Capítulo 6

Cuestión de infraestructura

Este problema de cerrar la brecha para llegar a cierta parte que funcione es ingeniería. Requiere un estudio serio de problemas de diseño... hay un largo camino por recorrer entre los principios básicos y un diseño práctico y económico.

RICHARD FEYNMAN
Física. Volumen II: Electromagnetismo y Materia

En los capítulos anteriores de esta memoria se han descrito algunas de las técnicas de clasificación que se utilizan para modelar grandes conjuntos de datos (capítulo 2), se ha propuesto un modelo de clasificación a medio camino entre los árboles y las listas de decisión (el modelo ART, capítulo 3), se ha analizado cómo pueden formularse hipótesis al construir el clasificador utilizando técnicas de extracción de reglas de asociación (el algoritmo TBAR, capítulo 4) y se ha estudiado el procesamiento de información cuantitativa para construir modelos simbólicos discretos (capítulo 5). Todas las técnicas examinadas son, sin duda, de gran utilidad para resolver problemas reales en situaciones concretas. Sin embargo, no se puede aprovechar todo su potencial utilizándolas de forma aislada: es imprescindible interpretarlas como herramientas particulares utilizables en un marco más general que las englobe. El estudio de un sistema que proporcione la infraestructura necesaria para tal fin es el objeto del presente capítulo.

Aún hoy, los sistemas informáticos son mucho mejores recopilando datos que utilizándolos de una forma razonable [156]. Por este motivo, durante los últimos años se han desarrollado multitud de técnicas de *Data Mining*, entre las que se encuentra el modelo de clasificación ART propuesto esta memoria. Las grandes posibilidades que ofrecen estas técnicas en aplicaciones de todo tipo han atraído la atención de las grandes multinacionales de la Informática y han hecho posible la creación de un floreciente mercado en el que numerosas empresas desarrollan su actividad [78] [124].

El objetivo de este capítulo es plantear una arquitectura general que facilite la implementación y utilización de técnicas de *Data Mining*, como el modelo de clasificación ART propuesto en esta memoria.

Los sistemas de información, entendidos como sistemas informáticos orientados a la resolución de problemas de un determinado tipo, se dividen básicamente en sistemas de procesamiento de transacciones (OLTP: *On-Line Transaction Processing*) y sistemas de ayuda a la decisión (DSS: *Decision-Support Systems*), entre los que se hallan las aplicaciones OLAP (*On-Line Analytical Processing*), las cuales suelen emplear técnicas de *Data Mining*.

Los sistemas de ayuda a la decisión son sistemas cuyo objetivo es facilitar al usuario la toma de decisiones en situaciones no estructuradas [139]. Dichos sistemas tienen necesidades específicas que no pueden resolverse utilizando sistemas de información convencionales [29] [37] [161]. Los sistemas OLTP tradicionales suelen trabajar con fragmentos relativamente pequeños de información (transacciones) mientras que las aplicaciones de ayuda a la decisión requieren el análisis eficiente de cantidades enormes de datos para satisfacer las expectativas de los denominados 'trabajadores del conocimiento' (ejecutivos y analistas).

Para hacer factible el procesamiento de las ingentes cantidades de datos que organizaciones de todo tipo recopilan durante su funcionamiento cotidiano, es necesario el desarrollo de técnicas de *Data Mining* que, además, sean escalables [128]; es decir, técnicas cuyo rendimiento no se degrade en exceso cuando crece el volumen de datos sobre el que han de trabajar. Las investigaciones en este tema se centran principalmente en la búsqueda de algoritmos más eficientes que reduzcan el espacio de búsqueda de posibles soluciones,

mejoren la calidad de las heurísticas empleadas u optimicen el uso de los recursos computacionales disponibles (por ejemplo, integrando los sistemas de extracción de conocimiento con los gestores de bases de datos que almacenan los datos). También pueden resultar de utilidad el uso de técnicas de muestreo (que permiten reducir el tamaño de los conjuntos de datos), el diseño de técnicas incrementales (que permiten actualizar modelos existentes sin tener que volver a reconstruirlos por completo) y, sobre todo, la implementación de técnicas de procesamiento distribuido y paralelo.

En este capítulo se describe un modelo general de sistema de cómputo adecuado para la resolución de problemas de *Data Mining* y, en general, de cualquier tipo de problemas que requieran gran capacidad de cómputo (como pueden ser, por ejemplo, las aplicaciones científicas tradicionalmente asociadas al uso de supercomputadores). El modelo conceptual de un sistema de este tipo se presenta en la sección 6.1. La implementación real de tal sistema se podría llevar a cabo en el futuro construyendo un sistema distribuido basado en componentes. En cierto modo, los requerimientos de este sistema serían similares a los de un sistema multiagente [21] [144]. De hecho, un sistema de agentes inteligentes, tal como se suele definir en Inteligencia Artificial, no es más que un sistema distribuido basado en componentes desde el punto de vista de la Ingeniería del Software, con la peculiaridad de permitir la migración de los agentes entre distintos nodos del sistema (esto es, no sólo se transfieren datos, sino que también se transfiere el código que ha de ejecutarse).

Los aspectos más relevantes del sistema propuesto se discuten en las siguientes secciones. En el apartado 6.2 se comenta la evolución histórica de los sistemas distribuidos y se especifican las necesidades de infraestructura que un sistema distribuido de cómputo ha de satisfacer. La sección siguiente, 6.3, se centra en el estudio de la arquitectura de un sistema basado en componentes. Los sistemas de este tipo resultan especialmente interesantes para el desarrollo de aplicaciones de *Data Mining* porque proporcionan la infraestructura necesaria para que podamos centrar nuestros esfuerzos en el diseño, implementación y evaluación de algoritmos de extracción de conocimiento. El capítulo se cierra con una discusión sobre algunas decisiones particulares de diseño e implementación que se pueden tomar para crear un sistema como el propuesto a

nivel teórico (sección 6.4) y una declaración de intenciones sobre el papel que las técnicas descritas en este capítulo pueden desempeñar en el futuro (sección 6.5).

6.1. Modelo conceptual del sistema

En Ingeniería del Software es importante que la descripción de un sistema se realice de forma totalmente independiente a cómo se vaya a implementar. La construcción de un modelo independiente de cualquier tecnología particular dota de mayor libertad al diseñador y permite el desarrollo de tecnologías compatibles con tantas técnicas de implementación y estándares como se desee. Esta cualidad está ligada al enfoque MDA (*Model-Driven Architecture*), el cual se basa en la idea de que la estructura y el comportamiento esencial de un sistema ha de capturarse de forma abstracta de forma que pueda adaptarse a diferentes tecnologías en su implementación [11]. Este enfoque no sólo permite que el diseñador pueda escoger la tecnología más apropiada para implementar el sistema, sino que también dota al sistema de la capacidad de sobrevivir a la inevitable evolución tecnológica.

Siguiendo el enfoque MDA, en este capítulo se propone un sistema abierto que permita el desarrollo de algoritmos de extracción de conocimiento en bases de datos. Este sistema ha de incluir la posibilidad de que se puedan realizar tareas computacionalmente costosas para analizar enormes conjuntos de datos, agrupar datos, construir modelos de clasificación y extraer patrones y reglas de asociación. Dicho sistema, de hecho, ha de ser utilizable en la realización de cualquier aplicación de cálculo intensivo (p.ej. simulaciones científicas) y debe proveer los mecanismos necesarios para acceder a los datos de una forma eficiente. En general, el modelo conceptual de un sistema de este tipo se ajusta al de la figura 6.1.

El usuario de un sistema de este tipo ha de trabajar con grandes conjuntos de datos, para lo cual el sistema ha de emplear técnicas y herramientas de *Data Mining*. Los datos, que pueden provenir de fuentes heterogéneas, y los algoritmos de extracción de conocimiento disponibles se emplean para construir modelos que pueden tener funciones descriptivas (esto es, facilitan el análisis

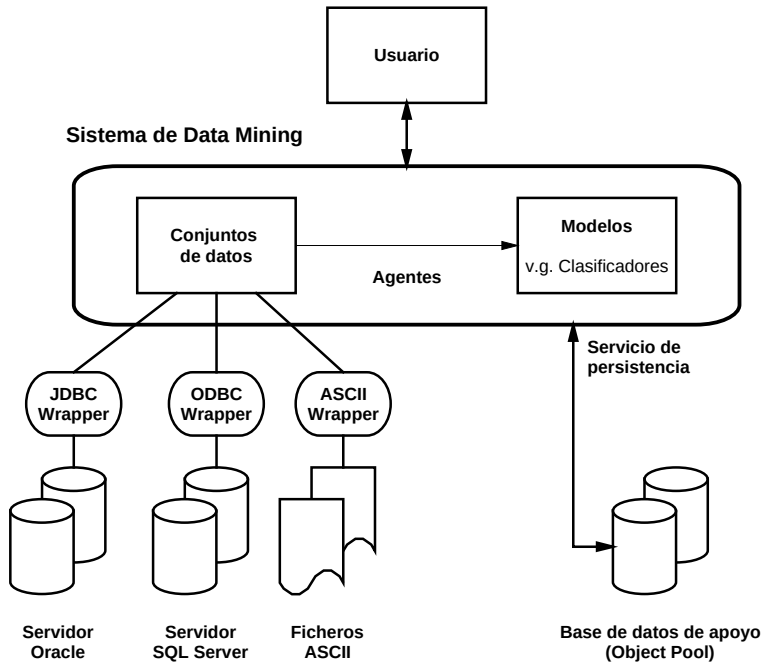

Figura 6.1: Modelo conceptual del sistema

de los datos existentes) o predictivas (cuando son de ayuda en situaciones novedosas). Los modelos construidos pueden también emplearse como entrada de algoritmos de extracción de conocimiento de segundo orden si el usuario desea analizar los modelos ya obtenidos y pueden, también, complementar a los datos de entrada cuando éstos han de ser analizados. Tanto los modelos obtenidos como los metadatos que caracterizan a los conjuntos de datos empleados para obtener dichos modelos han de almacenarse de forma permanente en una base de datos que permita su utilización posterior.

Cualquier sistema de información puede describirse mediante una estructura, una serie de mecanismos y una política según el modelo SMP (*Structure-Mechanism-Policy*) de Perry y Kaiser [123]. La estructura viene dada por los objetos y agregaciones de objetos sobre los cuales operan los mecanismos, que son las herramientas que nos permiten resolver problemas, mientras que la política consiste en una serie de reglas y estrategias impuestas por el entorno.

En nuestro sistema, se ha de proporcionar la infraestructura que permita almacenar los objetos sobre los que se actúa (servicio de persistencia) y mantenga a disposición del usuario los mecanismos que le permiten operar sobre dichos objetos (denominados agentes en el campo de los sistemas inteligentes, o procesos en el ámbito de la Ingeniería del Software). Tanto los objetos como los agentes forman parte del sistema y son ciudadanos de primera clase en el mismo.

- Los **objetos** sobre los que se actúa pueden ser conjuntos de datos almacenados en distintos formatos, ontologías [28] que nos permiten definir jerarquías de conceptos mediante las cuales caracterizar los datos, modelos de clasificación, etcétera.

  Los objetos son entidades permanentes y han de almacenarse de forma persistente para evitar su destrucción (salvo, obviamente, que el usuario expresamente desee eliminarlos). El almacenamiento de tales objetos ha de realizarse de forma transparente al usuario, sin que éste necesite saber ni cómo ni donde se guardan. No obstante, sí es necesario un sistema de recuperación de información que permita al usuario acceder a los objetos existentes en el sistema.

- Los **agentes** proporcionan los mecanismos que se emplean para trabajar con los objetos anteriores: algoritmos y procesos mediante los cuales se resuelven problemas de extracción de conocimiento en bases de datos, incluyendo tareas de preprocesamiento (muestreo, discretización, selección de características...) y de post-procesamiento (exploración de reglas, simplificación y poda, resúmenes...).

  Las instancias particulares de los agentes, a diferencia de los objetos, sólo existen durante la ejecución de la tarea que tengan encomendada, por lo que no es necesario almacenarlos de forma permanente una vez que hayan concluido su misión. Sin embargo, sí resulta necesario almacenar el estado de los agentes para conseguir un sistema fiable. Idealmente, se ha de conseguir una *persistencia ortogonal*, entendida ésta como la posibilidad de que un agente, tras ser interrumpida su ejecución por motivos internos o externos al sistema, tenga la posibilidad de continuar su ejecución por donde se encontrase. En la práctica, no obstante, tendremos que conformarnos con permitir que el agente repita parte del trabajo que hubiese realizado antes de detener su ejecución.

En cuanto al tercer componente del modelo SMP, la existencia de reglas y estrategias que permitan el funcionamiento del sistema, de cara al usuario el sistema es una caja negra respecto a la cual puede desempeñar distintos roles.

El sistema de información de la figura 6.1 puede considerarse un sistema de gestión de modelos (MMS: *Model Management System*). En este tipo de sistemas de ayuda a la decisión hay que diferenciar distintos tipos de usuarios: los usuarios de los modelos existentes se limitan a explorar e interactuar con modelos previamente desarrollados, mientras que los constructores de modelos son los usuarios que crean modelos a partir de los datos y modelos ya existentes. Ambos tipos de usuarios pueden ser científicos, ejecutivos o analistas sin conocimientos informáticos más allá del uso básico de un ordenador personal, por lo cual el sistema ha de ser fácil de usar y su complejidad interna debe quedar oculta de cara al usuario.

En un sistema clásico de ayuda a la decisión, un tercer conjunto de usuarios se encarga de implementar los modelos definidos para que puedan ser uti-

lizados por los usuarios anteriores (en nuestro caso, los agentes que permiten obtener nuevos modelos), mientras que un cuarto y último grupo de usuarios es el responsable de ajustar parámetros, depurar, evaluar, validar y mantener los modelos del sistema. Estas actividades requieren conocimientos que no suelen poseer los usuarios finales del sistema y, además, suelen ser automatizables en parte. Las tareas automatizables pueden diseñarse para que el usuario final se encargue de gestionarlas y goce así de mayor autonomía. El objetivo final es que un usuario no experto pueda sacar el máximo partido de un sistema avanzado de *Data Mining*.

En definitiva, el diseño del sistema debe ayudar a que los usuarios puedan descubrir información útil a partir de sus datos sin tener que ser expertos en técnicas de *Data Mining* [122] y ser fácil de usar [89], pero tampoco debe olvidar que se trata de un entorno dinámico que ha de permitir el uso de nuevas técnicas en cuanto estén disponibles. Para mantener un tiempo de respuesta aceptable de cara al usuario, el sistema se puede implementar en un entorno distribuido (sección 6.2). Por otro lado, dado que el sistema ha de estar preparado para adaptar su configuración a situaciones cambiantes y para evolucionar incorporando nuevas técnicas y algoritmos, es recomendable diseñarlo como un sistema basado en componentes en el cual se puedan añadir y eliminar agentes de forma dinámica (sección 6.3).

6.2. Sistemas distribuidos

Los sistemas distribuidos pueden ofrecer servicios valiosos compartiendo los recursos computacionales de un gran número de usuarios, tanto ciclos no utilizados de CPU como espacio libre de almacenamiento en disco. Tales sistemas permiten formar redes virtuales cuyos nodos de procesamiento se pueden encontrar distribuidos geográficamente, a diferencia de los grandes y costosos sistemas centralizados utilizados mayoritariamente en la actualidad, y pueden disponer de una potencia de cómputo similar a la de los supercomputadores más potentes.

6.2.1. Evolución y tendencias

El modelo tradicional cliente/servidor en el cual un "pequeño" cliente solicita y recibe información de un "gran" servidor está comenzando a dejar paso a otro tipo de arquitectura: un modelo en el cual todos los participantes comparten responsabilidades y son aproximadamente iguales. Este modelo es conocido por el acrónimo P2P, el cual proviene de la expresión inglesa *Peer-to-peer*, y muchos ven en él una de las tecnologías que marcarán el futuro de Internet [72] [160].

6.2.1.1. Sistemas P2P

Los sistemas P2P se caracterizan por ser sistemas distribuidos que no dependen inherentemente de puntos centralizados de control, aunque también es cierto que no excluyen la posibilidad de utilizarlos cuando sea conveniente. Los nodos de una red P2P se relacionan con sus vecinos (desde el punto de vista lógico) y se caracterizan por su capacidad para descubrir otros nodos y conectarse con ellos sin que haya una relación preestablecida entre ellos, tal como sucede en los sistemas cliente/servidor.

Los sistemas P2P tienen el potencial de aprovechar tres recursos de Internet que, hoy en día, el modelo cliente/servidor utiliza por debajo de sus posibilidades: la información disponible, el ancho de banda de las redes que componen Internet y, especialmente, la capacidad de cómputo y de almacenamiento de las máquinas conectadas a la Red. Además de la evidente infrautilización de recursos, el modelo actual ha propiciado la aparición de varios problemas:

- Independientemente del ancho de banda disponible, determinadas zonas de Internet se siguen saturando cuando todo el mundo accede a portales como Yahoo!, desea leer las últimas noticias de la CNN o intenta leer su correo electrónico de alguno de los grandes proveedores de acceso a Internet. Al mismo tiempo, el tráfico en otras zonas de la Red se mantiene muy por debajo de su capacidad.

- A pesar de las mejoras tecnológicas, las necesidades de cómputo y almacenamiento se han acumulado en torno a centros de procesamiento de

datos cuyo crecimiento y necesidades energéticas ha dado lugar incluso a problemas de suministro eléctrico.

- La existencia de sistemas centralizados facilita la censura y el espionaje, comprometiendo la privacidad de la información que se transmite a través de la red.

Frente a la infrautilización de los recursos disponibles y los problemas ocasionados por el modelo cliente/servidor, las tecnologías P2P pueden emplearse en distintos tipos de aplicaciones:

- **Sistemas de publicación de información**

  El intercambio de ficheros de sonido en formato MP3 y de vídeo en formato DivX ha sido, sin duda, la aplicación que ha conseguido alcanzar la masa crítica necesaria para el desarrollo de las tecnologías P2P. Además de compartir ficheros, los usuarios de estos sistemas comparten responsabilidades legales. No obstante, al ser relativamente fácil mantener el anonimato en estos sistemas, resulta difícil (si no imposible) anular su expansión. Algunos sistemas pertenecientes a esta categoría son muy conocidos:

  – Napster, un sistema híbrido C/S-P2P que puso en pie de guerra a la industria discográfica,

  – Gnutella, que sí es un auténtico sistema P2P, aunque mal diseñado porque su algoritmo de enrutamiento por inundación genera muchísimo tráfico innecesario,

  – Kazaa, tristemente conocido por ser un medio eficaz para difusión de virus informáticos,

  – MojoNation, que incluye mecanismos de recompensa a los usuarios en función de los servicios que prestan; y,

  – eDonkey, que transfiere fragmentos de archivos en paralelo y de forma independiente para mejorar el rendimiento del sistema.

También existen otros sistemas de este tipo con aspiraciones más nobles, como Publius (un sistema resistente a cualquier tipo de censura) o Freenet (un almacén de información virtual que permite a cualquier persona publicar o acceder a cualquier tipo de información, asegurando la privacidad del lector/editor así como la seguridad, la integridad y la disponibilidad de la información publicada [36]).

- **Sistemas de comunicación interpersonal**

  Desde los grupos de noticias de UseNet o el intercambio de mensajes entre BBSs con FidoNet hasta llegar a los programas de chat (ICQ o mIRC) y los sistemas de mensajería instantánea (como MSN Messenger), los sistemas distribuidos de comunicación siempre han gozado de gran popularidad entre los internautas. Dichos sistemas, distribuidos y descentralizados, se pueden considerar ejemplos clásicos de sistemas P2P. Estos sistemas existían antes de que se acuñase el acrónimo P2P, si bien utilizan básicamente las mismas técnicas que los sistemas P2P actuales. Algunos de los sistemas más recientes, como Jabber, aprovechan el desarrollo actual de las tecnologías P2P para mejorar el rendimiento de los sistemas clásicos.

  Por otro lado, la evolución de los productos software orientados a facilitar el trabajo en grupo (*groupware*) también ha dado lugar a sistemas P2P como los comercializados por Groove Networks para la gestión de proyectos distribuidos geográficamente en los que pueden colaborar miembros de distintas organizaciones.

- **Sistemas distribuidos de cómputo**

  Un tercer grupo de sistemas que ya han sido implementados con éxito lo constituyen las aplicaciones de cálculo intensivo. La arquitectura de estos sistemas es de especial interés para cualquier investigador interesado en el desarrollo de sistemas de *Data Mining*, los cuales no sólo necesitan una gran potencia de cálculo, sino que requieren una distribución eficiente de los datos sobre los que se trabaja.

  El proyecto SETI@Home fue pionero en este ámbito. Mediante una apli-

cación específica que se ejecuta como protector de pantalla, se aprovecha el tiempo de CPU no utilizado por miles de usuarios que tienen su ordenador conectado a Internet para buscar posibles señales de inteligencia extraterrestre analizando la información recibida del espacio por radiotelescopios. Otros proyectos actuales de este tipo son *eOn* (que simula procesos químicos a escala atómica), *Genome@Home* o *Folding@Home* (ambos dedicados a profundizar en los secretos de la vida). Además de estos proyectos sin ánimo de lucro, también existen empresas que suministran servicios de cómputo distribuido como Parabon Computation, Entropia, United Devices o Avaki.

En la actualidad, existen numerosos proyectos de investigación y desarrollo relativos a la construcción de supercomputadores distribuidos autoconfigurables, proyectos a los que se hace referencia con el anglicismo *grid computing*. El proyecto *Globus* (`http://www.globus.org`) es actualmente el estándar de facto en este campo, si bien la Unión Europea subvenciona un proyecto paralelo denominado *DataGrid*.

Aparte de las aplicaciones descritas en los párrafos anteriores, los sistemas P2P pueden llegar a utilizarse para construir sistemas distribuidos de recuperación de información que eliminen las limitaciones de los sistemas actuales (ya que ningún motor de búsqueda puede mantener actualizado un catálogo completo del contenido de Internet) o directorios distribuidos (que, en su día, podrían sustituir el sistema de nombres actual, DNS, cuya disponibilidad depende de un conjunto de servidores centrales repartidos por medio mundo). También existen soluciones comerciales que hacen uso de técnicas P2P para distribuir contenidos por Internet (*streaming* de audio y vídeo) o realizar copias de seguridad de forma automática.

Dado el auge de estos sistemas distribuidos semi-autónomos, las grandes multinacionales informáticas han mostrado su interés en ellos. Aunque la viabilidad comercial de los sistemas P2P está aún por demostrar, nadie quiere perder la posibilidad de obtener su cuota de mercado en este sector. Puede que el modelo P2P no resulte adecuado para la mayor parte de las aplicaciones informáticas, pero sin duda puede resultar útil para aplicaciones específicas de

carácter científico y técnico.

La popularidad del acrónimo P2P ha propiciado que muchos sistemas sean anunciados como sistemas P2P de forma más que discutible (igual que sucedió en su momento con las bases de datos relacionales o los sistemas orientados a objetos). Por ejemplo, el controvertido proyecto *Hailstorm* de Microsoft (rebautizado como plataforma *.NET My Services*) incorpora un sistema de autentificación global (*.NET Passport*) para facilitar el comercio electrónico y un sistema de notificación (*.NET Alerts*) que pretende ayudar a las empresas a mejorar su eficacia y las relaciones con sus clientes. La plataforma, destinada a "construir aplicaciones centradas en el usuario", se ofrece como un conjunto de servicios web y, si bien difícilmente puede considerarse un sistema P2P, descentraliza algunas funciones que estarían centralizadas en un sistema cliente/servidor clásico.

A pesar de que la plataforma promovida por Microsoft no sea completamente distribuida y abierta, sí marca un giro en las tendencias actuales: Los sistemas distribuidos tienden a utilizar protocolos abiertos de comunicación y XML para intercambiar datos [40]. Proyectos alternativos, como el proyecto JXTA (de 'yuxtaposición') iniciado por Sun Microsystems [162], pretenden llegar más allá y generalizar el uso de protocolos P2P abiertos que permitan comunicarse a cualquier dispositivo (desde teléfonos móviles a ordenadores personales y PDAs) y colaborar con otros dispositivos de la red sin estar sujetos a ningún estándar propietario. Los servicios web [41] y el lenguaje XML [16] se perfilan como la base sobre la que se construirán los sistemas distribuidos del futuro.

Anderson y Kubiatowicz van más allá de los sistemas P2P proponiendo la creación de un sistema operativo global que permitiese conectar la capacidad de procesamiento y de almacenamiento de millones de ordenadores independientes [10]. Su supercomputador virtual, gestionado por ISOS (*Internet-Scale Operating System*), permitiría alquilar tiempo y espacio en ordenadores personales, haciendo en gran medida innecesarios los costosos centros de procesamiento de datos conocidos popularmente como "granjas de servidores". No obstante, no todo el mundo está de acuerdo en que sistemas P2P de tal magnitud lleguen a existir algún día [61].

6.2.1.2. La taxonomía IFCS

Si empleamos la taxonomía IFCS (*Individual-Family-City-State*) que Perry y Kaiser utilizaron para caracterizar la evolución de los entornos de desarrollo de software [123], podemos observar cómo los sistemas de cómputo intensivo necesarios para determinado tipo de aplicaciones (como es el caso de los sistemas utilizados para resolver problemas de *Data Mining*) parecen desarrollarse de un modo similar al delineado por dicha taxonomía:

- **Individuo (I)**: Multiprocesadores y supercomputadores de propósito específico se emplean para resolver problemas de cálculo intensivo.

- **Familia (F)**: Conjuntos de ordenadores conectados a una red local de alta velocidad coordinan su funcionamiento para formar multicomputadores.

- **Ciudad (C)**: Múltiples sistemas, puede que geográficamente distribuidos, cooperan en la resolución de problemas específicos, como es el caso de los sistemas P2P actuales.

- **Estado (S)**: Un sistema de coordinación, como podría ser un futuro ISOS [10] a nivel global, proporciona la infraestructura necesaria para que la cooperación entre ordenadores independientes no se limite a casos puntuales.

6.2.2. Requisitos del sistema

El desarrollo de un sistema distribuido flexible que permita resolver problemas de cálculo reservados a los supercomputadores más potentes y pueda ser empleado eficazmente en aplicaciones OLAP requiere tener en cuenta distintos aspectos:

6.2.2.1. Comunicación entre nodos de procesamiento

La existencia de un mecanismo de comunicación común es imprescindible para que conjuntos de agentes (desarrollados potencialmente de forma independiente) puedan coordinar su trabajo de una forma útil. Un lenguaje estándar de comunicación entre agentes garantiza que los mensajes enviados sean correctos sintácticamente y puedan ser interpretados de un modo consistente desde un punto de vista semántico.

De hecho, cualquier sistema distribuido necesita algún mecanismo de comunicación que sirva de enlace entre los distintos nodos del sistema. Un sencillo protocolo como SOAP [41] es suficiente para que sistemas autónomos puedan comunicarse entre sí aprovechando los protocolos estándar de Internet, pues SOAP puede funcionar sobre HTTP (el protocolo utilizado para transmitir páginas web en Internet) o SMTP (utilizado para enviar mensajes de correo electrónico).

Más importante si cabe que el mecanismo concreto de comunicación es la topología de la red de comunicación. Básicamente, existen cuatro arquitecturas generales que se utilizan para construir topologías más complejas:

- En un sistema centralizado múltiples clientes acceden a un servidor central. La figura 6.2 muestra la topología de los sistemas cliente/servidor típica de las redes en estrella. En estos sistemas, si el servidor central falla, todo el sistema se viene abajo. Además, las escalabilidad del sistema viene limitada por la capacidad del servidor central (su capacidad de cómputo y el ancho de banda al que pueda dar servicio).

- Un sistema conectado en anillo como el de la figura 6.3 permite un mayor grado de tolerancia ante la presencia de fallos en máquinas concretas del anillo y es más escalable.

- Un sistema jerárquico, figura 6.4, tiene topología en forma de árbol. Los servidores de nombres en Internet que proporcionan el servicio DNS se organizan de esta forma. Los sistemas jerárquicos sobrellevan mejor las limitaciones de los sistemas centralizados y han demostrado sobradamente su escalabilidad (de hecho, el servicio DNS original sobrevive aún a la explosión de Internet).

- Una arquitectura completamente descentralizada, como la que aparece en la figura 6.5, es la empleada por algunos de los sistemas P2P actuales (p.ej. Gnutella). Estos sistemas son difíciles de gestionar y pueden presentar graves problemas de seguridad. Además, su escalabilidad es difícil de evaluar por la carga adicional que supone mantener la coherencia y la consistencia de un sistema no estructurado.

Los resultados más interesantes se consiguen, no obstante, cuando se combinan distintas topologías para conseguir arquitecturas híbridas que aprovechan las ventajas y disminuyen los inconvenientes de las topologías que las forman:

- Un sistema centralizado en el cual se sustituye el servidor por un anillo, figura 6.6, resulta especialmente útil para mantener la sencillez de un sistema centralizado y la redundancia que ofrece un anillo. Esta arquitectura es útil para mejorar la escalabilidad de un sistema centralizado.

- De forma alternativa, el anillo puede combinarse con un modelo centralizado para facilitar la implementación de aplicaciones de propósito específico [55], tal como muestra la figura 6.7.

- También se puede construir un sistema jerárquico introduciendo conexiones redundantes entre los nodos del árbol para mejorar la tolerancia del sistema ante posibles fallos. El sistema FreeNet se basa en una topología de este tipo [36].

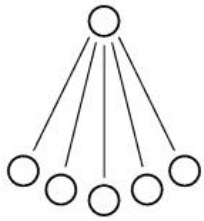

Figura 6.2: Sistema centralizado cliente/servidor

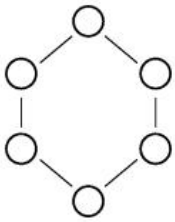

Figura 6.3: Red en anillo

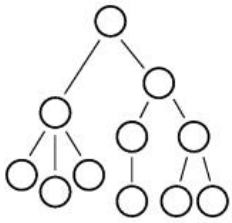

Figura 6.4: Sistema jerárquico

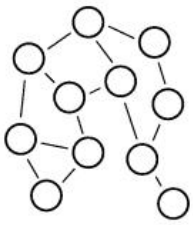

Figura 6.5: Sistema descentralizado

- Un sistema descentralizado híbrido en el que partes del sistema se centralizan, como el de la figura 6.8, goza de mayor flexibilidad y es más fácil de gestionar que un sistema descentralizado puro. Este modelo es especialmente útil cuando el sistema rebasa los límites de una organización concreta y se convierte en un servicio compartido por distintas entidades autónomas. El correo electrónico en Internet puede tomarse como modelo clásico de este tipo de sistemas: un conjunto de servidores de correo se encargan de intercambiar mensajes mientras que los usuarios finales del sistema siempre acceden a servidores concretos.

Como parece lógico, esta última arquitectura es la que consigue un compromiso mejor entre la sencillez de una topología centralizada y la flexibilidad extrema de un sistema descentralizado. Cada nodo central de la red estará conectado a otros nodos centrales y tendrá, además, conexiones con nodos satélite que pueden aportar recursos computacionales sin tener acceso completo al resto de la red.

6.2.2.2. Acceso a los datos

Otro aspecto importante relativo a la comunicación entre los distintos componentes de un sistema distribuido cuando pretendemos construir un sistema OLAP es decidir qué estrategia seguir para acceder a los datos que se utilizan como fuente de información a la hora de construir modelos.

Tradicionalmente, los algoritmos de aprendizaje se han implementado de forma que el usuario accede a ellos a través, posiblemente, de una interfaz gráfica. Si bien dichos algoritmos suelen acceder a datos que se hallan almacenados en ficheros locales, existe la posibilidad de que los datos estén almacenados en una base de datos a la cual se accede utilizando un lenguaje de consulta estándar (como es el caso de SQL para las bases de datos relacionales). En esta situación, además, se han de tener en cuenta las distintas posibilidades de acoplamiento que pueden existir entre la implementación del algoritmo de *Data Mining* y el sistema gestor de bases de datos, tal como se estudian en [6]. En el artículo de Maniatty y Zaky en SIGKDD Explorations [106] se comentan distintas técnicas que se pueden utilizar a bajo nivel para

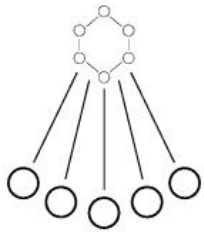

Figura 6.6: Sistema centralizado con anillo.

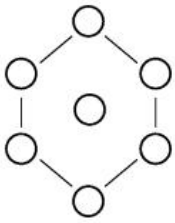

Figura 6.7: Sistema en anillo con coordinación centralizada.

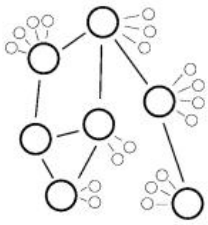

Figura 6.8: Sistema híbrido centralizado-descentralizado

permitir un acceso eficiente a los datos, los cuales pueden provenir de fuentes de datos heterogéneas.

6.2.2.3. Dinámica del sistema

Además de utilizar un mecanismo básico de comunicación entre nodos de procesamiento y de poseer una topología específica, un sistema distribuido dinámico necesita disponer de servicios adicionales que le permitan funcionar correctamente. Estos servicios serán los responsables de que el sistema se adapte a situaciones cambiantes de forma autónoma.

Los estándares en que se basan los servicios web [41] pueden sernos de gran utilidad en este sentido: el lenguaje WSDL (*Web Services Description Language*) permite dar a conocer los servicios que cada nodo del sistema ofrece, así como la forma de acceder a ellos, mientras que un directorio UDDI (*Universal Description, Discovery, and Integration*) facilita a los usuarios del sistema el acceso a los distintos proveedores de servicios a modo de 'guía telefónica'. Es decir, WSDL puede emplearse para divulgar los servicios que ofrece cada nodo del sistema distribuido y UDDI puede utilizarse para que nuevos nodos sean capaces de descubrir sistemas y servicios, así como averiguar cómo sumarse al sistema distribuido.

Aparte de los servicios descritos en el párrafo anterior, que son comunes a todas las aplicaciones distribuidas que han de funcionar en un entorno dinámico, un sistema distribuido de cómputo debe proveer los mecanismos necesarios para poder delegar tareas en otros nodos, transfiriendo el código que ha de ejecutarse y suministrando los datos que sean necesarios. Si consideramos que las tareas de cómputo son realizadas en el sistema por agentes semi-autónomos, podemos afirmar que es imprescindible que tales agentes sean capaces de migrar de un nodo a otro. La movilidad de los componentes del sistema permite distribuir de una forma razonable la carga del sistema en su conjunto [117] pero también puede verse forzada por razones externas al sistema en sí (por ejemplo, cuando se pierden conexiones con una región determinada de la red sobre la que se opera el sistema distribuido). Un mecanismo transparente de migración debería, en principio, ser capaz de continuar la ejecución del agente desde el punto en el que se encontrase antes de migrar, sin perder los cálculos

que ya hubiese efectuado e, incluso, sin que el agente sea consciente de que ha sido trasladado.

Además de los mecanismos básicos que permiten que el sistema vaya evolucionando de una forma dinámica, sería en principio deseable que el sistema de comunicación no fuese tan rígido como los estrictos protocolos utilizados habitualmente para implementar software en sistemas distribuidos. Mediante la especificación explícita de políticas de funcionamiento no implícitas en la implementación del sistema, se pueden conseguir sistemas reconfigurables en los cuales los nodos gozan de cierta capacidad de decisión. Algunos trabajos realizados en el campo de los agentes inteligentes [21] pueden ser muy útiles en este sentido.

Por ejemplo, se pueden utilizar autómatas finitos para hacer que la comunicación entre nodos se realice mediante una conversación más flexible que una secuencia estructurada de mensajes [21]. Si A realiza una oferta para hacer uso de un servicio ofrecido por B, B puede aceptar la oferta de A, puede declinarla, puede no responder siquiera o podría, incluso, pedir una clarificación sobre la oferta realizada u ofrecer una estimación del tiempo que tardará en realizar la tarea encomendada (para saber si la oferta seguirá en pie o expirará si B decide aceptarla más adelante).

6.2.2.4. Seguridad y fiabilidad

Un sistema distribuido dinámico como el descrito en párrafos anteriores ha de incorporar los mecanismos de seguridad necesarios para garantizar la integridad de los datos que se transmiten entre los distintos nodos de procesamiento, así como evitar la pérdida de datos cuando parte del sistema sufre una avería.

La seguridad es esencial para evitar la inspección y/o manipulación tanto de los datos que se manejan como del código que se ha de ejecutar en su tránsito de un nodo a otro. Si utilizamos una arquitectura híbrida como la de la figura 6.8, se ha de establecer un perímetro de seguridad en torno a los nodos centrales de la red. Estos nodos son los que poseen información sobre la topología de la red y sirven de intermediarios cuando se transmiten datos de un punto a otro. La seguridad en los nodos periféricos no ha de ser tan

estricta, aunque siempre es conveniente que se utilicen técnicas criptográficas de protección de datos a la hora de transmitir y almacenar información. Dichas técnicas permiten un intercambio seguro de información confidencial a través de un medio compartido con usuarios y sistemas [26].

Por otro lado, para asegurar la fiabilidad del sistema se ha de introducir cierta redundancia que permita mejorar su disponibilidad y su tolerancia frente a posibles fallos. La transparencia que otorga la movilidad de código y datos descrita en el apartado anterior permite, además, que el sistema pueda replicar agentes y ejecutarlos en máquinas independientes por motivos de seguridad o fiabilidad, así como hacer copias de seguridad de la información que sea vital para el funcionamiento del sistema en su conjunto.

El uso de recursos en el sistema debe estar monitorizado y controlado en cada nodo, para mantener una contabilidad del consumo de cada agente y detectar comportamientos anómalos. La existencia de mecanismos de monitorización es esencial para detectar intentos de acceso no autorizados y mantener el uso autorizado de recursos dentro de unos límites razonables (de forma que no interfiera con otras tareas paralelas).

6.2.2.5. Planificación y asignación de recursos

Además de los mecanismos descritos en las secciones anteriores que permiten que los distintos nodos del sistema se comuniquen entre sí de una forma segura, un sistema distribuido necesita disponer de una estrategia de planificación y asignación de recursos.

En un sistema descentralizado como el de la figura 6.8, en el que un nodo particular sólo posee información parcial acerca del estado de la red en su conjunto, se han de diseñar heurísticas que permitan a los nodos del sistema negociar de forma autónoma con sus vecinos el reparto de tareas entre ellos. Este reparto ha de realizarse en función de la capacidad de cómputo, espacio de almacenamiento y ancho de banda disponibles en los distintos nodos de procesamiento, si bien también pueden influir otros factores.

Cuando los distintos nodos del sistema distribuido no están bajo el control de una misma organización, además, es necesario establecer un mecanismo de compensación que remunere a los distintos nodos del sistema en función de

los servicios prestados (tiempo de CPU o espacio de almacenamiento). Esta recompensa puede corresponderse con un pago real, aunque también puede emplearse para formar un sistema económico virtual en el cual se ofrezcan incentivos que motiven la cooperación de los integrantes del sistema. Un sistema de este tipo es análogo a un banco en el que cada cliente posee una cuenta en la que cobra por los servicios prestados a terceros y mediante la que paga por lo que consume. Por tanto, aun siendo un sistema básicamente descentralizado, es necesaria una entidad central que se encargue de contabilizar el uso de recursos que realiza cada agente del sistema. Obviamente, este sistema económico es innecesario si toda la infraestructura que da soporte al sistema distribuido de cómputo se encuentra bajo el control de una única organización (como puede ser el caso de la Intranet de una empresa).

6.3. Sistemas basados en componentes

Durante los últimos años [57] [95], hemos presenciado el vertiginoso desarrollo de los sistemas basados en componentes, hoy en día utilizados en los sistemas informáticos de casi todas las empresas. Dichos sistemas pretenden ayudar a los desarrolladores en la construcción de sistemas cada vez más complejos. El objetivo final de tales sistemas es mejorar la productividad del proceso de desarrollo de software promoviendo la reutilización de componentes y de patrones de diseño, un sueño perseguido por la Ingeniería del Software desde sus orígenes.

El origen de los sistemas actuales basados en componentes se puede encontrar en métodos de análisis y diseño como ROOM (*Real-Time Object-Oriented Modeling*), tal como se comenta en [11]. A diferencia de otros métodos de análisis que se centran en un modelado estructural utilizando básicamente diagramas de clases y diagramas de modelización de datos (E/R o CASE*Method), ROOM interpreta el software como un conjunto de procesos que interactúan: se sustituyen los tipos de datos abstractos (clases de objetos definidos por una especificación independiente de su representación) por actores que encapsulan procesos además de datos. Estos actores (agentes, si utilizamos la terminología actual) disponen de una serie de puertos que definen los men-

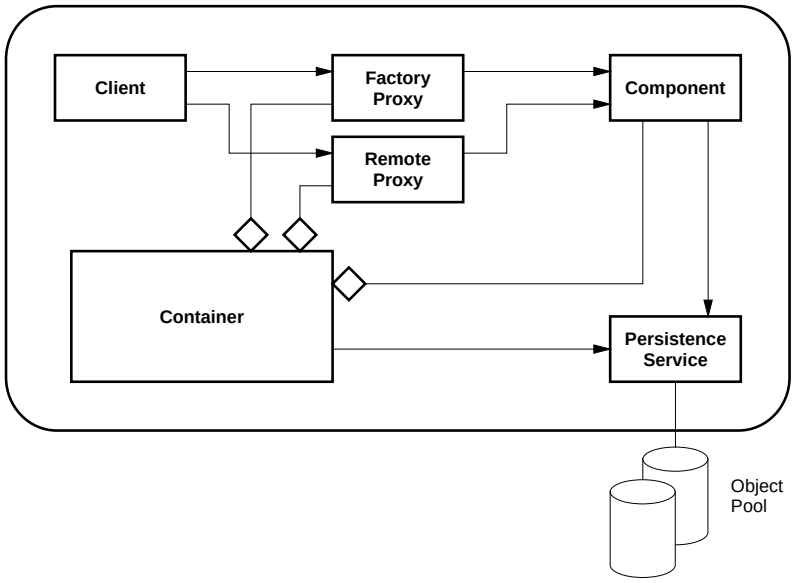

Figura 6.9: Patrón de diseño de los sistemas basados en componentes

sajes que pueden recibir o enviar; es decir, definen la interfaz del componente. Aunque se ideó antes de la aparición de la moda que han marcado tecnologías basadas en componentes como CORBA, COM/COM+ o Java Beans, ROOM incluye las características básicas de un sistema basado en componentes.

6.3.1. Patrón de diseño

Los sistemas basados en componentes utilizados en la actualidad, como los Enterprise JavaBeans de Java 2 Enterprise Edition [99] o la plataforma .NET de Microsoft [113], se basan todos en un patrón arquitectónico común [91]. La figura 6.9 muestra una representación simplificada de este patrón de diseño [65]. El patrón, que se puede modelar como una colaboración parametrizada en UML [116], incluye seis roles que aparecen como rectángulos en la figura 6.9:

- **Cliente (*client*):** Cualquier entidad que solicita un servicio de un componente del sistema. En los sistemas actuales (OLTP), tal solicitud se realiza invocando un método de alguna de las interfaces de los componentes. En un sistema OLAP, la solicitud se podría realizar utilizando un lenguaje de consulta específico para tareas de *Data Mining*, como DMQL u *OLE DB for Data Mining* (de Microsoft).

  En vez de acceder al componente directamente, el cliente utiliza internamente un par de intermediarios que transmiten las llamadas del cliente al componente. Este nivel de indirección, oculto desde la perspectiva del cliente, hace posible que el cliente no tenga que ser consciente de la localización real del componente al que accede y permite además que se puedan interceptar los mensajes que se transmiten con el fin de depurar, controlar y monitorizar el funcionamiento del sistema.

- **Intermediario constructor (*factory proxy*):** A través de él se facilita el acceso a los métodos estáticos de las clases que definen el interfaz de los componentes. Es el encargado de permitir operaciones genéricas como la creación de componentes o la búsqueda de un componente concreto.

- **Intermediario remoto (*remote proxy*):** Se encarga de facilitar el acceso a los métodos que gobiernan el estado y funcionamiento de las instancias particulares de los distintos componentes del sistema. Maneja operaciones específicas para cada tipo de componente, como inspeccionar su estado, fijar sus parámetros, etcétera.

- **Componente (*component*):** Son los bloques lógicos que forman el sistema. En el caso particular que nos ocupa, tanto los conjuntos de datos como los modelos construidos a partir de ellos son componentes del sistema. Los agentes que implementan los algoritmos de extracción de conocimiento también pueden considerarse componentes del sistema, si bien usualmente sólo se emplean a través del intermediario constructor para invocar la construcción de modelos de conocimiento (aunque tampoco se descarta la posibilidad de que se pueda acceder a ellos mientras dure su ejecución).

- **Contenedor (*container*):** Representa el entorno del sistema en tiempo de ejecución y es la parte del sistema encargada de proporcionar la infraestructura necesaria para su funcionamiento. El contenedor contiene tanto a los componentes como a los intermediarios que permiten a un cliente acceder a los componentes, tal como muestran las agregaciones en el diagrama de la figura 6.9.

  El contenedor ha de ofrecer los servicios necesarios para un sistema de cómputo distribuido, incluyendo dispositivos de seguridad que protejan los datos de accesos no autorizados, mecanismos de comunicación entre procesos/agentes, un servicio de persistencia que permita almacenar de forma permanente las instancias concretas de los distintos tipos de componentes del sistema y la capacidad de agregar dinámicamente al sistema nuevos agentes y componentes sin tener que detener su ejecución, algo conocido en los sistemas actuales por la expresión 'despliegue en caliente' (*hot deployment*).

- **Servicio de persistencia (*persistence service*):** Este servicio permite el almacenamiento permanente de los componentes del sistema y su funcionamiento ha de ser gestionado y coordinado por el contenedor de forma autónoma. En la base de datos que da soporte a este servicio (*object pool*) han de almacenarse tanto los metadatos que se empleen para caracterizar los conjuntos de datos utilizados como los modelos obtenidos a partir de ellos y toda la información relativa a las sesiones que los usuarios tengan abiertas en el sistema (esto es, los agentes que estén en ejecución). La información relativa a los objetos (conjuntos de datos y modelos) se almacena para permitir su uso futuro en otras sesiones, mientras que la relativa a los agentes se guarda por cuestiones de fiabilidad. Con el objetivo de permitir que el sistema sea tolerante a fallos, se preserva el estado del sistema en ejecución para que éste se pueda recuperar tras sufrir algún tipo de avería o corte de suministro eléctrico, por ejemplo.

Los sistemas de este tipo existentes en la actualidad están orientados al desarrollo de aplicaciones OLTP, por lo que únicamente suelen suministrar servicios básicos de procesamiento de información. Aquí se propone confeccionar un sistema de este tipo adaptado a las necesidades de las aplicaciones OLAP y de los problemas de extracción de conocimiento en bases de datos. Por extensión, un sistema como el planteado en este capítulo también resulta adecuado para un gran abanico de aplicaciones científicas y técnicas que requieren una elevada capacidad de cómputo sólo obtenible con supercomputadores o sistemas distribuidos como los descritos en la sección anterior de este capítulo.

Los sistemas basados en componentes actuales, como los Enterprise JavaBeans o los componentes COM+, suelen incluir entre sus servicios la posibilidad de realizar transacciones distribuidas (conjuntos de operaciones que han de efectuarse de forma atómica, de modo que, si algunas de ellas no se completa con éxito, los efectos que hayan producido las demás han de anularse). En un sistema multiagente como el propuesto aquí este servicio es innecesario. Sin embargo, se necesitan otra serie de servicios que usualmente no ofrecen los sistemas existentes: son imprescindibles mecanismos de planificación y asignación de recursos que permitan afinar el rendimiento del sistema distribuido y también son necesarios mecanismos de monitorización y notificación que permitan gestionar la ejecución de las tareas que realizan los agentes del sistema. Así mismo, también se necesitan mecanismos que doten al sistema de mayor flexibilidad, de forma que sea capaz de reconfigurarse sobre la marcha y evolucionar [9].

Los servidores de aplicaciones existentes en el mercado desempeñan las funciones del contenedor en el modelo de la figura 6.9. Aunque sin duda son adecuados para proporcionar soluciones de comercio electrónico, como demuestra su popularidad, tanto en sistemas B2B (*business-to-business*) como aplicaciones B2C (*business-to-customer*), los sistemas actuales carecen de los mecanismos de control adicionales que requieren las aplicaciones de cálculo intensivo como las de extracción de conocimiento en bases de datos. Los sistemas basados en componentes diseñados para este tipo de aplicaciones deben incluir un núcleo que proporcione la capacidad de monitorizar el uso de recur-

sos en el sistema (tanto tiempo de CPU como ocupación de memoria, espacio de almacenamiento disponible o ancho de banda consumido por las operaciones de entrada/salida), gestionar la evolución dinámica del sistema y, a la vez, ofrecer a los programadores un entorno lo suficientemente flexible en el que diseñar sus algoritmos y desarrollar nuevas técnicas de computación distribuida. En el siguiente apartado se comentan brevemente algunas de las características que debe poseer el núcleo de un sistema basado en componentes apto para el tipo de aplicaciones que nos interesa (esto es, aplicaciones OLAP de ayuda a la decisión).

6.3.2. El kernel del sistema

El núcleo o kernel de un sistema basado en componentes como el descrito en el apartado anterior debe ejecutarse en cada una de las máquinas que forman el sistema distribuido. Dicho núcleo, al que de aquí en adelante denominaremos ETREK*, es responsable de la realización efectiva de las siguientes tareas:

- Respecto a los clientes del sistema (que pueden ser usuarios accediendo al mismo a través de una interfaz web u otros programas que contratan los servicios del sistema), ETREK se encarga de
  - gestionar las sesiones abiertas (requiriendo la autentificación de los usuarios cada vez que se conectan al sistema),
  - procesar las solicitudes de los usuarios (por ejemplo, construir un modelo mediante la realización de una tarea de *Data Mining* por parte de un agente),
  - recopilar metainformación acerca de los conjuntos de datos que se utilizan como punto de partida en el proceso de extracción de conocimiento, y

*ETREK es un acrónimo que proviene de *EnTerprise Run-time Environment Kernel*, el núcleo de un sistema basado en componentes, aún en fase de diseño, que tiene su origen en el desarrollo de *TMiner*, un sistema de *Data Mining* que incluye algoritmos de extracción de reglas de asociación, construcción de clasificadores y algunos métodos de agrupamiento.

  - dar acceso a los modelos de conocimiento que hayan sido obtenidos con anterioridad por el usuario o por otros usuarios del sistema (teniendo en cuenta posibles restricciones de acceso).

- En relación con los agentes del sistema (procesos encargados de realizar las tareas de cálculo necesarias para la obtención de modelos de conocimiento), ETREK se encarga de la planificación y asignación de recursos, así como de la monitorización del funcionamiento del sistema. También es responsable de la notificación al usuario de la terminación de una tarea cuando éste así lo solicite. Dicha notificación puede realizarse via correo electrónico, por ejemplo.

- Finalmente, la instancia de ETREK que se ejecuta en un nodo particular de un sistema distribuido es responsable de otras tareas relativas al funcionamiento interno del propio sistema:

  - El kernel tiene la misión de gestionar la base de datos que da soporte al servicio de persistencia del sistema: el almacén de información en el que se guarda la metainformación acerca de los datos, los modelos de conocimiento obtenidos a partir de ellos y toda la información que sea necesaria para que el sistema sea fiable y tolerante a fallos, como las sesiones de usuario abiertas, los agentes en ejecución, etc..
  - El kernel del sistema, además, debe proveer los mecanismos necesarios para poder añadir nuevos componentes de forma dinámica sin paralizar su funcionamiento (despliegue “en caliente” o *hot deployment*).
  - Por otro lado, el kernel del sistema local ha de coordinar su trabajo con otras instancias de ETREK que se estén ejecutando en otros nodos del sistema distribuido. En primer lugar, debe mantener información relativa a otros nodos para ser capaz de transmitir datos siguiendo la ruta óptima (capacidad de enrutamiento) y de comprobar la disponibilidad y descubrir, en su caso, la presencia de nuevos recursos en la red (capacidad de descubrimiento), como

pueden ser nuevos nodos que se añaden dinámicamente a la red o nodos ya existentes que ofertan nuevos servicios. Así mismo, una instancia concreta de ETREK también ha de tener constancia de la carga que soportan sus nodos vecinos en el sistema para poder tomar las decisiones adecuadas que permitan balancear la carga del sistema.

ETREK debe realizar todas las funciones descritas en los párrafos anteriores de la forma más transparente posible de cara al usuario del sistema, con el objetivo de facilitarle su uso en la medida que sea posible. Transparencia y usabilidad son las dos cualidades básicas que el sistema ha de tener, y ambas determinan muchas de las decisiones concretas de diseño que han de tomarse para implementar un sistema que posea las características expuestas en esta sección. A continuación, en el siguiente apartado de esta memoria, se analizan algunas de esas decisiones de diseño y se comentan algunas de las tecnologías existentes en la actualidad que permiten la implementación de un sistema como el propuesto.

6.4. Diseño e implementación

Los grandes sistemas de software se encuentran entre los sistemas más complejos creados nunca por el hombre.

FREDERICK P. BROOKS
The Mythical Man-Month

En este apartado se discutirán distintos aspectos particulares relativos al diseño arquitectónico de un sistema que cumpla con los requisitos expuestos en las secciones anteriores y algunas de las tecnologías disponibles en la actualidad que permiten la implementación real del sistema.

En primer lugar, se comentarán brevemente los principios de diseño a los que resulta aconsejable dar prioridad para poder llevar a cabo con éxito proyectos de desarrollo de software de la envergadura del sistema propuesto. Es esencial en proyectos de cierto tamaño que todos las personas involucradas

compartan una visión común que les permita dirigir todos sus esfuerzos en la misma dirección [70], independientemente de las técnicas de Ingeniería del Software que se empleen para planificar y gestionar el proyecto de desarrollo [109] [110].

En el apartado 6.4.2 se incluye un ejemplo de la aplicación de los principios comentados. En concreto, se ha escogido el diseño de una parte fundamental del sistema: el subsistema que permite acceder a datos de fuentes potencialmente heterogéneas. El uso de conocidos patrones de diseño permite modelizar de una forma elegante los conjuntos de datos a los que se ha de acceder y permite aislar al resto del sistema de las peculiaridades que puedan presentar las distintas fuentes de datos.

La sección siguiente, el apartado 6.4.3, se centra en otro de los componentes esenciales del sistema propuesto: el servicio de persistencia. La utilización de modelos de datos [77] flexibles permite obtener un diseño sencillo y eficaz de la base de datos que se utiliza para almacenar de forma persistente toda la información del sistema.

Una vez estudiado el diseño de los mecanismos que facilitan el acceso a los datos y del servicio que permite almacenar datos de forma persistente, en el apartado 6.4.4 se comentará brevemente una de las tecnologías posibles que se podrían utilizar para implementar el sistema: la plataforma Java y todos los estándares que la rodean. Esta alternativa resulta especialmente adecuada para un sistema distribuido que ha de funcionar en un entorno heterogéneo en el que coexisten distintos sistemas operativos.

Esta sección, en la que se delinea una estrategia para la implementación de un sistema distribuido basado en componentes, se cierra con el apartado 6.4.5, en el que se muestra la configuración del sistema que podría resultar de la aplicación de las directrices que analizamos a continuación.

6.4.1. Principios de diseño

El esfuerzo de diseño requerido para el desarrollo del sistema propuesto en este capítulo debe estar siempre orientado hacia la consecución de dos cualidades: transparencia (6.4.1.1) y usabilidad (6.4.1.2). El grado de transparencia y la facilidad de uso del sistema determinan, sin duda alguna, las posibilidades de éxito de un proyecto de este tipo. Las soluciones aplicadas en proyectos previos, modeladas mediante patrones de diseño (6.4.1.3), ofrecen una inestimable ayuda en el diseño de nuevos sistema como en el caso que nos ocupa.

6.4.1.1. Transparencia

El sistema diseñado ha de ser lo más transparente posible, tanto para los usuarios del sistema como para los programadores que le añaden funcionalidad implementando nuevos componentes.

De cara a los programadores, el sistema ha de ofrecer una serie de interfaces bien definidos que les permitan desarrollar nuevos componentes con relativa facilidad. Estas interfaces han de ocultar la complejidad interna del sistema, ofreciendo una especificación de la funcionalidad del sistema independiente de la representación que se haya escogido en su implementación. Ocultar detalles de implementación mediante su encapsulación es un objetivo esencial en el desarrollo de cualquier sistema de cierta envergadura, para lo cual ha de aplicarse una estrategia “divide y vencerás”. Esta estrategia nos permite resolver problemas complejos descomponiéndolos en otros más simples. La descomposición realizada determina la estructura del sistema y, gracias a nuestra capacidad de abstracción, se eliminan detalles que dificultarían el diseño del sistema. La eliminación de los detalles adecuados es, obviamente, la que conduce a diseños de mayor calidad. Este proceso, no automatizable, es uno de los mayores atractivos del desarrollo de software.

Por otro lado, respecto a los usuarios finales del sistema, los cuales no tienen por qué ser conscientes de la complejidad del sistema al que acceden, también es importante que el sistema oculte su complejidad interna. Sólo así se podrá lograr el cumplimiento de nuestro segundo objetivo: el relacionado con la usabilidad del sistema.

6.4.1.2. Usabilidad

La facilidad de uso del sistema, su usabilidad [89], es esencial si queremos que tenga éxito un sistema distribuido basado en componentes orientado a la resolución de problemas de extracción de conocimiento. Como cualquier otro sistema software, un sistema así es utilizado por personas, de modo que su facilidad de uso determina el grado de aceptación del que llegan a gozar. Este factor es más crítico aún si tenemos en cuenta que los 'trabajadores del conocimiento' (los analistas y ejecutivos a los que se orienta el sistema propuesto) no suelen ser expertos en Informática.

Entre las funciones del sistema se ha de incluir la capacidad de almacenar cualquier tipo de información para que los usuarios puedan volver a utilizarla en el futuro, algo de lo que se encargará el servicio de persistencia descrito en el apartado 6.4.3.

Dado que el objetivo final del sistema es la obtención de modelos descriptivos y predictivos, es esencial que el conocimiento extraíble de ellos se represente y comunique de la forma adecuada. Es imprescindible que el sistema facilite a los usuarios mecanismos mediante los que puedan compartir la información que hayan obtenido en sus sesiones de trabajo. Por tanto, sería aconsejable incluir herramientas que permitan la colaboración entre grupos de usuarios (conocidas por el término inglés *groupware*). La implementación de estas herramientas ha de ser especialmente cautelosa respecto a cuestiones de seguridad: se ha de garantizar la privacidad de la información que cada usuario obtiene de sus datos. Una política de seguridad adecuada en esta situación es hacer que, por defecto, se niegue siempre el acceso de un usuario a cualquier documento o modelo creado por otro usuario distinto, salvo que este último le haya concedido privilegios de lectura o modificación.

6.4.1.3. Patrones de diseño

Por suerte, el diseño de un sistema de las características expuestas a lo largo de este capítulo es factible hoy en día gracias a la experiencia acumulada a lo largo de los últimos años en el desarrollo de complejos sistemas software. Distintos autores se han encargado de recopilar esta experiencia en forma

de patrones de diseño [62] [65] [77], los cuales recogen soluciones que han demostrado ser de utilidad en el desarrollo de software.

El conocimiento de patrones de diseño complementa al enfoque educativo tradicional que se centra en el estudio de metodologías y técnicas de resolución de problemas para el análisis y diseño de software, como, por ejemplo, las descritas en las excelentes colecciones de ensayos de Bentley [15] o Plauger [126].

Uno de los patrones de diseño más conocidos es el modelo MVC (Modelo-Vista-Controlador) en el cual se separan físicamente los módulos o clases que modelizan información (modelos) de aquéllos que permiten acceder a ellos de distintas formas (vistas), los cuales se enlazan a sus respectivos modelos utilizando mecanismos auxiliares que también se implementan de forma independiente (controladores). La arquitectura resultante de la aplicación de este modelo es excelente porque produce una buena modularización del código (alta cohesión y acoplamiento débil), lo que facilita el mantenimiento del sistema.

Como se mencionará en el apartado 6.4.4, una implementación en Java del modelo MVC como Struts [44] puede facilitar la infraestructura necesaria para desarrollar la interfaz de usuario del sistema planteado en este capítulo. No obstante, antes de profundizar en posibles vías de implementación, analizaremos en los apartados siguientes el diseño de dos partes fundamentales del sistema que nos ocupa: las que proporcionan acceso a los datos (6.4.2) y el servicio de persistencia (6.4.3).

6.4.2. Modelización de conjuntos de datos

Consideremos, en primer lugar, el problema de acceder a los conjuntos de datos que sirven de entrada a los algoritmos ejecutados por los agentes del sistema para construir modelos, ya sean éstos predictivos o descriptivos.

Muchas herramientas de extracción de conocimiento, especialmente aquéllas que implementan técnicas de aprendizaje automático, trabajan con conjuntos de datos en forma de tablas (en el sentido del término empleado por las bases de datos relacionales), si bien es cierto que algunos sistemas existentes funcionan directamente sobre bases de datos multidimensionales [75]. Cada tabla contiene un conjunto de tuplas de longitud fija que se puede obte-

ner de una base de datos relacional utilizando interfaces estándares de acceso a bases de datos como ODBC o JDBC. Los datos también pueden provenir de otras fuentes, como pueden ser servidores DSTP [Data Space Transfer Protocol, National Center for Data Mining, University of Illinois at Chicago, 2000], documentos en formato XML [eXtensible Markup Language] o simples ficheros ASCII, por ejemplo.

Todos los conjuntos de datos tabulares incluyen un conjunto de columnas a las cuales se les puede denominar atributos o campos. A las distintas columnas de los conjuntos de datos se les suele asociar un identificador único para poder hacer referencia a ellas y un tipo que nos indica el dominio de sus valores (cadenas, números, fechas, etc.). Además, se debe permitir la definición de relaciones de orden entre los valores de los distintos atributos y se ha de facilitar la posibilidad de agrupar dichos valores para formar jerarquías de conceptos.

Por otro lado, hemos de tener en cuenta que los conjuntos de datos pueden provenir de fuentes de distinta naturaleza, a pesar de lo cual sigue siendo imprescindible interpretarlos de una forma uniforme que facilite su utilización y nos permita implementar algoritmos independientes del formato particular que tengan los datos. Esto es, el sistema ha de encargarse de gestionar de una forma transparente para los agentes los datos que provienen de fuentes heterogéneas.

El subsistema de acceso a los datos debería, por tanto, ser capaz de efectuar consultas heterogéneas que accedan a distintas bases de datos y fuentes de información. Los conjuntos de datos a los que se accede de forma independiente, aun proveniendo de fuentes heterogéneas, han de poder combinarse unos con otros con el objetivo de estandarizar la representación de conceptos (integración), eliminar redundancias y errores (limpieza), agregar información (resumen) o, simplemente, eliminar la parte de los datos que no sea de nuestro interés (filtrado).

Todas las operaciones mencionadas arriba podrían efectuarse utilizando modelos formales y lenguajes de consulta. No obstante, los usuarios típicos del sistema puede que no gocen de la preparación necesaria para dominar tales lenguajes de consulta y poder definir sin ayuda externa los conjuntos de datos que necesiten. Dichos usuarios, probablemente descartarían directamente el uso de un sistema que les exija conocer cualquier tipo de lenguaje de

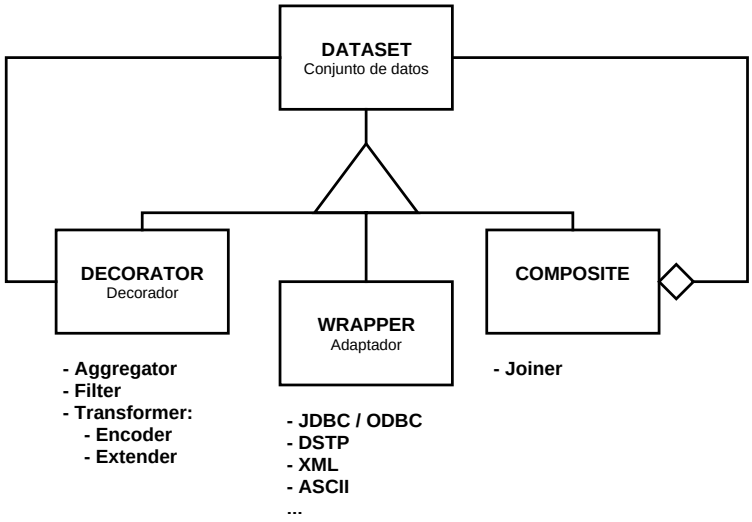

Figura 6.10: Diagrama de clases que ilustra el patrón de diseño utilizado en la modelización de los conjuntos de datos.

especificación de consultas.

Con el objetivo de facilitar la aceptación de un sistema en el que no se limite la capacidad de formular consultas complejas y, a la vez, resulte fácil de utilizar por usuarios con distinta preparación, en esta sección se propone utilizar algunos de los patrones de diseño estructurales más conocidos [65]. En concreto, se pueden emplear los patrones de diseño *wrapper* (adaptador), *decorator* (decorador) y *composite* (compuesto) para modelar cualquier conjunto de datos tal como se muestra en el diagrama de clases de la figura 6.10.

En vez de tener que emplear un lenguaje de consulta más o menos complejo, el usuario sólo tendrá que ir construyendo de forma ascendente los conjuntos de datos que desee utilizar. Para ello, el usuario empleará una familia de componentes de construcción de conjuntos de datos que le proporcionarán los mecanismos necesarios para construir sus propios conjuntos de datos personalizados a partir de las fuentes de datos disponibles (p.ej., bases de datos o ficheros en un formato determinado).

Los componentes que le permiten al usuario construir sus propios conjun-

tos de datos se describen a continuación[**]:

- Los adaptadores (***wrappers***) son responsables de facilitar un interfaz de acceso uniforme a fuentes de datos de distintos tipos. Cuando los datos se almacenan en bases de datos relacionales, los conjuntos de datos pueden expresarse como el resultado de realizar consultas SQL a través de interfaces estándares como ODBC, JDBC o BDE. Tales interfaces son independientes de la base de datos particular y permiten acceder a la mayor parte de los sistemas gestores de bases de datos existentes en la actualidad (Oracle, IBM DB2, Microsoft SQL Server, InterBase...). Cuando los datos se almacenan en otros formatos (localmente en ficheros ASCII o XML, o de forma remota en servidores DSTP, por ejemplo), se requieren adaptadores específicos para cada formato. Como es lógico, cada tipo de fuente de información necesita su propio adaptador específico.

- Los integradores (***joiners***) se emplean para reunir múltiples conjuntos de datos independientes, interpretando esta reunión en el sentido del álgebra relacional. Estos componentes, provenientes del uso del patrón de diseño *composite* (compuesto), permiten combinar la información que proviene de diferentes fuentes. Con ellos se puede añadir información a los registros de un conjunto de datos particular (como cuando su utiliza un *data warehouse* con un esquema en estrella) y también permiten establecer relaciones maestro/detalle entre dos conjuntos de datos.

- Los agregadores (***aggregators***) nos sirven para resumir los conjuntos de datos disponibles para poder obtener una visión más general de ellos. Las agregaciones son especialmente útiles en muchas aplicaciones de análisis OLAP, en las cuales las tendencias generales presentes en los datos son mucho más interesante que los detalles particulares. Las funciones de agregación más comunes incluyen contar el número de datos existentes (COUNT), sumarlos (SUM), calcular su media aritmética

[**]En este apartado, como sucede en otras partes de este capítulo, aparecen determinados anglicismos por ser de uso habitual y se mantienen en el texto para no desorientar al lector que esté familiarizado con ellos al utilizar traducciones que puedan llevar a confusión.

(AVG) y obtener la varianza (VAR), así como recuperar el valor mínimo (MIN), el valor máximo (MAX), los valores más altos (TOP) y los valores más bajos (BOTTOM).

- Los filtros (***filters***) se emplean para seleccionar parte de un conjunto de datos y quedarnos con un subconjunto del conjunto de datos original. Desde el punto de vista del álgebra relacional, los filtros nos permiten realizar proyecciones (escoger columnas determinadas de un conjunto de datos) y selecciones (quedarnos con algunas de las tuplas del conjunto de datos). En aplicaciones de *Data Mining*, los filtros pueden usarse para muestrear datos, formar conjuntos de entrenamiento y prueba en un proceso de validación cruzada o, simplemente, para elegir una parte de los datos cuyo procesamiento posterior pueda resultar de interés para el usuario.

- Los transformadores (***transformers***) también son necesarios para que el usuario pueda modificar las columnas de un conjunto de datos. Dentro de ellos, se pueden identificar dos tipos principales:

  - Los codificadores (***encoders***) se emplean para codificar conjuntos de datos. Por ejemplo, se pueden usar para establecer un formato de codificación uniforme en la representación de información que, aun teniendo un significado único, se puede presentar de distintas formas según cuál sea su procedencia. De hecho, es frecuente que una entidad determinada pueda tener distintas representaciones en un mismo conjunto de datos. Por tanto, los codificadores resultan esenciales para realizar tareas de limpieza e integración de datos.
  - Los extensores (***extenders***), por su parte, permiten añadir nuevas columnas a un conjunto de datos. Esos atributos adicionales, a los que se les suele denominar campos calculados, son útiles para convertir unidades de medida y, sobre todo, para gestionar fechas (p.ej. se puede obtener el día de la semana, la semana del año, el mes, el trimestre o la temporada correspondiente a una fecha concreta). El valor de un campo calculado ha de venir completamente

determinado por los valores de los demás campos de un registro o tupla (en caso contrario, tendríamos que utilizar un agregador o un integrador en vez de un extensor). Normalmente, el cómputo de un campo calculado se especifica utilizando una expresión aritmética (con operadores como +, -, * o /), si bien también se suelen permitir funciones predefinidas y expresiones más complejas que involucren sentencias condicionales (del tipo *if-then-else*, por ejemplo).

La familia descrita de componentes permite que el usuario construya sus propios conjuntos de datos agregando unos componentes con otros. Al combinar los distintos tipos de componentes, se puede construir una estructura en forma de árbol análoga a la que construiría el planificador de consultas de un sistema de bases de datos. Dada esta analogía, la estructura en árbol creada por el usuario es apta para la aplicación de las técnicas estándar de optimización de consultas [118], con lo que se puede mejorar notablemente el rendimiento del sistema a la hora de acceder a los datos.

La consecuencia más importante de esta forma de acceder a los datos, que ofrece la misma flexibilidad que cualquier lenguaje de consulta por complejo que éste sea, es que el usuario puede enlazar fácilmente los componentes descritos para modelar cualquier conjunto de datos que desee utilizar.

6.4.3. Servicio de persistencia

El servicio de persistencia ha de encargarse de almacenar de una forma fiable toda la información utilizada por el sistema, tanto la metainformación relativa a los conjuntos de datos como los modelos obtenidos por los distintos usuarios y el estado actual del sistema. Para ello puede emplear una base de datos de apoyo en la que se almacenen datos sobre los usuarios y grupos de usuarios del sistema, los permisos de acceso de los que goza cada uno de ellos, los modelos y los conjuntos de datos utilizados por cada usuario, las sesiones activas en el sistema, las tareas pendientes (esto es, los agentes que se encuentran en ejecución) y cualquier otra información referente a la configuración del sistema.

Siguiendo la filosofía de los patrones de diseño, en vez de diseñar una base de datos de la forma tradicional (creando una tabla para cada tipo de entidad que pueda existir en el sistema), se puede diseñar una base de datos que permita la evolución de los componentes existentes en el sistema y facilite su mantenimiento. En este tipo de diseños se suele diferenciar un nivel operacional, que incluye los datos propiamente dichos, de un nivel de conocimiento en el que se guarda metainformación acerca de los datos almacenados en la base de datos. De esta forma, si el esquema lógico de la base de datos se ve modificado, sólo es necesario modificar el contenido de las tablas del nivel de conocimiento, sin tener que alterar el esquema físico de la base de datos. En cierta medida, este tipo de diseños utiliza las mismas ideas en que se basa la implementación del catálogo en los sistemas modernos de gestión de bases de datos.

El diagrama entidad/relación de la figura 6.11 muestra un diseño flexible que no requiere modificación alguna sean cuales sean los cambios que se produzcan en el sistema.

En esta base de datos es imprescindible almacenar información acerca de los usuarios por motivos de seguridad. Una clave de acceso, que siempre ha de almacenarse encriptada utilizando algoritmos como MD5 o SHA, es la que permite al usuario autentificarse y acceder a los recursos del sistema. También resulta aconsejable mantener información acerca del uso que el usuario hace del sistema, para poder monitorizar su funcionamiento y ser capaz de detectar situaciones anómalas que puedan provenir de accesos no autorizados.

En el diseño propuesto, el usuario creará sesiones de trabajo en las cuales podrá almacenar todo tipo de información, como veremos más adelante. Las sesiones creadas por él serán de su propiedad y solamente él tendrá la capacidad de dar acceso a la información de la sesión a otros usuarios del sistema.

Los usuarios del sistema se agrupan en comunidades o grupos de usuarios que pueden compartir sesiones, de forma que el usuario propietario de una sesión puede permitir el acceso a esa sesión únicamente a las grupos de usuarios que él decida. A través del acceso compartido a sesiones, el usuario propietario de una sesión concede, de forma indirecta, permisos de acceso a toda la información relativa a su sesión. De esta forma, los conjuntos de datos y modelos obtenidos a partir de ellos pueden compartirse entre grupos de usuarios.

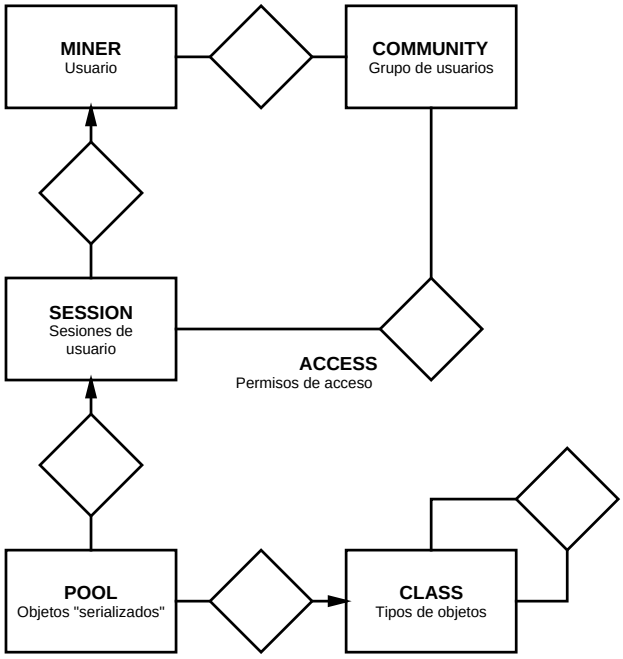

Figura 6.11: Diagrama entidad/relación de la base de datos que da soporte al servicio de persistencia y permite definir permisos de acceso para grupos de usuarios del sistema.

La información almacenada referente a cada sesión de usuario se guarda dividida en dos partes: un almacén de objetos "serializados" (nivel operacional) y una jerarquía de tipos asociados a dichos objetos que nos permite navegar por el contenido de la base de datos de cada sesión (nivel de conocimiento). Al almacenar información relativa a los tipos de los objetos presentes en la base de datos, este sencillo diseño proporciona capacidades de introspección al sistema. Además, el sistema resulta extremadamente flexible, pues no es necesario modificar nunca el esquema de la base de datos aunque cambien los tipos de componentes existentes en el sistema.

En esta base de datos se almacena toda la información de la que dispone el usuario acerca de los conjuntos de datos que emplea, así como los distintos modelos que haya ido obteniendo con anterioridad, de forma que puede reutilizarlos en el futuro si así lo desea. Por ejemplo, el usuario podría almacenar modelos de clasificación para ser capaz de clasificar nuevos datos en el futuro o podría utilizar los resultados producidos por un método de agrupamiento para seleccionar el fragmento de los datos correspondiente a uno de los agrupamientos y analizarlo utilizando cualquiera de las técnicas de las que disponga en el sistema.

Aparte de la información con la que conscientemente trabaja el usuario, la base de datos que da apoyo al servicio de persistencia también ha de mantener las sesiones activas y las tareas pendientes en cada momento con el objetivo de que el sistema sea tolerante a fallos. De esta forma, se puede reducir el impacto que podría tener en el sistema un corte de suministro eléctrico, por ejemplo.

6.4.4. Implementación del sistema en Java

En las secciones anteriores se han analizado algunas cuestiones relativas al diseño de la infraestructura necesaria para que el sistema funcione. En esta sección se comentará brevemente un conjunto de tecnologías existentes que permite la implementación de un sistema como el descrito en este capítulo.

La plataforma Java, que incluye el lenguaje de programación Java, su amplia gama de bibliotecas estándar y un entorno de ejecución portable (la máquina virtual Java), resulta especialmente adecuada para desarrollar aplicaciones que hayan de funcionar en sistemas heterogéneos porque existen máquinas

virtuales Java para casi todos los sistemas operativos existentes en el mercado (como Windows, Linux y las distintas variantes de UNIX). Este hecho permite que el código escrito en un entorno de desarrollo particular pueda funcionar en multitud de sistemas operativos sin requerir modificaciones de ningún tipo. Dado que el sistema distribuido propuesto en este capítulo debería funcionar en sistemas heterogéneos, Java parece ser una buena opción a la hora de implementarlo.

Java resulta especialmente adecuado para el desarrollo de aplicaciones distribuidas en Internet o en intranets por el amplio apoyo que su gama de bibliotecas estándar ofrece al programador de aplicaciones, lo que permite utilizar distintos modelos de cómputo distribuido y paralelo [121] [155]. Mediante el uso de RMI [*Remote Method Invocation*], Java proporciona la posibilidad de realizar llamadas a métodos remotos y construir sistemas distribuidos. La propia arquitectura de Java, con su cargador de clases dinámico, su capacidad de introspección y su modelo de componentes (JavaBeans), facilita el desarrollo de sistemas dinámicos basados en componentes. Además, tecnologías relacionadas como Jini [33] [149] permiten crear con facilidad la infraestructura necesaria para la implementación de un sistema como el propuesto en este capítulo. En la plataforma Java se puede encontrar la misma funcionalidad que ofrecen los servicios web citados como estándares actuales en el desarrollo de sistemas distribuidos (sección 6.2). A grandes rasgos, RMI, JavaBeans y Jini se corresponden con SOAP, WSDL y UDDI, respectivamente.

Gracias a su independencia de la plataforma sobre la que se ejecuta, un sofisticado modelo de seguridad, RMI y la capacidad de "serializar" objetos (esto es, la capacidad de empaquetar los objetos para su almacenamiento o transmisión y posterior restauración en la misma o en otra máquina virtual), el lenguaje Java ha sido escogido como estándar de facto para el desarrollo de sistemas multiagente [25]. Aglets (de IBM), Grasshopper (IKV++), MOLE (Universidad de Stuttgart), Paradigma (Universidad de Southampton), RONIN (Universidad de Maryland), Voyager (ObjectSpace), Concordia (Mitsubishi Electric ITA Horizon Labs), Odyssey (General Magic) o Jumping Beans (Ad Astra) son sólo algunos de los sistemas multiagente que ya se han implementado utilizando Java.

Respecto a la ejecución de los agentes en un sistema como el analizado en este capítulo, hay que tener en cuenta varios aspectos si finalmente se decide utilizar Java:

- **Movilidad:** La movilidad de los agentes desarrollados en Java está limitada, porque no se puede almacenar el estado de un agente en tiempo de ejecución para que después restaure su estado y prosiga su ejecución por donde fuese. En Java se puede serializar el estado de un objeto, pero no su entorno de ejecución (la pila de la hebra correspondiente al agente). No es, pues, posible la implementación de un mecanismo de *persistencia ortogonal* en Java (aquél que permitiese mover un agente de un nodo a otro del sistema distribuido y reanudar su ejecución de forma transparente). Es más, la migración de un agente de un nodo a otro no es transparente a no ser que se utilicen versiones especializadas de la máquina virtual Java (p.ej. Aroma).

- **Seguridad interna (control de acceso):** Java, con su modelo de seguridad basado en permisos, sí resulta especialmente adecuado para construir sistemas distribuidos reconfigurables, con políticas de seguridad flexibles, fácilmente adaptables y extensibles que permitan controlar localmente la ejecución del software.

- **Seguridad externa (transmisión de datos):** Las bibliotecas estándar de Java incluyen implementaciones de todo tipo de técnicas criptográficas de protección de datos que permiten un intercambio seguro de información confidencial a través de un medio compartido con otros usuarios y sistemas. En el caso de Internet, la seguridad se suele conseguir empleando SSL [*Secure Sockets Layer*], que utiliza técnicas criptográficas de clave pública.

- **Monitorización del uso de recursos: tiempo de CPU, memoria y ancho de banda.** Es quizá uno de los puntos más débiles de la plataforma Java, que ha dado lugar a la propuesta de extensiones como JRes [42] [43]. El recolector de basura de Java, por ejemplo, ofrece poco control sobre el uso real de memoria.

Característica	Tecnologías y técnicas
Portabilidad	Java
	HTML dinámico estándar
Interfaz	Struts [44]
	Servlets & Pushlets
	JavaMail
Interoperabilidad	JDBC [Java DataBase Connectivity]
	Familia de protocolos TCP/IP: HTTP, SMTP...
Concurrencia	Hebras
Distribución	RMI [Remote Method Invocation]
	Jini
	Arquitectura multicapa (proxies)
Movilidad	Serialización
Seguridad	Claves encriptadas (SHA o MD5)
	HTTPS (SSL [Secure Socket Layer])
Persistencia	Base de datos de apoyo (figura 6.11)
Soporte multilingüe	Ficheros de recursos

Tabla 6.1: Algunas de las características deseables del sistema y las técnicas que facilitan su implementación.

- **Flexibilidad:** El cargador de clases dinámico de la máquina virtual Java y la capacidad de introspección de los objetos son, sin duda, las mayores bazas de Java frente a otros lenguajes monolíticos como C o C++.

En [142] se puede leer un interesante artículo que describe algunas de las mejores y de las peores características de Java con las que uno se topa cuando se quiere implementar un sistema de procesamiento de consultas cuyos requerimientos son similares en cierta medida a los del sistema propuesto en este capítulo. La tabla 6.1 resume algunas de las características deseables del sistema, así como las tecnologías y técnicas que permiten que la implementación del sistema sea factible.

6.4.5. Despliegue del sistema

Dado que un sistema como el descrito en este capítulo no puede implementarse utilizando servidores de aplicaciones tradicionales, orientados al desarrollo de aplicaciones OLTP, ha sido necesario diseñar un sistema distribuido de propósito especial, que no de utilidad limitada. El principal rasgo diferencial del sistema propuesto es su capacidad para controlar mejor los recursos computacionales de los se dispone en un entorno distribuido.

La figura 6.12 muestra la configuración que un sistema así podría tener si se siguen las directrices expuestas en los apartados anteriores. En el sistema de la figura, un servidor web Apache equipado con un contenedor de servlets sirve de interfaz del sistema con el usuario a través de web, mientras que la base de datos empleada por el servicio de persistencia puede residir en cualquier sistema gestor de bases de datos relacionales.

6.5. Una mirada hacia el futuro

La predicción es difícil, especialmente cuando se trata del futuro
NIELS BÖHR

En este capítulo se han abordado distintas cuestiones que han de resolverse para poder implementar un sistema distribuido basado en componentes que resulte adecuado para tareas de extracción de conocimiento en bases de datos y, en general, para cualquier aplicación de cálculo intensivo.

Los servidores de aplicaciones comerciales, máximos exponentes hoy en día de los sistemas distribuidos basados en componentes, se restringen al campo de las aplicaciones OLTP, las cuales realizan un procesamiento de datos a relativamente bajo nivel. Este cápítulo, utilizando el mismo modelo arquitectónico que rige a estos sistemas, propone la implementación de sistemas más capaces, sistemas capaces de realizar tareas computacionalmente más complejas que tradicionalmente se han asociado a otras arquitecturas de propósito específico y a sistemas OLAP en los que se implementan aplicaciones de ayuda a la decisión.

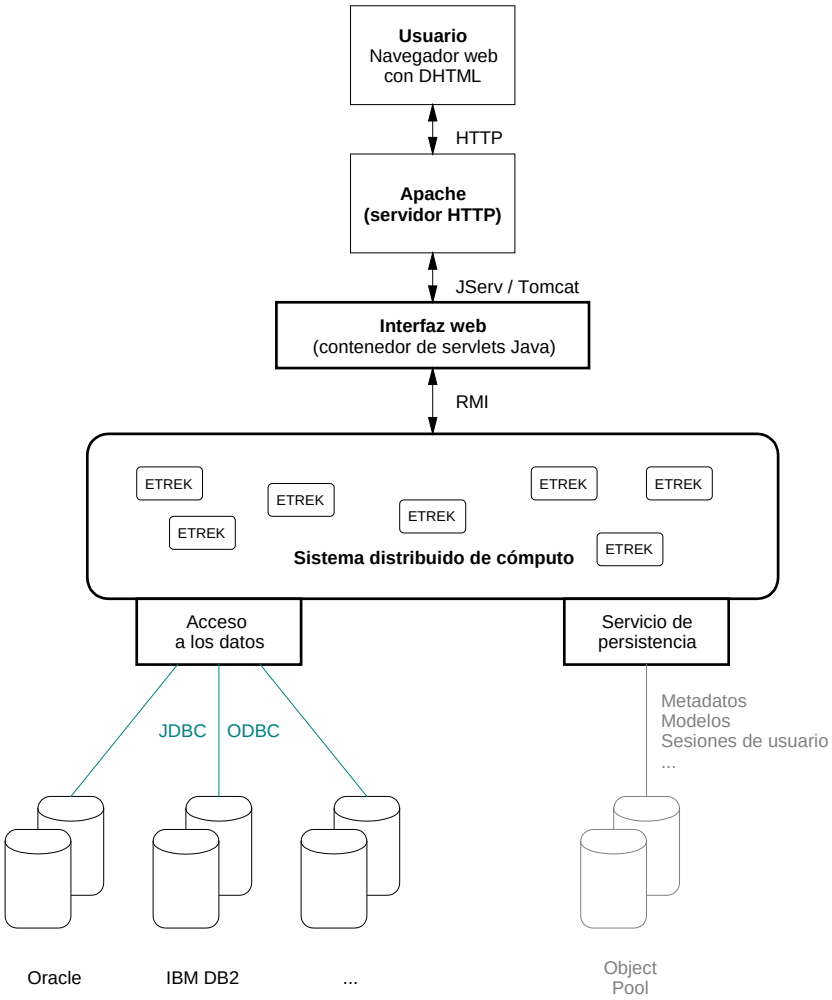

Figura 6.12: Diagrama de despliegue que muestra una posible configuración del sistema.

El modelo de cómputo utilizado en la actualidad infrautiliza los recursos de los que disponemos conectados a redes locales, intranets corporativas o la red de redes, Internet, por lo que no resulta del todo descabellado centrar nuestros esfuerzos en el desarrollo de nuevos sistemas que aprovechen al máximo la capacidad de cálculo de los ordenadores personales autónomos que se hallan interconectados por modernas redes de comunicación y transmisión de datos. El modelo expuesto en este capítulo puede ser de utilidad como base para construir un supercomputador virtual, flexible y potente. Flexible porque se utiliza un modelo basado en componentes y potente porque puede aprovechar los recursos no utilizados de los dispositivos conectados a los sistema distribuidos (tiempo de CPU y espacio de almacenamiento, principalmente).

Quién sabe si algún día sistemas como el propuesto en este capítulo harán de la capacidad de cómputo y almacenamiento de datos un servicio público como puede ser el suministro eléctrico en el mundo desarrollado actual. Solamente el tiempo decidirá si algo así puede llegar a suceder.