(ex); }
    }//resmineria
} //msgdifnull

/*CALCULO DE RESULTADOS DE MINERIA*/
try{
    actual_idpm= basedat.Pregunta("Select Min(idpm) as RES from sitios_disp where band_activo=1");
    ultimo_idpm= basedat.Pregunta("Select Max(idpm) as RES from sitios_disp where band_activo=1");


    if (actual_idpm==0){
        //System.out.println("No hay procesos activos");
        System.out.println( "[" + myAgent.getLocalName() + "] En espera de mensajes y/o procesos activos");
        block();
    }
    else {
    while (actual_idpm<=ultimo_idpm){ //para cada idpm
        List resultado_reglas = new ArrayList();
        List resultado_tambd = new ArrayList();
        List resultado_nositios = new ArrayList();
        int result_tambd;
        int temp;
        int cont_sitios_res;
        int s;
        int no_sitios_clonados;
        String hora_idpm;

        //Determinar si todos los sitios clonados ya mandaron su resultado
        no_sitios_clonados = basedat.Pregunta("Select Count(distinct nom_sitio) as RES"
        +" from sitios_disp where band_clono=1 and idpm="+actual_idpm);
        no_sitios = basedat.Pregunta("Select Count(distinct nom_sitio) as RES"
        +" from sitios_disp where band_result=1 and idpm="+actual_idpm);
        hora_idpm = basedat.PreguntaS("Select hora as RES from mineria where idpm="+actual_idpm);

//Forzar une espera en el calculo del analisis de dos minutos
if (ExcedioTiempo(hora_idpm,1)==false){
            //System.out.println("Espera forzada");
            actual_idpm++;
            continue;
 }
//Determinar si excedio el tiempo en 59 minutos maximo
if (ExcedioTiempo(hora_idpm,59)==false && no_sitios_clonados>no_sitios){
            //System.out.println("Para el idpm:"+actual_idpm);
            //System.out.println("NO Todos los sitios mandaron sus resultados");
            actual_idpm++;
```

```
            continue;
 }
if (ExcedioTiempo(hora_idpm,59) && no_sitios==0){
    //No se clono en ningun sitio
    System.out.println("Para el idpm:"+actual_idpm);
    System.out.println("Se excedio y no hay sitios");
    //Resultado nulo
    Regla regla_vacia = new Regla();
    resultado_reglas.add(regla_vacia);
}
else{
    if (no_sitios_clonados==no_sitios){
        //System.out.println("Para el idpm:"+actual_idpm);
        System.out.println("Todos los sitios mandaron sus resultados");
    }
    System.out.println("Para el idpm:"+actual_idpm);
    sitios_idpm= basedat.PreguntaL("Select nom_sitio as RES"
    +" from sitios_disp where band_result=1 and idpm="+actual_idpm);

    for (int sit=0; sit<no_sitios; sit++){ //Para cada sitio en el idpm_actual
        String primer_sitio = (String) sitios_idpm.get(sit);
        System.out.println(primer_sitio);
        cad=basedat.PreguntaS("Select resmineria as RES"
        +" from sitios_disp where idpm="+actual_idpm+" and nom_sitio='"+primer_sitio+"'");
        int tambd_regla = ObtenerTambd(cad);
        result_tambd=tambd_regla;

        List reglas_primersitio = ObtenerReglas(cad);

        //Para cada regla buscarla en todos los demas sitios
        if (no_sitios==1){
            System.out.println("resultado unico");

            if (reglas_primersitio != null) {
                Iterator it = reglas_primersitio.iterator();
                while (it.hasNext()) {
                    Regla reglax = (Regla) it.next();
                    reglax.setTambd(result_tambd);
                    reglax.setNumsitios(1);
                    resultado_reglas.add(reglax);
                }
            }
        }
        else{ //si hay varios sitios
                int total_reglas = reglas_primersitio.size();
                System.out.println("Total reglas="+total_reglas);
                for (int r=0; r<total_reglas; r++){
                    Regla regla_buscar = (Regla) reglas_primersitio.get(r);
                    System.out.println("Buscar "+regla_buscar.toString());
                    if (regla_buscar.getSoportea()==0){
                        System.out.println("Regla vacia");
                        continue;
                    }
                    if (Contiene(resultado_reglas,regla_buscar)>=0 ){
                            System.out.println("Regla ya calculada");
                            continue;
                    }
                    result_tambd=tambd_regla;
                    System.out.println("Los demas sitios");
                    cont_sitios_res = 0;
                    int contador=0;
                    s=sit+1; //sit=1
                    if (s==no_sitios) s=0;
```

```
                        while (s<no_sitios && contador<no_sitios-1){
                            String sitiox = (String) sitios_idpm.get(s);
                            System.out.println(sitiox);
                            cad=basedat.PreguntaS("Select resmineria as RES"
                            +" from sitios_disp where idpm="+actual_idpm+" and nom_sitio='"+sitiox+"'");
                            List reglas_sitiox = ObtenerReglas(cad);
                            int indice=Contiene(reglas_sitiox,regla_buscar);
                            if (indice>=0){
                                System.out.println("Regla encontrada");
                                Regla reglax = (Regla) reglas_sitiox.get(indice);

                                temp = reglax.getSoporteab()+regla_buscar.getSoporteab();
                                regla_buscar.setSoporteab(temp);
                                temp = reglax.getSoportea()+regla_buscar.getSoportea();
                                regla_buscar.setSoportea(temp);
                                result_tambd+=ObtenerTambd(cad);
                                cont_sitios_res++;
                            }
                            else{
                                System.out.println("Regla no encontrada");
                            }
                            s++;
                            if (s==no_sitios) s=0;
                            contador++;
                        }//while sitios

                        if ( (cont_sitios_res+1) >= Math.round(no_sitios*0.50f)){
                            regla_buscar.setTambd(result_tambd);
                            regla_buscar.setNumsitios(cont_sitios_res+1);
                            resultado_reglas.add(regla_buscar);
                        }
                    }//for reglas
                }//fin sihayvariossitios
            }//finforparacadasitio
        }//fin excediotiempo

            //Presentar el resultado final
            System.out.println("RESULTADO para el idpm:"+actual_idpm);
            String miresultado = "Reglas: ";
            if (resultado_reglas != null) {
                Iterator it = resultado_reglas.iterator();
                while (it.hasNext()) {
                    Regla reglaRes = (Regla) it.next();
                    System.out.println("Regla = "+ reglaRes.toStringRes());
                    miresultado=miresultado+reglaRes.toStringRes()+" ";
                }
            }
            //Guardar el resultado en la tabla mineria
            basedat.Actualizar("Update mineria SET resmineria='"+miresultado+"' where idpm="+actual_idpm);
            int numcloncontesta=basedat.Pregunta("Select count(nom_sitio) as RES"
            +" from sitios_disp where band_clono=1 and idpm="+actual_idpm);
            basedat.Actualizar("Update mineria SET num_sitios_clon="+numcloncontesta+" where idpm="+actual_idpm);
            //Actualizar la band_activo = 0
            basedat.Actualizar("Update sitios_disp SET band_activo=0 where idpm="+actual_idpm);

            //Obtener el tiempo del sistema y guardarlo en la tabla restiempo
            String hora;
            Calendar hoy = Calendar.getInstance();
            int hor = hoy.get(Calendar.HOUR_OF_DAY);
            int min = hoy.get(Calendar.MINUTE);
            int seg = hoy.get(Calendar.SECOND);
    //System.out.println("Hora Actual= "+hor+":"+min+":"+seg);
            hora=hor+":"+min+":"+seg;
```

```
                basedat.Actualizar("Insert into restiempo values ("+actual_idpm+",'"+hora+"')");
                actual_idpm++;
                }//fin while sitios
     }//fin si no hay procesos activos
           }
           catch(SQLException ex) { System.out.println(ex); }
            } catch (Codec.CodecException fe) {
                System.err.println("FIPAException in fill/extract Msgcontent:" + fe.getMessage());
              }
              catch (OntologyException fe) {
                System.err.println("OntologyException in getRoleName:" + fe.getMessage());
              }
        }//fin de action
   }); //fin del comportamientocliclico
   }//fin de setup

   protected void takeDown() {
        // Deregister with the DF
            try {
                DFService.deregister(this);
                System.out.println(getLocalName()+" DEREGISTERED WITH THE DF");
            } catch (FIPAException e) {
                e.printStackTrace();
            }
            basedat.CerrarBd();
   }//fin takedown

   public Integer ObtenerIdpm(String cad){
        StringBuffer cad_temp= new StringBuffer();
        int i=6;
        while (cad.charAt(i) != ' '){
             cad_temp.append(cad.charAt(i));
             i++;
        }
        int idpm= new Integer(cad_temp.toString());
        return idpm;
}

public String ObtenerSitio(String cad){
       StringBuffer cad_temp= new StringBuffer();
       int i=cad.indexOf("Sitio:");
       i+=7;
       while (cad.charAt(i) != ' '){
           cad_temp.append(cad.charAt(i));
           i++;
       }
       String sitio = cad_temp.toString();
       return sitio;
}

public Integer ObtenerTambd(String cad){
       StringBuffer cad_temp= new StringBuffer();
       int i=cad.indexOf("Tambd:");
       i+=7;
       while (i<cad.length()){
            cad_temp.append(cad.charAt(i));
            i++;
       }
       int tam = new Integer(cad_temp.toString());
       return tam;
}
```

```
public List ObtenerReglas(String cad){
       List misreglas = new ArrayList();
       Regla reglaX;
       List conj_a;
       List conj_b;
       Articulo miart;
       long temp;

       int i=cad.indexOf("Reglas:");
       i+=8;
       int j = cad.indexOf("Tambd:");

       while (i<j){
            if (cad.charAt(i) == '['){
               reglaX = new Regla();
               i+=2;
               conj_a = new ArrayList();
               //construccion del conj_a
               while (cad.charAt(i) != '}'){
                   StringBuffer cad_temp4= new StringBuffer();

                   while (cad.charAt(i) != ' '){
                       cad_temp4.append(cad.charAt(i));
                       i++;
                   }
                   temp = new Long(cad_temp4.toString());
                   miart = new Articulo();
                   miart.setId(temp);
                   conj_a.add(miart);
                   i++;
               }

               //construccion del conj_a
               conj_b = new ArrayList();
               i+=2;
               while (cad.charAt(i) != '}'){
                   StringBuffer cad_temp5= new StringBuffer();

                   while (cad.charAt(i) != ' '){
                       cad_temp5.append(cad.charAt(i));
                       i++;
                   }
                   temp = new Long(cad_temp5.toString());
                   miart = new Articulo();
                   miart.setId(temp);
                   conj_b.add(miart);
                   i++;
               }
               //obtencion del soporteab
               StringBuffer cad_temp6= new StringBuffer();
               i+=13;
               while (cad.charAt(i) != ' '){
                   cad_temp6.append(cad.charAt(i));
                   i++;
               }
               int sopab = new Integer(cad_temp6.toString());

               //obtencion del soportea
               StringBuffer cad_temp7= new StringBuffer();
               i+=11;
               while (cad.charAt(i) != ']'){
                   cad_temp7.append(cad.charAt(i));
                   i++;
```

```
                }
                int sopa = new Integer(cad_temp7.toString());

                reglaX.setConjuntoa(conj_a);
                reglaX.setConjuntob(conj_b);
                reglaX.setSoporteab(sopab);
                reglaX.setSoportea(sopa);
                misreglas.add(reglaX);
                i+=2;//para el sig [
             }//fin if [
             else{
                break;
             }
        }//fin while
        return misreglas;

}

//Contiene lista donde buscar, regla a buscar
public Integer Contiene(List reglas, Regla miregla){
   if (reglas != null) {
        Iterator it = reglas.iterator();
        int i = 0;
        while (it.hasNext()) {
            Regla reglax = (Regla) it.next();
            if (reglax.Igual(miregla)){
                return i;
            }
            i++;
        }
   }
   return (-1);
}

public static boolean ExcedioTiempo(String hora, int minutos){
   int dif_hora=0, dif_min=0, dif_seg=0;
   //Obtener el tiempo del sistema
   Calendar hoy = Calendar.getInstance();
   Date mihoy = hoy.getTime();
   int hora1 = hoy.get(Calendar.HOUR_OF_DAY);
   int min1 = hoy.get(Calendar.MINUTE);
   int seg1 = hoy.get(Calendar.SECOND);
   //Hora del idpm
   StringBuffer cad_temp1= new StringBuffer();
   int i=0;
   while (hora.charAt(i) != ':'){
        cad_temp1.append(hora.charAt(i));
        i++;
   }
   int hora2 = new Integer(cad_temp1.toString());
   StringBuffer cad_temp2= new StringBuffer();
   i++;
   while (hora.charAt(i) != ':'){
        cad_temp2.append(hora.charAt(i));
        i++;
   }
   int min2 = new Integer(cad_temp2.toString());
   StringBuffer cad_temp3= new StringBuffer();
   i++;
   while (i<hora.length()){
         cad_temp3.append(hora.charAt(i));
         i++;
   }
```

```
    int seg2 = new Integer(cad_temp3.toString());
    if (seg2>seg1){
        dif_seg = 60-seg2+seg1;
        min2++;
        if (min2==60){
            min2=0;
            hora2++;
            if (hora2==24)  hora2=0;
        }
    }
    else{
        dif_seg = seg1-seg2;
    }
    if (min2>min1){
        dif_min = 60-min2+min1;
        hora2++;
        if (hora2==24)  hora2=0;
    }
    else{
        dif_min = min1-min2;
    }
    if (hora1<hora2)
        dif_hora = 24-hora2+hora1;
    else
        dif_hora = hora1-hora2;
    if (dif_hora>0)
        return true;
    if (dif_hora==0 && dif_min>=minutos)
        return true;
    return false;
  }//fin excediotiempo

}
```

B.4. Agente Minero

```
package misprogramas.minominero; import java.util.Vector; import
java.awt.*; import java.awt.event.*; import javax.swing.*; import
java.util.Iterator; import jade.core.Location; import
jade.core.behaviours.*; import jade.domain.mobility.*; import
jade.domain.FIPANames; import jade.content.lang.Codec; import
jade.content.lang.sl.SLCodec; import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate; import javax.swing.border.*;
import javax.swing.table.*; import java.util.*; import java.io.*;
import jade.core.*; import jade.domain.mobility.*; import
misprogramas.minominero.ontologia.*; import jade.util.leap.List;
import jade.util.leap.ArrayList; import jade.content.*; import
jade.content.abs.*; import jade.content.onto.*; import
jade.content.onto.basic.*; import jade.content.lang.*; import
jade.content.lang.sl.*;

public class AgenteMinero extends Agent {
  public ContentManager manager     = (ContentManager)getContentManager();
  public Codec          codec       = new SLCodec(); //LEAPCodec
  public Ontology       ontology    = OntologiaMineria.getInstance();
```

```
  public int mi_idpm;
  public String basedatos;
  public String tabla;
  public String usuario;
  public String contrasenia;
  public int tipocond;
  public String condicion;
  public String condicion2;
  public int soporte;
  public int confianza;

  public void setup() {
      //Registrar el lenguaje de contenido de la ontologia, llamado SL0
        getContentManager().registerLanguage(new SLCodec(), FIPANames.ContentLanguage.FIPA_SL0);
      //Registrar la ontologia movil
        getContentManager().registerOntology(MobilityOntology.getInstance());
      //Registrar el lenguaje y ontologia-mineria
        manager.registerOntology(ontology);

        /*Aqui se reciben los argumentos y se ubican en la variable correspondiente*/
      Object[] args = getArguments();
      if (args != null && args.length > 0){
        mi_idpm = (Integer) args[0];
        basedatos = (String) args[1];
        tabla = (String) args[2];
        usuario=(String) args[3];
        contrasenia=(String) args[4];
        tipocond=(Integer) args[5];
        condicion=(String) args[6];
        condicion2=(String) args[7];
        soporte=(Integer) args[8];
        confianza=(Integer) args[9];
        //agrega una tarea para buscar los contenedores existentes (un contenedor por sitio)
        addBehaviour(new ObtenerLocalidades(this,mi_idpm));
       }
       else{
            System.out.println("No se recibieron los argumentos necesarios");
       }
   }

    public void takeDown() {
          System.out.println(getLocalName()+" ha terminado");
    }

      protected void beforeClone() {
        System.out.println(getLocalName()+" iniciara el proceso de clonacion.");
      }

      protected void afterClone() {
       System.out.println(getLocalName()+" arribo a esta localidad, se debe iniciar el proceso de mineria.");
       getContentManager().registerLanguage(new SLCodec(), FIPANames.ContentLanguage.FIPA_SL0);
       getContentManager().registerOntology(MobilityOntology.getInstance());
       this.addBehaviour(new AsociaciondeReglas(
       mi_idpm,usuario,contrasenia,basedatos,tabla,tipocond,condicion,condicion2,soporte,confianza));
     }

      protected void afterLoad() {
                super.afterLoad();
                afterClone();
      }
}//fin de la clase MobilMiner

public class ObtenerLocalidades extends SimpleAchieveREInitiator {
```

```
private ACLMessage request;
public int mi_idpm;

public ObtenerLocalidades(AgenteMinero a, int idpm) {
  // Llamada al constructor FipaRequestInitiatorBehaviour
  super(a, new ACLMessage(ACLMessage.REQUEST));
  mi_idpm=idpm;
  request = (ACLMessage)getDataStore().get(REQUEST_KEY);

  request.clearAllReceiver();
  request.addReceiver(a.getAMS());
  request.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
  request.setOntology(MobilityOntology.NAME);
  request.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);

  try {
    Action action = new Action();
    action.setActor(a.getAMS());
    action.setAction(new QueryPlatformLocationsAction());
    a.getContentManager().fillContent(request, action);
  }
  catch(Exception fe) {
       fe.printStackTrace();
  }
  reset(request);
}

protected void handleNotUnderstood(ACLMessage reply) {
  System.out.println(myAgent.getLocalName()+ " handleNotUnderstood : "+reply.toString());
}

protected void handleRefuse(ACLMessage reply) {
  System.out.println(myAgent.getLocalName()+ " handleRefuse : "+reply.toString());
}

protected void handleFailure(ACLMessage reply) {
  System.out.println(myAgent.getLocalName()+ " handleFailure : "+reply.toString());
}

protected void handleAgree(ACLMessage reply) {
}

protected void handleInform(ACLMessage inform) {
  String content = inform.getContent();
  System.out.println("Obtener localidades para: "+myAgent.getLocalName()+" con idpm:"+mi_idpm);
  final String AgenteMineroOriginal = myAgent.getLocalName();

  try {
    Result results = (Result)myAgent.getContentManager().extractContent(inform);
 int contador=1;
 Location nextSite;
 Iterator sitios;
 sitios=results.getItems().iterator();
 SequentialBehaviour sb = new SequentialBehaviour(){
     public int onEnd() {
     reset();
         return super.onEnd();
     } };

 /*Conexion a la base de datos*/
 Basedatos basedat = new Basedatos();
 basedat.ConectarBd("mercado","root","ruben");
```

```
    for ( contador=1; sitios.hasNext(); contador++) {
        Object obj = sitios.next();
        nextSite =(Location)obj;
        System.out.println("Sitio "+contador+" :"+nextSite.getName());
        final int conta = contador;  //variables final no se les puede asignar valor solo en su declaracion
        final Location siguiente = nextSite;
        final String AgenteOriginal = myAgent.getLocalName();

        /* Se quitar el sitio donde se creo el agente original*/
        if (nextSite.getName().equals(myAgent.here().getName()) ){
           //System.out.println("Es el contenedor del original:"+myAgent.here().getName());
           contador--;
           continue;
        }

        //Guardar el sitio en la tabla sitios_disp
        try{
           basedat.Actualizar("insert into sitios_disp values ("+mi_idpm+",'"+nextSite.getName()+"',0,0,1,null,0)");
        }
        catch(SQLException ex) { System.out.println(ex); }
        sb.addSubBehaviour(new OneShotBehaviour(myAgent) {

        public void action() {
            if(myAgent.getLocalName().equals(AgenteOriginal)) {
                    System.out.println("Clonar "+myAgent.getLocalName()+"  a:"+siguiente.getName()+" Contador: "+conta);
                ((AgenteMinero)myAgent).doClone(siguiente,"Agente"+AgenteOriginal+"_"+conta);
            }
            else {
                System.out.println("Son diferentes");
                }

             } //fin de action

            } );   //fin del comportamiento sb
       }//fin for

        //mandar el numero de clones
        int numclones = contador-1;
        System.out.println("numero de clones"+numclones);
        //Almacenar el numero de sitios disponibles para la clonacion en la tabla Mineria
        try{
            basedat.Actualizar("Update mineria SET num_sitios_disp="+numclones+" where idpm="+mi_idpm);
        }catch(SQLException ex) { System.out.println(ex); }
        basedat.CerrarBd();

    sb.addSubBehaviour(new OneShotBehaviour(myAgent) {
          public void action() {
             if(myAgent.getLocalName().equals(AgenteMineroOriginal)) {
                System.out.println("Proceso de Clonacion CONCLUIDO");
                myAgent.doDelete();
         }
          }
        });
     myAgent.addBehaviour(sb);
     //Al terminar el comportamiento sb se ejecuta su funcion OnEnd
     }//fin try
     catch(Exception e) {
       e.printStackTrace();
     }
   }//fin de handleinform
}

class AsociaciondeReglas extends Behaviour {
```

```
public static String ALERTA = "ALERTA";
private int mi_idpm;
private String usuario;
private String password;
private String basededatos;
private String tabla;
private int tipocond;
private String condicion;
private String condicion2;
private int min_sop_orig;
private int min_confianza;

public AsociaciondeReglas (int idpm,String user,String pwd,String bd,String tab,int tipoc,
String cond,String cond2,int sop,int conf) {
    mi_idpm=idpm;
    usuario=user;
    password=pwd;
    basededatos=bd;
    tabla=tab;
    tipocond=tipoc;
    condicion=cond;
    condicion2=cond2;
    min_sop_orig = sop;
    min_confianza = conf;
}

public void action(){
     //Registrar el lenguaje y ontologia-mineria
     myAgent.getContentManager().registerLanguage(new SLCodec());
     myAgent.getContentManager().registerOntology(OntologiaMineria.getInstance());

     // Preparacion del mensaje INFORM con contenido CLONCONFIRMA
     ACLMessage msgconf = new ACLMessage(ACLMessage.INFORM);
     msgconf.setSender(myAgent.getAID());
     msgconf.addReceiver(new AID("AgenteCoordinador", AID.ISLOCALNAME));
     msgconf.setLanguage((new SLCodec()).getName());
     msgconf.setOntology(OntologiaMineria.getInstance().getName());
     ClonConfirma miconfirma = new ClonConfirma(mi_idpm,myAgent.here().getName());
     try{
         myAgent.getContentManager().fillContent(msgconf, miconfirma);
     }
     catch (Codec.CodecException fe) {
             System.err.println("FIPAException in fill/extract Msgcontent:" + fe.getMessage());
     }
     catch (OntologyException fe) {
             System.err.println("OntologyException in getRoleName:" + fe.getMessage());
     }

     myAgent.send(msgconf);
     System.out.println( "[" + myAgent.getLocalName() + "] Envio mensaje INFORM-CLONCONFIRMA");

 System.out.println("soporte: "+min_sop_orig+"  y confianza: "+min_confianza);
 System.out.println("basedatos: "+basededatos+"  y tabla: "+tabla);
 /*Creacion del agenteMonitor*/
 try {
     /*CREACION DEL AGENTE DE MONITOREO*/
     // Toma una instancia en tiempo de ejecucion JADE
     Runtime rt = Runtime.instance();

     // crea un enlace al contenedor local
     AgentContainer ac = myAgent.getContainerController();

     //Crea el agente Monitor de este agenteMinero
```

```
        final String agentepropietario = myAgent.getLocalName();
        Object args[] = new Object[8];
        args[0] = agentepropietario;
        args[1] = usuario;
        args[2] = password;
        args[3] = basededatos;
        args[4] = tabla;
        args[5] = tipocond;
        args[6] = condicion;
        args[7] = condicion2;

        AgentController agente1 = ac.createNewAgent("AgenteMonitorde"+agentepropietario,
        "misprogramas.minominero.AgenteMonitor",args);
        agente1.start();

        /*TAREA DE MINERIA DE DATOS*/
        System.out.println("Metodo Apriori para Mineria de datos");
        Basedatos basedat = new Basedatos();
        int total_tick = 0;
        ListaL L= new ListaL();                //Lista de L[k]
        int k=1, finalizar=0;
        ListaConjuntosL punteroL;
        ListaConjuntosL mezclada;
        ListaConjuntosL descartados;// = new ListaConjuntosL();
        int regreso;
        int min_soporte;

        /** Conectarse a la Base de Datos que esta en Mysql **/
        basedat.ConectarBd(basededatos,usuario,password);

    do{
        regreso=0;
        k=1;
        finalizar=0;
        //Encontrar L1 que es la lista de articulos con conjuntos tamanio 1
        //y contando el numero de apariciones de cada uno

        //Busca en toda la base de datos para
        //generar pares conjunto(articulos)-totalapariciones
        L.ConjuntoBase(tabla,basedat,tipocond,condicion);
        /** Obtener el total de tickets de la base de datos **/
        try{
           //total_tick = basedat.ContarTickets("detalleticket");
           total_tick = basedat.ContarTickets("detalleticket",condicion2);
        }
        catch(SQLException ex) { System.out.println("Excepcion de Contartickets:"+ex);  }

        //calcular el minimo soporte proporcional al tamaño de la base de datos
        min_soporte = Math.round(total_tick*min_sop_orig/100.00f); //round
        System.out.println("min soporte recalculado="+min_soporte);

        descartados = new ListaConjuntosL();
        punteroL = L.PosicionL(k);
        punteroL.ContarSoporte(tabla,basedat,condicion2);
        //Elimina de la lista los articulos que no cumplen con el soporte minimo establecido
        punteroL.EliminarConjuntos(min_soporte);
        System.out.println("Conjunto Base L(1)= "+punteroL.MostrarLista());

        k=2;
        //Hasta que una lista quede vacia por no cumplir con el soporte
        //o que tenga un unico elemento que si cumple con el soporte
        while (finalizar==0)
        {
```

```
        punteroL = L.PosicionL(k-1);
        System.out.println("L("+(k-1)+")= "+punteroL.MostrarLista());

        //Crea una nueva lista mezclando los conjuntos de L[k-1] (c2) que cumplen con el mínimo soporte establecido
        mezclada= punteroL.MezclarApriori();
        System.out.println("Despues de Mezclar LONGITUD L("+k+")= "+mezclada.Longitud());

        mezclada.EliminarIguales();

        //Elimina los conjuntos que tienen subconjuntos que fueron
        //descartados por no cumplir con el soporte
        mezclada.ConjuntosYaDescartados(descartados);

        //Obtener soportes solo de los conjuntos potenciales al quitar los descartados
        //Contar cuantas veces aparece un conjunto en la base de datos
        mezclada.ContarSoporte(tabla,basedat,condicion2);
        System.out.println("Con soporte L("+k+")= "+mezclada.MostrarLista());
        System.out.println("Despues de contar soporte LONGITUD L("+k+")= "+mezclada.Longitud());
        if (mezclada.Longitud()==0){
            finalizar=1;
            System.out.println("Lista vacia, se utilizara el L("+(k-1)+")");
            break;
        }
        System.out.println("Con soporte L("+k+")= "+mezclada.MostrarLista());
        descartados=mezclada.EliminarConjuntos2(min_soporte);
        System.out.println("Despues de eliminar conjuntos LONGITUD L("+k+")= "+mezclada.Longitud());

        //Termina si L[k] solo tiene un conjunto
        //o L[k] esta vacio pues ningun conjunto cumplio con el min soporte
        if (mezclada.EsVacio())
        {   finalizar=1;
            System.out.println("vacio");
            break;
        }

        L.InsertarFinal(mezclada);
        k++;

        if (mezclada.Longitud()==1)
        {   finalizar=1;
            break;
        }

        /*REVISA SI HAY UN MENSAJE DE ALERTA*/
        //En un subproceso de mineria se Escucha
        //si llega un mensaje ALERTA para reiniciar el proceso
        ACLMessage msg = myAgent.receive(MessageTemplate.MatchPerformative(ACLMessage.INFORM));
        if (msg != null) {
            if (ALERTA.equalsIgnoreCase(msg.getContent())) {
                // Si el mensaje que llegó fue ALERTA entonces reinicia el proceso de mineria
                System.out.println(myAgent.getLocalName()+" recibio una ALERTA de: "+msg.getSender().getLocalName());
                System.out.println("Se reiniciara el proceso de mineria.....");
                //Para reiniciar el proceso
                regreso=1;
                finalizar=1;
            }
        }
        else {
            System.out.println("No llego ningun mensaje y se continua el proceso");
            regreso=0;
        }

    }//fin while
```

```
} while (regreso==1);//fin while regreso

    basedat.CerrarBd();
    ListaConjuntosL ListaFinal= new ListaConjuntosL(); //L.Ultimo(),null);
    ListaFinal = L.Ultimo();
    System.out.println("Lista final = "+ListaFinal.MostrarLista());
    System.out.println("Longitud Lista final = "+ListaFinal.Longitud());

    /******** Generacion de reglas a partir de la ListaFinal *********/
    //variables para generar reglas
    int nart, nconjuntos, sop_l, sop_s, longtemp;
    int i,j;
    String cadena;
    ConjuntoL punteroC;
    ConjuntoL punteroCtemp;
    ListaConjuntosL punteroLista;
    ConjuntoL c;
    ConjuntoL conjunto_l;
    ConjuntoL conjunto_s;
    ConjuntoL conjunto_l_s;
    //variables para mensaje
    ResMineria mires = new ResMineria();
    List misreglas = new ArrayList();
    Regla reglaX;
    List conj_a;
    List conj_b;
    Articulo miart;

//verificar si hay un conjunto en la lista final
if (ListaFinal.Longitud()<1){
    System.out.println("NO SE OBTUVO NINGUN CONJUNTO PARA GENERAR REGLAS");
    mires.setIdpm(mi_idpm);
    mires.setContenedor(myAgent.here().getName());
}
else{

    //Inicio del proceso de generacion de reglas
    nconjuntos=ListaFinal.Longitud();
    punteroC = ListaFinal.PosicionC(1);
    nart=punteroC.Longitud();
    System.out.println("LONGITUD de los conjuntos= "+punteroC.Longitud());
  if (punteroC.Longitud()>1){
    for (i=1; i<=nconjuntos; i++){
        //Elige un conjunto de ListaFinal(i)
        punteroC = ListaFinal.PosicionC(i);

        //Generacion de subconjuntos del ListaFinal(i)
        for (j=1; j<((int)(Math.pow(2,nart)))-1; j++)
        {
          cadena=Integer.toBinaryString(j);
          while (cadena.length()<nart)
          {
          cadena="0"+cadena; //deja la cadena con longitud k
          }

          System.out.println("Numero binario="+cadena);

          //crear el subconjuntos tomando el articulo representado con un 1
          c = new ConjuntoL();
          for (int z=0; z<nart; z++){
             if (cadena.charAt(z)=='1') {
                c.InsertarFinal(punteroC.MostrarContenidoArticulo(z+1));
```

```
   }
}

/** Buscar soporte de conjunto s**/
longtemp = c.Longitud();
punteroLista = L.PosicionL(longtemp);
//Buscar en L de tamanio longtemp el conjunto s (llamado c) y obtener su soporte
punteroCtemp = punteroLista.BuscarConjunto(c);
sop_s = punteroCtemp.MostrarSoporte();
c.NuevoSoporte(sop_s);
System.out.println("Conjunto "+j+" ("+longtemp+"): "+c.MostrarConjunto());
conjunto_s=c;
/** Buscar soporte de conjunto l **/
longtemp = punteroC.Longitud();
punteroLista = L.PosicionL(longtemp);
//Buscar en L de tamanio longtemp el conjunto l y obtener su soporte
punteroCtemp = punteroLista.BuscarConjunto(punteroC);
sop_l = punteroCtemp.MostrarSoporte();
conjunto_l=punteroC;
/** Obtener el conjunto complemento l-s **/
conjunto_l_s= conjunto_l.ConjuntoComplemento(conjunto_s);
System.out.println("Regla numero "+j);
System.out.println(conjunto_s.MostrarConjunto()+" => "+conjunto_l_s.MostrarConjunto());

/** Imprimir regla y su confianza**/
System.out.println("Soporte de l = "+sop_l);
System.out.println("Soporte de s = "+sop_s);

double confianzaR= (sop_l/(sop_s*1.0))*10000;
double soporteR=(sop_l/(total_tick*1.0))*10000;  //previamente se obtuvo total_tick
Float f = new Float(confianzaR);
confianzaR=f.intValue();
confianzaR=confianzaR/100;
f = new Float(soporteR);
soporteR=f.intValue();
soporteR=soporteR/100;
System.out.println("Confianza de la regla = "+confianzaR);
System.out.println("Soporte de la regla = "+soporteR);

//contruccion de RESMINERIA :reglaX
conj_a = new ArrayList();
conj_b = new ArrayList();
for (int y=1; y<=conjunto_s.Longitud(); y++){
  miart = new Articulo();
  miart.setId(conjunto_s.MostrarContenidoArticulo(y));
  conj_a.add(miart);
}
for (int y=1; y<=conjunto_l_s.Longitud(); y++){
  miart = new Articulo();
  miart.setId(conjunto_l_s.MostrarContenidoArticulo(y));
  conj_b.add(miart);
}
reglaX = new Regla();
reglaX.setConjuntoa(conj_a);
reglaX.setConjuntob(conj_b);
reglaX.setSoporteab(sop_l);
reglaX.setSoportea(sop_s);
//System.out.println("REGLAX="+reglaX.toString());

if (confianzaR<min_confianza) {
  System.out.println("Regla no cumplio confianza");
}
else
```

```
            {
              //Añadir al mensaje esta regla porque cumple con la confianza
              misreglas.add(reglaX);
            }

          }//fin for generar subconjuntos
        }//fin for para cada conjunto
    }//fin if longitud>1

        //contruccion de RESMINERIA (idpm,reglas,tambd)
        mires.setIdpm(mi_idpm);
        mires.setContenedor(myAgent.here().getName());
        if (misreglas.isEmpty() || punteroC.Longitud()<=1 ){
            System.out.println("NO SE GENERO NINGUNA REGLA");
        }
        else{
            mires.setReglas(misreglas);
        }

    }
        mires.setTambd(total_tick);
        System.out.println("RESMINERIA="+mires.toString());

        /*Enviar los resultados al AgenteRecolector*/
        // Preparacion del mensaje INFORM con contenido RESMINERIA
        System.out.println( "[" + myAgent.getLocalName() + "] Creando mensaje INFORM");
        ACLMessage msgres = new ACLMessage(ACLMessage.INFORM);
        msgres.setSender(myAgent.getAID());
        msgres.addReceiver(new AID("AgenteRecolector", AID.ISLOCALNAME));
        msgres.setLanguage((new SLCodec()).getName());
        msgres.setOntology(OntologiaMineria.getInstance().getName());
        try{
            //System.out.println("Mensaje resmineria= "+msgres);
            myAgent.getContentManager().fillContent(msgres, mires);
        }
        catch (Codec.CodecException fe) {
                System.err.println("FIPAException in fill/extract Msgcontent:" + fe.getMessage());
        }
        catch (OntologyException fe) {
                System.err.println("OntologyException in getRoleName:" + fe.getMessage());
        }

        myAgent.send(msgres);
        System.out.println( "[" + myAgent.getLocalName() + "] Se envio mensaje INFORM-RESMINERIA");
        System.out.println("mensaje de RESULTADOS enviado");
        agente1.kill();
    }//fin try
    catch(Exception e) { System.out.println("Excepcion principal de AsociacionReglas");e.printStackTrace(); }
   }//fin action

   public boolean done(){
        myAgent.doDelete();
    return true; //true=fin del comportamiento
   }
} //fin de clase
```

B.5. Agente Monitor

```
package misprogramas.minominero; import jade.core.Agent; import
jade.core.behaviours.*; import jade.core.behaviours.TickerBehaviour;
import jade.core.behaviours.OneShotBehaviour; import java.sql.*;
import jade.core.AID; import jade.domain.FIPAAgentManagement.*;
import jade.domain.DFService; import jade.domain.FIPAException;
import jade.core.behaviours.CyclicBehaviour; import
jade.lang.acl.ACLMessage; import jade.lang.acl.MessageTemplate;

public class AgenteMonitor extends Agent {
    public int umbral=10;
    public String basededatos;
    public String nomtabla;
    public String usuario;
    public String contrasena;
    public int tipocond;
    public String condicion;
    public String condicion2;
    public int tamoriginal;
    public int tamactual;
    final Basedatos basedat = new Basedatos();
    public static String ALERTA = "ALERTA";

    protected void setup(){

      /*Aqui se reciben los argumentos y se ubican en la variable correspondiente*/
      Object[] args = getArguments();
      if (args != null && args.length > 0){
        final String agentepropietario = (String) args[0];
        usuario=(String) args[1];
        contrasena=(String) args[2];
        basededatos = (String) args[3];
        nomtabla = (String) args[4];
        tipocond=(Integer) args[5];
        condicion=(String) args[6];
        condicion2=(String) args[7];

    System.out.println("Propietario:"+agentepropietario+" BD:"+basededatos+" Tabla:"+nomtabla);
    System.out.println(getLocalName()+" habilitado para VERIFICACION DE LA BD");

    /*El agenteMonitor se registra en el DF*/
    try {
        // create the agent descrption of itself
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());
        // register the description with the DF
        DFService.register(this, dfd);
        System.out.println(getLocalName()+" registrado con el DF");
    } catch (FIPAException e) {
        e.printStackTrace();
    }

    /*Conexion a la base de datos*/
    basedat.ConectarBd(basededatos,usuario,contrasena);

    /*Primero verifica la base de datos y obtiene el tamaño original*/
    //Obtener el tamaño original de la base de datos
    try{
```

```
            tamoriginal=basedat.ContarTickets(nomtabla,condicion2);
         System.out.println("tamanio de la BD: "+tamoriginal);
    } //fin try
    catch(SQLException ex) {
        System.out.println("Excepcion de AgenteMonitor:"+ex);
        basedat.CerrarBd();
        this.doDelete();
    }

    /*Verifica cada 9 segundos el tamaño de la base de datos */
    final String nomtab = nomtabla;
    addBehaviour(new TickerBehaviour(this,9000) {
        public void onTick() {
        //Obtener el tamaño actual de la base de datos
        //tamactual = (int) (tamorig*1.20);
        try{
                tamactual=basedat.ContarTickets(nomtab,condicion2);//tipocond,condicion);
             System.out.println("tamanio actual de la BD: "+tamactual);
        } //fin try
        catch(SQLException ex) { System.out.println("Excepcion de AgenteMonitor(ContarTickets):"+ex); }

        //Verifica que el tamaño actual no exceda el umbral
        //del tamaño original
        if ( tamactual > (int)(tamoriginal*(1+umbral/100)) ){
            System.out.println("Se excedio el tamanio:"+tamoriginal+" a "+tamactual);
            //retomando el nuevo valor
            tamoriginal=tamactual;

            /*Enviar los resultados al AgenteRecolector*/
            ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
            msg.setContent(ALERTA);
            msg.addReceiver(new AID(agentepropietario, AID.ISLOCALNAME));
            myAgent.send(msg);
            System.out.println("mensaje de Alerta enviado");

        }
        else {
            System.out.println("Se mantiene el tamanio:"+tamoriginal+" a "+tamactual);
        }
        }//fin onTick
    }); //fin comportamiento Ticker
      }
      else {
    System.out.println("NO HAY ARGUMENTOS, No se creo el Agente Monitor");
    doDelete();
      }
    }//fin de setup

    protected void takeDown() {
    basedat.CerrarBd();
            try {
                DFService.deregister(this);
                System.out.println(getLocalName()+" DEREGISTERED WITH THE DF");
            } catch (FIPAException e) {
                e.printStackTrace();
            }
    }//fin takedown
}
```

Apéndice C

Tablas en SQL del proyecto

A continuación se muestra el script en SQL para las tablas utilizadas en el proyecto:

```
Create database mercado;
use mercado;

create table articulo (
 id_articulo bigint(14) NOT NULL,
 nombre_art varchar(20) NOT NULL,
 depto_art varchar(15) NOT NULL,
 precio_art real(6,2) NOT NULL,
 PRIMARY KEY (id_articulo),
 unique index ind_articulo (id_articulo)
);

create table ticket (
 id_ticket int(8) NOT NULL,
 id_tienda int(4) NOT NULL,
 cliente varchar(30) NOT NULL,
 fecha date,
 hora time,
 cajera varchar(20),
 caja tinyint(2),
 PRIMARY KEY (id_ticket),
 unique index ind_ticket (id_ticket)
);

create table detalleticket(
 id_ticket int(8) NOT NULL,
 id_articulo bigint(14) NOT NULL,
 cant_art int(6) NOT NULL,
 precio_art real(6,2) NOT NULL,
 PRIMARY KEY (id_ticket,id_articulo),
 FOREIGN KEY (id_ticket) REFERENCES ticket(id_ticket),
 FOREIGN KEY (id_articulo) REFERENCES articulo(id_articulo)
);

create table mineria(
 idpm int AUTO_INCREMENT NOT NULL,
 id_tienda tinyint(2) NOT NULL,
```

```
 fecha date,
 hora time,
 tipocond tinyint(1) NOT NULL,
 condicion text,
 condicion2 text,
 soporte int(2) NOT NULL,
 confianza int(2) NOT NULL,
 resmineria text,
 num_sitios_disp int(3),
 num_sitios_clon int(3),
 PRIMARY KEY (idpm)
);

create table sitios_disp(
 idpm int NOT NULL,
 nom_sitio varchar(20) NOT NULL,
 band_clono tinyint(1),
 band_result tinyint(1),
 band_activo tinyint(1),
 resmineria text,
 total_tickets int,
 PRIMARY KEY (idpm,nom_sitio),
 FOREIGN KEY (idpm) REFERENCES mineria(idpm)
);

create table parametros(
 idpm int NOT NULL,
 cond tinyint(1),
 cond_art text,
 cond_depto text,
 fecha_inicio varchar(8),
 fecha_fin varchar(8),
 PRIMARY KEY (idpm),
 FOREIGN KEY (idpm) REFERENCES mineria(idpm)
);

create table restiempo(
 idpm int NOT NULL,
 hora time,
 PRIMARY KEY (idpm),
 FOREIGN KEY (idpm) REFERENCES mineria(idpm)
);
```

Índice de Tablas

Índice de Figuras

Abreviaturas

ACL	Agent Communication Language, Lenguaje de Comunicación del Agente
BT Labs	British Telecommunications Laboratories, Laboratorio de Telecomunicaciones Británico
CSELT	Centro Studio e Laboratori Telecomunicazione, Centro de Estudio y Laboratorio de Telecomunicaciones
DM	Data Mining, Minería de datos
DMQL	Data Mining Query Language, Lenguaje de Consultas de Minería de Datos
FIPA	Foundation for Intelligent Physical Agents, Fundación para Agentes Físicos Inteligentes
HTML	HyperText Markup Language, Lenguaje de Marcas de Hipertexto
JADE	Java Agent DEvelopment framework, Ambiente de Desarrollo de Agentes en Java
Jafmas	Java Agent-based Framework for Multi-Agent Systems, Ambiente de Desarrollo basado en Java para Sistemas Multiagente
JATLite	Java Agent Template, Lite; Plantilla de Agentes Java, Lite
KSE	Knowledge Sharing Effort
KQML	Knowledge Query and Manipulation Language, Lenguaje de Consulta y Manipulación de Conocimiento
MASIF	Mobile Agent System Interoperabilities Facility
OLAP	On-Line Analytical Processing, Procesamiento Analítico en Línea
OMG	Object Manager Group

RMA	Remote Management Agent, Agente de Gerencia Remoto
SMA	Sistemas Multiagentes
SQL	Structured Query Language, Lenguaje Estructurado de Consultas
WM	Web Mining, Minería Web
WML	Wireless Markup Language, Lenguaje Inalámbrico de Marcas
Zeus	Zeus Agent Building Toolkit, Herramientas para la Construcción de Agentes Zeus