UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS FÍSICAS

Departamento de Informática y Automática

TESIS DOCTORAL

Desarrollo de sistemas de ayuda inteligente mediante integración de tecnologías y reutilización de información

TESIS DOCTORAL

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Baltasar Fernández Manjón

Directores:

Alfredo Fernández-Valmayor Crespo
Luis Hernández Yáñez

Madrid, 2002

ISBN: 978-84-669-0404-9

DESARROLLO DE SISTEMAS DE AYUDA INTELIGENTE MEDIANTE INTEGRACIÓN DE TECNOLOGÍAS Y REUTILIZACIÓN DE INFORMACIÓN

Memoria que presenta para optar al grado de Doctor en Ciencias Físicas

Baltasar Fernández Manjón

Dirigida por los profesores

Alfredo Fernández-Valmayor Crespo

Luis Hernández Yáñez

Departamento de Informática y Automática
Facultad de Ciencias Físicas
Universidad Complutense de Madrid

Noviembre 1996

AGRADECIMIENTOS

Mi más profundo agradecimiento a Alfredo Fernández-Valmayor y a Luis Hernández Yáñez, quienes me iniciaron en el mundo de las aplicaciones educativas de las computadoras. Sin su conocimiento, dedicación y paciencia este trabajo no hubiera sido posible. A Carmen Fernández Chamizo por su constante consejo, tanto humano como técnico, y su constructiva visión crítica. A Antonio Vaquero por su apoyo en todo momento y como pionero en este campo educativo. A Manuel Ortega, cuyas animadas discusiones siempre han mantenido un agradable clima de trabajo.

Gracias a Manuel de Buenaga Rodríguez y a Pedro González Calero. Sus aportaciones a este trabajo han sido innumerables y de toda índole. Ellos han contribuido con ideas técnicas, teóricas, así como colaborado en el contraste y análisis de las distintas fases de este trabajo.

A Juan Manuel Cigarrán, Juan José Blanco y el resto de personas que han intervenido, de algún modo, en el desarrollo y prueba de los sistemas Argos, Aran y Copérnico. A Juan Lanchares, Carlos Villarrubia Jiménez, Bonifacio de Andrés y Toro, José María Gómez Hidalgo, Mercedes Gómez Albarrán, Ana Fernández-Pampillón y a todos los compañeros del grupo de trabajo del Departamento de Informática y Automática que, con un ambiente inmejorable, han contribuido al desarrollo de esta investigación.

A José Antonio Sánchez Fernández y a Xavier Menéndez-Pidal que me apoyaron especialmente durante mi periodo como profesor en la Universidad Politécnica de Madrid.

A todos aquellos investigadores que me han proporcionado información complementaria sobre su trabajo. A algunos de ellos he tenido el placer de conocerlos personalmente, como a los profesores A. Bork, A. Kobsa, Y. Gil o A. Puerta. Otros como J. Breuker o M. Hecking, aún no conociéndome, amablemente han respondido a mis peticiones.

Gracias especialmente a mi familia y todos los amigos de los que he recibido un apoyo incondicional. Todos ellos me han ayudado a soportar y a superar los momentos difíciles durante estos años de trabajo, por lo que merecen mi más sincero agradecimiento.

A mis padres

INDICE

CAPITULO 1

INTRODUCCION

Parece claro que cada vez resulta más importante que los usuarios de entornos informáticos extensos puedan disponer de ayuda que facilite la utilización y la comprensión de dichos entornos. En este trabajo proponemos que la mejor forma de proporcionar esta ayuda es mediante sistemas de ayuda inteligente[1]. Hoy en día, el uso de las computadoras se ha generalizado a muchas facetas de la vida diaria. Las computadoras se utilizan como una herramienta para la realización de tareas muy diversas, de forma que su uso se ha hecho algo habitual para todo tipo de personas y no sólo para expertos. Además, las aplicaciones informáticas cada vez presentan una complejidad mayor, ya que continuamente incorporan nuevas funcionalidades y permiten operaciones más sofisticadas. Estas circunstancias hacen que, a pesar de las mejoras en la tecnología de interacción hombre-computadora, resulte difícil diseñar una interfaz que haga que estos sistemas, inherentemente complejos, sean fáciles de manejar y de entender por un usuario que no disponga de ningún otro tipo de ayuda adicional. Lo cual implica que sin un buen sistema de ayuda, el usuario debe dedicar mucho tiempo al aprendizaje previo del funcionamiento de los programas y aplicaciones, o debe limitarse a usar sólo un pequeño porcentaje de su potencial.

El análisis de la arquitectura y funcionalidades de los tutores inteligentes (ITS) constituye nuestro punto de partida en la construcción de sistemas de ayuda inteligente. Aunque actualmente los ITS no se utilizan de una forma generalizada en la educación, en nuestra opinión, se trata de una línea de trabajo prometedora y constituye la base de nuestra propuesta. El campo educativo es una de las áreas en las que desde un principio más se han utilizado las computadoras, proponiéndose que éstas fuesen un complemento o incluso un sustituto de los tutores humanos. Entre las aplicaciones educativas más avanzadas cabe destacar los ITS que son aquellas aplicaciones que utilizan técnicas de inteligencia artificial en el desarrollo de sistemas de aprendizaje. Los ITS construyen las secuencias instructivas para

---

[1] Como traducción de *Intelligent Help Systems* que es el término más habitual con el que se nombran estos sistemas en la bibliografía. También se podrían haber utilizado otras denominaciones como sistemas inteligentes de ayuda o ayudas inteligentes.

explicar una determinada materia, adecuada a cada alumno concreto, a partir de representaciones explícitas (o modelos) del dominio, del alumno y de la experiencia tutorial.

Los sistemas de ayuda (o asistentes) tienen en principio un campo de aplicación muy amplio, por lo que en nuestro trabajo nos limitaremos a los sistemas de ayuda para aplicaciones o entornos software de propósito general. Es decir, aplicaciones tales como los sistemas operativos, que son utilizadas por muy diversos tipos de usuarios. En este contexto, un asistente es un programa que proporciona ayuda inmediata al usuario sobre los problemas concretos que se le presentan en el uso de la aplicación, con el fin de que pueda completar la tarea que está realizando. Dentro de la amplia variedad de programas de ayuda posibles nos centraremos en los sistemas de ayuda inteligente. Es decir, en aquellos sistemas con capacidad de adaptación a las circunstancias (por ejemplo, al usuario o a la tarea) en las que se solicita la asistencia. Esta capacidad de adaptación, al igual que sucede con los ITS, se basa en la utilización de modelos explícitos, tanto del dominio como del usuario, como base para proporcionar la ayuda. En nuestra propuesta, consideramos que ofrecer ayuda inteligente a los usuarios puede ser una solución adecuada al problema que plantea la demanda de facilidad de uso frente a la creciente complejidad de los sistemas informáticos.

Con este trabajo se trata de demostrar que es posible proporcionar ayuda efectiva a los usuarios de aplicaciones informáticas complejas a un coste razonable. Nuestro estudio se centra en la forma de suministrar asistencia a los usuarios de las aplicaciones manteniendo un enfoque instruccional. De este modo, al mismo tiempo que se proporciona ayuda, se aprovecha esta interacción para ampliar el conocimiento que dichos usuarios tienen sobre el sistema que utilizan. Para desarrollar nuestra propuesta, comenzamos presentamos un modelo de sistema de ayuda. El objetivo de este modelo es facilitar la construcción de sistemas de ayuda eficaces, aplicables a dominios reales y que tengan un coste razonable de desarrollo y mantenimiento. Nuestra propuesta nos permite construir sistemas de ayuda pasivos, que son activados por el usuario cuando encuentra un problema y que facilitan el acceso a información de referencia sobre la aplicación. El uso de un modelo explícito del usuario es un punto clave para adecuar la información presentada a cada usuario concreto. Su desarrollo se basa en la reutilización de información a distintos niveles y en la integración en el mismo sistema de diferentes tecnologías estándares para obtener las funcionalidades de ayuda. Las tecnologías que integramos en nuestros sistemas son: la recuperación de información, el modelado de usuario, el hipertexto y la representación explícita del conocimiento. Para verificar la viabilidad de nuestro modelo, hemos construido dos sistemas, Argos y Aran. Ambos sistemas se han desarrollado para un entorno informático concreto, el sistema operativo Unix. Se ha escogido este sistema porque para él se dispone de una amplia documentación y porque sobre él otros grupos de investigación han realizado diversos sistemas de ayuda, de modo que hay información diversa que es posible reutilizar. No obstante, el enfoque propuesto en nuestro modelo, y ejemplificado en los sistemas Argos y Aran, es generalizable a otros dominios de características similares al entorno Unix.

El primer sistema, Argos, implementa parcialmente nuestro modelo, incorporando un amplio conjunto de funcionalidades que permiten proporcionar ayuda al usuario del sistema operativo Unix. Argos ayuda al usuario a encontrar cuál es la información relevante a su necesidad en cada momento, a partir de la extensa documentación existente en el entorno. En él juegan un papel central los aspectos de interacción hombre-computadora para lograr un asistente eficaz y de fácil manejo. El comportamiento adaptativo se consigue mediante el modelado de usuario. La interacción con el sistema de ayuda se simplifica mediante el uso de una interfaz gráfica con capacidades de hipertexto y mediante la aplicación de técnicas de recuperación de información, que hacen posible que el usuario exprese sus peticiones en lenguaje natural.

El sistema Aran ejemplifica completamente nuestra propuesta de modelo de sistemas de ayuda inteligente. Supone una continuación del trabajo desarrollado en Argos pero incluye una representación explícita del conocimiento para mejorar la efectividad de la ayuda. El núcleo central de Aran es su base de conocimiento. En ésta se representan fundamentalmente los objetos del dominio y las acciones sobre esos objetos. El contenido de la base de conocimiento constituye un modelo del dominio que se utiliza con dos propósitos diferentes. Por una parte, se trata de disponer de un modelo de la aplicación (el sistema operativo Unix en este caso) que el usuario pueda inspeccionar y que le facilite su aprendizaje. Por otra parte, este modelo se utiliza para organizar, mediante indexación, cualquier otra información disponible sobre el dominio. Esta nueva representación permite además implementar un modelo de usuario dinámico más detallado, teniendo en cuenta sus características (p.e. interés o nivel de experiencia) en relación con las diferentes áreas del sistema, así como su evolución a lo largo de una sesión con Aran.

La memoria que presentamos está organizada en siete capítulos, comenzando con una revisión del estado del arte para seguir con el desarrollo detallado de nuestra investigación.

En el capítulo 2 se presenta una visión general de los diversos modos de utilización de las computadoras en el mundo educativo. Se hace una breve introducción histórica de la enseñanza asistida por computadora para posteriormente dedicar una atención especial a los tutores inteligentes (ITS), detallando su estructura y fundamentos. También se tratan otros enfoques educativos del uso de las computadoras, destacando las contribuciones más relevantes. A continuación, se realiza un análisis de la problemática asociada con el uso de los ITS en la formación y se estudian las razones por las que su uso no se ha generalizado. Finalmente, se proponen los sistemas de ayuda como una aproximación más sencilla y realista a un sistema de enseñanza basado en computadora.

En el capítulo 3 se estudian los sistemas de ayuda y los aspectos relacionados con la asistencia al usuario para facilitar el uso y la comprensión de las aplicaciones informáticas. Se justifica la necesidad de los sistemas de ayuda y se analizan diversos enfoques que tratan de simplificar el uso del software. A continuación, se presentan los sistemas de ayuda, las diferentes formas de entender esa ayuda y qué se entiende por ayuda inteligente. Se realiza una clasificación de los posibles tipos de sistemas de ayuda según las opciones de diseño consideradas. También se

expone el modelo conceptual del uso de la ayuda por parte del usuario y los diferentes criterios que hay que considerar para la evaluación de un asistente. Posteriormente, se analizan tres ejemplos de sistemas de ayuda que hemos considerado significativos para nuestro trabajo. Finalmente, se estudian tres proyectos de bases de conocimiento que pueden contribuir a simplificar el proceso de reutilización de información y de construcción de asistentes inteligentes.

En el capítulo 4 se propone un modelo de construcción de sistemas de ayuda inteligente para aplicaciones o entornos software de amplio espectro. El modelo tiene como objetivo obtener sistemas de ayuda eficaces, aplicables a entornos reales y que tengan un coste razonable de desarrollo y mantenimiento. Se presentan los fundamentos en los que se basa la implementación del modelo: la integración de diversas técnicas estándares en un mismo sistema y la reutilización de información. A continuación, se analizan cada una de las técnicas que se integran para proporcionar la ayuda: la recuperación de información, el hipertexto, el modelado de usuario y la representación explícita del conocimiento. Finalmente, se estudian proyectos de bases de conocimiento que contribuirán a simplificar el desarrollo de sistemas basados en conocimiento.

En el capítulo 5 se analizan las características del dominio elegido para la ejemplificación de nuestro modelo y se presenta el sistema Argos. Se exponen el diseño y la construcción del sistema Argos, detallando sus objetivos, funcionalidades y arquitectura. Se describen los diversos módulos que componen el sistema -el módulo de recuperación de información, el modelo del usuario y la interfaz- proporcionándose detalles sobre su implementación.

En el capítulo 6 se presenta el sistema Aran. Este sistema materializa completamente nuestro modelo, ya que además de las técnicas integradas en Argos, incluye también una representación explícita del conocimiento. Comenzamos tratando las características de la documentación para que la ayuda sea efectiva, destacando el problema de su insuficiente estructuración y dificultad de acceso. A continuación se detalla la arquitectura del sistema, cómo se integran las diversas tecnologías para ofrecer las funcionalidades de ayuda, a la vez que se reutiliza información y se usa un sistema estándar de representación del conocimiento para reducir el esfuerzo de desarrollo. Se describen con detalle las distintas partes del sistema, haciendo hincapié en el modelo del usuario y en el modelo del dominio.

En el capítulo 7 resumimos las principales aportaciones realizadas, enumerando las posibles líneas de continuación, tanto en un futuro inmediato como a más largo plazo.

La memoria concluye con un apéndice que contiene sesiones de ejemplo del sistema Argos y detalles de implementación del sistema Aran, así como la lista de referencias bibliográficas utilizadas.

CAPÍTULO 2

COMPUTADORAS Y EDUCACION

2.1 Introducción

Prácticamente desde el mismo momento en que aparecieron las computadoras, surgió el interés por aplicarlas en el campo educativo. A lo largo del tiempo se han propuesto distintas formas para la utilización de la computadora como un elemento más del proceso formativo. En este capítulo llevamos a cabo una revisión crítica de las propuestas más importantes de este área de aplicación de la Informática, incidiendo en sus objetivos y métodos, así como también en sus limitaciones. Esta revisión nos ayudará a encuadrar nuestra propuesta para el desarrollo de sistemas de ayuda inteligente dentro de un marco educativo más amplio, el de los tutores que aplican técnicas de inteligencia artificial.

Comenzaremos con una presentación, desde una perspectiva histórica, de los distintos usos de la computadora en los entornos educativos. Prestaremos una atención especial a los tutores inteligentes, estudiando su arquitectura, fundamentos de diseño y características, dado que los hemos considerado como el punto de partida para desarrollar nuestra propuesta de sistemas de ayuda inteligentes. También realizaremos un análisis de otras tendencias o líneas de aplicación de las computadoras en la enseñanza.

Finalizaremos con un análisis crítico del estado actual de la aplicación de los tutores informáticos en la enseñanza. Los resultados de este análisis nos llevarán a presentar el marco dentro del cual propondremos los sistemas de ayuda como una aproximación más simple, a la vez que factible y eficiente, de un sistema de enseñanza basado en computadora.

El campo de la educación asistida por computadora es multidisciplinar. Además de en las teorías educativas, este tipo de aplicaciones llevan sus raíces a los campos de la inteligencia artificial, la psicología, las ciencias cognitivas, la lingüística computacional y la interacción hombre-máquina. Sin ánimo de restar importancia a los demás fundamentos y debido a que resulta ser lo más relevante para nuestra propuesta, profundizaremos en la vertiente informática y técnica de los sistemas

analizados, y no abundaremos en el estudio de las teorías psicológicas y cognitivas que subyacen a los sistemas.

2.2 Una perspectiva histórica

En los últimos años se ha publicado un gran número de trabajos en los que se analiza el impacto que ha producido en el mundo educativo, desde los años 60, la introducción de la computadora. Entre ellos se pueden destacar [O'Shea 85], [Bork 86] y [Bork 91]. Otros trabajos, tanto libros como recopilaciones de artículos, que tratan sobre la aplicación de las computadoras en la enseñanza con una visión particular desde el campo de la inteligencia artificial son [Sleeman 82, Wenger 87, Kearsley 87a, Kearsley 87b, Lawler 87, Self 88a, Ercoli 88, Clancey 90, Kopec 92, Costa 92].

Aunque las diversas aplicaciones de las computadoras en la educación se pueden presentar y clasificar de muy variadas formas, nosotros aquí vamos a tratar de respetar en lo posible la secuencia histórica en su presentación y para clasificarlas pondremos especial énfasis en las técnicas utilizadas. Este estudio no pretende ser exhaustivo, sino más bien proporcionar una muestra de los distintos intentos de utilización de la computadora como un elemento instructivo más. Otras revisiones con esta misma línea se pueden encontrar en [Yazdani 87, Sokolnicki 91]. Otras clasificaciones con enfoques diferentes, que hacen más hincapié en los paradigmas cognitivos sobre los que se sustentan las aplicaciones se pueden encontrar en [Cabrera 95, Bork 86, Vaquero 87].

2.2.1 La Enseñanza Asistida por Computadora (CAI)

La Enseñanza Asistida por Computadora, o CAI (por *Computer Assisted Instruction*), se puede considerar que surge a finales de los años 50 con los "programas lineales" para la instrucción programada. Estas aplicaciones estaban claramente influidas por la teoría psicológica conductista, cuyo principal exponente era Skinner. Según esta metodología, el material a enseñar debe organizarse de modo que se maximice el número de respuestas correctas del alumno, debido a que las respuestas incorrectas producen un reforzamiento negativo del alumno hacia el aprendizaje. El conocimiento a transmitir se encuentra organizado en bloques de texto (*frames*) con un orden de presentación fijo. Algunos de esos bloques pueden ser preguntas cuya respuesta analiza el sistema e informa al estudiante de si son correctas o no; sin embargo, las respuestas no se tienen en cuenta para modificar el comportamiento del programa, que sigue con su secuencia de instrucción prefijada.

En los años 60 se empieza a considerar la utilización de las respuestas proporcionadas por el estudiante para controlar el material que se le muestra. Surgen así los programas de ramificación (*branching*) o de programación intrínseca. Estos programas contemplan acciones correctivas, adaptando el método de enseñanza de acuerdo con las respuestas proporcionadas por el alumno. La información sigue estando prefijada, pero se presenta o no en función de las

respuestas obtenidas. Ya se tienen en cuenta los aspectos de la realimentación al estudiante y de la individualización, aunque todavía de una forma bastante simple. Estos sistemas CAI resultaban mucho más efectivos, pero su diseño e implementación resultaban muy complejos, debido a la necesidad de prever todas las posibles interacciones [O'Shea 85].

Para intentar disminuir esa dificultad de diseño e implementación, aparecen los "lenguajes de autor", posteriormente evolucionados en los "sistemas de autor". Aunque con enfoques diferentes, esas herramientas lo que intentan es facilitar la realización del material CAI, sobre todo para los autores no expertos en el campo informático. Un ejemplo de sistema de autor en castellano es el SIETE [Vaquero 86, Hernández 90, Hernández 89] con el cual, mediante una interfaz en castellano, se pueden desarrollar lecciones sin que apenas sea necesario disponer de conocimientos de programación. Otros autores producen cursos tutoriales con un alto grado de sofisticación utilizando herramientas de programación clásicas y con la ayuda de metodologías de ingeniería del software desarrolladas para la optimización del proceso (en esta línea trabaja, por ejemplo, Alfred Bork en la Universidad de California, Irvine). Los sistemas de CAI basados en ramificación se siguen utilizando de forma comercial, viéndose continuamente reforzados con el desarrollo de nuevas tecnologías que se pueden denominar genéricamente hipermedia, (tales como el vídeo disco interactivo, el CD-ROM o el CD-I) y de nuevas y mejores interfaces gráficos [Woolf 92]. Este proceso de mejora se ve acelerado por la aparición de nuevos lenguajes y entornos de programación (p.e., HyperCard [Mitchel 91]) que integran todas esas nuevas tecnologías, facilitando y reduciendo el coste del desarrollo de las aplicaciones [Montero 93].

En los años 70 aparecen los sistemas "generativos", que toman su nombre del hecho de que es la propia computadora la que genera automáticamente parte del material de enseñanza que se le presenta al estudiante (el término generativo se utiliza aquí con un sentido opuesto al utilizado en [Kass 89]). Estos sistemas surgen con el deseo de facilitar la tarea del autor en la preparación del material de enseñanza y también de una filosofía educativa diferente, que sostiene que en ciertas situaciones los estudiantes aprenden mejor enfrentándose a problemas de una dificultad apropiada que atendiendo a explicaciones sistemáticas [O'Shea 85]. A partir de una estrategia de enseñanza determinada, el sistema es capaz de generar el árbol con las posibles interacciones. Las ventajas potenciales de estos sistemas los hacían muy prometedores, ya que no resultaría necesario anticipar las interacciones con los estudiantes y los ejercicios se adaptarían a las necesidades y el nivel de conocimientos del alumno. Las posibilidades adaptativas de estos sistemas se basaban en los algoritmos de selección de tareas y en medidas simples de la dificultad de los ejercicios y del rendimiento del alumno. Sin embargo, esta posibilidad generativa se quedó limitada a dominios muy concretos, tales como la aritmética o el vocabulario, y a estrategias de ejercicios (*drill and practice*). Para tales dominios simples esta técnica resultaba suficientemente robusta, habiéndose construido sistemas instruccionales que se han utilizado durante más de una década (p.e., los sistemas de Stanford y Leeds) [Sleeman 82].

Los programas tradicionales de CAI son unos contenedores estructurados, organizados estáticamente para albergar el conocimiento pedagógico y del dominio,

de un profesor experimentado. De la misma forma, la tarea de creación de material instructivo desde el enfoque CAI puede llegar a ser un proceso extremadamente sofisticado de incorporación en un programa de las decisiones pedagógicas de un profesor. No obstante, resulta necesario anticipar todas las circunstancias en las cuales hay que llevar a cabo alguna acción, para así poder escribir el código apropiado que tenga en cuenta esas decisiones. La potencia del enfoque del CAI tradicional reside en aprovechar la experiencia pedagógica de los profesores humanos y plasmarla directamente, mediante una adecuada codificación, en el comportamiento de los programas sin una articulación expresa de los principios en los que se basa [Kass 89].

2.3 Los tutores inteligentes (ITS)

A partir del año 1970 los investigadores comienzan a aplicar técnicas de inteligencia artificial (IA) en el desarrollo de entornos de aprendizaje. En ese mismo año aparece el artículo de Carbonell "AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction" [Carbonell 70], que se puede considerar el punto de partida de los tutores inteligentes, denominados genéricamente tanto ICAI (Intelligent CAI) como ITS (término propuesto en [Sleeman 82]). En el artículo se plantea el abandono de los sistemas CAI *"ad hoc-frame-oriented"* y su sustitución por otros basados en la estructura de la información, en los que se trate de representar el conocimiento y proporcionar unos procesos que lo manipulen automáticamente, como lo hace el propio ser humano. Las principales limitaciones que identifica en los CAI son: el estudiante no tiene iniciativa o ésta es muy limitada; no se puede utilizar el lenguaje natural en las respuestas; son demasiado rígidos, carentes de iniciativa propia (ya que su comportamiento está preprogramado) y no poseen un "conocimiento" real. Como alternativa a esas limitaciones, Carbonell presenta SCHOLAR, su tutor sobre geografía de América del Sur.

En general, el propósito de la investigación en sistemas instruccionales con aplicación de técnicas de inteligencia artificial es capturar el conocimiento que permite a los expertos crear una interacción instructiva. En lugar de centrarse en las decisiones obtenidas de la transmisión de un determinado conocimiento, es el propio conocimiento el que se representa explícitamente, en una forma que pueda ser utilizada por el sistema informático. Así, es responsabilidad de los propios programas el componer las interacciones instruccionales dinámicamente, tomando decisiones en base al conocimiento que se les ha proporcionado. El objetivo sigue siendo el mismo que el del CAI, que el estudiante adquiera los conocimientos y habilidades necesarias para realizar de forma satisfactoria las tareas del dominio o materia que trata el tutor.

No hay una división clara entre los tutores tradicionales y los que aplican técnicas de IA. Los CAI más sofisticados tienen algunas capacidades autónomas, como por ejemplo la generación de ejercicios o la adaptación del nivel de dificultad en base a alguna medida del rendimiento del estudiante. Por otra parte, los modelos de experiencia creados en los ITS varían mucho en su generalidad y en el grado en el cual se hace explícito el conocimiento subyacente a las decisiones específicas. No se

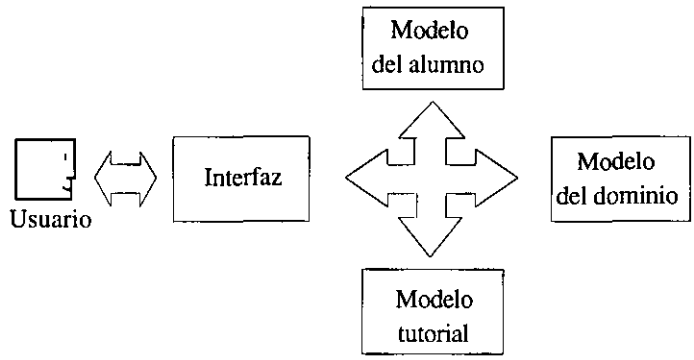


Figura 2.1: Estructura de módulos de un tutor inteligente

ha conseguido el objetivo ideal de crear un sistema capaz de razonar pedagógicamente de una forma completamente autónoma, pero la transferencia de la experiencia de los expertos en lugar de plasmar sus decisiones hace que sea posible que los sistemas se comporten de formas no anticipadas por los expertos. Esto ha hecho que los tutores inteligentes se hayan convertido en un campo de investigación experimental para las teorías de la psicología cognitiva [Lewis 87]. En todo caso, la diferenciación entre ITS y CAI no se basa en el rendimiento de los sistemas conseguidos. La construcción de tutores inteligentes sigue siendo todavía algo pretecnológico, sin que exista una metodología clara y contrastada para su desarrollo, algo que, sin embargo, sí existe para la creación de sistemas CAI, lo que por el momento permite conseguir rendimientos superiores [Dear 87, Breuker 90].

Se han desarrollado tutores inteligentes para materias muy diversas, pero a grandes rasgos y sin entrar en sus particularidades todos se ajustan a una arquitectura compuesta por los cuatro módulos que se muestran en la ilustración (Figura 2.1). Sin pretender ser exhaustivos, vamos a citar algunos ejemplos en base a sus dominios de aplicación: la enseñanza de la programación y el análisis de programas —Lisp Tutor [Anderson 90], PROUST [Jonhson 86], Bridge [Bonar 88], CAPRA [Fernández-Castro 92, Verdejo 92] —, el diagnóstico médico —Guidon [Clancey 82, Clancey 87], FysioDisc [Winkels 92a] —, el campo de las matemáticas —Algebra Tutor [Anderson 90], LMS [Sleeman 82], MACSYMA Advisor [Genesereth 82] —, la geografía —SCHOLAR [Carbonel 70] — o el entrenamiento industrial —RBT [Woolf 87b], STEAMER [Hollan 87], FORMIP [Díaz 92], INTZA [Gutierrez Serrano 94] —. Otras revisiones y clasificaciones de tutores se pueden encontrar en [Boulay 88, Yazdani 87, Sokolnicki 91, Rolston 88, Wenger 87].

2.3.1 Estructura de un tutor inteligente

Aunque se han desarrollado gran cantidad de sistemas de CAI inteligente o ITS, con enfoques muy diversos, existe un amplio consenso en la bibliografía que establece que la arquitectura mínima de estos sistemas estaría constituida por los siguientes bloques o módulos [Kearsley 87a, Clancey 90]:

- Modelo del dominio o del experto: un modelo explícito del dominio o tema a enseñar.

- Modelo del alumno: modelo que identifique qué es lo que entiende el estudiante.

- Modelo tutorial: modelo que proporcione la estrategia para la enseñanza.

Cada vez aparece más frecuentemente como otro módulo relevante la interfaz con el alumno, debido a que es el encargado de conducir la comunicación con el estudiante. De hecho, su importancia es cada vez mayor y puede llegar a determinar una gran parte del diseño de un ITS [Wenger 89, Lesgold 87, Dubs 91, Kass 89].

- Modelo de interfaz: modelo que proporciona la comunicación con el estudiante.

Otros autores distinguen como bloque autónomo el dispositivo de simulación o el entorno instructivo utilizado [Pitts 92, Gutierrez Serrano 94].

También hay opiniones discordantes respecto a esta organización, como [Breuker 90, Winkels 92b]. No sólo consideran que esta división es artificial, sino también que debería hacerse una descomposición funcional, debido a que hay funciones en las cuales los módulos se solapan.

2.3.1.1 El modelo del dominio o del experto

Este módulo contiene el conocimiento del sistema sobre lo que se enseña (dominio). Normalmente, representa el conocimiento que un experto o un estudiante ideal debería tener. Incorpora dos funcionalidades fundamentales [Wenger, 87]: primero, actuar como fuente del conocimiento que se va a presentar al alumno, sirviendo también para la generación de explicaciones, preguntas, respuestas y tareas; segundo, ser una referencia y patrón de comparación para la evaluación del comportamiento del estudiante. Por esto, la representación no es tan sólo una descripción de los conceptos y habilidades que debe adquirir el estudiante, como en un curriculum, sino que debería servir para generar soluciones en el mismo contexto que el alumno de forma que se puedan valorar sus respuestas.

En un tutor se pueden incluir diferentes tipos de conocimiento. Según [Woolf 87a] estos tipos son: conocimiento *declarativo*, constituido por los conceptos de un dominio y sus interrelaciones, siendo normalmente la fuente primaria de información sobre un dominio; conocimiento *procedimental*, constituido por los procesos mediante los cuales se utilizan los datos para la resolución de problemas; conocimiento *heurístico*, que describe las acciones de un experto en ese dominio; y conocimiento de *simulación*, necesario para plasmar, normalmente de forma gráfica, el sistema al cual se aplica el tutor. La presencia o ausencia de estos tipos de información condiciona las capacidades y las posibles aplicaciones del ITS.

Un punto clave en el desarrollo de un ITS es el tipo de representación que se utilice para el conocimiento sobre el dominio, ya que va a determinar en gran medida el tipo de información utilizada en otras partes del sistema. Por ejemplo,

influenciará el contenido de la interacción tutorial, la estructura de objetivos que gobierna la selección de ejemplos, preguntas y explicaciones, así como el tipo de errores de concepto que se pueden representar. En la literatura sobre el tema [Lawler 87, Kearsley 87a, Kramer 87, Clancey 90, Frasson 92] se encuentran una gran cantidad de representaciones posibles, como por ejemplo los sistemas de producción, las arquitecturas basadas en esquemas, las representaciones procedimentales o las representaciones basadas en la lógica. Aunque se utilizan incluso terminologías más específicas, se puede decir que en la práctica totalidad de los casos, las técnicas informáticas utilizadas para este modelado se pueden clasificar dentro de dos grandes grupos [Fernández-Valmayor 90]: las basadas en los modelos de reglas o producciones y las basadas en modelos asociativos (p.e. redes semánticas y jerarquías de frames) [Barr 81]. Actualmente, también existen aproximaciones mixtas en las cuales se tratan de combinar las ventajas de los dos tipos de representación, siendo cada vez más difícil establecer una frontera clara.

La planificación es otro tema importante que está muy relacionado con la representación del conocimiento, ya que todo el proceso de enseñanza se puede considerar como la generación, ejecución y revisión de un plan para la obtención de ese objetivo. La búsqueda de modelos ejecutables ha llevado a la codificación de parte del conocimiento que se necesita en un tutor en forma de planes [Verdejo 92]. Esto significa disponer de medios para representar el problema en términos de lo que hay que aprender, qué conceptos se involucran y los posibles errores que puede llevar asociados; por tanto, también influye en el proceso de diagnosis [Sokoknicki 91]. Un ejemplo de sistema que utiliza planes es PROUST [Johnson 86]; para analizar cuáles son las intenciones de un estudiante al programar un determinado código, se tiene una serie de planes indexados en base a los objetivos que se alcanzan con ellos. Otros sistemas que utilizan planes son Bridge [Bonar 88], MACSYMA Advisor [Genesereth 82] y CAPRA [Fernández-Castro 92].

Debido al éxito y el desarrollo de los sistemas expertos para la resolución de tareas complejas, se han convertido en modelos utilizados también en el desarrollo de tutores [Merril 87]. Los sistemas expertos son muy adecuados para capturar el conocimiento procedimental y heurístico, pero el que se disponga de la información necesaria para la resolución de problemas no implica que ésta sea por sí sola suficiente para ser utilizada con fines educativos [Clancey 82, Clancey 87]. No es lo mismo solucionar un problema que explicar por qué se hace de una determinada forma y no de alguna otra, lo que dificulta en gran medida la reutilización de las bases de conocimiento existentes para la resolución de problemas.

Como ejemplos paradigmáticos de las dos formas diferenciadas de representar el conocimiento, con el modelo asociativo destaca el ya mencionado SCHOLAR [Carbonell 70] que utiliza una red semántica. Como ejemplo de proyecto basado en el modelo de producción se puede destacar GUIDON [Clancey 82], un tutor de diagnóstico médico construido sobre el sistema experto MYCIN. En [Rolston 88] se puede encontrar una clasificación de los ITS en función del tipo de formalismo de representación utilizado para el dominio.

2.3.1.2 El modelo del alumno

Este módulo contiene un modelo del conocimiento del alumno y su evolución a lo largo de la interacción con el sistema. Es una representación de lo que el sistema cree que sabe el estudiante, de lo que no sabe o de lo que sabe pero es incorrecto. Debe tener una parte dinámica que al actualizarse represente el cambio en el estudiante y refleje su proceso de aprendizaje [Dubs 91].

Idealmente, este módulo debería incluir todos los aspectos del conocimiento y comportamientos del estudiante que tengan repercusiones en su rendimiento y aprendizaje. El proceso de formación y mantenimiento de este modelo se hace a partir del análisis de la interacción con el estudiante. De esta forma se captura dentro del modelo qué es lo que conoce el alumno o qué es lo que ha entendido mal. La elección del modelo del alumno afecta de forma determinante a los procesos de diagnosis que se pueden realizar a partir de él. La adaptabilidad de un sistema instructivo está muy determinada por la extensión y precisión del modelo del alumno, que es el que permite tomar decisiones de forma individualizada. Por esto, hay investigadores que afirman que el modelado dinámico del estudiante es la contribución fundamental que diferencia a los ITS de los tutores clásicos [Self 94, McCalla 92].

Los usos que se hacen del modelo del alumno son muy variados. Self ha encontrando más de 20 aplicaciones diferentes y los ha catalogado, distinguiendo seis categorías: correctiva, elaborativa, estratégica, de diagnóstico, predictiva y evaluativa [Self 88b]. Entre los usos más habituales cabe destacar: el control de la presentación de información, evaluándose el nivel de conocimiento sobre un determinado concepto para, una vez que se considera asimilado, pasar a otro relacionado; la presentación de consejos o pistas no solicitados cuando el sistema cree que son necesarios; la generación de problemas o tareas adecuadas al nivel de conocimiento actual; y la adaptación de las explicaciones instructivas en función de términos o materias que el alumno ya conoce.

Existen muchas formas de representar el modelo del alumno [Kass 89]. Dado que el sistema debe tener necesariamente conocimiento sobre el dominio, en la mayoría de los casos el modelo del alumno sigue la misma filosofía que la utilizada para representar al experto. Entre los enfoques que más ampliamente se han utilizado destaca el modelo de *subconjunto* u "overlay", en el que la representación se realiza como una parte del conocimiento representado en el modelo del dominio, de forma que no se puede representar información incorrecta. Otros enfoques son: el modelo *diferencial*, que es una variación del modelo de subconjunto en el cual se divide el conocimiento del estudiante en dos categorías, el conocimiento que el estudiante debe conocer y el conocimiento que no se puede esperar que conozca; el modelo de *perturbaciones*, modelo del estudiante similar al del experto pero con pequeñas variaciones o perturbaciones en algunos conocimientos que probablemente sean incorrectos; los modelos de errores, en los que se trata no sólo de representar el conocimiento correcto sino también el conocimiento incorrecto del estudiante; y los modelos de proceso o simulación, en los que se simula el proceso por el cual el estudiante resuelve el problema [Dubs 91].

Aunque no es estrictamente necesario disponer de un modelo del alumno que sea perfecto y completo para poder tomar decisiones pedagógicas razonables, el disponer incluso de un modelo parcial resulta un objetivo difícil y costoso de conseguir. Este alto coste hace que algunos sistemas se limiten a un modelo del alumno muy simple, que básicamente consiste en un historial de la interacción y alguna medida de rendimiento. Incluso hay autores que consideran que el desarrollo de este modelo no es siempre positivo para un tutor, ya que se dedica demasiado esfuerzo a su creación y mantenimiento, lo que hace que se descuiden otros objetivos más importantes como la efectividad de la enseñanza [Feifer 91].

2.3.1.3 El modelo del tutor

El modelo del tutor (o modelo pedagógico) representa el conocimiento didáctico del sistema, incluyendo las estrategias instructivas. Es el responsable de determinar qué información presentar al alumno, cuándo y de qué forma. Teniendo en cuenta los datos existentes en el modelo del alumno, la representación del dominio, los principios pedagógicos y los objetivos instructivos que se tienen, este módulo debe determinar cuál es la camino a seguir para llevar a cabo la enseñanza. Es decir, debe determinar qué conceptos hay que enseñar y en qué orden, la forma de presentación de una determinada materia o el momento apropiado para interaccionar con el estudiante.

El modelo pedagógico puede implementar una de las estrategias de enseñanza descritas en la pedagogía o integrar varias de ellas [Fernández-Valmayor 93]. Aquí, el problema reside en que el proceso de aprendizaje todavía no se comprende totalmente, por lo que existen una serie de teorías incompletas sobre el aprendizaje que, además, llegan a ser contradictorias entre sí. Entre las utilizadas para el desarrollo de curricula cabe destacar la psicología conductista, la psicología cognitiva y la psicología evolutiva [Bork 86]. Una recopilación de las teorías sobre la enseñanza y el aprendizaje se puede encontrar en [Kearsley 96].

Dependiendo de a quién corresponda la iniciativa y del grado de control que se ejerza sobre la comunicación, se pueden distinguir distintos tipos de interacción tutorial. En un extremo tenemos el modelo de entrenador (*coaching*) o de descubrimiento guiado, en el cual el estudiante tiene la mayor parte del control de la actividad, determinando qué partes quiere practicar o acceder. Cuando se detecta un error o un problema, el sistema proporciona consejo sobre la medida a tomar, de forma que el estudiante pueda seguir adelante. En un punto medio se encuentran los modelos de iniciativa mixta, como los diálogos socráticos, en los cuales el control del proceso educativo está compartido entre el tutor y el alumno. A veces, es el alumno el que realiza las preguntas; otras veces, es el tutor el que las plantea para averiguar qué es lo que el estudiante trata de hacer o qué es lo que conoce. En el otro extremo se encuentran los sistemas monitores, en los que el tutor posee el control absoluto del proceso de enseñanza y el alumno se tiene en cuenta sólo para llevar a cabo una adaptación de acuerdo con su rendimiento.

2.3.1.4 Interfaz

Este es el módulo que define la forma en que se realiza la interacción entre el tutor y el alumno. Es el componente que traduce la representación interna del sistema, cualquiera que ésta sea, a un lenguaje comprensible por el usuario. Este módulo es fundamental para la efectividad del ITS [Lewis 87]. Dependiendo de la forma en que el sistema presente un tema, éste puede ser más o menos comprensible. Además, el usuario se crea una imagen del sistema en función de su interacción con él, por lo que las cualidades como la facilidad de uso o el que resulte atractivo, llegan a ser críticas para su aceptación. De hecho, la introducción generalizada de la tecnología multimedia proporciona herramientas cuyo poder comunicativo es cada vez mayor, influyendo cada vez más en el diseño de todo el sistema.

No sólo es necesario que un tutor comunique su conocimiento, sino también que lo haga de una forma versátil, siendo deseable que pueda comunicar una misma materia de formas diferentes, dependiendo de a quién se dirija y en qué situación lo haga. La importancia de la forma de comunicación ha hecho que haya sistemas que incluyan como interfaz todo un entorno de aprendizaje normalmente basado en simulaciones. STEAMER [Hollan 87] es un sistema pionero en el desarrollo de una potente interfaz gráfica. Se trata de una herramienta para entrenar a ingenieros que van a trabajar en grandes barcos. Proporciona una simulación interactiva e inspeccionable que, mediante una simulación numérica, desarrolla un modelo mental del sistema de propulsión de un barco. El alumno puede observar el sistema con diferentes niveles de detalle y desde diferentes puntos de vista. Otro sistema que también hace hincapié en la interfaz es Recovery Boiler Tutor [Woolf 87b], un tutor interactivo que realiza el entrenamiento de operadores de una caldera de recuperación de una factoría de papel. Proporciona un entorno de aprendizaje reactivo e inspeccionable, de forma que el alumno puede experimentar e interactuar visualizando los distintos contadores e informes.

2.4 Otros enfoques educativos

A continuación vamos a presentar otros enfoques recientes de la utilización de las computadoras en el campo educativo. Las fronteras que dividen los distintos enfoques no están claramente definidas y, por tanto, la clasificación es básicamente orientativa. Algunas propuestas son radicalmente diferentes a los tutores inteligentes, por el objetivo perseguido o por las ideas del diseño subyacente, como la enseñanza basada en casos o los micromundos. Otras propuestas, sin embargo, se podrían considerar evoluciones de los ITS, como los entornos interactivos de aprendizaje o la enseñanza mediante descubrimiento guiado.

Nuestro objetivo aquí es proporcionar una visión más amplia con otras ideas que se están aplicando en el campo educativo, ideas que son significativas para encuadrar nuestro trabajo. En la exposición, intentamos destacar algunos aspectos diferenciadores respecto de la propuesta de los ITS más clásicos. En general, en estos enfoques educativos se tiende hacia sistemas que dan una gran importancia a la interfaz de usuario, permiten un mayor control por parte del alumno y tratan de

evitar algunos de los procesos más costosos del desarrollo de los ITS (p.e., la diagnosis o el modelado del alumno) [Kommers 92].

2.4.1 Simulaciones y micromundos

Existen algunos planteamientos radicalmente diferentes sobre la aplicación de las computadoras en la educación que se basan en las teorías de Piaget y en la psicología cognitiva, como la propuesta por Seymour Papert [Papert 80]. El aprendizaje se plantea como una responsabilidad del alumno. La responsabilidad del profesor no se encuentra en la organización y la transmisión del conocimiento, sino que, por el contrario, se considera una educación abierta en la cual su responsabilidad es la de ayudar al estudiante a convertirse en un mejor alumno. No se establece un contenido específico de conocimientos que el estudiante ha de dominar al final de un cierto periodo de formación y, por tanto, no se trata de optimizar la forma en la que el estudiante alcanza el objetivo.

La computadora no es un agente en la formación, sino una herramienta que debe potenciar el desarrollo de las habilidades cognitivas del estudiante, mejorando los procesos cognitivos y modificando los patrones de acceso al conocimiento. No se trata de proporcionar al alumno un conocimiento explícito sobre el proceso y las estrategias de aprendizaje, lo que lleva sin duda al desarrollo de aplicaciones más estructuradas y dirigidas. Por el contrario, el objetivo aquí es la creación de un entorno de aprendizaje en el cual el alumno pueda enfrentarse con los problemas esenciales, encontrando mediante la exploración y la experimentación una forma particular de resolverlos. Se prima la iniciativa del estudiante para que explore y así aprenda, pasando el profesor a ser un asesor de ese aprendizaje, ayudando a resolver las dudas presentadas [Papert 87a]. La visión de la computadora que enseña al alumno cambia por la del alumno que controla completamente a la computadora, siendo capaz de "programarla" para de esa forma adquirir conceptos científicos o técnicos. La idea es que se pueden diseñar sistemas informáticos que permitan una comunicación más natural en el proceso de aprendizaje y que esta comunicación puede modificar de forma positiva la forma en que se realizan otros tipos de aprendizaje.

Siguiendo este enfoque se desarrolló LOGO, un lenguaje de programación con una sintaxis sencilla por medio de la cual el estudiante controla la posición de un puntero (tortuga) en una ventana gráfica. Lo que se propone es que con la creación de simulaciones o micromundos [Paper 87b] se estimula el aprendizaje y se consigue una actitud más positiva por parte del alumno. Por ejemplo, resulta más fácil aprender el concepto de ángulo jugando con la tortuga de LOGO que proporcionando su definición abstracta para posteriormente practicar en un entorno completamente dirigido.

2.4.2 La Enseñanza Basada en Casos (Case-Based Teaching)

Otra forma de utilización de las computadoras en la educación es la Enseñanza Basada en Casos o CBT (Case-Based Teaching), desarrollada por Roger Schank en el Institute for the Learning Sciences de la Northwester University [Schank 94]. Se

trata de un planteamiento educativo que surge a raíz de la teoría del razonamiento basado en casos [Riesbeck 89]. Se fundamenta en dos suposiciones: la primera es que el razonamiento humano consiste en el acceso y la combinación de casos o historias de situaciones previamente experimentadas; la segunda es que la mejor forma de enseñar es establecer situaciones reales para que el estudiante construya adecuadamente esos casos. Como se ha observado que las personas utilizan este tipo de razonamiento para situaciones muy diversas, se desprende que una forma de apoyar este tipo de razonamiento es proporcionando al estudiante los casos concretos.

Las personas aprenden bien cuando se encuentran inmersas en una tarea que les resulta interesante y que es un desafío para ellos. Como no se puede presuponer que un alumno este motivado *a priori* a usar el sistema, en el CBT se involucra al estudiante en una tarea que le proporcionará unas posibilidades muy ricas para aprender. El sistema aprovecha estas oportunidades para presentar historias que ayuden al alumno a aprender de esa situación. De esta forma, se aprende mediante la interacción con la tarea y mediante las historias que se encuentran como resultado de las intervenciones. El proceso de interpretación e integración de la nueva información es más sencillo, ya que los ejemplos se muestra dentro de un contexto que los hace más comprensibles. Idealmente, el sistema sólo presenta una historia al estudiante cuando es relevante para la situación actual, lo que permite una mejor asimilación de la información. En los CBT se prima la iniciativa de interacción del estudiante y por tanto se le da una gran importancia al desarrollo de la interfaz del sistema.

Hay dos factores importantes para el desarrollo de sistemas de este tipo. Uno es la forma en la que se indexan los casos concretos y el vocabulario concreto que se utiliza para indexarlos. El segundo es qué tipo de estrategias hay que utilizar para recuperar esas historias concretas. Una ventaja importante de esta arquitectura de sistema es que se pueden presentar los casos adecuados al estudiante sin que el sistema tenga que comprender completamente su contenido. De la misma forma que un razonador basado en casos es capaz de presentar historias sin entenderlas completamente, un sistema CBT puede presentar historias sin comprenderlas de la forma que lo hará el estudiante. Esto es debido a que los casos se indexan de acuerdo a en qué situación son útiles y no únicamente por su contenido.

> *"Traditionally intelligent tutoring systems (ITS's) have needed to understand all of the information that they want to impart to the student. This complete knowledge enables systems to identify "buggy" behavior, missing information or misconceptions on the part of the student. (...) A case-based teaching system sacrifices some of this ability in favor of ease of scale-up and a more expressive knowledge communication strategy." [Edelson 92]*

En los CBT no se incluye un modelo explícito del estudiante. Las consideraciones respecto al alumno se tienen en cuenta en el diseño del sistema, previendo de antemano cuáles son las interacciones o preguntas más probables, cuál es la estructura de la tarea y cuál es el objetivo del estudiante. Incluso hay autores que creen que este modelado es negativo por ser un proceso muy gravoso que no

garantiza la obtención de una buena enseñanza [Feifer 91] (este mismo autor ha desarrollado ITS con modelo de alumno [Feifer 92]).

Los sistemas CBT no se basan en algoritmos o técnicas de inteligencia artificial muy sofisticadas, sino que su funcionalidad se basa en las historias o casos contenidos y en la red que las indexa facilitando el acceso al alumno. Se trata de evitar parte de los aspectos más costosos de la construcción de los ITS, como la representación del alumno, del dominio o el proceso de diagnosis. Por ejemplo, cuando el estudiante falla, no se intenta determinar las causas, sino que se muestra un ejemplo que ayuda al propio estudiante a determinar cuál ha sido la razón de su fallo. No es que se descarten esos aspectos, sino que se tienen en cuenta para la organización y acceso a las historias o casos. Esto implica que la red de indexación de casos tiene que considerar en su diseño cuál es la estructura del proceso cognitivo (p.e., planificación, explicación, diagnosis), cuál es la naturaleza de la historias que se indexan y cuáles son los objetivos del estudiante.

Los tutores basados en casos se han utilizado en entornos muy diversos. Un ejemplo son los sistemas ASK [Ferguson 91] que se han aplicado para la enseñanza de materias tan diferentes como la consultoría bancaria (ASK Tom), la ayuda a las decisiones presidenciales (ASK The President) y las razones del éxito de las industrias de una determinada nación en el mercado global (ASK Michael). Otros ejemplos son el sistema CreANIMate [Edelson 92] para la enseñanza de zoología en la escuela elemental o el sistema VICTOR [Feifer 91] para el entrenamiento de los operadores de teléfonos.

2.4.3 Los Entornos Interactivos de Aprendizaje (ILE)

La dificultad práctica del desarrollo de modelos de alumno fiables, la evolución desde el punto de vista de la comunicación de información de los ITS hacia nuevas filosofías más constructivistas y el desarrollo de nuevos medios para dar soporte a las interacciones educativas, han provocado la aparición de otros enfoques como los Entornos Interactivos de Aprendizaje o ILE (Interactive Learning Environment) [Self 94, Self 90b]. Estas nuevas ideas están influyendo en los propios desarrolladores de ITS, que a menudo se utilizan más como entornos para la ejercitación de conocimientos ya adquiridos que para impartir contenidos completamente nuevos.

La evolución hacia los ILE tiende a enfatizar las siguientes características, ninguna de las cuales es intrínsecamente opuesta al diseño de ITS: la autonomía del estudiante, tendiéndose a proporcionar más libertad al estudiante para que siga su propia agenda de aprendizaje de la que se da en un ITS basado en curriculum; las concepciones alternativas, reconociéndose que un estudiante puede (o incluso debe) tener una perspectiva diferente del conocimiento del dominio, perspectiva que no tiene por qué ser necesariamente incorrecta; el conocimiento sobre el dominio ya no tiene un papel tan fundamental porque se busca más el desarrollo de habilidades que la transferencia de ese conocimiento; la existencia de varios estilos de enseñanza que incluyan también la colaboración y el diálogo con el estudiante; el uso de multimedia, buscándose conseguir una interacción más rica con el usuario;

los ILE pretenden que el aprendizaje sea más realista, planteando situaciones próximas al contexto en el cual se van a utilizar los conocimientos.

El término "entorno de aprendizaje" se ha utilizado tanto en el campo de los ITS [Clancey 90] como en el de los micromundos [Kommers 92], pero aquí se usa con un sentido más amplio, propuesto por Self. Este enfoque ha sido concretado y demostrado con diferentes prototipos: desde SAFE, un sistema basado en simulación, o EPIC, un sistema basado en planificación, hasta PEOPLE-POWER, que utiliza técnicas de aprendizaje máquina y adopta el papel de un colaborador en el aprendizaje con el alumno [Self 94].

2.4.4 Los Tutores de Descubrimiento Guiado (GDT)

La enseñanza mediante descubrimiento guiado (Guided Discovery Tutoring o GDT) [Elsom-Cook 88] es otro enfoque para la construcción de tutores inteligentes. Consiste en una mezcla de la concepción mas directiva de los CAI e ITS con la de los entornos de aprendizaje en los que los estudiantes pueden explorar y aprender de una forma más libre [Elsom-Cook 90]. Proponen una supervisión "benigna" del tutor, ya que creen que el control realizado en los ITS puede ser excesivamente rígido para muchos estudiantes.

La interacción tutorial impone restricciones al estudiante y, por tanto, sería deseable que el tutor pudiera variar su estilo de enseñanza según las condiciones de cada momento, las necesidades del alumno o el propio dominio. La educación se plantea como un proceso de negociación entre dos agentes, uno de los cuales dispone de la información sobre el dominio y la forma de enseñar (el tutor) y el otro tiene un conocimiento previo y unas determinadas preferencias de aprendizaje (el alumno). No es únicamente un proceso de transferencia de información, sino también, y quizá más importante, de adaptación de esa información para que pueda ser asimilada por el estudiante. Se propone un modelo aproximado del alumno en el cual se tienen unas expectativas o límites dentro de los cuales se encuentra el alumno. Este modelo aproximado es suficiente para la toma de decisiones tutoriales que orienten al alumno, ya que no se trata de implementar un estilo tutorial muy directivo que necesite un modelo muy preciso.

> *"Tutoring styles should be thought of as existing points along a continuum of guided discovery teaching styles. Tutoring systems should aim to cover a range of adjacent styles on that continuum, and be prepared to switch between them. It is important that guided discovery learning systems possess domain-specific knowledge which can be used to ensure that the pupil does not move too far away from fruitful model of the domain. The user modeling in these systems should relate to models of the learning process, and should operate by imposing fuzzy bounds on the knowledge of the pupil rather than attempting to build an exact model." [Elsom-Cook 88]*

No debe pensarse que dando libertad al estudiante se esté promoviendo una interacción descontrolada. Los principios del GDT, como los del trabajo original de

Seymour Paper [Paper 80], sugieren que se puede confiar en el estudiante para que realice una parte de exploración no estructurada y guiarlo sutilmente en la dirección más apropiada para él. Esta desviación controlada puede tomar la forma de una secuencia tutorial completa o puede ser simplemente una forma particular de presentar una visión de un dominio [Eisenstadt 93]. Por ejemplo, IMPART [Elsom-Cook 88] es un tutor que aplica estos principios en el campo de la programación, guiando a los estudiantes en el aprendizaje de LISP. La aplicación observa cómo interacciona el alumno con el sistema LISP y proporciona consejos sobre la semántica del lenguaje cuando considera que el alumno ha hecho inferencias incorrectas sobre el comportamiento de una función LISP.

2.4.5 Los entornos cooperativos

El enfoque de los entornos cooperativos está en concebir la computadora como un elemento que sirve para la compartición y elaboración de información por parte de los estudiantes, de manera que se promueva un aprendizaje activo y colaborativo. El objetivo es crear entornos de construcción de conocimiento con herramientas que permitan la articulación, integración, organización y diseminación de información en diversos formatos [Scardamalia 93].

Se trata de introducir a los estudiantes en el mismo tipo de procesos intelectuales o culturales que se usan en la comunidad científica y que sirven para lograr avances en el conocimiento. Cada vez resulta más frecuente tener que trabajar en colaboración con otras personas, trasladándose esta idea al entorno de enseñanza, de modo que cada alumno trate de aportar a la comunidad (clase) información que aumente el conocimiento común. Esto se refleja en el desarrollo de aplicaciones que facilitan el proceso en general, pudiéndose crear escuelas virtuales mediante correo electrónico, conferencias por computadora, sistemas de compartición de noticias, programas de hipermedia y otros sistemas más específicos que se ven beneficiados por el desarrollo que se ha producido en las redes de comunicaciones (p.e., Internet) [Ibrahim 94, Verdejo 95, Fernández Manjón 95b]. Se trata de generalizar el concepto de aula, independientemente de la distancia y como un sistema abierto que facilite el acceso e intercambio de información.

Este planteamiento no está exento de problemas, ya que normalmente se pone el énfasis en la compartición y el acceso a la información, lo que puede llevar a los problemas de sobrecarga informativa y desorientación [Conklin 87, Kommers 92, Buenaga 95]. Para evitar esos riesgos y que se pueda producir un verdadero aprendizaje, el estudiante necesita utilidades que le ayuden a comprender mejor la información, relacionándola con lo que ya conoce, y que le permitan identificar, comparar e integrar diferentes interpretaciones de una misma idea. CLARE (Collaborative Learning and Research Environment) [Wan 94] es un ejemplo de aplicación bajo estos supuestos. Otro ejemplo es CSILE (Computer-Supported Intentional Learning Environment) [Scardamalia 91], un sistema de hipermedia cuyo núcleo es una base de datos generada por los estudiantes. El objeto básico con el que se trabaja es la nota o idea. Los estudiantes pueden expresar y almacenar sus ideas, sus dudas y comentarios, de modo que posteriormente puedan recuperarlas, enlazarlas con otra información, extenderlas, aclararlas, etc. Se busca

lograr un aprendizaje activo y totalmente controlado por el alumno, lo que se produce mediante la compartición y elaboración de la información disponible en el sistema.

2.5 Un análisis crítico de los tutores inteligentes

Los primeros proyectos que aplicaron técnicas de inteligencia artificial en el campo educativo (recopilados en los ya clásicos trabajos de Brown y Sleeman, de 1982, y Wenger, de 1987) crearon unas expectativas que actualmente parecen difíciles de cumplir. A pesar de ser un área de trabajo muy activa (lo que queda patente en el gran número de publicaciones y congresos sobre el tema), su uso en la enseñanza no ha alcanzado el grado de generalización esperado y la aplicación de tutores inteligentes en las aulas actualmente no es algo habitual [Boulay 92, Bork 91, Bork 92, Frasson 92].

Varias son las causas que pueden ayudar a explicar esta situación. Por ejemplo, en [Major 92] se destaca que, en general, los tutores están restringidos a un dominio particular, no resultando fácil adaptarlos y configurarlos para otros dominios. Además, implementan una determinada estrategia de enseñanza y no permiten modificarla o personalizarla. Nosotros creemos que el principal problema que hoy en día aqueja al diseño y construcción de programas tutoriales de calidad es su enorme complejidad y la poco favorable relación coste/eficacia de los mismos [Fernández Manjón 94, Fernández Manjón 95a]. Las principales dificultades en la construcción de tutores se encuentran por una parte en aspectos puramente informáticos, tales como la falta de metodologías de desarrollo o las limitaciones de las técnicas de inteligencia artificial utilizadas para representar los distintos modelos (del alumno, del tutor y del dominio). Otras dificultades se deben a que algunos aspectos relacionados con la psicología educativa, como la motivación, las estrategias tutoriales y el aprendizaje por parte del alumno, aún no se han llegado a comprender completamente [O'Shea 85, Kearsley 87, Breuker 90].

2.5.1 Limitaciones de los modelos del dominio y del alumno

Una primera consideración es que la pretensión de lograr un modelado completo, tanto del dominio de conocimiento como del alumno, para construir un ITS es un objetivo actualmente demasiado ambicioso.

El proceso de adquisición de modelos es la fase más costosa de la construcción de los sistemas basados en conocimiento (entre los que se encuentran los ITS), y más en el caso de los tutores debido a la diversidad del conocimiento [Eisenstatd 93]. No sólo hay que adquirir información sobre el dominio, sino también sobre cómo enseñar la materia concreta y cómo plasmar finalmente la estrategia de enseñanza teniendo en cuenta al estudiante [Woolf 87c]. Un ITS exige que se disponga de modelos muy completos que actualmente son excesivamente costosos de adquirir y mantener.

El modelo del dominio debe ser completo, ya que contiene todo el conocimiento del tutor sobre la materia que se enseña. Este modelo está normalmente constituido por bases de conocimiento específicas, muy dependientes de la aplicación final para la que han sido diseñadas y, por tanto, difícilmente trasladables a otras tareas (difícil reutilización). La especificidad de las técnicas de implementación provoca que cuando se construye un tutor, el desarrollo de la base de conocimiento tenga que empezar desde cero. Esto encarece el proceso y limita su tamaño, lo que unido a la necesidad de disponer de un modelo completo, dificulta su aplicación en dominios extensos y complejos. En el modelo del alumno, con el objetivo de adaptarse al estudiante se trata de determinar en cada momento cuál es su estado cognitivo (qué sabe, qué no sabe o qué ha entendido mal), para lo que es necesario disponer de una información muy precisa. Un ejemplo de esta situación es el proceso de diagnosis, mediante el cual cuando se produce un error se trata de determinar cuál es la causa (si algo se ha entendido mal, si no sabe algún concepto u otra razón) para poder planificar una acción correctora.

2.5.2 La motivación y los entornos de aplicación

La motivación de los estudiantes es un tema clave que no ha sido suficientemente tratado y que nos lleva a considerar de forma crítica la suposición de que las computadoras, o por lo menos los ITS, tienen un papel importante que jugar en todas las situaciones educativas. Más bien, creemos que la situación sería la inversa: los entornos y situaciones educativas en las que los ITS son la respuesta más adecuada son muy específicos y deben ser definidos con toda la precisión posible.

El hecho es que en muchas situaciones en las que se proporciona un tutor inteligente, los usuarios no lo utilizan. Este problema de aceptación de los tutores está motivado, en parte, por la excesiva rigidez que se impone en la interacción. Como la interfaz es crucial en la aprobación del sistema por parte de los estudiantes, hay que dedicar una mayor atención a su desarrollo, de modo que sea atractivo, sencillo de manejar y que permita una mayor iniciativa al alumno [Woolf 92]. Por otro lado, es frecuente que el usuario no desee una información hasta que no le surge la necesidad de utilizarla [Fernández Manjón 93, Carroll 88]. Por tanto, hay que tender hacia aplicaciones específicas de los tutores que planteen la integración de la enseñanza con el uso real de esos conocimientos.

Los ITS resultan adecuados, por ejemplo, para las tareas de entrenamiento en entornos industriales, debido a que puede resultar poco recomendable o poco factible el uso de las instalaciones reales para realizar prácticas, por el coste que conllevaría o el riesgo que podría suponer (p.e., las plantas industriales complejas o las centrales nucleares) [Woolf 87b, Díaz 92, Gutiérrez Serrano 94]. Su uso se ha demostrado que es efectivo en entornos profesionales y centros de trabajo para reciclaje y perfeccionamiento del personal, pero este resultado se ve matizado debido a que en estos casos su utilización resulta obligatoria [Pitts 92, Feifer 91]. No obstante, y a pesar de que su aplicación real es escasa, también se han obtenido algunos éxitos en la aplicación de los tutores inteligentes en las aulas, como por

ejemplo con los sistemas de Anderson, para la enseñanza de la programación (Lisp Tutor), de la geometría o del álgebra [Anderson 90].

2.5.3 La experiencia tutorial

Para construir tutores inteligentes realmente efectivos es preciso comprender mejor el problema del aprendizaje por parte del alumno y de la experiencia tutorial. Se necesitan modelos eficaces, que permitan diferentes enfoques sobre cómo enseñar y comunicar la información del dominio a la vez que se tiene en cuenta al estudiante [Breuker 88, Wenger 87].

En la mayoría de los casos, en los trabajos de aplicación de las computadoras en la educación no se establecen de una forma precisa los principios pedagógicos sobre los que se fundamenta la enseñanza [Sokolnicki 91]. Por otra parte, como destaca Merril, las teorías educativas disponibles, además de ser parciales y muchas veces contradictorias, son de muy alto nivel siendo difícil su traspaso real a un tutor.

> *"Unfortunately, much available information on learning and instruction is rather non specific —requiring considerable interpretation to translate these general educational principles into the precise rules needed to implement an intelligent instructional system." [Merril 87]*

De hecho, actualmente no están disponibles módulos generales de estrategia tutorial que se puedan utilizar en la construcción de un tutor. Por tanto, en cada nuevo sistema se desarrollan tácticas específicas para el dominio particular [Kommers 92, Lewis 87].

2.5.4 La metodología de desarrollo

La construcción de tutores inteligentes sigue siendo todavía pretecnológica, sin que exista una metodología clara y contrastada para su desarrollo [Dear 87, Breuker 88, Breuker 92, Winkels 92a, Eisentadt 93]. No existe una noción común, ampliamente aceptada, sobre lo que significa diseñar un ITS y qué fases hay que seguir. Podría decirse que en la construcción de un ITS se dan, en grado máximo, todos los problemas a los que se hace referencia en la ingeniería del software [Pressman 93].

Esta falta de metodología lleva consigo otros problemas asociados como: el aumento del coste de producción, tanto en tiempo como en dinero; la práctica inexistencia de herramientas o técnicas que simplifiquen la tarea de construcción de un ITS; y las limitaciones en fiabilidad y escalabilidad de los enfoques actuales, y por tanto su viabilidad a largo plazo [Solkolnicki 91, Pitts 92].

Uno de los obstáculos del desarrollo de los sistemas ITS es la escasa disposición de entornos de desarrollo (*shells*) o de conjuntos de herramientas suficientemente probadas que simplifiquen el desarrollo. Ha habido algunos intentos de construcción de tales shells para tutores inteligentes como, por ejemplo, COCA [Major 92], pero su éxito ha sido muy limitado. Según Breuker, las razones que pueden explicar esto son que los ITS tienen una arquitectura más articulada y compleja, y que emplean

formalismos heterogéneos para representar el conocimiento necesario para el proceso educativo [Breuker 90].

2.6 De los tutores a los sistemas de ayuda inteligente

A pesar de los problemas que acabamos de analizar, la utilización de la Informática en la educación es una experiencia claramente positiva y con cada vez más futuro. Los tutores inteligentes han contribuido de forma importante a la difusión de ideas novedosas tales como el modelado del alumno (usuario) y el desarrollo de interfaces amigables, actualmente de uso común en otras muchas áreas de aplicación de la Informática [Sullivan 91]. El trabajo desarrollado con los tutores ha sido positivo y es el punto de partida para nuestra propuesta de sistemas de ayuda inteligentes (también denominados asistentes). En este trabajo consideramos que utilizando técnicas similares a las de los tutores, integradas con ideas de otros enfoques educativos (una mayor capacidad de control y de iniciativa para el usuario o una interfaz amigable que permita distintos tipos de interacción), es posible construir sistemas de ayuda eficientes.

Los sistemas de ayuda inteligentes son una alternativa educativa, más simple pero útil, y de cuyos resultados se pueden obtener conclusiones para la construcción de futuros tutores inteligentes. Esta idea está en la misma línea que otros autores:

> *"It seems paradoxical that while learning to use computing equipment is difficult, computers are seen as having the potential to enable major advances in education. How can one learn with a tool that it is itself hard to learn? A resolution of this puzzle may reside in the application of artificial intelligence techniques to education and assistance. Intelligent training and help might ease the difficulties of learning to use computing equipment and at the same time pioneer new educational technology." [Carroll 88]*

Los aspectos que hemos considerados sobre los tutores nos han llevado a centramos en situaciones educativas en las que la necesidad de enseñanza (ayuda) informatizada sea prácticamente indiscutible. En los sistemas de ayuda, el usuario se encuentra con una dificultad muy clara que tiene que resolver y para ello recurre al sistema, por lo que la motivación está garantizada. En estos casos existe una necesidad a corto plazo, la resolución de su problema, y no únicamente una motivación a largo plazo, como llegar a saber más o conocer mejor el sistema final, que es la situación habitual en un tutor.

Planteamos que la arquitectura de un sistema de ayuda inteligente puede seguir la línea creada por los ITS con algunas simplificaciones [Fernández Manjón 95c]. Estas simplificaciones las consideramos en dos dimensiones. Desde un punto de vista arquitectural, un sistema de ayuda se puede considerar como un ITS en el cual el módulo tutor se ha simplificado mucho, siguiendo el enfoque aplicado en los tutores basados en casos [Schank 91]. Desde el punto de vista del significado de la información, consideramos que en los tutores se plantea un modelado completo,

tanto del dominio como del estudiante, que hoy en día es un proceso demasiado complejo.

Los sistemas de ayuda no tienen exigencias tan fuertes como los tutores, ya que se plantean como apoyo y no como la única fuente de información. Al ser un complemento que el usuario puede utilizar, incluso aunque en algunas situaciones no se obtengan los resultados deseados, sigue resultando útil [Wilensky 89]. Esta circunstancia, unida a la iniciativa del usuario, plantea unos condicionantes pedagógicos menores que en un tutor, lo que permite evitar parte de los procesos más costosos de su construcción. Por ejemplo, si a partir de una consulta el asistente presenta información sobre el dominio que facilite al usuario la obtención de las respuestas que necesita, se puede evitar el complejo proceso de diagnosis. Además, se puede trabajar con un modelo parcial del dominio que se puede refinar y completar de modo incremental. Finalmente, un modelo de usuario aproximado es suficiente para lograr un comportamiento adaptativo por parte del sistema. En este trabajo preferimos utilizar el término modelo del usuario y no modelo del estudiante, ya que su objetivo es mantener toda la información relevante a la comunicación entre el sistema y el usuario, sin la pretensión de que refleje el estado cognitivo de esa persona [Elsom-Cook 88, Fernández Manjón 94].

Por otro lado, en un sistema de ayuda no sería necesario inicialmente incluir explícitamente un módulo tutorial, sino que a partir del contenido del modelo del usuario y del modelo del dominio se pueden implementar estrategias simples de presentación de información. Por ejemplo, se puede presentar la información que resulte adecuada para el interés o el nivel de experiencia del usuario, o proporcionar información previa, que se presupone desconocida, y que es necesaria para la comprensión de algunos de los temas presentados.

El planteamiento de otros factores pragmáticos, como la escalabilidad o el coste de producción de los asistentes, es crucial debido a que como ya hemos tratado para los ITS influyen de modo decisivo en su uso real. Este aspecto se aborda mediante el desarrollo modular, la reutilización de la información y la integración de técnicas que previamente hayan sido aplicados con éxito. Tecnologías como la recuperación de información y el hipertexto se utilizan para simplificar la interacción con el usuario y el acceso a la información Por lo que a las técnicas más propias de los tutores se refiere, como la representación del conocimiento y el modelado del usuario, se tratará de reutilizar trabajos anteriores y realizar su implementación con herramientas estándar que simplifiquen su ampliabilidad y mantenimiento..

2.7 Resumen

En este capítulo hemos realizado, en primer lugar, una introducción histórica de la Enseñanza Asistida por Computadora (CAI), sus fundamentos, limitaciones y evolución hasta la aparición de los tutores inteligentes (ITS). Un ITS hace uso de técnicas de inteligencia artificial, que permiten una representación explícita del conocimiento, tanto sobre la materia a enseñar como sobre las decisiones tutoriales, y una mayor adaptación a las necesidades del estudiante. Su estructura básica está compuesta por cuatro módulos que pretenden modelar los aspectos del proceso

educativo: un área, o dominio, de conocimiento; el estado cognitivo del alumno durante el proceso de aprendizaje; una estrategia pedagógica; y una interfaz, o entorno, que facilite la comunicación entre el estudiante y la computadora.

También hemos mostrado otros enfoques de la utilización de la computadora en la enseñanza, proporcionando una visión más amplia del campo y centrándonos en las aportaciones relevantes para nuestro trabajo, bien por ser directrices a seguir o por ser ideas con las que contrastar. Hemos tratado las simulaciones y micromundos, la enseñanza basada en casos, la enseñanza mediante descubrimiento guiado, los entornos interactivos de aprendizaje y los entornos cooperativos.

Posteriormente hemos realizado un análisis de la problemática asociada con el uso de los tutores inteligentes en la formación y estudiado las razones por las que su uso no se ha generalizado. Estas consideraciones nos han llevado a proponer los sistemas de ayuda como una aproximación más sencilla y realista de un sistema de enseñanza basado en computadora. Con un sistema de ayuda podemos lograr parte de los objetivos pedagógicos perseguidos por un tutor, teniendo unas características menos exigentes que los tutores. En la misma línea de lo propuesto por otros autores [Marchionini 95], consideramos la ayuda y la enseñanza como dos partes de un mismo proceso; la ayuda es una enseñanza dirigida por un objetivo claro que es la resolución de un problema y no únicamente por el propio aprendizaje.

El objetivo de este capítulo ha sido presentar una visión general de los diversos modos de utilización de las computadoras en el mundo educativo, lo que permite encuadrar nuestra propuesta de un sistema de ayuda en este campo. Hemos dedicado una atención especial a los tutores inteligentes, que constituyen el punto de referencia de nuestro trabajo, y posteriormente hemos analizado otros enfoques educativos que incorporan ideas relevantes para nuestro sistema. Finalmente, hemos tratado los problemas asociados con el uso de las computadoras en la enseñanza y presentado de forma muy breve los sistemas de ayuda inteligentes.

Nuestra propuesta, que se encuadrará y detallará en los siguientes capítulos, es un modelo para el desarrollo de sistemas de ayuda inteligentes que surge como una simplificación del modelo utilizado por los tutores. Consideramos que con la integración de diversas técnicas desarrolladas por los ITS, junto con otras como el hipertexto y la recuperación de información, es posible construir un sistema de ayuda eficiente, de bajo coste y que se adapte de forma adecuada a las necesidades del usuario.

CAPÍTULO 3

SISTEMAS DE AYUDA Y SISTEMAS DE AYUDA INTELIGENTE

3.1 Introducción

El desarrollo de sistemas de ayuda es un campo relativamente reciente en el diseño de aplicaciones informáticas, no siendo habitual encontrarlos en programas anteriores al año 1975. Sin embargo, ahora la mayor parte de las aplicaciones incluyen algún tipo de ayuda. En teoría, con la mejora de las interfaces y del diseño de programas, actualmente sería posible construir aplicaciones tan sencillas de utilizar que el usuario no necesitase ningún tipo de ayuda adicional. Sin embargo, circunstancias como la creciente complejidad de los sistemas informáticos, su continua evolución y su uso generalizado por usuarios no expertos, hacen que los sistemas de ayuda sean un componente fundamental para simplificar el uso y el aprendizaje de los entornos informáticos.

En los últimos años se han publicado numerosos trabajos que versan sobre los sistemas de ayuda, lo que indica que se trata de un campo de trabajo muy activo, aunque a pesar de ello el marco de estos sistemas no está claramente definido [Winkels 92a, Kearsley 88, Duffy 92, Boy 91, Breuker 90]. Esta falta de una definición precisa se debe principalmente a dos razones: las diferentes formas de entender lo que es la ayuda informatizada y las diferentes áreas de aplicación. Los sistemas de ayuda se aplican en muy variados campos, proporcionando ayuda a los usuarios en situaciones muy distintas. Así, por ejemplo, existen sistemas de ayuda tanto para las instalaciones industriales complejas como para la navegación aérea o la toma de decisiones bancarias. Dado el marco de aplicación de nuestro trabajo, este estudio se centrará en los sistemas de ayuda utilizados para proporcionar asistencia a los usuarios de aplicaciones informáticas de uso general, tales como los sistemas operativos, que son utilizados de un modo generalizado por muy distintos tipos de usuarios (los términos sistema informático, entorno informático o aplicación software los utilizaremos en este contexto como sinónimos de aplicación informática de uso general).

En este capítulo comenzaremos estudiando la necesidad de los sistemas de ayuda para facilitar la correcta utilización de los entornos informáticos. Continuaremos con un análisis de los enfoques que buscan facilitar al mismo tiempo el aprendizaje

y el uso de las aplicaciones. Pasaremos a estudiar los factores más importantes que intervienen en el diseño y la construcción de los sistemas de ayuda. Este estudio, junto con el de tres de los sistemas de ayuda más relevantes que llevaremos a cabo en el último apartado, constituirá el marco de referencia para el desarrollo de nuestro trabajo. Finalmente, analizamos tres proyectos de creación de bases de conocimiento que en un futuro podrían contribuir a simplificar la construcción de asistentes inteligentes como los propuestos en esta tesis.

3.2 La necesidad de los sistemas de ayuda en los entornos informáticos

Uno de los objetivos ideales de cualquier aplicación software bien diseñada sería tener una estructura tan evidente y una interfaz tan comprensible para el usuario, que no fuera necesario proporcionar ningún tipo de ayuda adicional para su perfecto uso. No obstante, en general, en las aplicaciones dicho objetivo está lejos de cumplirse, por lo que se hace necesario proporcionar asistencia a los usuarios. En nuestra opinión, la mejor forma de proporcionar dicha asistencia es mediante la incorporación de sistemas de ayuda en las aplicaciones [Fernández Manjón 94]. Nuestra visión coincide con la opinión de autores como [Breuker 90, Duffy 89, Bork 92, Kearsley 88].

Aunque la tecnología de desarrollo de interfaces ha mejorado considerablemente, la generalización del uso de los sistemas informáticos y su creciente complejidad hacen que siga habiendo problemas asociados con su utilización. Los mayores esfuerzos que se han llevado a cabo para facilitar el uso de las aplicaciones informáticas se han dedicado a la mejora de las interfaces, haciéndolas lo más amigables posible. Se intenta que las interfaces sean cada vez más intuitivas, que no requieran que el usuario posea una gran formación o una gran experiencia técnica. Tales interfaces consiguen que la necesidad de ayuda del usuario disminuya, haciendo incluso que en ocasiones la ayuda no resulte prácticamente necesaria. Esta es la base del éxito de las aplicaciones basadas en ventanas y conducidas por menús, observándose una implantación generalizada de las interfaces icónicas basadas en representaciones gráficas de los elementos (objetos) que se utilizan cotidianamente. Por ejemplo, el enfoque de la metáfora del escritorio adopta unos elementos (carpetas, archivos, papelera, etc.) que resultan familiares y que permiten identificar fácilmente los objetos manejados por una aplicación con aquellos que se encuentran en un entorno de trabajo convencional (por ejemplo, en una oficina). Sin embargo, a la vez que mejora la interacción con el software, también aumentan sus funcionalidades, resultando complicado diseñar interfaces que hagan que esos sistemas inherentemente complejos sean fáciles de manejar y de entender sin algún tipo de aprendizaje previo [Sullivan 91].

Varias son las nuevas circunstancias que hacen que aumente la necesidad de ayuda por parte de los usuarios. Por una parte, como ya hemos comentado, los programas ofrecen cada vez mayores funcionalidades, permitiendo realizar operaciones más complejas y sofisticadas. Por otra parte, y de forma simultánea, el número de aplicaciones software va creciendo y su uso se generaliza, introduciendo a nuevas

clases de usuarios en el mundo informático. Las computadoras ya no sólo son utilizadas por "expertos" con una formación técnica y una sólida base informática, sino también por otras personas, a menudo con escasos conocimientos de los conceptos básicos y con una mínima o nula experiencia. Muchas de las aplicaciones se utilizan de forma ocasional, o durante cortos periodos de tiempo, para satisfacer necesidades concretas (por ejemplo, la generación de presentaciones o la creación de gráficos en una hoja de cálculo), sin que se llegue a dominar su uso. Además de esa utilización ocasional, también es frecuente el cambio de un programa a otro dentro de la misma área de aplicación, como por ejemplo el cambio de un programa de edición de textos a uno de otra casa, así como la evolución a nuevas versiones del mismo programa que ofrecen nuevas capacidades (siendo habitual que se haya modificado la interfaz). En estos casos, los usuarios intentan aplicar su conocimiento anterior y, a pesar de su experiencia, es muy probable que se encuentren con problemas en su uso y necesiten ayuda [Duffy 92]. Otro porcentaje importante de usuarios de sistemas informáticos están interesados sólo en utilidades específicas del software y tienen un conocimiento limitado del resto del sistema con el que trabajan. Este es el caso de muchos usuarios de estaciones de trabajo que usan únicamente un programa de CAD/CAM, de cálculos estadísticos o de autoedición. Cuando estos usuarios tienen que hacer cualquier operación básica con otro software diferente o desde el sistema operativo, se encuentran con dificultades a veces insalvables.

Para solucionar estos problemas, la mejor alternativa se encuentra en la potenciación y mejora de los sistemas de ayuda. En este sentido enmarcamos nuestra propuesta. Mediante los sistemas de ayuda se puede integrar el uso y el aprendizaje de los programas, de modo que sea posible su utilización a partir de unos conocimientos mínimos y se proporcione asistencia al usuario cuando éste encuentre problemas que no sabe resolver [Carroll 88, Fischer 91].

3.3 Soluciones para facilitar el aprendizaje y la utilización de las aplicaciones software

Las aplicaciones software se diseñan para que los usuarios puedan llevar a cabo una determinada serie de operaciones (p.e., procesar textos, comunicar con otras computadoras o transferir información). Pero aún facilitando esas tareas, para la correcta utilización de una aplicación resulta cuanto menos conveniente disponer de algunos conocimientos de su estructura interna. Las habilidades que son útiles en circunstancias normales para completar determinadas tareas, dejan de serlo cuando se presenta alguna situación de error o cuando algunas circunstancias externas modifican las condiciones habituales de uso. En estas situaciones, para poder solucionar el error o para adaptarse a las nuevas condiciones, es necesario comprender qué es lo que está pasando y cuáles son los efectos de las operaciones que se realizan. Esta comprensión del funcionamiento de la aplicación por parte del usuario evita que se vuelvan a cometer los mismos errores en ocasiones posteriores.

La forma más habitual de aprendizaje del manejo de una aplicación software es mediante cursos especializados y el estudio de los manuales -en formato impreso-

que con ella se proporcionan, siendo la computadora un mero instrumento y no un elemento activo del proceso de aprendizaje. A continuación presentamos y analizamos formas alternativas de aprendizaje en las que se utilizan las propias computadoras para facilitar tanto ese aprendizaje como el uso en sí de las aplicaciones. Las alternativas están agrupadas de acuerdo con que su objetivo primario sea mejorar la interacción o bien ayudar/enseñar al usuario [Breuker 90]. Las fronteras entre las diferentes propuestas no están claramente definidas y las técnicas no son excluyentes, siendo común encontrar sistemas que integran varias [Maddix 90, Marchionini 91].

3.3.1 Mejora de la interacción con el usuario

Para una mejora de la interacción y comunicación con el usuario hay distintas propuestas que van desde el perfeccionamiento de las interfaces gráficas hasta el fomento de un comportamiento cooperativo en los propios sistemas que haga que éstos colaboren de forma automática con el usuario [Puerta 94].

3.3.1.1 Interfaces de manipulación directa y metáforas

Este tipo de interfaces intentan o bien proporcionar medios que faciliten el trabajo que supone la planificación de tareas, o bien proporcionar un acceso más transparente a los objetos que el usuario ha de manipular, o ambas cosas. Su diseño se basa en las ideas del modelo cognitivo de procesamiento de información. Por ejemplo, según el modelo los seres humanos tenemos una memoria de trabajo limitada (de aproximadamente siete elementos) y recordar la información nos exige más esfuerzo que únicamente reconocer dicha información [Marchionini 91]. Entre los distintos estilos de interfaces que siguen este modelo podemos citar los sistemas de menús, las consultas dirigidas por ejemplos, las metáforas y la manipulación directa. Por ejemplo, en las interfaces basadas en menús, el número de opciones siempre está limitado y los símbolos o nombres de las opciones sirven como mnemotécnicos de las acciones que representan. Esta forma de presentar las posibles operaciones facilita el trabajo de planificación que tiene que realizar el usuario, evitando que tenga que aprender cuáles son las operaciones posibles.

Otro estilo de interfaz ampliamente utilizado es el que gira en torno a metáforas. Lo que se intenta es explotar el conocimiento estructural o funcional que el usuario tiene, desarrollando interfaces más comprensibles al utilizar analogías con los elementos que los usuarios ya saben cómo funcionan o para qué sirven. Una metáfora relaciona el conocimiento previo del usuario con una función del sistema. Uno de los ejemplos más conocidos es el de la "papelera" asociada con el escritorio y utilizada por primera vez en las estaciones de trabajo Xerox Star y posteriormente generalizada con los sistemas operativos de Apple y, recientemente, de Microsoft. Con esta metáfora visual, la eliminación de un archivo se convierte en una secuencia natural de selección del objeto archivo (representado mediante un icono), su arrastre hasta el icono de la papelera y su depósito en ella [Maddix 90].

Este tipo de interacción resulta muy adecuada para los usuarios noveles u ocasionales, quienes necesitan una información más dirigida a la tarea en curso,

mientras que para los usuarios avanzados, que ya tienen un conocimiento suficiente del sistema, la interacción puede hacerse pesada, principalmente por ser más lenta. Asociadas con las representaciones gráficas o icónicas, se detectan ciertas dificultades, como el que no resulte sencillo realizar ciertas operaciones complejas (como, por ejemplo, copiar únicamente los archivos que hayan sido modificados) o que pueda resultar confuso el significado de un determinado icono [Downtown 91]. Cabe destacar que unido a la simplificación del uso, también se puede restringir la visión que tiene el usuario de algunos aspectos del sistema, de modo que esa facilidad de aprendizaje/uso puede llegar a interferir con la posibilidad de una utilización más versátil. Se puede concluir que no hay interfaces que resulten completamente evidentes y que satisfagan simultáneamente a todos los tipos de usuarios [Breuker 90].

3.3.1.2 Agentes

En la forma habitual de interacción con las aplicaciones, el usuario ha de tomar la iniciativa para todas las tareas que tiene que llevar a cabo. Además, debe tener en cuenta todas las circunstancias (p.e., mensajes, errores) que se pueden producir hasta conseguir finalizar cada tarea. Esto hace que la interacción pueda convertirse en un proceso complejo. Para facilitar este proceso, los agentes aparecen como apoyo complementario, fomentando un estilo de interacción que se denomina gestión indirecta. En este tipo de interacción, el usuario se ve involucrado en un proceso cooperativo en el cual, tanto él como el agente, pueden iniciar un proceso de comunicación, monitorizar los eventos y realizar tareas. El agente colabora con el usuario dentro del propio entorno de trabajo (como un asistente personal).

No es necesario que la incorporación del agente requiera la creación de una nueva interfaz entre la computadora y el usuario. De hecho, los agentes que han tenido más éxito son aquellos que complementan la interfaz actual y no impiden que el usuario pueda realizar las mismas acciones de una forma personal e independiente [Maes 94]. Los agentes pueden ayudar a los usuarios de diferentes formas como, por ejemplo, realizando de forma automática ciertas operaciones difíciles, ocultando la complejidad o los detalles de las operaciones, o monitorizando los eventos que se puedan producir.

Con este enfoque, además de permitir que el sistema tome la iniciativa en la interacción, también se modifica la forma en que el usuario puede expresar qué es lo que desea conseguir. Se trata de un enfoque de interacción basada en los objetivos. El usuario simplemente especifica un objetivo a cumplir y es el agente el que tiene que decidir cómo se debe llevar a cabo, en función de su base de conocimiento, del estado del sistema y de las órdenes de las que se dispone. El agente sintetiza dinámicamente la secuencia de órdenes apropiadas, ejecuta esas órdenes, gestiona los posibles errores y reintenta la ejecución de alguna orden si ha habido problemas. Se aplican técnicas de planificación y de aprendizaje para ampliar las capacidades de las órdenes de usuario, de modo que el usuario sólo debe preocuparse de expresar cuál es su necesidad, y es el agente el que tiene que conocer cuál es la forma de satisfacerla. Si se introduce una nueva utilidad en el sistema, sólo habrá que notificársela al agente y el usuario se estará beneficiando de ella sin tener que aprender a utilizarla. Esto hace que constituya una solución

muy adecuada para sistemas abiertos en los que los cambios y mejoras son frecuentes [Etzioni 92, Etzioni 93].

Uno de los problemas que se puede destacar en este enfoque viene dado por la dificultad que puede encontrar el usuario en especificar los objetivos que quiere alcanzar o, en términos más generales, la forma de comunicación entre el usuario y el agente. Además, con esta forma de interacción se impide en cierto modo que el usuario adquiera un conocimiento más profundo sobre la aplicación, ya que ésta queda "ocultada" por la capa constituida por el agente. En caso de pérdida del agente, o de falta de disponibilidad momentánea, el usuario vuelve a su nivel de experiencia, inferior (incluso de novato) respecto de las utilidades implementadas mediante el agente [Selker 94].

3.3.1.3 Interfaces inteligentes

Las llamadas interfaces inteligentes tratan de entender de forma activa qué es lo que el usuario intenta hacer e integran distintas ayudas dentro de la interfaz [Jerram-Smith 89, Duffy 89, Cooper 88]. Interfaz inteligente es un término muy amplio que engloba desde las interfaces más simples, que solucionan pequeños errores de escritura, hasta las interfaces que se basan en una planificación automática para adaptar su comportamiento al usuario y a la tarea que se está realizando en ese momento [Puerta 92, Winkels 92a]. Incluso hay autores que consideran este término en un sentido aún más amplio, incluyendo el acceso a la documentación y la provisión de ayuda interactiva, lo que se englobaría dentro de lo que se entiende como ayuda inteligente [Wasson 92].

Con un campo tan extenso y diverso, resulta difícil caracterizar una interfaz inteligente, aunque existe un cierto consenso general que establece que tales interfaces deben ser capaces de inferir y evaluar los planes e intenciones del usuario, con el fin de adaptar su comportamiento a éste y a la tarea que está realizando en ese momento (todo ello, basándose en una representación explícita del conocimiento) [Tyler 91]. El conocimiento de los objetivos del usuario y de los planes para llevarlos a cabo proporciona un contexto muy poderoso para la interacción, permitiendo que el sistema realice acciones por el usuario, sin que requiera que éste detalle las órdenes típicas, lo que siempre puede suponer un riesgo de errores. El sistema también puede realizar sofisticadas detecciones y correcciones de errores, encontrando debilidades en los planes del usuario a partir del conocimiento de que dispone de la aplicación. La interfaz inteligente también debe permitir una entrada multimodal, aceptando, por ejemplo, que se utilice un dispositivo apuntador (típicamente el ratón) a la vez que el lenguaje natural para realizar peticiones al sistema. Esta entrada multimodal facilita la interacción, asegurando que los usuarios puedan relacionar sus tareas con acciones que le sean más naturales y les suponga un menor esfuerzo que si lo tuvieran que hacer de una sola forma. Además, el conocimiento del que se dispone se puede utilizar a la hora de mostrar los resultados, pudiendo tener en cuenta al usuario, la tarea en curso y la propia naturaleza de la información para que el usuario detecte de una forma más sencilla cuáles son las relaciones relevantes.

Este tipo de interfaces eliminan parte de los problemas de comprensión y uso que están asociados con las interfaces tradicionales, al comprender los objetivos del usuario y colaborar con él. Sin embargo, también se pueden identificar algunos problemas derivados del uso de este enfoque. Por ejemplo, la necesidad de evaluar los planes y los objetivos del usuario inevitablemente es un proceso costoso. Además, el desarrollo de este tipo de interfaces sólo es posible de forma conjunta e integrada con el de la propia aplicación.

3.3.2 Sistemas de ayuda y sistemas de enseñanza

El otro método principal para facilitar el uso de las computadoras viene dado por la utilización de programas que proporcionen asistencia a los usuarios y simplifiquen la adquisición de las habilidades básicas necesarias para la correcta utilización del software. Las propias computadoras se utilizan como complemento (o incluso sustitutas) de los cursos y de los manuales de instrucción, cuyo objetivo es que el usuario pueda hacer un mejor uso de las aplicaciones. Para este propósito se han seguido distintos enfoques, como los sistemas de ayuda de muy diversa naturaleza, los sistemas de acceso a la documentación y los sistemas de entrenamiento (tutores). Todavía no se ha establecido una frontera clara entre todas estas alternativas.

Aquí consideramos que la diferencia fundamental entre los distintos enfoques se puede establecer mediante dos criterios:

a) cuál es el objetivo principal del sistema y

b) cuál es su contexto de uso

El objetivo de un sistema de ayuda es proporcionar la información precisa cuando sea necesaria para poder resolver un problema concreto que se presenta en el curso del trabajo. Por contra, el objetivo principal de los sistemas de enseñanza, ya sean inteligentes o tradicionales, es el aprendizaje a medio o largo plazo. En estos, la información se presenta de forma progresiva, de acuerdo con criterios pedagógicos, sin preocuparse por solucionar problemas concretos que aparezcan durante la realización de una determinada tarea.

El contexto de los sistemas de ayuda es el trabajo habitual. El usuario está inmerso en su propio trabajo en una situación productiva y necesita ayuda. Por el contrario, en los sistemas de enseñanza normalmente al usuario se le presentan una serie de situaciones creadas artificialmente (ejercicios) de modo que pueda practicar y adquirir una serie de habilidades en el uso de la aplicación. Es decir se realizan una serie de prácticas, no tareas reales.

Actualmente, cada vez es más frecuente que, como una tarea más del proceso de desarrollo de una aplicación, se produzca un tutorial sobre el uso del producto. Este tutorial puede venderse como un elemento aparte, pero lo más habitual es que se adjunte al propio programa, e incluso que se pueda acceder a él desde la aplicación. Sin embargo, si existe esa combinación aplicación/tutorial, normalmente la relación

entre ambos es superficial, ya que, por ejemplo, el tutorial no tiene en cuenta las actividades pasadas del usuario, y las opciones que proporciona suelen ser muy reducidas [Ibrahim 93]. Se han desarrollado sistemas de enseñanza eficientes, habiendo incluso cobrado un nuevo protagonismo gracias a los avances de las tecnologías multimedia, pero en general esos sistemas adolecen de los mismos problemas que ya identificamos para los tutores inteligentes en el capítulo anterior (véase la Sección 2.5).

En los apartados que siguen analizamos los sistemas de ayuda, tratando sus tipos y los distintos enfoques. Este estudio nos va a permitir encuadrar nuestra propuesta de modelo para la construcción de asistentes inteligentes.

3.4 Análisis de los factores que condicionan el diseño y la construcción de sistemas de ayuda

Un sistema de ayuda interactiva para una aplicación informática es un programa diseñado para asistir al usuario en el uso de la aplicación. Es decir, su propósito principal es el de proporcionar una respuesta inmediata a los problemas específicos que se presentan en la utilización de la aplicación [Duffy 92, Kearsley 88, Breuker 88, Winkels 92a, Harmon 87]. Sin embargo, en la bibliografía no existe consenso sobre cómo lograr ese propósito, qué es lo que se entiende como ayuda, la terminología a utilizar o las técnicas a emplear para implementar ese apoyo a los usuarios.

3.4.1 Ayuda interactiva

Los programas que proporcionan algún tipo de ayuda interactiva a los usuarios de las aplicaciones informáticas se han venido denominado de muy diversas formas y han utilizado enfoques muy diferentes. Entre otros, se han empleado términos como sistema de ayuda, asistente, sistema de ayuda inteligente, documentación interactiva, entrenador, consejero, interfaz inteligente e incluso combinaciones de esas palabras [Wasson 92, Duffy 92, Sullivan 91]. Cada denominación tiene un significado ligeramente diferente, pero todas coinciden en que el objetivo final es hacer que el uso de una aplicación resulte más simple para los usuarios. La diferencia se encuentra en cómo lograr el objetivo. Se pueden identificar dos tendencias principales: la que considera que estos sistemas deben buscar únicamente una mejora en el rendimiento y la que considera que además deben tener un cierto objetivo pedagógico.

Carroll *et al* utilizan el término *advisory interface* para referirse genéricamente a los entrenadores, a la documentación de referencia, a la ayuda interactiva y a otros tipos de aplicaciones que proporcionan soporte a los usuarios [Carroll 88]. Postulan que se trata de uno de los medios más importantes para facilitar el uso de los sistemas informáticos. Identifican el fenómeno que denominan *paradoja de la producción* como el conflicto existente entre aprendizaje y trabajo por lo que se refiere a los usuarios de una aplicación. El usuario quiere utilizar la computadora para realizar una tarea, pero no desea invertir tiempo en aprender y perfeccionar

las habilidades que le permitirían aumentar su efectividad y su eficiencia en el uso del sistema. Por esto, un objetivo de los asistentes debe ser el de mitigar el conflicto entre aprendizaje y trabajo, produciendo una mejor integración del tiempo y el esfuerzo de aprendizaje con el uso productivo de la aplicación.

Kearsley define un sistema de ayuda como uno o más programas diseñados para proporcionar asistencia al usuario sobre una aplicación o un sistema informático [Kearsley 88]. El sistema de ayuda se debe diseñar para proporcionar asistencia inmediata a los problemas específicos que encuentra el usuario. Distingue entre los sistemas de ayuda / sistemas de documentación, aquellos que proporcionan una información más descriptiva sobre la funcionalidad del sistema, y los sistemas de entrenamiento, cuyo principal propósito es instruir al usuario siguiendo estrategias pedagógicas.

Fischer denomina *intelligent support systems* a los programas que usan técnicas de inteligencia artificial para incorporar en el propio sistema el conocimiento necesario para ayudar al usuario en el aprendizaje y uso de una aplicación compleja. Se introduce conocimiento sobre el dominio y sobre el usuario para permitir una mejora del proceso incremental de aprendizaje. De esta forma, se puede proporcionar ayuda y documentación adecuados al usuario, para ampliar su comprensión de la aplicación y, por tanto, mejorar su utilización [Fischer 87].

Duffy *et al* denominan sistemas de ayuda interactivos a los programas que ayudan a los usuarios de computadoras a interiorizar esta tecnología de modo que se mejore el rendimiento [Duffy 89]. Es decir, los que ayudan al usuario a: i) establecer un contexto para el uso de una aplicación; ii) aprender cómo lograr sus objetivos utilizando esa aplicación; y iii) adquirir las habilidades suficientes para obtener un buen rendimiento. Dejan expresamente fuera de esta denominación a los sistemas de entrenamiento y a los tutoriales interactivos, que tratan principalmente de enseñar a los usuarios.

Harmon emplea el término *intelligent job aids* para identificar a los programas que proporcionan un soporte informático a los usuario en su trabajo (no sólo en el uso de las aplicaciones software) [Harmon 87]. Plantea que resulta más productivo proporcionar sistemas que mejoren el rendimiento de un trabajador en la realización de sus tareas, que intentar utilizar un enfoque más instructivo en el que el usuario deba comprender en qué teoría se basa o por qué una tarea se realiza de una determinada manera.

Nuestra opinión es que los sistemas de ayuda deben instruir al usuario sobre cómo operar con el software, de modo que se mejore el rendimiento a la vez que se promueve una mayor comprensión del funcionamiento del sistema [Fernández Manjón 94, Fernández Manjón 95a]. Así aumenta su conocimiento de la aplicación y, con el tiempo, disminuye su necesidad de ayuda. Un punto clave de esta aproximación es la utilización de información de referencia sobre la aplicación. Consideramos que un buen compromiso entre aprendizaje y uso de los programas se puede alcanzar con unos manuales de tamaño reducido y un aprendizaje incremental, controlado por el usuario y conducido por un sistema interactivo de ayuda.

3.4.2 Ayuda inteligente

A partir de los años 80 surgieron nuevos tipos de sistemas de ayuda que basan su aporte de ayuda en la utilización de técnicas de inteligencia artificial y en la inclusión de modelos explícitos, tanto del dominio como del usuario. Son los que se denominan sistemas de ayuda inteligente [Wilensky 84]. En palabras de Hecking,

> *"we define intelligent help systems (IHS) as systems in which AI principles, techniques, and tools are used in order to simulate the performance of a qualified expert in giving helping information to other users" [Hecking 88]*

Otras propuestas consideran que un sistema de ayuda inteligente implica un sistema que pueda tomar la iniciativa para proveer ayuda que no se ha solicitado. Un ejemplo es el proyecto Eurohelp, donde se define un asistente inteligente como aquel que captura la interacción del usuario con la aplicación, siendo capaz de responder preguntas y de irrumpir para ayudar al usuario cuando hay problemas o cuando se presenta una oportunidad adecuada para enseñar al usuario alguna nueva funcionalidad del entorno [Winkels 92a, Breuker 90].

Aunque hay ciertas discrepancias sobre lo que se puede considerar como ayuda inteligente, nosotros utilizaremos el término con un significado amplio, definiendo un sistema de ayuda inteligente como aquel que tiene capacidades de adaptación a las circunstancias concretas en las que se solicita la ayuda (un tratamiento más amplio de los aspectos relativos a la inteligencia y la ayuda se puede encontrar en [Shank 91, Rich, 94, So 94]). Esta capacidad de adaptación para las ayudas inteligentes también es destacada como factor discriminante por otros autores como [de Haan 93, Wasson 92, Jones 91, Kearsley 88]. No obstante, esa capacidad de adaptación y de adecuación de la ayuda proporcionada puede ser enfocada de forma muy variada, tanto en su concepción como en su implementación final. Por ejemplo, podemos encontrar desde el disponer de información diferente para cada tipo de usuario hasta una adaptación automática del comportamiento del programa al usuario concreto para seleccionar la información que se considere más adecuada (incluso realizándose modificaciones de la interfaz, lo que se podría clasificar como interfaz inteligente, de acuerdo con el apartado 3.3.1.3). Para conseguir este comportamiento adaptativo resulta clave la utilización de una representación explícita de las características de los usuarios (modelo de usuario).

En los sistemas de ayuda, la inclusión de técnicas de inteligencia artificial creó muchas expectativas, expectativas que, al igual que con los tutores inteligentes como ya hemos destacado, no se han satisfecho completamente. Las causas que han provocado esta situación también son similares a las del caso de los tutores, sobre todo el coste de desarrollo de los sistemas más sofisticados, como ya destacaba Kearsley:

> *"Artificial intelligence research holds much promise for the design of more effective help systems. However, at present, the amount of time and special expertise required to build such systems keeps them in the research domain and out of the practical realm." [Kearsley 88]*

En la actualidad se están desarrollando comercialmente sistemas de acceso a documentación y sistemas simples de ayuda interactiva, pero los auténticos asistentes que incorporan otras utilidades como, por ejemplo, la detección de usos ineficientes o una adecuación del apoyo a cada situación y usuario, continúan siendo únicamente prototipos de investigación [Duffy 92, Wasson 92]. Nosotros creemos que es posible desarrollar asistentes inteligentes para entornos reales mediante la integración de diferentes técnicas y evitando los procesos más costosos como, por ejemplo, la introducción de planificación o la codificación de forma expresa de todas las posibles tareas que se pueden realizar en un determinado dominio. Esto hace que nos centremos en sistemas más sencillos y realistas, en los que el usuario tiene la iniciativa y se le proporcionan diversos modos de interacción para lograr los objetivos que en otros sistemas se obtienen mediante técnicas más complejas y costosas (como la comprensión del lenguaje natural o el reconocimiento de planes del usuario).

3.4.3 El contexto de los sistemas de ayuda

Se han realizado diversos estudios experimentales sobre los problemas asociados con el uso de las aplicaciones complejas de uso general que tienen un gran número de funcionalidades, tales como los sistemas operativos y, a menor escala, las hojas de cálculo o los procesadores de texto. Estos estudios empíricos (que en su totalidad contemplan como dominio de estudio, o como uno de sus dominios, el sistema operativo Unix) [Draper 84, Sutcliffe 87, Fischer 96] demuestran que los usuarios tienen un conocimiento incompleto del sistema, utilizando de forma habitual sólo una pequeña parte de las utilidades de las que dispone.

El hecho de que muchos usuarios utilicen la computadora únicamente como una herramienta que facilita la realización de sus tareas hace que no suelan desear aprender nada que no les resulte imprescindible para conseguir sus objetivos. Los usuarios conocen tan sólo un conjunto de operaciones básicas, con las que son capaces de realizar sus tareas habituales de forma satisfactoria, pero de manera que hay partes del sistema que les resultan completamente desconocidas. Esto impide que aprendan otras utilidades más sofisticadas u otras formas de trabajo que les permitirían conseguir un mejor uso de la aplicación.

A partir de estos experimentos, Fischer propone un modelo de la utilización por parte de los usuarios de una aplicación compleja [Fischer 96], modelo que consideramos muy adecuado para explicar cuál es el papel que debe jugar un sistema de ayuda inteligente. Este modelo, que se muestra en la Figura 3.1, es generalizable a gran parte de los sistemas informáticos.

En el esquema, D_4 representa el sistema real, con el conjunto de funcionalidades (conceptos y órdenes) que están disponibles para el usuario. El subconjunto de conceptos y órdenes que el usuario utiliza de forma regular y sin problemas, está contenido en D_1. Las funcionalidades incluidas en D_2 son el subconjunto que se usa sólo de forma ocasional. El usuario normalmente no conoce todos los detalles o los efectos de su uso (como tampoco otro tipo de efectos secundarios). D_3 representa el modelo mental que posee el usuario sobre el sistema. Este modelo mental es la

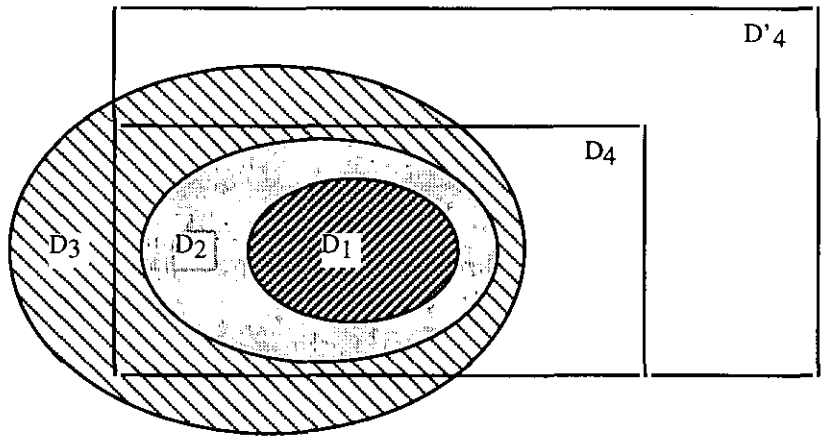


Figura 3.1: Modelo de niveles de utilización de un sistema informático complejo [Fischer 96]

representación interna que tiene el usuario sobre la aplicación, es decir, lo que el usuario cree o piensa que es la aplicación y las funcionalidades que ofrece. Aquí no sólo se incluyen suposiciones correctas sino que también existen otras erróneas. Como muestra la Figura 3.1, la elipse D_3 incluye partes que no están comprendidas dentro del rectángulo D_4. La diferencia entre D_4 y D_3 son las operaciones que el sistema permite pero que son completamente ajenas al usuario.

El aumento en las capacidades y funciones proporcionadas por los entornos informáticos está provocando la evolución hacia una situación representada por D'_4, incrementándose la porción del sistema que es desconocida para el usuario. Facilitar el aprendizaje, acceso y aplicación de las capacidades ofrecidas por el entorno permitirá que suba el porcentaje de las funcionalidades del sistema que se utilicen. También redundará en una mejora de la eficiencia y de la efectividad en la realización de las tareas.

No obstante, el proceso de adquisición de un modelo correcto y completo de la aplicación no es sencillo. Hay partes que como ya están comprendidas dentro del modelo mental del usuario (la intersección entre D_3 y D_4, o entre D_3 y D'_4) serían más sencillas de adquirir aunque el usuario no las haya utilizado previamente. El problema surge con los conceptos de los que no tiene idea ni siquiera de su existencia y que es improbable que los pueda descubrir sin algún tipo de asistencia. Como Fischer plantea, los conceptos ya presentes en el modelo mental pueden adquirirse mediante exploración libre, pero para los otros se tiene que proporcionar ayuda complementaria.

Nuestra propuesta va dirigida a que este apoyo se proporcione mediante un sistema de ayuda inteligente, que se adapte a los usuarios y que presente la información dentro de un contexto que facilite su asimilación y su comprensión. Además, esta asistencia tiene que mostrar otra información relacionada que permita un aprendizaje incremental bajo demanda, de forma que sea el usuario el que según sus necesidades decida aumentar su conocimiento del sistema.

3.4.4 Documentación electrónica y sistemas de ayuda

La base de conocimiento más simple y menos costosa en que pueden basarse los sistemas de ayuda viene dada por la documentación electrónica, por lo que vamos a analizar las características de esa documentación y sus posibles formas de utilización para mejorar el conocimiento que un usuario tiene de una aplicación. Aunque este análisis puede generalizarse a otros contextos, nuestro interés se centra en los documentos electrónicos asociados a las aplicaciones informáticas. La documentación electrónica presenta diversas ventajas sobre el medio impreso que la hacen especialmente adecuada para su utilización con el propósito de ayuda [Duffy 89]:

a) Proporciona una mayor disponibilidad de la información, ya que una única copia puede ser accedida por muchos usuarios, quienes además, con el auge de las redes, no tienen por qué estar próximos a la fuente de información [Buenaga 95, Fernández Manjón 95b].

b) Facilita el acceso, ya que se pueden incluir mecanismos eficientes para la recuperación de la información relevante, que de otro modo habría de buscarse de forma manual.

c) Posibilita una mayor interacción entre el usuario y la información, ya que incluso aunque esta información sea estática, los mecanismos de acceso pueden variarse en función del estado de la aplicación. Por ejemplo, para determinar qué información presentar al usuario, se pueden monitorizar sus acciones para detectar cuál es su interés o cuáles han sido sus errores y proporcionar la información adecuada en cada ocasión.

d) Es más barata de almacenar, producir y actualizar que el medio escrito, que requiere un ciclo de producción más largo. Esta facilidad de actualización también incide en que se disponga de una información más precisa y correcta. En el caso concreto de las aplicaciones software, sucesivas versiones tienen como objetivo mejorar y corregir los errores detectados con posterioridad a la distribución del producto. El elevado coste que tiene modificar la información escrita hace que los manuales existentes sean incompletos o inexactos.

e) Permite la integración de distintos tipos de información multimedia, pudiendo mezclar el texto con dibujos, animaciones o sonidos.

También existen desventajas asociadas con la utilización del medio electrónico como fuente de información. Algunas desventajas son tecnológicas (p.e., en muchos casos no se puede simultanear el uso del sistema de ayuda con el de la aplicación), otras se deben a problemas asociados con los dispositivos físicos (p.e., es más fácil leer un texto impreso que un texto en pantalla). Sin embargo, estos factores son cada vez menos relevantes gracias a la generalización de los entornos multitarea y de los terminales gráficos de altas prestaciones.

Existen otros problemas conceptuales que consideramos más importantes, como por ejemplo la necesidad de adquirir nuevos hábitos de trabajo. Prácticamente todo el

mundo está acostumbrado a consultar libros o manuales, sabiendo cómo usar la tabla de contenidos o el índice de palabras para buscar información. Sin embargo, con la computadora hay que adquirir nuevas destrezas y aprender nuevos modos de interacción.

Es un hecho que la documentación electrónica cada vez tiene una mayor importancia, en particular en el caso de las aplicaciones software, ya que cada vez es más habitual que los productos sólo proporcionen manuales en este formato [MetaCard 95] o que, existiendo documentación impresa, ésta haya que adquirirla como un accesorio más del producto. Lo más frecuente hoy en día es que la documentación electrónica se suministre en CD-ROM, como en el caso, por ejemplo, de la versión Solaris 2.0 del sistema operativo Unix de Sun [Sun 92].

3.4.5 Tipos de sistemas de ayuda: alternativas de diseño

El diseño de un asistente interactivo es un proceso complejo en el que hay que considerar muchas alternativas diferentes y llegar a compromisos entre distintos factores. Como ya hemos comentado, se trata de un campo muy diverso, ya que existen desde sistemas de ayuda que presentan breves mensajes cuando se comete un error, hasta sistemas que proporcionan una completa información sobre la aplicación y que llevan integrado un módulo de enseñanza.

A continuación presentamos una clasificación de los sistemas de ayuda que tiene en cuenta los criterios que tienen una mayor influencia en el diseño. Estos criterios son: la iniciativa de la interacción, el grado de integración entre la ayuda y la aplicación, y el tipo información proporcionada [Tattersal 92, Duffy 92, Kearsley 88].

3.4.5.1 Iniciativa en la interacción: sistemas pasivos y sistemas activos

La distinción entre sistemas de ayuda pasivos y activos se basa en quién es el que toma la iniciativa dentro del proceso de ayuda. En el caso de los sistemas pasivos, es el usuario el que suspende la realización de una tarea para solicitar la ayuda. Por el contrario, en los activos, es el sistema de ayuda el que interviene y supervisa el diálogo entre el usuario y la aplicación, con el fin de detectar problemas, ofrecer consejos y corregir las interacciones potencialmente erróneas del usuario.

Estos dos tipos de ayuda imponen unos requisitos diferentes a la arquitectura del asistente. Los sistemas activos precisan disponer de una información muy completa sobre cuál es el estado de la aplicación y, en consecuencia, requieren una mayor comunicación entre el sistema de ayuda y la aplicación. Para obtener esa información sobre el estado actual existen dos posibilidades: a) comprobar o solicitar a la propia aplicación cuál es su estado, en caso de que ello resulte posible; o b) simular la aplicación. Todas las interacciones del usuario con la aplicación son capturadas también por el sistema de ayuda, quien mediante un programa interno emula cuál es el efecto sobre la aplicación.

Un sistema activo tiene que averiguar cuáles son los objetivos locales del usuario (qué es lo que está tratando de hacer en un determinado momento) para poder analizar sus acciones y así diagnosticar cuáles son las posibles causas de los errores o ineficiencias, lo que le permitirá proporcionar una ayuda realmente útil [Kuah 92]. La utilidad de la ayuda es un aspecto clave, ya que debe ser una información relevante y en el momento oportuno, lo que resulta muy complejo, tal como demuestran los estudios experimentales [Carroll 88]. El entorno completo es muy dinámico, ya que el usuario es un elemento activo que modifica tanto su estado como el de la aplicación. Las acciones de usuario pueden modificar de formas muy variadas el estado del sistema. Este dinamismo hace que la diagnosis aún se complique más, debido a que un consejo sólo es útil en un momento preciso pero no antes o después, ya que con los cambios realizados deja de ser válido [Duffy 92]. Por otra parte, la necesidad de una mayor información sobre el estado de la aplicación limita su aplicación efectiva a los sistemas ya existentes. La simulación completa de la aplicación es un proceso muy costoso, siendo habitual que se produzcan desfases entre la simulación y la aplicación [Breuker 88].

Lógicamente, los sistemas pasivos son más sencillos de desarrollar y se pueden aplicar a cualquier tipo de entorno ya existente. En este caso, es el usuario el que activa el asistente, cuando se encuentra con un problema, y realiza una consulta en la que expresa cuáles son sus objetivos, por lo que no hay necesidad de averiguarlos de otra forma. Esta simplificación también tiene sus inconvenientes, como por ejemplo en el caso de un usuario novel que no sepa qué preguntar, en que términos realizar una pregunta o incluso que no se dé cuenta de que realmente necesita ayuda y continúe realizando las tareas de forma muy ineficiente.

Nuestro interés se centra en los sistemas de ayuda pasivos, debido a factores de coste de producción y de aplicabilidad a entornos ya existentes. Normalmente, los sistemas de ayuda activa se combinan con otros de ayuda pasiva, debido a que no pueden resultar efectivos en todas las situaciones [Breuker 90]. Además de las razones técnicas, también hay otras más conceptuales, como el riesgo de pasar fácilmente de proporcionar una información efectiva a llegar a molestar al usuario con comunicaciones que éste no desea [Carroll 88].

3.4.5.2 Integración del sistema de ayuda y la aplicación

Otra clasificación de los sistemas de ayuda se basa en el grado de integración con la aplicación software a la que proporcionan soporte. De acuerdo con este criterio, podemos encontrar sistemas divorciados, separados e integrados, además de otras situaciones intermedias [Tattersal 92]. Los sistemas divorciados se caracterizan por utilizar una interfaz diferente y no tener información sobre cuál es el estado de la aplicación. Los sistemas separados usan también una interfaz propia, pero habitualmente mantienen información sobre el estado de la aplicación. Los sistemas integrados normalmente se desarrollan de forma conjunta con el resto del software, compartiendo tanto la interfaz como el estado de la aplicación.

Los sistemas de ayuda convencionales suelen estar divorciados de la aplicación, proporcionando, en el mejor de los casos, una información contextual básica, tal como un texto de explicación diferente en función de cuál sea la posición del cursor o

del ratón en la pantalla. Esta ayuda se suele denominar "sensible al contexto", y es muy limitada, ya que lo único que se hace es elegir un texto determinado en función de ciertos datos simples, sin tener en cuenta otros factores como por ejemplo las acciones previas del usuario. Hay sistemas que no tienen suficiente información dinámica sobre el estado del sistema y lo resuelven aplicando sofisticadas técnicas especificas de adquisición de contexto (como un tratamiento del lenguaje natural). Por ejemplo, en el sistema AQUA [Quilici 89] el usuario puede realizar una pregunta en lenguaje natural, pero se le pide que proporcione detalles sobre el contexto en el cual la realiza.

Otra aproximación consiste en crear un entorno de trabajo nuevo y diferente en el que se integra la aplicación final como una subparte del nuevo entorno creado por el sistema de ayuda (la aplicación ya existe y como mucho se modifica ligeramente para integrarla). Un ejemplo es COACH (COgnitive Adaptive Computer Help) [Selker 94], un sistema de ayuda/enseñanza activa del lenguaje de programación LISP. La interfaz del lenguaje de programación LISP aparece como una ventana mas del entorno de COACH. Aquí es el asistente el que ha integrado la aplicación a la que proporciona soporte, modificando su interfaz; no es el sistema de ayuda el que se incluye en el entorno de la aplicación.

La situación ideal viene dada por los sistemas integrados que se desarrollan de forma conjunta con la aplicación, compartiendo tanto la interfaz como el estado. El problema es que si se quiere desarrollar una interfaz para un sistema existente no siempre es posible modificar su código para acceder a la información. Debido a este problema y a las dificultades ya mencionadas de las simulaciones, consideramos que un enfoque general a la vez que factible consiste en una situación intermedia: el desarrollo de un sistema de ayuda separado en el que se obtiene toda la información que sea posible de la aplicación. La falta de una información completa sobre el contexto se puede paliar proporcionando una mayor capacidad de interacción al usuario.

3.4.5.3 Tipo de información proporcionada y tipos de usuarios

Los sistemas de ayuda también se pueden clasificar atendiendo al tipo de información que proporcionan y el tipo de usuarios a los que están dirigidos [Kearsley 88, Duffy 89].

- *Tipo de información.* Una de las clasificaciones típicas de los sistemas de ayuda es por el tipo de información que proporcionan. Entre otros, se distingue entre tutoriales, guías de referencia y manuales de usuario. Esta es una clasificación implícita en función del propósito de la información presentada, teniendo en cuenta si es fundamentalmente instructiva, especificativa, procedimental o explicativa [Duffy 92].
  El problema de esta clasificación es que no existe una correspondencia unívoca entre una necesidad del usuario y un tipo concreto de información. Cuando el usuario se encuentra con un problema, habitualmente para resolverlo necesitará una combinación de estos tipos de información: datos sobre los objetos que intervienen (especificación), datos sobre cómo realizar

una operación concreta (procedimental) e incluso algún tipo de explicación más general (explicativa o instructiva).

- *Tipo de usuario*. Otra distinción usual de los sistemas de ayuda viene dada por el tipo de usuario para el que se diseña. Por ejemplo, se puede diferenciar a los usuarios de acuerdo con su experiencia de uso del sistema. Hay autores como Kearsley que no sólo consideran un tipo genérico de experiencia, sino que distinguen también entre la experiencia en el uso de computadoras, la experiencia con el dominio y la experiencia con la aplicación software concreta. Además, se sugiere que cada uno de los tipos de usuario necesitaría informaciones de ayuda ligeramente diferentes [Kearsley 88].
  Considerar el conocimiento del usuario es un factor clave para la producción de un sistema de ayuda efectivo, pero su inclusión real en el diseño conlleva diversos problemas de implementación [Wasson 92]. Salvo que se realice una simplificación excesiva de los posibles tipos de usuario, desde un punto de vista de coste, es impracticable el desarrollo de una documentación particular para cada uno de ellos. Además, al incluir esos contenidos diferenciados, surge la cuestión de cuál de ellos presentar a cada usuario concreto. En este caso, el sistema debe determinar cuál es el tipo de usuario y presentarle la información apropiada. No es adecuado solicitar al propio usuario que se clasifique él mismo, ya que no sólo se distrae su atención de la resolución del problema que tiene, sino que además puede clasificarse incorrectamente (el usuario puede considerarse novato mientras que el diseñador lo clasificaría como intermedio, o viceversa). Es más, como demuestran estudios experimentales [Draper 84], hay pocos individuos que sean novatos o expertos en todos los aspectos de una aplicación, por lo que habría que contemplar especializaciones para las diversas facetas, pudiendo un mismo usuario requerir diversos tipos de información.

Estos dos criterios (tipo de usuario y tipo de información) están muy relacionados, por lo que deben analizarse de forma conjunta. Normalmente, los distintos tipos de usuario, según su nivel de experiencia y su interés en el uso de la aplicación, requieren distintos tipos de información [Marchionini 95]. Por ejemplo, tomando en cuenta el nivel de experiencia, los usuarios noveles prefieren una información más orientada a la tarea, mientras que los usuarios expertos prefieren acceder a un material de referencia más completo. Una posibilidad es proporcionar información detallada sobre la realización de las tareas concretas. La otra posibilidad es proporcionar una información más genérica sobre el funcionamiento y los procesos involucrados, y que sea el usuario el que obtenga los detalles procedimentales necesarios.

La conclusión es que no existe un único tipo de información que pueda satisfacer todas las necesidades a la vez [Johnson 93]. Por esto, nuestro objetivo es el desarrollo de un sistema que sea capaz de combinar distintos tipos de información. En aplicaciones con muchas funcionalidades (como un sistema operativo), que pueden utilizarse con propósitos muy diferentes, el número de tareas es muy amplio, lo que dificulta la inclusión de información detallada sobre cada una de ellas. Esto nos lleva a integrar distintos tipos de información, dando un papel preponderante a la información de referencia, y a considerar apropiada la

utilización de un modelo de usuario para tratar de adaptar la información presentada al interés y experiencia de la persona que utilice el sistema en cada momento.

3.4.6 Modelos conceptuales de la ayuda

En el estudio de los sistemas de ayuda es necesario entender no sólo cómo realizan los usuarios las tareas con la aplicación, sino también cómo reaccionan a los problemas que se presentan y cómo puede contribuir el sistema de ayuda a que el usuario pueda continuar con su tarea.

Kearsley propone un modelo conceptual para representar el proceso de solicitud o recepción de información de ayuda [Kearsley 88]. En este modelo las situaciones de ayuda surgen siempre a partir de una discrepancia entre el comportamiento esperado y lo que realmente sucede. La discrepancia puede ser detectada por el sistema (ayuda activa) o por parte del usuario (ayuda pasiva). En cualquiera de los dos casos se ha de proporcionar información complementaria que permita resolver la discrepancia. Nuestro estudio se centra en el caso de que es el usuario el que detecta la discrepancia y, por tanto, la necesidad de información se soluciona mediante la utilización de un sistema de ayuda pasiva.

Las funcionalidades que debe tener un sistema de ayuda para producir una mejora en el rendimiento son las siguientes: a) ayuda en la diagnosis, permitiendo al usuario el acceso a la información en función de los problemas que se le presenten; y b) proporcionar la información específica que le de al usuario una o mas estrategias de reparación. Tradicionalmente, una tarea para la que no se proporciona un soporte significativo es la diagnosis de los problemas, que a su vez se ve dificultada por la existencia de mensajes de error crípticos.

Para la solución de un problema con un sistema de ayuda, consideramos especialmente adecuado por su nivel de detalle, el modelo conceptual de tareas propuesto en [Duffy 92]. En la utilización de un sistema de ayuda, un usuario debe realizar las siguientes ocho tareas: 1) identificar el problema; 2) acceder al sistema de ayuda; 3) seleccionar un tema de ayuda; 4) evaluar la información presentada; 5) comprender la información presentada; 6) seleccionar la información necesaria; 7) acceder a otros temas nuevos o relacionados; y 8) transferir la información a la aplicación.

En este modelo, cuando un usuario tiene un problema que no puede resolver, su primera operación es identificarlo, a continuación tratar de formularlo en los términos en los que supone que aparecerá en el sistema de ayuda. A continuación, acceder al asistente y seleccionar el tema de ayuda que considera más adecuado, el que parezca coincidir más con su necesidad. El usuario obtiene una determinada información que debe evaluar para ver si le resulta apropiada, en cuyo caso habrá de comprenderla y extraer los datos relevantes para poder continuar su tarea con la aplicación. Si considera que la información es incompleta, o que no es relevante, podrá acceder a otros temas de ayuda, repitiendo algunos de los pasos anteriores. Si no consigue tener éxito en su búsqueda de información, tendrá que refinar y

reformular su representación del problema y volver a realizar el proceso. El paso final consistirá en transferir la información a la aplicación.

Este modelo conceptual es el utilizado en este trabajo como referencia para el diseño de los sistemas de ayuda. A partir de él se puede analizar el conjunto de funcionalidades que debe proporcionar un sistema de ayuda, el tipo de información a presentar o los métodos de acceso que se proporcionan.

3.4.7 Evaluación de los sistemas de ayuda

Un sistema de ayuda eficaz es aquel que ayuda al usuario a diagnosticar y solucionar su problema con un mínimo coste de tiempo y esfuerzo. Es decir, aquel que provoque una menor interrupción en la tarea del usuario. El proceso completo de evaluación de un sistema de ayuda es muy complejo, ya que hay que considerar un gran número de factores y hacer amplios estudios experimentales con usuarios, por lo que en esta tesis no se realiza una evaluación formal (evaluación sumativa [Bork 87]) de los sistemas de ayuda desarrollados (Argos y Aran, que se presentarán en los Capítulos 5 y 6). A pesar de que la evaluación no se considera en este trabajo, es interesante proporcionar ciertos criterios de evaluación, que junto con el modelo de tareas previamente presentado, influyen en las decisiones de diseño de los asistentes. Además, estos criterios se han tenido en cuenta en las evaluaciones formativas [Bork 87, Duffy 92] realizadas de Argos y Aran.

Kearsley propone que la evaluación de un sistema de ayuda se base en dos factores: su interfaz y su contenido [Kearsley 88]. La evaluación de la interfaz cubre todos los aspectos de la interacción con el usuario, como por ejemplo si se tiene que solicitar la asistencia o de qué forma se proporciona la ayuda. La evaluación del contenido gira en torno al tipo de información que se suministra con el sistema de ayuda, su adecuación y su completitud. La evaluación del sistema debe tener en cuenta no sólo si los usuarios pueden encontrar la información necesaria, sino también lo eficientemente que se pueda encontrar esa información. Además, hay que considerar la calidad de la propia información proporcionada: si está bien redactada, si es comprensible para un amplio rango de usuarios o si es suficientemente precisa.

Duffy *et al.* plantean que en el estudio de la utilidad y efectividad de un sistema de ayuda se deben tener en cuenta los siguientes cuatro aspectos [Duffy 89]:

1) La precisión de la información presentada

2) La completitud de la información presentada

3) La facilidad de acceso a la información presentada

4) La facilidad de comprensión de esa información

Un déficit significativo en alguna de esas características provocaría que el sistema no fuera útil. Por ejemplo, proporcionar una base de datos con una información muy

extensa sirve para bien poco si la información que se proporciona es imprecisa. No obstante, no sólo importa la precisión y la completitud de la ayuda sino que también se debe abundar en su facilidad de comprensión, porque no es muy útil ofrecer un consejo o un texto que determinados grupos de usuarios no pueden entender. El acceso a la información también es clave, debiéndose incluir métodos que simplifiquen ese acceso, ya que el sistema no servirá para mucho si el usuario no puede encontrar la información que precisa o si los datos relevantes están desperdigados por todo el sistema.

3.5 Análisis de tres sistemas de ayuda para aplicaciones software

El campo de los sistemas de ayuda para aplicaciones software es muy amplio, lo que queda reflejado en la existencia de muchos programas que utilizan diferentes aproximaciones [Duffy 92, Kearsley 88]. Como ya se ha mencionado, a partir de los años 80 surgieron los sistemas de ayuda inteligente que se basan en la utilización de técnicas de inteligencia artificial y en la inclusión de modelos explícitos, tanto del dominio como del usuario. En este apartado vamos a describir brevemente tres de los proyectos de investigación más extensos y referenciados en la bibliografía de este campo, como son Unix Consultant [Wilensky 84, Wilensky 89], Eurohelp [Breuker 90] y Sinix Consultant [Hecking 88], proyectos que son especialmente significativos para este trabajo.

Los tres sistemas siguen diferentes enfoques. Por ejemplo, Unix Consultant es un sistema de ayuda pasivo y en el cual la interacción con el usuario se basa únicamente en la comprensión efectiva del lenguaje natural que se utiliza. Los sistemas Eurohelp y Sinix Consultant son sistemas de ayuda que funcionan tanto de forma activa como pasiva y que utilizan simulaciones del sistema operativo para obtener información complementaria para el procesamiento del lenguaje natural y para el reconocimiento de planes. A pesar de sus diferencias, estos sistemas tienen unas características comunes, adoptadas en nuestra propuesta de construcción de sistemas de ayuda, como son:

a) El uso de un modelo explícito del dominio.

b) La inclusión de un modelo de usuario para obtener un comportamiento adaptativo.

c) La aplicación a dominios reales. Concretamente, todos ellos ejemplifican sus prototipos sobre el sistema operativo Unix (o alguna de sus variantes como Sinix).

Otras características, como el procesamiento del lenguaje natural para la obtención de sistemas conversacionales, o la planificación, con inclusión de planes explícitos de realización de tareas, no se contemplan en nuestro modelo de sistema de ayuda (presentado en el siguiente capítulo), fundamentalmente por razones de coste. Nuestro enfoque es diferente, ya que la mejora de la calidad de la ayuda que se

pretende obtener con estas técnicas, en nuestro sistema se logra con la inclusión de un interfaz multimodal que proporciona un mayor control al usuario y que admite distintos tipos de interacción (no únicamente el lenguaje natural, como en los sistemas que describimos aquí).

3.5.1 Unix Consultant

El Unix Consultant (UC) [Wilensky 84, Wilensky 89], desarrollado en la Universidad de Berkeley, ha sido un proyecto pionero en el campo de los sistemas de ayuda inteligente. Su objetivo era la obtención de un asistente que, mediante un interfaz inteligente en lenguaje natural, permitiera a los usuarios, sin experiencia previa, aprender nociones del sistema operativo Unix. Es un sistema de ayuda pasivo, que puede entender preguntas de diversos tipos que los usuarios realizan en inglés mediante un interfaz de línea.

En este proyecto no se planteó como objetivo la obtención de un modelo general para construir sistemas de ayuda. Los puntos claves del desarrollo giran en torno al procesamiento del lenguaje natural, el razonamiento automático y la representación del conocimiento, aplicados a un dominio extenso y complejo. El primer prototipo se construyó a partir de módulos previamente existentes integrados para constituir un asistente informático.

Una característica destacable es que la representación del conocimiento utilizada en todas las componentes del sistema es única. El conocimiento se representa en KODIAK, un lenguaje orientado a la relación, en el que los objetos se consideran elementos atómicos y el conocimiento se captura a partir de las relaciones existentes entre esos objetos. También tiene un componente de modelado de usuario, KNOME [Chin 89], basado en clases de usuarios y en una clasificación de las órdenes de Unix por su nivel de dificultad. Este modelo de usuario ha constituido el punto de partida que hemos utilizado para el desarrollo de nuestro asistente Argos, descrito en un posterior capítulo.

A pesar de no partir de un modelo general de sistema de ayuda o quizá debido a que se plantea como una plataforma de experimentación de técnicas de inteligencia artificial en un dominio real, desde un primer momento se tiene en cuenta la facilidad de modificación y ampliación del sistema. Se puede aumentar la información que posee el sistema, es decir su conocimiento, tanto de tipo lingüístico como respecto del dominio particular de aplicación, el sistema operativo Unix. Esto permite una construcción incremental del sistema, muy adecuada para aplicaciones reales que son dinámicas y, por tanto, exigen la revisión de esa información.

3.5.2 EUROHELP

EUROHELP [Breuker 88, Breuker 90, Winkels 92a, Winkels 92b] es un proyecto que ha sido financiado por la Unión Europea con el objetivo de desarrollar sistemas de ayuda inteligente para aplicaciones informáticas (IPS, Information Processing Systems). Este trabajo ha estado muy influenciado por otros trabajos anteriores del

campo de los tutores inteligentes. Así, se plantea que un sistema de ayuda inteligente puede considerarse como un caso especial de tutor.

Los objetivos principales del proyecto eran la creación de una *shell* y una metodología general de desarrollo de sistemas de ayuda para paquetes software. El núcleo del entorno es la shell, conteniendo conocimiento y procedimientos independientes del dominio, de modo que la tarea del desarrollador de un sistema de ayuda para una aplicación específica consistiría en rellenar esa shell con una representación de los conceptos del dominio (modelo de la aplicación); es decir, cuáles son las órdenes, la sintaxis, los métodos de referencia a los objetos, etcétera, del sistema particular. EUROHELP proporciona la metodología y las herramientas para ayudar al desarrollador a llevar a cabo estas operaciones. Esta metodología se ha plasmado mediante el desarrollo de prototipos de sistemas de ayuda para aplicaciones concretas como, por ejemplo, para el subsistema de correo del sistema operativo Unix (Eurohelp P0).

La producción de la ayuda inteligente se basa en: a) el empleo de técnicas de generación de lenguaje natural, ya que se considera como un prerrequisito imprescindible para cualquier tipo de adaptación; b) la utilización de información contextual sobre el estado de la aplicación (mediante una simulación de la aplicación) y sobre la tarea que está realizando el usuario, para poder determinar cuándo es necesario proporcionar la ayuda, cuál es la necesidad específica del usuario, y cuál es la forma y contenido adecuado para la respuesta generada; y c) el modelado del conocimiento que tiene el usuario sobre la aplicación.

Se trata tanto de un sistema de ayuda activo, pudiendo tomar la iniciativa para proporcionar información no solicitada, como de un sistema de ayuda pasivo al que se le pueden formular preguntas. Para obtener información respecto de la tarea actual del usuario, se capturan e interpretan sus interacciones con el sistema. De esta forma, se intenta determinar cuál es su objetivo y si en algún momento necesita ayuda (aunque el usuario no se haya dado cuenta de que la necesita). Esto se realiza comparando los planes del usuario (reconocimiento de planes) con planes eficientes, de expertos. En la práctica, se utilizan también otras indicaciones de la posible existencia de un problema del usuario, como el intento de ejecutar órdenes que no existen. Esos errores e ineficiencias se diagnostican para decidir qué deficiencia de conocimiento por parte del usuario el lo que provoca el problema. EUROHELP también funciona como un sistema pasivo al cual el usuario puede formular consultas cuando se encuentra con un problema que no sabe resolver. Con la información obtenida mediante los dos modos de operación se mantiene el contexto de trabajo del usuario, lo que permite mantener un diálogo más rico que de otra forma no sería posible. Uno de los resultados del proyecto indica que, debido a la dificultad que implica la simulación y emulación del estado del IPS en cada momento, sería aconsejable el desarrollo simultáneo e integrado del sistema de ayuda y del propio sistema de procesamiento de información.

Una vez identificado el problema del usuario, se planifican las acciones de remedio, que posteriormente se transforman en expresiones en lenguaje natural. Hay dos tipos principales de acciones correctoras: ayuda y enseñanza. El primer objetivo de EUROHELP es que el usuario pueda completar la tarea en la que está involucrado,

para lo que se le proporcionará la ayuda necesaria. La ayuda busca, por tanto, una solución local que resuelva el problema actual de rendimiento. El otro objetivo, a más largo plazo, es el de formar al usuario, mejorando su visión del funcionamiento del IPS (*expansión*). Se sigue una estrategia oportunista, aprovechando las situaciones en las que el usuario necesita ayuda para enseñarle nuevas características del IPS. El aumento y mejora del conocimiento que tiene el usuario del sistema es la mejor forma de evitarle futuros errores.

La información de que dispone el sistema está constituida por dos bases de conocimiento, una que representa el dominio o modelo de aplicación y otra que representa el modelo del usuario. El modelo de aplicación representa los aspectos funcionales y estructurales del IPS de una forma diferenciada. El conocimiento funcional (operacional) viene dado por descomposiciones de las tareas que son las formas normales de realizar operaciones (planes). Esta base de conocimiento está organizada como una jerarquía de planes, en la que se representan tanto los planes del usuario como los planes de interacción relacionados. El conocimiento de soporte contiene fundamentalmente la información relativa a los procedimientos del sistema y a cuáles son sus efectos. El modelo de usuario se deriva del modelo de aplicación y, en gran medida, es un subconjunto de éste. El gestor de este modelo —User Modeller— mantiene la información sobre el uso correcto o incorrecto que hace un usuario particular de los conceptos del dominio y, concretamente, de los procedimientos del sistema. Esta información se utiliza para conseguir una mejor interpretación de las preguntas e interacciones, así como para personalizar las explicaciones y las expansiones de conocimiento que se proporcionan.

Es interesante destacar otros dos objetivos de la investigación de EUROHELP: a) la verificación de si el marco de entrenamiento creado para el sistema de ayuda serviría para un tutor inteligente y b) la generalización de los resultados para obtener una metodología general de construcción de tales tutores. Estos objetivos sólo se han alcanzado parcialmente. Un ejemplo de esos intentos es FysioDisc [Winkels 92a], sistema en el que se han reutilizado los resultados para la creación de un entrenador inteligente en el diagnóstico en fisioterapia.

3.5.3 Sinix Consultant

El Sinix Consultant (SC) [Hecking 88] es un sistema de ayuda inteligente para el sistema operativo Sinix. Es un sistema de ayuda pasivo que proporciona respuestas a preguntas realizadas en lenguaje natural (en alemán) sobre conceptos y órdenes del Sinix. También es un sistema activo que intercepta la comunicación del usuario con el sistema operativo, pudiendo proporcionar consejos al usuario que éste no ha solicitado previamente si detecta que hay operaciones que no se están realizando de forma óptima.

Se ayuda a los usuarios que tienen problemas con el sistema operativo mediante un diálogo cooperativo que adapta la ayuda proporcionada al nivel de experiencia y a los conocimientos de cada usuario concreto. No sólo se explica la orden necesaria para realizar una determinada operación, sino que también se proporciona información sobre los conceptos asociados, que son imprescindibles para la

comprensión de dicha orden. Además, se intenta no fatigar al usuario con explicaciones o ejemplos que no necesita o con información que ya conoce.

Estas funcionalidades se basan en la existencia de una completa base de conocimiento y en un modelado explícito del usuario. En la base de conocimiento se representa la información técnica del dominio, así como la visión y el uso que hace de ella un usuario del sistema operativo. Además de información sobre los conceptos y órdenes existentes, se incluye también información sobre aspectos pragmáticos y tutoriales como, por ejemplo, cuáles son las órdenes relacionadas o los conceptos previos para entender una orden. Una parte de la base de conocimiento del SC, así como de su estructuración y organización se ha reutilizado en nuestro sistema Aran, que se describe en el capítulo seis (una vez traducida al inglés y recodificada en nuestro sistema de representación).

El sistema construye un modelo explícito de cada usuario en el que se representa el conocimiento de ese usuario como un subconjunto de los conceptos de la base de conocimiento. Se utiliza una aproximación basada en cuatro prototipos (novato, principiante, intermedio y experto), obtenidos mediante un estudio experimental, y que proporcionan las suposiciones sobre el conocimiento esperado de un sujeto. Estas suposiciones son importantes al principio de una sesión, cuando todavía no se dispone de información más precisa para obtener un modelo individual del usuario.

3.6 Bases de conocimiento

En los sistemas de ayuda inteligente habitualmente se utiliza una representación explícita del conocimiento que proporciona soporte a otros módulos del sistema [Jones 91]. Por tanto, los asistentes inteligentes, al igual que los tutores inteligentes, se encuadran dentro de los sistemas basados en conocimiento (SBC), y además, como una de sus aplicaciones más complejas [Breuker 90, Winkels 92a]. En los SBC el proceso de la adquisición y representación de la información para que el sistema pueda procesarla es costoso y difícil [Gruber 90, Musen 92]. La situación habitual es que al construir un SBC se tenga que desarrollar su base de conocimiento desde el principio, produciendo un significativo aumento del coste. Además, también se ve limitado el tamaño y la amplitud de la información que se puede incluir en un determinado sistema y, por tanto, la robustez del sistema final.

No obstante, estas circunstancias han empezado a cambiar, ya que se están desarrollando una serie de proyectos para la construcción de bases de conocimiento, de propósito general, que permitirán su utilización en SBC. El objetivo fundamental es disminuir el esfuerzo de desarrollo de este tipo de sistemas mediante la reutilización de la información ya existente en la base de conocimiento, así como la disponibilidad de representaciones ampliamente probadas y sobre dominios reales [Lenat 90].

Resulta interesante considerar tres tipos de proyectos orientados al desarrollo de grandes bases de conocimiento, ya que en un futuro podrían simplificar la producción de asistentes inteligentes como los propuestos en este trabajo (este punto se trata más ampliamente en el epígrafe 4.3.2). El primero, representado por

el proyecto Cyc, se propone desarrollar una base de conocimiento de propósito general con información de todo tipo [Lenat 90, Guha 94]. El segundo, representado por proyectos como WordNet, es más restrictivo, se orienta fundamentalmente al desarrollo de bases de conocimiento léxico, incluyendo información relativa a todos los términos existentes en una lengua, como el Inglés [Miller 93]. El tercero es el proyecto Knowledge Sharing Effort (KSE) del DARPA [Patil 92], que busca facilitar la compartición y comunicación de conocimiento entre diferentes aplicaciones mediante el desarrollo de ontologías (una especificación explícita de la conceptualización de un dominio). A continuación presentamos brevemente estos proyectos.

3.6.1 CYC: una base de conocimiento de propósito general

CYC [Lenat 90, Guha 94, MCC 95] es un proyecto de la corporación MCC que pretendía construir en el transcurso de una década una base de conocimiento a escala real, que capturase un conocimiento similar al contenido en una pequeña enciclopedia. Concretamente, se detallan tres tareas fundamentales en el proyecto [Lenat 90]: i) el desarrollo de un lenguaje para la representación del conocimiento; ii) la creación de un conjunto de procedimientos para la manipulación del conocimiento; y iii) la construcción de la base de conocimiento, en base a los puntos anteriores.

Los aspectos i) y ii) han sido cubiertos mediante la definición de un lenguaje, CycL, que constituye una ampliación de la lógica de predicados [Lenat 90]. En CycL se encuentra representada la base de conocimiento, CycKB, que es lo que resulta de mayor interés para nuestro trabajo. Se trata de construir de forma manual una gran base de conocimiento general que contenga información de todo tipo. Se plantean tanto los problemas relacionados con la inclusión de grandes cantidades de conocimiento, como su uso y aplicación posterior. Por ejemplo, se tienen en cuenta factores como la eficiencia del acceso, la depuración y el refinamiento de la información, así como su coherencia y facilidad de comprensión para los desarrolladores finales. Se han desarrollado herramientas para simplificar esas tareas [MCC 95].

El desarrollo de CycKB, con las dimensiones marcadas en su inicio, parece no encontrarse aún próximo a su fin, por lo que se han modificado parte de los objetivos [Guha 94]. Se va a continuar incluyendo conocimiento de acuerdo con dos criterios: primero, que permita incrementar el uso del lenguaje natural en la introducción de nueva información (se destaca el problema existente con la información introducida en Cyc en 1990 y su desfase cuatro años más tarde); segundo, la inclusión de información necesaria para el desarrollo de aplicaciones concretas.

En el estado actual del proyecto, se considera ya terminada la estructura básica de CYC y se proporciona un acceso libre mediante Internet a una versión restringida que incluye los 3.000 conceptos superiores de la jerarquía y parte de los enlaces entre ellos [CYC 96]. Esta base de conocimiento tiene como objetivo servir como una ontología estándar, de modo que fije el vocabulario básico de conceptos y su

esquema de organización, formando una base para las nuevas aplicaciones a desarrollar. Entre las aplicaciones concretas de la base de conocimiento destacamos dos con las que estamos involucrados en este trabajo, la recuperación conceptual de información y el modelado de usuario. Además, el contenido de la base de conocimiento se puede combinar con el de otras bases más especializadas (para dominios concretos), facilitando así la construcción de sistemas más robustos que los actuales.

3.6.2 WordNet: una base de conocimiento léxico

WordNet es un sistema con información fundamentalmente léxica sobre el idioma Inglés, pero organizada de forma semántica, en lugar de en forma alfabética [Miller 93]. Se ha generado de forma semiautomática, a partir de diccionarios. Originalmente, el desarrollo del sistema partió de un proyecto cuyo objetivo era producir un diccionario, orientado a permitir búsquedas conceptuales en vez de alfabéticas, en línea con las ideas de las teorías psicolingüísticas que consideran que la memoria de las personas es un sistema léxico con herencia [Quillian 67]. Se trata de facilitar el acceso a la información que está contenida en un diccionario normal y que es muy difícil de localizar, debido a la organización exclusivamente alfabética, lo que ha provocado la aparición de diccionarios especializados, tales como los enciclopédicos o los de sinónimos y antónimos.

En WordNet, el objeto básico es el formado por un conjunto de sinónimos estrictos denominado *synset*. Por definición, cada synset en que una palabra aparece es un significado diferente de la palabra. De forma intuitiva podemos hacer corresponder a un synset con un concepto. Para la mayoría de los conceptos se proporciona una definición breve en lenguaje natural y en algunos casos se proporciona también algún ejemplo de su uso. También se almacena información relativa a relaciones definidas entre palabras y synsets, o conceptos. Además de la relación de sinonimia, mediante la cual se forman los synsets, se consideran también relaciones de subordinación (es-un) y de estructuración (tiene-un). El uso de la relación de subordinación permite la construcción y organización semiautomática de los nombres en una jerarquía de conceptos, similar a las definidas en las redes semánticas, y con la importante característica de sus grandes dimensiones. Con los verbos y adjetivos se hace un tratamiento similar. En la actualidad, WordNet es un sistema totalmente desarrollado que incluye una base de datos de 12 Megabytes (tiene 70.100 synsets) en su versión 1.4 [Miller 93]. La profundidad de la jerarquía no es muy grande, pero incluye herencia, de forma que las características diferenciadoras de un concepto se aplican también a todos sus conceptos subordinados.

WordNet, al estar disponible de forma gratuita y contener información muy extensa de propósito general, constituye un buen punto de partida para el desarrollo de un SBC. Además, demuestra la viabilidad de la construcción semiautomática de grandes bases de datos que incluyen una cierta representación del mundo. Sin embargo, debido a su generalidad, para aplicaciones concretas se hace necesario complementar esa información con una particularización del dominio. Por ejemplo, hay términos que por ser muy técnicos no aparecen en un diccionario normal (como

carácter especial), así como otros que en los entornos informáticos se utilizan con un significado distinto o tienen implicaciones diferentes, como por ejemplo dispositivo, que en Unix tiene una correspondencia con archivo debido a que todos los dispositivos se tratan como archivos especiales. Esta complementación especializada de la información que proporciona WordNet es la aproximación que se ha seguido en el sistema SIFT [O'Sullivan 95], un sistema inteligente de recuperación de información en el dominio de los editores de texto.

3.6.3 Knowledge Sharing Effort

El Knowledge Sharing Effort (KSE) del DARPA [Patil 92, Gruber 90, Musen 92] es un consorcio que trata de desarrollar convenios que faciliten la compartición y la reutilización de bases de conocimiento. Trata de definir, desarrollar y probar la infraestructura y tecnología de soporte necesaria para permitir la construcción de SBC mucho mas amplios y con mayor cobertura de la que sería posible alcanzar trabajando de forma independiente. El objetivo es hacer posible que los investigadores puedan seleccionar componentes de una biblioteca de módulos de conocimiento reutilizables e integrarlos para desarrollar nuevos sistemas. Además de crear especificaciones e implementaciones de dominio público para la creación de estas bases de conocimiento y su documentación correspondiente, también se proponen el desarrollo de bibliotecas reusables que demuestren esta aproximación.

Si en lugar de tener que partir de cero para construir un SBC, su base de conocimiento se pudiera crear mediante la combinación de distintos componentes reusables, entonces el desarrollador se podría centrar en la inclusión del conocimiento especializado y de los procesos de tratamiento necesarios para la tarea específica de su sistema. De esta forma, se promueve la compartición de conocimiento, a la vez que su uso de forma extensiva posibilita el refinamiento de los módulos de la biblioteca, permitiendo crear sistemas más extensos y fiables con un coste menor.

Para lograr estos objetivos se han creado cuatro subgrupos de trabajo que se ocupan del desarrollo de: a) mecanismos para la traducción entre bases de conocimiento representadas en diferentes lenguajes; b) versiones comunes de lenguajes y módulos de razonamiento dentro de las familias paradigmáticas de representación; c) protocolos para comunicación entre módulos distintos basados en conocimiento, así como entre sistemas basados en conocimiento y bases de datos; y d) bibliotecas de "ontologías", o modelos básicos de bases de conocimiento de aplicación específica en un determinado área. En este proyecto ya se ha obtenido la especificación e implementación de un lenguaje de representación del conocimiento, KIF (Knowledge Interchange Format) y de traductores a los formalismos más habitualmente utilizados (p.e., a lenguajes de la familia KL-ONE y, concretamente, al lenguaje Loom utilizado en este trabajo).

Como resultado de este proyecto cabe destacar la creación de una biblioteca de bases de conocimiento (ontologías) sobre diversos campos, de libre disposición y accesibles mediante Internet. Una ontología es una especificación explícita de la conceptualización de un dominio. Es una representación formal y declarativa que

incluye el vocabulario para identificar a los elementos existentes en ese dominio, así como las definiciones lógicas que describen qué son esos términos y cómo se pueden relacionar con los otros elementos existentes. De momento, sólo se dispone de representaciones sobre dominios sencillos [Gruber 94], como las magnitudes físicas o las referencias bibliográficas, pero debido a su accesibilidad y a la capacidad de traducción automática a otros formalismos, supone un gran avance hacia la reutilización del conocimiento.

3.7 Resumen

El objetivo de este capítulo ha sido dar una visión general del campo de los sistema de ayuda, y más concretamente de los sistemas de ayuda inteligente, donde se encuadran los asistentes desarrollados en este trabajo. Como dominio nos hemos restringido a los sistema de ayuda para aplicaciones informáticas de uso general.

Hemos realizado, en primer lugar un análisis de la evolución de las aplicaciones informáticas y de las nuevas condiciones que plantea la generalización en su uso. La conclusión de este análisis es la necesidad de proporcionar asistencia a los usuarios para lograr una correcta utilización de las aplicaciones. A continuación hemos estudiado dos enfoques que buscan al mismo tiempo facilitar el aprendizaje y el uso de estos entornos informáticos. Estos enfoques se diferencian en que su énfasis principal es o bien mejorar la interacción (interfaces de manipulación directa y metáforas, agentes, interfaces inteligentes) o bien ayudar/enseñar al usuario (sistemas de ayuda, sistemas de enseñanza). Nuestra propuesta de este trabajo consiste en afirmar que la mejor forma de proporcionar asistencia a los usuarios de aplicaciones es mediante la utilización de sistemas de ayuda inteligente.

Esta propuesta nos ha llevado al estudio de los factores más importantes que intervienen en el diseño y la construcción de sistemas de ayuda. Primero hemos definido qué entendemos por ayuda interactiva y por ayuda inteligente. Consideramos que la ayuda debe permitir al usuario la resolución de los problemas concretos que encuentra en el uso de la aplicación, a la vez que aumenta y completa su conocimiento sobre el entorno. La característica que creemos más destacable de los sistemas de ayuda inteligente, además del uso de técnicas basadas en conocimiento, es la capacidad de adaptación a las circunstancias en las que se solicita la ayuda. Se ha presentado el contexto de estos sistemas de ayuda, destacando cuál es la utilización que hacen los usuarios de estas aplicaciones. Se han analizado las características de esta documentación y sus implicaciones en la ayuda. Posteriormente hemos presentado las principales alternativas de diseño de los sistemas de ayuda, el modelo conceptual del proceso de ayuda y las pautas a seguir para el proceso de evaluación de un asistente.

Finalmente hemos estudiado tres ejemplos de sistemas de ayuda inteligente que, junto con el análisis anterior, constituyen el marco de referencia de nuestro trabajo. Además hemos presentado tres proyectos de creación de bases de conocimiento que, en un futuro, podrían simplificar el proceso de construcción de asistentes inteligentes.

CAPÍTULO 4

EL MODELO DE SISTEMA DE AYUDA PROPUESTO

4.1 Introducción

De acuerdo con lo visto en el capítulo anterior, la producción de sistemas de ayuda (asistentes) es una tarea compleja para la que no existen modelos estándar aplicables a su desarrollo. En el proceso de diseño de un asistente hay que llegar a una serie de compromisos que son determinantes, no sólo de la funcionalidad que ofrece el sistema de ayuda sino también de su coste o esfuerzo de desarrollo. En este capítulo describimos nuestra propuesta de modelo para la construcción de sistemas de ayuda inteligente, que se ejemplificará mediante dos sistemas realizados, Argos y Aran, que se presentarán en los capítulos siguientes.

El objetivo fundamental del modelo que proponemos es la obtención de sistemas de ayuda eficaces, aplicables a dominios reales y que tengan un coste razonable de desarrollo y mantenimiento. El dominio elegido es el de las aplicaciones software de uso general descritas en el capítulo anterior. Para conseguir este objetivo, nuestra propuesta es un modelo para la construcción de sistemas de ayuda pasivos, que proporcionen fundamentalmente información de referencia sobre cada aplicación concreta y que hagan uso de un modelo de usuario que permita adecuar la información presentada a cada usuario particular. Las dos ideas en las que se basa la factibilidad del modelo son la integración en un mismo sistema de diferentes tecnologías y la reutilización de información de distinto tipo. Es decir, con respecto al último punto se propone la creación de asistentes basados en documentación ya existente, para la que se proporcionan nuevos métodos de acceso y una mejor estructuración, mediante la utilización de distintas técnicas.

La ayuda proporcionada tendrá un doble propósito: por una parte, el objetivo de ayuda a corto plazo, que busca una mejora del rendimiento, proporcionando información que permita al usuario continuar con su trabajo; por otra parte, se aprovechan las solicitudes de ayuda para aumentar el conocimiento que tiene el usuario sobre la estructura y funcionamiento de la aplicación. Este último es un objetivo educativo a largo plazo; en nuestra opinión tan importante como el primero, ya que si el usuario del sistema conoce mejor su funcionamiento interno, podrá evitar las situaciones de error o será capaz de solucionarlas por sí mismo.

En este capítulo comenzamos presentando las características y objetivos del modelo propuesto para la construcción de sistemas de ayuda y los dos pilares en que se basa: la integración de tecnologías y la reutilización de información. A continuación presentamos las técnicas que integramos en nuestro modelo: la recuperación de información, el hipertexto, el modelado de usuario y la representación de conocimiento. Finalmente presentamos un breve resumen y las conclusiones. La reutilización de información depende completamente del dominio de aplicación por tanto se trata más ampliamente en los sistemas de ayuda desarrollados Argos y Aran que se presentan en capítulos posteriores.

4.2 Objetivos y características del modelo de sistema de ayuda

El objetivo de este trabajo es la obtención de un modelo que permita la construcción de sistemas (de ayuda inteligente) efectivos y aplicables a entornos reales. El dominio de aplicación de estos asistentes será el de las aplicaciones software.

A continuación presentamos los objetivos fundamentales que hemos definido para nuestro modelo. Seguidamente, se describen las principales características de nuestra propuesta de modelo para obtener un asistente inteligente. Por último, se analiza como nos proponemos obtener los dos propósitos enunciados en nuestra definición de ayuda del capítulo anterior, es decir, cómo se puede lograr una ayuda a corto plazo que mejore el rendimiento y una enseñanza a más largo plazo.

4.2.1 Objetivos del modelo

Debido a las características del dominio de aplicación y a nuestra visión particular del proceso de ayuda, proponemos como objetivos fundamentales del modelo los siguientes:

a) ser aplicable a entornos (dominios) reales, para lo que deberá contemplar la creación de ayuda para programas ya existentes que no se puedan modificar.

b) presentar un comportamiento adaptable al usuario.

c) tener en cuenta aspectos pragmáticos de implementación, tales como el coste de desarrollo y de mantenimiento.

En nuestra opinión, para producir un sistema de ayuda eficaz es crucial que se proporcione un interfaz de usuario amigable (intuitivo y fácil de utilizar) y que permita diversos modos de comunicación hombre-computadora. A diferencia de otros proyectos, como los descritos en [Breuker 90, Hecking 88, Wilensky 89, Bauer 93], no consideramos que sean condiciones imprescindibles de un asistente inteligente las siguientes: a) producir un sistema conversacional mediante procesamiento de lenguaje natural; y b) el uso de planificación con la inclusión de planes explícitos de usuario. En sistemas que contemplan esas condiciones el interfaz se convierte en un aspecto secundario, ya que toda la interacción y

adaptación se basa en la comprensión del lenguaje natural y/o en la determinación correcta de esos planes de usuario (que pueden incluso no existir [Carroll 88]). La mejora en la calidad de la ayuda que se pretende obtener con esas técnicas, se consiguen en nuestro modelo mediante la incorporación de un interfaz multimodal, que proporcione un mayor control al usuario.

4.2.2 Características del modelo de sistema de ayuda propuesto

Teniendo en cuenta los objetivos anteriores, nuestra propuesta de modelo se centra en la construcción de sistemas de ayuda pasivos, basados en el acceso a una documentación de referencia sobre la aplicación en cuestión y que utilicen un modelo explícito del usuario para lograr un comportamiento adaptativo. Además, dedicamos una atención especial al interfaz de usuario, con el fin de conseguir que sea potente a la vez que amigable.

A continuación analizamos cada una de estas características:

- *Sistema activado por el usuario.* En un sistema de ayuda pasivo la interacción tiene que estar promovida por el usuario. Esto podría considerarse una limitación seria, ya que, por ejemplo, podría ocurrir que el usuario no fuera consciente de que tiene problemas o que utilizara métodos muy ineficientes para realizar una determinada tarea. Sin embargo, la alternativa de un sistema de ayuda activo presenta problemas que limitan su aplicación real. Entre ellos cabe destacar: a) el problema de la diagnosis; y b) el problema de la información sobre el estado de la aplicación. La diagnosis, que es el proceso de detección del momento en que el usuario necesita ayuda, implica una captura de la interacción entre el usuario y la aplicación, así como la introducción explícita de planes y objetivos del usuario. En entornos extensos, con gran cantidad de tareas posibles y con distintas formas alternativas de realización de esas tareas, este proceso resulta muy complejo (incluso más que en los tutores inteligentes, ya que con un asistente el usuario realiza su propia tarea y no una tarea impuesta por el sistema). El otro problema consiste en que para que la asistencia activa sea útil, se debe disponer de información detallada sobre el estado interno de la aplicación, como soporte para la diagnosis y para decidir qué ayuda proporcionar. Esto no es siempre posible cuando el programa ya existe y no se puede modificar su código. Por otra parte, la opción de crear una emulación de la aplicación que proporcione esta información de estado se ha descartado porque incrementa mucho el coste de desarrollo. Un sistema de ayuda pasivo, en cambio, no requiere tantos datos sobre el estado de la aplicación, ya que los que se pueden obtener se complementan con los proporcionados por el usuario cuando realiza sus solicitudes. Así, se tiene en cuenta la restricción de que el modelo de sistema de ayuda se pueda utilizar con aplicaciones ya existentes.

- *Ayuda basada en el acceso a la documentación.* El sistema de ayuda que proponemos facilita el acceso a la documentación sobre la aplicación. En lugar de dar información detallada, paso a paso, sobre todas y cada una de las posibles tareas que se pueden realizar, proporciona información de referencia

sobre el programa y su funcionalidad. Una vez que se le proporciona la documentación, el usuario tiene que analizarla para obtener la información que le permita completar su tarea. De este modo, el usuario adquiere y amplía su conocimiento sobre la aplicación, obteniéndose dos beneficios claros. Primero, se logra un objetivo educativo que hará disminuir la necesidad de ayuda (este aspecto se trata más ampliamente en el siguiente apartado). Segundo, aunque en nuestro modelo también se puede incluir información detallada sobre las tareas, al centrarse en información de referencia se simplifica su aplicación a dominios reales. Durante el ciclo de vida del desarrollo de la aplicación se genera mucha información genérica que se puede reutilizar en el asistente; producir una ayuda detallada para todas y cada una de las tareas significaría un esfuerzo mucho mayor. No obstante, para que la documentación pueda utilizarse como ayuda, hay que proporcionar métodos eficientes de acceso y una adecuada estructuración que simplifique su comprensión y asimilación por parte de los usuarios.

- *Adaptación al usuario.* La adaptación del asistente al usuario se realiza por medio de un modelo de usuario basado en las características más significativas de los usuarios que, complementadas con los datos disponibles sobre el estado interno de la aplicación, permitirán conseguir un comportamiento adaptativo. Como en nuestro modelo la principal información que se proporciona es la documentación sobre la aplicación, cuyo contenido es fijo, la adaptación se traduce en una modificación del funcionamiento del asistente. Es decir, se varían las estrategias de recuperación y presentación de documentos para adecuarlas a los intereses y conocimientos de cada usuario concreto.

- *Interfaz multimodal.* En los sistemas de ayuda, el interfaz juega un papel fundamental en la aceptación por parte de los usuarios y en la facilidad de uso. Independientemente de la sofisticación de las técnicas incluidas en un asistente, no es realista considerar que pueda solucionar todas las solicitudes de los usuarios y sin embargo seguir siendo útil. En nuestro caso las posibles deficiencias en la respuesta del sistema de ayuda se tratan de paliar proporcionando un interfaz multimodal, que permita una mayor capacidad de interacción y en el que el usuario siempre tiene la iniciativa. De esta forma, cuando no obtiene la respuesta deseada, el usuario siempre tiene la opción de reformular la solicitud o elegir otro modo de interacción alternativo.

4.2.3 Ayuda y aprendizaje

Como solución a los problemas con los entornos infomáticos estudiados en el capítulo anterior (p.e. su creciente complejidad o los nuevos tipos de usuarios) nuestra propuesta es proporcionar asistentes inteligentes que ayuden a superar problemas concretos, que radiquen en el uso de las aplicaciones [Fernández Manjón 95c]. Nuestro enfoque de la asistencia (presentado en el epígrafe 3.4.1) consiste en que los sistemas de ayuda deben proporcionar la información que permita al usuario completar la tarea en curso, a la vez que promueven una mayor comprensión de la estructura del sistema [Fernández Manjón 94]. El asistente

tiene, por tanto, un doble objetivo; por una parte un objetivo a corto plazo que persigue una mejora del rendimiento, a la vez que se aprovechan las solicitudes de ayuda por parte del usuario para instruirle de modo que aumente su conocimiento sobre el dominio (la aplicación). Esto último es un objetivo a largo plazo denominado por algunos autores como *enseñanza oportunista* [Breuker 90].

En un sistema de ayuda que siga las directrices establecidas en nuestro modelo la mejora en el rendimiento se obtiene proporcionando unos métodos eficientes de acceso a la información. Esto posibilita que el usuario obtenga la información necesaria para resolver su problema con la menor interrupción posible de su tarea principal. La consecución de este objetivo se plasma en una o varias indexaciones (organizaciones) de la información y las interfaces correspondientes que se proporcionan a partir de ellas. Por ejemplo, la documentación electrónica sobre las utilidades del sistema se puede indexar utilizando técnicas de recuperación de información con el modelo del espacio vectorial (como se hace en Argos, descrito en el capítulo 5), o también indexarla en función de un conjunto de descriptores controlados, que dan idea de su funcionalidad, y de acuerdo con el modelo del dominio (como se hace en Aran, descrito en el capítulo 6). Las diferentes indexaciones permiten al usuario acceder a la información de distintas formas. Así, puede expresar su necesidad de ayuda en lenguaje natural, o seleccionar los descriptores que mejor definen su petición o simplemente realizar una exploración de la base de conocimiento donde la documentación se encuentra agrupada en torno a los conceptos más significativos del dominio.

El objetivo educativo se concreta en proporcionar al usuario una visión completa del sistema y no únicamente en ayudar a resolver problemas concretos. Se persigue que el usuario adquiera un modelo adecuado del sistema, ya que, como han demostrado diversos estudios experimentales, el aprendizaje mejora también el rendimiento a corto plazo [Marchionini 91, Jerram-Smith 89]. La consecución de este propósito se obtiene mediante la interrelación de la información y su contextualización, de modo que también se simplifica su comprensión. Cuando se le muestra documentación al usuario siempre se presenta dentro de contexto mostrando, por ejemplo, las relaciones existentes con otros documentos o mostrando los conceptos implicados. De este modo se mejora la comprensibilidad y la asimilación de la filosofía del sistema. La pieza clave de nuestro modelo para aumentar la comprensión de la estructura y operación del sistema es la base de conocimiento. Por ejemplo, en el sistema Aran (capítulo 6) esta base contiene un modelo explícito del sistema operativo Unix, directamente inspeccionable desde el interfaz, posibilitando así que el usuario aprenda los términos que se utilizan, cuál es su significado preciso, cuáles son los conceptos que están relacionados con uno determinado o cuáles son las utilidades que realizan una determinada operación. Para mejorar el contexto en el que se presenta la información, desde los otros modos de interacción, lenguaje natural o descriptores, se puede acceder directamente al modelo del dominio y ver cuáles son los conceptos que indexan la documentación (conceptos relacionados).

Por tanto, mediante el asistente se trata tanto de facilitar el acceso por parte del usuario a la información, como modo de resolver problemas (ayuda), proporcionando un aprendizaje bajo demanda, es decir, cuando el usuario la necesita [Fischer 91].

Es decir, en nuestro sistema se integra sin diferenciación el aprendizaje con el propio trabajo [Carroll 88].

4.3 Fundamentos para la viabilidad de la implementación del modelo

La factibilidad de nuestra aproximación para los dominios reales nos lleva a cuestionar dos aspectos. En primer lugar, la especificidad de las técnicas utilizadas hace que la mayoría de los sistemas de ayuda que hacen uso de técnicas de inteligencia artificial sean muy caros de construir y mantener, lo que impide su generalización [Kearsley 88]. En segundo lugar, esas mismas circunstancias hacen que la mayor parte de los sistemas desarrollados no abandonen nunca su condición de prototipo, o bien que sólo sean utilizados en dominios muy restringidos.

Las bases sobre las que se fundamenta la viabilidad de implementación de nuestro modelo son dos:

a) La integración en el mismo sistema de diferentes tecnologías bien conocidas y ampliamente probadas en distintas áreas. En nuestro caso estas técnicas son: la recuperación de información, el hipertexto, el modelado de usuario y la representación explícita del conocimiento.

b) La reutilización de diferentes tipos de información. Esto implica desde la reutilización de la documentación electrónica de ayuda existente, hasta la reutilización parcial de bases de conocimiento o de modelos de usuario usados previamente en otros sistemas en este u otro dominio.

En los dos siguientes epígrafes se analizan estos dos aspectos con más profundidad. Hay que destacar que la integración de técnicas en un mismo sistema es independiente del dominio concreto de aplicación, por lo que en puntos posteriores se realizará una exposición más extensa y detallada sobre estas tecnologías. Sin embargo, en este tema no se realizará una exposición detallada sobre las distintas clases de reutilización de información ya que depende completamente del dominio para el que se desarrolla el sistema de ayuda. Los detalles sobre la información reutilizada se proporcionan en los capítulos 5 y 6 donde se describen los dos asistentes Argos y Aran desarrollados en esta tesis.

4.3.1 Integración de tecnologías

La situación habitual es que los desarrolladores solucionen el problema de proporcionar ayuda mediante la utilización de técnicas *ad hoc* para esa aplicación y dominio [Breuker 92]. Esto implica mayores dificultades cuando se quiere aplicar estas soluciones en áreas diferentes (cambio de dominio), y cuando se quiere modificar, refinar o ampliar su contenido (actualización y escalabilidad). En el modelo que proponemos en este trabajo se ofrecen al usuario una serie de funcionalidades de ayuda mediante la combinación en un mismo sistema de técnicas bien conocidas y ampliamente probadas en diversos campos. Como ya

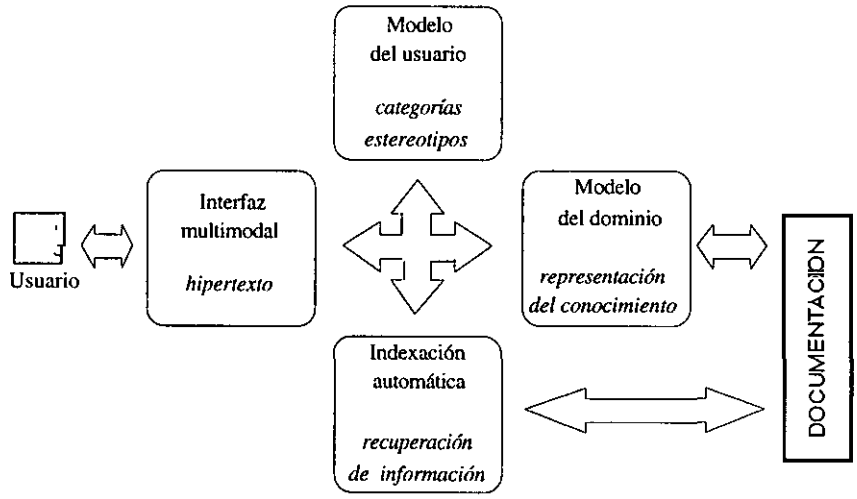


Figura 4.1: Esquema del modelo de sistema de ayuda propuesto.

hemos mencionado, estas tecnologías son: recuperación de información, hipertexto, modelado de usuario y representación explícita del conocimiento.

- *Recuperación de información.* La funcionalidad básica de un sistema de ayuda es la de proporcionar la información que permita al usuario continuar con su trabajo. La recuperación de información (RI), permite recuperar documentos determinando cuáles son los relevantes de entre un gran corpus de textos, a partir de una determinada consulta [Salton 89, Croft 93]. Esta técnica hace posible que el usuario exprese su necesidad con un lenguaje que le resulte sencillo, como por ejemplo mediante el uso del lenguaje natural, especificando la información que desea obtener sin tener que aprender algún otro método formal de petición. Si además se realiza un tratamiento de los textos en función de un vocabulario controlado (palabras clave o descriptores), se puede soslayar un problema típico de los sistemas de ayuda, como es el que el usuario deba utilizar un vocabulario ajeno al dominio de aplicación [Callan 93, Frakes 94]. Las técnicas de RI no permiten la construcción de sistemas conversacionales, que busquen una comprensión profunda de lo que se expresa, como la realizada en el proyecto Unix Consultant [Wilensky 89], ya que para esto hacen falta unas extensas bases de conocimiento. No obstante, además de facilitar las consultas, la RI presenta otra serie de ventajas, como ser una técnica robusta que se puede aplicar a grandes conjuntos de documentación, permitiendo su uso en dominios reales con un coste bajo, siendo directamente trasladable de un campo de aplicación a otro distinto y haciendo que el proceso de inclusión de nueva información resulte sencillo y automatizable.

- *Modelado de usuario.* Un sistema de ayuda inteligente debe presentar, en general, un comportamiento adaptativo, adecuando la información que

proporcione al interés y el conocimiento del usuario [Boy 91]. Para conseguir este objetivo de interacción "inteligente", de modo que el sistema sea capaz de exhibir un comportamiento cooperativo de amplio rango, es frecuente el uso del modelado de usuario [Finin 89, Kass 91]. De hecho, hay autores que consideran que este modelado es un prerrequisito inevitable para lograr esa cooperación [Wahlster 89]. El modelo debe contener las suposiciones explícitas sobre todos los aspectos del usuario que esté en ese momento interactuando con el sistema y que además sean relevantes para adaptar el comportamiento del sistema interactivo al usuario [Kobsa 94]. No obstante, no se pretende simular los cambios cognitivos del usuario sino simplemente obtener información que permita, por ejemplo, clasificar al usuario según su experiencia, de modo que se pueda realizar la adaptación. La adecuación al usuario puede plasmarse de modos diferentes como, por ejemplo, mediante la generación *on line* [Breuker 90, Jonhson 94, Wilensky 89] o la modificación de la información presentada en función de los datos que se tienen sobre la tarea y el utilizador [Boyle 94]. En nuestra aproximación, en la que se trabaja con documentación fija y primordialmente ya existente, la adaptación se traduce en la utilización del modelo de usuario para implementar estrategias de presentación de información como, por ejemplo, recomendar cuáles son los documentos que se consideran más interesantes en cada momento.

- *Hipertexto.* En el diseño del interfaz es preciso tener muy presentes los criterios de una adecuada interacción hombre-máquina [Downton 91], de forma que se obtenga un interfaz gráfico, que además de ser versátil y sencillo de utilizar, permita distintos modos de operación. Se han realizado estudios experimentales que demuestran que la aceptación de un sistema de ayuda por parte de los usuarios depende en gran medida de su interfaz [Jerram-Smith 89]. En la interacción con los usuarios hemos decidido emplear técnicas de hipertexto debido a dos razones: su facilidad de uso que no requiere unos grandes conocimientos previos para su manejo y su adecuación para presentar información sobre dominios amplios y diversos [Shneiderman 89a, Jonassen 90c]. Por ejemplo, permite presentar información de forma incremental y con el control del usuario. Como un asistente no puede ser efectivo en todas las situaciones, se trata de proporcionar al usuario un método eficaz de interacción en el cual tiene la iniciativa.
  Esta combinación de hipertexto con otras técnicas como, por ejemplo, la recuperación de información, ha demostrado ser muy efectiva y con gran aceptación por parte de los usuarios [Jonhson 93]. Algunos autores consideran que esta integración es la forma de obtener asistentes efectivos:

  "We believe that the most effective learning systems we are capable of designing today would have the advantages both of hypertext and of query-based exploratory environments." [Mayes 90]

- *Representación explícita del conocimiento.* En los asistentes inteligentes es habitual encontrar una representación explícita del conocimiento que proporcione un soporte para otros módulos del sistema, facilitando por ejemplo la generación de respuestas en lenguaje natural [Hecking 88] o las

estrategias de presentación de información [Winkels 92a, Breuker 90]. En nuestro caso, uno de los objetivos de la representación explícita del dominio es la estructuración y organización de la información. Esta representación normalmente se realiza mediante métodos y formalismos específicos [Selker 92] que son difícilmente trasladables de dominio y presentan inconvenientes de escalabilidad. Para evitar los problemas derivados de la especificidad de los métodos de representación, hemos optado por el uso de un entorno estándar de construcción de sistemas inteligentes, concretamente Loom [MacGregor 91]. Loom es un lenguaje de representación basado en lógicas descriptivas que, al incluir un clasificador automático, facilita la creación de la base de conocimiento, así como su posterior depuración y ampliación. Su formalismo pertenece a la familia de lenguajes de representación descendientes de KL-ONE, por lo que, además de ser ampliamente utilizado [Wright 93, Heinshon 94], existen programas que permiten la traducción automática a o desde otras representaciones con similar capacidad expresiva [Gruber 90, Gruber 92].

4.3.2 Reutilización de la información

En el desarrollo de sistemas de ayuda, un aspecto clave, además del contenido, es el modelo seguido en los diferentes módulos del sistema. Es decir, además de la información que se ha de proporcionar, resulta determinante cuál sea el modelo del dominio que se utilice, qué factores se tengan en cuenta en el desarrollo o cuál sea el contenido del modelo de usuario que permita un comportamiento adaptativo.

El planteamiento típico en el desarrollo de los sistemas de ayuda consiste en crear "desde cero" los contenidos y en utilizar modelos propios y específicos para cada aplicación [Wilenski 89, Selker 92, Breuker 92]. En este trabajo planteamos la reutilización, a distintos niveles, de los resultados obtenidos en desarrollos anteriores. De esta forma, se intenta aprovechar la documentación previamente producida, usar modelos anteriormente probados y con los que se tiene experiencia, y aprovechar las bases de conocimiento construidas para otros sistemas.

- *Reutilización de documentación*. La forma más básica de reutilización la tenemos en el aprovechamiento de la información textual que se ha generado durante el desarrollo de la aplicación software. Dado que se dispone de una información muy completa, que incluso tiene en cuenta distintos puntos de vista e intereses respecto del software, de lo que se trata es de reusar esa documentación para proporcionar la ayuda. Para lograr este propósito se han de proporcionar métodos efectivos de acceso y una estructuración que facilite su comprensión a los usuarios.

- *Reutilización de modelos de usuario*. Para exhibir un comportamiento adaptativo en la interacción, es habitual que los programas tengan en cuenta ciertos aspectos del usuario. Si la información sobre el usuario se representa de forma explícita y tiene asociados unos ciertos mecanismos de adquisición y mantenimiento, entonces se considera que constituye un modelo de usuario

[Whalster 89]. Como ya se ha tratado para el modelo de alumno de los tutores inteligentes, la obtención de un modelo completamente acertado es un objetivo demasiado ambicioso. Por tanto, se buscan modelos más prácticos, de adquisición rápida y mantenimiento simple, pero que a la vez sean razonablemente eficientes. Las condiciones particulares de la ayuda aconsejan además que su adquisición se realice de forma automática, sin que el usuario tenga que proporcionar información de forma expresa, y que el modelo tenga algún poder predictivo [Chin 93]. En ciertos sistemas, la recopilación de datos sobre el usuario se realiza creando un modelo propio y exclusivo para la aplicación [Selker 92, Kaplan 93]. En este trabajo, por el contrario, se utilizan como punto de partida enfoques de modelado previamente probados con éxito, complementándolos y adaptándolos a nuestras necesidades específicas. Concretamente, nos centramos en modelos basados en clases de usuarios y en estereotipos [Rich 89, Kobsa 93, Chin 89]. Este planteamiento se verá facilitado por la generalización de las *shells* de modelado de usuario de propósito general, que incorporan la noción de estereotipo [Kobsa 95, Kay 95, Finin 89]. En ellas, el desarrollador sólo necesita especificar las condiciones del modelo y es el propio entorno el que se encarga de proporcionar una serie de funcionalidades al sistema, como por ejemplo, las de mantener la consistencia del modelo, clasificar a un usuario dentro de alguna de las clases definidas o proporcionar suposiciones sobre los intereses o el nivel de conocimiento del usuario.

- *Reutilización de conocimiento.* La creación de la base de conocimiento (BC) es la tarea más costosa del proceso de desarrollo de un sistema inteligente [Gruber 90]. Si se parte desde cero en la construcción de esta base de conocimiento, no sólo aumenta el coste, sino que también se restringe su cobertura y robustez, siendo difícilmente aplicable a dominios extensos. El hecho de que esta tarea sea tan laboriosa hace necesaria la búsqueda de nuevos métodos que faciliten el desarrollo de sistemas inteligentes aprovechando esfuerzos de codificación de conocimiento anteriores. La solución se encuentra en la reutilización del conocimiento. Se pueden distinguir diferentes niveles de reutilización [Musen 92]. Podemos utilizar directamente bases de conocimiento previamente desarrolladas (o partes de las mismas), podemos modificar o ampliar una determinada BC desarrollada quizá con algún otro propósito, podemos simplemente cambiar el formato de representación, etcétera. Cada nivel de reutilización conlleva un grado distinto de factibilidad y dificultad; por ejemplo, actualmente es improbable que se pueda encontrar disponible una base de conocimiento completa para un determinado dominio. La situación más frecuente es que se puedan reutilizar sólo algunas partes del conocimiento y que, en la mayoría de los casos, haya que traducirlo a una nueva representación. La elección de un formalismo como Loom [MacGregor 91] simplifica en cierta medida estos problemas; es de amplia utilización en distintos entornos [Wright 93], tiene un gran poder expresivo que facilita la traducción y es uno de los lenguajes considerados en el Knowledge Shared Effort de DARPA [Patil 92, Fikes 92], con lo que se tendrá acceso a las bases de conocimiento desarrolladas en dicho proyecto.

Como ya hemos mencionado previamente, la descripción detallada de la información reutilizada en nuestros sistemas Argos y Aran se realiza en los temas 5 y 6. No obstante, cabe destacar que además de la característica de la dependencia de dominio de la información, la reutilización de información, en cuanto a modelos y a bases de conocimiento, plantea un mayor grado de dificultad que la integración de tecnologías. Proporcionar una solución completa y general al proceso de reutilización de información es un propósito muy amplio y ambicioso que no se considerado como uno de los objetivos del trabajo desarrollado en esta tesis.

4.4 Integración de tecnologías para la construcción de sistemas de ayuda

Una de las bases de nuestra propuesta para la construcción de sistemas de ayuda es la integración de tecnologías. A continuación describimos brevemente estas tecnologías, destacando las técnicas que consideramos más relevantes para nuestro trabajo. Los detalles de estas técnicas se concretarán en capítulos posteriores, al presentar los dos sistemas implementados: Argos y Aran.

4.4.1 Recuperación de información

La recuperación de información (RI) es la técnica que tiene que ver con la representación, almacenamiento, organización y acceso a elementos de información [Salton 83]. En principio, no se impone ninguna restricción sobre el tipo de la información a tratar, pero en este trabajo nos centramos de forma más precisa en el tratamiento automático de documentación textual. Por tanto, una definición de recuperación de información más próxima a nuestro planteamiento es la de técnica (o conjunto de técnicas) que proporciona un acceso efectivo a grandes bases de datos cuya información es fundamentalmente textual [Croft 93]. La tarea principal de un sistema de RI es la de responder a una solicitud de información realizada por un usuario proporcionando un conjunto de documentos relevantes para dicha petición [Callan 93]. Este también es uno de los objetivos de los sistemas de ayuda basados en el acceso a documentación, por lo que la RI resulta una técnica muy adecuada para la implementación de sistemas de ayuda [Fernández Manjón 95c, Buenaga 95].

Las técnicas de RI no son excesivamente recientes, pudiéndose encontrar a lo largo de la historia de la Informática diversos sistemas de clasificación desarrollados para los documentos de algunas bibliotecas. Sin embargo, ha sido en las dos últimas décadas cuando se han dado las circunstancias que han hecho que estas técnicas cobren un inusitado auge. La cantidad de información disponible ha ido creciendo a un ritmo exponencial. Ese crecimiento ha provocado problemas de organización y de recuperación de la información. Como ejemplo, ya en los años 80 la Biblioteca del Congreso de los EE.UU. llegó a tener que incluir alrededor de 3.500 nuevos documentos cada día, complicando extremadamente la gestión en general y, de forma especial, la recuperación de aquellos documentos relacionados con un determinado tema. Ese volumen de información, unido al aumento de la capacidad de almacenamiento de los sistemas informáticos y a la generalización del uso de las

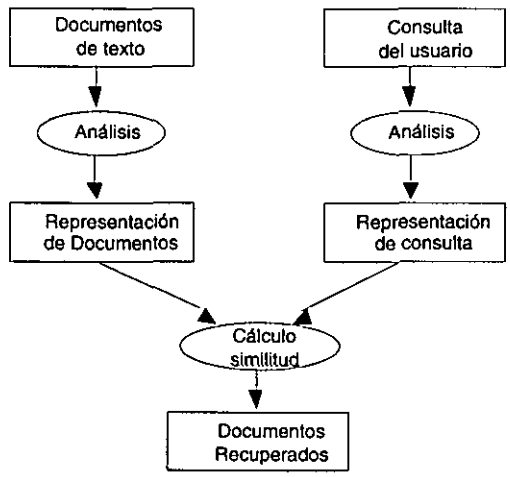


Figura 4.2: El proceso de recuperación de información.

redes de computadoras, han hecho que actualmente exista un gran número de documentos completos en formato electrónico y disponibles de forma inmediata. Por ejemplo, con el impulso de la red Internet, hoy en día es habitual que muchas universidades proporcionen acceso a sus informes técnicos [Fernández Manjón 95b]. Este nuevo panorama hace que cada vez sea más necesario disponer de métodos de tratamiento automático de los textos, permitiendo su caracterización, indexación y posterior recuperación a partir de una solicitud formulada por un usuario. La RI se integra en nuestro modelo de sistema de ayuda ya que esta problemática es similar a la de un usuario que desea encontrar, dentro de la extensa documentación existente en una aplicación, aquella que le permita resolver su problema y continuar con su trabajo.

A grandes rasgos, podemos señalar tres procesos básicos (Fig. 4.2) que están presentes en los sistemas de RI [Croft 93]: a) representación de la necesidad de información del usuario; b) representación del contenido de los documentos; y c) comparación de ambas representaciones para decidir qué documentos se han de recuperar. A continuación trataremos estos tres procesos, incidiendo en los métodos de tratamiento automático de los documentos y en los tipos de procesamiento de las consultas que permiten. Concretamente en la representación basada en términos o palabras clave, que propone la representación del contenido de un documento (o consulta) mediante una lista de términos o descriptores. La indexación basada en términos se puede hacer mediante vocabulario controlado o vocabulario no controlado. Un análisis de otros enfoques diferentes se puede encontrar en [Salton 83, Frakes 92, Frakes 94, Callan 93].

4.4.1.1 El modelo del espacio vectorial

En la bibliografía se pueden encontrar diferentes modelos orientados a la implementación de sistemas de RI, basándose la mayoría en criterios estadísticos. En este trabajo para la representación mediante vocabulario no controlado

utilizamos el modelo del espacio vectorial [Salton 86], que aúna la efectividad con la sencillez de implementación. Como expresa Callan,

> *"Text representation, or indexing, has been one of the major foci of research in IR. One major result is that simple word-based representation, when combined with appropriate retrieval models, are surprisingly effective as well as being efficient and straightforward to implement" [Callan 93]*

En este modelo tanto los documentos como las solicitudes se representan como vectores de descriptores (términos), que se obtienen automáticamente del propio texto. Si m es el número total de términos relevantes de toda la colección, entonces un documento D_i se representa como:

$$D_i = \langle d_{i1}, d_{i2}, \cdots, d_{im} \rangle$$

donde d_{ij} es el peso o importancia del término T_j para el documento Di. Las solicitudes también se representan mediante vectores:

$$Q = \langle q_1, q_2, \cdots, q_m \rangle$$

El proceso de recuperación requiere la existencia de un mecanismo para determinar la proximidad entre un documento y una solicitud (*función de similitud*). Para esta representación se pueden definir funciones de similitud variadas [Salton 83, Salton 89]. En este trabajo hemos elegido una función de similitud basada en el coseno del ángulo formado por los dos vectores. Esta función ha sido ampliamente utilizada y, en término medio, proporciona buenos resultados [Araya 90].

Este modelo del espacio vectorial presenta distintas ventajas sobre otras propuestas (como, por ejemplo, el modelo booleano). Entre ellas, cabe destacar que: a) la consulta se puede construir de forma automática a partir de la necesidad de información del usuario expresada en lenguaje natural; b) se le puede asignar un peso a los términos para reflejar su importancia relativa; c) la aplicación de una función de similitud permite la ordenación de los resultados, controlando su tamaño y recuperando documentos que sólo coinciden parcialmente con la solicitud [Araya 90, Salton 89].

No obstante, incluso disponiendo de una representación adecuada de los documentos y de las consultas, así como de una buena función de similitud, normalmente no todos los documentos que se recuperan contienen información relevante para la necesidad del usuario. Con el fin de evaluar la efectividad del proceso de recuperación de información, resulta habitual utilizar dos medidas clásicas: la *precisión* y el *recall*. El *recall* da cuenta de la proporción de material que se recupera del conjunto total de documentos relevantes, mientras que la precisión es la proporción del material recuperado que es pertinente a la petición del usuario. En la práctica se ha comprobado que estos dos índices varían inversamente, de modo que es difícil recuperar todos los documentos relevantes y a la vez no obtener otra información no deseada [Salton 86].

4.4.1.2 Representación mediante vocabulario controlado

La representación mediante vocabulario controlado es similar a la descrita en el modelo vectorial, pero con un conjunto de términos fijo (descriptores), que son los únicos que pueden ser utilizados para la representación de los documentos y para la especificación de las consultas [Salton 83].

El primer paso para su aplicación es obtener el conjunto capaz de especificar cualquier documento de ese dominio [Salton 89]. Tanto el proceso de la obtención de este conjunto de descriptores, como el de la representación de los documentos, son susceptibles de ser realizados de forma automática a partir del análisis de los propios textos. No obstante, hemos decidido realizar estos procesos de modo semiautomático, haciendo un filtrado de los descriptores de significado similar, para lograr un conjunto más reducido. Esto simplificará la interacción con el usuario, al que se le mostrará la lista de descriptores a partir de los cuales puede formular su solicitud. Además se ha considerado que todos los descriptores tienen el mismo peso y que un documento es relevante si aparecen en su representación todos los términos de la consulta.

Con estas decisiones se obtienen dos ventajas: a) se realiza un proceso semiautomático de indexación, con un coste bajo por lo que es aplicable a grandes colecciones de documentos; y b) se proporciona un método simple y rápido, a la vez que potente y bien fundamentado, para la formulación de peticiones. Como inconvenientes de esta aproximación cabe destacar que no se pueden ordenar por relevancia (se obtienen valores de relevancia 0 ó 1), ni por tanto fijar un umbral para controlar la cantidad de documentos que se recuperan.

4.4.1.3 Mecanismos de mejora de la efectividad del proceso de recuperación de información

Experimentalmente se ha constatado que, con frecuencia, los usuarios tienen dificultades para utilizar eficazmente los sistemas de RI [Araya 90]. Esta dificultad haría disminuir la adecuación de esta técnica para la construcción de sistemas de ayuda, por tanto se incluyen diversos mecanismos correctores.

Para paliar parte de los problemas de los usuarios en el tratamiento mediante el modelo vectorial, además de permitir que la formulación de las consultas se pueda realizar en lenguaje natural, se han venido utilizado diversos mecanismos complementarios. El objetivo es conseguir una mayor efectividad del proceso global de recuperación, simplificando la formulación de las solicitudes y mejorando la representación de los documentos y las consultas. Los mecanismos que consideramos en este trabajo son [Frakes 92, Fox 92]:

- *Lista de parada* (*stoplist*). Podemos detectar un conjunto de palabras que, teniendo una alta frecuencia de aparición, no tienen un contenido semántico que resulte de alguna utilidad para la caracterización de un documento o de una consulta. Por tanto, podemos eliminarlas del conjunto de descriptores sin que perdamos información relevante; utilizamos una lista de "palabras

vacías" (tales como "and", "or", "but" u "of"), lista que particularizaremos para el dominio de aplicación concreto.

- *Eliminación de sufijos (stemming).* Disponemos también de una serie de algoritmos de eliminación de sufijos o de extracción de raíces que simplifican los términos mediante la eliminación de las variaciones morfológicas en las palabras con un significado común. Esos algoritmos suprimen las terminaciones correspondientes a plurales, gerundios, formas verbales, etc., de las palabras para conseguir *términos* unificadores.

- *Pesos de términos.* A los términos que se utilizan para representar los documentos y las consultas podemos asignarles un peso que refleje su importancia relativa dentro del *corpus* (conjunto de documentos). Para calcular dicho peso, se tiene en cuenta el número de apariciones en cada documento y el número total de apariciones en el corpus. Así, podemos primar los términos con una frecuencia media de aparición, que son, según demuestran estudios experimentales, los más adecuados para caracterizar los documentos. De esta forma, evitaremos dar una importancia excesiva a los términos que aparecen en pocos documentos o a los que aparecen muchas veces [Salton 88].

No obstante, se pueden destacar dos factores como los más importantes que provocan estos problemas: a) la dificultad de formular la solicitud de una forma que sea correcta y se corresponda con la necesidad de información; y b) la falta de conocimiento sobre el vocabulario específico usado para describir o indexar los documentos. Estas dos circunstancias se complican por el hecho de que, en muchos casos, el usuario no tiene una idea clara de lo que está buscando (aunque, por otra parte, sí es capaz de reconocer fácilmente la información que busca). Para solucionar estos dos problemas en el modelo vectorial se aplica la realimentación por relevancia, y en la indexación con vocabulario controlado el sistema muestra cuáles son los posibles descriptores.

- *Realimentación por relevancia.* Una vez completada una búsqueda, el usuario es capaz de determinar cuáles de los documentos recuperados le son relevantes y cuáles no. Podemos utilizar esa información para realizar una reformulación o una expansión de la consulta. Este proceso se denomina realimentación por relevancia.
  En su forma general, la modificación de la consulta se realiza de acuerdo con la siguiente fórmula:

$$Q_{nueva} = \alpha \cdot Q_{anterior} + \beta \cdot \sum_{Ri \in Rel} Ri + \gamma \cdot \sum_{Nj \in Nrel} Nj$$

  De esta forma, la nueva consulta se construye automáticamente en función de la consulta anterior y de la información que proviene de los documentos considerados relevantes o irrelevantes por el usuario. En esta fórmula *Rel* es el conjunto de documentos relevantes recuperados y *Nrel* es el número de documentos no relevantes recuperados. Además hay que establecer cuál es el valor adecuado para las constantes α β y γ, que se utilizan para ponderar los términos que provienen de las distintas fuentes de información. Por otra

parte, también hay que decidir cuáles y cuántos documentos relevantes o irrelevantes se utilizarán en la reformulación de la consulta.

- *Presentación de descriptores admisibles*. La adecuada formulación de las solicitudes por el usuario depende en gran medida de su conocimiento del dominio y más concretamente del correcto conocimiento de la terminología que se emplea en el mismo. Para simplificar la realización de la consulta se hace seleccionando, de entre las palabras que presenta el propio sistema, aquellas que mejor describen su necesidad. De esta manera, se puede hacer una construcción incremental de las consultas y además durante el proceso se obtienen los descriptores que son compatibles con los que ya ha especificado el usuario, de modo que se impide la especificación de consultas que no obtengan ningún documento.

4.4.2 Hipertexto

El hipertexto se puede definir como una forma no secuencial de organización, acceso y presentación de la información textual [Jonassen 90a, Barret 89, Shneiderman 89b]. El término hipertexto se asocia habitualmente a un determinado tipo de interfaces en los que se utilizan elementos visuales (p.e., botones o iconos) y la interacción se lleva a cabo mediante dispositivos apuntadores como un ratón, aunque el concepto de hipertexto es más amplio. Esta técnica también se puede considerar como un paradigma de representación del conocimiento en el que parte de la información se encuentra representada en forma textual, mientras que otra parte del conocimiento se representa mediante conexiones (enlaces) entre partes de los textos [Mayfield 93]. En este trabajo utilizaremos ambos enfoques del hipertexto. Por una parte lo utilizamos como una forma de interacción, ya que proporciona versatilidad y no impone grandes requisitos para conseguir un uso efectivo por parte del usuario. Por otro lado, resulta muy adecuado para representar el conocimiento sobre dominios extensos en los que la información no está organizada y estructurada de una forma rígida. Estas cualidades lo hacen idóneo para su aplicación en un sistema de ayuda basado en el acceso a documentación. Un estudio más amplio sobre diferentes consideraciones y usos del hipertexto se puede encontrar en [Conklin 89, Nielsen 90, Jonassen 90c, Barret 89].

La idea original subyacente al hipertexto fue introducida por Bush en 1945, cuando describió su dispositivo *memex* para la organización y recuperación de información. Su planteamiento surgió a partir de los problemas derivados de la explosión en la cantidad de literatura científica, que dificultaba incluso a los especialistas el seguimiento del desarrollo de su propio campo de trabajo; se hacía necesario disponer de un sistema que almacenara toda esa información y que permitiera realizar consultas con una mayor rapidez y flexibilidad que con el formato impreso equivalente [Bush 45]. La idea resurge con un gran impulso a partir de los años 60 (cuando se acuña el término hipertexto), gracias a la generalización del uso de las computadoras [Balashubramanian 93].

4.4.2.1 Características y tipos de hipertextos

Los elementos más característicos de un hipertexto son los nodos y los enlaces. Un hipertexto se puede conceptualizar como una red de elementos de información, los *nodos*, y una serie de conexiones entre ellos, los *enlaces*. La esencia del hipertexto es un enlazamiento de conceptos que permite al lector seguir sus preferencias de forma instantánea y manteniendo el control. Esta modularización de la información permite que el usuario decida a qué nodo acceder en cada momento, imponiendo así al contenido del hipertexto su propia estructura (que puede ser diferente a la prevista por el autor).

El nodo (también denominado tarjeta, documento o artículo) es la unidad de información y normalmente representa una idea o concepto individual. Entre sus características más destacables de los nodos se encuentran el tamaño y el tipo. Su tamaño es muy dependiente de cada aplicación concreta y determina el grado de granularidad de la representación. Algunos estudios experimentales demuestran que es preferible tender hacia documentos no muy extensos, aunque sin que aumente de forma incontrolada el número de nodos. Por otro lado, también es importante distinguir entre los sistemas que permiten la existencia distintos tipos de nodos y aquellos en los que todos son iguales. Por ejemplo, puede haber nodos especiales que sirvan para estructurar la información o que sirvan para contener las referencias de un hiperdocumento.

Los enlaces definen las relaciones establecidas entre los nodos. En su nivel más básico, se trata de una conexión entre dos unidades de texto. Son elementos que actúan como puntos de entrada, permitiendo, al ser seleccionados, el acceso a una información relacionada. Los enlaces son el medio más importante de navegación a través de los documentos que componen un hipertexto. Existe una gran diversidad de posibilidades en cuanto a la organización de los enlaces: si se colocan en el propio texto o al margen de forma diferenciada, si se produce un enlace con otro nodo completo o también se permite el acceso directo a uno de sus fragmentos, y si se permiten tipos diferentes de enlaces o no. Por ejemplo, hay sistemas como el World Wide Web que no soportan distintos tipos de enlaces, de forma que si el autor quiere diferenciarlos de alguna manera, debe incluir una anotación en el propio texto indicando cuál es el propósito.

Además de por sus distintas posibilidades estructurales, tanto para los nodos como para los enlaces, podemos encontrar diferentes tipos de hipertextos de acuerdo con el tipo de interacción que permiten. Existen hipertextos que únicamente permiten una visualización de una información previamente existente, otros en los que el usuario puede añadir nueva información e incluso algunos que permiten la colaboración entre distintos autores. También hay notorias diferencias en la organización que se impone a la documentación y en las ayudas que se proporcionan al usuario para encontrar un contenido determinado. Hay sistemas que organizan su información de forma jerárquica, otros que no proporcionan ningún tipo de estructura e incluso algunos que soportan simultáneamente diferentes organizaciones para los contenidos.

Con el desarrollo de los distintos sistemas de hipertexto se han identificado una serie de problemas que están asociados con su utilización. Entre otros, se pueden citar la desorientación y la sobrecarga cognitiva [Conklin 87]. Los usuarios normalmente están acostumbrados a utilizar documentación impresa en la que encuentran la información que necesitan con la ayuda de tablas de contenido o índices. Esa costumbre hace que se puedan "perder" o desorientar en los hipertextos por los nuevos modos de interacción que éstos permiten. Para reducir esa desorientación es necesario que los sistemas integren nuevos mecanismos de ayuda para los usuarios. A medida que las bases de datos aumentan en extensión, se hacen necesarias herramientas más avanzadas de navegación, tales como mapas globales o locales sobre la información y sus enlaces, o medios automáticos de localización de nodos particulares en base a su contenido. Así, es preciso complementar el hipertexto con otras técnicas como, por ejemplo, la recuperación de información [Allan 95].

Por otro lado, los hipertextos incluyen tal cantidad de información, que se produce lo que se conoce como sobrecarga cognoscitiva. Cuando el usuario lee un hiperdocumento, debe estar continuamente tomando decisiones sobre qué enlaces seguir y cuáles ignorar, pudiendo producirse un problema de navegación y desconcierto. Sin embargo, no todos los autores están de acuerdo en que éstas sean realmente deficiencias, considerando algunos, por ejemplo, que una cierta desorientación puede ayudar a ampliar el conocimiento del usuario sobre el tema que se le presenta [Landow 90, Mayes 90].

4.4.2.2 Hipertexto y sistemas de ayuda

A pesar de los problemas reseñados en el apartado anterior, nosotros consideramos que el hipertexto, junto con un adecuado interfaz, resulta muy adecuado para la interacción con el usuario final en un sistema de ayuda [Fernández-Manjón 95a, Sullivan 91, Legget 90]. Su uso efectivo tiene muy pocos requisitos previos, no siendo necesario que el usuario posea un amplio conocimiento del dominio o esté familiarizado con la Informática. En palabras de Shneiderman,

> *"Hypertext systems offer a surprising and satisfying freedom to explore. With little training in computer concepts and with little knowledge of the subject domain, hypertext users can casually traverse nodes and links looking for something of interest. Direct manipulation enables an easy-to-use mode of interaction; reversibility of action conveys a sense of safety and security." [Shneiderman 89a]*

Estas cualidades hacen que se utilice ampliamente en aplicaciones educativas o de ayuda [Jonassen 90c, Jonassen 90b]. Además, presenta la ventaja de ser muy adecuado para la presentación de grandes cantidades de información, sin que sea necesario que ésta tenga previamente una organización muy definida [Mayes 90].

> *"Hypertext is becoming a popular approach to many computer applications, especially those dealing with the on-line presentation of large amounts of loosely structured information such as on-line documentation or computed-aided learning." [Nielsen 90]*

Además del auge del hipertexto como interfaz y su adecuación para el uso en aplicaciones educativas [Maybury 93, Kommers 92], es interesante destacar dos de las tendencias actuales en la construcción de hipertextos:

a) La generación automática o semiautomática de hipertexto a partir de información preexistente [Allan 90].

b) El uso de técnicas de inteligencia artificial, originando nuevas estructuraciones de la información, lo que también se llama hipertexto de dos niveles [Mayfield, en prensa, Balashubramanian 93].

Estos enfoques los hemos contemplado en el desarrollo de nuestros sistemas Argos y Aran, sistemas que presentamos en los siguientes capítulos.

4.4.3 Modelado de usuario

El modelo del usuario (MU) está constituido básicamente por el conocimiento que posee un sistema sobre un usuario y que el sistema usa para mejorar su interacción con aquel [Finin 89]. Su objetivo principal es el de adaptar el comportamiento del sistema a cada usuario concreto [Carberry 92]. Esta adaptación puede ser de índole muy diversa. Por ejemplo, se puede adecuar la respuesta al nivel de conocimientos del usuario p.e., realizando un filtrado selectivo de la información.

Una definición de modelado del usuario más específica, y más adecuada al enfoque de este trabajo, es la que surge en el marco de los sistemas de interacción mediante lenguaje natural [Wahlster 89], siendo extensible a otros sistemas [Kass 91]:

> *"A user model is a knowledge source in a (natural-dialog) system which contains explicit assumptions on all aspects of the user that may be relevant to the behavior of the system. These assumptions must be a separable by the system from the rest of the system's knowledge" [Wahlster 89]*

El modelo de usuario debe contener, por tanto, las suposiciones explícitas sobre todos los aspectos del usuario que utiliza el sistema y que son relevantes para adaptar el comportamiento interactivo del sistema al usuario. Para que pueda ser considerada propiamente como un modelo, la información debe estar representada explícitamente y poder separarse (por el propio sistema) del resto de conocimiento que se posea [Kobsa 94]. El componente de modelado de usuario es la parte del sistema que se encarga de la gestión del modelo: de su adquisición incremental; del almacenamiento y actualización; del mantenimiento de su consistencia; y de proporcionar esta información sobre el usuario al resto de los componentes.

Otras características deseables del MU son [Finin 89]: a) una representación explícita que permita realizar inferencias; b) una representación declarativa en lugar de procedimental; y c) una representación general que permita diferentes usos de la información contenida.

Los modelos de usuario no son justificables en todos los sistemas interactivos, ya que su creación y gestión suponen un coste añadido. Su utilización es adecuada si por lo menos se cumple alguna de las siguientes características [Kass 91]:

1. El sistema busca adaptar su comportamiento a cada uno de los usuarios de forma individual.

2. El sistema asume responsabilidades (o comparte responsabilidades con el usuario) para asegurar el éxito de la comunicación usuario-sistema.

3. Los usuarios potenciales del sistema, o los usos potenciales del sistema, son diversos.

Nuestra propuesta de sistema de ayuda satisface esas tres condiciones, por lo que está completamente justificada la aplicación de un modelo de usuario. Además, compartimos la opinión de muchos autores, quienes consideran la existencia de este modelo como un prerrequisito necesario para que un sistema sea capaz de exhibir un comportamiento cooperativo de amplio rango [Wahlster 89, Kok 91].

4.4.3.1 Consideraciones iniciales sobre el modelado

Existe una estrecha relación entre el modelo de usuario que planteamos en nuestros sistemas de ayuda y el modelo de alumno que se utiliza en los tutores inteligentes [Kass 89]. Pero, no obstante, tienen diferentes pretensiones, ya que aquí no se trata de capturar el estado cognitivo del estudiante, sino de mejorar la interacción y, por tanto, las demandas sobre los modelos de usuario no son tan exigentes. Las características particulares de la ayuda hacen que no sean adecuados los modelos clásicos de modelado de alumno en tutores (p.e., de *overlays* o de perturbaciones) [Chin 89]. Las dos condiciones principales que según Chin hacen que un modelo sea adecuado para su uso en un sistema de ayuda son [Chin 93]:

a) que sea de rápida adquisición.

b) que proporcione un cierto poder predictivo en función de un conocimiento parcial.

Los dos tipos de modelos que utilizamos en nuestros asistentes (y que presentamos más detalladamente en los apartados siguientes), las clases de usuarios como los estereotipos, se han mostrado como un enfoque adecuado para áreas de aplicación en las que se necesita una evaluación rápida, aunque no necesariamente precisa, de las características del usuario. En el modelo del sistema Argos aplicamos las clases de usuarios, con una clasificación doble que tiene en cuenta tanto la experiencia del usuario como su principal interés en el uso del sistema. Además, también realizamos una doble clasificación de las órdenes en función de su propósito y dificultad de uso. En el momento de la activación de Argos se realiza una clasificación del usuario que no se modifica a lo largo de la sesión. En el sistema Aran realizamos un modelado más detallado mediante una jerarquía de estereotipos e implementamos un mecanismo de mantenimiento de la coherencia del modelo. De esta forma, la asignación de estereotipos de usuario es dinámica en

función de la información de la que se va disponiendo. Estos dos sistemas se describen con detalle en capítulos posteriores.

Como se pueden producir errores en la adquisición del modelo, se ha dedicado un esfuerzo especial a que el usuario pueda inspeccionar y modificar los datos que el sistema tiene de ellos. Lo que se obtienen son modelos de usuario inspeccionables por el propio usuario. Así, el usuario tiene un mayor control y puede comprender mejor el funcionamiento del sistema de ayuda [Kay 95, Orwant 95, Boyle 94].

4.4.3.2 Caracterización del modelado de usuario

El campo del modelado del usuario es muy amplio y existen enfoques muy diversos. A continuación realizamos una breve descripción de las posibles opciones de modelado en cuanto a quién modelar, cuál es el contenido del modelo, cómo adquirir y mantener la información y cuál es el uso que se hace de esos datos [Finin 89, Kok 91].

- *Tipo de modelo utilizado.* Los dos aspectos considerados en el tipo de modelo son el periodo o duración temporal que se tiene en cuenta para este modelado y la granularidad de la representación. La duración temporal se refiere a la persistencia del conocimiento codificado en el modelo del usuario, es decir si esa información se mantiene a largo plazo o si se descarta al final de la sesión en curso. En los sistemas de ayuda desarrollados en esta tesis se desarrolla un modelo a corto plazo. Dos son las razones para esta decisión: la primera es que las circunstancias de cada persona entre dos usos distintos del sistema pueden haber cambiado mucho (por ejemplo, su conocimiento sobre la aplicación se puede haber incrementado significativamente debido a que haya realizado un curso); la segunda es que los modelos propuestos son de rápida adquisición, no siendo necesario guardarlos por razones de cálculo o de tiempo, además de que en muchas ocasiones habría que tratar con información contradictoria. Por otra parte, la creación de un modelo de largo plazo es más complicado y si se cometen errores en la adquisición, éstos tienen consecuencias más graves que si ocurren en características transitorias [Kobsa 93].
  La granularidad es el grado de especialización del modelo de usuario, que puede variar desde el modelado genérico único hasta el modelado completamente individual. Como punto intermedio en la escala se encuentran el uso de varios modelos genéricos fijos que representen las diferentes clases de usuarios con características homogéneas respecto a la aplicación. Hay sistemas que, por ejemplo, clasifican a un usuario dentro de uno de los siguientes grupos: novato, principiante, intermedio o experto. Este tipo de clasificación facilita la inclusión de las suposiciones que se pueden realizar sobre los individuos pertenecientes a estos grupos. En el extremo contrario de esta escala se encuentra el modelado individual del usuario, donde se tiene un modelo propio para cada uno de los que utilizan el sistema.
  También existen enfoques mixtos que combinan dos de estas aproximaciones. Por ejemplo, se puede tener a los usuarios clasificados en grupos, pero permitiendo la especialización de esa información para cada uno de los usuarios individuales. Este es el caso de las clases de usuarios [Chin 89] y los

estereotipos [Rich 89], que trataremos con más detenimiento en el siguiente apartado.

- *Contenido del modelo.* El contenido del MU viene dado por la elección de las características contempladas en el modelo. Existe una gran variedad de características o rasgos a modelar, ya que dependen mucho del tipo de aplicación. No obstante, este conocimiento se puede clasificar dentro de cuatro amplias categorías [Kobsa 94]: a) objetivos y planes; b) capacidades; c) aptitudes o preferencias; y d) conocimientos o creencias (*belief*). En nuestro trabajo modelamos el conocimiento que tiene el usuario sobre el dominio específico del sistema. De una forma implícita también tenemos en cuenta el interés (objetivo) del usuario en la búsqueda de información (caracterizándolo mediante el vocabulario que utiliza). Conocer qué es lo que el usuario cree que es cierto (conoce) en el dominio de aplicación, es especialmente útil en los sistemas de ayuda, ya que permite adaptar la respuesta a la experiencia y el interés.

- *Adquisición y mantenimiento del modelo.* La adquisición del modelo viene dada por el conjunto de técnicas utilizadas para obtener los datos sobre los usuarios. El mantenimiento implica la incorporación de nueva información en el modelo existente y la solución de cualquier discrepancia o contradicción que se pudiera producir debido a ese nuevo conocimiento.
  La adquisición puede ser explícita o implícita. En la adquisición implícita, el usuario tiene la iniciativa en la interacción y no se le solicitan datos específicos para el modelado, sino que es el propio sistema el que los va adquiriendo, completando y actualizando, durante el uso, de una forma no intrusiva [Self 90a] (algunos autores la denominan adquisición automática o pasiva [Kobsa 92]). También se puede realizar una adquisición explícita en la que el usuario debe proporcionar los datos necesarios para el modelado, por ejemplo rellenando un formulario o indicando su interés o experiencia sobre un área [Jonhson 94]. Ciertas peculiaridades de los programas de ayuda, como que su uso es ocasional y las posibles diferencias entre el punto de vista del diseñador del sistema y el del usuario respecto a las características modeladas, desaconsejan esa adquisición explícita. Un estudio más amplio de las distintas opciones en adquisición de modelos se encuentra en [Chin 93].

- *Utilización del modelo.* El conocimiento que proporciona el modelo del usuario se puede utilizar con distintos objetivos que dependen de la aplicación. Sin embargo, sus usos se pueden clasificar genéricamente como ayuda para [Finin 88]: a) reconocer e interpretar el comportamiento del usuario en la búsqueda de información; b) proporcionar ayuda y consejo al usuario; c) completar la información suministrada por el usuario en la interacción; y d) proporcionar información complementaria al usuario.
  En nuestros sistemas se integran varias de estas opciones, ya que el objetivo concreto es la adecuación de la ayuda a cada usuario. Esto se reflejará en una modificación de los procesos de búsqueda y presentación de la información, debido a que se trabaja fundamentalmente con textos prealmacenados y no es posible la modificación de la propia información, lo que sí llevan a cabo otros sistemas [Boyle 94, Jonshon 94].

4.4.3.3 Clases de usuarios y estereotipos

En este trabajo, para el modelado de los usuarios seguimos dos enfoques mixtos [Kobsa 89, Kobsa 93]: uno basado en categorías de usuarios [Chin 89] y otro basado en un refinamiento de esa aproximación, los estereotipos [Rich 89]. Estos enfoques se fundamentan en la evidencia empírica de que hay conjuntos de características de las personas que se presentan frecuentemente agrupadas. Por tanto, estas regularidades en las características se pueden utilizar para clasificar a los usuarios en distintos subgrupos, a cada uno de los cuales se les da un tratamiento diferente. Esto permite realizar suposiciones sobre un usuario concreto, teniendo como base únicamente una información parcial, que permita clasificarlo dentro de un determinado subgrupo. Estas predicciones o suposiciones son muy útiles en ausencia de datos mas concretos en los que basarse, ya que se pueden realizar en función de un número pequeño de evidencias que es posible adquirir antes de que sea necesario su uso.

En estos enfoques, el desarrollador del modelado de usuario de una aplicación tiene que completar las siguientes tareas:

- *Identificación de los subgrupos de usuarios.* Se deben identificar los subgrupos que clasifiquen a los usuarios esperados y cuyos miembros es muy probable que tengan unas características homogéneas para la aplicación.

- *Identificación de características clave.* Se debe identificar un pequeño grupo de características clave que permitan identificar que un usuario pertenece a un determinado subgrupo. La presencia o ausencia de estas características debería ser reconocida automáticamente por el sistema.

- *Representación explícita de estas características.* Las características relevantes de los subgrupos de usuarios identificados deberían formalizarse en un sistema apropiado de representación. En los estereotipos, la colección de todas las características representadas para un subgrupo de usuarios se llama estereotipo del subgrupo. Si el contenido de un estereotipo es un subconjunto de otro, entonces se puede aplicar un sistema de jerarquía con herencia. De esta forma, los contenidos de los estereotipos más generales son heredados por los estereotipos que son especializaciones.

Este tipo de modelado de usuario, utilizando como criterio de agrupación los diferentes niveles de conocimiento de los usuarios, se ha empleado para adaptar información y consejos a los usuarios de sistemas operativos [Chin 89, Nessen 89, Boyle 94]. La mayoría de los autores postulan una jerarquía lineal (plana) de clases correspondientes a principiantes, intermedios y expertos (fig. 4.3). Habitualmente, un usuario puede pertenecer sólo a una de esas clases. Sin embargo, a pesar de la probada utilidad de las clases de usuarios, es dudoso que una clasificación tan simple pueda justificarse empíricamente. De hecho, estudios experimentales en el campo del sistema operativo Unix [Sutcliffe 87, Draper 84], han obtenido como resultado que para tener en cuenta la familiaridad de los usuarios con las órdenes de un sistema operativo es más adecuado la identificación de agrupaciones de conocimiento que sean coherentes para el usuario, que aplicar únicamente un

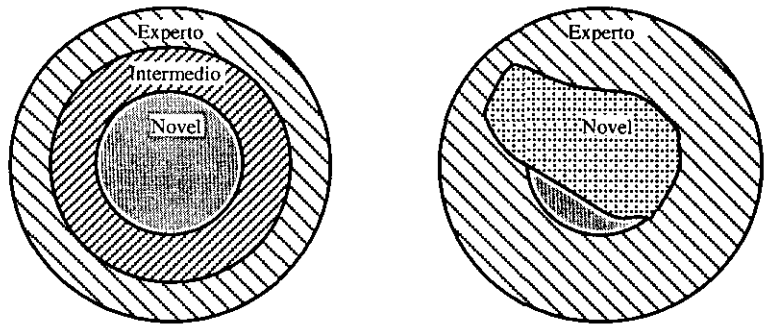


Figura 4.3: Modelos de usuario. En la parte izquierda se muestra el modelo clásico de capas o niveles y a la derecha el modelo de especializaciones (estereotipos), que considera que parte del conocimiento de los usuarios noveles puede estar al nivel de los usuarios expertos.

criterio general de nivel de experiencia. Esta circunstancia se puede representar mediante un modelo de especializaciones que tendrán su reflejo en distintos estereotipos, que permiten realizar clasificaciones de los usuarios de acuerdo a diferentes criterios y es posible la aplicación de diversos estereotipos a un mismo usuario [Höök 95].

4.4.4 Representación explícita del conocimiento

La representación del conocimiento es la formalización explícita de la información sobre un determinado dominio, de modo que sea procesable automáticamente por un sistema que utilice técnicas de inteligencia artificial [Kramer 87, Gonzalez 93]. Como enuncia Barr,

> *"In artificial intelligence, a representation of knowledge is a combination of data structures and interpretative procedures that, if used in the right way in a program, will lead to a knowledgeable behavior." [Barr 81]*

Esto supone que, además de las operaciones de almacenamiento y recuperación directa de la información, esa representación debe permitir también la realización de inferencias a partir de ella [Brachman 85]. Esta información, denominada base de conocimiento, debe ser principalmente declarativa y algunos autores postulan que debe ser lo bastante completa como para que sea asimilable a la que posee un experto humano sobre el tema concreto [Tansley 93]. Los sistemas que utilizan una base de conocimiento se denominan sistemas basados en conocimiento (SBC) [Hayes-Roth 94].

Existe una gran diversidad de aproximaciones diferentes para la representación del conocimiento. Entre otras, se pueden destacar las basadas en la lógica, los sistemas basados en reglas, los sistemas de razonamiento estadístico, los sistemas de marcos

(*frames*), y las redes semánticas [Barr 81, Weida 91, Rich 94, Brachman 90, Gonzalez 93, Quillian 67]. Además, cada enfoque se plasma en formalismos o lenguajes de representación concretos y diferentes, que se muestran especialmente adecuados para determinadas aplicaciones particulares. Por ejemplo, los lenguajes lógicos estándar son apropiados en situaciones en las que se necesita expresar principalmente aserciones de hechos. Los sistemas de producción son muy útiles cuando se dispone de conocimiento heurístico, que se puede expresar mejor mediante construcciones del tipo condición/acción. Los sistemas de *frames* o marcos, son muy adecuados cuando se desea describir conjuntos de objetos con una estructura y relaciones complejas, y cuando el dominio presenta una fuerte taxonomía jerárquica en los objetos a representar [Devanbu 91]. Por tanto, como ya enunciaba Brachman, no se puede afirmar que ninguno de estos formalismos sea mejor que el resto en todas las situaciones, sino que hay que llegar a un compromiso entre las ventajas e inconvenientes que presentan [Brachman 85]. Además, en el desarrollo de un SBC, hay que tener en cuenta no sólo factores como la riqueza expresiva del sistema de representación y las posibilidades de inferencia que proporciona, sino también otras como la mantenibilidad y el coste de desarrollo [Boy 91].

Entre las propuestas de este trabajo se encuentra la reutilización de las bases de conocimiento preexistente, permitiendo el desarrollo de asistentes aplicables a dominios más extensos y contribuyendo a la disminución del coste final. Sin embargo, hay circunstancias que dificultan el proceso de reutilización de la información de aplicaciones previas, incluso en un mismo dominio de aplicación, entre las que cabe destacar la especificidad de los formalismos y de las técnicas de representación utilizadas [Musen 92]. Para lograr el objetivo de reutilización, proponemos el uso de un método estándar de representación de conocimiento que proporcione una gran capacidad expresiva. Así simplificamos el aprovechamiento de desarrollos anteriores, ya que al utilizar una representación ampliamente difundida, es más probable que haya bases de conocimiento que usen ese formalismo. Por otra parte, si existe conocimiento reutilizable expresado con una representación diferente, la gran capacidad expresiva facilitará el proceso de traducción. Otras características deseables para la representación son que permita una construcción incremental de la base de conocimiento y que proporcione herramientas que simplifiquen su depuración y mantenimiento.

Todo lo anterior hace que propongamos las lógicas descriptivas como herramienta de representación del conocimiento para el desarrollo de sistemas de ayuda [Borgida 92]. Las lógicas descriptivas (también denominadas sistemas basados en clasificación o sistemas terminológicos) surgen a partir del sistema KL-ONE y constituyen uno de los intentos más consistentes para obtener un lenguaje de representación del conocimiento de propósito general [Brachman 90, Weida 91]. Se trata de sistemas centrados en los objetos, siguiendo con la tradición de los sistemas de "frames" y de las redes semánticas, y además integran esas potencialidades con las reglas y la lógica. Estos formalismos se han utilizado con éxito en diversas áreas, como los sistemas expertos, en aplicaciones de recuperación o de visualización conceptual de bases de conocimiento [Devanbu 91, Brachman 91], en tratamiento del lenguaje natural [Speel 95] y en adquisición del conocimiento [Gil 94]. El sistema concreto que utilizamos para la representación del conocimiento en

este trabajo es Loom [MacGregor 91]. Loom es un sistema híbrido que aúna un modelo profundo del dominio con gran riqueza expresiva y que permite también la inclusión de reglas.

En el siguiente apartado presentamos una breve descripción de los fundamentos en los que se basa la representación del conocimiento propuesta por las lógicas descriptivas y que se ha utilizado para construir la base de conocimiento de nuestro sistema Aran. Además, introducimos la terminología y las principales características del formalismo concreto utilizado, Loom, de modo que se puedan comprender mejor los ejemplos concretos del sistema Aran descritos en el capitulo 6.

4.4.4.1 Lógicas descriptivas: sistemas basados en clasificación

Los sistemas basados en lógicas descriptivas surgen con el sistema KL-ONE y actualmente entre sus descendientes se pueden destacar KL-TWO, BACK, LOOM y CLASSIC. Suponen una línea de trabajo muy activa, con un amplio uso y con la aparición continua de nuevas versiones con mayores capacidades, con las que se pretende lograr un lenguaje de representación del conocimiento de propósito general [Weida 91, Brachman 91]. Son entornos híbridos que combinan diferentes paradigmas de programación, aunando en un mismo sistema y de forma integrada las ventajas proporcionadas por los objetos, las reglas y la lógica.

Las características comunes a todos estos sistemas son [McGregor 91]:

- Están basados en la lógica. Su semántica se obtiene a partir de la lógica de primer orden.

- Realizan una diferenciación entre conocimiento terminológico y conocimiento asercional. Proporcionan un lenguaje específico para la definición estructural de términos abstractos o clases de objetos (denominados *conceptos*) y un lenguaje que se usa para asertar la existencia de individuos de esas clases (llamados *instancias*) y que también permite asertar hechos sobre esos individuos.

- Incluyen un *clasificador* que organiza los términos (conceptos) en una taxonomía mediante las relaciones de especialización-generalización entre pares de términos.

Habitualmente, en estos sistemas se trabaja con una semántica de mundo abierto (*open-world semantic*), de modo que si una proposición no se puede probar como cierta, no se asume directamente que es falsa, sino que se considera como desconocida.

4.4.4.2 Conceptos, relaciones e instancias

Los tres tipos básicos de objetos que se manipulan en estos lenguajes son los *conceptos*, las *relaciones* y las *instancias*.

- *Conceptos.* Un concepto representa una clase genérica de objetos del mundo real. Los conceptos se definen, utilizando el lenguaje terminológico, mediante un conjunto de condiciones necesarias y suficientes, expresadas por medio de operadores lógicos. Al definir un nuevo concepto, éste se inserta en una taxonomía de conceptos con herencia múltiple, de modo que los conceptos más generales se establecen como ascendientes y los conceptos más específicos como descendientes. Un concepto C_1 es más general (*subsumes*) que un concepto C_2, si y sólo si todos los individuos descritos por C_2 también están descritos por C_1. Este es el proceso automático de *clasificación.*
  También se pueden definir *conceptos primitivos*, que únicamente incluyen condiciones necesarias. Sirven para definir los objetos que son demasiado complejos o demasiado generales para definirlos de una forma precisa. Habitualmente son los que componen los niveles superiores de la taxonomía, y a partir de los cuales se derivan los conceptos más específicos.

- *Relaciones.* Las *relaciones* (*roles*) sirven para expresar propiedades de los objetos y para relacionarlos entre sí. Son similares a los atributos de los sistemas de frames o a los enlaces "has-a" de las redes semánticas, pero pueden tener más de un valor de relleno (*fillers*). El lenguaje terminológico se puede utilizar para definir las propiedades de una relación, como por ejemplo, restricciones de número y/o de tipo para los valores posibles.

- *Instancias.* Los *individuos o instancias* representan objetos concretos del dominio. Se obtienen por la instanciación de un concepto. Se declaran utilizando el lenguaje de asertos, a partir del cual el sistema puede realizar varios tipos de razonamiento automático. De esta forma, a los individuos se les pueden asignar propiedades afirmando que satisfacen un determinado concepto, o dando valores a las relaciones asociadas con él y dejando que el sistema clasifique automáticamente al individuo por debajo del concepto más específico.

4.4.4.3 Inferencias

Los sistemas de representación basados en lógicas descriptivas permiten una serie de inferencias deductivas, independientes del dominio, que simplifican la creación de la base de conocimiento y el desarrollo de aplicaciones. Estas inferencias son [Brachman 91]:

- *Completamiento.* Se calculan las consecuencias lógicas de las aserciones respecto a las instancias y a las descripciones de los conceptos.

  Herencia. Las restricciones que se aplican a las instancias de un concepto son heredadas (se aplican) también a las instancias de todas sus especializaciones.
  Combinación. Las restricciones sobre los conceptos e instancias se combinan para obtener restricciones más estrictas.
  Propagación. Cuando se hace un aserto sobre una instancia, ésta se propaga, ya que puede tener consecuencias lógicas para otros individuos relacionados.

Detección de contradicciones. Se detectan automáticamente cuando se hacen dos asertos sobre un individuo que no pueden ser ciertos simultáneamente.
Detección de conceptos incoherentes. Se detectan automáticamente cuando las restricciones impuestas en la definición de un concepto hacen lógicamente imposible la existencia de instancias de dicho concepto.

- *Clasificación y generalización-especialización.* Se realizan inferencias relativas a las definiciones estructurales de los objetos y a la taxonomía.

  Clasificación de conceptos. Cuando se define un nuevo concepto, se determinan todos los conceptos más generales y todos los conceptos más específicos.
  Clasificación de instancias. Se determinan todos los conceptos cuya definición es satisfecha por una instancia.

- *Aplicación de reglas.* Existen reglas simples de razonamiento hacia delante (*forward-chaining*) que tienen conceptos como antecedentes y como consecuentes. Cuando se determina que una instancia satisface el antecedente de una regla, se aserta que también satisface el consecuente.

4.4.4.4 Loom

Loom es un entorno diseñado para la construcción de sistemas expertos y otros sistemas basados en conocimiento. Ha sido desarrollado en el Information Sciences Institute de la Universidad de Southern California [McGregor 91, ISX 91, Brill 93].

El entorno Loom proporciona:

- Un lenguaje preciso y expresivo para la especificación de modelos declarativos.

- Unos mecanismos deductivos muy potentes, que incluyen razonamiento estricto y por defecto, junto con un mecanismo automático de mantenimiento de la consistencia.

- Inclusión de diferentes paradigmas de programación integrados con la especificación declarativa del modelo.

- Funcionalidades de gestión de bases de conocimiento, como un completo lenguaje de consulta con el poder expresivo de la lógica de primer orden, el mantenimiento de múltiples bases de conocimiento o la carga y descarga de objetos de la base de conocimiento.

Loom trata de evitar parte de los problemas detectados en los primeros sistemas basados en reglas, como la dificultad de mantenimiento y ampliabilidad, o como la no diferenciación entre distintos tipos de conocimiento (p.e., estructural y de control). Mediante la definición de términos se hace posible la construcción de definiciones ricas y precisas de los conceptos del dominio. El clasificador automático

que incluye proporciona una validación de la definición de los conceptos y su organización automática en una jerarquía de acuerdo con su nivel de abstracción, permitiendo así el desarrollo de grandes taxonomías de conceptos. Una vez que se ha desarrollado un modelo profundo del dominio, mediante la especificación de la jerarquía de conceptos, es más sencillo para el desarrollador proceder con la definición de las reglas que se aplicarán a situaciones específicas.

Además de las capacidades genéricas de inferencia descritas para las lógicas descriptivas, el sistema Loom ofrece:

- *Eliminación de hechos previamente asertados (retraction).* Permite la eliminación de conocimiento previamente asertado. Para ello, incluye un mecanismo de mantenimiento de la verdad que, cuando se aserta o retracta información, revisa el conjunto de inferencias realizadas.

- *Razonamiento por defecto.* Se incluyen distintas características de razonamiento por defecto. Por ejemplo, se pueden especificar los valores por omisión de una relación si no se proporciona otra información más específica. También se puede tener un cierto control sobre las inferencias que se realizan, ya que se pueden especificar características específicas sobre un concepto o sobre una relación, como por ejemplo, que se aplique la semántica de mundo cerrado.

A continuación presentamos una serie de ejemplos (comentados) de Loom, para ilustrar la potencia y posibilidades que ofrece la integración de esta tecnología en la construcción de sistemas de ayuda. Se muestran los diversos tipos de objetos con los que se puede trabajar, y se destacan sus particularidades para hacer más comprensible el formalismo utilizado. Estos ejemplos (algunos son descripciones parciales o simplificadas) se han tomado de los objetos utilizados en la base de conocimiento del sistema Aran.

En Loom, como raíz de la taxonomía se encuentra el concepto primitivo *Thing*, a partir del cual descienden el resto de los conceptos. La definición de conceptos se realiza mediante *defconcept*. Por ejemplo, a continuación se define un proceso de Unix, con el identificador *Process*, como un concepto primitivo (mediante *:is-primitive*):

```
(defconcept Process
  "un proceso unix"
  :is-primitive
  (:and unix-entity
   (:all has-ID number)
   (:exactly 1 has-ID)
   (:the has-owner number)
   (:the has-priority number))
  :implies
  (:the has-parent number))
```

El proceso se declara como subconcepto de *unix-entity* (mediante *:and unix-entity*) con cuatro propiedades o características representadas por las relaciones *has-ID,*

has-owner, *has-priority* y *has-parent*. Para *has-ID* se restringe el valor de relleno a que sea un número (con *:all has-ID number*) y a que sólo pueda haber un valor de relleno (mediante *:exactly 1 has-ID*). Para el resto de las relaciones se definen las mismas restricciones pero con una sintaxis más compacta usando *:the relación number*. *Number* es un concepto predefinido del sistema que representa a cualquier número. Las relaciones y conceptos incluidos dentro de la cláusula *:is-primitive* (o *:is* si la definición es no primitiva) expresan las condiciones necesarias y suficientes para la pertenencia de un individuo a dicho concepto. Mediante *:implies* se denotan las condiciones necesarias pero no suficientes de pertenencia a un concepto, de modo que esta construcción también se puede ver como asertos que se realizan sobre un individuo una vez que se ha clasificado como perteneciente a un concepto.

A partir de esta definición de proceso se puede definir completamente, es decir de forma no primitiva, los subconceptos proceso de sistema (*System-Process*) y proceso de usuario (*User-Process*). Un proceso de sistema se define como un tipo de proceso cuyo propietario es el de número de identificación cero (*:filled-by has-owner 0*). Un proceso de usuario se define como un proceso cuyo propietario tiene un número de identificación positivo (*:the has-owner (:through 1 +INFINITY)*). Además, para este último mediante *:defaults* se proporciona un valor por omisión para la prioridad.

```
(defconcept System-Process
  :is
  (:and Process
       (:filled-by has-owner 0)))

(defconcept User-Process
  :is
  (:and Process
   (:the has-owner (:through 1 +INFINITY)))
  :defaults
  (:filled-by has-priority 28))
```

La definición de relaciones se hace mediante *defrelation*. A continuación, se define la relación *has-owner* restringiendo el rango de valores que puede tomar a valores numéricos. Mediante *:characteristics* se puede modificar el tipo de razonamiento que realiza el sistema, tanto con las relaciones como con los conceptos, y en este caso se afirma que es univaluada.

```
(defrelation has-owner
  :range number
  :characteristics (:single-valued))

(defrelation has-priority
  :domain Process
  :range number)
  :characteristics (:closed-world))
```

Para la relación *has-priority*, se restringe su dominio de aplicación a los procesos y su rango de posibles valores a valores numéricos. Además, se determina que en el razonamiento con dicha relación se puede aplicar la semántica de mundo cerrado (por omisión, se utiliza semántica de mundo abierto).

Los asertos se realizan mediante *tell* o *tellm*. Por ejemplo, el siguiente aserto respecto a *process01*, haría que el sistema lo reconociera y clasificara directamente como un proceso de sistema, aunque sólo se afirma que es un proceso y que su propietario tiene número cero.

```
(tellm (:about process01 Process
  (has-owner 0)))
```

Esto es así, debido a que se componen las restricciones de que un proceso tiene un único propietario, que el propietario de un proceso de sistema tiene número cero y que la relación *has-owner* es univaluada.

En Loom, además, se realiza una diferenciación explícita entre el conocimiento declarativo de modelado y el conocimiento procedimental de comportamiento. El conocimiento declarativo es el formado por las definiciones, implicaciones (reglas simples), hechos y reglas por defecto. El conocimiento procedimental es el formado por las reglas, las acciones y los métodos, que se definen mediante *defproduction*, *defaction* y *defmethod*, respectivamente. Una acción es un objeto que especifica una operación procedimental genérica o un conjunto de ellas. Los métodos son las funciones específicas que se ejecutan, dependiendo del contexto, cuando se invoca una acción genérica. Las producciones son reglas dirigidas por datos que se usan para detectar situaciones determinadas y provocar la ejecución de tareas formadas por una o más acciones genéricas.

A continuación se muestra la definición de la producción *nuevo-prototipo*, que detecta cuándo se clasifica un nuevo individuo del concepto *user-prototype* (con *:when (:detects (user-prototype ?user))*) y provoca la ejecución de la acción *asign-prototype* (mediante *:schedule (asign-prototype ?user)*). La acción *asign-prototype* está implementada por todos los métodos que tengan ese mismo nombre y en ella se especifica que si hubiera más de uno aplicable, se elegiría el más específico (con *:filters (:most-specific)*). El método que se muestra será aplicable si el argumento *?user* es una instancia de *user-prototype* (por *:situation (user-prototype ?user)*) y provocará la evaluación de su argumento *:response*, dado por funciones Lisp.

```
(defproduction nuevo-prototipo
    :when (:detects (user-prototype ?user))
    :schedule (asign-prototype ?user))

(defaction asign-prototype (?user)
    :filters (:most-specific))

(defmethod asign-prototype (?user)
  :title "Asigna los prototipos aplicables un usuario"
  :situation (user-prototype ?user)
  :response
  ((format t"~% Asignando prototipos ...~%~A~%" (asignar-
prototipos ?user))))
```

Es decir, se distingue entre las reglas que son puramente deductivas y las reglas procedimentales, e incluso dentro de estas últimas se diferencia el conocimiento utilizado para la ejecución de una regla y la selección entre las diversas acciones o respuestas posibles a esa regla.

De esta forma, alrededor del modelo de dominio explícito y declarativo, e integrados en él, se combinan tres paradigmas de programación diferentes [Weida 91]:

1. Programación orientada a objetos, con operadores y métodos.

2. Programación dirigida por datos, con un sistema de producción y un control basado en agenda.

3. Programación basada en restricciones con mantenimiento de la verdad.

Hay otras características que hacen que Loom sea especialmente adecuado para el desarrollo de aplicaciones, como la inclusión de conceptos predefinidos que simplifican la definición de nuevos conceptos y el hecho de estar plenamente integrado en el entorno Lisp, pudiendo así utilizar toda su potencialidad. Por ejemplo, mediante la construcción *predicate,* permite incluir tests procedimentales programados en Lisp, para determinar si un individuo satisface una definición. Como inconveniente, cabe destacar la ausencia de entorno gráfico que facilite la manipulación de la base de conocimiento. Esta circunstancia ha llevado a nuestro grupo de trabajo al desarrollo de un interfaz gráfico, Copérnico [Blanco 95, González-Calero 96, González-Calero (en prensa)], que simplifica la creación, visualización, depuración y mantenimiento de bases de conocimiento desarrolladas en Loom.

4.5 Resumen

A lo largo de este capítulo hemos presentado el modelo de sistema de ayuda inteligente que proponemos en este trabajo, modelo que ejemplificaremos mediante los sistemas Argos y Aran, descritos en los siguientes capítulos.

En primer lugar hemos presentado los objetivos iniciales que debe cumplir el modelo de sistema de ayuda inteligente. Estos objetivos son: que sea aplicable a entorno reales, que presente un comportamiento adaptable al usuario y que tenga en cuenta aspectos pragmáticos de implementación. Para cumplir dichos objetivos proponemos un modelo de sistema de ayuda inteligente cuyas características clave son: activación del asistente por el usuario, proporción de ayuda basada en acceso a documentación, adaptación mediante un modelado explícito del usuario y la utilización de una interfaz multimodal. La asistencia proporcionada debe seguir nuestro enfoque del doble propósito de la ayuda, proporcionando la información que permita completar la tarea en curso, a la vez que promueve una mayor comprensión de la estructura del sistema.

A continuación hemos tratado los dos aspectos en los que se basa la viabilidad de implementación de este modelo: la integración de tecnologías y la reutilización de información. Las funcionalidades de ayuda se implementan mediante la integración en el sistema de diferentes tecnologías bien conocidas y ampliamente probadas en distintas áreas. Estas tecnologías son: la recuperación de información, el hipertexto, el modelado de usuario y la representación explícita del conocimiento. La

combinación de estas técnicas permitirá facilitar tanto el acceso a la información como su comprensión por diversos tipos de usuarios.

Con el objetivo de disminuir el coste de desarrollo también proponemos la reutilización de distintos tipos de información. Esto implica desde la reutilización de documentación electrónica, hasta la reutilización -parcial- de bases de conocimiento o de modelos de usuario usados previamente en otros sistemas. En esta tesis no se pretende dar una solución general a la reutilización de información, ya que es un proceso que depende de cada dominio. No obstante, hay prácticas que contribuyen a su simplificación, como por ejemplo la utilización de un formalismo estándar y de gran poder expresivo para la representación explícita del conocimiento.

Finalmente hemos descrito brevemente cada una de las tecnologías integradas en el modelo, destacando las técnicas más relevantes para este trabajo y su aplicación en el proceso de construcción de los sistemas de ayuda inteligentes. Los detalles de estas técnicas, así como la información específica que se reutiliza, se concretarán posteriormente en la implementación de los asistentes desarrollados, Argos y Aran.

CAPÍTULO 5

EL SISTEMA ARGOS

5.1 Introducción

El objetivo principal de esta tesis es la construcción de sistemas de ayuda para aplicaciones informáticas en dominios reales. Para la ejemplificación de nuestro enfoque se ha elegido el sistema operativo Unix, debido a que es un entorno amplio y complejo, con usuarios de muy diversos tipos y en el que existe una extensa documentación disponible electrónicamente. El sistema Argos se ha concebido como un sistema de ayuda inteligente (un asistente) para el sistema operativo Unix.

Las dos ideas clave en las que se basa el modelo de sistema de ayuda propuesto son: la integración en el mismo sistema de diferentes tecnologías y la reutilización de información. El uso de tecnologías estándar, junto con la reutilización de información a distintos niveles (información textual y modelos anteriormente utilizados) permite el desarrollo de asistentes efectivos para dominios complejos y reduce sus costes de desarrollo. En Argos se han integrado la recuperación de información, el hipertexto y el modelado de usuario. Argos utiliza como principal fuente de información la extensa documentación en formato electrónico que se proporciona con el sistema operativo. El modelo de usuario es una aportación basada en otros modelos utilizados por diversos sistemas en dominios similares.

En la concepción de Argos, se tuvo presente como objetivo fundamental la definición de un marco de aplicación concreto sobre el que poder investigar la utilización e integración de diferentes tipos de técnicas relacionadas con la recuperación de información o el modelado de usuario, para el desarrollo efectivo de sistemas de ayuda [Buenaga 93a, Fernández-Manjón 93]. De esta forma, el sistema se ha desarrollado con vistas a presentar una utilidad práctica inmediata, teniendo presentes criterios como los relacionados con la interacción hombre-máquina y la facilidad de utilización por el usuario [Maddix 90]. Por otra parte, el sistema Argos se ha concebido como un sistema modular que permite la modificación o la inclusión de nuevos módulos. No se ha tratado únicamente de crear un sistema concreto que funcione para una determinada aplicación o entorno, sino seguir una aproximación más amplia, aplicable a otros productos del dominio del software que tengan unas características similares. De hecho los principios utilizados en Argos han sido

adaptados al desarrollo de sistemas similares en otros dominios [Fernández Chamizo 95, González Calero (en prensa)].

En este capítulo comenzamos analizando las características del dominio de aplicación, el sistema operativo Unix. A continuación se describen los objetivos, funcionalidades y estructura del sistema de ayuda Argos. En puntos sucesivos se analizan con detalle las técnicas empleadas en su construcción. Finalmente se presenta un breve resumen.

5.2 El dominio de aplicación: el sistema operativo Unix

En el diseño de Unix los objetivos principales han sido la versatilidad y la potencia, considerándose menos importantes otros factores como la facilidad de aprendizaje [Sutcliffe 87, Duffy 92]. Esta dificultad de uso y aprendizaje unida a su amplia utilización, tanto en entornos académicos como industriales y con usuarios muy diversos, hace deseable la disposición de sistemas de ayuda. Su amplia disponibilidad y su arquitectura abierta ha provocado que sea un dominio de prueba comúnmente utilizado para la ejemplificación de distintas aproximaciones para ayudar al usuario [Breuker 90, Winkels 92a, Hecking 88, Wilensky 89, Kuah 92, So 94]. Es por tanto un marco de aplicación real y sobre el que existen diversas experiencias previas, con las que se puede contrastar este enfoque y de las que se puede reutilizar información.

A continuación se tratan las características del interfaz de Unix. Posteriormente se presenta una breve perspectiva de los diversos intentos para simplificar el aprendizaje y uso de este sistema operativo. Finalmente se analizan las características de la amplia documentación en formato electrónico que se proporciona en el sistema.

5.2.1 El interfaz de UNIX

El sistema Unix fue desarrollado inicialmente por programadores para su uso particular de modo que han primado objetivos como la potencia y la versatilidad frente a otros objetivos tales como la facilidad de uso por usuarios no expertos.

> *"The designers of Unix have emphasized the principles of simplicity and elegance. (...) Some benefits of this design are that is elegant, concise, and even easy to learn and use -for matematically-oriented programmers.The drawback, however, is that it is dreadfully difficult for the novice to learn." [So 94]*

El interfaz del sistema operativo está basado en la ejecución de órdenes (*command-driven*). Cada orden (también llamada herramienta, utilidad o comando) habitualmente está implementada mediante un programa independiente que lleva a cabo un único propósito. El usuario interacciona con Unix mediante un interprete de órdenes llamado *shell*, cuya función principal es ejecutar alguna de las cientos de utilidades que proporciona el sistema operativo [Coffin 90]. La mayoría de las

órdenes de Unix son programas pequeños, con un formato sucinto de entrada y salida, y algorítmicamente directas. El sistema también proporciona mecanismos (p.e. redirección de entrada/salida o encauzamiento de órdenes -pipe-) para combinar estas órdenes de propósito simple creando utilidades más complejas de forma sencilla. De esta forma se obtiene una interacción concisa e incluso fácil de aprender y de usar, pero sólo para programadores expertos. Por otra parte, presenta un inconveniente común a los lenguajes de órdenes: es difícil de aprender para los usuarios noveles [Boy 91]. Este problema es más destacable en el caso del Unix porque al ser un sistema de propósito general dispone de un conjunto muy amplio y diverso de órdenes que además tienen nombres crípticos [Doane 91].

5.2.2 Soluciones para facilitar el uso y aprendizaje del sistema Unix

El hecho de que Unix sea un sistema amplio y complejo, unido a otras características como que es un sistema abierto, de uso muy extendido, con aplicaciones diversas y diferentes tipos de usuarios, ha provocado que sea un dominio muy utilizado para el desarrollo y prueba de distintos enfoques para ayudar al usuario [Breuker 90]. Con este propósito se han utilizado diversas aproximaciones, que van desde una mejora únicamente de la interacción mediante modificaciones en el intérprete de órdenes, hasta intentos más ambiciosos como el desarrollo de sistemas activos de ayuda o de tutores inteligentes. A continuación vamos a presentar brevemente dos de estos enfoques.

5.2.2.1 Mejora de la interacción

En la ayuda a los usuarios mediante la mejora de la interacción se pueden distinguir dos líneas principales:

- *Mejora del interprete de órdenes (shell).* El interprete de órdenes se ha sustituido por otros que añaden nuevas funcionalidades como, por ejemplo, un lenguaje de programación o corrección automática de errores simples de escritura [Coffin 90, tcsh 91]. La mayor limitación que presentan estos sistemas es que se proporciona una ayuda exclusivamente sintáctica y su uso sigue siendo complejo para usuarios no experimentados [Boy 91].

- *Interfaces gráficos de usuario (GUI).* La interacción se simplifica con la representación gráfica de los diversos elementos del sistema (p.e. archivos, directorios) mediante elementos gráficos (iconos y ventanas) y su manipulación directa [Maddix 90]. De hecho actualmente, el sistema X Window System forma parte de muchas de las distribuciones comerciales de Unix. Aunque estas mejoras facilitan la operación, sobre todo a los usuario noveles y ocasionales, un problema destacable es que en la interfaz no se dispone de forma gráfica de todas las posibilidades de manipulación, por lo que es necesario seguir realizando operaciones desde el interprete de ordenes. Prototipos como Ishell [Borg 90], tratan de ampliar el número de operaciones disponibles desde el entorno gráfico.

5.2.2.2 Sistemas de ayuda

El objetivo de los sistemas de ayuda es proporcionar herramientas al usuario para que pueda resolver los problemas que encuentre en la realización de sus tareas con el sistema [Duffy 92, Kearsley 88]. En el dominio del Unix se han realizado diversos sistemas para proporcionar ayuda que se pueden clasificar en dos grandes grupos:

- *Sistemas de ayuda en línea (on-line) o sistemas pasivos de ayuda.* Estos sistemas ayudan al usuario mediante un formato de solicitud o pregunta y la obtención de una respuesta. La investigación en sistemas de ayuda en línea tiene diferentes enfoques, entre los que se puede destacar la utilización de sistemas expertos [Mamone 90], el uso de técnicas de documentación [Duffy 89], uso de técnicas de recuperación de información [Maarek 91a, Maarek 91b, Prieto-Diaz 87, Prieto-Diaz 91], la utilización de técnicas de organización y acceso de la información tales como hipertexto y multimedia [Barret 89, Conklin 87, Boyle 94], la generación y comprensión del lenguaje natural [Wilensky 84, Wilensky 89, Pilkington 92, Quilici 89] o la utilización de un modelado del usuario para crear sistemas adaptativos de ayuda [Jones 88, Nessen 89]. Algunos de los sistemas incluyen varias de estas aproximaciones, p.e., combinando técnicas de tratamiento del lenguaje natural y de modelado de usuario [Wilensky 89], o como el sistema Metadoc [Boyle 94] que combina modelado de usuario e hipertexto. Un problema destacable de este enfoque es el formato de petición-respuesta, ya que supone que el usuario debe tener unos ciertos conocimientos previos y debe tomar la iniciativa de la interacción. Un usuario novel puede saber tan poco que no sea capaz de hacer la pregunta adecuada con la terminología precisa, por lo que no obtendría la información deseada. Por tanto hay que proporcionar mecanismos que simplifiquen la interacción con el usuario haciéndola más efectiva. Por otra parte, como el asistente no ofrece ayuda al usuario sin que éste la pida, se pueden estar utilizando procedimientos muy ineficientes, mientras se permanece ajeno a otros procedimientos más efectivos por los que podría preguntar. Para evitarlo simultáneamente con la información de ayuda se debe ampliar, en la medida de lo posible, el conocimiento que tiene el usuario sobre la aplicación.

- *Sistemas activos de ayuda.* Un sistema de ayuda activa tiene en cuenta las acciones del usuario y toma la iniciativa para proporcionar consejos cuando lo considera oportuno. No hay mucha investigación en sistemas activos de ayuda en general, ni en particular en el dominio del UNIX [So 94]. La mayor parte de la investigación actual se centra en el desarrollo de prototipos focalizados en los aspectos cognitivos tales como el modelado de usuario, el reconocimiento de planes o la combinación de ayuda con aspectos educativos [Quilici 89, Jones 88, Breuker 90, Winkels 92a, Kuah 92]. Algunos sistemas como el Sinix Consultant [Hecking 88] no se plantean únicamente como un sistema activo de ayuda, sino que tienen como objetivo específico su integración con un sistema de ayuda pasiva. Un problema que cabe destacar en estos sistemas activos es que hay que diagnosticar cuándo y qué tipo de problemas esta teniendo el usuario para poder proporcionar la ayuda adecuada. Esto en dominios tan amplios y dinámicos como el Unix y en el cual

los usuarios pueden tener intereses tan diferentes, es un proceso muy costoso. Presenta dificultades añadidas cuando se quiere desarrollar un asistente de este tipo para aplicaciones comerciales que ya existen, como es nuestro caso, debido a que es difícil obtener la información completa sobre el estado del sistema. Esto en ocasiones lleva a tener que emular el producto para el que se quiere producir la ayuda, p.e. la emulación del sistema de mail en el prototipo de EUROHELP [Breuker 90], o la emulación del propio Sinix en el Sinix Consultant [Hecking 88], lo que es muy complejo y aumenta el coste.

Además existen otras aproximaciones para proporcionar ayuda y aprendizaje que se han descrito en el capitulo anterior y tienen su reflejo en el dominio del Unix. Por ejemplo, se han desarrollado *tutores inteligentes* como Unix Tutor [Wang 92]. La principal diferencia destacable de los tutores respecto a los sistemas de ayuda es que en ellos el usuario no realiza su propio trabajo, sino los ejercicios propuestos por el tutor. Otra aproximación son los *agentes* que tratan de ampliar el poder expresivo del interfaz, de modo que un usuario sólo tenga que preocuparse de especificar cuál es su objetivo sin tener que preocuparse por cómo llevarlos a cabo o monitorizar los posibles problemas en su realización [Etzioni 92, Etzioni 93]. Una aproximación mixta que aúna la mejora de la interacción y la ayuda al usuario son los *interfaces inteligentes* [Jerrams-Smith 89].

El presente trabajo se centra en proporcionar ayuda en línea al usuario cuando éste la solicita. El sistema Argos basa esta ayuda en proporcionar un acceso eficiente a la documentación electrónica existente en el sistema. En el siguiente apartado se analizan las características de esta documentación.

5.2.3 Caracterización de la documentación de Unix

El sistema operativo Unix proporciona en la distribución estándar una información muy extensa en formato electrónico sobre las órdenes disponibles en el sistema. Sin embargo, para una información tan amplia, sólo se proporcionan medios básicos de acceso, como la orden *man* o *xman*, y una estructuración muy limitada. Estas circunstancias unidas a la especificidad del vocabulario utilizado y que el foco de atención sea la documentación de las órdenes y no cómo utilizarlas en la realización de tareas, provoca que el manual electrónico resulte difícil de manejar y comprender para muchos usuarios [Coffin 90, Duffy 92]. Por tanto es muy interesante proporcionar medios que simplifiquen el acceso, la búsqueda y la comprensión de esta información [Fernández-Manjón 94, Draper 84].

El acceso básico a la documentación lo proporciona la orden estándar de Unix *man* que presenta la información relacionada con una orden. La pregunta es siempre "¿Cómo se usa la orden *x*?" o "¿Qué hace la orden *x*?, formulada mediante *man x* y la respuesta es la página de manual de la orden *x*. Para acceder a la ayuda es necesario conocer el nombre de la orden (*xman* proporciona un listado de los nombres de las órdenes pero no de su propósito). Además las páginas del manual que se obtienen como respuesta están concebidas para usuarios con una cierta experiencia pudiendo plantear problemas de comprensión para usuarios noveles [Draper 84].

Esta documentación se agrupa en ocho secciones: 1) órdenes y utilidades de usuario; 2) llamadas de sistema; 3) funciones de biblioteca en lenguaje C; 4) dispositivos físicos e interfaces de red; 5) formatos de archivos del sistema; 6) juegos y demostraciones; 7) miscelánea; y 8) órdenes y utilidades de administración. Esta organización se ha realizado considerando únicamente un determinado criterio, p.e., el nivel de permiso necesario para la ejecución (usuario normal o administrador), si se puede utilizar directamente como orden interactiva o si es necesaria incluirla en un programa (orden normal o llamada al sistema). En cada sección hay también un documento *intro*, en el que se da una idea del contenido de esa sección; si existe alguna otra categorización de los documentos (p.e. el nombre de los documentos de la primera sección relacionados con la comunicación con otros sistemas finalizan en .1C), así como el nombre y una descripción corta de cada una de las utilidades. Además en cada archivo del manual aparece una sección SEE ALSO que cita de una forma escueta cuál es la información relacionada.

No es difícil comprobar que para un usuario normal la organización de la información proporcionada es insuficiente. Por ejemplo, un usuario final no programador que busque información y conozca esta estructura en secciones, puede descartar algunas de ellas (p.e. administración, llamadas al sistema, funciones en C), pero aun así sólo en la primera sección hay documentación sobre alrededor de quinientas órdenes (el número depende ligeramente de la versión, en nuestro caso, SunOS 4.1.X, entre 497 y 524). Además estas órdenes tienen nombres crípticos de modo que es difícil relacionarlas con la tarea a realizar. Por tanto, es primordial proporcionar medios efectivos de acceso a esos documentos, que simultáneamente simplifiquen su comprensión y asimilación.

El sistema Argos simplifica el acceso a esta documentación mediante un interfaz que permite las consultas en lenguaje natural y que utiliza técnicas de hipertexto. El sistema Aran (que describiremos en un capítulo posterior), además de las funcionalidades proporcionadas por Argos, incluye una base de conocimiento sobre Unix que estructura mejor la información simplificando su acceso y su comprensión.

5.3 Planteamiento y diseño del sistema Argos

El sistema Argos es un sistema de ayuda (asistente) a la utilización del conjunto de órdenes del sistema operativo UNIX [Fernández-Manjon 93, Fernández-Manjon 94]. Ayuda al usuario a encontrar cuál es la información relevante a su necesidad en cada momento, a partir de la extensa documentación existente en el entorno. Tratamos a continuación los objetivos marcados en el desarrollo del sistema, su funcionalidad y su arquitectura.

5.3.1 Objetivos del sistema

En la concepción del sistema Argos se han contemplado como objetivos principales los tres siguientes:

1. *Definición de un sistema práctico de ayuda en un entorno real.* Para la consecución de un sistema de ayuda efectivo, orientado a su utilización práctica por un conjunto de usuarios significativo, se han integrado diversas técnicas de las propuestas en el capítulo 3. Concretamente en Argos se integra la *recuperación de información* [Salton 89], el *hipertexto* [Smith 88, Balasubramanian 93] y el *modelado de usuario* [Kobsa 89, Kobsa 94]. También se ha prestado una atención especial a los criterios de *interacción hombre-máquina* [Maddix 90], para obtener un interfaz intuitivo y de uso sencillo.

2. *Definición de una aproximación generalizable.* El sistema Argos se ha desarrollado para el dominio específico del conjunto de utilidades que proporciona el sistema operativo como órdenes y que se encuentran documentadas en la sección 1 del manual [Sun 89]. Sin embargo, la aproximación seguida en Argos resulta de fácil adaptación a otros entornos y aplicaciones en los que exista documentación disponible. En concreto, adaptaciones de la aproximación seguida en Argos se han realizado a la biblioteca de clases de Smalltalk del entorno VisualWorks [González Calero (en prensa)].

3. *Definición de un marco de aplicación concreto orientado a la investigación.* Argos ha sido concebido como un marco de aplicación concreto, sobre un dominio concreto (p.e. la documentación de UNIX) orientado al estudio y evaluación de la integración de diferentes tipos de técnicas en el proceso de recuperación de información [Fernández-Manjón 93, Buenaga 93a, Buenaga 93b]. Argos, de hecho, proporciona la definición del marco concreto de aplicación de los sistemas Aran [Fernández-Manjón 95a, Fernández-Manjón 95c] y Ares [Buenaga 95, Buenaga 96]. Estos sistemas se centran en la inclusión de una representación explícita del dominio, y en el uso de técnicas de procesamiento de lenguaje natural, respectivamente.

Estos tres aspectos del sistema son tratados en apartados siguientes y pueden encontrarse detalles adicionales en la bibliografía [Fernández-Manjón 93, Buenaga 93a, Buenaga 93b, Arroyo 93, Fernández-Manjón 94, Buenaga 94, Fernández-Manjón 95a, Fernández-Manjón 95c, Díaz 95, Buenaga 95, Cigarrán 95, Buenaga 96, Cigarrán 96]. El punto 3 es el más importante para nuestro trabajo y de él trataremos con más detalle en este capítulo y en siguientes.

5.3.2 Funcionalidades del sistema

En la versión actual de Argos el usuario se encuentra trabajando en el entorno X Window en una ventana de órdenes. El sistema es accesible como una aplicación más. En la figura 5.1 podemos ver el interfaz del sistema (concretamente de la versión 4). Las principales funcionalidades implementadas por el sistema y los criterios utilizados para su definición son:

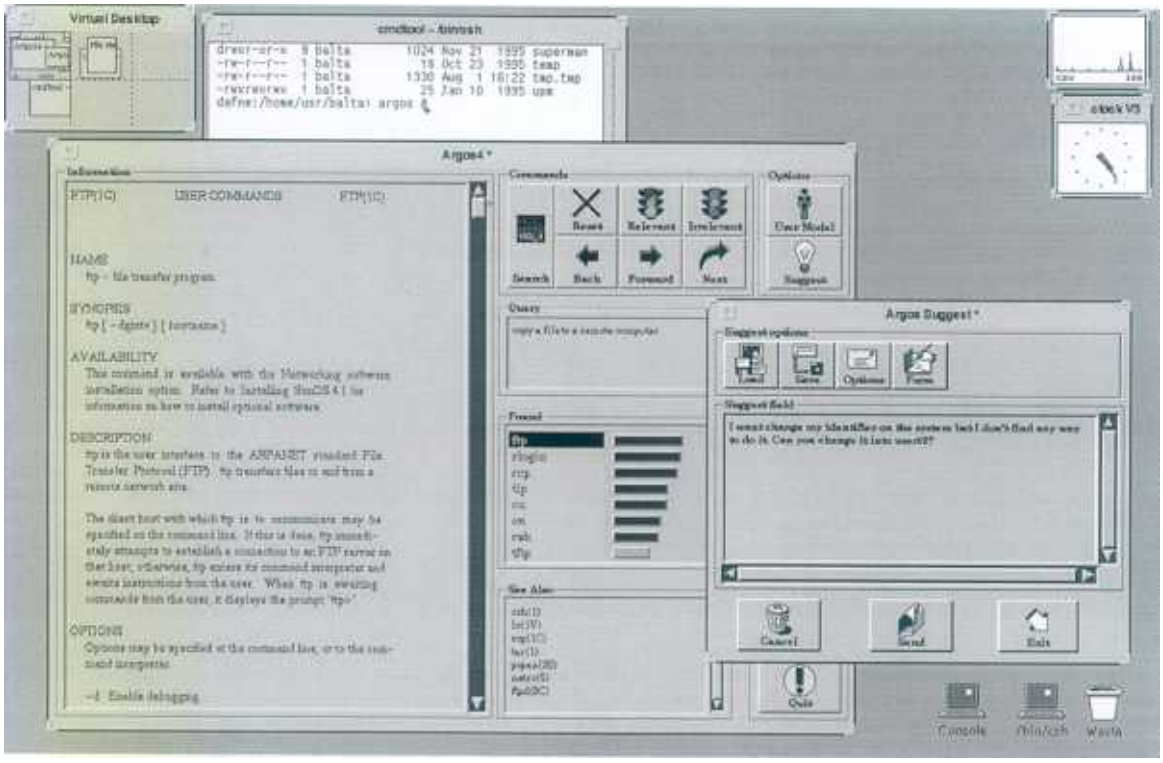


Figura 5. Interfaz del sistema Argos en el entorno X Window

Especificación de consultas en lenguaje natural. El usuario utiliza el lenguaje natural (Inglés) como medio para especificar su necesidad. De esta forma, cuando el usuario se encuentra ante una determinada necesidad, puede formular (en el panel *Query*) consultas como "communicate with other user", "copy a file to a remote machine", "get the name of the current directory", etc. Conviene subrayar que el lenguaje natural se utiliza sólo para expresar la necesidad del usuario de forma declarativa. Las órdenes en sí mismas (p.e. la orden de comenzar la búsqueda, la de presentar un documento o la de finalizar) son realizadas mediante la interacción basada en otros tipos de lenguajes (p.e. botones y menús).

Interacción basada en un lenguaje de manipulación directa. Bajo el término *lenguaje de manipulación directa* (*direct manipulation language*) queda englobada la interacción basada en la utilización de botones, iconos, ventanas y ratón como dispositivo señalizador. La asignación de los lenguajes a los diferentes elementos de interacción, buscando su mejor adecuación, se ha realizado siguiendo criterios de *Interacción Hombre Máquina* (*Human Computer Interaction, HCI*) [Maddix 90]. De esta forma, la solicitud de ejecución de órdenes concretas se realiza mediante botones (p.e. la orden de comenzar la búsqueda mediante *Search* y la de fin de ejecución mediante

Quit). También, la representación de la relevancia de los documentos recuperados se realiza mediante barras gráficas de distinta longitud y color.

- *Presentación de la documentación en lenguaje natural.* La selección de las órdenes relevantes a la necesidad del usuario se basa en el cálculo de la similitud entre la necesidad del usuario y los documentos asociados a las órdenes. En la ventana *Found* se presentan los nombres de las órdenes con una representación gráfica de sus valores de relevancia. En la ventana *Information* el texto de la orden más relevante, o de las seleccionadas por el usuario mediante el ratón. El orden de presentación de la información, obtenido mediante el cálculo de similitud, puede verse modificado en función del contenido del modelo de usuario.

- *Funciones de navegación basadas en hipertexto.* El proceso formado por la secuencia de consultas y la presentación de la información por parte del sistema, puede concebirse como uno de navegación por un *hipertexto* (*hypertext*) [Conklin 87, Smith 88, Nielsen 90, Allan 95], generado automáticamente. En este hipertexto los *nodos* son los ítems de información del manual (p.e. cada documento asociado a cada orden) y los *enlaces* quedan definidos por su secuencia de presentación. Con este fin se incluyen funciones de navegación especialmente adecuadas para este tipo de sistemas, como *Back, Forward* y *Next*. El cálculo de la similitud de los documentos con las consultas y el uso de la información del modelo de usuario, constituyen la base de la generación de enlaces dinámicos. La información sobre órdenes relacionadas incluidas en la sección *See also* del final de cada documento proporciona enlaces estáticos adicionales.

- *Realimentación.* En Argos se incluyen funciones orientadas a la introducción de *realimentación* (*feedback*) en el proceso de recuperación [Salton 89]. La realimentación se basa en la reformulación de consultas mediante la utilización de documentos proporcionados por el sistema al usuario y proporciona importantes mejoras en la efectividad del proceso de recuperación en su conjunto. En concreto, si parte del texto que aparece en *information* es especialmente relevante, el usuario puede seleccionarlo mediante el ratón, utilizándose este fragmento del texto como una nueva consulta. Por otra parte, cuando la documentación que se presenta en *Information* es irrelevante o muy relevante al problema del usuario, éste puede indicarlo (mediante los botones *Relevant* e *Irrelevant*). Esta información relativa a la relevancia es utilizada en el procesamiento de siguientes consultas relativas a una misma necesidad o tema de búsqueda. El botón *Reset* sirve para indicar un cambio en la necesidad o tema de búsqueda.

- *Modelo de usuario inspeccionable.* El modelo de usuario de Argos centraliza todos los datos que se usan para adaptar la información que presenta al interés y nivel de experiencia del usuario [Kobsa 94]. El usuario puede inspeccionar e incluso modificar el modelo, si considera que el proceso de adquisición de alguna de las características representadas no ha sido acertado. Este es un aspecto interesante ya que facilita a los utilizadores un acceso simple a los datos (mediante el botón *User model*) que el programa

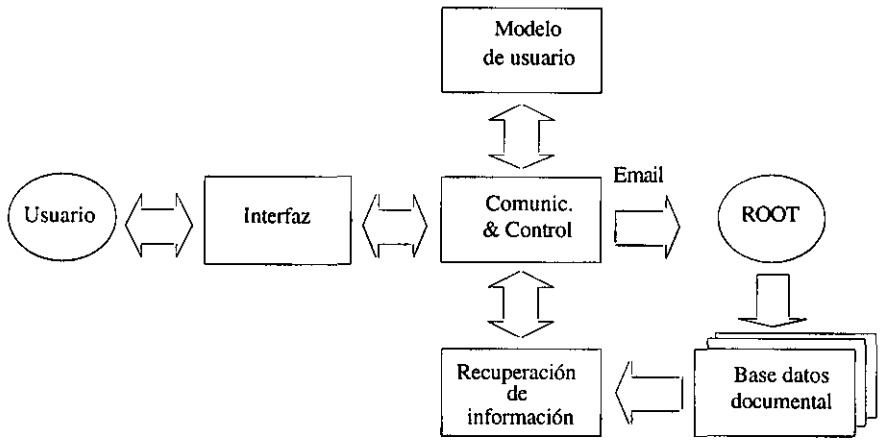


Figura 5.2 Estructura de módulos del sistema Argos

utiliza para adaptarse a ellos, de modo que pueden comprender mejor su funcionamiento y disponen de un mayor control sobre el asistente [Boyle 94].

- *Adición de información textual.* Además los usuarios pueden incluir información específica, adicional a la existente en el manual de UNIX. Información que puede resultar de mayor interés para el grupo de trabajo del usuario y más adecuada a su entorno de trabajo, o simplemente, más concisa. Tras la edición de esta información mediante la orden *suggest* (figura 5.1), los documentos son enviados por Argos, mediante correo electrónico, al administrador del sistema para su supervisión. De esta forma, la base de datos documental utilizada por Argos está formada por los ficheros del manual y las sugerencias redactadas por los usuarios (y admitidas por el supervisor), obteniéndose de este modo capacidades propias de los *entornos cooperativos supervisados* [Fernández-Manjón 93].

5.3.3 Estructura del sistema

Argos consta de cuatro módulos (figura 5.2): *interfaz*, que se ocupa de la interacción con el usuario; *modelo de usuario*, que contiene la información disponible sobre el usuario; *recuperación de información*, que implementa las funciones de indexación de los documentos y cálculo de la similitud; y un módulo de *comunicación y control*, que proporciona la adecuada integración del resto de los módulos. La base de datos documental está formada por los archivos de texto del manual de UNIX y los documentos añadidos por los usuarios y supervisados por el superusuario (root). En su diseño se ha tratado en todo momento de mantener una estructura modular de forma que se pueda modificar o sustituir alguna de sus partes con facilidad.

- *El módulo de recuperación de información.* El módulo de recuperación de información (RI) realiza las órdenes de búsqueda que recibe del módulo de control del sistema. El contenido esencial de las órdenes de búsqueda que

recibe el módulo de RI es una lista de palabras con pesos asociados, confeccionada a partir de la consulta del usuario y la información existente en el modelo del usuario. Esta lista de palabras caracteriza la necesidad del usuario y es la utilizada para realizar el cálculo de la similitud de los documentos en la base de datos con respecto a la necesidad del usuario.

- *Modelo del usuario*. El módulo del modelo de usuario (MU) es un módulo cuyo objetivo es adecuar la información que se presenta a cada usuario concreto. Para ello utilizamos un modelo de usuario incremental e inspeccionable (el usuario puede incluso desactivarlo) [Kobsa 92]. La información utilizada en el MU proviene tanto de la interacción con el usuario (a través de sus criterios de relevancia o feedback mencionados en párrafos anteriores), como de su clasificación en función de su interés en el uso del sistema y de su grado de experiencia. Estos datos se utilizan para modificar las búsquedas y el orden de presentación de los documentos [Fernández-Manjón 93, Fernández-Manjón 95b].

- *Interfaz*. En el diseño del interfaz gráfico se ha tenido en cuenta la sencillez de manejo y la uniformidad. Es importante evitar la sobrecarga cognoscitiva del usuario y la dificultad de uso del sistema [Downton 91, Fernández-Manjón 94]. En el diseño del interfaz de Argos se han tenido presentes criterios de interacción hombre-máquina y elementos tomados del diseño de interfaces para sistemas de hipertexto. Detalles adicionales de implementación se pueden encontrar en [Arroyo 93, Cigarrán 96].

En los siguientes apartados proporcionamos más detalles acerca de la implementación de estos módulos.

5.4 El módulo de recuperación de información

El módulo de Recuperación de Información (RI) lleva a cabo las órdenes de búsqueda de documentos que recibe del módulo de control del sistema. El contenido esencial de estas órdenes de búsqueda es una lista de pares término-peso, que se confecciona a partir de la consulta del usuario y se enriquece con la información existente en el modelo del usuario (concretamente en la memoria de corto plazo, en la que se almacenan los pares término-peso procedentes de las preguntas anteriores). Esta lista de palabras caracteriza la necesidad del usuario y es la utilizada para realizar la búsqueda de los documentos relevantes en el corpus (manual de Unix).

Este módulo de RI implementa, esencialmente, un algoritmo de cálculo de similitud entre consultas y documentos basado en el modelo del espacio vectorial. Se consideran como documentos más relevantes aquellos cuyas descripciones sean más parecidas a la de la solicitud del usuario. Se computa la similitud de todos los documentos con la consulta para obtener una lista con los documentos ordenados decrecientemente por su valor de similitud. Los documentos más relevantes de esta lista se mostrarán en el interfaz de usuario como resultado de la búsqueda.

Como la elaboración de consultas efectivas es una tarea difícil para el usuario, en Argos el modelo vectorial se complementa con diversos mecanismos que mejoran la efectividad del proceso. De esta forma se mejora la representación de las consultas y de los documentos mediante el reemplazamiento de algún término de su caracterización por otro mejor y la introducción o eliminación de términos. Estos mecanismos pueden ser previos a la realización de la consulta, como son la utilización de pesos de términos, el uso de una lista de parada y la obtención de raíces de las palabras, o posteriores a ella, como es la reformulación de la consulta mediante realimentación. En siguientes apartados se tratan con mayor detalle estos aspectos.

Esta funcionalidad de Argos se corresponde con la de un sistema de recuperación de texto. El esquema clásico de estos sistemas se complementa en Argos, con la información adicional procedente del modelo del usuario (figura 5.3). En la implementación de las funcionalidades del módulo de recuperación de información, se ha partido de otros sistemas previos de recuperación de texto [Cigarrán 95, Díaz 95], que se han adaptado para incluir el modelo de usuario.

5.4.1 Método de indexación y cálculo de similitud

Antes de que se realicen las búsquedas de documentos, en la que el cálculo de la similitud entre los documentos y una consulta determinada constituye la base para la selección de los documentos relevantes, se hace un análisis inicial de los textos existentes en la base documental (páginas del manual de UNIX). En este proceso de indexación de los documentos se genera un archivo invertido de índices, mediante el cual se relaciona un término con los documentos en los que aparece. Además se aprovecha para realizar una caracterización de cada documento mediante sus términos más representativos, que será utilizada en el proceso de realimentación por relevancia (descrito en el siguiente apartado).

En el desarrollo del módulo de RI [Cigarrán 95] se ha optado por la utilización de un conjunto de técnicas que son sencillas de implementar a la vez que efectivas [Salton 89, Araya 90]. Los elementos básicos utilizados en el proceso de los documentos y de las consultas son los siguientes:

- *Uso de lista de parada (stoplist)* para la eliminación de palabras de alta frecuencia de aparición (como "and", "or", "but" o "of"), que no tienen contenido semántico y por tanto no ayudan a caracterizar los documentos. La lista utilizada ha sido tomada de [Fox 92]. Originalmente contenía 429 palabras, pero ha sido adaptada al dominio del Unix. Esto ha supuesto la eliminación de esta lista de palabras como *who* o *which*, que normalmente son palabras vacías, pero no pueden considerarse vacías en este dominio, debido a que son nombres de órdenes del sistema.

- *Uso de una rutina de eliminación de sufijos (stemming).* Se obtiene la raíz de las palabras, eliminado las terminaciones de los plurales, gerundios, los pasados de las formas verbales, etc., con lo que se obtiene un *término*. Se

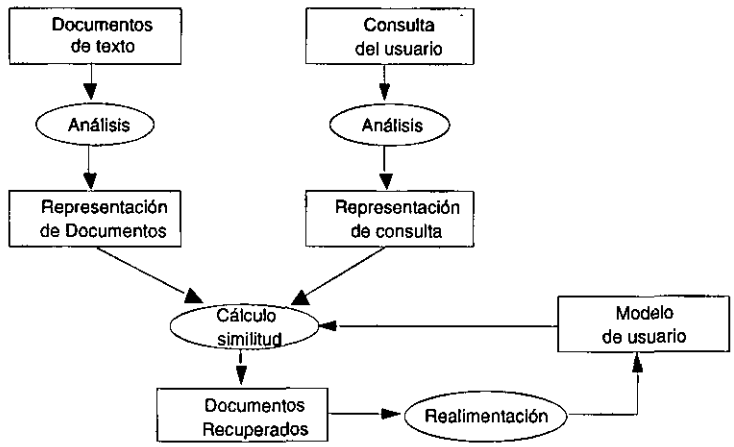


Figura 5.3: Proceso de recuperación de documentos en Argos

utiliza el algoritmo de extracción de raíces de Porter [Frakes 92], basado en un autómata de estados finitos, con 88 reglas de eliminación de sufijos.

- *Peso de los términos.* Una vez aplicada la lista de parada y el algoritmo de extracción de raíces a las palabras aparecidas en el manual, se obtiene un conjunto de m términos de indexación. Estos términos son los que se utilizan para representar los documentos y las consultas. En concreto, m = 6390 términos para la sección 1 del manual de referencia. Para cada término $term_i$ del conjunto obtenido, se computa en cuantos documentos aparece, df_i, y se le asigna un peso en la colección, w_i, mediante la expresión:

$$w_i = log_2 (n / df_i)$$

en donde n es el número de documentos existentes en el manual. En concreto, n = 524 para la sección 1 del manual de referencia.

- *Representación de los documentos.* Para cada documento d_j, se computa la frecuencia de aparición de cada término $term_i$ en él, tf_{ji}, y el peso del término en el documento, wd_{ji}, mediante la expresión:

$$wd_{ji} = tf_{ji} \cdot w_i$$

Cada documento queda representado, o indexado, por un vector de dimensión m, con los pesos asignados a cada uno de los términos de indexación:

$$\langle wdj1, wdj2, \cdots, wdjm \rangle$$

- *Representación de las consultas.* Para el procesamiento de las consultas se utiliza la lista de parada y el algoritmo de extracción de raíces, obteniendo los términos de indexación aparecidos en ellas, de forma análoga a los documentos. Para una consulta q_k se calcula la frecuencia de aparición de

cada termino de indexación $term_i$ en ella, tfq_{ki}, y se le asigna a cada término un peso en la consulta:

$$wqki = tfqki \cdot wi$$

De esta forma, la consulta queda representada por un vector de dimensión m, con los pesos asignados a cada uno de términos de indexación:

$$\langle wqk1, wqk2, \cdots, wqkm \rangle$$

- *Función de similitud.* El valor de la similitud entre un documento d_j y una consulta q_k se obtiene mediante la fórmula del coseno del ángulo entre ambos vectores:

$$sim(d_j, q_k) = \frac{\sum_{i=1}^{m} wd_{ji} \cdot wq_{ki}}{\sqrt{\sum_{i=1}^{m} wd_{ji}^2 \cdot \sum_{i=1}^{m} wq_{ki}^2}}$$

En el interfaz los valores numéricos de relevancia calculados mediante esta función se presentarán como barras gráficas para simplificar la comprensión por parte del usuario. Esta función de similitud se utiliza ampliamente en el campo de la RI y, en media, proporciona un buen criterio de similitud [Salton 88, Araya 90].

5.4.2 Realimentación en Argos

Una vez efectuada una búsqueda, el estudio del conjunto de documentos recuperados se puede utilizar para construir una nueva consulta, que se adapte mejor a las necesidades del usuario, y que por tanto recupere un mayor número de documentos relevantes. El uso de esta información para hacer una reformulación o expansión de la consulta se denomina realimentación (*feedback*).

Esta reformulación de la pregunta se ha mostrado como una técnica muy efectiva, sobre todo para usuarios con un conocimiento limitado del vocabulario utilizado en el dominio. También es útil para las situaciones en las que el usuario no sabe expresar su necesidad o no tiene una idea clara de lo que está buscando (aunque por otra parte si es capaz de reconocer fácilmente la información que busca una vez que se le presenta) [Araya 90].

5.4.2.1 Métodos de realimentación

En Argos se diferencian dos métodos de realimentación, según el tipo de intervención del usuario en el proceso y el tratamiento que se realice de los documentos. La realimentación puede ser:

- *Realimentación mediante texto.* Se produce cuando el usuario considera que uno o varios párrafos de un documento que se le está presentando, describen o complementan la descripción de su necesidad, y quiere obtener más información relacionada. En este caso el usuario puede seleccionar dicho texto con el ratón y realizar una nueva búsqueda (mediante Search). Argos considera este texto como complemento de la especificación de la pregunta del usuario para la siguiente búsqueda. Es siempre una realimentación positiva que incluye nuevos términos en la consulta o incrementa el peso de los ya existentes. Esta funcionalidad ha demostrado ser muy útil en sistemas de acceso a información [TMC 91]. Es muy adecuada en el dominio del Unix debido a que hay órdenes, que realizan operaciones diferentes en función de sus parámetros, de modo que su página del manual trata estos distintos aspectos.

- *Realimentación en función de la relevancia.* La otra forma de realimentación considerada está referida al interés que tienen para el usuario los documentos que se le muestran. En función de este interés se tienen dos tipos de realimentación que son negativa y positiva.
  La fórmula general de modificación de la consulta mediante realimentación por relevancia (presentada en apartado 4.4.1.2), se adapta con los siguientes valores de las constantes de ponderación, $\alpha = 1$, $\beta = 1/2$ y $\gamma = 1/4$. De esta manera, se le da un mayor peso a la realimentación con documentos relevantes ($\beta = 1/2$, realimentación positiva) que a la realizada con documentos irrelevantes ($\gamma = 1/4$, realimentación negativa) debido a que, según estudios empíricos, la realimentación positiva es más importante para reformular la consulta que la realimentación negativa [Salton 88]. Así se obtiene:

$$Q_{nueva} = Q_{anterior} + \frac{1}{2} \cdot \sum_{Ri \in Rel} Ri - \frac{1}{4} \cdot \sum_{Nj \in Nrel} Nj$$

  Donde *Rel* es el conjunto de documentos relevantes recuperados y *Nrel* es el número de documentos no relevantes recuperados. Además, en la versión actual de Argos, se limita a uno el número de documentos relevantes o irrelevantes que se usan para reformular la consulta. De este modo, cuando el usuario indica que un documento es relevante o irrelevante, automáticamente Argos modifica la consulta y realiza una nueva búsqueda. En esta modificación automática se utiliza la caracterización de cada documento (en función de sus términos más significativos) que se ha realizado en el proceso de indexación.
  La realimentación negativa se produce cuando el documento que se está visualizando no es interesante para el usuario y éste lo indica (mediante el botón *Irrelevant*) para que Argos lo tenga en cuenta en la siguiente búsqueda. El resultado es que los pesos de los términos más relevantes de ese documento se añaden en el modelo del usuario pero con peso negativo. Si el término ya existía previamente su peso disminuye y si no existía se incluye con peso negativo de acuerdo con la fórmula anterior. De esta forma es posible corregir situaciones en las que el usuario utiliza en la formulación de su solicitud palabras no apropiadas que desvían la búsqueda.

La realimentación positiva se produce cuando el usuario considera que el documento presentado es interesante y quiere utilizarlo para enriquecer su petición de información (esto se hace pulsando el botón *Relevant*). El efecto producido es que los términos más relevantes de dicho documento se añaden, con peso positivo, a la memoria de corto plazo del modelo de usuario. Si el término ya existía su peso se incrementa y si no existía se incluye (de acuerdo con la fórmula anterior). Esta complementación de la pregunta y la subsiguiente búsqueda, obtendrá nuevos documentos o modificará los valores de similitud de los anteriores, ya que todos estos términos adicionales si son significativos en este dominio.

5.5 Modelo del usuario

El Modelo de Usuario (MU) se define como el módulo cuyo objetivo es adaptar el comportamiento del sistema a cada usuario [Wahlster 89, Carberry 92]. Este modelo es esencial para una comunicación efectiva con potenciales usuarios con diferentes niveles de experiencia y que pueden tener intereses distintos en el uso del sistema. Dicho con otras palabras el MU trata de adecuar la información que proporciona a cada usuario concreto.

5.5.1 Caracterización del usuario y del dominio

El modelo de usuario de Argos es incremental, con adquisición automática y mixta e inspeccionable por el usuario [Kobsa 92, Chin 93]. El MU de Argos es incremental y con adquisición automática porque en él es el usuario quien tiene la iniciativa en la interacción pero no se le piden datos específicos para modelarlo (como por ejemplo que evalúe su nivel de experiencia), sino que es el propio sistema el que los va adquiriendo, completando y actualizando, de una forma no intrusiva durante su uso [Self 90a]. El sistema tiene en cuenta las características del usuario y analiza sus interacciones, como, por ejemplo, las preguntas que realiza. Es un modelo de adquisición mixta, ya que se pide al usuario que indique de forma expresa cuándo cambia el objetivo de su búsqueda (mediante el botón *Reset* del interfaz), así como cuál es el grado de interés que tiene para él la información presentada (proceso de realimentación). Otra particularidad destacable del modelo es que sea visualizable e inspeccionable (figura 5.4). El usuario (mediante el botón *User Model*) tiene acceso a la información que Argos utiliza para adaptar su comportamiento y éste puede modificar el contenido del modelo, si considera que se ha cometido algún error en el proceso de adquisición [Kobsa 94]. Además el usuario, mediante el botón *On/Off* del modelo, puede incluso desactivar el MU de modo que Argos funcione simplemente como un sistema de recuperación de información puro sin modelado.

Para la caracterización de los usuarios concretos se ha utilizado una aproximación mediante categorías de usuarios [Chin 89]. Estas categorías se establecen en función de una serie de características comunes y sirven para que, en ausencia de una información más concreta, se puedan realizar ciertas suposiciones sobre el interés o grado de experiencia de un miembro concreto de dichas categorías. Para la categorización de los usuarios se han tenido en cuenta dos criterios: a) el nivel de

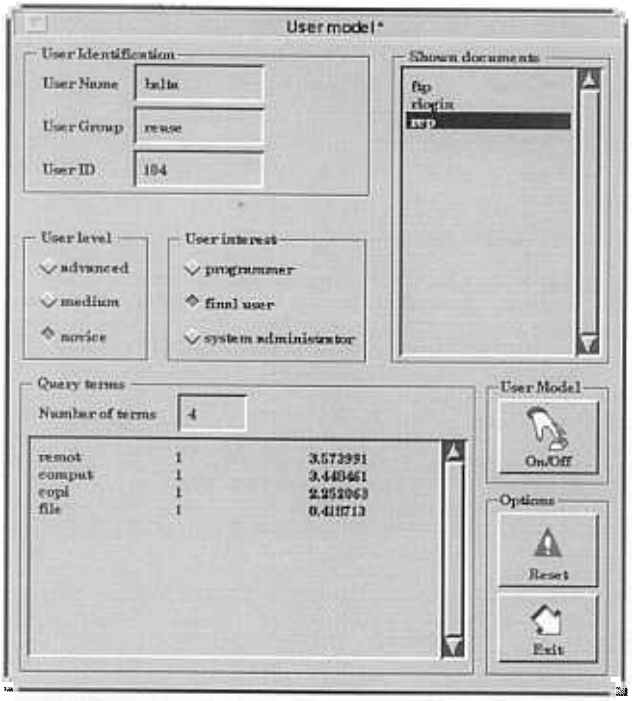


Figura 5.4: Visualización del modelo de usuario de Argos, cuando se ha realizado la consulta "copy a file to a remote computer" y después de acceder a los documentos *ftp, rlogin* y *rcp*. Destacar el botón On/Off mediante el cual se puede desactivar este modelo.

conocimiento o experiencia del dominio, y b) el principal uso que el usuario hace del sistema, p.e. únicamente como usuario final, o como desarrollador de aplicaciones (programador). De esta forma se ha llegado a la siguiente clasificación:

- *Tipo de usuario*
  - *nivel de experiencia*: novel, medio, avanzado.
  - *principal uso del sistema*: usuario final, administrador y programador.

Esta clasificación da lugar a nueve tipos diferentes de usuarios, pero se establece la restricción de que si el usuario es administrador entonces únicamente puede ser un usuario avanzado, para reflejar la situación habitual en los sistemas Unix.

Además de clasificar a los usuarios también se ha categorizado la información según su nivel de dificultad y por el objetivo de su uso. Para categorizarlas por su dificultad se ha hecho una agrupación de las órdenes de Unix, teniendo como criterio principal su posición normal en la curva de aprendizaje del sistema, es decir, cuándo las aprendería normalmente un usuario normal. De esta forma se han obtenido las siguientes categorías:

- *Tipo de orden*
  - *nivel de dificultad*: básica, intermedia, avanzada.
  - *propósito de uso*: general, administración, programación y específica.

En principio esta clasificación genera doce posibles clasificaciones de las órdenes, pero se han realizado simplificaciones. El nivel de dificultad, que supone una complejidad creciente, sólo se aplica a las órdenes de uso general. Las otras categorías de propósito de uso reflejan peculiaridades del dominio, como, por ejemplo, la existencia de herramientas de programación. La categoría de administrador corresponde a aquellas utilidades que sólo el superusuario puede ejecutar. Las órdenes específicas son aquellas que no se pueden situar en un lugar determinado de dicha curva, ya que representan necesidades específicas de algunos usuarios y que, por tanto, pueden ser conocidas por un usuario novel y no serlo por un usuario avanzado. Un ejemplo de orden específica sería *spice*, un programa utilizado para simulación de circuitos, que normalmente sólo conocerán los que se dediquen a simulación electrónica, independientemente de su conocimiento general del Unix. Se ha tenido también en cuenta que hay funcionalidades y documentación del sistema que trata de operaciones que sólo se pueden realizar mediante programación, como son las llamadas al sistema o las funciones de biblioteca en lenguaje C. La caracterización de las órdenes se realiza una única vez y de forma manual. Con este objetivo se crea un archivo para cada nivel de dificultad (excepto para las intermedias), que contiene las órdenes de propósito general que están comprendidas en esa categoría. También se crea un archivo para cada una de las categorías de programación, administración y específica. Si una orden no aparece en ninguno de los archivos se considera como de propósito general y de dificultad intermedia.

5.5.2 Adquisición, contenido y mantenimiento del modelo

Como ya se ha dicho, la clasificación del usuario por su grado de experiencia y su interés en el uso del sistema la realiza Argos de modo automático. No se considera adecuado solicitar a la propia persona que se clasifique dentro de alguno de los grupos, ya que su visión puede ser parcial y diferente de la considerada en el sistema [Kok 91]. Concretamente Argos clasifica al usuario mediante el estudio de las últimas órdenes que éste ha utilizado en la interacción con el sistema operativo (se aprovecha el *history* en el que Unix almacena el histórico de las últimas interacciones del usuario). A cada usuario, en función de esta interacción previa, se le asigna un uso principal del sistema y un nivel de conocimiento. En este razonamiento se asume que si el usuario ha utilizado una orden es porque la conoce (aunque podría haber sido por error o por casualidad). Las heurísticas utilizadas en el proceso de clasificación de un usuario están directamente relacionadas con la clasificación que se ha realizado de las órdenes. Estas reglas son:

- *Respecto al principal uso del sistema*: se considerará que es un usuario final a menos que en la interacción previa aparezca alguna orden de programación, en cuyo caso se clasificará como programador, o que su número de identificación sea cero, en cuyo caso tiene capacidad de ser supervisor del sistema y por tanto se considera que es un administrador.

- *En cuanto al nivel de experiencia*: se considera que es un usuario intermedio a menos que el 90% de las órdenes generales utilizadas sean básicas y no aparezca ninguna orden avanzada, en cuyo caso se clasifica como novel; si por

lo menos el 10% de las ordenes utilizadas son avanzadas el usuario se considera como avanzado.

En la figura 5.5 se presenta la especificación completa en EBNF del modelo de usuario que utiliza Argos. Al principio de una sesión con el sistema de ayuda el MU contiene la siguiente información: la identificación del usuario y el tipo del usuario. El tipo del usuario, es decir su nivel de experiencia y su principal uso del sistema se adquieren automáticamente aplicando las heurísticas previamente descritas. Los datos de identificación, es decir el número e identificador del usuario y el grupo al que pertenece, se obtienen directamente del entorno Unix. Estas características se determinan al principio de la sesión con Argos y a partir de ahí son estáticas. No se considera la circunstancia de que el usuario pueda modificar su nivel de experiencia en el transcurso de una sesión, ya que típicamente éstas no serán muy largas y, por tanto, no se incluyen reglas para la reclasificación del usuario.

Este conjunto de datos se va completando y grabando en el modelo del usuario en base a la interacción con Argos. En esta actualización del modelo se van rellenando la lista de información mostrada y la memoria de corto plazo. Esta es una información dinámica, cuyo objetivo es focalizar cuál es el interés real del usuario en una búsqueda determinada y evitar presentar documentos que el usuario ya conoce porque han sido previamente mostrados. En la lista de información mostrada se van almacenando los documentos que se presentan al usuario, el tipo de orden al que corresponden y el interés que ha tenido esa información para él. La memoria a corto plazo está formada por pares término-peso que se obtiene a partir de las consultas del usuario y de la información de relevancia según el interés que tengan para él los documentos presentados (realimentación). Este proceso de realimentación se trata con detalle en el apartado 5.4.2. En el modelo descriptivo de la figura 5.5 se destaca el diferente origen de los pares término-peso de esta memoria, distinguiendo entre los que provienen directamente de la consulta del usuario, los que provienen de la selección de texto y los que provienen de la realimentación. La memoria de corto plazo puede ser reinicializada expresamente por el usuario cuando cambia de tema (mediante el botón *reset*). En este caso los términos procedentes de consultas anteriores o de la realimentación ya no son considerados en las siguientes búsquedas.

En Argos no se almacena información entre diferentes sesiones separadas en el tiempo. En nuestro modelo, los intereses del usuario pueden cambiar completamente a lo largo del tiempo, por lo que la información de un uso anterior no ayudaría en la caracterización de la necesidad de ayuda del usuario. Como el proceso de adquisición del modelo es rápido, no es necesario por razones de cálculo o de tiempo almacenar modelos anteriores, además de que esto complicaría el proceso debido a que en ciertas ocasiones habría que tratar con información contradictoria. Las circunstancias de cada persona entre dos usos distintos del sistema pueden haber cambiado significativamente siendo muy difícil tener en cuenta, por ejemplo, que su conocimiento sobre el sistema operativo puede haberse incrementado debido a la realización de un curso. Otro ejemplo es el cambio de uso primario del sistema; un usuario puede de forma ocasional programar alguna utilidad, con lo que se clasificaría en algunas situaciones como programador, mientras que el resto del tiempo esa información no le interesa, por ser únicamente un usuario final. Aunque

esta información entre sesiones podría ser un complemento para la caracterización inicial del nivel y del uso principal del sistema, no se ha considerado oportuno tenerlo en cuenta, debido a que los problemas que ocasiona son superiores a las ventajas que presenta. Un error en la adquisición del modelo a largo plazo es más grave que si se produce en las características transitorias [Kobsa 93]. Por otro lado, en este modelado un usuario avanzado que en sus ultimas interacciones haya utilizado sólo órdenes básicas quedaría catalogado como usuario novel. No obstante esta circunstancia no se considera muy importante ya que, como el modelo de usuario es inspeccionable, se le proporciona la posibilidad de que modifique directamente su clasificación.

5.5.3 Aplicación del modelo

El objetivo principal de este modelo es la adaptación de la información al nivel de experiencia y al interés de cada usuario. No obstante, como en Argos se reutilizan las páginas del manual previamente existentes, el contenido de los documentos presentados no puede modificarse. Por tanto, se trata de determinar cuál es la necesidad del usuario, para modificar las siguientes búsquedas, y de obtener cuál es su nivel de experiencia y objetivo de uso, para aplicar en el interfaz factores correctivos a los resultados del módulo de Recuperación de Información.

Esta adaptación se traduce en:

- *Modificación de la búsqueda de información.* El proceso de búsqueda de documentos relevantes cuando el usuario pide ayuda sobre una tarea, que realiza el módulo de recuperación de información, no va a tener en cuenta únicamente la última solicitud del usuario, sino que ésta se complementa con la memoria a corto plazo. Normalmente, en un sistema del tipo pregunta-respuesta como Argos, no se obtiene la información deseada mediante una única petición, sino que esta petición se debe refinar sucesivamente hasta obtener la contestación adecuada [Gindon 88, Hartley 88a, Hartley 88b]. El sistema colabora automáticamente con el usuario, teniendo en cuenta toda esta serie de interacciones. Además, de esta forma se dispone de más texto, lo que mejora el proceso estadístico de recuperación.
  En todo momento se tiene en cuenta la realimentación por parte del usuario hacia el sistema (este proceso de realimentación se trata con detalle en el apartado 5.4.2). Si el interés del usuario cambia, éste lo debe indicar (mediante el botón *reset*) de modo que Argos reinicialice esta memoria de corto plazo. Por otra parte podría ocurrir que el vocabulario utilizado no fuese el más adecuado y, por ello, se obtuviesen documentos que no fuesen interesantes para el usuario. En este caso puede indicarlo (mediante el botón *Irrelevant*) y el sistema lo toma en cuenta para búsquedas sucesivas. De manera similar, se puede enriquecer la consulta indicando que un documento es interesante (mediante *Relevant*) y que se desea obtener más información relacionada. Por otro lado, si el usuario encuentra uno o varios párrafos que describen o complementan su necesidad, y sobre los que quiere obtener más información, puede seleccionarlos mediante el ratón, y se considerarán también como caracterización de su petición.

```
<modelo de usuario> ::=
     (    <identificación de usuario>
          <tipo de usuario>
          { <información mostrada> }
          <memoria de corto plazo> ) ;

<tipo de usuario> ::= (<nivel de experiencia> <principal uso del
sistema> ) ;

<nivel de experiencia> ::= novel | medio |avanzado ;

<principal uso del sistema> ::= usuario final| administrador |
programador ;

< información mostrada> ::=
     (    <identificador de información>
          <tipo de información>
          <interés para el usuario> ) ;

<tipo de información > ::= <propósito de uso> | <propósito de uso>
<nivel de dificultad> ;

<propósito de uso>::= general | administración | programación |
específica ;

<nivel de dificultad> ::= básica | intermedia | avanzada ;

<interes para el usuarior> ::= relevante | normal | irrelevante ;

<memoria de corto plazo> ::=
     (    { <término consulta> <peso normalizado> }
          { <término seleccion> <peso normalizado> }
          { <término realimentación> <peso normalizado> } ) ;

<término consulta> ::= < término> ;

<término seleccion> ::= < término> ;

<término realimentación> ::= <término> ;

<término> ::= cadena ;
     (* raices obtenidas mediante el algoritmo de Porter, una vez
          eliminadas las palabras sin contenido semántico *)

<peso normalizado> ::= real ;

< identificación de usuario> ::= (<nombre de usuario> <número de
usuario> <grupo de usuario>) ;

<grupo de usuario> ::= cadena ;

<nombre de usuario> ::= cadena ;

<número de usuario> ::= entero ;

<identificador de información> ::= cadena ;
```

Figura 5.5: Especificación del modelo de usuario de Argos

Este tipo de funcionalidad ha demostrado ser muy útil y se incorpora en otros sistemas de acceso a información como WAIS [TMC 91]. Por otra parte, el conjunto de documentos previamente presentados no influye en el proceso de búsqueda, sino que sólo se utilizan para evitar modificar el orden de presentación como se describe a continuación.

- *Modificación de la presentación de información.* La otra forma de adaptación consiste en la modificación del proceso de presentación de información. El proceso básico es presentar los documentos según su orden de adecuación proporcionado por el sistema de RI, de forma que aparezca directamente el más relevante en el panel *Information*. Para mejorar la usabilidad del sistema, de modo que se evite en lo posible la repetición y la presentación de información inoportuna, se produce un proceso de filtrado o reordenación de los documentos a mostrar. Con este propósito se utiliza la lista de información mostrada del MU y la categorización de las órdenes. El orden de presentación obtenido mediante RI se modifica de la siguiente manera: primero se muestran los documentos de propósito general cuyo nivel de dificultad es más adecuado a la experiencia del usuario; si el usuario es administrador o programador los documentos con la clasificación correspondiente se muestran según su relevancia, si no se visualizan después de todos los de propósito general; por último, si un documento aparece en la lista de información mostrada se presentará al final.
  De esta manera, si ya se ha mostrado un texto en esa sesión se logra evitar su presentación como primera respuesta a una petición. Como el usuario ya sabe cuál es su contenido, o por lo menos qué tema trata, si le parece interesante volver a consultarlo puede seleccionarlo directamente con el ratón.
  En todo momento se intenta presentar primero la información más adecuada a su nivel de conocimiento, o al interés específico del usuario. Por ejemplo intenta evitar que el sistema presente, como primera respuesta a una petición, una orden que sólo puede ejecutarse siendo superusuario cuando el demandante no lo es. Otro caso está relacionado con una de las partes del manual de UNIX que trata de funciones de programación en lenguaje C; aunque estos documentos tengan información sobre un tema determinado, se estima que sólo le interesarán a los usuarios que estén programando y no a un usuario normal. No obstante, hay que evitar que este filtrado pueda ocultar información de forma no adecuada. Es posible que un usuario normal esté intentando hacer algo que sólo puede hacer el superusuario, como por ejemplo instalar una nueva impresora, o aumentar la prioridad de ejecución de sus procesos. Únicamente se modifica el orden de presentación de los documentos, de modo que el usuario puede acceder a toda la información obtenida. Si averigua que la operación deseada no la puede hacer él, puede solicitar al supervisor que la realice enviándole un mensaje (mediante el botón *Suggest*).
  No obstante, aunque la reordenación se realice de acuerdo al MU hay que destacar que en todo momento se continua mostrando de forma gráfica la información de relevancia (mediante las barras que aparecen en el campo *Found*). De esta forma el usuario podría acceder directamente al documento más relevante aunque no sea el correspondiente a su modelo.

Este sistema de clasificación doble, de los usuarios y de la información, presenta también la ventaja de que es fácil considerar la inclusión de un nuevo documento. Sólo hay que determinar cuál es su propósito de uso, y si este es de propósito general, cuál es su dificultad, para así poder incluirlo en el archivo correspondiente a su categoría. Esto permite por ejemplo clasificar de forma sencilla los documentos que crean los propios usuarios del sistema, o los textos asociados con las nuevas aplicaciones que se instalan (proceso muy común en un sistema abierto como el Unix). Por omisión, si no se indica de forma expresa, se supone que la ayuda asociada a las nuevas aplicaciones pertenece a la categoría específica.

Como conclusión se desea destacar que en Argos se realiza un modelado eminentemente pragmático, que es sencillo de implementar a la vez que razonablemente eficiente (sólo se ha realizado una evaluación informal), ya que se reutilizan y complementan modelos ampliamente probados en este campo de aplicación. Este tipo de modelado de los usuarios es un refinamiento de otros que han sido utilizados en este dominio con distintos propósitos [Wilensky 89, Chin 89, Doane 91, Chin 93, Kobsa 93]. En todos ellos se realiza una categorización de los usuarios por su nivel de experiencia, pero sólo en el Unix Consultant se realiza además la clasificación de la información por su nivel de dificultad [Wilensky 89, Chin 89]. En Argos se mejora tanto la clasificación de los usuarios como la de la documentación en relación con estos sistemas. Respecto a los usuarios no sólo se considera el nivel de experiencia, sino que también se tiene en cuenta el uso principal que hacen del sistema. Con la documentación se considera su nivel de dificultad en cuanto a aquella que es de propósito general, mientras que se tienen en cuenta peculiaridades del dominio, diferenciando otras tres clases de información de propósito específico. Además este modelado cumple las dos principales condiciones que según Chin [Chin 93] debe tener el modelo de cualquier sistema de ayuda: es de rápida adquisición y tiene un cierto poder predictivo en función de un conocimiento parcial.

5.6 Interfaz

En el diseño del interfaz de Argos se ha tenido en cuenta la sencillez de manejo y la uniformidad (figura 5.6). En un asistente es importante que el acceso a la ayuda proporcionada no suponga una dificultad añadida. Por tanto, hay que evitar la sobrecarga cognoscitiva del usuario y la dificultad de uso del sistema [Downton 91]. Con este objetivo se respeta la apariencia (*look-and-feel)* y la forma de trabajo del resto de aplicaciones del sistema X Window, y se incluyen unas instrucciones sencillas sobre su manejo (accesibles mediante el botón *help*).

5.6.1 Hipertexto en Argos

Argos utiliza técnicas de hipertexto como herramienta de navegación a través de la información y como forma de interacción con el sistema [Conklin 87, Nielsen 90, Bevilacqua 89, Barret 89]. Un hipertexto es también es un sistema de representación de información, cuya principal característica es que el contenido textual se complementa con enlaces que permiten acceder de forma directa a otros

textos relacionados. Por tanto, los elementos fundamentales de estos sistemas son los nodos y los enlaces. Normalmente un hipertexto va asociado con un interfaz gráfico que permite la interacción mediante botones, iconos y el uso de dispositivos apuntadores, p.e. ratón, además de sobre el propio texto.

La información disponible sobre el sistema Unix cumple las tres condiciones que propone Shneiderman para que sea adecuada la construcción y aplicación de un hipertexto (*the Golden Rules of Hypertext* [Shneiderman 89b]). Estas son: a) que exista una gran cantidad de información organizada en numerosos fragmentos; b) que estos fragmentos estén relacionados entre sí; y c) que el usuario necesite sólo una pequeña cantidad de esta información en cada momento. El hipertexto es un enfoque especialmente apropiado para sistemas de ayuda, ya que permite la presentación a los usuarios de grandes cantidades de información, sin necesidad de que esté muy estructurada como es el caso del manual de Unix [Nielsen 90, Mayes 90].

La interacción mediante técnicas de hipertexto tiene también una serie de limitaciones que hay que considerar y solucionar. Según Conklin, la desorientación del usuario y la sobrecarga cognitiva son los problemas principales de los sistemas de hipertexto, que pueden limitar en gran parte su utilidad final [Conklin 87]. El problema de la desorientación surge, por ejemplo, cuando se está perdido sin saber cómo acceder a una información que se sabe o se supone que existe, o cómo volver a acceder a un texto previamente presentado. La sobrecarga cognitiva se produce debido a que el usuario tiene que tomar continuamente decisiones sobre qué camino seguir, a qué información acceder, o qué información previa debe tener en cuenta. En Argos se incluyen diversos elementos propuestos en la bibliografía, con el objetivo de resolver estos problemas [Shneiderman 89a, Barret 89, Balasubramanian 93]. Por ejemplo, se proporciona la función *back* que permite al usuario volver a la información presentada anteriormente según la secuencia temporal. Por otra parte, en todo momento el usuario puede hacer una pregunta al sistema para acceder a la información deseada, lo que es especialmente adecuado para cantidades muy grandes de documentos. Además siempre se proporciona un camino o ruta de navegación por omisión (mediante el botón *next*), de forma que se presenta la información secuencialmente ordenada según su adecuación, sin que el usuario tenga que tomar decisiones.

Una opción posible para convertir el manual de UNIX en un hipertexto es introducir enlaces dentro del texto. Este es el enfoque seguido en las primeras versiones de la aplicación AnswerBook [Sun 92] para estaciones de trabajo Sun, donde se ha digitalizado el manual y se han introducido conexiones entre la información relacionada. En esta visión particular, todos los enlaces están presentes explícitamente en el hipertexto. Aunque la simplicidad de este enfoque lo hace directamente accesible a los usuarios, tiene diversos inconvenientes, entre los que cabe destacar: a) la construcción de enlaces de forma manual es un proceso caro, muy propenso a cometer errores, y difícilmente escalable cuando la documentación es muy extensa; b) no se pueden enlazar nuevos textos con los ya existentes sin que ambos estén presentes durante el proceso de creación de enlaces; c) es un enfoque muy rígido, los enlaces se crean una vez y a partir de ahí quedan

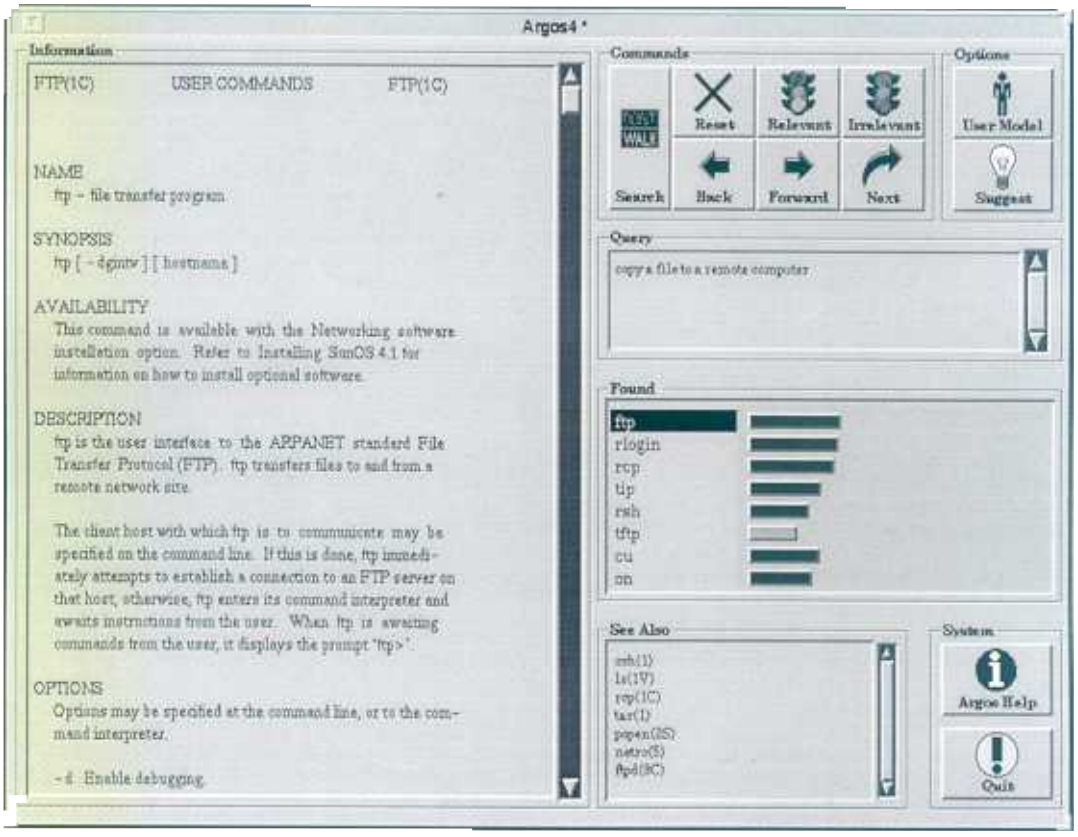


Figura 5.6: Interfaz de Argos después de haber realizado la consulta "copy a file to a remote computer". Los valores de relevancia de los documentos recuperados en el campo *Found* se muestran mediante barras gráficas de distintas longitudes y colores.

fijados; y d) los enlaces reflejan únicamente la forma en que el autor de ese hipertexto entiende la estructura y flujo de esa información.

En Argos se evitan estos problemas mediante la utilización de ideas de lo que algunos autores han denominado hipertexto dinámico, modelo de resolución de enlaces, consulta basada en preguntas (*query-based browsing*) o modelo procedimental de dos niveles [Mayfield (en prensa), Balasubramanian 93]. De esta manera, en Argos las páginas del manual constituyen la base para la generación de un hiperdocumento, en el cual los enlaces entre las diferentes partes no están completamente prefijados y que tiene en cuenta al usuario para el cómputo de estos enlaces. Es decir existen enlaces dinámicos entre la información que se generan cuando el usuario lo solicita. En la creación de los enlaces se usa la información contenida en la memoria de corto plazo del modelo de usuario (MU). Es decir los términos de las solicitudes previas, así como los provenientes de la realimentación. A continuación tratamos la solución aportada por Argos a los dos elementos más característicos de un hipertexto que son los nodos de información y los enlaces entre ellos.

5.6.1.1 Nodos del hipertexto

El hipertexto construido en Argos es de grano grueso; en el que cada documento o página es un nodo. La documentación de Unix es muy regular, presentando para la descripción de las órdenes, funciones o archivos de configuración, una serie de secciones fijas que aparecen en casi todos (p.e. name , synopsis, description, see also) y otras secciones de aparición variable (p.e. warnings, diagnostic, examples, files). En la figura 5.7 se presenta una lista de las secciones más habituales y una breve descripción de su contenido.

Debido a estas regularidades se podría haber creado una representación más detallada para cada página (pequeño hipertexto), incluyendo estas secciones como enlaces estructurales del documento. Esto no se ha hecho así debido a que la longitud media de cada documento es pequeña y por ello no se ha considerado necesario (típicamente es inferior a 2 páginas impresas, p.e. en el caso de la sección 1 del manual de usuario es de 1.14 páginas). Únicamente se ha utilizado el apartado *see also* para obtener los enlaces estáticos, ya que en este apartado el documentalista ha reflejado de forma breve cuáles son los documentos relacionados.

Argos se permite añadir nuevos documentos (nodos) de una forma supervisada para evitar la inclusión de información errónea o no relevante. Cuando se quiere incluir un nuevo texto, se puede generar un mensaje (mediante el botón *suggest*) que se envía al supervisor del sistema. Si el supervisor lo considera oportuno incluirá esta información en la base de documentos, indexándola según el proceso descrito en el apartado de recuperación de información. Si este nuevo documento se desea enlazar de forma manual con alguno de los ya existentes, lo único que hay que hacer es incluir una sección *see also* en la que se indican los documentos relacionados.

5.6.1.2 Enlaces estáticos y enlaces dinámicos

En Argos existen dos clases de conexiones entre los nodos (documentos) del hipertexto que son los enlaces estáticos y dinámicos.

- *Enlaces estáticos*: Los enlaces estáticos se obtienen a partir de la sección *see also* de cada página del manual. Como se ha tratado previamente esta sección contiene los documentos que el documentalista considera relacionados con el actual. Este contenido se presenta de forma automática en la ventana *see also*, de modo que está disponible para el usuario que puede acceder directamente a esos textos mediante el ratón. Pulsando sobre el nombre de un documento, éste se presenta en el panel *Information*.

- *Enlaces dinámicos*: Son enlaces con otros documentos que se generan dinámicamente, cuando el usuario lo requiere y teniendo en cuenta su interés en ese momento. Cuando el usuario quiere conocer más sobre un tema presentado en un documento, puede marcar con el ratón el fragmento de texto que lo trata y pulsar el botón de búsqueda (*Search*). Argos genera automáticamente (mediante RI) el enlace con otros documentos que tienen información relacionada. En este proceso se utiliza la información existente en el MU. Estos enlaces dinámicos se crean automáticamente mediante el

Secciones fijas

NAME: Nombre de la orden y descripción corta de la función del comando.
SYNOPSIS: Uso o usos de la orden, indicando las diferentes opciones que puede tener y como tiene que aparecer en la línea de órdenes.
DESCRIPTION: Hace una descripción, más detallada, de su comportamiento y uso.
OPTIONS: Comenta las diferentes opciones enumeradas en SYNOPSIS, y como modifica el comportamiento de la orden cada una de estas opciones.
SEE ALSO: Incluye otras órdenes relacionadas.

Secciones variables

FILES: Archivos del sistema relacionados con la orden.
WARNINGS: Avisos sobre su uso que hay que tener en cuenta.
DIAGNOSTIC: Estudio del valor devuelto por la función.
RESTRICTION: Restricciones de uso.
AVAILAVITY: Indica en que versiones está disponible.
BUGS: Problemas.
EXAMPLES: Ejemplos de aplicación de estas órdenes.
RETURN VALUES: Lista de valores devueltos y su significado (sección 2 y 3).

Figura 5.7: Descripción de las secciones más habituales de las página del manual de Unix

análisis estadístico y léxico del texto, tratado en el epígrafe de recuperación de información. Esto posibilita la ausencia de enlaces preestablecidos en el texto.

De esta forma se tiene un hipertexto de dos niveles: uno formado por todos los documentos y otro superior en el que se calculan los enlaces entre esos documentos. Este cálculo de enlaces permite la adaptación al usuario: aunque los documentos sean fijos, los enlaces entre ellos dependen del interés del usuario en cada momento, que se captura mediante los datos del MU.

Una de las limitaciones destacables de este hipertexto generado automáticamente es la semántica de los enlaces; básicamente se proporciona un único tipo de enlace, que supone un vínculo a otra información relacionada sin más especificación. Los métodos de recuperación de información usados son de naturaleza estadística; aunque pueden descubrir asociaciones entre textos no pueden averiguar cuál es el tipo de relación, simplemente indican que dos textos son similares en función de una definición estadística de similitud. Esto hace que sea el usuario el que deba descubrir el tipo de relación existente entre los documentos.

5.7 Resumen

En este capítulo hemos descrito el diseño, construcción y funcionamiento del sistema Argos, un sistema de ayuda en el entorno Unix. Argos proporciona una potente interfaz gráfica para facilitar el acceso a la documentación y permite la formulación de consultas en lenguaje natural. Argos constituye un sistema práctico, efectivo, generalizable y proporciona un marco de aplicación concreto orientado a nuestra investigación sobre sistemas de ayuda.

Primero hemos realizado un análisis del dominio del Unix, estudiando su interfaz y aquellos otros sistemas que han tratado de simplificar su utilización. También se han estudiado las características de su documentación electrónica. A continuación hemos definido los objetivos, las funcionalidades y la estructura de Argos. Finalmente, se han tratado con detalle las tres técnicas estándar que se integran en el sistema, la recuperación de información, el modelado del usuario y el hipertexto (que es crucial en la interfaz del sistema), así como su relación con las funcionalidades ofrecidas por Argos.

En el apéndice final se incluyen varios ejemplos de uso de Argos para ilustrar el modo de trabajo con el sistema. Con este objetivo se utilizan diferentes consultas y distintos contenidos del modelo del usuario. Estas interacciones con Argos se han tomado de la fase de experimentación y prueba del sistema (evaluación formativa), que se ha realizado con estudiantes de doctorado y con otros miembros del grupo de trabajo (la evaluación completa, sumativa, no se ha considerado objetivo de este trabajo posponiéndose para una siguiente fase). No obstante, en esta experimentación hemos detectado que a pesar de proporcionar una potente interfaz, la influencia del conocimiento del vocabulario del dominio continúa siendo clave en el uso eficaz del sistema. Además hemos constatado una insuficiente estructuración de la información.

En conclusión, Argos representa un ejemplo de aplicación parcial de nuestro modelo de sistemas de ayuda en el dominio del sistema operativo Unix. Los resultados y la experiencia obtenida con la construcción de Argos nos permite ser positivos con respecto a las ideas inicialmente planteadas en nuestro modelo de asistente y plantearnos una segunda fase que materializaremos en el desarrollo del sistema Aran. Argos proporciona una mejor definición del marco de aplicación de Aran, que se centra en el uso de técnicas de representación del conocimiento y que presentamos en un capítulo siguiente. Como veremos en el capítulo siguiente, el uso de técnicas de representación del conocimiento en el sistema Aran permite mejorar aquellos aspectos del sistema de ayuda que como, por ejemplo, la estructuración y presentación de la información o el modelado del usuario, han quedado insuficientemente resueltos en Argos.

CAPÍTULO 6

EL SISTEMA ARAN

6.1 Introducción

Aran es un sistema de ayuda inteligente para el sistema operativo Unix. El sistema Aran continúa el trabajo iniciado con Argos, introduciendo una representación explícita de los objetos, conceptos y relaciones utilizados en el dominio del Unix, con el fin de mejorar la efectividad de la ayuda. Aran refleja nuestra visión del doble objetivo de la ayuda. Además de proporcionar la información necesaria para solucionar los problemas del usuario, se aprovechan las solicitudes del usuario para aumentar su conocimiento sobre el sistema operativo.

En nuestro modelo para la construcción de sistemas de ayuda, las cuestiones pragmáticas son fundamentales. En los sistemas de ayuda basados en conocimiento la construcción de la base de datos, en que se basa su funcionamiento, es la parte más costosa y que exige mayores recursos. La solución que se propone en este trabajo, y que se ejemplifica con Aran, es la reutilización (al menos parcial) de bases de conocimiento preexistentes y el uso de un lenguaje de representación de conocimiento estándar. De esta forma se puede incluir información sobre temas más complejos al integrar el esfuerzo de distintos grupos de trabajo manteniendo al mismo tiempo un coste razonable. Además, en dominios que evolucionan tan rápidamente como es el caso de los sistemas informáticos, es vital que el conocimiento representado sea fácilmente actualizable, permitiendo la inclusión de nuevas utilidades así como el refinamiento o la modificación de la información existente. Este aspecto ha determinado la elección de un lenguaje de representación de conocimiento basado en lógicas descriptivas, que, al incluir un clasificador automático, simplifica tanto el proceso de codificación como el de actualización.

El núcleo central de Aran es su base de conocimiento. En esta se representan entre otras cosas los objetos del dominio y las acciones sobre esos objetos. Este contenido de la base de conocimiento, es un modelo del dominio que se utiliza con dos propósitos diferentes. Por una parte, se trata de disponer de un modelo de la aplicación (en nuestro caso, el sistema operativo Unix) que el usuario pueda inspeccionar y que le facilite su aprendizaje. Por otra parte, este modelo se utiliza para organizar, indexándola, cualquier otra información disponible sobre el

dominio. Finalmente, y con el objetivo de que el sistema de ayuda presente un comportamiento adaptativo, también se incluye como parte de la base de conocimiento información sobre el usuario.

Aunque este trabajo se centre en el desarrollo de un sistema de ayuda para el sistema operativo Unix, la aproximación seguida puede utilizarse para otros dominios informáticos. Unix es un dominio particularmente adecuado para el desarrollo de este tipo de sistemas de ayuda, ya que es un entorno real cuya utilización está muy extendida y que incluye información textual en formato electrónico. Además, como se han desarrollado previamente diversos sistemas de ayuda para Unix existe distintos tipos de información que se pueden reutilizar.

En este capítulo comenzamos analizando la información y su estructuración para que se puedan alcanzar los objetivos de ayuda y enseñanza. Seguidamente se pasa a describir el planteamiento y la estructura del sistema Aran. En apartados sucesivos se analizan y describen con detalle cada uno de los elementos que componen el sistema:

- La interfaz del sistema.
- La indexación automática de la información.
- El modelado del usuario.
- El modelo del dominio.

Al final presentamos un breve resumen.

6.2 La información como ayuda en Aran

Los entornos informáticos tienen determinadas peculiaridades que hay que tener en cuenta en la creación de sistemas de ayuda. Como ya hemos mencionado previamente, en la generalización del uso de los asistentes en entornos reales un factor clave es su coste de desarrollo. La información que se proporciona en el asistente determina una parte importante de este coste. Otra característica a destacar es que las aplicaciones informáticas son sistemas que evolucionan, siendo habitual que aparezcan sucesivas versiones que incluyen nuevas funcionalidades. Esto implica que los asistentes para entornos informáticos deben ser modificables y ampliables de forma sencilla. Además, estos entornos son utilizados por usuarios de diversos tipos, con distintos conocimientos e intereses, y esto influye en el tipo de información a proporcionar.

En nuestro modelo, el desarrollo de sistemas de ayuda se basa fundamentalmente en el uso de información de referencia previamente existente que, debidamente estructurada y con unos medios de acceso adecuados, permita obtener asistentes efectivos y aplicables a dominios reales. Aran supone la ejemplificación de este modelo en el dominio del sistema operativo Unix.

6.2.1 Estructuración de la información mediante el conocimiento del diseño

En la creación de un sistema informático, y como una fase más de su desarrollo, se producen documentos en los cuales se proporciona, con gran nivel de detalle, información sobre su diseño, implementación, funcionamiento y uso [Pressman 93]. Sin embargo, en este proceso de documentación del software se pone más énfasis en la acumulación de información a lo largo de las diferentes fases del ciclo de vida, que en su integración, interrelación y facilidad de utilización por parte de los distintos tipos de usuarios del sistema [Jonhson 94]. Esto hace que si se quiere tener una visión completa de una aplicación, haya que acudir a documentación de muy diverso tipo, desde una información general sobre el diseño de alto nivel, hasta una información más detallada sobre la implementación o las funcionalidades que se proporcionan. Concluyendo, resulta habitual disponer de una completa documentación sobre las aplicaciones, pero sin embargo no se suele aprovechar esa documentación para proporcionar ayuda a los usuarios.

La estructura de las aplicaciones informáticas está determinada por su fase de diseño, en la cual se identifican sus diversas partes, sus interrelaciones, las condiciones que se deben cumplir y sus restricciones. Este conocimiento es fundamental para obtener un modelo mental correcto de la aplicación que facilite su uso, conocimiento que sin embargo se "dispersa" durante el proceso de codificación o implementación. Esta circunstancia es especialmente importante en los sistemas software complejos, que al ser tan amplios hacen que resulte difícil que una persona tenga una imagen completa de su funcionamiento y estructura (como en los analizados en el punto 3.4.3). Esto crea lo que algunos autores han llamado el problema de descubrimiento (*discovery problem*); el proceso de aprendizaje de un sistema para poder utilizarlo o modificarlo [Devanbu 91]. Esta complejidad inherente al tamaño, a las interrelaciones entre las diversas partes del sistema y a sus particularidades, hace que la información sólo pueda ser comprendida teniendo un conocimiento más amplio sobre el funcionamiento y diseño del entorno completo. La dificultad de comprensión se ve agravada por el hecho de que estos sistemas no sean estáticos, sino que evolucionen (en informática la actualización de versiones es un proceso habitual y constante), añadiendo nuevas funcionalidades o modificando el comportamiento de las que ya existan. Como caso extremo tenemos los sistemas abiertos (de los cuales Unix es un ejemplo paradigmático), en los que los cambios pueden ser todavía mas frecuentes, ya que se han diseñado para ser ampliables.

A continuación vamos a particularizar estas reflexiones sobre la información existente y las restricciones de diseño para el sistema operativo Unix, y cómo se han utilizado en el desarrollo de Aran. El caso del Unix es peculiar, como ya se ha indicado en el capítulo de Argos, ya que en la distribución estándar se incluye una información muy extensa en formato electrónico sobre lo que está disponible en el sistema: órdenes de usuario, mandatos de administración, archivos de configuración, llamadas al sistema y funciones en lenguaje C. Sin embargo, para esa información tan amplia sólo se proporcionan medios básicos de acceso y una estructuración muy limitada. Por otra parte, a lo largo de todo el manual del Unix se encuentra distribuida una cierta información sobre su concepción de diseño, como por ejemplo sobre los archivos de configuración, los distintos tipos de dispositivos o

la estructuración de directorios, pero no se proporciona ningún medio específico para facilitar su acceso. Por ejemplo, en la sección 5 se incluye información sobre la estructura de distintos tipos de archivos del sistema, como los directorios o los archivos de ayuda, y en la sección 4 se describen los controladores de dispositivos incorporados. Este tipo de reflexiones se han tenido en cuenta en la fase de diseño de este software y son imprescindibles para tener un modelo adecuado del sistema operativo. Para encontrar agrupadas y organizadas todas estas consideraciones sobre su diseño y estructura hay que acudir a libros y manuales sobre su diseño e implementación.

La información ofrecida a los usuarios como ayuda, no debe ser únicamente correcta y pertinente a sus necesidades, sino que también debe poder ser comprendida y asimilada por distintos tipos de usuarios, algunos más bien inexpertos, si se quiere obtener un beneficio y mejora en su rendimiento [Draper 84, Duffy 89]. Además de facilitar el acceso a la documentación, también es primordial simplificar la comprensión y la asimilación de estos documentos. Para lograr estos dos objetivos, la información se debe estructurar e indexar de modo que tenga significado para el usuario [Curtis 89]. En Aran esto se logra principalmente con la existencia de un modelo conceptual del Unix, en el que se trata de capturar esa información que se ha tenido en cuenta en la fase de concepción del sistema. Se incluyen las restricciones y decisiones de diseño que se han perdido en el proceso de codificación. Esta representación de conocimiento, que es un modelo explícito del dominio, se obtiene a partir de libros de referencia sobre el sistema operativo, del análisis de la documentación y de la reutilización parcial de otras bases de conocimiento. Este modelo de los objetos, acciones y estructura, se utiliza para facilitar la comprensión por parte del usuario de los conceptos involucrados.

6.2.2 Integración de diferentes tipos de información

La solución adoptada en Aran permite integrar, interrelacionar y estructurar diversos tipos de información. Por ejemplo, los usuarios con menor experiencia pueden encontrar en la base de conocimiento las definiciones de los conceptos relacionados, con su significado preciso (p.e. qué es un controlador *-driver-* de dispositivo), o ejemplos de uso de las órdenes. También permitiría incluir información detallada, paso a paso, sobre tareas aunque esta faceta no se aborda en este trabajo por motivos de coste. De este modo, con un único sistema se pueden satisfacer las necesidades de distintos tipos de usuarios, que normalmente tendrían que acudir a fuentes de información diferenciadas. Además, este modelo también es útil para usuarios con mayor experiencia, ya que, como demuestran algunos estudios experimentales, hay que modificar ciertas suposiciones respecto a este tipo de usuarios [Draper 84, Kobsa 94]. Cuando el sistema es muy amplio, con aplicaciones muy variadas, como es el caso del Unix, es más correcto hablar de especializaciones en diversos campos que de expertos globales, de modo que a este tipo de usuarios también les resulta útil disponer de este modelo conceptual [Sutcliffe 87].

6.3 Planteamiento y diseño del sistema

El sistema Aran supone una ejemplificación completa de nuestro modelo de sistema de ayuda inteligente. Supone una continuación del trabajo desarrollado en Argos, pero con la inclusión de una representación explícita del conocimiento (fig. 6.1).

El núcleo de Aran es su base de conocimiento, que como ya hemos mencionado incluye un modelo conceptual del sistema operativo y que además mantendrá la información sobre el modelo del usuario. La existencia de una conceptualización explícita del área de aplicación, ayuda a la comprensión de los términos involucrados y de sus interrelaciones en ese dominio [Breuker 90]. Su propósito principal es la estructuración de información, de modo que el usuario pueda localizar la información requerida para solucionar su problema, sin necesidad de tener un extenso conocimiento previo del vocabulario utilizado en los sistemas informáticos.

6.3.1 Tecnologías integradas en Aran

En el sistema Aran se integran las cuatro técnicas propuestas en el modelo y descritas en el epígrafe 4.3.1. Estas tecnologías son: recuperación de información, hipertexto, modelado de usuario y representación explícita del conocimiento.

- *Recuperación de información.* La recuperación de información (RI), permite determinar y recuperar cuáles son los documentos relevantes de un gran corpus de textos, a partir de una determinada consulta. Posibilita que el usuario exprese su necesidad con un lenguaje sencillo como, por ejemplo, mediante el uso del lenguaje natural, no necesitando aprender otro método formal consulta. En Aran se reutiliza el trabajo realizado en Argos, realizándose también un tratamiento de los textos mediante un vocabulario controlado (descriptores). Esto permite soslayar un problema típico de este tipo de sistemas, como es el que el usuario utilice un vocabulario ajeno al dominio de aplicación [Callan 93, Frakes 94].

- *Modelado de usuario.* El modelo del usuario es el componente clave para realizar la adaptación de la ayuda. Se utiliza para complementar las consultas y para reordenar los resultados del módulo de RI, de forma similar a como se hace en Argos. En Aran se dispone de nuevas formas de interacción y de una representación explícita del dominio. Por tanto, este modelo se usa también para implementar estrategias de presentación de información, sobre el modelo del dominio y con los documentos recuperados mediante selección de descriptores. Este modelo se integra completamente con el resto del sistema, ya que usa el mismo formalismo de representación que el modelo del dominio.

- *Hipertexto.* En la interfaz de usuario se utilizan técnicas de hipertexto. La novedad es que en Aran, además de las funcionalidades incorporadas en Argos, se puede interactuar con el modelo del dominio. Este modelo es directamente inspeccionable por el usuario. Mediante el uso del ratón se

puede navegar a través de la taxonomía de conceptos y a partir de cada concepto se puede acceder a la información relacionada. Por ejemplo a las órdenes que realizan una determinada operación o a la descripción textual de un determinado término. Por tanto, estos conceptos son nodos hipertextuales que estructuran e interrelacionan los distintos tipos de información contenida en Aran.

- *Representación explícita del conocimiento.* En Aran la información se estructura y organiza alrededor de la representación explícita del dominio. Para evitar los problemas asociados con la especificidad de los métodos de representación, se ha optado por un entorno estándar de construcción de sistemas inteligentes, Loom [MacGregor 91]. Este formalismo pertenece a la familia de lenguajes de representación descendientes de KL-ONE y como principales ventajas cabe destacar: a) su amplio uso [Wright 93]; b) la disponibilidad de traductores a/desde otros formalismos [Gruber 90]; y c) su eficiencia y capacidad expresiva [Heinsohn 94].

6.3.2 Información reutilizada en Aran

Aran también plasma los distintos tipos de reutilización de información planteados en nuestro modelo (tratadas en el apartado 4.3.2).

- *Reutilización de documentación.* Para proporcionar ayuda, al igual que Argos, Aran reutiliza la documentación electrónica del Unix. Esta información se estructura y organiza teniendo en cuenta el modelo del sistema, de modo que se facilite su acceso y su comprensión a los usuarios.

- *Reutilización de modelos de usuario.* En Aran se parte del modelo desarrollado en Argos, que se enriquece y completa para considerar las nuevas representaciones introducidas. El modelo anterior sigue siendo válido para la interacción en lenguaje natural, pero no permite contemplar algunas posibilidades introducidas por el modelo del dominio. Por tanto, se pasa a utilizar un modelo basado en estereotipos de usuarios [Rich 89]. Como punto de partida se reutiliza el modelo presentado en [Nessen 89, Hecking 88] y utilizado para generación de explicaciones, que se adapta y amplia para incorporarlo nuestro sistema. Además, en nuestro modelo se incluyen nuevos mecanismos de actualización y de mantenimiento de la coherencia, para evitar la existencia de información contradictoria.

- *Reutilización de conocimiento.* En la construcción de la base de conocimiento de Aran, se han analizado distintos sistemas que se centran en el dominio de aplicaciones software [Wilensky 89, So 94, Breuker 90, Selker 92, Hecking 88, Devanbu 91], con el propósito de reutilizar estas experiencias. Este proceso de reutilización se ha visto dificultado por el hecho de que generalmente no se ha podido acceder al código, ni a toda la información necesaria sobre estos sistemas. No obstante, en la base de conocimiento de Aran se han reutilizado,

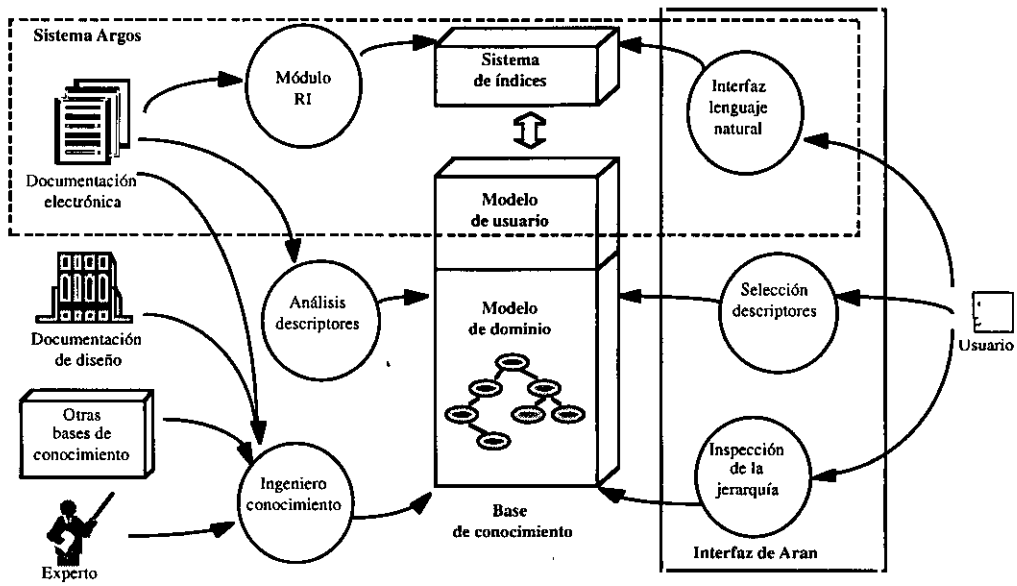


Figura 6.1: Representación esquemática del sistema Aran, tomando como punto de partida el sistema Argos. Se destaca la reutilización de distintos tipos de información y la integración en la base de conocimiento de los modelos de dominio y de usuario.

parcialmente y en distinta medida, las bases de conocimiento de los sistemas descritos en [So 94, Hecking 88, Devanbu 91].

6.4 Arquitectura del sistema

Aran consta de cinco módulos agrupados en tres bloques principales (fig. 6.2): La *interfaz*, que se ocupa de los distintos tipos de interacción con el usuario. La *indexación automática de información* compuesta por: a) *recuperación de información*, que implementa las funciones de indexación de los documentos y cálculo de la similitud según el modelo del espacio vectorial; y b) *caracterización formal de conceptos*, que indexa la documentación en función de palabras clave o descriptores. La *base de conocimiento* compuesta por: a) *modelo del usuario*, que contiene la información disponible sobre el usuario; y b) *modelo del dominio*, que es un modelo explícito e inspeccionable del dominio en torno al cual se organiza el resto de la información. A continuación realizamos una breve descripción de cada uno de estos módulos.

- *Interfaz*. En el diseño de la interfaz se ha buscado la sencillez de manejo, mediante un sistema gráfico de ventanas con menús, iconos y botones, y su operación mediante un ratón como dispositivo apuntador. Para facilitar al usuario el acceso a la información, se proporcionan tres métodos alternativos de interacción. Se puede interactuar directamente mediante una inspección del modelo del dominio, mediante la selección de las palabras clave que

describan la información deseada o realizar la solicitud en lenguaje natural. El usuario elige el método que prefiere y en la interfaz siempre tiene la opción de acceder a los otros modos de interacción.

- *Indexación automática de información.*

  a) *Módulo de recuperación de información.* El módulo de recuperación de información (RI) lleva a cabo la búsqueda de documentación a partir de la petición en lenguaje natural del usuario. Implementa una indexación automática basada en el modelo del espacio vectorial, reutilizando el trabajo desarrollado en Argos, y aplica los mismos mecanismos de mejora de la eficiencia.

  b) *Caracterización formal de conceptos.* El objetivo de este módulo es indexar un documento en función de una serie de términos que describan su propósito. Los descriptores que caracterizan cada texto se obtienen a partir del epígrafe de descripción corta que tienen los documentos del manual electrónico de Unix. Esta indexación permite una recuperación sencilla por parte del usuario, quien sólo tiene que seleccionar, de entre los términos que le ofrece Aran, aquellos que mejor describen su necesidad. Al mismo tiempo, esta recuperación está lógicamente bien fundamentada en el análisis formal de conceptos. De esta forma, se evitan los problemas debidos a una utilización de vocabulario ajeno al dominio, así como a la especificación de descriptores que lleven a la no recuperación de documentos.

- *Base de conocimiento*

  a) *Modelo del usuario.* El módulo del modelo de usuario (MU) tiene por objetivo adecuar la información que se presenta a cada usuario concreto. Se realiza un modelo de usuario pragmático e incremental. El enfoque utilizado se basa en estereotipos o clases prototípicas de usuarios. Se reutiliza parcialmente el modelo previamente desarrollado para Argos, sobre todo para la interacción en lenguaje natural. Este modelo se complementa para tener en cuenta las nuevas fuentes de información incorporadas en Aran, como es el modelo del dominio. Debido a que se trabaja principalmente con información ya existente, la adaptación se traduce en una modificación de los procesos de búsqueda y presentación de esa información.

  b) *Modelo del dominio.* El modelo del dominio es una parte de la base de conocimiento y contiene una representación explícita del conocimiento sobre el sistema operativo Unix. Es directamente inspeccionable por el usuario. Supone una representación de las decisiones de diseño del sistema operativo y en torno a ella se realiza una indexación basada en el conocimiento de la información del sistema. Los elementos principales de

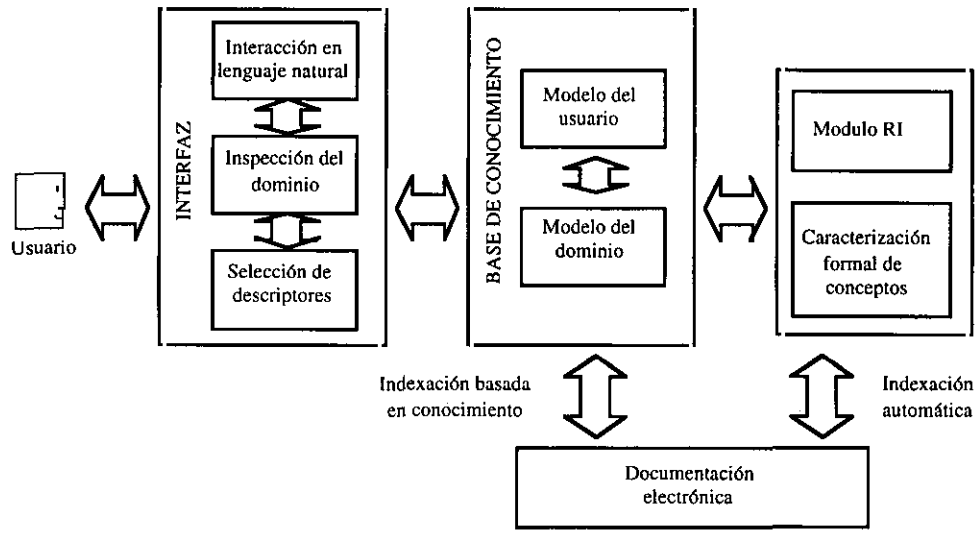


Figura 6.2: Arquitectura de los módulos del sistema Aran

este modelo son los objetos identificados en el dominio y las operaciones que se pueden realizar sobre ellos.

En los apartados siguientes tratamos con detalle los distintos módulos que componen el sistema y los problemas específicos de implementación. En el Apéndice final se incluyen detalles adicionales de codificación.

6.5 Interfaz

La interfaz de Aran permite distintos modos de interacción, que se corresponden con los diferentes tipos de indexación que se realiza sobre la información textual. Los diversos métodos de acceso permiten que el usuario pueda elegir aquel con el que se sienta más cómodo, a la vez que se complementan e interrelacionan para mejorar la recuperación y la efectividad de la ayuda. Por ejemplo, un problema identificado con los sistemas de recuperación de información, que limita mucho su eficiencia, es que el usuario utilice un vocabulario ajeno al campo de aplicación [Callan 93]. Aunque consideramos que el usuario típico de Aran tendrá al menos unos conocimientos básicos de los términos utilizados en un sistema operativo y de las operaciones a realizar, no obstante, incluso si no se poseen estos conocimientos se puede interactuar con el sistema, ya que en la base de conocimiento se incorpora información sobre los conceptos involucrados y su significado concreto en Unix.

Los modos de interacción con Aran son:

- *Interacción en lenguaje natural.* En este tipo de interacción el usuario puede expresar su necesidad de ayuda en lenguaje natural (en inglés) y el sistema

recupera la documentación que considera relevante para esa petición. Tanto la consulta como los documentos se caracterizan por métodos fundamentalmente estadísticos y, mediante una función de calculo de similitud, se determina cuál es la información más apropiada. El proceso de tratamiento de la consulta y de la posterior presentación de los documentos se ve modificado y enriquecido por la utilización de la información contenida en el modelo del usuario. Esto supone una continuación de la línea presentada en el capítulo anterior con el sistema Argos.

- *Interacción mediante selección de descriptores.* Además de la indexación anterior, cada documento se encuentra también indexado individualmente mediante un conjunto de palabras clave, que dan una idea de su funcionalidad y contenido. Estas palabras clave o descriptores se eligen dentro de un conjunto controlado que supone una caracterización suficientemente amplia del dominio. Así, para buscar información el usuario especifica de forma incremental aquellas claves, de entre las que le ofrece Aran, que mejor describen la documentación que busca (fig. 6.3). Este es un proceso incremental donde, en cada paso, mediante el análisis formal de conceptos, se le presentan al usuario los documentos candidatos y el conjunto exacto de descriptores significativos admisibles para refinar su petición, considerando los descriptores elegidos hasta el momento. Este procedimiento garantiza que por lo menos se obtiene un documento, ya que no se pueden especificar claves contradictorias [Lindig 95]. El usuario puede acceder a los documentos recuperados pulsando sobre ellos con el ratón de modo que se presentan en una nueva ventana. El orden de presentación de los documentos se realiza en función del contenido del modelo del usuario.

- *Interacción mediante inspección de la jerarquía de objetos.* El modelo del dominio es inspeccionable por los usuarios y además permite una interacción directa mediante el ratón. Se presenta la taxonomía de conceptos (objetos y acciones) y mediante ellos se puede acceder a la información asociada. Se trata de un grafo que es por tanto una representación gráfica de la red semántica formada por este modelo (aunque habitualmente se presentará como un árbol, tal como muestra la figura 6.4). El tipo de interacción con redes semánticas que se puede proporcionar a los usuarios finales es un problema abierto que no tiene soluciones generales [Mayfield 93]. La solución adoptada en Aran ha sido identificar un conjunto de operaciones útiles para el usuario y proporcionarlas directamente en la interfaz, bien como opciones generales del menú o como acciones asociadas a cada uno de los conceptos (fig. 6.4). Mediante las distintas opciones de menú de la ventana se puede, por ejemplo, acceder a la documentación asociada a un concepto o realizar búsquedas de objetos utilizando expresiones regulares. El acceso a su definición formal, a sus instancias u otras opciones de navegación a través del grafo se presentan asociadas a cada concepto.
  Existen diversas opciones de configuración sobre la visualización de los grafos del dominio y siempre se puede interactuar con ellos mediante el ratón. Sus elementos son nodos de un hipertexto que permiten acceder a nuevos conceptos o instancias (en el epígrafe 6.8.5 se completa la descripción de este tipo de interacción y su influencia en la ayuda y la enseñanza).

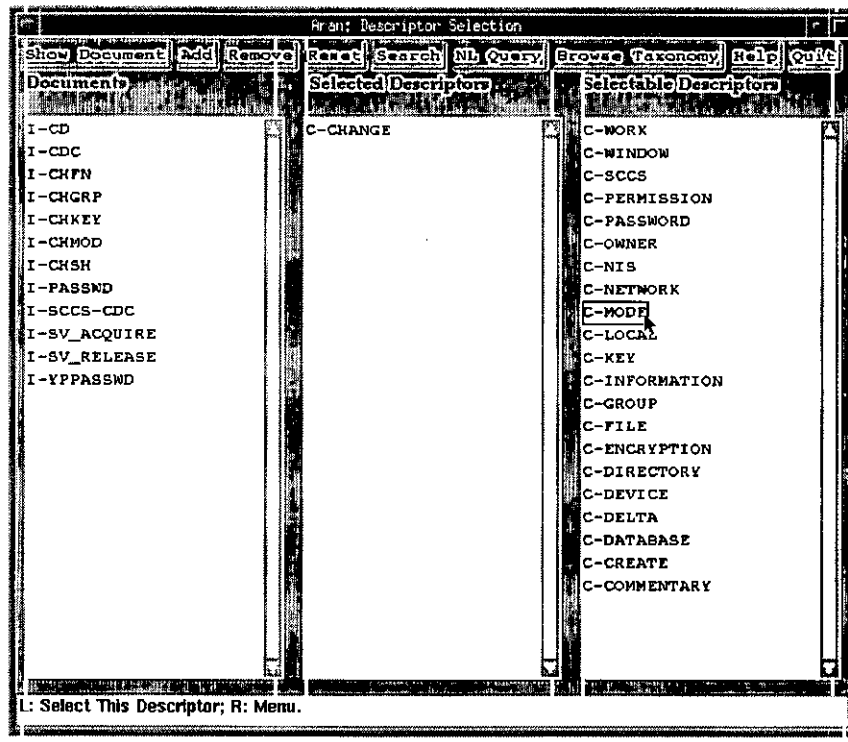


Figura 6.3: Interfaz de Aran para la interacción mediante especificación incremental de descriptores. A la izquierda se muestran los documentos que contienen el descriptor *change* y se está seleccionando el descriptor *mode*.

El contenido del modelo del usuario también influye en esta visualización de la información, como se describe en el apartado 6.7.5. En este modo de interacción se ha reutilizado parcialmente la experiencia y el código del sistema Copérnico [Blanco 95, González-Calero 96]. Copérnico es una herramienta desarrollada por nuestro grupo de trabajo que facilita la creación, visualización y mantenimiento de bases de conocimiento representadas en Loom (se tratará más extensamente en el apartado 6.8.4).

Estos modos de interacción con el sistema no son independientes, siempre se puede pasar de un modo a otro y, además, están integrados e interrelacionados en la interfaz. Por ejemplo, una vez que se ha obtenido un documento adecuado para el usuario, bien mediante lenguaje natural o mediante descriptores, se puede acceder a la inspección del dominio. En este caso, la visualización se realizará centrada en los conceptos que clasifican ese documento, de modo que se obtienen los objetos y/o acciones involucradas, que permiten el acceso a otra documentación relevante.

La integración de técnicas propuesta en nuestro modelo de asistente proporciona otras ventajas además de los diferentes modos de interacción. La utilización de la indexación automática evita situaciones en las que, si no se incluyera, el comportamiento de Aran sería peor que el de un sistema más simple como es Argos.

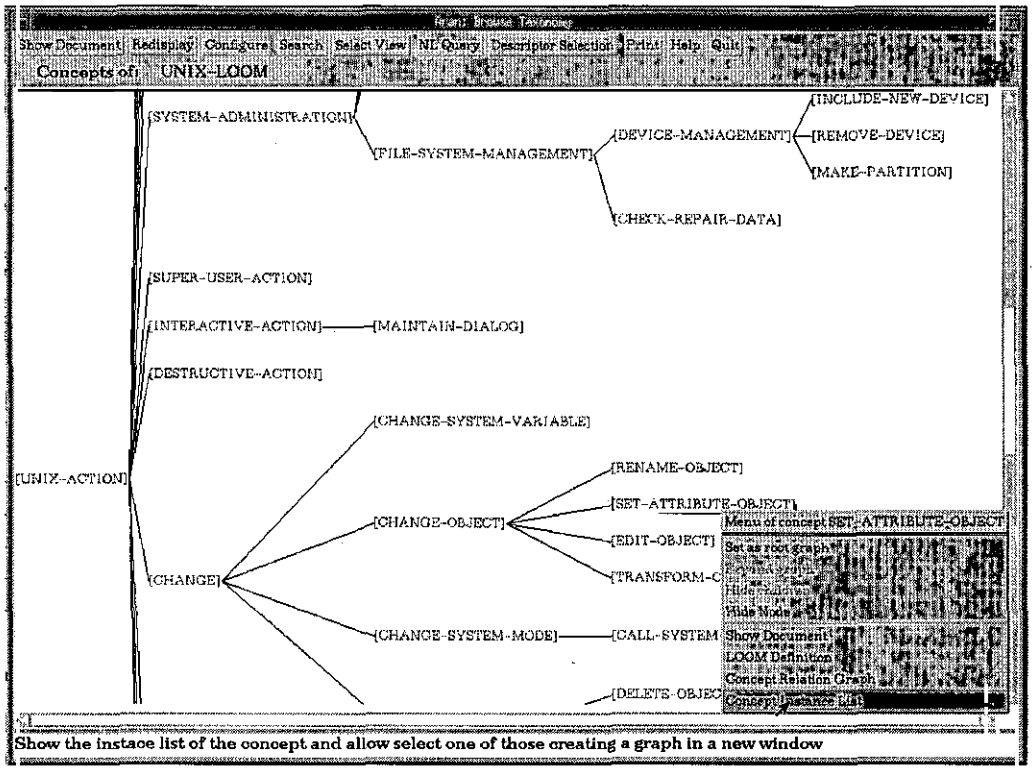


Figura 6.4: Interfaz de visualización del modelo del dominio, presentando únicamente los conceptos, y centrada en las acciones definidas en el Unix (*unix-action*). Se muestra el menú asociado a la acción *set-attribute-object* donde está seleccionada la opción que permite acceder a su lista de instancias

La indexación de la información basada en palabras y en descriptores es útil debido a que puede haber partes del sistema que no se hayan codificado en el modelo del dominio (base de conocimiento parcial) y, por tanto, no sería posible la obtención de información relacionada utilizando únicamente la representación de conocimiento. Otra situación similar se produce cuando se incluyen nuevas utilidades, en cuyo proceso de instalación se introduce su página de manual relacionada. Con estas técnicas automatizables se pueden indexar fácilmente esos documentos (y por tanto acceder a ellos), cuando aún no han sido considerados en el modelo del dominio, puesto que este proceso es manual y más complejo.

Se ha tratado en todo momento de aunar en la interfaz la potencia con la simplicidad de uso. Aran proporciona un entorno gráfico (con menús, botones y uso de ratón) en el que para lograr dichos fines se han aplicando criterios de interacción hombre-computadora y técnicas hipertextuales [Maddix 90, Maybury 93]. No obstante, por si fuera necesario, en todo momento existe un botón de ayuda para obtener asistencia sobre el propósito, el funcionamiento o las operaciones disponibles en cada momento.

6.6 Indexación automática de información

En Aran se realizan dos tipos de indexación de la información: una indexación manual, en función del contenido del modelo del dominio, y otra automática, basada en técnicas de recuperación de información (RI). La indexación automática se implementa mediante dos módulos, el de RI, que realiza una indexación en función de todos los términos de un documento, y el de caracterización mediante descriptores, que realiza una caracterización de los textos mediante palabras clave y con un vocabulario controlado. A continuación se describen con mayor detalle estos dos módulos.

6.6.1 Módulo de RI

El módulo de Recuperación de Información implementa, esencialmente, un algoritmo de cálculo de similitud entre consultas y documentos basado en el modelo del espacio vectorial. Es el mismo modelo que utiliza Argos y también se le aplican los mecanismos de mejora de la efectividad descritos en el apartado 4.4.1.3 (es decir, los pesos de términos, la extracción de raíces, una lista de palabras vacías y la realimentación).

Como se ha descrito en el capítulo anterior, tanto las consultas del usuario como los documentos se caracterizan mediante todos los términos que aparecen en ellos. Se realizan optimizaciones eliminando las palabras sin contenido semántico (palabras vacías) y obteniendo las raíces de las palabras para evitar variaciones con el mismo significado. El peso o la importancia de un término se obtiene a partir de su número de apariciones en el documento concreto y en el corpus de documentos. Además, cada documento se caracteriza mediante sus términos más significativos (de mayor peso). Esto permite la reformulación automática de consultas según la relevancia que tienen para el usuario los textos recuperados. Este conjunto de técnicas presentan dos ventajas frente a otras aproximaciones: su sencillez de implementación y su efectividad [Salton 89, Frakes 94]. Posibilitan que el usuario formule su petición en inglés de modo que no tiene que aprender un lenguaje formal de construcción de consultas y, además, le permiten expresarse de modo muy específico, ya que no se restringe *a priori* el vocabulario a utilizar. El problema clásico de este tipo de sistemas, como ya se destacó en Argos, es que el usuario utilice vocablos diferentes a los presentes en los documentos, haciendo que el proceso de recuperación falle [Callan 93, Araya 90]. El hecho de realizar diferentes indexaciones de la información y de proporcionar métodos alternativos de interacción disminuye este problema. Mediante el modelo contenido en la base de conocimiento o los descriptores, el usuario puede familiarizarse con el vocabulario de este dominio.

Se ha reutilizado el trabajo desarrollado en Argos, usándose el mismo indexador y el mismo caracterizador de documentos. El cálculo de la similitud y la interfaz se han recodificado en Lisp para poder integrarlos con el resto de los módulos de Aran. Más detalles de las fórmulas utilizadas, así como sobre la implementación, se pueden encontrar en el capítulo anterior y en [Cigarrán 96].

6.6.2 Módulo de ayuda basado en la caracterización formal de conceptos.

Los documentos del manual de Unix incluyen bajo el epígrafe *name* una descripción de una o dos frases de la funcionalidad de la orden que documentan. Esta descripción se considera una especificación privilegiada de la orden, frente a la explicación más detallada que contiene el resto del texto. En Aran consideramos privilegiadas estas descripciones cortas en el sentido de que utilizan un vocabulario (un conjunto de términos de alto contenido semántico) que describe de forma muy precisa el dominio y que es comprensible para un amplio rango de usuarios. Aran utiliza los términos de estas descripciones privilegiadas para construir una representación de los conceptos básicos del sistema y para proporcionar al usuario una forma alternativa de acceso a la documentación. Aran obtiene este vocabulario de descriptores y lo pone a disposición del usuario para que éste pueda utilizarlo en sus peticiones de ayuda. Con este fin, Aran incluye el módulo que describimos en éste y siguientes apartados y que básicamente consta de una interfaz amigable en la que subyace un potente mecanismo de clasificación e indexación.

Como ya se indicó en la explicación sobre los sistemas de ayuda (en el apartado 3.4.6 sobre los modelos conceptuales), cuando el usuario encuentra un problema que no sabe resolver, su primera tarea es identificar cuál es el problema para luego formular la petición correspondiente al asistente. La obtención de ayuda en un sistema pasivo como Aran, se basa en la formulación de sucesivas preguntas hasta que se obtiene la información deseada. No obstante, y este es el punto importante, la adecuada formulación de las solicitudes por el usuario depende en gran medida de su conocimiento del dominio y más concretamente del correcto conocimiento de la terminología que se emplea en el mismo. El objetivo de este módulo es dar al usuario una alternativa que minimice el problema de tener que conocer *a priori* esta terminología específica. Para ello en Aran se indexan las órdenes Unix mediante los términos de contenido semántico específico que aparecen en las descripciones cortas, a los que anteriormente nos hemos referido como descriptores. La interfaz de este módulo proporciona un sencillo método de acceso mediante el cual el usuario puede construir sus consultas basándose en una lista de descriptores seleccionables que le es ofrecida explícitamente. Así se logra una forma sencilla y eficiente de acceso a la información, ya que el usuario no necesita conocer previamente los términos adecuados, sino únicamente identificar aquellos, de entre los propuestos, que mejor describen su problema.

6.6.2.1 Obtención de descriptores

Para obtener el conjunto de términos adecuados para la indexación aplicamos técnicas estándar de RI al conjunto de las descripciones cortas. Como en el caso de una representación en función de palabras clave con vocabulario controlado, primero obtenemos un conjunto de descriptores fijos capaces de especificar cualquier documento de ese dominio [Salton 89]. Para lograr este conjunto de claves, se hace un tratamiento semiautomático de las descripciones cortas (además de en cada página, las descripciones cortas también se encuentran agrupadas en el archivo de introducción de cada una de las secciones del manual: *man intro <número_sección>*). Este proceso es similar al utilizado por Argos en el

procesamiento de documentos y consultas. Primero se filtran las palabras vacías (se utiliza la misma lista de parada que en Argos), y de las palabras así obtenidas se eliminan manualmente los términos redundantes, como los plurales y otras variaciones con similar significado. Este conjunto de descriptores se podría reducir aún más, mediante la utilización de un algoritmo de obtención de raíces de palabras, que a la vez evitara el proceso manual de suprimir los términos redundantes, pero en este caso no sería adecuado, ya que estos descriptores también se utilizan como modo de interacción en la interfaz. El uso de un algoritmo de reducción de palabras (como el de Porter [Frakes 92] utilizado en Argos), provocaría una sobre-radicación que en muchos casos proporcionaría términos poco comprensibles y difícilmente reconocibles para el usuario. En la figura 6.5 se muestra un ejemplo de cuatro órdenes de Unix, con su descripción corta y los descriptores asignados.

6.6.2.2 Análisis formal de conceptos en Aran

El cómputo eficiente de las órdenes a recuperar como respuesta a una petición de ayuda del usuario y el cómputo eficiente de los descriptores significativos a proporcionar al usuario para guiarle en su petición de ayuda, están basados en el análisis formal de conceptos [Davey 90]. Utilizando este análisis se definen los conceptos del sistema como pares formados por un conjunto de órdenes Unix y un conjunto asociado de descriptores. Estos conceptos se estructuran como un retículo completo.

En el análisis formal de conceptos, un concepto está determinado por su extensión (*extent*) y por su definición (*intent*). La extensión es el conjunto de todos los objetos que pertenecen al concepto y la definición es el conjunto de los atributos comunes a todos los objetos que pertenecen al concepto. Como la determinación completa de un concepto puede ser muy compleja, se define un *contexto* para fijar (limitar) el número de objetos y de atributos a considerar en un dominio conceptual concreto. Así pues, un contexto se define como una tupla *(G, M, I)*, donde G es un conjunto de objetos, M es un conjunto de atributos e I es una relación binaria $I \subseteq G \times M$. Mediante gIm se denota que el objeto g tiene el atributo m. Para todo subconjunto A y B, $A \subseteq G$ y $B \subseteq M$, se definen los conjuntos

$$A' = \{m \in M | (\forall g \in A) g \, \mathrm{Im}\},$$
$$B' = \{g \in G | (\forall m \in B) g \, \mathrm{Im}\};$$

de este modo, A' es el conjunto de todos los atributos comunes a todos los objetos en A y B' es el conjunto de todos los objetos que tienen en común todos los atributos en B. Entonces un concepto del contexto *(G, M, I)* es la dupla *(A, B)*, donde $A \subseteq G$ y $B \subseteq M$ y $A' = B$ y $B' = A$. Es decir, la dupla *(A, B)* determina un concepto si se cumple que B es el conjunto de todos los atributos comunes a todos los objetos en A (su definición) y A es el conjunto de todos los objetos que tienen en común todos los atributos en B (su extensión). En consecuencia, un subconjunto A de G es la extensión de algún concepto, si y sólo si $A'' = A$, en cuyo caso el único concepto del cual A es su extensión es *(A, A')*. De modo paralelo esto también es aplicable a los subconjuntos B de M que sean la definición de algún concepto.

Orden	Descripción corta	Descriptores
cd	change working directory	*change work directory*
chgrp	change the group ownership of a file	*change group owner file*
chkey	create or change encryption key	*create change encryption key*
chmod	change the permissions mode of a file	*change permission mode file*

Figura 6.5: Ejemplo de órdenes de Unix junto con sus descripciones cortas y los descriptores asignados

El conjunto de todos los conceptos del contexto *(G, M, I)* se denota mediante β *(G, M, I)*. Entre los conceptos (A_1, B_1) y (A_2, B_2) de β *(G, M, I)* se define la siguiente relación de orden parcial: $(A_1, B_1) \leq (A_2, B_2)$, y se dice que (A_1, B_1) es un subconcepto de (A_2, B_2) o que (A_2, B_2) es un superconcepto de $(A1, B_1)$, si A_1 es subconjunto de A_2 (o lo que es equivalente, si B_2 es un subconjunto de B_1). El conjunto de todos los conceptos de un contexto con la relación de orden así definida forma un retículo completo, que se denota $\langle \beta(G, M, I); \leq \rangle$ y que se denomina retículo de conceptos del contexto *(G, M, I)*.

Aplicando este análisis a nuestro dominio se define el contexto *(G, M, I)*. Donde *G* es el conjunto de todas las órdenes Unix comprendidas en la sección uno del manual, *M* es el conjunto de todos los descriptores utilizados en las descripciones cortas de estas órdenes (los términos obtenidos por el proceso previamente descrito) e *I* es la relación binaria, *gIm*, que representa que en la descripción corta de la orden *g* se utiliza el descriptor *m*. Para todo *A* que sea un subconjunto de las órdenes de Unix y para todo *B* que sea subconjunto de los descriptores, los conjuntos *A'* y *B'* serán: *A'* el conjunto compuesto por todos los descriptores comunes a todas las órdenes en *A* y *B'* el conjunto formado por todas las órdenes Unix que tienen en común todos los descriptores en *B*.

De esta forma, en el contexto de las órdenes Unix y de sus descripciones cortas, un concepto puede caracterizarse como la dupla *(A, B)* siempre que se cumpla que *A'* (el conjunto de todos los descriptores comunes a todas las órdenes en *A*) sea igual a *B* (su conjunto de descriptores). De forma recíproca se puede caracterizar un concepto partiendo del conjunto B'. Además, en nuestro dominio también un subconjunto de órdenes *A* será la extensión de un concepto, si y sólo si $A'' = A$ y ese concepto estará representado por la dupla *(A, A')*. Es decir, si no existen más órdenes en el sistema que tengan en común el mismo conjunto de descriptores. De la misma forma, un subconjunto de descriptores *B* será la definición de un concepto en nuestro dominio, si y sólo si $B'' = B$ y el concepto estará representado por la dupla *(B', B)*. Nos interesa considerar los dos casos extremos que estas definiciones plantean:

a) cuando el conjunto *A* se reduce a una sola orden *g*

b) cuando el conjunto B se reduce a un sólo descriptor m

En el primer caso obtenemos que para casi todas las órdenes, g, de nuestro dominio $\{g\}'' = \{g\}$, o lo que es lo mismo, casi todas las órdenes de nuestro dominio definen el concepto $(\{g\}, \{g\}')$. Como excepciones a esta regla cabe destacar las órdenes que comparten una misma página de manual, como por ejemplo, *compress* y *uncompress* que realizan tareas opuestas porque comprimen y descomprimen archivos respectivamente. Además, estos son los conceptos más específicos del sistema, es decir, los conceptos que no tienen subconceptos, y que están caracterizados de forma unívoca por sus descriptores. Otro caso peculiar son los grupos de órdenes que permiten formas alternativas de realizar más o menos la misma tarea como, por ejemplo, *at* y *batch,* que permiten la ejecución de trabajos a una hora determinada. Visto desde este punto de vista, el proceso de ayudar al usuario a encontrar con que orden puede realizar una determinada tarea, es el proceso de guiarle para encontrar los conceptos más específicos del sistema (aquellos que no tienen subconceptos).

El análisis del segundo caso nos lleva a reconocer que en nuestro dominio no existen muchos conceptos que estén caracerizados por un único descriptor m. Es decir, en nuestro dominio generalmente se cumple que $\{m\}'' \supset \{m\}$. En este caso, el concepto asociado a un descriptor será el par $(\{m\}', \{m\}'')$ formado por todas las órdenes del dominio que utilizan el descriptor y por el conjunto de todos los descriptores comunes a las mismas. Por ejemplo, el concepto asociado al descriptor *directory* es *({cd, dircmp, du, ls, mkdir, organizer, pwd, rm, rmdir, whois}, {block, change, compare, content, disk, display, file, internet, ip, list, make, manager, name, number, pathname, remove, service, tcp, unlink, user, work.})*. Desde este punto de vista, y empezando por un solo descriptor, lo que obtenemos (en el caso general) es un concepto que si tiene subconceptos y el proceso de ayuda podemos concebirlo como el de facilitar al usuario el concepto más general al que se refiere el descriptor inicialmente seleccionado. En el ejemplo, al seleccionar *directory* se obtienen las posibles formas de manipular directorios en el sistema (mediante *cd, dircmp, du, ls, mkdir, organizer, pwd, rm, rmdir*) y además como se puede trabajar con el servicio de directorio de nombres de Internet (mediante *whois*).

Existen diversas alternativas para representar mediante un retículo completo todos los conceptos de un contexto dado [Davey 90]. La que mejor se adapta a nuestro propósito es un diagrama de Hasse que utiliza simultáneamente órdenes (objetos) y descriptores (atributos) y proporciona un análisis del dominio en el que cada descriptor es el supremo (*joint*) del conjunto de órdenes en cuya descripción interviene, y cada orden es el ínfimo (*meet*) del conjunto de descriptores utilizados para describir la orden. En esta representación, los conceptos no aparecen de forma explícita, es decir, no aparecen como nodos del retículo, pero tiene la ventaja que permite visualizar mejor el proceso de ayuda esbozado en los párrafos anteriores (fig. 6.6).

Basándonos en la discusión anterior, el modo de ayuda que este módulo brinda al usuario tiene como punto de partida las extensiones correspondientes a cada descriptor individual. Es decir, siendo m el descriptor inicialmente seleccionado por el usuario de entre el conjunto de todos los descriptores, se construye $\{m\}'$ que es el

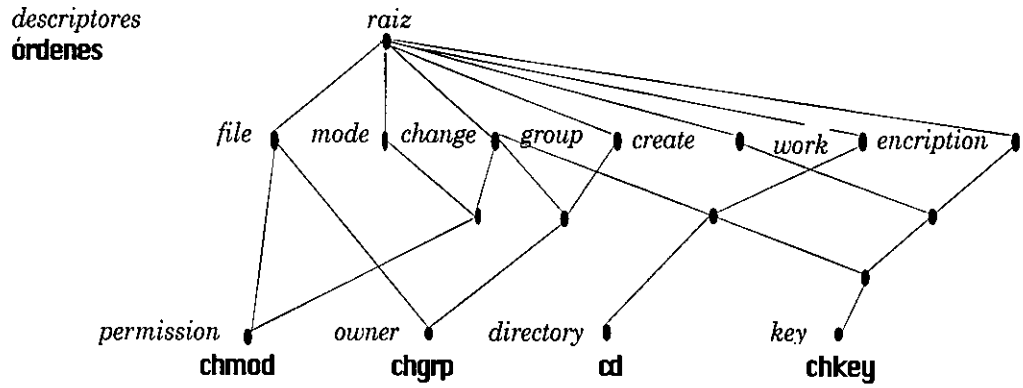


Figura 6.6: Diagrama de Hasse del retículo de conceptos para las cuatro órdenes de Unix previamente descritas

conjunto de todas las órdenes que utilizan el descriptor *m*. A continuación se construye el conjunto *S* de todos los descriptores utilizados en las órdenes en {m}' y se obtiene *{m}"*: el conjunto de todos los descriptores comunes a todas las órdenes de *S*. Entonces *({m}',{m}")* es el concepto más general del dominio que incluye el descriptor *m* y sobre el que el usuario recibe la información pertinente. Parte de esta información consiste en facilitar al usuario el conjunto $S \setminus \{m\}"$ de los descriptores seleccionables, aquellos que pueden llevarle a conceptos (subconceptos) más específicos.

En el caso de que el conjunto $S \setminus \{m\}$ de descriptores seleccionables no sea el conjunto vacío, el usuario puede seleccionar nuevamente un descriptor *d* de este conjunto. En el paso siguiente *{m,d}'* nos dará el conjunto de órdenes que utilizan ambos descriptores. Volvemos a formar un nuevo conjunto *S* con todos los descriptores utilizados en todas las ordenes seleccionadas en este nuevo paso y se construye de nuevo, *{m, d}"* , entonces *({m,d}',{m,d}")* es un subconcepto sobre el que el usuario recibirá más información. Nuevamente, la diferencia $S \setminus \{m, d\}"$ será el conjunto de descriptores seleccionables para encontrar un subconcepto de *({m, d}', {m, d}")* si el usuario tiene necesidad de información más específica. El proceso termina cuando la diferencia $S \setminus \{m, d\}"$ es el conjunto vacío, el concepto no tiene subconceptos y no se puede ser más específico con respecto a la información a suministrar al usuario. Como indicamos al comienzo de esta discusión esta será la situación cuando lleguemos a la mayoría de las órdenes del sistema.

6.6.2.3 Implementación del módulo y ejemplos de uso

Para implementar este módulo, en Aran se crea una base de conocimiento terminológica en la que los descriptores, alrededor de 500 para la primera sección del manual, se declaran como conceptos primitivos, agrupados debajo de un concepto *raiz* (que es el nombre dado al origen en esta representación). Por ejemplo, la definición en Loom de los conceptos *change*, *mode*, *file* y *permission* que intervienen en la descripción de *chmod* (fig. 6.5) es:

```
(defconcept c-change :is-primitive raiz)
(defconcept c-mode :is-primitive raiz)
(defconcept c-permission :is-primitive raiz)
(defconcept c-file :is-primitive raiz)
```

Aplicando el mismo proceso de obtención de términos a cada una de las descripciones cortas, se consiguen los descriptores de cada orden del sistema Unix. Las órdenes del sistema se declaran entonces como instancias de los conceptos terminológicos. Por ejemplo, la representación de las órdenes descritas en la figura 6.5 es:

```
(tell (:about i-cd
    c-change c-work  c-directory))
(tell (:about i-chgrp
    c-change c-group c-owner c-file))
(tell (:about i-chkey
    c-create c-change c-encryption c-key))
(tell (:about i-chmod
    c-change c-permission c-mode c-file))
```

La orden *chmod*, por ejemplo, se declara como una instancia de los conceptos *change, mode, permission* y *file*. Esto significa que esta orden queda indexada por estos conceptos, de modo que cuando el usuario seleccione cualquiera de ellos, *chmod* aparecerá como una orden candidata (fig. 6.7 y también la fig. 6.3 que muestra esta operación desde el interfaz). Además, durante los sucesivos pasos de la formulación de la solicitud, se hace uso del retículo para obtener el conjunto de órdenes y el conjunto de descriptores, que son compatibles con la especificación parcial realizada hasta ese momento. De esta forma, se obtendrán, entre otros, el resto de los términos que intervienen en la descripción de *chmod*. El usuario puede refinar la petición con alguno de los descriptores compatibles, pero en este proceso se tiene la garantía de que por lo menos se obtiene como resultado una orden (y mediante ella se podrá acceder al documento asociado). En la figura 6.7 se muestra un ejemplo de la especificación incremental de la petición, concretamente, se obtiene un descriptor (*file*) que ya no sirve para especializar la solicitud, por tanto aunque el usuario no lo haya seleccionado expresamente en el interfaz aparecerá como un descriptor seleccionado (y no como un descriptor seleccionable). En la figura 6.8 se muestra este proceso de forma gráfica sobre el diagrama de Hasse del retículo.

6.6.2.4 Conclusiones y trabajo relacionado

En Aran al considerar las descripciones cortas como privilegiadas se obtienen dos ventajas: por un lado se realiza un proceso semiautomático de indexación, con un coste no muy elevado y que permite reducir el número de descriptores; y por otro lado se proporciona un método simple, a la vez que potente y bien fundamentado, de formulación de peticiones.

Existen otros sistemas en este mismo dominio como Proteus [Frakes 94] o [Lindig 95] que utilizan una aproximación similar de indexación por descriptores o palabras

paso	descriptores seleccionados	utilidades	descriptores seleccionables
1	--	todas	todos
2	*change*	chsh chfn passwd sccs-cdc cd sv_release sv_acquire chgrp chmod yppasswd chkey	work window permission owner nis network sccs password mode key information group file encryption directory device delta database create commentary
3	*change mode*	chmod sv_acquire sv_release	window file device owner permission group
4	*change mode permission*	chmod	***file***

Figura 6.7: Construcción incremental por el usuario de una solicitud mediante la especificación, en sucesivos pasos, de los descriptores *change, mode y permission*. Como *file* ya no permite especializar más la petición (para el conjunto de órdenes actualmente indexadas) en el interfaz aparecería como un descriptor seleccionado y no como un descriptor seleccionable.

clave. Las diferencias con Aran son fundamentalmente el tipo e implementación de la indexación, el número de descriptores usados y la especificación de la búsqueda. En Aran se indexan 472 órdenes de la sección 1 del manual, utilizando 551 términos y únicamente se crean 331 nuevos conceptos que indexan órdenes. Proteus es un sistema para comparar distintos enfoques en la caracterización y recuperación de componentes software, también indexa automáticamente las utilidades del Unix. En las búsquedas permite la utilización de operadores booleanos y se usan aproximadamente 3000 descriptores. El tipo de búsqueda junto con un número excesivo de términos complica la interacción con el usuario (no explican como obtienen los descriptores, pero no realizan ningún filtrado de términos redundantes). El prototipo de Lindig es un sistema para demostrar la utilidad de los retículos y del análisis formal de conceptos en la recuperación de componentes software. Utiliza sólo 92 términos, obtenidos manualmente y con los que se realiza una indexación manual. Está centrado exclusivamente en la recuperación y en las búsquedas admite la especificación incremental de claves.

Este enfoque del uso del análisis formal de conceptos tal y como se ha aplicado en Aran es generalizable a otros entornos. Por una parte se obtiene una recuperación rápida y sencilla, a la vez que se organiza la información a facilitar al usuario como ayuda en torno a los conceptos definidos por el propio contexto del sistema. Además, se mantiene un coste bajo del proceso de indexación por lo que es aplicable a grandes colecciones de documentos (u otros elementos que se puedan caracterizar

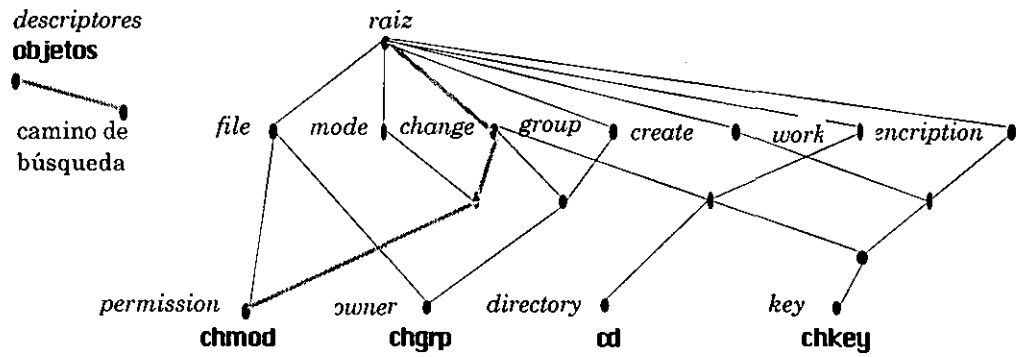


Figura 6.8: Diagrama de Hasse del retículo de conceptos para las cuatro órdenes de Unix previamente descritas, en el que se muestra el camino de búsqueda para obtener *chmod* descrito en la figura 6.7.

mediante descripciones similares a las descripciones cortas que nosotros hemos utilizado).

6.7 Modelado del usuario en Aran

El punto de partida del modelado de usuario en Aran es el modelo desarrollado en Argos, que se enriquece y completa para tener en cuenta las nuevas formas de interacción y las nuevas representaciones de la información introducidas por Aran [Fernández-Manjón 95a].

En Argos se ha obtenido un modelado efectivo y sencillo de implementar pero de grano grueso. Las categorías de usuarios son muy amplias, ya que sólo se pueden considerar dos características (interés de uso y nivel de experiencia), y el tipo de suposiciones que se pueden hacer son limitadas (p.e. que a un usuario le interesarán más los documentos de propósito general correspondientes a su nivel de experiencia, o los documentos de propósito específico si está clasificado dentro de esa categoría concreta). En consecuencia en Argos el poder predictivo del modelo es limitado, ya que no es posible hacer suposiciones sobre qué conceptos son conocidos o desconocidos para el usuario.

En Aran se aprovecha la conceptualización explícita que se hace del sistema operativo Unix (modelo del dominio) para realizar un modelado de grano más fino, que incluye los mecanismos propuestos en la bibliografía para obtener un modelo efectivo [Kobsa 95, Kay 95]. Este es un modelado más general que proporciona las siguientes funcionalidades para: a) realizar suposiciones sobre el usuario a partir de su interacción con la aplicación; b) representar y almacenar estas suposiciones; c) realizar inferencias a partir de esas suposiciones; d) mantener la coherencia de las suposiciones sobre el usuario, mediante procesos que detecten y solucionen las inconsistencias; y e) proporcionar a Aran las suposiciones sobre el usuario cuando las necesite para adaptar su comportamiento al usuario. Este objetivo de adaptación se concreta en la modificación de la búsqueda y del orden de

visualización de documentos, así como en la implementación de diferentes estrategias de presentación de la información particularizadas al conocimiento e interés del usuario.

6.7.1 Caracterización del modelado

El modelo de usuario de Aran, al igual que el de Argos, es incremental, con adquisición automática y mixta por el usuario [Kobsa 92, Self 90a, Chin 93] (en Argos se tratan estas características con más detalle). Se realiza un modelado pragmático [Orwant 95], tratando de obtener un conjunto suficiente de datos para obtener un comportamiento adaptativo. El fin es que se puedan realizar inferencias sobre el conocimiento del usuario, que permitan implementar estrategias adecuadas de presentación de información, sin pretender conseguir un modelado completo de su estado cognitivo.

El enfoque utilizado para el modelado del usuario es el de *estereotipos* o tipos genéricos de usuarios que permite una mayor flexibilidad que las clases de usuarios utilizadas en Argos. Esta aproximación se ha mostrado muy útil en aplicaciones en las que es necesario disponer de información rápida, aunque no sea necesariamente completa ni muy detallada, sobre los conocimientos previos del usuario [Rich 89]. Hay ciertas características humanas que normalmente se presentan agrupadas y la idea subyacente es aprovechar estas regularidades para realizar predicciones. Es habitual que las personas presupongan bastante información sobre otros individuos, a partir de la observación de algunas características que ellos consideran clave. Así se obtiene una orientación rápida, aunque no sea completamente precisa. Estas suposiciones sobre el utilizador se basan en una cantidad de evidencias relativamente pequeña, que es posible adquirir antes de tener que tomar decisiones.

Para implementar esta aproximación hay que completar tres tareas:

a) la identificación de tipos de usuarios.

b) la identificación de las características claves de estos tipos (los atributos y sus posibles valores).

c) su representación mediante estereotipos jerárquicamente ordenados.

En una primera fase, hay que identificar subgrupos dentro de los posibles usuarios, cuyos miembros es muy probable que tengan características similares con respecto a la aplicación, como el mismo nivel de conocimientos o los mismos intereses. Una vez que se tienen estos grupos o tipos, hay que detectar un número reducido de características clave que permitan reconocer a los miembros de un determinado tipo (la presencia, o ausencia, de dichas características debería ser automáticamente detectable por el programa). Por último, una vez identificados los tipos y sus características distintivas es preciso formalizarlas en un determinado sistema de representación. La colección de características distintivas de un determinado tipo de usuarios, es decir cuáles son sus atributos y sus posibles valores, es lo que se denomina estereotipo. Si el contenido de un estereotipo es un subconjunto de otro, o

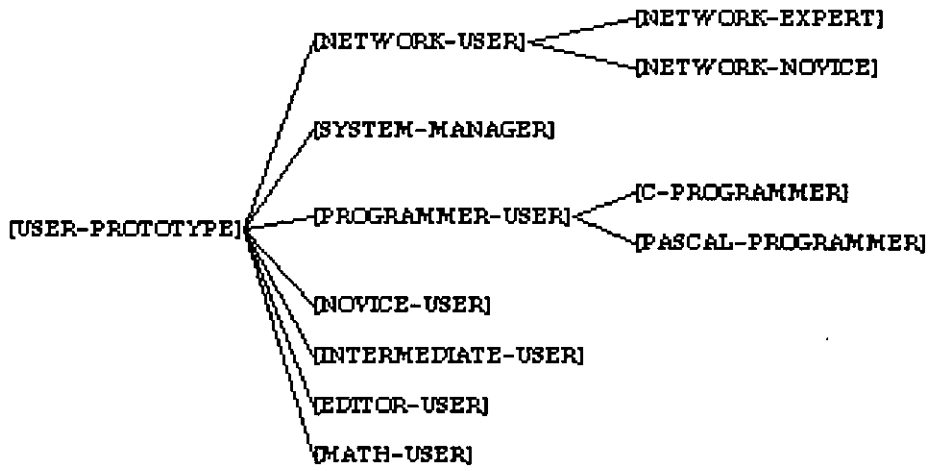


Figura 6.9: Jerarquía de estereotipos de usuario definida en Aran

se puede construir como la combinación de dos previamente existentes, se debe construir una jerarquía con herencia. De este modo el contenido de los estereotipos más generales es heredado por los subordinados, evitando la repetición de datos comunes [Kobsa 94].

En Aran actualmente se distinguen once tipos diferentes de usuarios como, por ejemplo, usuarios de la red con distintos grados de experiencia, o usuarios que realizan tareas de programación. Los estereotipos de usuario de Aran se organizan jerárquicamente a partir de la raíz *user-prototype,* formando un grafo dirigido acíclico (fig. 6.9), de modo que sería posible incluir nuevos estereotipos, o definir otros en función de los ya existentes (siempre y cuando se respeten las restricciones mencionadas más adelante). Esta jerarquía es estática y los usuarios se clasificarán como instancias de uno o más de los estereotipos (conceptos) que la componen. En la definición de los estereotipos se declara también cuáles son las dependencias y restricciones entre ellos. Existen especializaciones de estereotipos que son compatibles, como, por ejemplo, que un usuario se clasifique de forma simultánea como programador en lenguaje C y en Pascal (un usuario que sea instancia de *c-programmer* y *pascal-programmer*). Por el contrario, existen otras que no son compatibles, ya que no sería razonable que se considere que un usuario tiene un conocimiento avanzado de la red y a la vez sea un novato en su uso (un usuario que sea instancia de *network-novice* y *network-expert*).

Otra de las ventajas de este modelo es que, con una adecuada representación, permite hacer suposiciones sobre el conocimiento o desconocimiento de conceptos del dominio. En los estereotipos se establecen correspondencias entre las órdenes del sistema operativo y los conceptos de la representación del dominio. Así, en función de las órdenes que ha utilizado un usuario y que por tanto conoce, se puede presuponer cuáles son los conceptos con los que está familiarizado y con cuáles no, hasta que se disponga de una información más precisa.

La representación del modelo del usuario se realiza con el mismo formalismo de representación del conocimiento utilizado para el modelo del dominio. Con esta elección se persiguen dos objetivos: a) tener una completa integración con el resto del sistema Aran; b) utilizar los mismos recursos de clasificación automática, de herencia o de detección de incoherencias, para simplificar la adquisición y mantenimiento del modelo.

La clasificación de los grupos de usuarios (estereotipos) realizada en Aran no pretende ser completa para el dominio del Unix. Podrían haberse definido más grupos o especializado alguno de los existentes, pero según ha quedado demostrado en las pruebas realizadas con el sistema es suficiente, ejemplificando la potencia y la validez de nuestro enfoque.

6.7.2 Contenido y definición de los estereotipos de Aran

En Aran todos los estereotipos tienen el mismo esquema de representación, con un número fijo de relaciones definidas en la raíz y que heredan todos, pero cuyo contenido se particulariza para cada una de los grupos de usuarios identificados. De esta forma se pueden fijar cuáles son los conceptos y órdenes que se suponen conocidos o desconocidos, por el hecho de pertenecer a un determinado estereotipo. Además, ésta es la información que heredará un usuario cuando se le aplique este estereotipo, de modo que inicialmente y en ausencia de otros datos más concretos, se pueda presuponer qué es lo que conoce, lo que no conoce o cuál es su interés.

En la figura 6.10 se muestra la definición (en Loom) de *user-prototype* que es el concepto raíz de la jerarquía de estereotipos de usuario. Posteriormente mostraremos como ejemplo la definición detallada uno de los estereotipos de usuario (*Network-Novice*). Detalles adicionales sobre la codificación de los estereotipos puede encontrarse en los apéndices finales.

En el esquema utilizado para la representación de los estereotipos se pueden diferenciar tres tipos de datos: los de identificación del usuario, los datos objetivos, sobre los que se tiene una certeza completa, y por último las suposiciones que se realizan sobre el usuario. En la raíz (en *User-Prototype*) estos tres bloques de datos no tienen almacenado ningún valor. Los dos primeros se inicializan y actualizan cuando el usuario interacciona con Aran. El tercer bloque, es decir las suposiciones, se inicializa en la definición de cada uno de los estereotipos de la jerarquía y posteriormente se actualiza durante la interacción con el usuario. A continuación se describen con más detalle:

- *Identificación del usuario:* Las tres primeras relaciones de la definición de *user-prototype* (*has-identifier*, *has-id* y *has-group*) se utilizan para identificar al usuario. Almacenan respectivamente el identificador de ese usuario en el sistema operativo, su número de identificación y el número de grupo al que pertenece. Estos datos los adquiere Aran directamente del sistema operativo en el momento de su invocación.

```
(defconcept User-Prototype
 "concepto que agrupa los estereotipos reconocidos en Aran"
 :is-primitive
 (:and Unix-Thing

       (:the has-name Identifier)
       (:the has-id Number)
       (:the has-group Number)

       (:exactly 1 term-list)
       (:exactly 1 history-list)
       (:all accessed-documents Unix-Thing)
       (:all accessed-concepts Unix-Thing)

       (:all key-commands Unix-Action)
       (:all presupposed-known-commands Unix-Action)
       (:all presupposed-unknown-commands Unix-Action)
       (:all presupposed-known-concepts  Unix-Thing)
       (:all presupposed-unknown-concepts Unix-Thing)
      ))
```

Figura 6.10: Definición del concepto *user-prototype*

- *Datos objetivos*: Las relaciones *term-list, history-list, accessed-documents* y *accessed-concepts*, recogen los datos que se han adquirido directamente de la interacción Aran-usuario o de la interacción previa Unix-usuario, sin realizar ningún tipo de suposición. La relación *term-list* se utiliza para la interacción en lenguaje natural. Almacena los pares término-peso obtenidos a partir de las consultas y de la realimentación proporcionada por el usuario (equivale a la memoria a corto plazo de Argos). Mediante *history-list* se almacena el histórico de órdenes utilizadas por el usuario en el sistema Unix, antes de acceder a Aran, y que a diferencia de Argos aquí se almacena expresamente en el MU. Su propósito es poder realizar una clasificación inicial de ese usuario. Con las relaciones *accessed-documents* y *accessed-concepts* se consideran los documentos y los conceptos a los que se ha accedido, respectivamente.

- *Suposiciones:* Las cinco últimas relaciones de *user-prototype* (*key-commands, presupposed-known-commands, presupposed-unknown-commands, presupposed-known-concepts* y *presupposed-unknown-concepts*) almacenan las suposiciones que se realizan sobre los usuarios. Permiten representar cuáles son las órdenes y los conceptos del modelo del dominio que se suponen conocidos o desconocidos. Los valores de estas relaciones se particularizan, con la información que se supone conocida o no, en la definición de los estereotipos concretos. La relación *key-commands* contiene las órdenes cuya utilización por un usuario determina directamente la aplicabilidad de dicho estereotipo.

Seguidamente se presenta la definición del estereotipo de un usuario que utiliza la red, pero que está poco familiarizado con su uso (*network-novice*) y por tanto no se puede presuponer que conozca o entienda todos los conceptos relacionados:

```
(defconcept Network-Novice
  "Un usuario poco familiarizado con la red."
  :is-primitive
  (:and Network-User
        (:filled-by presupposed-unknown-concepts local-network
         protocol file-server )
        (:filled-by presupposed-unknown-commands "ftp" "uucp"
          "telnet")
        (:filled-by presupposed-known-commands "mosaic"
"netscape"))
:in-partition $Network-User$
)
```

Este estereotipo especializa la información contenida en la definición de sus superconceptos en la jerarquía, ya que como se ha mencionado previamente existe herencia. En este caso particular los conceptos y órdenes conocidos o desconocidos, se complementan con los definidos en su concepto padre *Network-User*. Es importante destacar que la cláusula *:in-partition $Network-User$,* impone la restricción de que la clasificación de un usuario en este estereotipo es incompatible con su pertenencia a cualquier otra especialización de *Network-User*. De esta forma, la definición de los estereotipos fija las órdenes y agrupaciones de órdenes, cuyo conocimiento o desconocimiento, permitirá la clasificación de un usuario como perteneciente a un estereotipo.

6.7.3 Adquisición del modelo inicial del usuario

En el modelo individual de un usuario se mantiene toda la información y todas las suposiciones actuales del sistema sobre dicho utilizador. Cuando se ejecuta Aran se crea una instancia de *user-prototype;* mediante funciones que interactúan con Unix se rellena con los datos de identificación y se obtiene el conjunto de órdenes utilizadas en el sistema operativo (histórico) previamente al uso del sistema de ayuda (si no ha habido órdenes previas en esa sesión se utiliza la lista de la sesión anterior). Aran a partir de esa información inicial y de la definición de los estereotipos, determina todos y cada uno de los estereotipos que son asignables a ese utilizador concreto del sistema. De esta forma se obtienen unas suposiciones iniciales para ese usuario que se complementarán y actualizarán mediante su interacción posterior con Aran. El modelo de usuario de Aran no almacena explícitamente información entre diferentes sesiones, por los mismos motivos ya discutidos en Argos (sección 5.5.2), pero si utiliza el histórico de la interacción con el usuario que proporciona el sistema operativo.

En el proceso de adquisición del modelo de un usuario concreto, el primer paso es la determinación de los estereotipos definidos en el sistema que son aplicables a este usuario. Aquí existen dos situaciones diferentes:

a) *Especificación completa del estereotipo*. Cuando es posible realizar una especificación completa de un estereotipo en función de sus características, es decir mediante la formulación de las condiciones necesarias y suficientes para su aplicación. En este caso, a partir de los datos iniciales de identificación y

de la interacción anterior, mediante el proceso de clasificación se obtienen directamente los estereotipos aplicables.
Como ejemplo de especificación completa, en Unix si el número de usuario es cero, entonces ese usuario tiene todos los permisos sobre el sistema (es un superusuario). Esta peculiaridad se utiliza para que un usuario con dicha característica se clasifique automáticamente como una instancia del estereotipo de administrador de sistema, cuya definición es:

```
(defconcept System-Manager
    :is
    (:and User-Prototype
        (:filled-by has-id 0))
    :implies
       (:and Network-Expert Programmer-User))
```

Nótese que también se pueden establecer interdependencias y restricciones entre los estereotipos, como ya se ha mencionado previamente. En este caso, una vez que se ha clasificado un usuario como administrado de sistema se infiere automáticamente que tiene un conocimiento avanzado de la red y que es programador (mediante *:implies (:and Network-Expert Programmer-User)*). Este tipo de restricción ya se utilizaba en Argos, pero aquí mediante una adecuada definición y haciendo uso de la clasificación, tanto la detección de un usuario como administrador de sistema como las inferencias asociadas son automáticas.

b) *Especificación incompleta del estereotipo*. Cuando no es posible realizar una especificación completa. En este caso, a partir de los datos iniciales de identificación, de la interacción anterior y de la definición de los estereotipos, se aplican reglas heurísticas que determinan qué estereotipos son asignables. La aplicabilidad de un estereotipo se determina en función del contenido de las relaciones *key-commands*, *presupposed-unknown-commands* y *presuppossed-known-commands*. Concretamente en Aran se dispone actualmente de las dos siguientes reglas para determinar la aplicabilidad inicial de un estereotipo:

   1) si en la interacción con el S.O. Unix, previamente a la ejecución de Aran, el usuario ha utilizado alguna de las órdenes contempladas en la relación *key-commands*, entonces se aplica este estereotipo.

   2) si en la interacción con el S.O. Unix, previamente a la ejecución de Aran, el usuario ha utilizado por lo menos un determinado tanto por ciento de las órdenes contempladas en *presupposed-known-commands* (actualmente basta que haya usado más de una de las órdenes contempladas) y ninguna de las contenidas en *presupposed-unknown-commands*, entonces se aplica este estereotipo.

Como la especificación completa de los estereotipos es compleja en la mayor parte de los casos, se acude al uso de reglas ya que permite una mayor flexibilidad para la determinación de los estereotipos aplicables. Para realizar estas inferencias iniciales se parte de dos hipótesis: i) si un usuario utiliza

una orden se puede suponer que conoce esa orden; y ii) el interés y experiencia de un usuario se puede determinar a partir del estudio de la interacción del usuario con el sistema operativo, antes de la activación del sistema de ayuda. Es plausible que el problema actual del usuario, que le ha hecho acudir al asistente, esté relacionado con las órdenes utilizadas recientemente. Aran dispone de las dos heurísticas descritas previamente, programadas como reglas de activación automática cuando se detecta una nueva instancia de *User-Prototype*. Dichas reglas en función de la información inicial de la instancia y de los contenidos de las definiciones de los estereotipos determinan cuáles de éstos son aplicables.
Estas reglas permiten contemplar dos tipos de situaciones diferentes. Como ejemplo del primer caso, la detección de que el usuario utiliza el enlazador del sistema (*linker*) permite su clasificación como programador. El uso de alguna de las utilidades relacionadas con el lenguaje C (compilador, preprocesador, formateador de programas) permite su clasificación directa como programador en lenguaje C. Como ejemplo de la segunda situación, tomando la definición de *Network-Novice* presentada en el apartado anterior, un usuario que usa la red mediante los programas "mosaic" y "netscape", y que no utiliza ninguna de las otras órdenes de Unix contempladas en *presupposed-unknow-commads*, se clasificaría como un usuario novato de la red.

6.7.4 Actualización y mantenimiento del modelo

Además de los datos del modelo provenientes de la fase de clasificación inicial y de inferencias iniciales, este modelo se completa a partir de la interacción del usuario con Aran. El contenido del modelo de usuario supone una información por omisión, que es útil en ausencia de datos más concretos, pero que posteriormente puede ser actualizada cuando se dispone de nuevos datos sobre el usuario. El proceso de completar el modelo puede dar lugar a la existencia de información incoherente en el MU que el sistema debe detectar y corregir.

En las relaciones *accessed-documents* y *accessed-concepts* respectivamente, se van incluyendo los documentos y los conceptos a los que accede el usuario. Para los documentos se tiene en cuenta que puede acceder desde cualquiera de los tres modos del interfaz (descriptores, lenguaje natural o inspección del modelo del dominio). A los conceptos se accede siempre desde la interacción con el modelo del dominio. Esta información es incremental y se guarda a lo largo de la sesión con Aran. Los términos que se utilizan en las búsquedas mediante el interfaz en lenguaje natural y los que provienen de la realimentación (según el mismo proceso descrito en Argos) se añaden como valores de la relación *term-list*. No sólo se añade información sino que también se puede eliminar. Cuando el usuario indica expresamente que cambia el objetivo de su búsqueda (mediante el botón *reset*), entonces se reinicializa el contenido de *term-list*.

La adquisición de suposiciones sobre los usuarios es un proceso más complejo. Se realiza mediante la aplicación de reglas que describen las suposiciones que se pueden realizar, a partir de interacciones específicas con el interfaz de Aran.

Actualmente Aran dispone de dos reglas, que se presentan a continuación y que ejemplifican su potencialidad. En Aran se supone que si se solicita que se presenten los conceptos filtrados por el sistema porque los ha supuesto conocidos (p.e. mediante *Do not UM Filter* fig. 6.11), y el usuario accede expresamente a alguno de ellos, entonces se concluye que antes no lo conocía. También, si un usuario solicita expresamente acceder a los ejemplos de uso de una orden, es porque no estaba previamente familiarizado con ella. Las suposiciones adquiridas mediante interacción se almacenan en las relaciones *pressuposed-(un)known-concepts* y *pressuposed-(un)known-commands* complementando las suposiciones obtenidas mediante herencia de los estereotipos asignados. El sistema Aran contempla la posibilidad de incluir nuevas reglas de adquisición de suposiciones, sin tener que realizar cambios en el modelo del usuario.

6.7.4.1 Activación y desactivación de estereotipos

La inclusión de nuevos datos en el modelo no supone sólo un cambio en el valor de algunas de las relaciones, sino que puede provocar la existencia de información contradictoria o que no se aproveche toda la potencialidad de los estereotipos definidos en el sistema. Una de estas situaciones se produce cuando debido a la interacción se averigua que un usuario no conocía previamente una orden y esto entra en conflicto con los datos heredados por la asignación previa de un estereotipo concreto a ese usuario. En este caso hay que solucionar la discrepancia, realizando una reclasificación del usuario en función de los datos actuales, o bien eliminando los valores que producen la incoherencia. Esto supone que, además de las heurísticas de aplicación inicial de estereotipos, el sistema debe poseer unas reglas que permitan la aplicación de nuevos estereotipos según la nueva información que se adquiere. Dichas reglas deben permitir también la desasignación de algunos de los estereotipos previamente aplicados, si se comprueba que ya no son apropiados.

Aran implementa este tipo de heurísticas como reglas de ejecución automática, que se disparan cuando se modifica el valor de determinadas relaciones. En concreto actualmente se dispone de dos grupos de reglas simples de mantenimiento del modelo:

a) *Desasignación de estereotipos*. Si existe incoherencia entre los valores de las relaciones obtenidos por interacción con el usuario y los provenientes (mediante herencia) de la asignación de un estereotipo concreto, se determina cuál es el estereotipo que provoca la contradicción y se desasigna. En estas reglas se tienen en cuenta todas las relaciones que almacenan suposiciones sobre el conocimiento del usuario (es decir las relaciones *key-commands, presupposed-known-concepts, presupposed-unknown-concepts, presupposed-known-commands* y *presupposed-unknown-commands*).

b) *Eliminación de valores incoherentes*. Si existe incoherencia entre valores de relaciones obtenidos mediante interacción y que representan situaciones opuestas, hay que eliminar alguno de los valores. Estas reglas consideran diversas situaciones como, por ejemplo, que un concepto aparezca simultáneamente como conocido y desconocido (es decir como valor de las relaciones *presupposed-known-concepts* y *presupposed-unknown-concepts*). En

este caso se elimina de la lista de conceptos conocidos, ya que es menos grave repetir una información conocida, que filtrarla de una forma inapropiada porque se supone conocida. A las órdenes se les aplica un proceso similar considerando las relaciones *presupposed-known-commands* y *presupposed-unknown-commands*.

6.7.5 Utilización del modelo de usuario para implementar estrategias de presentación de la información

Este modelo supone la centralización de forma diferencia de toda la información que tiene el sistema sobre el usuario y su representación de una forma suficientemente expresiva, de modo que se pueda utilizar con distintos propósitos. El objetivo principal de este modelo, al igual que se hace en Argos, es la adaptación de la información al nivel de experiencia y al interés de cada usuario. A diferencia de en otros sistemas en los cuales este modelo se usa para modificar la generación de la información [Jonhson 94, Chin 89] o su contenido [Boyle 94], en Aran se utiliza para modificar las búsquedas y establecer estrategias en la presentación de esa información.

Esta adaptación se traduce en:

a) *Modificación de la búsqueda de información RI*. En el modo de interacción mediante lenguaje natural, al igual que se hace en Argos, el proceso de búsqueda de documentos relevantes mediante RI, cuando el usuario pide ayuda sobre una tarea, no considera sólo la última solicitud del usuario, sino que ésta se complementa con el contenido de la relación *term-list* del MU. El sistema colabora automáticamente con el usuario, teniendo en cuenta las preguntas anteriores y la información sobre relevancia producida por esas interacciones. Como se ha reseñado en Argos de esta forma se mejora el proceso de recuperación debido a que: por un lado se dispone de más texto en la consulta; por otro lado, con la información de relevancia y el texto que se puede seleccionar mediante el ratón, en la solicitud se utiliza un vocabulario más adecuado al dominio. En los apartados de modelado de usuario y de recuperación de información de Argos, se pueden obtener más detalles sobre estos procesos.

b) *Modificación del orden de presentación de los documentos obtenidos mediante RI*. El proceso básico es presentar los documentos según su orden de adecuación proporcionado por el sistema de RI, de forma que aparezca directamente el más relevante. Se realiza una reordenación de los documentos similar a la realizada en Argos, pero ahora la adecuación de los documentos se determina según el contenido del modelo de usuario. Concretamente se presentan primero los documentos recuperados que corresponden a alguna orden contenida en la relación *presupposed-unknown-commands*. Además, para evitar mostrar información repetida los documentos ya accedidos en esa sesión (contenidos en *accessed-documents*) se colocaran al final.

c) *Ordenación de los documentos obtenidos mediante especificación de descriptores*. Cuando el usuario selecciona descriptores se obtienen una serie de documentos candidatos, pero sin ningún criterio de adecuación al usuario. Entonces se realiza una ordenación con los siguientes criterios: los documentos localizados que se suponen desconocidos (es decir que aparezcan como valores de la relación *presupposed-unknown-commands*) se presentan primero; los que se suponen conocidos se presentan después (es decir los contenidos en *history-list* y en *presupposed-known-commands*); por último se colocan aquellos a los que se ha accedido en esa misma sesión (contenidos en *accessed-documents*).

d) *Modificación de la visualización de las órdenes (instancias) del dominio*. En la presentación gráfica de las instancias se realiza un filtrado de la información en función del contenido del modelo del usuario. En la representación de las órdenes del Unix se han tenido en cuenta aspectos pragmáticos y tutoriales (se describe con detalle en el epígrafe 6.8.3.3) . Entre otros, se han codificado cuáles son los conceptos prerrequisitos (mediante la relación *prerrequisite-concept*) que deberían ser conocidos para comprender la orden y cuáles son otros conceptos y órdenes relacionadas (relación *has-related-concept* y *has-related-command*). Esto posibilita mostrar junto con la instancia otros conceptos que ayudan a la asimilación de esa orden, aunque no la indexen directamente (no sean superconceptos suyos). También se han incluido ejemplos de su uso (relación *has-example*). En la visualización de esta información tutorial se filtran aquellos conceptos de los relacionados o que son prerrequisito que ya se suponen conocidos. Es decir que están contenidos en las relaciones *presupposed-known-concepts* y *accessed-concepts*. Si se dispone de ejemplos de uso de una orden, estos no se mostraran directamente si el usuario esta considerado como un usuario experto en algún tema (actualmente sólo si está clasificado como superusuario o como experto en la red). No se ha considerado oportuno filtrar las órdenes relacionadas ya que contribuyen a un mejor conocimiento del repertorio de órdenes del Unix y contienen órdenes no contempladas en la sección *see also* de la página del manual.
El filtrado se realiza exclusivamente en la visualización. Además, se aprovecha que el usuario tiene la iniciativa en la interacción, para proporcionarle la opción en el interfaz de recuperar la información filtrada o incluso para que pueda desactivar el MU. En la figura 6.11 se muestra el menú asociado a la instancia que permite recuperar los ejemplos y los conceptos que hayan sido filtrados.

La utilización del MU para realizar un filtrado o una modificación en el proceso de presentación de información, entra en conflicto con una posible ocultación no deseable. Como solución a este problema el usuario puede desactivarlo (botón *Do not UM Filter*) de modo que no se realicen las modificaciones anteriormente expuestas (excepto en la interacción mediante descriptores donde se realiza exclusivamente una reordenación de los documentos recuperados).

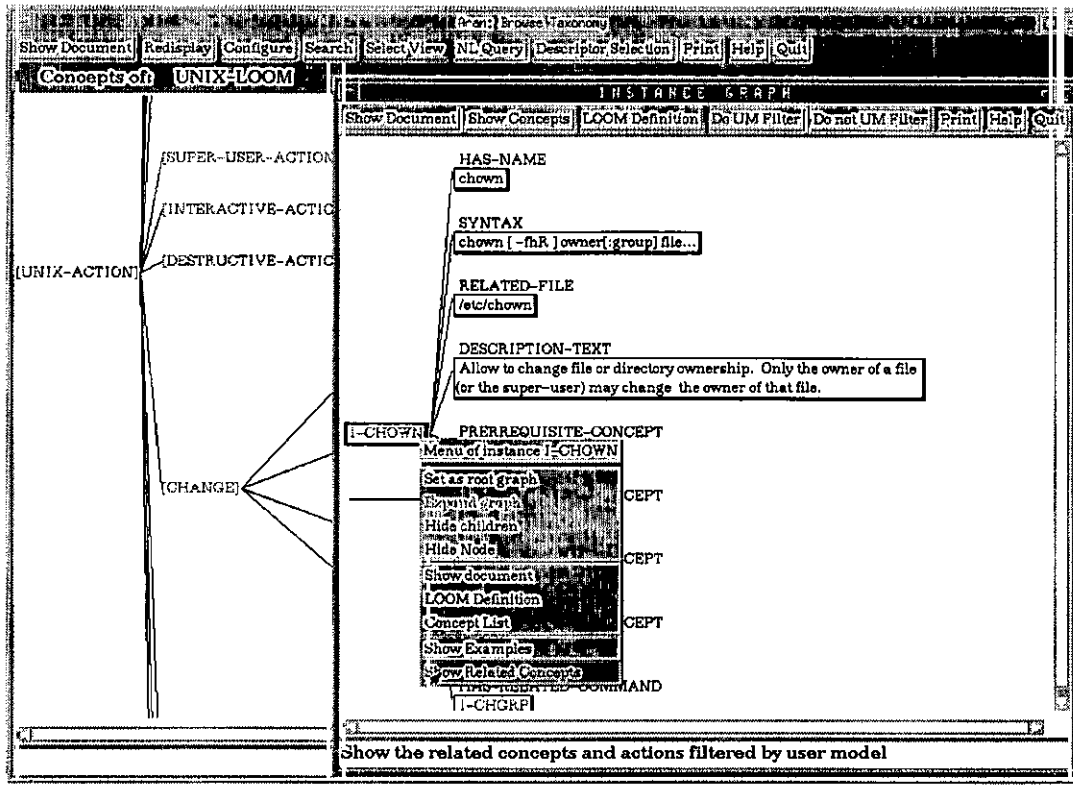


Figura 6.11: El modelo del usuario se utiliza para modificar la visualización de la información sobre el dominio. No obstante el usuario siempre tiene la posibilidad de acceder a la información filtrada por el MU e incluso a desactivar este filtrado.

6.7.6 Trabajo relacionado

En el campo del modelado de usuario, temas como representación formal y razonamiento no han tenido un papel muy importante. Con la excepción de lenguajes de la familia KL-ONE, las representaciones formales no han sido ampliamente utilizadas. Ahora la situación está cambiando debido a las mayores demandas, tanto inferenciales como de representación, que se hacen sobre los modelos de usuario. En concreto están apareciendo entornos (shells) de modelado de usuario que tratan de simplificar estos procesos de adquisición, mantenimiento y uso de un modelo de usuario en aplicaciones informáticas.

Existen modelados de usuario que limitan el número o el tipo de estereotipos asignables, como en el sistema GUMS [Kass 91] en el que se puede asignar a un usuario a un único estereotipo, o el sistema BGP-MS [Kobsa 95] que limita esta asignación únicamente a los estereotipos que aparecen como hojas terminales de la representación. Por el contrario en Aran no se establece otra limitación en la asignación de estereotipos, que las detalladas expresamente en su especificación y la de la coherencia de la información contenida en los mismos. Por ejemplo, no es

permisible que un usuario se clasifique simultáneamente como usuario novato y como usuario experto en el uso de la red, pero esto es debido únicamente a que contienen información contradictoria. El hecho de que se puedan asignar estereotipos no terminales del grafo permite, por ejemplo, clasificar a un usuario como programador, a la vez que se dispone de especializaciones de esta clase de usuarios como es la de programadores en lenguaje C. Esta circunstancia permite suponer que un usuario programador estará interesado en las llamadas al sistema Unix, pero no en las funciones específicas en lenguaje C, mientras que un programador en C podría estar interesado en ambas.

En la bibliografía sobre modelado de usuario [Kobsa 93, Kobsa 95, Whalster 89] no existe un conjunto de reglas genéricas para la adquisición de suposiciones, sino que normalmente se utilizan heurísticas que dependen del dominio y que se codifican en cada aplicación particular. Tampoco se dispone de métodos independientes del dominio para solucionar el proceso de actualización del modelo. No obstante, en este último caso, si existen esquemas genéricos que se pueden particularizar para cada aplicación. Por ejemplo Kobsa [Kobsa 95] cita las siguientes reglas generales para asignar o desasignar estereotipos: a) si se conocen todos los elementos contenidos en una lista; b) si se desconocen todos los elementos contenidos en una lista; c) si se satisface por lo menos un determinado tanto por ciento del estereotipo para ese usuario; d) si se conocen por lo menos un tanto por ciento de los elementos de una lista; y e) si se desconocen por lo menos un tanto por ciento de los elementos de una lista.

6.7.7 Consideraciones finales sobre el modelado

Esta aproximación mediante estereotipos permite mejorar la clasificación de los usuarios. El enfoque utilizado en Argos, aunque se ha mostrado como muy útil y supone un refinamiento de un modelo ampliamente utilizado, es una clasificación excesivamente simple de los usuarios. Por ejemplo, en el estudio empírico realizado por Draper [Draper 84] y confirmado por otros experimentos posteriores [Sutcliffe 87, Kobsa 93] obtuvo como resultado que en un entorno como el Unix, es mas adecuado hablar de especialización en ciertas áreas que de expertos generales. La familiaridad de los usuarios con las órdenes del sistema operativo requiere la identificación de agrupaciones de conocimiento y no únicamente un nivel de experiencia genérico. Esto permite, por ejemplo, que se pueda diferenciar la experiencia que se tiene en el uso de la red, o que se pueda considerar si un usuario está interesado en el uso de los utilidades de procesamiento de texto que proporciona Unix. Además permite establecer interdependencias entre ellas como, por ejemplo, que un usuario clasificado como administrador de sistema se considera también como experto en el uso de la red.

Cabe destacar que el modelo utilizado en Aran diferencia los distintos tipos de información que posee, por un lado, las suposiciones e inferencias sobre el conocimiento que posee el usuario, y por otro lado, los datos que corresponden a observaciones objetivas. Esta es una cualidad deseable en el modelado de usuario, ya que no es igual suponer que el usuario conoce un determinado concepto

relacionado con una orden, que saber a qué documentos ha accedido y por tanto se tiene la certeza de que se conoce su contenido.

El nuevo tipo de representación de la información posibilita el desarrollo y experimentación de nuevos modelos de usuarios mas completos. Por ejemplo, como trabajo futuro se podrían plantear nuevas características, como el modelo a largo plazo o la clasificación más detallada de los usuarios con la consideración de nuevos estereotipos. Esto también plantearía nuevos problemas en el mantenimiento de la coherencia del modelo y en la definición de reglas que permitan la reubicación a un usuario en otros estereotipos.

De igual manera que se ha tratado en Argos, Aran ha sido concebido como un marco de aplicación concreto orientado a la investigación sobre la aplicación e integración de diversas técnicas. Por tanto, las reglas y heurísticas empleadas en este modelo de usuario no pretenden ser completas. Su objetivo principal (como también se ha destacado para la clasificación de los estereotipos de usuario) es la ejemplificación de la potencia y de la validez del tipo de modelado propuesto.

6.8 Modelo del dominio

La parte más extensa de la base de conocimiento de Aran es la dedicada a modelar el dominio del Unix. La información contenida en ella es una conceptualización (un modelo conceptual) del sistema operativo y su objetivo primordial es la estructuración de la información disponible sobre el sistema. Esta porción de la base de conocimiento define la ontología del dominio, es decir cuáles son los elementos identificados y representados en ella, para después organizar el resto de la información en función de estos términos.

Para construir esta representación se ha decidido modelar el dominio utilizando una representación centrada en los objetos y en las acciones que se realizan sobre ellos. Esta aproximación es similar a la seguida en proyectos como el Sinix Consultant [Hecking 88, Kemke 87] y Lassie [Devanbu 91]. También se han considerado las alternativas seguidas por otros sistemas que consideran los planes de los usuarios o que consideran una descripción detallada de las tareas realizables [Breuker 90, Wilensky 89, Bhansali 91]. Estas otras aproximaciones se han descartado fundamentalmente por un factor de coste, ya que en un entorno tan extenso como el del sistema operativo Unix, su realización es muy compleja. De hecho sistemas como el Unix Consultant [Wilensky 89] y EUROHELP [Breuker 90], a pesar de ser proyectos muy amplios en este dominio, sólo llegaron a codificar información sobre partes concretas del sistema (el sistema de ficheros y el sistema de correo, respectivamente).

En nuestra base de conocimiento (además de la información del modelo del usuario) se describen los conceptos o entidades que se han considerado significativos para modelar el dominio (fig. 6.12). En función de esta descripción, los conceptos se agrupan y se organizan, mediante relaciones de generalización-especialización, produciendo una jerarquía. Este modelo del dominio está estructurado en dos partes diferenciadas:

- información sobre los objetos del dominio
- información sobre las acciones.

Se representan los tipos de objetos que se distinguen en el sistema, tanto reales como virtuales, por ejemplo, dispositivos físicos, programas, características del software, entrada estándar, etc., dando su descripción, características y sus interrelaciones con otros objetos. Por otro lado, también se representan las posibles acciones u operaciones que se pueden realizar sobre estos objetos, por ejemplo, manipulación de archivos, comunicaciones entre usuarios u operaciones interactivas.

Las dos partes que hemos diferenciado en el modelo del dominio están asociadas con dos puntos de vista diferentes del sistema. La que hemos denominado representación de los objetos del dominio supone una visión arquitectural del diseño; es decir cuáles son los elementos hardware y software considerados en nuestro modelo, cuáles son las relaciones estructurales entre ellos, y qué características diferenciadoras presentan. Trata de dar respuesta a preguntas como qué tipos de procesos se pueden distinguir, o cuantos tipos de ficheros diferentes reconoce el sistema y cómo se caracterizan. La representación de las acciones supone un punto de vista operativo, relativo a las acciones que se pueden llevar a cabo sobre los objetos y cuáles son las órdenes que las realizan. Esta visión del sistema clasifica las utilidades que proporciona el Unix en función de dos criterios que son la agrupación temática, y la similitud de las operaciones que realizan (agrupación semántica).

Esta clasificación es jerárquica y supone una descripción con diferentes niveles de abstracción de los conceptos, por lo que es deseable disponer de una representación con herencia. De este modo las características comunes se modelan en los atributos de los conceptos más generales, que son heredadas automáticamente por los conceptos que se encuentran por debajo de ellos. Estas consideraciones nos llevan a utilizar un formalismo de representación centrado en los objetos, basado en una lógica descriptiva y con herencia. Este formalismo es Loom [McGregor 91] que además al incluir un clasificador automático, simplifica el proceso de creación del modelo del dominio.

A continuación comenzaremos describiendo la metodología seguida para la construcción de la base de conocimiento que modela el dominio de Aran. En apartados posteriores se trata con más detalle la representación que se ha realizado de los objetos y las acciones del dominio, así como su utilización. Se analiza también un aspecto fundamental de Aran, como es la ampliabilidad y el mantenimiento del modelo del dominio. Finalmente se estudia como se han logrado los propósitos de ayuda y enseñanza, para los que este modelo es una pieza clave.

6.8.1 Metodología de construcción de la base de conocimiento

El proceso de ingeniería (del conocimiento) que supone la construcción de una base de conocimiento (BC), hasta que se obtiene una representación suficientemente completa del dominio, es complejo y costoso [Rich 94]. En la actualidad no existe una metodología estándar, ampliamente probada y admitida, para la construcción de bases de conocimiento [Boy 91, Brachman 91]. Incluso metodologías para la construcción de sistemas basados en conocimiento como KADS [Wielinga 92], no proporcionan una metodología completa y detallada para la construcción del modelo del dominio [Linster 92]. Un objetivo de este trabajo es la reutilización de esfuerzos previos en la creación de la BC. No obstante, el hecho de que todavía no existan BC completas disponibles, ni toda la información deseable sobre otras aplicaciones, implica que hay que dedicar un mayor esfuerzo a la construcción de nuestra base de conocimiento, en la que hemos ido integrando información procedente de otros sistemas.

En Aran se han seguido y adaptado a nuestro caso las directrices de una metodología propuesta en [Brachman 91]. Brachman presenta un método de ingeniería del conocimiento para el sistema CLASSIC, perteneciente a la familia KL-ONE. Además de la adaptación también se ha tenido que incluir el aspecto de la reutilización del conocimiento previamente codificado para otros sistemas. La clave en el proceso de desarrollo de una BC, usando una representación centrada en los objetos, es encontrar la forma adecuada de describir el dominio, mediante objetos y las relaciones que existen entre esos objetos. Esto implica la especificación de los elementos individuales sobre los que se podrá asertar u obtener información (individuos o instancias del dominio), así como la especificación de las clases de esos elementos que comparten propiedades comunes (los conceptos). Las propiedades de los individuos y las interrelaciones entre ellos se representan mediante relaciones. Este proceso se complica por el hecho de que, dependiendo del nivel de abstracción, hay elementos que se puede representar como individuos o como conceptos, así como que hay términos que se pueden representar de forma similar como relaciones o como conceptos. Esto son decisiones de diseño en función del uso que se haga de la base de conocimiento.

Esta BC en Aran tiene un doble propósito, por una parte fija los términos sobre los que se incluye y en función de los que se estructura la información; por otra parte, se utiliza para indexar las órdenes de Unix que se representan como instancias (concretamente como instancias de las acciones identificadas).

A continuación se presenta la metodología seguida, con una descripción de los pasos que consta, y en la que se proporcionan ejemplos de la base de conocimiento de Aran:

1. *Enumerar los tipos de objetos.* Se trata de enumerar todos los objetos sobre los que se va a introducir información o sobre los que interesa proporcionar explicaciones al usuario. Se identifican todos los elementos significativos del dominio y que por tanto el sistema debe tener en cuenta. Como se ha tratado previamente, hay tres tipos distintos de objetos: los que representan clases o

tipos de elementos, que se denominan conceptos; los que representan características de los objetos (propiedades y partes), que se denominan relaciones; y finalmente los que representan elementos individuales, denominados instancias. En Aran se ha diferenciado entre dos tipos de conceptos, los que representan a los tipos de objetos de Unix y las clases de acciones que se pueden realizar sobre estos objetos (en la misma línea de otros sistemas como [Hecking 88, Devanbu 91]). Las órdenes y utilidades de Unix se representan como instancias.

2. *Distinguir entre conceptos y relaciones (roles).* Tomando las clases de objetos obtenidos en el paso anterior, los que tienen existencia independiente se consideran como conceptos, mientras que aquellos cuya existencia dependa de otros se consideraran como relaciones. Por ejemplo, los procesos se consideran como conceptos ya que existen como objetos independientes, mientras que una característica de los procesos como es que tienen una determinada prioridad o que tienen un número de identificación se ha representado mediante relaciones.

3. *Desarrollar una taxonomía de conceptos y relaciones.* Agrupar los elementos identificados creando una taxonomía jerárquica, de modo que los conceptos más generales "agrupen" a otros conceptos más específicos. Los conceptos se clasifican jerárquicamente bajo una raíz terminológica *unix-thing*, a partir de la cual se organizan en dos ramas diferenciadas: por un lado las entidades físicas o características software (objetos), cuya raíz es *unix-entity*, y por otro las acciones cuya raíz es *unix-action* (fig. 6.12 y fig. 6.13). Cabe destacar la agrupación de las acciones tomando como criterios principales la similitud de operación y el área temática, de la misma forma que se hace en [Kemke 87]. Las relaciones también se pueden clasificar jerárquicamente, aunque no es habitual realizar una representación con muchos niveles de profundidad. Por ejemplo, para establecer conexiones entre los objetos y su implementación real, se definen las relaciones *related-file* y *related-directory* que a su vez se agrupan debajo de una relación más general denominada *related-code*. Esta taxonomía es un grafo acíclico dirigido, de modo que un concepto (o relación) determinado puede ser descendiente de más de un concepto (o relación).

4. *Identificar los individuos (instancias).* Identificar el conjunto clave de individuos que son importantes para todos los usos de la aplicación y determinar todos los conceptos que son aptos para intervenir en su descripción. Aunque en principio en Aran se representan como instancias únicamente las ordenes de Unix, existen otros elementos individuales que son necesarios para completar las definiciones de conceptos. Por ejemplo, los interpretes de órdenes estándar del Unix (*shell*) son importantes para la descripción de un usuario, ya que es necesario asociarle uno de ellos. Por tanto, se puede definir el concepto *standard-shell*, por extensión, enumerando todos los posibles individuos a los que comprende (normalmente *csh, sh, rsh y ksh*). Su codificación es:

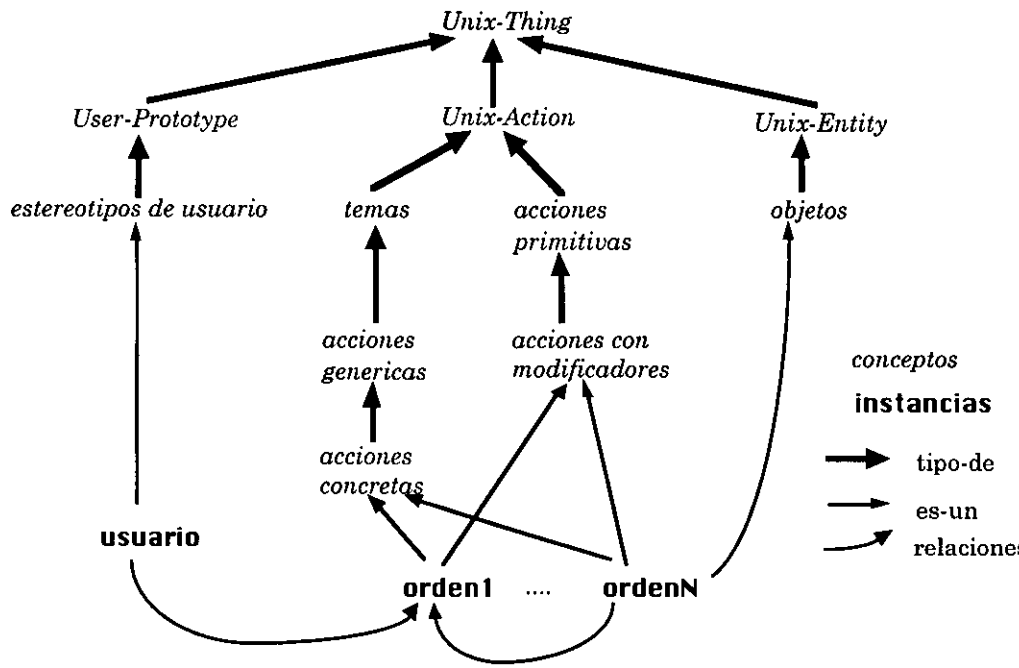


Figura 6.12: Esquema de la organización y estructura de la base de conocimiento de Aran. Se destacan sus dos partes principales: el modelo del usuario y el modelo del dominio.

```
(defconcept Standard-Shell
  "Interprete de comandos estandar de Unix. Es el proceso que se
ejecuta cuando un usuario se conecta. Las shell estandar son la c-
shell, tc-shell Korn-shell bourne-shell"
  :is
  (:and Shell
    (:one-of csh tcsh ksh sh)))
```

5. *Realizar la especificación y definición de los conceptos.* Hay que decidir la especificación que se realiza de los conceptos. Esto supone, entre otras cosas, determinar las propiedades y partes de estos objetos, las posibles dependencias y restricciones en sus propiedades o cuáles son sus interdependencias con los otros conceptos de la BC. A continuación se detallan estos puntos.

   5.1 Determinar propiedades y partes. Hay que determinar la descripción de la estructura interna de los objetos. Para cada concepto se consideran su lista de propiedades o características, tanto intrínsecas como extrínsecas, y sus diferentes partes. Así mismo hay que tener en cuenta las conexiones entre miembros individuales de ese concepto y otros elementos, que podrían no ser considerados exactamente propiedades o partes (p.e. un proceso tiene un propietario). A cada una de estas características se le debe asignar una relación.

5.2 Determinar las restricciones de tipo y número de los posibles valores. Para cada concepto y para cada relación que sea relevante para su descripción, se debe determinar el tipo de los objetos (conceptos) y la cardinalidad del conjunto de valores (instancias) que pueden actuar como relleno (*filler*). Por ejemplo, un fichero en Unix está determinado unívocamente por su número de *i-nodo*, de forma que se puede establecer que esta relación tendrá sólo un valor posible y este tiene que ser de tipo numérico.

5.3 Detallar las restricciones de valores no previamente representadas. Es necesario asegurarse que existen en la taxonomía los conceptos apropiados para expresar las restricciones de los posibles valores. Si no existe algún concepto necesario para describir este tipo de valores se debe incluir.

5.4 Determinar las dependencias o conexiones entre las relaciones. Para cada concepto hay que enumerar cualquier posible dependencia o interrelación entre sus relaciones que pueda ser importante en ese dominio. El valor de una relación de un concepto puede depender de otra relación definida en algún otro concepto.

5.5 Distinguir entre las propiedades esenciales y las no esenciales. Las propiedades esenciales constituirán la definición del concepto, es decir las condiciones necesarias y suficientes para determinar la clasificación de un individuo. Las otras propiedades que no son necesarias para esta clasificación, se expresarán como consecuentes de reglas asociadas al concepto. Por ejemplo, para representar que al editar el contenido de un objeto también se podría destruir su contenido, se establece que una vez que se afirme que una orden es de edición también sea clasificada automáticamente como una acción destructiva (mediante *:implies Destructive-Action*) . La codificación correspondiente es:

```
(defconcept Edit-Object
    "se edita el contenido de un objeto, lo que implica su
posible modificacion de modo destructivo"
  :is-primitive
  Change-Object
  :implies Destructive-Action)
```

6. Distinguir conceptos primitivos y conceptos definidos. Determinar si la definición propuesta de cada concepto es completa, es decir, si se conocen las condiciones suficientes y necesarias que debe cumplir un individuo para ser reconocido y clasificado automáticamente como perteneciente a un determinado concepto. Si no se tiene la definición completa entonces el concepto se considera primitivo. En nuestra base de conocimiento gran parte de los conceptos son primitivos, ya que es fundamentalmente una base terminológica, alrededor de la cual se realiza la indexación manual de la información.

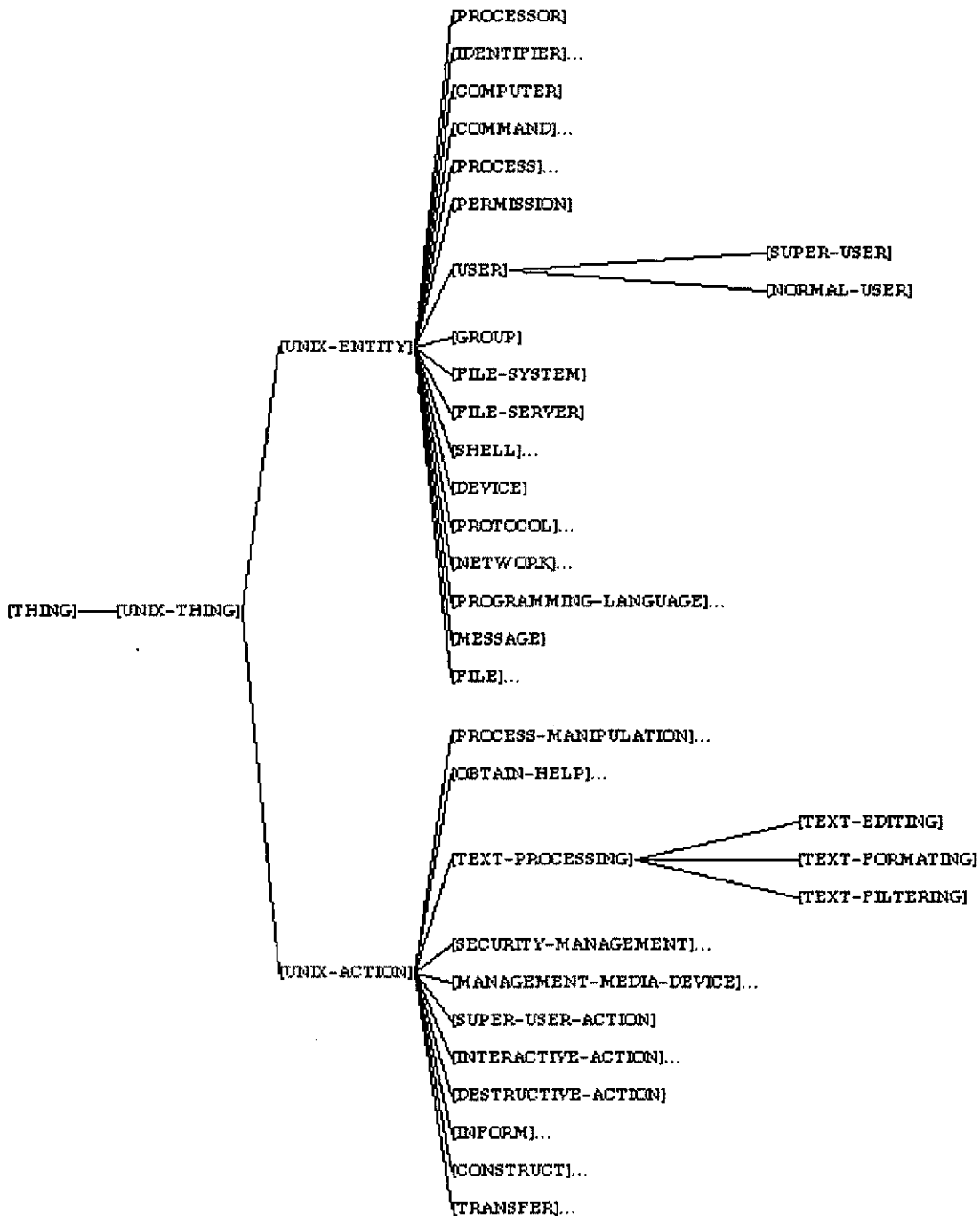


Figura 6.13: Muestra parcial de la jerarquía de conceptos de Aran que representan los objetos y las acciones del dominio

7. *Reutilizar esfuerzos previos.* En cada uno de los pasos anteriores se analizarán otras bases de conocimiento candidatas a ser reutilizadas, aprovechando aquellas partes que sean similares. Hay fases que exigen mucho esfuerzo, en las que se pueden considerar distintas alternativas, como por ejemplo la clasificación taxonómica, y que se ven simplificadas por la existencia de bases de conocimiento previas, basadas en estudios empíricos que les aportan un sólido fundamento. Por ejemplo, en la clasificación de las acciones de Aran se ha reutilizado y ampliado la descripción usada en [Kemke 87]. A pesar de la dificultades debidas a la insuficiente información disponible sobre otras aplicaciones, en la BC de Aran se ha integrado parte del conocimiento previamente utilizado en los sistemas descritos en [So 94, Kemke 87, Hecking 88, Devambu 91].

Resultado de aplicar esta metodología es que en la BC de Aran se han representado como conceptos los elementos, objetos y acciones, que conceptualizan el dominio, así como los estereotipos que representan los diferentes subgrupos de usuarios (fig. 6.9). Las ordenes de UNIX se representan como instancias (individuos) de la base de conocimiento y se indexan en función de las acciones identificadas. Para contener la información concreta de cada usuario, se creará dinámicamente una instancia de *User-Prototype* que se indexará en función de los prototipos asignables (según el proceso descrito en los epígrafes 6.7.3 y 6.7.4)

6.8.2 Representación de los objetos del dominio en la BC de Aran

La parte de los objetos de la base de conocimiento es una representación de las clases de objetos (conceptos) y elementos necesarios para la comprensión del sistema operativo. Su objetivo es obtener un modelo que capture los términos utilizados en el dominio de los sistemas operativos y cuál es su significado concreto en Unix. Por ello se representan tanto tipos de objetos con existencia física real, p.e., usuario, fichero, dispositivo, como otros elementos conceptuales, p.e., entrada estándar, o variable de sistema.

La jerarquía de objetos de Aran fija el vocabulario a utilizar, cuál es su significado concreto en este contexto y cuáles son las interrelaciones y dependencias entre los objetos. Trata por tanto de capturar y representar las consideraciones que se han tenido en cuenta en el diseño del sistema. De esta manera se puede incluir informaciones variadas como, por ejemplo, que un fichero en el sistema operativo Unix es simplemente una secuencia de bytes que almacena datos, cuál es la información completa que almacena el sistema sobre un fichero (se obtiene mediante la llamada de sistema *stat*) o cuáles son los siete posibles tipos de ficheros reconocidos por el sistema.

Todos los objetos están agrupados jerárquicamente debajo de una raíz *unix-entity* (fig. 6.14). No se ha utilizado un esquema fijo de representación, sino que siguiendo la metodología propuesta, para cada uno de ellos se proporciona su posición en la jerarquía y su conjunto de atributos característicos (relaciones). Es de destacar que también se han considerado las relaciones con el código del sistema operativo

siempre que ha sido posible. La definición de un objeto (concepto) puede ser completa, es decir mediante las condiciones necesarias y suficientes de pertenencia a ese concepto. No obstante, en la mayor parte de los casos se realiza una definición parcial o primitiva, que es suficiente para los objetivos terminológicos de este conocimiento.

A continuación como muestra de la información contenida sobre los objetos presentamos la codificación del concepto usuario del sistema, de sus subconceptos usuario normal y superusuario, y de algunas de las relaciones implicadas en estas definiciones. En este ejemplo se observa la capacidad expresiva de este formalismo y cómo se relaciona el contenido de la base de conocimiento con el código del sistema operativo.

```
(defconcept User
    "usuario del sistema unix. Caracteristicas: numero, nombre,
grupo (o grupos), shell y directorio de usuario.Los ficheros
relacionados son /etc/group y /etc/passwd. Por omision se le
asigna la shell csh"
  :is
  (:and Unix-Entity
        (:the has-ID Number)
        (:the has-name User-Name)
        (:all has-group Number)
        (:at-least 1 has-group))
  :implies
      (:and
            (:the home-directory Absolute-Path)
            (:the has-shell Standard-Shell)
            (:filled-by related-file "/etc/group" "/etc/passwd"))
  :defaults
       (:filled-by has-shell csh))
```

Como puede verse en este ejemplo, el concepto de usuario del sistema (*user*) se define como subconcepto de *unix-entity*, con las relaciones *has-id, has-name, has-group, home-directory, has-shell* y *related-file* y se proporciona un comentario en lenguaje natural que describe brevemente el tipo de objeto representado. Se hace una definición completa, no primitiva. Las propiedades esenciales del usuario son las tres primeras y aparecen en la cláusula *is*; son las que se utilizan para reconocer que una instancia pertenece a esta clase. Establecen que el usuario tiene un único identificador numérico, que tiene un único nombre, que tiene que ser del tipo *user-name*, y que pertenece por lo menos a un grupo, expresado como un número (puede pertenecer a más de uno). Las otras tres propiedades que aparecen en la cláusula *implies*, son no esenciales y se tienen en cuenta una vez que un individuo ha sido clasificado como usuario. La cláusula *defaults* permite asociar valores por omisión a alguna de las características que no hayan sido expresamente establecida; en este caso se asocia a *has-shell* el intérprete de órdenes C-shell (*csh*) indicando que es el estándar en nuestro sistema. La relación *related-file* establece una correspondencia con el código ya que indica que los ficheros del sistema relacionados con los usuarios son */etc/passwd* y */etc/group*.

```
(defconcept Super-User
    "el superusuario  es un usuario privilegiado que tiene todos
los permisos en el sistema y su numero de identificacion es 0.
Normalmente su identificador en el sistema es root, aunque puede
haber otros superusuarios con distinto nombre"
  :is
  (:and User
       (:filled-by has-ID 0)))

(defconcept Normal-User
  "un usuario normal tiene numero de identificacion distinto de 0
y positivo"
  :is
  (:and User
(the has-ID (through 1 +INFINITY))))
```

Como subconceptos de *user* se definen los dos tipos de usuarios que existen en Unix: los superusuarios y los usuarios normales (*super-user* y *normal-user*). Si el identificador de un usuario es cero se clasifica como superusuario y si es mayor que cero como usuario normal.

```
 (defrelation has-ID
    "relacion entre una entidad y un numero que la identifica
univocamente"
  :range Number
  :characteristics (:single-valued))

(defrelation has-name
    "correspondencia entre un objeto y un nombre unico"
  :characteristics (:single-valued))

(defrelation has-group
    "relacion que establece pertenecia a grupo"
  :characteristics (:closed-world))
```

En el ejemplo previo se presenta la definición de algunas de las relaciones que intervienen en la descripción de estos conceptos, como *has-id, has-name* y *has-id.* Es fundamental establecer las propiedades de las relaciones, ya que influye en el tipo de razonamiento que se puede realizar sobre ellas (por omisión se utiliza sintaxis de mundo abierto). Así las relaciones *has-name* y *has-id* se declaran como unievaluadas, y en la última se establece que el valor debe ser de tipo numérico. La relación *has-group* se declara con sintaxis de mundo cerrado, de modo que se razona con el contenido que tenga en cada momento.

Las conexiones con el sistema no están restringidas a especificaciones únicamente descriptivas, como cuál es el posible código relacionado, sino que se pueden asociar predicados que calculen el valor de relleno de una relación o que determinen si una cierta instancia cumple alguna condición impuesta por el sistema. Por ejemplo, para que un nombre de usuario sea válido en un sistema Unix concreto, tiene que ser una cadena de caracteres que aparezca en el fichero "/etc/passwd". En la siguiente definición *exist-user* es un predicado Lisp que verifica que la cadena proporcionada como nombre de usuario aparece en dicho fichero.

```
(defconcept User-Name
    "identificador (nombre) de usuario existente en el sistema.
Tiene que estar en el fichero /etc/passwd"
  :is-primitive Identifier
  :predicate ((?name)
         (exist-user ?name)))
```

Como se ha mencionado previamente, estas descripciones tienen en cuenta las consideraciones de diseño del sistema operativo de modo que, además de fijar el vocabulario, establecen las restricciones y relaciones entre los objetos. De hecho, como muestra el siguiente ejemplo, si las definiciones de los objetos (conceptos) son completas y se proporciona la especificación completa de un objeto individual (instancia) el sistema es capaz de clasificarlo automáticamente. Si se afirma que *user01* es un objeto de Unix, que tiene un identificador numérico, que tiene un nombre y que pertenece a un grupo, el sistema no sólo lo reconoce como un usuario, sino que determina que es un usuario normal y no un superusuario, ya que su identificador es positivo y distinto de cero.

```
USER(18):  (tellm (about user01 unix-entity
           (has-id 104)
           (has-name balta)
           (has-group 69)))
*
Recognition changes at agent time 3:
   entry: USER01 |C|NORMAL-USER
   entry: USER01 |C|USER
   entry: USER01 |C|UNIX-ENTITY
   entry: USER01 |C|UNIX-THING
4
```

Mediante la llamada (*pi user01*) se muestra la representación interna de esta nueva instancia. Se observa que además de los datos proporcionados, existen otros que provienen de su clasificación como usuario normal, como son las relaciones *has-shell* y *related-file* que se han rellenado automáticamente.

```
USER(19): (pi user01)

(TELL (:ABOUT USER01 NORMAL-USER (HAS-SHELL CSH) (RELATED-FILE
"/etc/passwd") (RELATED-FILE "/etc/group")(HAS-GROUP 69) (HAS-NAME
BALTA) (HAS-ID 104)))
```

Además de la riqueza expresiva que permite el sistema de representación, hay que destacar los mecanismos que proporciona para la detección de definiciones incoherentes o el uso de conceptos no definidos, que ayudan en la creación de la base de conocimiento. Este aspecto se analizará con detalle en el apartado 6.8.4.

6.8.3 Representación de las acciones del dominio

La parte de las acciones de la base de conocimiento es una representación de las operaciones que se han identificado sobre los objetos del dominio. Su propósito

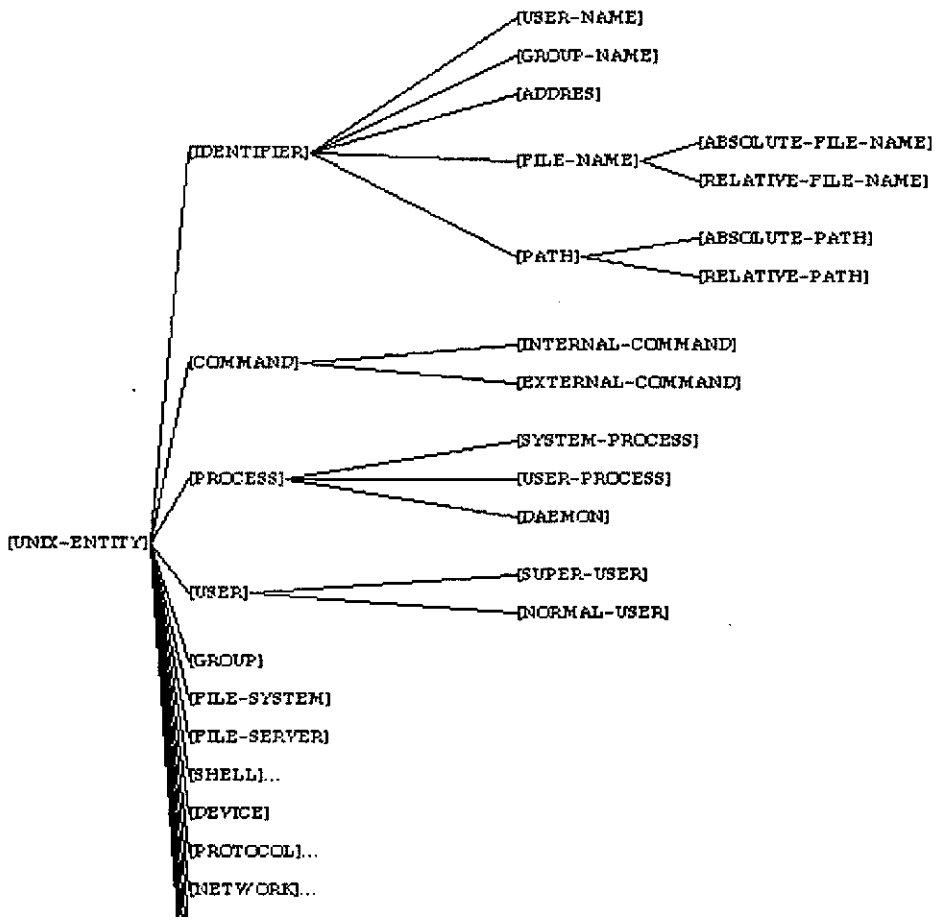


Figura 6.14: Representación parcial de los objetos del dominio

fundamental es realizar una indexación (basada en conocimiento) de las utilidades del sistema operativo.

Para la representación de las acciones se ha utilizado una estructura de marco *(frame),* similar a la utilizada por otros sistemas de tratamiento de lenguaje natural basados en gramáticas de casos [Buenaga 95, Girardi 93, Girardi 94]. De este modo se puede hacer una descripción en función de los casos gramaticales, imponiendo restricciones sobre ellos, por ejemplo, sobre qué tipo de objetos se realiza la acción, quién es el que puede ejecutar una determina orden, o cuál es el destino de una operación. Para la obtención de las relaciones a considerar, se han analizado las representaciones utilizadas en distintos sistemas, en este y otros dominios [Hecking 88, Devanbu 91, Buenaga 96].

La jerarquía de acciones de Aran, está formada principalmente por conceptos primitivos, que se utilizan como una base terminológica a partir de la cual se va a indexar la información relativa a las utilidades concretas (del S.O. Unix) representadas como instancias. El objetivo principal no es realizar una descripción

completa de cada orden, de modo que el sistema sea capaz de clasificarla de modo automático, sino que se va a realizar una indexación manual, mediante la descripción de cuáles son los conceptos de los que depende esa orden. No obstante, siempre que es posible se introducen restricciones en la definición de estas acciones genéricas, de modo que los propios mecanismos de integridad de la base de conocimiento ayuden a detectar información incoherente.

Esta jerarquía de acciones, cuya raíz es una acción genérica *unix-action* que las agrupa a todas, supone una descripción de las operaciones mediante diferentes niveles de abstracción. Las acciones de mayor nivel (más próximas a la raíz) representan operaciones muy generales, que se corresponden con términos del lenguaje natural poco específicos (p.e. *change, create, text-processing*). A medida que se desciende de nivel se van haciendo más concretas, de modo que en los niveles inferiores se tendrán acciones que describen sólo una utilidad, o un grupo reducido de utilidades del sistema (p.e. *send-standard-mail*).

Este planteamiento permite simultáneamente la agrupación de las ordenes de Unix teniendo en cuenta distintos criterios, como similitud en su función, que operen sobre el mismo tipo de objetos, o que se encuentren comprendidas dentro del mismo conjunto de operaciones (p.e., utilidades de correo, ordenes de administración del sistema). Se han considerado dos clasificaciones principales: una basada en áreas temáticas, similar a la realizada en los libros de texto o manuales, y otra basada en la similitud del efecto de las operaciones, de modo que se clasifican en función de un pequeño número de acciones primitivas o básicas. Una aproximación similar se utiliza en el Sinix Consultant [Kemke 87]. A continuación se tratan con más detalle estas dos clasificaciones realizadas.

6.8.3.1 Clasificación de las acciones por área temática

El objetivo de la clasificación por área temática es describir las órdenes a un nivel global y agruparlas de una forma significativa para el usuario, de la misma manera que se realiza la agrupación por temas en los libros de texto. En esta representación de las acciones se consideran tres niveles principales: temas, órdenes abstractas y órdenes concretas.

Primero, como subconceptos directos de la raíz *unix-action* (fig. 6.15), se hace una clasificación de acciones genéricas, llamadas temas, debido a que se toman principalmente de la estructuración en temas del manual y otros libros de referencia [Sun 92, Deitel 90, Leffler 89]. Estas descripciones de acciones son muy generales, p.e. comunicación con usuario o administración del sistema, y su propósito es que un usuario que no esté familiarizado con el sistema, pero que por lo menos tenga una ligera idea de sus principales características y capacidades, sea capaz de focalizar y restringir su búsqueda a una parte específica de la base de conocimiento.

En un siguiente nivel se consideran las órdenes abstractas, que tratan de representar términos "canónicos" del lenguaje (fundamentalmente verbos y ocasionalmente con algunos modificadores). Ejemplos de estas órdenes abstractas serían leer correo, enviar correo o mantener diálogo. Para clasificar una acción en

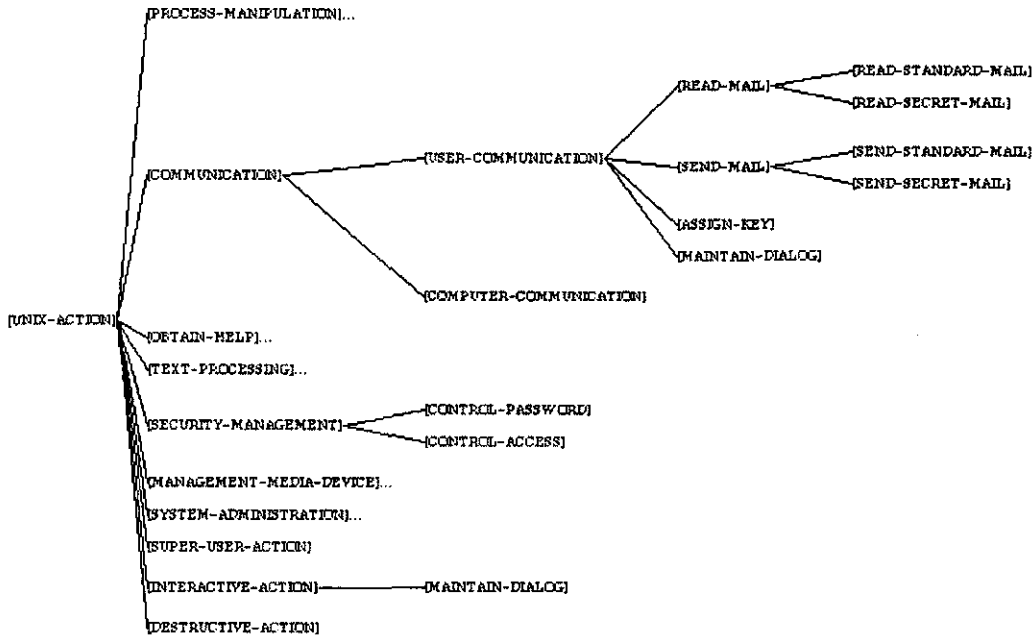


Figura 6.15: Representación parcial de la clasificación de las acciones por área temática. Nótese que *maintain-dialog* está repetido debido a que la jerarquía se muestra como un árbol a pesar de ser un grafo.

este nivel se consideran los términos que se utilizan en lenguaje natural para describirla y también el modelo mental del utilizador. Por tanto, se trata de agrupar operaciones y relacionarlas con otros modelos que el usuario ya conoce, por ejemplo, con las operaciones de correo ordinario o con la comunicación telefónica.

A partir de este nivel pueden existir una o varias órdenes del sistema que realicen la acción representada, o pueden existir varias que lleven a cabo funciones mas específicas. Si hay varias órdenes con propósitos diferentes o particularidades, habrá que introducir nuevos niveles de acciones cada vez más específicas (órdenes concretas) para tener en cuenta esos atributos (p.e.*read-standard-mail, read-secret-mail*).

A continuación se muestran ejemplos de una parte de la BC que representa la descripción de las acciones de comunicación, distinguiéndose la comunicación entre usuarios de la comunicación con una computadora.

```
(defconcept Communication
  "concepto que agrupa todas las funciones de comunicacion. En
toda comunicacion se transmite un mensaje"
  :is-primitive
  (:and Unix-Action
        (:all has-object Message)
        (:at-least 1 has-object)))
```

Nótese que siempre que es posible se han incluido restricciones sobre el tipo de objeto con el que se opera, o el origen o destinatario de la acción; así, por ejemplo, se ha establecido que en toda comunicación se transmite por lo menos un mensaje.

```
(defconcept User-Communication
  "comunicacion entre usuarios. El destino y la fuente deben ser
usuarios"
  :is
  (:and Communication
        (:the has-destination User)
        (:the has-source User)))

(defconcept Computer-Communication
  "comunicacion con una computadora"
  :is
  (:and Communication
        (:the has-destination Computer)))
```

Dentro de las operaciones de comunicación se han diferenciado las comunicaciones entre usuarios y las comunicaciones con una computadora.

```
 (defconcept Maintain-Dialog
    "comunicacion interactiva entre usuarios"
  :is
  (:and User-Communication Interactive-Action))

(defconcept Read-Mail
    "lectura de correo por un usuario"
  :is-primitive
  User-Communication)
```

Estos dos últimos ejemplos muestran como se pasa de acciones más abstractas a acciones más concretas según se profundiza en el nivel de la jerarquía. Las acciones *Read-Mail* y *Maintain-Dialog* ya se corresponden cada una con un pequeño grupo de ordenes de Unix. Nótese que *Maintain-Dialog* se define además como la combinación de una acción de comunicación entre usuarios y de una acción interactiva.

6.8.3.2 Clasificación semántica de las acciones

La idea básica de la agrupación semántica es describir y clasificar las órdenes según sus efectos sobre los objetos, y en función de un conjunto restringido de acciones primitivas. La naturaleza de estas acciones se define en términos de un cambio de estado de los objetos que, a su vez, se describen como conceptos de la base de conocimiento.

Se ha definido un conjunto de acciones primitivas que se considera pueden cubrir el conjunto completo de las órdenes del sistema operativo Unix. Estas acciones primitivas o comandos genéricos se han construido a partir del análisis de las descripciones de las órdenes del Unix, apoyándose en estudios empíricos previos y reutilizando el conjunto utilizado en [Hecking 88]. El conjunto de órdenes genéricas debe cumplir dos condiciones:

a) ser capaz de describir todas las operaciones posibles del sistema operativo.

b) permitir la organización de estas operaciones creando una taxonomía de acciones.

En la creación de esta clasificación se podrían haber tenido en cuenta distintos criterios como, por ejemplo, el dominio de la tarea o el tipo de objeto sobre el que se trabaja. No obstante, se ha decidido hacer esta división tomando como discriminante fundamental la naturaleza de la operación realizada por las órdenes. La clasificación realizada se basa en los siguientes criterios:

- el tipo de efecto causado por la acción
- los objetos involucrados en la acción
- las variaciones de una orden debidas a modificadores

Por ejemplo, las acciones "*kill a process*" y "*delete a file*" están relacionadas, ya que son operaciones de eliminación de un determinado objeto, mientras que, si se tuviera en cuenta únicamente el dominio de aplicación, no lo estarían ya que una trabaja sobre archivos y otra sobre procesos. Nótese, que la opción de la agrupación por el dominio de la tarea ya se ha tenido en cuenta en la clasificación temática, de modo que aquí se trata de realizar otra agrupación diferente para organizar la información por similitud de operación. Mediante las primitivas (órdenes genéricas) seleccionadas se ha creado una jerarquía que debería proporcionar la base para representar prácticamente cualquier orden del Unix y de cualquier otro lenguaje de órdenes.

En un primer nivel se asume que cualquier orden puede expresarse y clasificarse, de acuerdo con su propósito principal, en función de una de las siguientes cuatro categorías básicas de primitivas:

- *funciones de cambio*, que modifican objetos del sistema o sus características
- *funciones de información*, que muestran entidades del sistema o proporcionan datos sobre ellas
- *funciones de transferencia*, que tienen en cuenta operaciones de movimiento o traslado de objetos entre dispositivos
- *funciones de construcción*, que crean nuevos objetos, o calculan nuevos valores a partir de objetos ya existentes.

Estas cuatro categorías básicas de acciones primitivas se especializan considerando los criterios secundarios, es decir los objetos involucrados en la acción y las variaciones debidas a modificadores. En la figura 6.16 se muestra una representación gráfica de la clasificación semántica, en la que las acciones de cambio (change) y de construcción (construct) están completamente detalladas. La

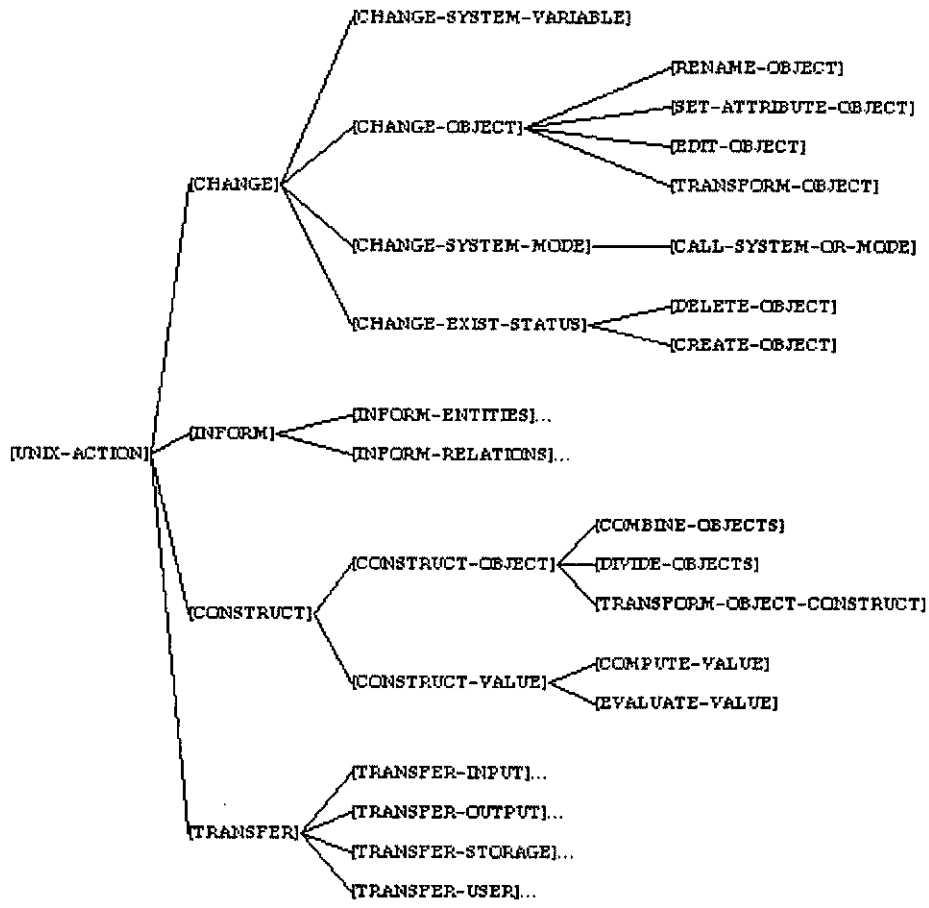


Figura 6.16: Representación de la clasificación semántica de las acciones de Aran

lista completa de acciones, así como detalles de codificación se pueden encontrar en el los apéndices finales.

6.8.3.3 Descripción de las órdenes

Las órdenes y utilidades del sistema operativo se representan como instancias de las acciones, obtenidas mediante la clasificación temática y mediante la clasificación semántica, previamente descritas. Se realiza una indexación manual de las órdenes, que se definen como individuos de todas y cada una de las acciones que le sean aplicables y que contribuyan a su descripción. Además, esta descripción se completa con otras relaciones que consideran otros aspectos, como sus dependencias con el código del sistema, con los objetos del sistema o con otras órdenes. Estas relaciones complementarias enriquecen la representación y permiten la inclusión de estrategias de presentación de información (utilizando el modelo de usuario).

La representación de cada orden se realiza mediante un marco (*frame*) con un conjunto fijo de atributos. Los atributos usados para describir las utilidades de

UNIX se pueden clasificar teniendo en cuenta las características que consideran, en:

- aspectos sintácticos
- aspectos semánticos
- aspectos pragmáticos/tutoriales

Los atributos correspondientes a aspectos sintácticos se usan para describir la estructura sintáctica de la orden y su relación con el código real del sistema. Es decir su nombre, su formato de escritura y cuál es el fichero que la implementa (mediante las relaciones *has-name, syntax y related-file*). Se podrían haber tenido en cuenta otros detalles complementarios como los parámetros o las opciones, pero no se ha hecho debido a que esta información está contenida en la página asociada del manual, a la que el usuario tiene acceso desde esta representación.

La semántica de una orden se expresa mediante una cadena de caracteres en la cual se describe su propósito y las características reseñables. Se incluye en la descripción de la orden mediante la relación *desription-text*. Este texto explicativo complementa la información proporcionada por la propia clasificación de la orden como instancia de distintas acciones genéricas.

Los aspectos pragmáticos o tutoriales de una orden determinada, son muy importantes para un sistema de ayuda y se contemplan en varias relaciones. Con este tipo de información se trata de prevenir los posibles errores del usuario y facilitar la comprensión de la utilidad. Por ejemplo, si se proporciona como respuesta a una pregunta sólo una orden, en muchos casos el usuario no sabe como proceder, por lo que se incluyen ejemplos concretos de uso, que pueden ser especialmente útiles para usuarios con poca experiencia. También hay que ayudar al usuario a integrar las nuevas funciones que aprende dentro del modelo mental que tiene del sistema. Para esto se representan mediante *has-related-command* y *see-also* las órdenes relacionadas, bien porque tienen un efecto similar o porque pertenecen al mismo subdominio (parte de las cuales se han obtenido de la sección *see also* del manual). Esto permite la inclusión de nuevos enlaces con otras órdenes que no se contemplaban en la sección *see also*, bien por motivos de espacio, o bien porque se han incluido con posterioridad. La relación *has-related-command,* que se muestra a continuación, se representa de modo que se restringen sus posibles valores órdenes (mediante *:range Unix-Action*, sólo se permiten instancias de *Unix-Action*), a la vez que declara como inversa de sí misma (mediante *:inverse has-related-command*).

```
(defrelation has-related-command
    "ordenes relacionadas"
  :inverse has-related-command
  :range Unix-Action
  )
```

Esta representación simplifica la codificación de la información, ya que si se afirma que una orden está enlazada con otra por medio de esta relación, esta segunda

orden queda automáticamente enlazada con la anterior (situación que no siempre se respeta en el manual de Unix). Mediante estas relaciones se le ayuda al usuario a recordar funciones que es posible que previamente haya utilizado, y a clasificar las nuevas utilidades que aprende dentro del subconjunto de ordenes que ya conoce.

Otro aspecto importante considerado en la descripción tutorial de una utilidad, son los conceptos relacionados y los conceptos los que se presupone que deben conocerse para asimilar y usar la orden de forma correcta. Con este motivo se declaran dos relaciones *has-related-concept* y *prerrequisite-concept*, cuyos valores serán conceptos de la base de conocimiento (y no instancias que es lo habitual). Esta circunstancia permite enriquecer la interrelación entre la información representada en la BC, a la vez que se mantiene un criterio bien fundamentado de organización, que es el impuesto por la clasificación mediante la relación de pertenencia a un determinado concepto. Como ejemplo, a continuación se muestra una descripción completa de la orden *chmod*:

```
(tell (:about i-chmod
        Set-Attribute-Object
        File-Manipulation
        (has-name "chmod")
        (syntax  "chmod [ -fR ] mode filename")
        (related-file "/bin/chmod")
        (description-text "permite modificar los permisos de
escritura, lectura o ejecución de un fichero o directorio. El
permiso de ejecucion con un directorio implica que se puede
acceder y listar ese directorio")
        (perrequisite-concept file)
        (prerrequisite-concept directory)
        (prerrequisite-concept permission)
        (prerrequisite-concept group)
        (has-related-concept file-system)
        (has-related-command i-chgrp)
        (see-also i-ls)
        (see-also i-chown)
        (see-also i-sh)
        (see-also i-csh)
        (has-example "La orden, chmod 777 foo, proporciona todos
los permisos a todos los usuarios")
        (has-example "La orden, chmod +x foo, anade el permiso de
ejecucion al fichero")))
```

La orden *chmod* se declara como una instancia de *set-attribute-object* y *file-manipulation*. De su clasificación como perteneciente a estos dos conceptos (acciones), se obtiene que es una operación que modifica algún atributo del objeto sobre el que opera y que el tipo de objeto sobre el que actúa es un fichero. Con las relaciones *prerrequiste-concept* y *has-related-concept* se establece que para comprender la orden hace falta conocer previamente los conceptos *file, directory, permission* y *group*, y que otro elemento relacionado es *file-system*. Las relaciones *has-related-commad, see-also* y *has-example* determinan las cinco órdenes relacionadas y dos ejemplos de uso, respectivamente. Estas referencias a otros conceptos y órdenes relacionadas con la solicitud del usuario son útiles para extender y completar el conocimiento que tiene el usuario sobre el dominio.

La inclusión de esta información tutorial en la representación del dominio, unida a la existencia de un modelo de usuario que permite hacer suposiciones sobre el conocimiento del utilizador, posibilita la implementación de diversas tácticas para la presentación de información. Por ejemplo, si es un usuario novel cuando accede a la descripción de una orden se le muestran los ejemplos, mientras que si el usuario tiene un mayor nivel de experiencia debe acceder a ellos expresamente mediante una opción del interfaz. Otro ejemplo es la presentación de los conceptos involucrados en la descripción, en la que no se muestran aquellos conceptos prerrequisitos o relacionados que se suponen conocidos por el usuario. Estas estrategias se podrían considerar como un modelo tutorial muy básico, en la terminología utilizada por los tutores inteligentes como se ha descrito en el capítulo dos.

6.8.4 Modificabilidad y ampliabilidad del modelo del dominio

Como ya se ha tratado Unix es un sistema abierto y muy extenso, para el cual la obtención de un modelo conceptual completo y completamente correcto consideramos que es, hoy en día, un objetivo demasiado ambicioso. La construcción de estas representaciones se verán facilitadas en un futuro por proyectos como los descritos en el apartado 3.6, pero actualmente exigen un esfuerzo de desarrollo que excede el ámbito de este trabajo. Por tanto se ha desarrollado un modelo de dominio parcial, en el que desde un primer momento se han tenido en cuenta la sencillez de los procesos de ampliación y modificación de la información representada.

Estos aspectos, junto con la reutilización de información previa, nos hicieron decidir el empleo de una representación basada en lógicas descriptivas y en concreto un formalismo como Loom. Además de su expresividad y de su eficiencia, proporciona unas capacidades de inferencia y razonamiento que simplifican el mantenimiento y desarrollo progresivo de la base de conocimiento (BC). Por ejemplo, permite trabajar con suposición de terminología abierta [Weida 95], de modo que pueden existir conceptos relevantes en este dominio que todavía no han sido explícitamente definidos en la base de conocimiento. Proporciona herramientas que permiten detectar la definición de conocimiento incoherente, así como obtener todos los elementos de la BC que dependen de uno dado, esto simplifica la eliminación de conceptos o relaciones, así como la modificación de otros existentes.

Además se dispone de un clasificador automático que es útil tanto para la construcción de la taxonomía como para el mantenimiento de su consistencia [Weida 91]. Posibilita realizar una interconexión e interrelación muy rica entre la información del dominio, sin que por esto aumente la dificultad de su gestión o de su coherencia. Los conceptos se estructuran mediante la relación privilegiada de generalización-especialización (tipo-de) y las instancias mediante la relación de pertenencia o clasificación debajo de un concepto (es-un).

No obstante, a pesar de todas estas capacidades cuando aumenta el número de elementos representados, se hace necesaria la existencia de una herramienta gráfica que simplifique la interacción con el sistema de representación. Como Loom no incluye ningún tipo de entorno gráfico, dentro de nuestro grupo de trabajo hemos

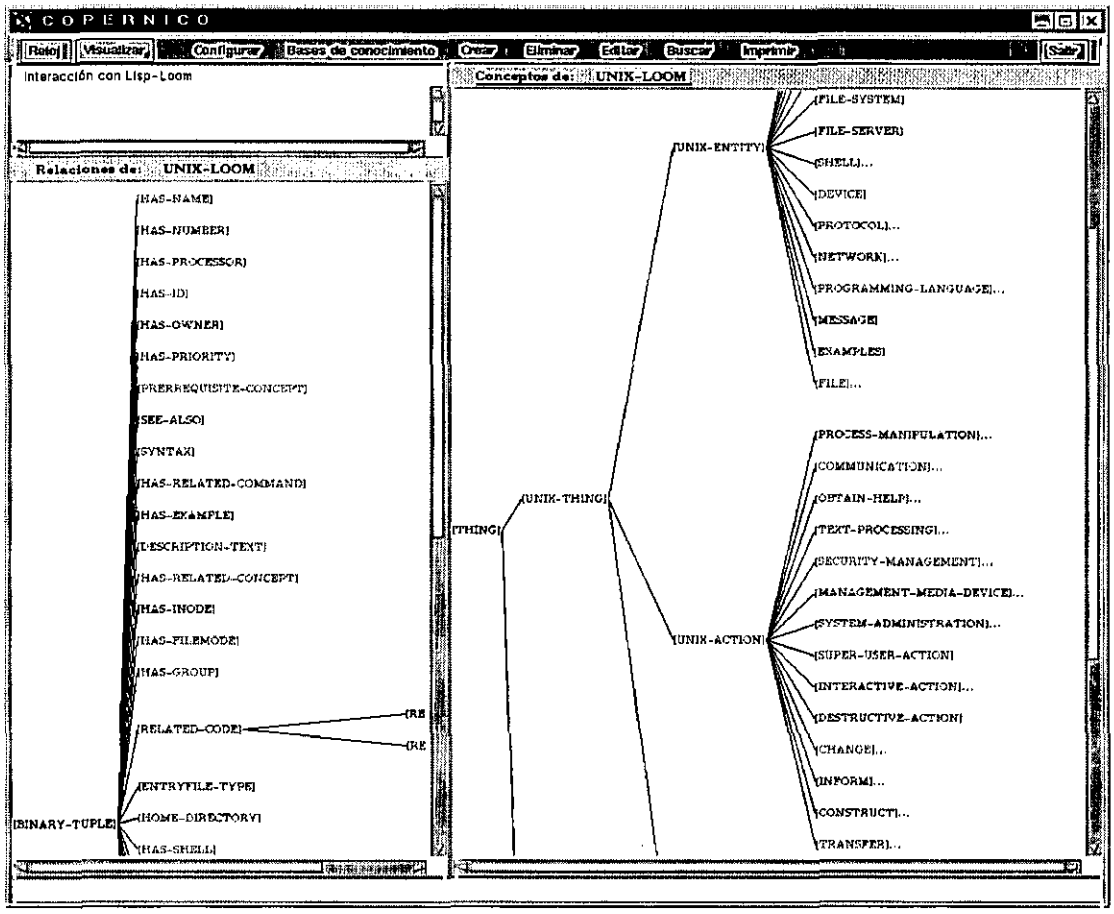


Figura 6.17: Pantalla de interacción con el sistema Copérnico. En las ventanas de Copérnico se visualizan una parte de los conceptos y relaciones de la base de conocimiento de Aran.

desarrollado el sistema Copérnico [Blanco 95] (fig. 6.17). Copérnico es una herramienta que facilita la creación, visualización y mantenimiento de bases de conocimiento, que ha sido esencial para la construcción de la BC de Aran y en concreto para el modelo del dominio. Además de simplificar los procesos de mantenimiento y depuración del contenido de la BC, su utilización es clave en los procesos de inclusión de conocimiento proveniente de otros sistemas. Facilita la determinación de los efectos que tiene la inclusión de esta nueva información en el modelo y la resolución de los conflictos que esta ampliación pueda provocar. Una descripción completa de las utilidades de Copérnico, su uso en el desarrollo de bases de conocimiento, así como detalles de su uso e implementación se pueden encontrar en [González-Calero 96].

6.8.5 Ayuda y aprendizaje en Aran

El objetivo del sistema Aran es proporcionar información al usuario cuando este encuentra un problema que no sabe resolver. Esta información debe permitir que el

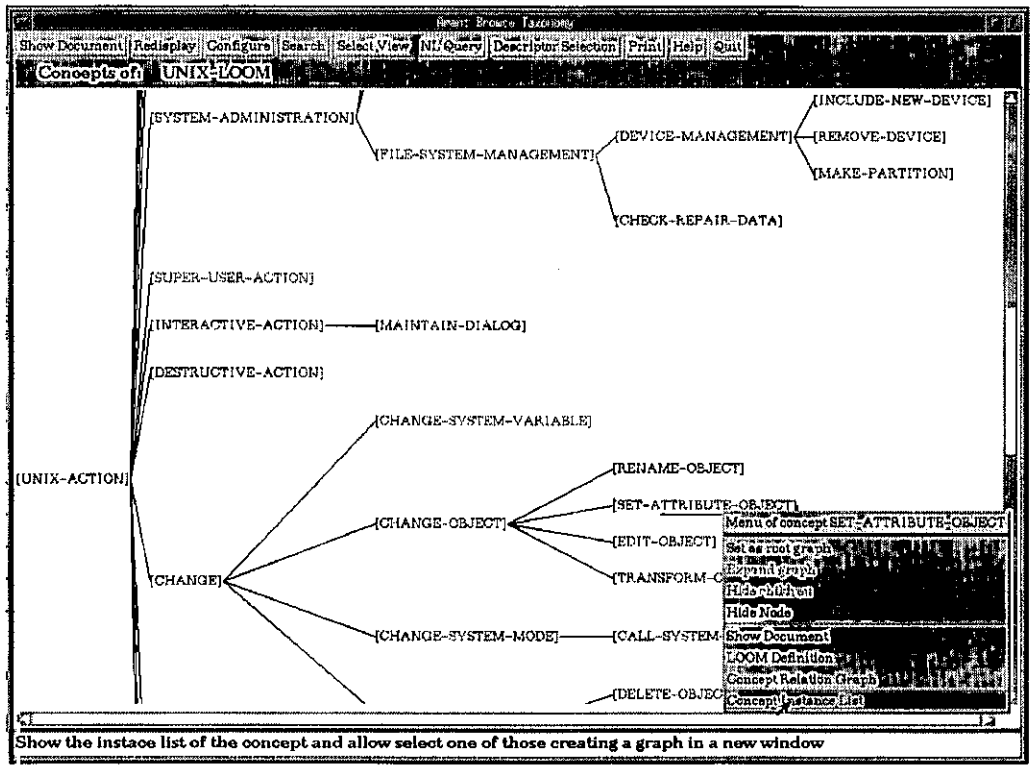


Figura 6.18: Interfaz de visualización del modelo del dominio, presentando únicamente los conceptos, y centrada en las acciones definidas en el Unix (*unix-action*). Se muestra el menú asociado a la acción *set-attribute-object* donde está seleccionada la opción que permite acceder a su lista de instancias

usuario continúe con su trabajo, a la vez que debe servirle para aumentar su conocimiento sobre el sistema (según nuestro planteamiento, descrito en el apartado 4.2.3). Aran tiene por una parte un objetivo de ayuda a corto plazo, en busca de una mejora del rendimiento, pero por otro lado Aran aprovecha las solicitudes de ayuda del usuario para facilitar un aumento del conocimiento que éste tiene sobre la estructura y funcionamiento del sistema operativo. Este último es un objetivo instructivo a largo plazo, en nuestra opinión tan importante como el primero, ya que si el que utiliza el sistema conoce mejor su funcionamiento interno, podrá evitar las situaciones de error o será capaz de solucionarlas por sí mismo.

En Aran, la mejora en el rendimiento se consigue mediante la inclusión de tres métodos eficientes de acceso a la información. Para este propósito se realizan tres indexaciones de la información en base a distintos criterios y, a partir de ellas, se proporcionan unas interfaces adecuadas. La documentación electrónica sobre las utilidades del sistema se indexa utilizando técnicas de recuperación de información con el modelo del espacio vectorial (al igual que se hace en Argos), se indexa también mediante un conjunto de descriptores controlados, que dan una idea de su funcionalidad, y finalmente se indexa en base al modelo del dominio. Estas diferentes indexaciones permiten al usuario acceder a la información de distintas formas. El usuario puede expresar su necesidad de ayuda en lenguaje natural,

seleccionar los descriptores que mejor definen su necesidad o simplemente realizar una exploración de la base de conocimiento, en la que la documentación (así como otra información complementaria) se encuentra agrupada en torno a los elementos significativos del dominio.

El objetivo educativo de Aran se concreta en proporcionar al usuario una visión completa del sistema, no únicamente apoyándose en la resolución de los problemas concretos. Se busca que el usuario adquiera un modelo adecuado del sistema. La consecución de este propósito se obtiene mediante la interrelación de la información y, sobre todo, mediante la base de conocimiento. Cuando se le muestra documentación al usuario siempre se presenta dentro del contexto, por ejemplo junto con las relaciones existentes entre las órdenes o mostrando los conceptos implicados. De este modo se mejora la comprensión y la asimilación de la filosofía del sistema. La pieza clave para aumentar la comprensión de la estructura y operaciones del sistema es el modelo explícito del sistema operativo Unix. Para mejorar el contexto en el que se presenta la información, desde los otros modos de interacción, mediante lenguaje natural o mediante descriptores, se puede acceder directamente al modelo del dominio y ver cuáles son conceptos que indexan un documento concreto (conceptos relacionados).

Este modelo conceptual es directamente inspeccionable desde la interfaz por el usuario, quien puede así aprender los términos que se utilizan en ese área concreta, cuál es su significado preciso, qué conceptos están relacionados con uno determinado o cuáles son las utilidades que realizan una determinada operación. Como ya se tratado en el apartado 6.5, la interfaz del sistema presenta un grafo con la taxonomía de conceptos del Unix (fig. 6.18). En este modo de interacción bien como opciones de menú o como opciones asociadas a los nodos se proporcionan una serie de operaciones que permiten la navegación a través del grafo y el acceso diferentes tipos de información. Aquí cabe destacar la opción *Concept Relation Graph,* que muestra gráficamente la definición de un concepto y cuál es su relación con los otros objetos del dominio (fig. 6.19).

Hay dos posibles vistas del dominio, una que sólo muestra los conceptos y otra que también incluye las instancias que representan a las órdenes u objetos concretos del Unix (la vista se elige mediante *Select View*). Como la información a mostrar es muy amplia, habitualmente se trabajará sólo con la vista de conceptos a partir de la cual se puede acceder a las instancias, como muestra la figura 6.20. La visualización de las instancias es, a su vez, otro grafo que se puede activar con el ratón. Cada uno de sus elementos son nodos de un hipertexto que permiten acceder a nuevos conceptos o instancias. Existen también diversas posibilidades de configuración de estas visualizaciones como, por ejemplo, si se quiere ver como un árbol con elementos duplicados o como un grafo, si se desea ver en vertical o en horizontal, o si se quieren mostrar con colores los diferentes elementos (utilizando el botón *Configure*). Además, la visualización de las instancias (fundamentalmente órdenes) se ve modificada por el contenido del modelo del usuario (como se ha tratado en 6.7.5). Por ejemplo, al mostrar una orden si no se ha desactivado el modelo del usuario no se presentan aquellos conceptos prerrequisito que ya se suponen conocidos.

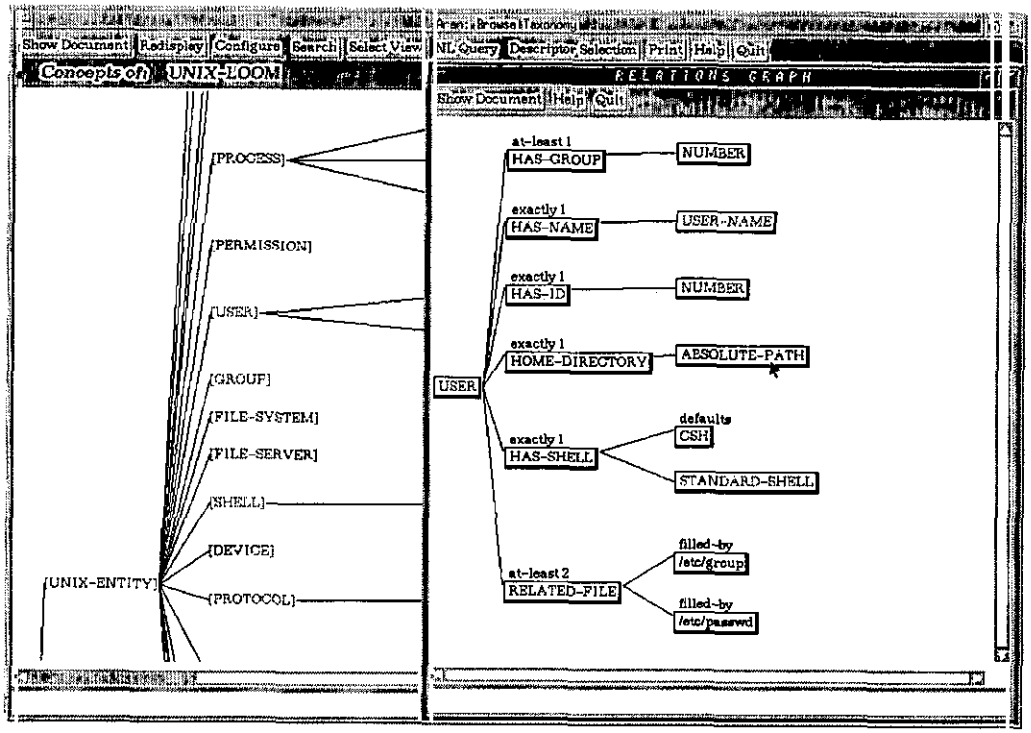


Figura 6.19: Visualización del grafo de relaciones del concepto *user*. Presenta de modo gráfico su definición y su relación con otros conceptos del dominio.

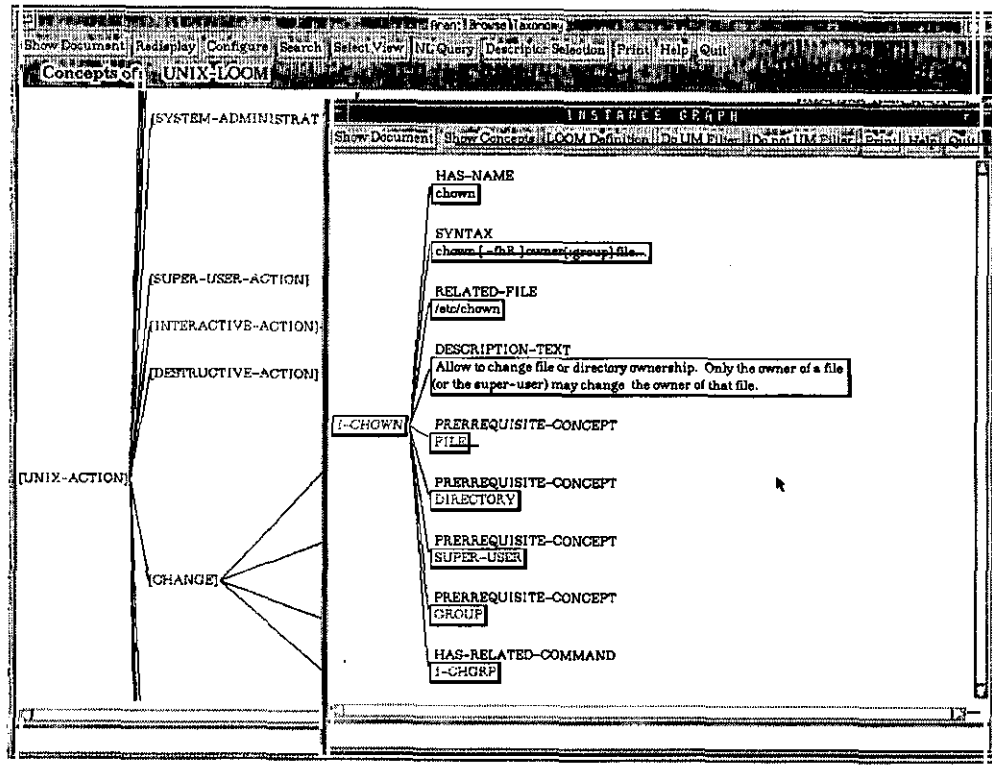


Figura 6.20: A partir de la taxonomía de conceptos se ha pasado a visualizar la orden *chown* que es una de las instancias de *set-attribute-objetct*.

6.9 Resumen

A lo largo de este capítulo se ha descrito el sistema Aran, un asistente inteligente para el sistema operativo Unix. Se trata de una continuación de la línea de trabajo desarrollada con Argos, introduciendo una representación explícita del conocimiento con el fin de mejorar la efectividad de la ayuda. Aran supone la materialización completa del modelo de sistema de ayuda inteligente propuesto en este trabajo.

Para desarrollar el sistema Aran hemos analizado, en primer lugar, cuales son las características de la documentación que hacen que ésta resulte útil para proporcionar una ayuda efectiva, destacando el problema de su insuficiente estructuración y su dificultad de acceso. La construcción de una representación de la información de diseño del sistema (modelo del dominio) permite estructurar e integrar distintos tipos de información. A continuación hemos detallado el planteamiento y la organización del sistema, cómo se integran las cuatro tecnologías para ofrecer las funcionalidades de ayuda, a la vez que se reutiliza información para reducir el esfuerzo de desarrollo.

Seguidamente se presenta la arquitectura del sistema con su tres bloques principales: la interfaz, la indexación automática de información y la base de conocimiento. Hemos descrito con detalle cada uno de los módulos de Aran: la interfaz, el módulo de RI, el análisis formal de conceptos, el modelo del usuario y el modelo del dominio, haciendo un hincapié especial en estos dos últimos. El modelo del dominio permite una indexación basada en conocimiento de la documentación, a la vez que posibilita el desarrollo de un modelo del usuario más detallado y completo. Ahora se realiza un modelado más dinámico y en el cual a un usuario no se aplica un sólo nivel de experiencia o de interés global, sino que se considera su posible especialización o mayor conocimiento de algunas áreas del S.O. Unix.

Aran demuestra que es posible ofrecer una serie de funcionalidades que proporcionan una ayuda eficaz a los usuarios, en un dominio real y con un coste razonable. Con la inclusión de nuevas técnicas, como la representación de conocimiento o la indexación mediante el análisis formal de conceptos, hemos solucionado los problemas que previamente habíamos detectados en Argos. Esto nos ha permitido introducir otros tipos de interacción en los que el conocimiento previo del dominio por parte del usuario ya no es crucial, como la inspección del dominio o la selección de descriptores. Además ahora se dispone de una información muy interrelacionada a la vez que fácilmente mantenible y ampliable. En este último aspecto, como en el desarrollo y mantenimiento del MU, ha sido clave la elección del lenguaje Loom, un formalismo de representación estándar que proporciona un amplio rango de funcionalidades. El MU permite establecer nuevas estrategias de presentación de información para los distintos modos de interacción.

Al igual que con Argos se ha realizado únicamente una evaluación formativa del sistema utilizando a estudiantes de doctorado y a miembros de nuestro grupo de trabajo. Esta evaluación ha permitido mejorar diversos puntos del sistema y sobre todo en la interacción. No se ha realizado una evaluación formal completa, sumativa, ya que no se contemplaba dentro de los objetivos de este trabajo, no obstante este es uno de los siguientes aspectos a considerar como futuro trabajo.

CAPITULO 7

CONCLUSIONES Y FUTUROS DESARROLLOS

7.1 Principales aportaciones

Mediante la realización del trabajo que se describe en esta memoria hemos podido confirmar nuestra principal hipótesis: mediante la integración de técnicas suficientemente probadas en diversas áreas de la Informática es posible construir sistemas de ayuda efectivos que se adaptan al usuario, y todo ello con un coste de desarrollo y mantenimiento razonable. La posibilidad de construir sistemas con estas características la hemos confirmado mediante la construcción de los sistemas Argos y Aran. Sin embargo, debemos añadir que la evaluación de la efectividad de los sistemas construidos tan sólo ha sido formativa. Es decir, sólo hemos realizado pruebas con un pequeño grupo de estudiantes de tercer ciclo y estas pruebas se han realizado, en su mayor parte, de forma simultánea a la construcción de ambos sistemas. Una correcta evaluación sumativa, como sucede con tantos otros sistemas informáticos, implica unos medios y unos periodos de tiempo que están más allá de los límites de nuestro trabajo.

La aproximación que hemos seguido para construir nuestros sistemas la hemos abordado inicialmente desde una revisión crítica de los principales aspectos del proceso de utilización de las computadoras en la enseñanza. Para ello, se ha analizado en primer lugar la problemática asociada con diversos enfoques educativos, dedicando una atención especial a los tutores inteligentes y a las razones que han limitado la generalización de su uso. Las consecuencias de este análisis se aplican a los sistemas de ayuda, que hemos planteado como una aproximación más sencilla y realista de los sistemas de enseñanza basados en computadora.

Nuestra visión de lo que debe ser la asistencia al usuario no es solamente proporcionar a éste la información necesaria para que complete la tarea con la que tiene dificultades, sino que mantenemos a la vez un enfoque instruccional, de modo que se promueve una mayor comprensión de la estructura del sistema. Para llegar a esta conclusión hemos realizado previamente un estudio de los sistemas de ayuda, centrándonos en los sistemas de ayuda para entornos software extensos, como son los sistemas operativos o algunas herramientas CASE de propósito general. Se han

identificado las principales necesidades de ayuda que se le plantean al usuario de estos entornos y se han analizado las diversas propuestas que buscan facilitar simultáneamente tanto el aprendizaje como el uso de estas aplicaciones informáticas. Hemos analizado los factores que influyen en el diseño y construcción de los sistemas de ayuda, concluyendo que la característica distintiva de la ayuda inteligente es la de la adaptación a las circunstancias en las que se solicita esta ayuda. Otra conclusión que hemos obtenido ha sido que la disposición de un modelo explícito del usuario es clave para lograr dicha adaptación, concibiendo este modelo como una simplificación pragmática del concepto de modelo de alumno utilizado en los ITS.

La construcción de los sistemas Argos y Aran siguiendo las directrices del modelo de sistema de ayuda propuesto en el capítulo 4 de esta memoria, constituye el núcleo de esta tesis. De esta forma, se demuestra la viabilidad de un modelo de sistema de ayuda capaz de proporcionar una ayuda efectiva, adaptada a los usuarios, en dominios reales y con un bajo coste de desarrollo. Las características clave de este modelo son: activación del asistente por el usuario, ayuda basada en acceso a documentación, adaptación mediante un modelado explícito del usuario y utilización de una interfaz multimodal.

Nuestro modelo resulta factible porque se basa en la integración de tecnologías ampliamente probadas y en la reutilización de información de distintos tipos. Las tecnologías integradas son: la recuperación de información, el hipertexto, el modelado de usuario y la representación explícita del conocimiento. La combinación de estas técnicas permite facilitar tanto el acceso a la información como su comprensión por diversos tipos de usuarios. La reutilización de información implica desde aprovechar documentación en formato electrónico, hasta la reutilización -parcial- de bases de conocimiento o de modelos de usuario usados previamente en otros sistemas. En cualquier caso, debemos señalar que con este trabajo no se ha pretendido dar una solución general al problema de la reutilización de información, ya que éste es un proceso que depende en gran medida de cada dominio concreto. No obstante, hay prácticas que contribuyen a su simplificación, como la adecuada elección de un formalismo estándar y con un suficiente poder expresivo para la representación explícita del conocimiento.

El primer sistema de ayuda desarrollado, Argos, es un sistema en el que se integran técnicas de recuperación de información, de modelado de usuario y de hipertexto. En la evaluación de este primer sistema quedó de manifiesto que las técnicas de interacción hombre-computadora juegan un papel fundamental para obtener un asistente eficaz y de fácil manejo. Sin embargo, en esta misma fase de experimentación y prueba se detectó que a pesar de la interfaz, la influencia del conocimiento previo del vocabulario del dominio por parte del usuario era un factor clave para el éxito del sistema. Además se constató que una insuficiente estructuración de la información también influía negativamente en su asimilación por parte del usuario.

Sobre la base de nuestra experiencia con Argos se desarrolla el sistema Aran que supone la materialización completa del modelo de asistente inteligente propuesto en este trabajo. Aran introduce una representación explícita del conocimiento con el fin

de mejorar la efectividad de la ayuda. Esta representación explícita constituye un modelo del dominio que permite una indexación basada en conocimiento de la documentación, a la vez que posibilita el desarrollo de un modelo del usuario más detallado y completo. Con la inclusión de las técnicas de representación de conocimiento y el análisis formal de conceptos, hemos solucionado los problemas previamente detectados en Argos. Esto nos ha permitido introducir otros tipos de interacción en los que el conocimiento previo del dominio por parte del usuario ya no es crucial. Esta nueva interacción permite la inspección directa del dominio y la selección de descriptores. Además ahora se dispone de una información más rica e interrelacionada, lo cual simplifica su comprensión y asimilación, sin afectar gravemente a la mantenibilidad y ampliabilidad del sistema.

Finalmente, cabe destacar que los modelos de usuario desarrollados para ambos sistemas tienen como principal característica su pragmatismo, lo cual posibilita que sean útiles y aplicables en dominios extensos. Estos modelos se basan en clases de usuarios en el caso de Argos y en estereotipos en el caso de Aran. El contenido de estos modelos ha permitido la adaptación a los usuarios concretos mediante estrategias de presentación de la información. Además, como en nuestros sistemas de ayuda el usuario tiene la iniciativa, se ha facilitado a éste la inspección de estos modelos así como su desactivación.

7.2 Futuros desarrollos

Las continuación de este trabajo tiene dos etapas principales, la primera relacionada con una evaluación más completa de los sistemas construidos. La segunda estaría basada en la primera y tendría como objetivo la profundización en el estudio de los modelos de usuario y su aplicación en áreas diferentes.

Realizar una evaluación más completa de los sistemas de ayuda desarrollados, utilizando conjuntos amplios de usuarios, es un tema fundamental y cuya realización consideramos que sería posible realizar próximamente. Esta evaluación quedaba fuera de los objetivos iniciales y posibilidades de este trabajo, por lo que únicamente se ha realizado una evaluación formativa anteriormente reseñada. La evaluación que ahora proyectamos permitirá introducir mejoras en las interfaces de los asistentes, detectando qué problemas surgen en la interacción con los nuevos usuarios, así como sofisticar otros aspectos como, por ejemplo, la inspección de los modelos de usuario, la adquisición de suposiciones y las estrategias de presentación de información.

Los resultados de esta primera etapa serían importante para realizar una segunda etapa que incluiría la adaptación del modelo de usuario, probado en los sistemas construidos, a otros conjuntos de documentos y con un propósito más específico. Nuestro proyecto es aplicarlo al filtrado de información. Actualmente hemos comenzado el diseño de una aplicación que utilizaría este modelo mejorado para realizar un filtrado de los mensajes que provienen de distintas listas de distribución existentes en Internet y referidas al sistema operativo Unix. En estas listas se produce un flujo de datos muy grande, en el que se incluye información muy valiosa para distintos tipos de usuarios, pero que por su amplitud, tanto en cantidad como

en variedad, hace que sea difícil de manejar, por lo que disminuye su utilidad. El usuario tiene que dedicar mucho tiempo a leer el correo, descartando los mensajes que no le son útiles. El objetivo sería diseñar un asistente que modelase las necesidades de información del usuario y las utilizara para seleccionar cuales de estos mensajes recibidos pueden ser interesantes. Este propósito de filtrado implica una ampliación del modelo que considere características a largo plazo, ya que los intereses de un usuario no varían tan a corto plazo como en el caso de los sistemas Argos y Aran.

Por otra parte, la experiencia en el desarrollo del modelo de usuario de Aran permite plantear la creación de un entorno (*shell*) de modelado de usuario basada en estereotipos. La disposición de este entorno simplificaría la inclusión de modelos de usuario en sistemas basados en conocimiento en general y en particular en otros sistemas de ayuda inteligente. Partiendo de la experiencia adquirida en la construcción de los sistemas Aran y Copérnico se puede investigar en la producción de un entorno de modelado que proporcione diversas funcionalidades como, por ejemplo, el mantenimiento de la coherencia o la revisión dinámica de los estereotipos aplicados, y que incluya una interfaz gráfica que facilite el proceso de definición y refinamiento de los estereotipos de usuario. Este trabajo puede contribuir también a que dichos modelos sean inspeccionables de una forma sencilla por parte de los usuarios.

APENDICE

EJEMPLOS DE USO DEL SISTEMA ARGOS Y DETALLES DE IMPLEMENTACION DEL SISTEMA ARAN

En este apéndice se presentan tres sesiones de utilización del sistema Argos, a modo de ejemplo de sus funcionalidades y destacando el papel del modelo del usuario en su comportamiento adaptativo. También se proporcionan detalles de implementación del sistema Aran, centrándonos en su base de conocimiento.

En el primer apartado se muestran ejemplos de uso del sistema Argos, que tratan de mostrar posibles interacciones de usuarios reales con el sistema. En las capturas de pantallas se incluye también la visualización del modelo del usuario. La descripción general del sistema Argos se proporciona en el capítulo 5. Detalles de implementación, pruebas y código de la versión actual se pueden encontrar en [Cigarrán 96].

En el segundo apartado se muestran ejemplos de la codificación del contenido de la base de conocimiento del sistema Aran. Concretamente, del modelo del usuario y de los estereotipos identificados, de la definición de los objetos y acciones del dominio, y de las órdenes concretas (instancias) que se han representado. La organización general del sistema Aran se describe en el capítulo 6 y los aspectos relacionados con su base de conocimiento en los epígrafes 6.7 y 6.8.

A.1 Ejemplos de sesiones con Argos

A continuación presentamos unas sesiones con Argos, mostrando diversas consultas y las respuestas que se proporcionan, a modo de ejemplo de posibles utilizaciones por parte de un usuario. Además se incluye también el modelo del usuario para ver cual es la evolución de su contenido y como afecta al comportamiento del sistema. Las solicitudes de los usuarios se ha partido de las utilizadas en [Maarek 91b] y de ejemplos de interacciones de alumnos que han participado en la evaluación informal del sistema.

A.1.1 Ejemplo 1

Cuando un usuario que está trabajando en el entorno XWINDOW encuentra algún problema que no sabe resolver puede activar el asistente mediante la orden *argos*. Como se muestra en la primera ilustración aparece la interfaz de Argos, donde el usuario puede realizar sus preguntas en lenguaje natural (inglés) en el campo *Query*. Automáticamente se detectan el interés y la experiencia del usuario,

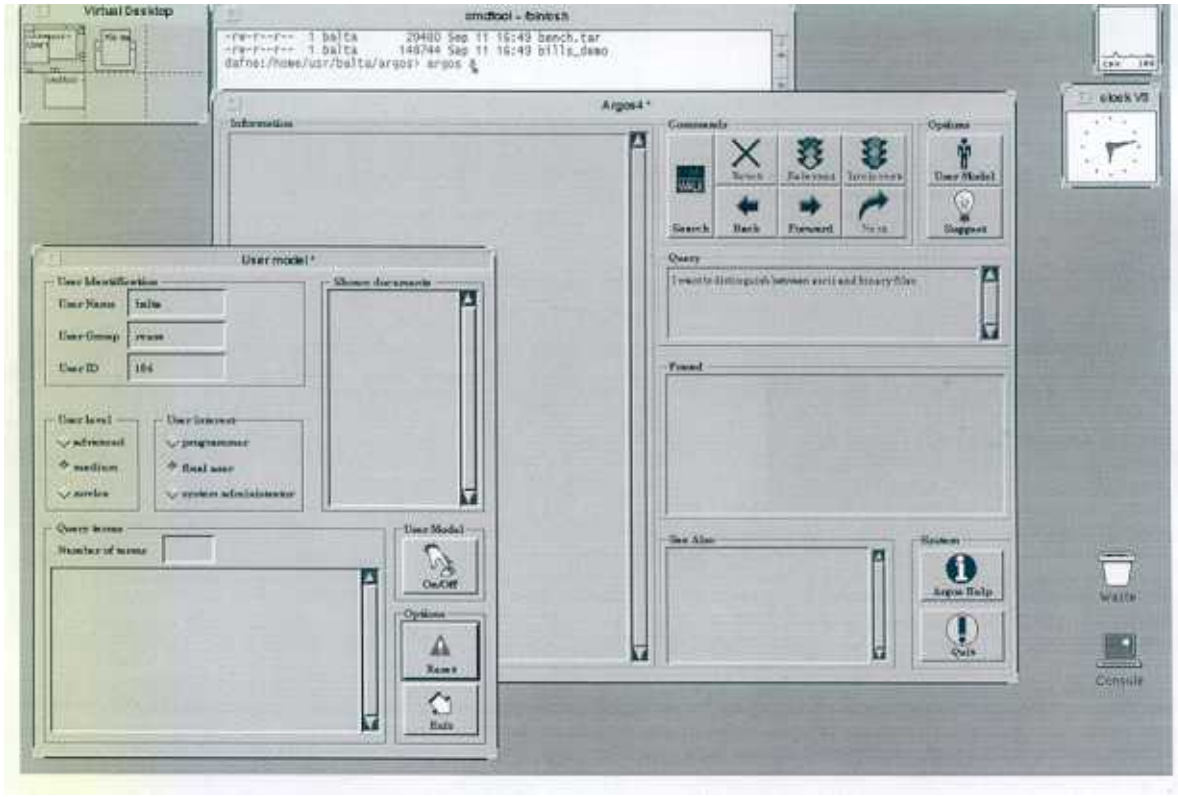


Figura A.1: Interfaz de Argos y visualización del modelo del usuario

mediante el análisis de su interacción anterior con el sistema operativo, que quedan reflejados en el modelo del usuario. En este caso se ha formulado la solicitud "I want to distinguish between ascii and binary files" y como muestra el modelo del

usuario se ha detectado que el usuario tiene un nivel medio y es un usuario final (están activados los campos *User level* y *User interest* de *User model*).

A continuación el usuario pulsa el botón *Search* para realizar la búsqueda y Argos produce como resultado los ocho documentos del campo *Found* (fig. A.2). El documento que el asistente considera más adecuado para el usuario se presenta directamente en el campo *Información* (en este caso *uudecode*). Como este documento no es relevante a la petición el usuario puede utilizar el botón *Next* para visualizar los otros documentos en el orden que Argos considera adecuado, teniendo en cuenta la relevancia obtenida por la función de similitud y las estrategias de presentación en función del contenido del modelo del usuario. Si el usuario reconoce

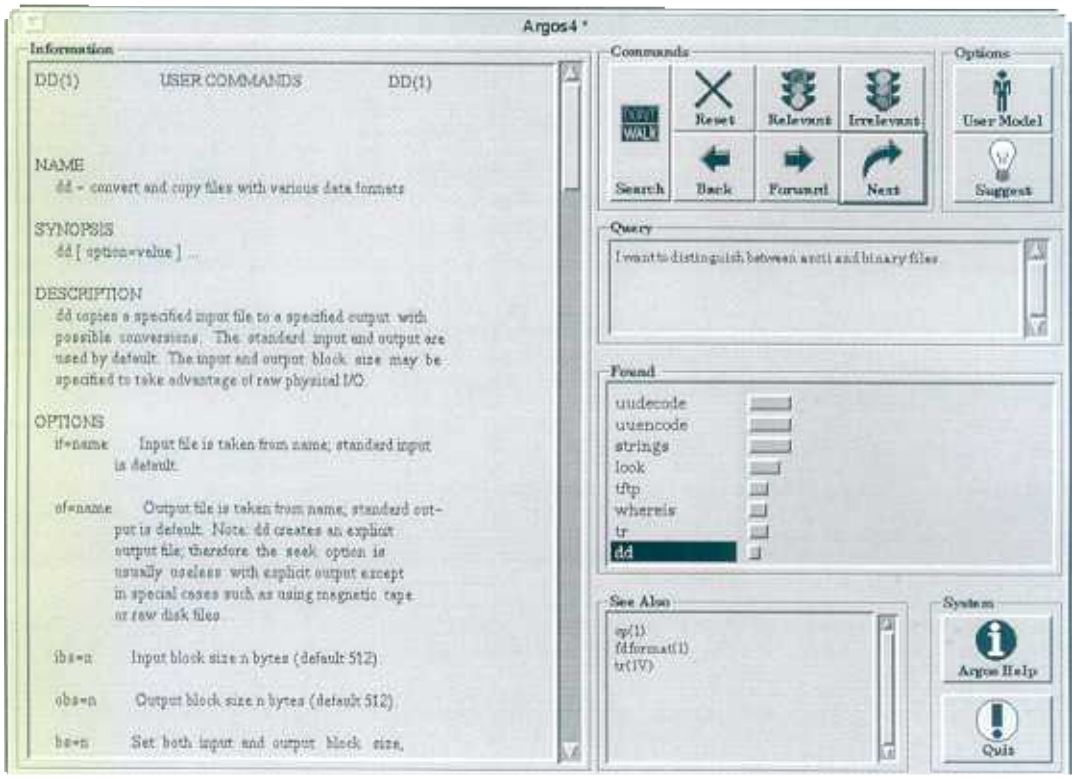


Figura A.2

como adecuado alguno de los nombres que aparecen en los paneles *Found* o *See Also* puede pulsar sobre ellos con el ratón para que se muestren en el panel *Information*.

Una vez realizada la consulta y cuando el usuario ha recorrido todos los documentos que le ha proporcionado Argos, el contenido del modelo del usuario es el mostrado en la figura A.3. Se puede observar como se ha procesado la pregunta realizada por el usuario, filtrando las palabras sin contenido semántico, para obtener los cuatro términos que se muestran en el panel *Query terms*. En el panel Shown documents se muestran los documentos a los que ha accedido el usuario a lo largo de esta

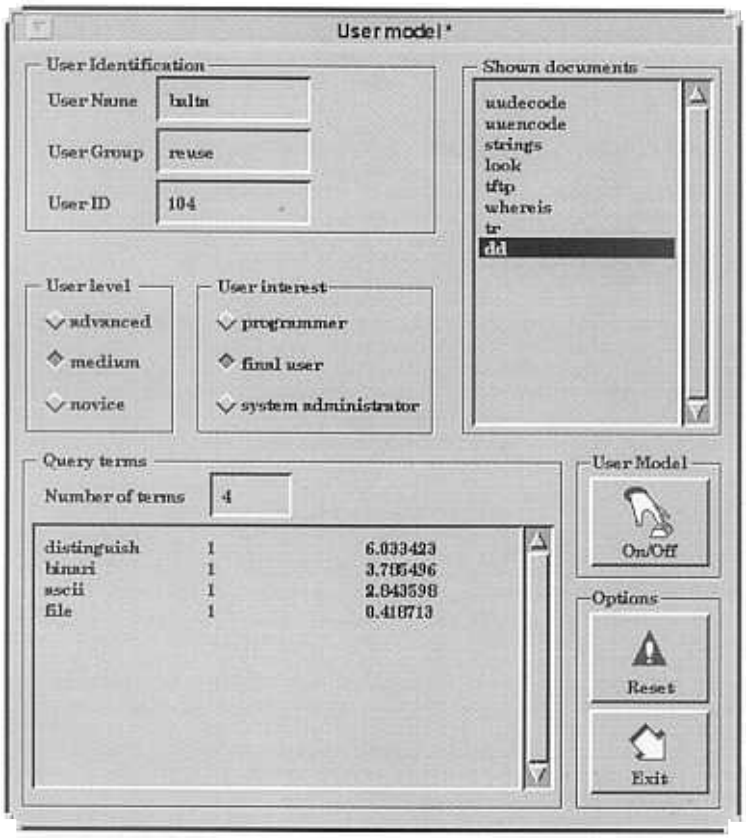


Figura A.3

sesión. En este caso ha accedido a todos los documentos, a los ocho que obtuvo respuesta y lo ha hecho en el orden proporcionado por el sistema.

Como con la primera solicitud no se ha obtenido la información deseada, el usuario la refina mediante la consulta "I want determine the contents type of a file". Una vez realizada la búsqueda el resultado obtenido es el mostrado en la figura A.4, que si es relevante para el usuario. Se obtiene como primer documento la información sobre la orden *file* que permite realizar la operación deseada. Aunque hay otros documentos cuya relevancia es superior como se puede observar mediante las barras gráficas situadas a continuación de los nombres, por ejemplo, *uudecode* o *uuencode*, como ya han sido mostrados al usuario ahora ya no se presentan directamente en el campo *Information*.

El contenido del modelo del usuario se muestra en la figura A.5. Se puede ver como ahora el número de términos utilizados para representar la consulta es de siete y que *file* apareció dos veces (campo *Query terms*). Además se puede observar el proceso de obtención de raíces que ha partir de *determine* ha producido *determin*.

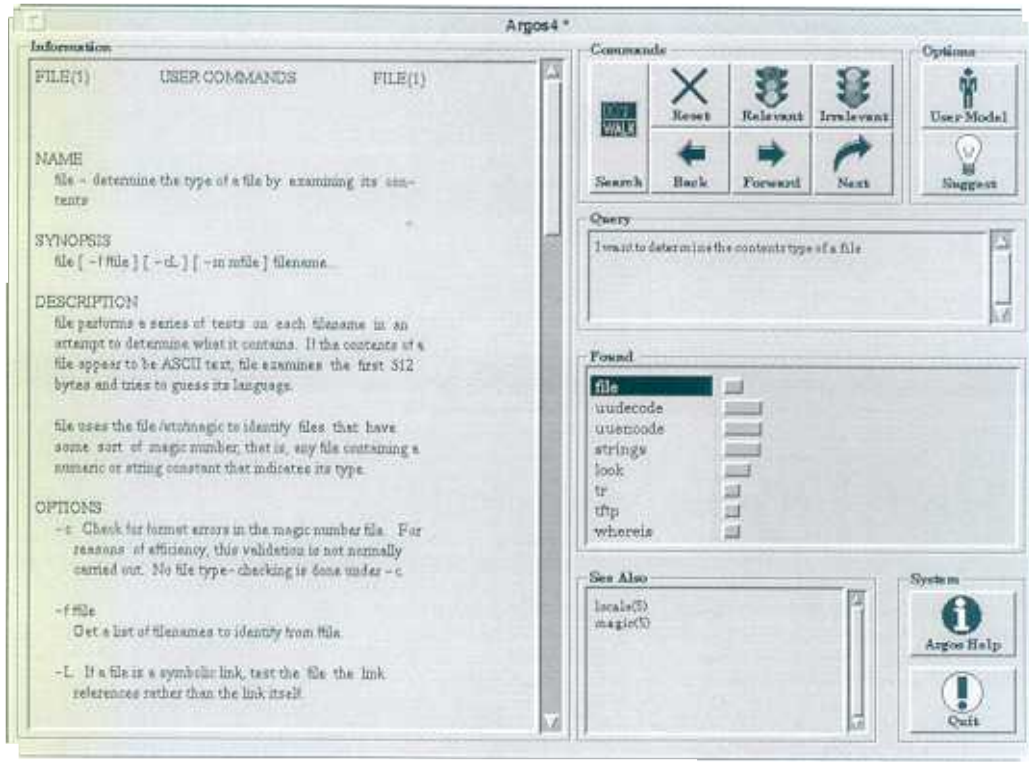


Figura A.4

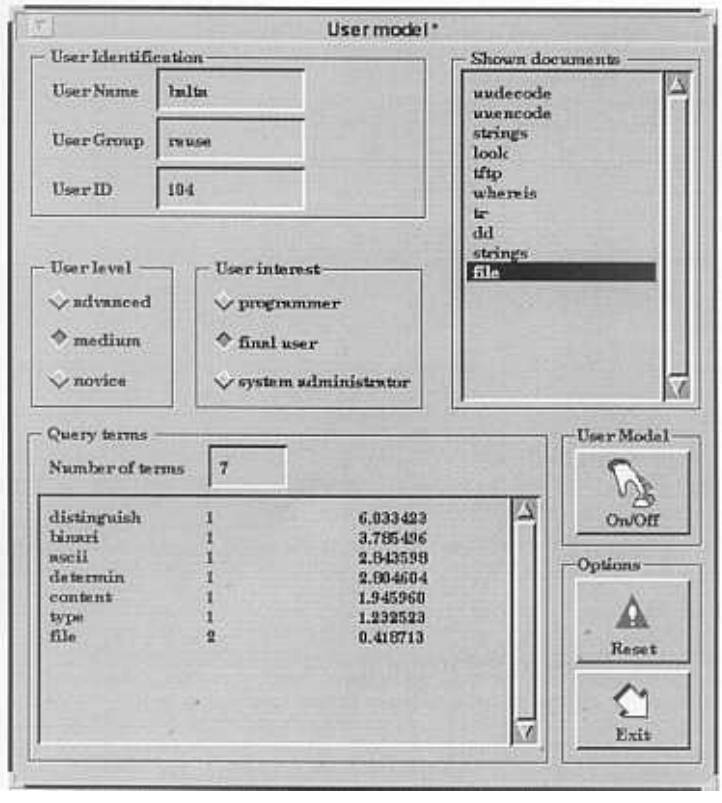


Figura A.5

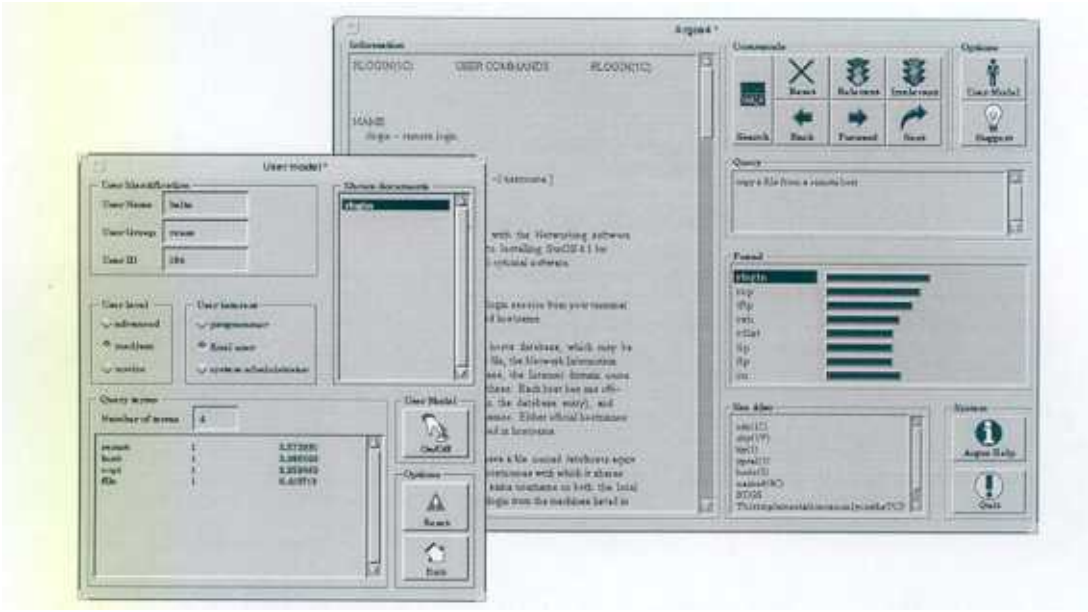


Figura A.6

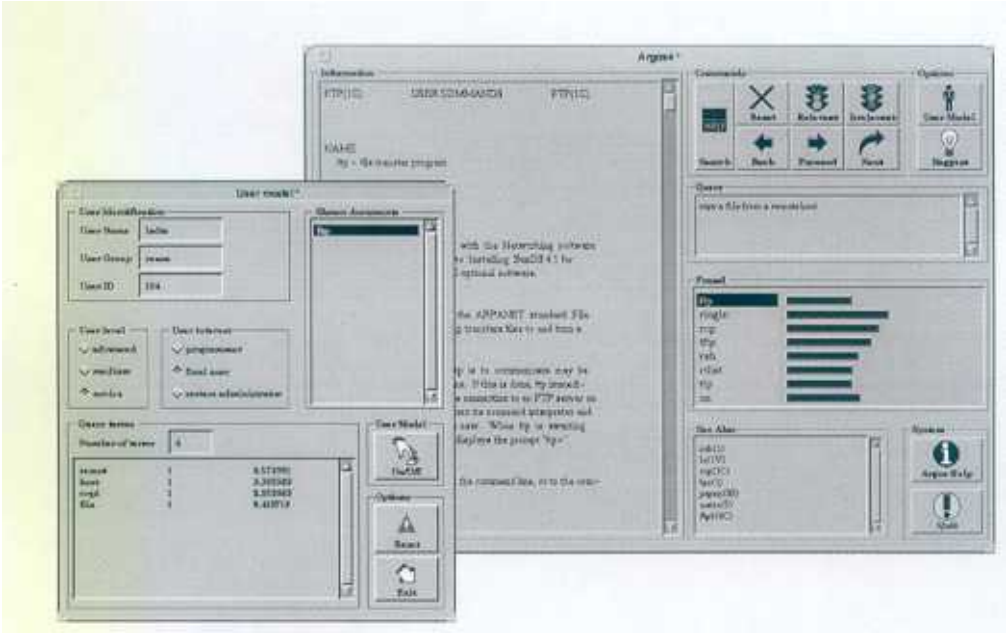
A.1.2 Ejemplo 2

En este ejemplo se muestra la influencia del modelo del usuario en el proceso de presentación de la información. Argos intenta siempre mostrar primero aquellos documentos más relevantes correspondientes al interés del usuario y a su nivel de experiencia.

Para ejemplificar este funcionamiento utilizamos la consulta "copy a file from a remote host" y analizamos los resultados proporcionados por Argos para diferentes contenidos del modelo del usuario. El resultado de realizar esta solicitud produce ocho documentos, de los cuales seis son de nivel de dificultad medio, junto con *ftp* que es de nivel básico y *on* que es de nivel avanzado.

Primero consideramos la situación en la que el contenido del modelo del usuario corresponde a un usuario final con un nivel de experiencia medio (valores de los campos *User level* y *User interest* de *User model* en la figura A.6). Observamos que en la lista de documentos presentados en *Found*, primero aparecen los de dificultad media ordenados por su valor de relevancia, a continuación aparece el documento catalogado como básico y por último el avanzado. El documento más relevante de los de dificultad media, en este caso *rlogin*, se muestra directamente en el campo *Information*.

En la figura A.7 se muestra el resultado de la consulta para un contenido del modelo del usuario correspondiente a un usuario final con poca experiencia (novel). En este caso se presenta directamente en el campo *Information* el documento de *ftp*



Figura

aunque no es el más relevante, ya que es el único que está catalogado como de dificultad básica. En el campo *Found* aparecen después de *ftp* los documentos de dificultad media y finalmente *on* que está clasificado como avanzado.

La figura A.8 muestra como se puede desactivar el modelo del usuario de Argos pulsando el botón *On/Off* del *User model* (el contenido de los paneles *User level* y *User interest* aparece en gris y no es seleccionable). En este caso las reglas de

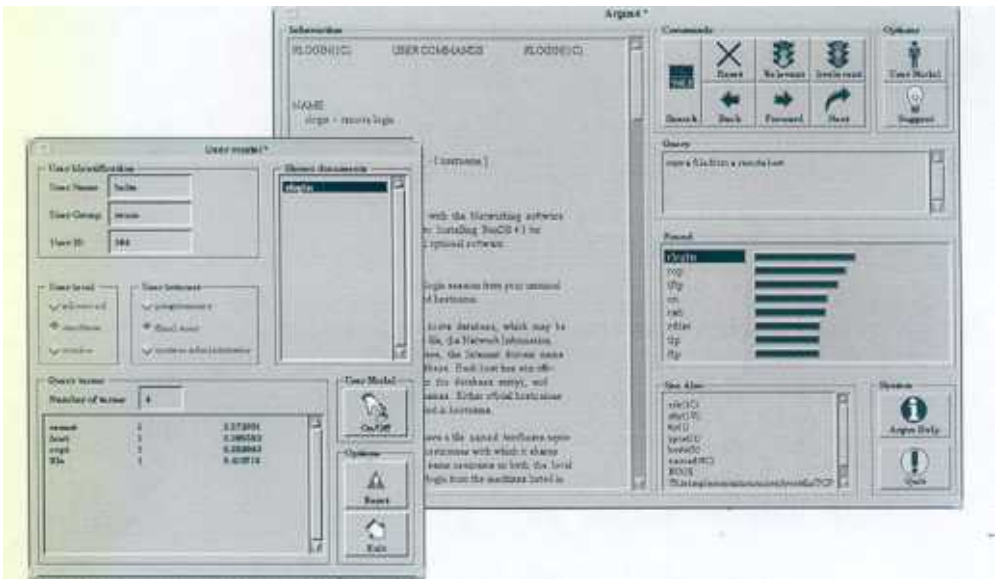


Figura A.8

presentación de información no se aplican y en el panel *Found* se muestran los documentos ordenados directamente por los valores de relevancia obtenidos por el módulo de recuperación de información.

A.1.3 Ejemplo 3.

En el siguiente ejemplo se continua mostrando algunas de las funcionalidad de Argos. Por ejemplo, se muestra el uso del hipertexto estático cuyo fin es el acceso directo a documentos que aparecen en el campo *See Also*. Así mismo se presenta el uso que se hace del modelo del usuario para no repetir documentos previamente mostrados. Además se usa la realimentación positiva para ejemplificar la búsqueda de información relacionada.

Se parte de la consulta utilizada previamente "copy a file from a remote host" con el contenido del MU correspondiente a un usuario final y novel. El resultado de la búsqueda es el mismo que se ha presentado en la figura A.7 y que ya se ha comentado. Destacar que en la consulta sólo se utilizan realmente los cuatro términos que aparecen en el campo *Query terms* del MU.

Una vez examinado el contenido de *ftp* el usuario pulsa el botón *Next* y en el campo *Information* se van presentando en orden los documentos del campo *Found*. Cuando se accede a *tftp* el usuario desea comprobar si en el sistema existen otros documentos además de los recuperados que tengan información relacionada (fig. A.9). Entonces pulsa el botón *Relevant* provocando que los términos más significativos de este documento se usen para complementar la consulta inicial y se realice una nueva búsqueda. Nótese que en el campo *See Also* se muestra un

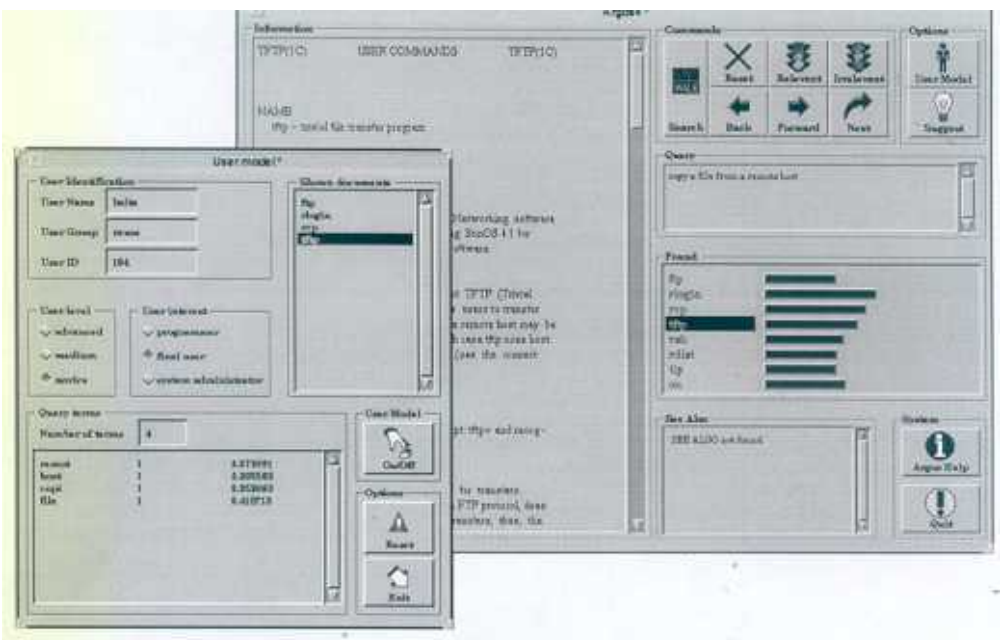


Figura A.9

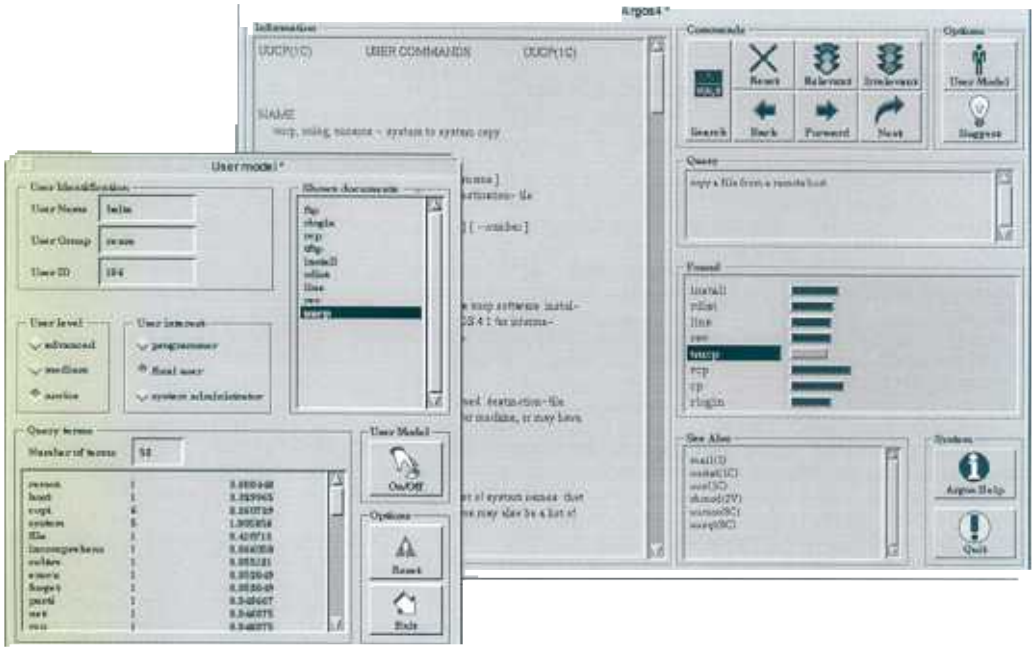


Figura A.10

mensaje de error debido a que en este documento no se ha respetado el formato estándar y esta sección no está presente (a pesar de que, por ejemplo, *ftp* es una orden relacionada).

En la figura A.10 se muestra el resultado de la búsqueda una vez que se ha utilizado el botón *Next* hasta llegar a *uucp* (campo *Found*). La orden *uucp* no aparecía previamente en los documentos recuperados. El campo *Query terms* de *User model* ahora tiene 52 términos, provenientes de la consulta inicial y de los 50 términos más significativos de *tftp*. Obsérvese que los documentos previamente presentados (que aparecen en el campo *Shown documents*) se presentaran más tarde a pesar de tener un valor de similitud superior para evitar fatigar al usuario con información repetida. El usuario ya conoce su contenido y si desea volver a ellos puede utilizar el botón *Back* o accederlos pulsando directamente sobre su nombre con el ratón.

En la figura A.11 se muestra como se ha accedido al documento *chmod* que aparecía en el campo *See Also* correspondiente al documento de *uucp*. Para ello simplemente se ha pulsado con el ratón sobre ese nombre.

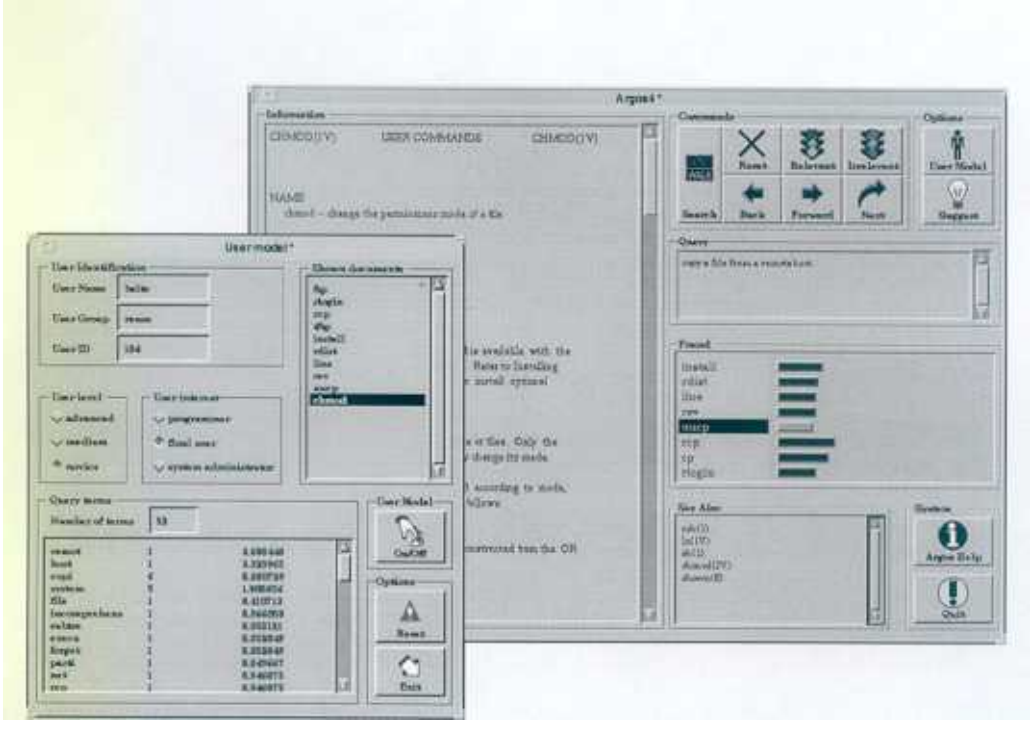


Figura A.1

A.2 Detalles de la base de conocimiento de Aran

Como se destacó en el capítulo 6 la base de conocimiento de Aran consta de dos partes principales:

- El modelo del usuario
- El modelo del dominio

A continuación se incluyen detalles de la codificación en Loom y en Lisp del contenido de estos dos módulos.

A.2.1 Modelo de usuario

En este punto se proporcionan los detalles de implementación de los estereotipos de usuario de Aran. También se incluyen ejemplos de las reglas que se utilizan para la asignación inicial de los estereotipos y para su mantenimiento dinámico posterior (tanto asignación de nuevos estereotipos como desasignación de los que ya no sean adecuados)

Definición de estereotipos

En este apartado se presenta la definición de los estereotipos de usuario y la de las relaciones necesarias para únicamente para este modelado.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Estereotipos (prototipos) de usuario en el dominio del Unix
;;
;; Baltasar Fernandez Manjon 1996
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defconcept User-Prototype
  "concepto primitivo que agrupa los prototipos reconocidos en la
aplicacion"
  is-primitive
  (and unix-thing
       ;; identificador numerico de usuario en el sistema que es
único
       (the has-id number)
       ;; identificador de grupo asignado en /etc/passwd
       (the has-group number)
       ;; identificador  de usuario con el que se entra en el sistema
       (the has-name identifier)
       ;; lista de pares termino peso utilizados en la memoria a
corto plazo
       (exactly 1 term-list)
       ;; lista de ordenes usadas recientemente en el sistema
       (exactly 1 history-list)
       ;; lista de documentos a los que ha accedido el usuario
       (exactly 1 accessed-documents)
        ;; conceptos a los que ha accedido el usuario
       (all accessed-concepts unix-thing)
       ;; ordenes que se presuponen conocidas por la pertenencia a un
prototipo
       ;; se limita su filler a unix-action debido a que todas las
ordenes estan
```

```
        ;; indexadas por las acciones
        (all presupposed-known-commands unix-action)
        ;; ordenes que se presuponen desconocidas por la pertenencia a
un prototipo
        (all presupposed-unknown-commands unix-action)
        ;; conceptos de la representación del dominio que se le
presuponen conocidos
        ;; debido a su pertenencia a un prototipo concreto
        (all presupposed-known-concepts unix-thing)
        ;; conceptos de la representación del dominio que se le
presuponen no conocidos
        ;; debido a su pertenencia a un prototipo concreto
        (all presupposed-unknown-concepts unix-thing)
        )
  )
;; relaciones especificas del modelado de usuario
(defrelation term-list
    characteristics (closed-world))
(defrelation history-list
    characteristics (closed-world))
(defrelation presupposed-known-concepts
    characteristics (closed-world))
(defrelation presupposed-unknown-concepts
    characteristics (closed-world))
(defrelation presupposed-known-commands
    characteristics (closed-world))
(defrelation presupposed-unknown-commands
    characteristics (closed-world))
(defrelation accessed-concepts
    characteristics (closed-world))
(defrelation accessed-documents
    characteristics (closed-world))

;; agrupa a los usuarios de redes y se un usuario se clasifica aqui
si no se puede
;; afirmar nada mas especifico sobre su uso de redes
;; se clasifica aqui si utiliza alguna de las ordenes de redes pero
no se puede aplicar un
;; prototipo mas especifico
;; se declara la particion network-user para evitar que pueda existir
una instancia de mas de uno de las posibles subclases de network-user
(defconcept network-user
    "prototipo que agrupa a los distintos tipos de usuarios de
redes."
  is-primitive
  (and User-Prototype
       (filled-by presupposed-known-concepts network ))
  partitions $network-user$)

(defconcept Network-Expert
  "Un usuario habitual de las redes que domina los conceptos
relacionados"
  is-primitive
  (and Network-User
       (filled-by presupposed-known-commands "ftp" "telnet" "uucp" )
       (filled-by presupposed-known-concepts file-server local-
network protocol ))
  in-partition $network-user$
  )

(defconcept network-novice
  "Un usuario poco familizarizado con la red."
  is-primitive
  (and Network-User
       (filled-by presupposed-unknown-concepts local-network protocol
file-server )
       (filled-by presupposed-unknown-commands "ftp" "uucp")
       (filled-by presupposed-known-commands "mosaic"))
  in-partition $network-user$
  )

;; si el usuario tiene numero 0 entonces tiene derecho de
administracion del sistema
;; y automaticamente se implica que es un usuario experto en la red
```

```
;; en el modo de IR esto implica que las ordenes de administracion se
presenten primero
(defconcept system-manager
    is (and user-prototype
         (filled-by has-id 0))
    implies (and network-expert programmer-user))

;;  informacion sobre llamadas de sistema (parte 2 del manual) que
este activado
;; este prototipo es condicion requerida para mostrar esa informacion
(defconcept Programmer-User
  "prototipo que captura la informacion del usuario como programador
usuario que programa en cualquier lenguaje."
  is-primitive
  ( and User-Prototype
     (filled-by presupposed-known-commands "ld")
     (filled-by presupposed-known-concepts programming-language)
      ))

;;informacion sobre funciones de C (parte 3 del manual) que este
activado este
;; prototipo es condicion requerida para mostrar esa informacion
(defconcept C-Programmer
    "usuario que programa en lenguaje C."
  is-primitive
  (and Programmer-User
       (filled-by presupposed-known-commands "cc" "lint")
       (filled-by presupposed-known-concepts c-language)
      ))

(defconcept pascal-Programmer
    "usuario que programa en lenguaje pascal."
  is-primitive
   (and Programmer-User
       (filled-by presupposed-known-commands "pascal")
       )
   )
```

Reglas de asignación y de actualización de estereotipos

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; PRODUCCIONES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; detecta cuando se anade un nuevo prototipo generico y le asigna
los prototipos
;; correspondientes
(defproduction nuevo-prototipo
    :when (:detects (user-prototype ?user))
    :schedule (asign-prototype ?user))

(defaction asign-prototype (?user)
  :filters (:most-specific))

(defmethod asign-prototype (?user)
  :title "Asigna los prototipos que se cumplen para un usuario"
  :situation (user-prototype ?user)
  :response
  ((format t"~% Asignando prototipos ...~%~A~%" (asignar-prototipos
?user))))

;; funcion de asignacion de prototipos a una instancia que hasta este
momento es unicamente
;; una instancia de user-prototype
;; prueba cuales de los subconceptos de user-prototype son asignables
a esa instancia
(defun asignar-prototipos (instancia)
             (let ((prototipos-asignables  nil)
               (prototipos (get-subconcepts (find-concept 'user-
```

```
prototype)))
                (historico (get-value (find-instance instancia) (find-
relation 'history-list))))
                (format t "~% el historico es~% ~A " historico )
                ;; prototipo asignable si la interseccion de las
listas no es vacia
                (setq prototipos-asignables
              (mapcar #'(lambda (prot)
                      (when (intersection
                          historico
                          (get-role-values
                           (find-concept prot) (find-relation
'presupposed-known-commands))
                          :test #'string=)
                        prot))
                    prototipos))
               ;; obtengo los nombres de los prototipos asignables y
elimino los elementos vacios
                (setq prototipos-asignables
              (mapcar
               #'object-name
              (delete-if #'null prototipos-asignables))) ;; elimino
los nil
                (format t "~% los prototipos asignables son~% ~A"
prototipos-asignables )
                ;; asignacion efectiva de los prototipos a la
instancia (si hay alguno que asignar)
                ;; tellm es una macro por eso hay que hacer el eval
con la sustitucion de instancia
                ;; y la expansion ,@ de la lista

              (eval `(tellm (about ,instancia ,@prototipos-
asignables)))))


;; modificacion de la informacion relacionada con la relacion
unknown-commands
;; se utilizara para desactivar prototipos
;; cuando solicite ver los ejemplos de uso de una determinada orden

(defproduction modificacion-prototipo
    :when (:detects (presupposed-unknown-commands  ?user ?commands))
    :schedule (retract-prototype ?user))

(defaction retract-prototype (?user)
  :filters (:most-specific))

(defmethod retract-prototype (?user)
  :title "Desasigna los prototipos que ya no se cumplen para un
usuario"
  :situation (user-prototype ?user)
  :response
  ((format t"~% Asignando prototipos ...~%~A~%" (desasignar-
prototipos ?user))))

;; prueba cuales de los subconceptos de user-prototype asignados
actualmente ya
;; no son asignables a esa instancia
(defun desasignar-prototipos (instancia)
  (let ((prototipos-asignados (get-types instancia :direct-p t)) ;;
solo subconceptos directos
     (prototipos-a-desasignar nil))
     (format t "~% los prototipos asignados son~% ~A" prototipos-
asignados )
    ;; si tiene asignado un prototipo diferente de user-prototype
    (when (not (member (find-concept 'user-prototype) prototipos-
asignados))
      ;; si la interseccion entre conocidos en el prototipo  y
desconocidos en la instancia
      ;; no es nula entonces se desasigna ese prototipo
      (setq prototipos-a-desasignar
     (mapcar #'(lambda (prot)
             (when (intersection
                (get-role-values
```

```
                (find-instance instancia) (find-relation
'presupposed-unknown-commands))
                (get-role-values
                 (find-concept prot) (find-relation 'presupposed-
known-commands))
                :test #'string=)
               prot))
          prototipos-asignados)))
      ;; obtengo los nombres de los prototipos asignables y elimino
los elementos vacios
    (setq prototipos-a-desasignar
      (mapcar
       #'object-name
       (delete-if #'null prototipos-a-desasignar))) ;; elimino los
nil
    ;; si existe algun prototipo a desasignar entonces se desasigna
    (when prototipos-a-desasignar
      (eval `(forgetm (about ,instancia ,@prototipos-a-
desasignar))))))
```

A.2.2 Modelo del dominio

Se presenta a continuación el conjunto de conceptos así como ejemplos de las instancias que forman el modelo del dominio utilizado en Aran. Parte del contenido de este modelo de dominio se ha reutilizado de otros sistemas previos como se ha destacado en los epígrafes 6.3.2 y en el 6.8.

A.2.2.1 Conceptos: objetos y acciones de Unix

En este punto se presentan los conceptos del modelo del dominio. Aunque conceptualmente no se diferencian objetos u entidades de acciones y de hecho estan contemplados en una única base de conocimiento se presentan agrupados en:

- Objetos (entidades)
- Acciones: clasificación temática
- Acciones: clasificación semántica

Objetos

Seguidamente se incluye la definición de la base de conocimiento y la de los objetos o entidades identificados en el dominio.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;  Conocimiento terminologico sobre UNIX
;;
;;  Baltasar Fernandez Manjon 1996
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Crea una base de conocimiento llamada unix-loom
(loom:use-loom "COMMON-LISP-USER" :dont-create-knowledge-base-p t)

(change-kb (cond ((find-kb 'unix-loom) 'unix-loom)
          (t (defkb unix-loom nil))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
```

```
;; Definicion de conceptos del dominio Unix
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Unix-thing es el concepto raiz de la representacion terminologica
sobre unix
;; De este concepto se colgaran todas las definiciones de
;; acciones, objetos (entidades) y prototipos de usuario
(defconcept unix-thing
  "unix-thing es la raiz de la representacion terminologica.sobre
unix"
  is-primitive thing)

;; a partir del concepto unix-entity se organizaran las definiciones
de objetos
;; reconcidos en el dominio que no son ni acciones ni prototipos de
usuario
(defconcept unix-entity
  "concepto generico a partir del cual se organizan los objetos del
dominio
que no son acciones o prototipos de usuario"
  is-primitive unix-thing)

;;
;; a partir del concepto unix-action se organizaran las definiciones
de
;; las acciones reconocidas en ese dominio
(defconcept unix-action
    "concepto generico a partir del cual se organizan las acciones
del dominio
(no otro tipo de objetos o prototipos de usuario) para clasificar
comandos.
Muchas de estas acciones no tendran verdadero sentido para
el sistema ya que no se podran diferenciar mediante condiciones
necesarias y suficentes
pero si tienen sentido para el ingeniero de conocimiento y para la
agrupacion
semantica de informacion "
  is-primitive unix-thing)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Entidades del dominio
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defconcept Computer
  "concepto de computadora. Caracterizada por un nombre y un numero "
  is-primitive
  (and unix-entity
       (the has-name identifier)
       (the has-number Number)
       (the has-processor processor)))

(defconcept processor
    is-primitive unix-entity)

 (defrelation has-name
    "correspondencia de un objeto y un nombre unico"
  :characteristics (:single-valued))

(defrelation has-number
    range Number)

(defrelation has-processor
  range string)

(defconcept Identifier
  "un identificador es una cadena"
  is-primitive
  (and unix-entity
       (the has-name string)))
```

```
;; relacion con el sistema real ya que se puede codificar la
condicion de que un usuario
;; existente en el sistema su nombre tiene que estar contenido en el
fichero /etc/passwd
;; para ello se incluye la llamada a una funcion lisp (existe-
usuario) que devuelve un
;; valor distinto de nil si este nombre esta en dicho fichero
;; Este concepto se trata como pseudo primitivo ya que loom no puede
razonar sobre el
;; contenido del predicado lisp
(defconcept user-name
    "identificador (nombre) de usuario existente en el sistema.
Tiene que estar en el fichero /etc/passwd"
  is-primitive identifier
  predicate ((?name)
         (exist-user ?name)))

(defconcept group-name
    "identificador (nombre) de grupo existente en el sistema.
Tiene que estar en el fichero /etc/group"
  is-primitive
  (and identifier))
;; (predicate ?nombre
;;      (existe-grupo ?nombre))

(defconcept addres
    "identificador de la direccion de un usuario"
  is-primitive identifier)

(defconcept file-name
    "identificador de un fichero"
  is-primitive identifier)

(defconcept absolute-file-name
    "identificador de fichero con nombre completo desde el directorio
raiz
por tanto comienza con un caracter /"
  is-primitive file-name)

(defconcept relative-file-name
    "identificador de un fichero con nombre de ruta relativo al
directorio actual"
  is-primitive file-name)

(defconcept path
    "identificador de una ruta en el arbol de directorios"
  is-primitive identifier)

(defconcept absolute-path
    "identificador de una ruta en el arbol de directorios a partir
del directorio raiz /"
  is-primitive path)

(defconcept relative-path
    "identificador de una ruta en el arbol de directorios a partir
del directorio actual"
  is-primitive path)


(defconcept Command
    "comando diciendo solo que tiene nombre"
  is-primitive
  (and unix-entity
       (the has-name string)))


(defconcept Internal-Command
  "comando que no tiene asociado un fichero ejecutable. Es
interpretado por
shell p.e. ls"
  is-primitive Command)

(defconcept External-Command
  "comando que tiene asociado un fichero ejecutable. Se puede anadir
```

```
la restriccion de que el fichero
tiene que ser ejecutable"
  is
  (and command
       (the has-file File)))

(defrelation has-file
  range File)

(defconcept Process
  "un proceso unix caracterizado mediante un identificador, un
propietarios  y una prioridad que son numeros"
  :is-primitive
  (:and unix-entity
     (:all has-ID number)
     (:exactly 1 has-ID)
     (:the has-owner number)
     (:the has-priority number))
  :implies
  (:the has-parent number))

(defrelation has-parent)

(defrelation has-ID
    "relacion entre una entidad y un numero que la identifica
univocamente"
  range number
  :characteristics (:single-valued))

(defrelation has-owner
    range Number
    :characteristics (:single-valued))

(defrelation has-priority
  domain Process
  range Number)

(defconcept System-Process
  "un proceso es del sistema si su propietario tiene numero 0"
  is
  (and Process
       (filled-by has-owner 0)))

(defconcept User-Process
  "un proceso es de usuario si su identificador de propietario es
positivo"
  :is
  (:and Process
       (:the has-owner (:through 1 +INFINITY)))
  :defaults
       (:filled-by has-priority 28))

(defconcept Daemon
    "un demonio es un proceso que normalmente esta siempre
ejecutandose -si esta activado-
p.e  el de impresion"
  is-primitive Process)

;;  "los distintos permisos en Unix: lectura escritura ejecucion o su
ausencia"
;; como instancias de permission se definen readable writable
executable ...
;; previamente definido como conjunto con defset
(defconcept  Permission
     "los distintos permisos en Unix: lectura escritura ejecucion o
su ausencia"
  is
  (and unix-entity
       (one-of READABLE WRITEABLE EXECUTABLE NON-READABLE NON-
WRITEABLE NON-EXECUTABLE)))

;; mensaje:informacion a intercambiar en un proceso de comunicacion
(defconcept message
    "cualquier informacion intercambiada en un proceso de
```

```
comunicacion"
  is-primitive
  (and unix-entity
       (the has-object string)))


(defrelation prerrequisite-concept
    "conceptos que deberian ser conocidos para comprender y asimilar
una determinada
orden u otro concepto")

(defrelation description-text
    "texto descriptivo sobre un objeto")

(defrelation see-also
    "comandos, llamadas del sistema o ficheros del sistema
relacionados con una utilidad")

(defrelation syntax
    "sintaxis de una orden"
  range string)

;; esta relacion se define como la inversa de si misma de forma
;; que si una orden esta relacionada con otra esta tambien lo esta
con la anterior
;; al declararlas las dos como instancias
;; de esta forma se simplifica la codificacion de la información
(defrelation has-related-command
    "ordenes relacionadas"
  inverse has-related-command
  :range unix-action
  )

(defrelation has-example
    "ejemplo de uso de una determinada utilidad")

(defrelation description-text
    "texto descriptivo de una utilidad")

(defrelation  has-related-concept
    "conceptos relacionados con una utilidad")

(defconcept File
  "definicion de fichero tomando parte de la informacion
proporcionada por la
llamada de sistema stat(): numero de inodo, modo del fichero, id del
propietario
grupo del propietario,  "
  is
  (and unix-entity
   (the has-Inode number)
   (all has-fileMode Permission)
   (the has-Owner number)
   (the has-group number)))

(defrelation has-Inode
    "un inodo es un numero que identifica univocamente a un fichero"
  domain File
  range number
  characteristics (single-valued closed-world))

(defrelation has-fileMode
    "modos o permisos de un fichero que controlan las operaciones que
se pueden
realizar con ellos"
  domain File
  range Permission)

 (defrelation has-group
    "relacion que establece pertenecia a grupo"
  characteristics (closed-world))

 (defconcept File-Type
    "distintos tipos de archivos obtenido de la documentacion de ls.
```

```
De aqui se cuelgan
los distintos tipos  posibles de archivos. Tiene unicamente un rol
que es el caracter que
va a indicar cada uno de los posibles tipos"
  is-primitive
  (and unix-entity
       (the entryFile-Type (one-of #\- #\d #\c #\b #\l #\p #\s))))

(defrelation related-code
    "relaciona los objetos del dominio con su codigo")

(defrelation related-file
    "relaciona los objetos del dominio con su codigo, mas
concretamente con nombres de ficheros
existentes en el sistema, p.e. en que fichero se describen los grupos
existentes, o en que fichero se puede averiguar cuales son los
usuarios del sistema"
   is-primitive related-code)

(defrelation related-directory
    "relaciona los objetos del dominio con su codigo, mas
concretamente con
nombres de directorios del sistema, p.e. en que directorio se
encuentran los drivers de los dispositivos"
    is-primitive related-code)

(defrelation entryFile-Type
  domain File-Type
  characteristics (single-valued))

(defconcept NormalFile
  "fichero normal.
   -  entry is a plain file"
  is
  (and File-Type
       (filled-by entryFile-Type #\-)))

(defconcept Directory
  "fichero de tipo directorio, contiene informaci n que relaciona un
nombre de fichero con su inodo
   a su vez puede contener ficheros o directorios
   d  entry is a directory"
  is
  (and File-Type
       (filled-by entryFile-Type #\d)))

(defconcept SpecialCharacterFile
  "c  entry is a character-type special file"
    is
  (and File-Type
       (filled-by entryFile-Type #\c)))

(defconcept SpecialBlockFile
  "b  entry is a block-type special file"
  is
  (and File-Type
       (filled-by entryFile-Type #\b)))

(defconcept SymbolicLinkFile
  "l  entry is a symbolic link;"
  is
  (and File-Type
       (filled-by entryFile-Type #\l)))

(defconcept NamedPipeFile
  "p  entry is a FIFO -also known as named pipe special file;"
  is
  (and File-Type
       (filled-by entryFile-Type #\p)))

(defconcept SocketFile
  "s  entry is an AF_UNIX address family socket"
  is
  (and File-Type
```

```
       (filled-by entryFile-Type #\-)))

(defconcept TextNormalFile
  "fichero normal cuyo contenido es texto"
  is-primitive NormalFile)

(defconcept SourceFile
  "fichero escrito en un lenguaje de programacion"
  is-primitive TextNormalFile)

(defconcept EnglishFile
  "fichero de texto en ingles"
  is-primitive TextNormalFile)

(defconcept PostScriptFile
  "fichero PostScript"
  is-primitive TextNormalFile)

(defconcept BinaryNormalFile
  "fichero binario"
  is-primitive NormalFile)

;;  la definicion de fichero ejecutable la hago en funcion de fichero
y de fichero normal binario
(defconcept ExecutableFile
  is
  (and File BinaryNormalFile
       (filled-by has-fileMode EXECUTABLE)))

(defconcept CompressedFile
  "fichero comprimido p.e. utilizando compress o gzip"
  is-primitive BinaryNormalFile)

(defconcept ObjectFile
  "fichero de codigo objeto producido por un compilador"
  is-primitive BinaryNormalFile)

(defconcept existence
    "concepto que da cuenta de la existencia o no de una determinada
entidad"
    is-primitive unix-thing)

(defconcept exist
    "concepto que da cuenta de la existencia de un determinado
objeto"
  is-primitive existence)

(defconcept non-exist
    "concepto que da cuenta de la no existencia de un determinado
objeto"
  is-primitive existence)

(defconcept User
    "usuario del sistema unix. Caracteristicas: numero, nombre,
grupo, shell y directorio
Los ficheros relacionados son /etc/group y /etc/passwd. Por omision
se le asigna la shell csh"
  is
  (and unix-entity
       (the has-ID Number)
       (the has-name user-name)
       (all has-Group  Number)
       (at-least 1 has-group))
  implies (and
        (the home-directory  absolute-path)
        (the has-shell standard-shell)
        (filled-by  related-file "/etc/group" "/etc/passwd"))
  defaults (filled-by has-shell csh))

(defrelation home-directory
    "directorio de usuario"
  characteristics (single-valued))

(defconcept Super-User
```

```
    "el superusuario tiene numero de identificacion 0. Normalmente su
identificador
en el sistema es root, aunque puede haber otros superusuarios con
distinto nombre"
  is
  (and User
       (filled-by has-ID 0)))

(defconcept Normal-User
  "un usuario normal tiene numero de identificacion distinto de 0 y
positivo"
  is
  (and User
       (the has-ID (through 1 +INFINITY))))

(defconcept group
    "agrupacion de usuarios que permite establecer niveles de
seguridad en la comparticion de
informacion entre distintos usuarios. Se caracteriza mediante un
nombre y un numero"
  is-primitive
  (and unix-entity
       (the has-number number)
       (the has-name group-name))
  implies
       (filled-by related-file "etc/group"))

(defconcept File-System
    "agrupacion logica de archivos. Consiste en un red de archivos y
directorios
estructurados en forma de arbol. Normalmente hay por lo menos un
file-system por dispositivo
??a completar y caracterizar"
  is-primitive unix-entity)

(defconcept file-server
    "computador que proporciona acceso a traves de la red a sus
archivos"
  is-primitive unix-entity)

(defconcept Shell
  "interprete de comandos de Unix. Es el proceso que se ejecuta
cuando un usuario se conecta"
  is-primitive  unix-entity)


;; las shell standard se podrian representar como instancias o como
conceptos (mediante
;; exahustive-partitions), o como las dos opciones
;; se realiza una definicion por extension enumerando todas sus
posibles instancias
(defconcept Standard-Shell
  "interprete de comandos estandar de Unix. Es el proceso que se
ejecuta cuando un usuario se conecta
las shell estandar son la c-shell, tc-shell Korn-shell bourne-shell"
  is
  (and shell
       (one-of csh tcsh ksh sh)))

(defrelation has-shell
    "establece cual es la shell de un determinado usuario"
  characteristics (single-valued))

(defconcept Device
    "componente hardware, p.e. como una impresora o un disco, que
actua como una unidad
 para realizar un funcion especifica se caracteriza por un nombre del
archivo que es el
que usa el sistema para acceder al dispositivo p.e /dev/rst0 para una
cinta de 1/4.
Estos ficheros se encuentran siempre en el subdirectorio /dev tiene
un numero de dispositivo
como parte del nucleo del sistema tiene que existir un device driver
que soporta el
```

```
acceso al periferico"
  is-primitive
  (and unix-entity
       (the has-file File)
       (the has-number Number)))

(defrelation has-file)

;; se crea la partition $network-partition$ con los miembros local-
network y wide-network
;;
(defconcept Network
    "red de comunicacion en general. Se caracteriza por el tipo de
cable, su longitud
y su protocolo tambien tiene un numero que debe ser una parte de la
direccion ip"
  is-primitive
  (and unix-entity
       (the has-number Number)
       (the has-cable CableType)
       (the cable-length CableLengthMeters)
       (the has-protocol Network-Protocol))
  partitions $network-partition$)

(defrelation has-cable)
(defrelation cable-length)
(defrelation has-protocol)

(defconcept Local-Network
  "red de comunicaciones limitada a un area especifica. Hasta 2000
metros de cable"
  is
  (and Network
       (<= cable-length 2000))
  in-partition $network-partition$)

(defconcept Wide-Network
  "red de comunicaciones no limitada a un area"
  is
  (and Network
       (> cable-length 2000))
    in-partition $network-partition$)

;;   "tipo de cable de red"
;; mejor definirlo como one-of o definirlos como instancias ???
(defset CableType
  is
  (one-of 'TWISTED 'THINETHERNET 'FATETHERNET 'OPTICFIBER))

(defconcept CableLengthMeters
  "longitud de cable en metros"
  is
  (and Number (through 0 +INFINITY)))

(defconcept Protocol
  "cunjunto de reglas que explican como se debe interactuar para
intercambiar informacion"
  is-primitive unix-entity)

(defconcept Network-Protocol
    "cunjunto de reglas que explican como debe interactuar el
hardware y el software
 de una red para transmitir informacion"
  is-primitive Protocol)

(defconcept Programming-Language
  "lenguaje de programacion mediante el cual se puede generar
ficheros ejecutables,
normalmente mediante un proceso de compilacion y lincado"
  is-primitive unix-entity)

(defconcept GeneralProgLanguage
  "lenguaje de programacion de proposito general"
  is-primitive Programming-Language)
```

```
(defconcept C-Language
  "lenguaje de programacion de proposito general y de nivel medio.
Especialmente adecuado
para la programacion de sistemas. Es un lenguaje privilegiado en Unix
debido a que la mayor
parte de su codigo esta escrito en este lenguaje"
  is-primitive GeneralProgLanguage)

(defconcept Shell-Language
  "lenguaje de programacion proporcionado por la shell de usuario de
Unix. Existen diversos lenguajes
funcion de la shell utilizada. Normalmente todos ellos ofrecen
estructuras de control de alto nivel"
  is-primitive GeneralProgLanguage)

(defconcept SpecialProgLanguage
  "lenguaje de programacion de proposito especial"
  is-primitive Programming-Language)

(defconcept Lex-Language
  "generador para el desarrollo de analizadores lexicos"
  is-primitive SpecialProgLanguage)

(defconcept Yacc-Language
  "generador para el desarrollo de compiladores"
  is-primitive SpecialProgLanguage)

 (defrelation has-actor
 "relacion definida para tener en cuenta quien lanza la accion"
 characteristics (closed-world))
(defrelation has-action)
(defrelation has-destination)
(defrelation has-source)
(defrelation has-location)
(defrelation has-object
  characteristics (closed-world))
(defrelation has-time)
```

Acciones: clasificación temática

A continuación se presenta la taxonomía de acciones que permite la clasificación temática de las órdenes.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Acciones del dominio
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

 (defconcept File-Manipulation
  "concepto que agrupa todas las funciones a realizar sobre un objeto
de tipo archivo"
  is
  (and unix-action
       (at-least 1 has-object)
       (all has-object (or File File-Type))))

(defconcept DestructiveFM
  "accion destructiva sobre archivo que supone perdida de
informacion"
  is-primitive File-Manipulation)

(defconcept DeletionFM
  " provoca la desaparicion de uno o mas archivos"
  is-primitive DestructiveFM)

(defconcept ReplacementFM
  "reemplazamiento de un archivo: se modifica completamente el
```

```
contenido del archivo al
ser sustituido por otro"
  is-primitive DestructiveFM)

(defconcept ModifyFM
  "se produce algun tipo de cambio en el fichero"
  is-primitive File-Manipulation)

(defconcept ModifyContent
  "se produce una modificacion del contenido de un archivo"
  is-primitive ModifyFM)

(defconcept MofifyProperties
  "se produce alguna variacion en las propiedades del archivo, como
propietario, tamano, nombre, ..."
  is-primitive ModifyFM)

(defconcept CreateFM
  "se crea un nuevo archivo. El destino tiene que ser un directorio"
  is-primitive
  (and File-Manipulation
       (the has-destination Directory)))

(defconcept UseFM
  "uso de un fichero que no implique modificacion, ni creaccion, p.e.
para buscar, visualizar"
  is-primitive File-Manipulation)

(defconcept Process-Manipulation
  "concepto que agrupa todas las funciones a realizar sobre un objeto
de tipo proceso"
  is
  (and unix-action
       (the has-object Process)))

(defconcept DestructivePM
  "accion que supone la desaparicion de un proceso"
  is-primitive Process-Manipulation)

(defconcept ModifyPM
  "accion que supone la modificacion de alguna caracteristica del
proceso"
  is-primitive Process-Manipulation)

(defconcept InformPM
  "accion que suponga la obtencion de informaci n sobre procesos"
  is-primitive Process-Manipulation)

(defconcept Communication
    "concepto que agrupa todas las funciones de comunicacion. En toda
comunicacion se transmite
por lo menos un mensaje"
  is-primitive
  (and unix-action
       (all has-object message)
       (at-least 1 has-object)))

(defconcept User-Communication
  "comunicacion entre usuarios. El destino y la fuente deben ser
usuarios"
  is
  (and Communication
       (the has-destination User)
       (the has-source User)))

;; comunicacion entre usuarios mediante dialogo on-line
(defconcept maintain-dialog
    "comunicacion interactiva entre usuarios"
  is
  ( and User-Communication interactive-action))

(defconcept read-mail
    "lectura de correo por un usuario"
  is-primitive
```

```
  User-Communication)

(defconcept send-mail
    "enviar correo"
  is-primitive
  User-Communication)

(defconcept assign-key
    "asignar clave para correo secreto"
    is-primitive
  User-Communication)

(defconcept read-standard-mail
    "lectura de correo normal"
    is-primitive
  read-mail)

(defconcept read-secret-mail
    "lectura de correo secreto"
    is-primitive
  read-mail)

(defconcept send-standard-mail
    "envio de correo normal"
    is-primitive
  send-mail)

(defconcept send-secret-mail
    "envio de correo secreto"
    is-primitive
  send-mail)

;; a desarrollar con conexion a computadora, transferencia de
informacion
(defconcept Computer-Communication
  "comunicacion entre computadoras. El destino es una computadora"
  is
  (and Communication
       (the has-destination Computer)))

(defconcept Obtain-Help
    "Obtencion de ayuda"
  is-primitive
  unix-action)

(defconcept documentation-access
  "acceso a documentacion"
  is-primitive
  obtain-help)

(defconcept information-search
  "busqueda de documentacion"
  is-primitive
  obtain-help)

(defconcept command-documentation
    "obtencion de informacion textual sobre un comando"
  is-primitive
  documentation-access )

(defconcept command-short-description
    "obtencion de informacion textual corta sobre un comando"
  is-primitive
  information-search )

(defconcept keyword-search
    "busqueda de ayuda mediante palabras clave"
  is-primitive
  information-search )

(defconcept natural-language-search
    "busqueda de ayuda a partir de peticiones en lenguaje natural"
  is-primitive
  information-search)
```

```
(defconcept text-processing
    "operaciones de proceso sobre texto"
  is-primitive
  unix-action)

(defconcept text-editing
    "operaciones de edicion de texto. Implica su creacion y su
modificacion"
  is-primitive
  text-processing )

(defconcept text-formating
    "operaciones de asignacion de formato a un texto"
    is-primitive
    text-processing )

(defconcept text-filtering
    "operaciones de filtrado de texto"
    is-primitive
    text-processing )

(defconcept security-management
    "operaciones relacionadas con seguridad del sistema o de la
informacion contenida"
  is-primitive
  unix-action)

(defconcept control-password
    "palabra clave que permite realizar una determinada operacion,
como p.e.
la conexion al sistema"
  is-primitive
  security-management)

(defconcept control-access
    "operaciones de restriccion o concesion de permisos de acceso"
  is-primitive
  security-management)

;; se puede continuar caracterizando
;; el objeto sobre el que se actua tiene que ser un dispositivo
;; el subdirectorio relacionado es /etc/dev
(defconcept management-media-device
    "gestion de dispositivos de almacenamiento"
  is-primitive
  unix-action)

(defconcept obtain-info
    "obtencion de informacion de un dispositivo"
  is-primitive
  management-media-device)

(defconcept backups
    "realizazion de copias de seguridad"
  is-primitive
  management-media-device)

(defconcept space
    "informacion respecto al espacio usado o libre de un dispositivo"
  is-primitive
  obtain-info)

(defconcept free-space
    "informacion respecto al espacio  libre de un dispositivo"
  is-primitive
  space)

(defconcept used-space
    "informacion respecto al espacio utilizado de un dispositivo"
  is-primitive
  space)

;; administracion general del sistema
```

```
;; normalmente incluira tareas del superusuario
(defconcept system-administration
    "administracion general del sistema"
  is-primitive
  unix-action)


(defconcept user-management
    "gestion de usuarios del sistema"
  is-primitive
  system-administration)

(defconcept group-management
    "gestion de grupos del sistema"
  is-primitive
  system-administration)

(defconcept system-accounting
    "gestion de estadisticas y contabilidad del sistema"
  is-primitive
  system-administration)

(defconcept file-system-management
    "gestion del file system del sistema"
  is-primitive
  system-administration)

(defconcept create-user
    "creacion de usuarios del sistema"
  is-primitive
  user-management)

(defconcept remove-user
    "eliminacion de usuarios del sistema"
  is-primitive
  user-management)

(defconcept set-user-characteristics
    "establecer o modificar caracteristicas de los usuarios del
sistema"
  is-primitive
  user-management)

(defconcept device-management
    "gestion de dispositivos del file system del sistema"
  is-primitive
  file-system-management)

(defconcept check-repair-data
    "repacion y verificacion de datos del file system del sistema"
  is-primitive
  file-system-management)

;; no hay una unica orden que realice esto, de modo que se podria
incluir
;; algun plan o descripcion del modo de operacion
(defconcept include-new-device
    "gestion de dispositivos del file system del sistema"
  is-primitive
  device-management )

(defconcept remove-device
    "eliminacion de un dispositivo del file system del sistema"
  is-primitive
  device-management )

(defconcept make-partition
    "particiones de los dispositivos del file system del sistema"
  is-primitive
  device-management )

(defconcept Super-User-Action
  "acciones que unicamente puede realizar el superusuario"
  is
```

```
  (and unix-action
       (the has-actor Super-User)))

(defconcept Interactive-action
    "accion realizada en modo interactivo"
  is-primitive unix-action)

(defconcept destructive-action
    "accion que genericamente puede suponer la destruccion de un
objeto o de
informacion"
  is-primitive unix-action)


 (defconcept examples
    "concepto que agrupara los ejemplos almacenados en la base de
datos"
  is-primitive unix-entity)
```

Acciones: clasificación semántica

A continuación se incluye la definición de la taxonomía de acciones que permite la clasificación semántica de las órdenes. Esta información se ha reutilizado (recodificándola en Loom) de la empleada en el Sinix Consultant [Hecking 88]. En el Sinix Consultant se consideran cuatro acciones basicas que son el primer nivel en la jerarquia: *change, inform, construct* y *transfer*, que posteriormente se especializan.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; reutilizacion de la clasificacion de acciones del sinix consultant
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;
;; CHANGE
;;
(defconcept change
    "accion generica que agrupa las acciones de modificacion de
entidades
cambia los objetos del sistema o sus caracteristicas
p.e. modificacion de variables, de objetos, de modo de trabajo o de
sistema y
del estado de existencia de un objeto"
  is-primitive unix-action)

(defconcept change-system-variable
    "modifica una variable de sistema"
  is-primitive change)

(defconcept change-object
    "modifica un objeto del sistema (no una variable)"
  is-primitive  change)


(defconcept change-system-mode
    "modifica el modo del sistema"
  is-primitive change)

;; hay que crear el concepto de existencia o no como en el sistema de
wisconsin
;; ya que asi se puede especificar esta accion
;; se ha anadido existence con exist y non-exist
(defconcept change-exist-status
    "modifica el estado de existencia de un objeto del sistema"
  is-primitive change)

(defconcept rename-object
    "se le cambia el identificador a un objeto del sistema"
  is-primitive change-object)
```

```
(defconcept set-attribute-object
    "se establece o modifica un determinado atributo de un objeto"
  is-primitive change-object)

;; aunque las funciones no se puedan caracterizar completamente (por
eso se definen como
;; primitivas, si que se pueden relacionar las diferentes
clasificaciones, como en este caso
;; la accion de editar un objeto es potencialmente destructiva esto
hace que al decir que
;; una accion es de edicion el sistema la clasifica tambien
automaticamente como destructiva
(defconcept edit-object
    "se edita el contenido de un objeto, lo que implica su posible
modificacion"
  is-primitive change-object
  :implies destructive-action)

(defconcept transform-object
    "se transforma el contenido de un objeto"
  is-primitive change-object)

;; a partir de aqui ya hay que tomar la decision de si las ordenes
concretas se
;; representan mediante conceptos, con lo que se tiene acceso al
razonamiento terminologico
;; que da la clasificacion; si se representan mediante instancias con
lo que se tiene
;; todo el poder que proporciona el procesador de consultas; o bien
si se representan de las
;; dos formas tanto como conceptos como instancias

(defconcept call-system-or-mode
    "operaciones que provocan el cambio de sistema o de modo de
trabajo"
  is-primitive change-system-mode)

(defconcept delete-object
    "operaciones destructivas sobre objetos"
  is-primitive change-exist-status)

;; el identificativo create plantea algun problema con loom
;; lo modifico a create-object
(defconcept create-object
    "operaciones que implican la creacion de nuevos objetos"
  is-primitive change-exist-status)

;;
;; INFORM
;;
(defconcept inform
    "accion generica que agrupa las acciones de obtencion de
informacion
respecto a entidades definidas en el sistema o a relaciones entre
ellas"
  is-primitive unix-action)

(defconcept inform-entities
    "obtencion de informacion respecto a las entidades del sistema,
su contenido,
tamano, tipo etc"
  is-primitive inform)

(defconcept inform-relations
    "obtencion de informacion respecto a relaciones entre entidades
del sistema"
  is-primitive inform)

(defconcept list-content
    "informa sobre el contenido de un determinado objeto"
  is-primitive inform-entities)

(defconcept get-info
    "informa sobre las caracteristicas de un determinado objeto"
```

```
  is-primitive inform-entities)

(defconcept help
    "proporciona informacion de ayuda"
  is-primitive inform-entities)

(defconcept search
    "operaciones de busqueda"
  is-primitive inform-relations)

(defconcept compare
    "operaciones de comparacion"
  is-primitive inform-relations)

;;
;; CONSTRUCT
;;
(defconcept construct
    "accion generica que agrupa las acciones de construccion de
entidades
construccion o calculo de un nuevo objeto o valor a partir de otros
ya existentes
p.e. la de objetos por combinacion o transformacion, o la
construccion de valores"
  is-primitive unix-action)

(defconcept construct-object
    "construccion de un objeto a partir de elementos ya existentes"
  is-primitive construct)

(defconcept construct-value
    "construccion de valores mediante computo o evaluacion"
  is-primitive construct)

(defconcept combine-objects
    "accion de combinacion de elementos para crear otro nuevo"
  is-primitive construct-object)

(defconcept divide-objects
    "accion de division  de elementos para crear otros nuevos"
  is-primitive construct-object)

(defconcept transform-object-construct
    "creacion de elementos mediante transformacion de otros
existentes"
  is-primitive construct-object)

(defconcept compute-value
    "obtencion de un valor mediante computo"
  is-primitive construct-value)

(defconcept evaluate-value
    "obtencion de un valor mediante evaluacion de una expresion"
  is-primitive construct-value)


;;
;; TRANSFER
;;

(defconcept transfer
    "accion generica que agrupa las acciones de transferencias de
objetos a otro destino
o a otro dispositivo
p.e. modificacion de variables, de objetos, de modo de trabajo o de
sistema y
del estado de existencia de un objeto"
  is-primitive unix-action)

(defconcept transfer-input
    "obtencion o transferencia de entrada"
  is-primitive transfer)

(defconcept transfer-output
```

```
    "obtencion o transferencia de salida"
  is-primitive transfer)

(defconcept transfer-storage
    "tranferencia a un medio de almacenamiento"
  is-primitive transfer)

(defconcept transfer-user
    "transferencia entre usuarios"
  is-primitive transfer)

(defconcept type-input
    "obtencion o transferencia de entrada"
  is-primitive transfer-input)

(defconcept print
    "impresion de una salida en un dispositivo"
  is-primitive transfer-output)

(defconcept display
    "visualizacion de una salida en un dispositivo"
  is-primitive transfer-output)

(defconcept write-storage
    "escritura en un medio de almacenamiento"
  is-primitive transfer-storage)

(defconcept move
    "transferencia entre medios de almacenamiento "
  is-primitive transfer-storage)

(defconcept archive
    "transferencia a un medio de almacenamiento permanente y
habitualmente
secundario"
  is-primitive transfer-storage)

(defconcept send
    "envio entre usuarios"
  is-primitive transfer-user)

(defconcept receive
    "recepcion entre usuarios"
  is-primitive transfer-user)

(defconcept talk-user
    "transferencia interactiva entre usuarios"
  is-primitive transfer-user)
```

A.2.2.2 Instancias: órdenes concretas

Seguidamente se presenta la codificación como instancias de tres órdenes concretas del sistema operativo Unix. Estas órdenes son: *chmod, chown* y *chgrp*. Además se proporciona un sencillo ejemplo de como la inclusión de algún tipo de restricción sobre una orden puede provocar su clasificación automática en otra acción.

```
(tell (about i-chmod
          set-attribute-object
          file-manipulation
          (has-name "chmod")
          (syntax  "chmod [ -fR ] mode filename")
          (related-file "/bin/chmod")
          (description-text "permite modificar los permisos de
escritura, lectura o ejecución de un
fichero o directorio. El permiso de ejecucion con un directorio
implica que se puede acceder y listar
ese directorio")
          (prerrequisite-concept file)
          (prerrequisite-concept directory)
          (prerrequisite-concept permission)
```

```
          (prerrequisite-concept group)
          (has-related-concept file-system)
          (has-related-command i-chgrp)
          (see-also i-ls)
          (see-also i-chown)
          (see-also i-sh)
          (see-also i-csh)
          (has-example "La orden, chmod 777 foo, proporciona todos
los permisos a todos los usuarios")
          (has-example "La orden, chmod +x foo, anade el permiso de
ejecucion al fichero")))

;; aunque no se indique chmod como orden relacionada, si aparece
debido a que has-related-command esta
;; definida como relacion inversa de si misma.
(tell (about i-chown
          set-attribute-object
          super-user-action
          (has-name "chown")
          (related-file "/etc/chown")
          (description-text "permite modificar el propietario de un
fichero o directorio. Es una orden que solo puede ejecutar el
superusuario")
          (prerrequisite-concept file)
          (prerrequisite-concept directory)
          (prerrequisite-concept super-user)
          (prerrequisite-concept group)
          (has-related-command i-chgrp)))


(tell (about i-chgrp
          set-attribute-object
           (has-name "chgrp")
          (related-file "/bin/chgrp")
          (description-text "permite modificar el grupo de un fichero
o directorio.")
          (prerrequisite-concept file)
          (prerrequisite-concept directory)
          (prerrequisite-concept permission)
          (prerrequisite-concept group)
          (has-related-command i-group)
          (has-example "La orden, > chmod 777 foo, proporciona todos
los permisos a todos los usuarios ")
          (has-example "La orden, > chmod +x foo, anade el permiso de
ejecucion al fichero")
          ))

;; se define ii-file como una instancia de file exclusivamente para
poder hacer
;; clasificaciones automaticas
;; esto permitiria y  facilitaria la inclusion de informacion sobre
nuevas utilidades
;; aunque no se tenga un conocimiento total de la base de
conocimiento
(tell (about ii-file file))
(tell (about ii-super-user super-user))

;; aqui mv se clasificaria automaticamente tambien como file-
manipulation
(tell (about i-mv
           rename-object
           move
           (has-object ii-file)))
```

BIBLIOGRAFÍA

Allan, J., 1995. *Automatic Hypertext Construction.* PhD Dissertation. Cornell University. USA.

Anderson, J.R., Boyle, F., Corbett, A., Lewis, M., 1990. "Cognitive Modeling and Intelligent Tutoring". En Clancey, J.W. & Soloway E. (Eds) *Artificial Intelligence and Learning Environments*. MIT Press. Cambridge. USA.

Araya, J., 1990. *Interactive Query Formulation and Feedback Experiments in Information Retrieval.* Doctoral Dissertation, TR90-1115, Cornell University, USA.

Arroyo y de Dompablo, M. I., Gallego Martinez, M. A., 1993. "Interface de sistema de ayuda". Proyecto de fin de carrera, E.U.I.T. de Telecomunicación, U.P.M.

Balasubramanian, V., 1993. "State of the Art Review on Hypermedia Issues And Applications". Technical Report, Graduate School of Management, Rutgers University, Newark, New Jersey

Barr, A, Feigenbaum, E.A., 1981. *The Handbook of Artificial Intelligence,* William Kauffman, California.

Barret, E., (Eds), 1989. *The Society of text: Hypertext, Hypermedia and the social construction of information,* MIT Press.

Bauer, M, Biundo, S., Dengler, D., Koehler, J., Paul, G., 1993. "PHI - A Logic-Based Tool for Intelligent Help Systems". En Bajcsi, R. (Ed) *Proceedings of the 13th International Joint Conference on Artificial Intelligence,* Morgan Kaufmann, San Francisco.

Beaumont, I., 1994. "User Modelling in the Interactive Anatomy Tutoring System ANATOM-TUTOR". *User Modelling and User-Adapted Interaction.* Vol 4, no. 1, pp 21-45.

Bevilacqua, Ann F., 1989. "Hypertext: Behind the Hype". *ERIC Digest*, Número: ED308882

Bhansali, S., 1991. *Domain-Based Program Synthesis Using Planning and Derivational Analogy*. Ph. D. Thesis. University of Illinois at Urbana-Champaing.

Blanco, J., Fernández-Manjón, B., Gonzalez-Calero, P., Fernández-Chamizo, C., 1995. "Copernico: una herramienta de ayuda para la construcción de bases de

conocimiento". En Actas de Caepia 95, VI Conferencia de la Asociación Española para la Inteligencia Artificial, pp. 269-278, Alicante, España.

Bonar, J., Cunningham, R., 1988. "Bridge: an intelligent tutor for thinking about programming". En Self, J. (Eds) *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction.* Edit. Chapman and Hall. New York

Borg, K., 1990. "IShell: A visual Unix Shell". In Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems, End User Modifiable Environment, pp. 201-207.

Borgida, A., 1992, "Description Logics are not just for the Flightless-Birds: A New Look at the Utility and Foundations of Description Logics". Tech Report, Dept. of Computer Science, Rutgers University, New Brunswick, NJ.

Bork, A., 1986. *El ordenador en la enseñanza*, Edit. Gustavo Gili, Barcelona.

Bork, A., 1991. "Technology in Education: An Historical Perspective". Chapter 3, pag 71-90, University of California, Irvine.

Bork, A., 1992. "Seven Keywords for Technology and Education". Internal Tech Report, Educational Technology Center, University of California, Irvine.

Bork, A., Antenore, F., 1987. "Precalculus Mathematics Course: Formative Evaluation, Tentative Plan". Internal Tech Report, Educational Technology Center, University of California, Irvine.

Bork, A., Levrat, B., Ibrahim, B. & Laustsen, B., 1992b. "Self-Instructional Software". Report, Educational Technology Center, University of California, Irvine.

Boss, R., 1991. "What Is An Expert System?" ERIC Digest. ED335058 ERIC Clearinghouse on Information Resources, Syracuse, N.Y.

Boulay, B., 1988. "Intelligent Systems for Teaching Programing". In Ercoli, P. & Lewis, R. (Eds), *Artificial Intelligence Tools in Education.* IFIP Working Conference Proceedings, Edit. North Holland, Amsterdam.

Boulay, B., 1992. "Programming Environment for Novices", en (ed.) Frasson, C. & Gauthier, *Intelligent Tutoring Systems, Proceedings of ITS'92*, Springer-Verlag.

Boy, G., 1991. *Intelligent Assistant Systems*. Edit. Academic Press, London.

Boyle, C., Encarnacion, A., 1994. "Metadoc: An Adaptive Hypertext Reading System". *User Modelling and User-Adapted Interaction.* Vol 4, no. 1, pp 1-19.

Brachman, R., 1990. "The Future of Knowledge Representation". In Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90), AAAI Press/MIT Press, Menlo Park, CA, pp.1082-1092.

Brachman, R., Levesque, H., (Eds), 1985. *Readings in Knowledge Representation.* Morgan Kaufmann Publishers, Inc, San Mateo, California.

Brachman, R., McGuinness, D., Patel-Scheneider, P., Resnick, L., Borgida, A., 1991. "Living With CLASSIC: When and How to Use a KL-ONE-Like Language". In Sowa, J. (Ed) *Principles of Semantic Networks: Exploration in the Representation of Knowledge*, pp 401-456, Edit. Morgan Kaufmann, San Mateo, California.

Breuker, J. (Eds), 1990. *Developing Intelligent Help Systems.* Report on the P280 ESPRIT Project EUROHELP. EC, Kopenhagen.

Breuker, J., 1988. "Coaching in help systems". En Self, J. (Eds). *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction.* Edit. Chapman and Hall. New York

Breuker, J., 1992. "Generality Watching: ITS Caught between Science and Engineering", en (ed.) Frasson, C. & Gauthier, *Intelligent Tutoring Systems, Proceedings of ITS'92*, Springer-Verlag.

Brill, D., 1993. *LOOM Reference Manual. Version 2.0.* ISI/University of Southern California. USA.

Buenaga, M., 1996. *Integracion de tecnicas de procesamiento del lenguaje natural para la recuperacion de informacion en bibliotecas de componentes software.* Tesis doctoral, Dept. de Informatica y Automática, Universidad Complutense de Madrid.

Buenaga, M., Fernández-Manjon, B. & Vaquero, A., 1993a. "An On-Line Intelligent Assistant for UNIX", *Proceedings of Teleteaching 93*, Short Contribution, Trondheim, Norway.

Buenaga, M., Fernández-Manjon, B., Fernández-Valmayor, A., 1995. "Information Overload At The Information Age". Betty Collis and Gordon Davies (Eds) *Adults in Innovative Learning Situations*, pp. 17-30, Elsevier Science B.V., Amsterdam.

Buenaga, M., Fernández-Manjón, B., Vaquero, A., 1993b. "Un asistente inteligente para Unix basado en la documentación". Revista ADIE (Asociación para el Desarrollo de la Informática Educativa), nº 2, pag 27-35, España.

Buenaga, M., Fernández-Manjon, Fernández-Valmayor, A., 1994. "Information Overload At The Information Age". Paper presented to IFIP Joint Working Conference, WG 3.3 and WG 3.6, *"Adults in Innovative Learning Situations"*, Nantes, France, October, 27-30, 1994

Bush, V., 1945. "As we may think". The Atlantic Monthly, July 1945.

Cabrera, A., 1995. "Informática Educativa: La Revolución Construccionista". *Revista de Informática y Automática.* AEIA.

Callan, J., Croft, B., 1993. "An Approach to Incorporating CBR Concepts in IR Systems". En Case-Based Reasoning and Information Retrieval. AAAI Press, Menlo Park, Ca , USA, 1993.

Carberry, S., Kobsa, A., 1992. *User Modeling and User-Adapted Interaction*, Tutorial of the Tenth National Conference on Artificial Intelligence (AAAI-92).

Carbonell, J., 1970. "AI in CAI: An Artificial Intelligence approach to Computer Aided Instruction", *IEEE Trans. Man-Machine Sys.*, Vol.11, No.4.

Carroll, J.M., Aaronson, A., 1988. "Learning by doing with simulated intelligent help". *Communications of the ACM*, vol 31, n 9, pp 1064-1079.

Carroll, J.M., McKendree, J., 1987. "Interface design issues for advice-giving expert systems". *Communications of the ACM*, vol 30, n 1, pp 14-31.

Chin, D. N., 1989. "KNOME: Modelling What the User Knows in UC". En A. Kobsa & W. Wahlster (Eds.), *User Models in Dialog Systems*, pp. 74-107, Sprinter-Verlag,

Berlin. (versión revisada en 1994, Departament of Information and Computer Sciences, University of Hawaii)

Chin, D. N., 1993. "Adquiring User Models". *Artifial Intelligence Review*, Vol. 7, No. 3-4, pp 185-197.

Cigarrán, J., Fernández-Manjón, B., Buenaga, M., 1995. *Sistema de recuperación de información con procesamiento de consultas en lenguaje natural en el dominio del sistema operativo unix.* Informe técnico 17/95. Departamento de Informática y Automática. UCM.

Cigarrán, J., Fernández-Manjón, B., Buenaga, M., 1996. *El sistema Argos.* Informe técnico 38/96. Departamento de Informática y Automática. UCM.

Clancey, J.W. & Soloway E. (Eds), 1990. *Artificial Intelligence and Learning Environments.* MIT Press. Cambridge. USA.

Clancey, J.W., 1982. "Tutoring rules for guiding a case method dialogue". En Sleeman, D & Brown, J.S. (Eds), *Intelligent Tutoring Systems*, Academic Press Inc.

Clancey, J.W., 1987. "Methodology for Building an Intelligent Tutoring System". En Kearsley, G. (Eds), *Artificial Intelligence & Instruction.* Edit. Addison-Wesley. Massachusetts. USA.

Coffin, S., 1990. *Unix System V Release 4: The Complete Reference.* McGraw-Hill, USA.

Conklin, J. 1987. "Hypertext: An Introduction and Survey", *IEE Computer*, Vol.20, No.9, pag.17-43.

Cooper, M., 1988. "Interfaces that adapt to the user". En Self, J. (Eds). *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction.* Edit. Chapman and Hall. New York

Costa, E., (Eds) 1992. *New Directions for Intelligent Tutoring Systems.* Vol. 91, NATO ASI Series. Springer-Verlag, Berlin.

Croft, W. B., 1993. "Knowledge-Based and Stadistical Approaches to Text Retrieval". *IEEE Expert*, Vol 8, No. 2.

Curtis, B., 1989. "Cognitive Issues in Reusing Software Artifacts". In *Software Reusing*, Edit. ACM Press.

CYC, 1996. *CYC: Common Sense Knowledge Base.* CYCorp . (Accesible Internet, *URL, http://www.cyc.com*)

Davey, B., Priestley, H., 1990. *Introduction to Lattices and Order.* Cambridge University Press, Cambridge, UK.

de Haan, G., van der Veer, G.C., 1993. "Etag as a Basis for Intelligent Help Systems". En van der Veer, G.C., Tauber, M.J., Bagnara, S. (Eds) *Proceedings ECCE-6. Human-Computer Interaction: Task and Organisation*, pp 271-283. CUD, Rome, Italy.

Dear, B., 1987. "AI and the Authoring Process". *IEEE Expert "Expert systems that teach"*, Vol 2, No 2.

Deitel, H., 1990. *Operating Systems, Second Edition.* Addison-Wesley Publishing Company, Massachusetts.

Devanbu, P., Ballard, B., Brachman, R. & Selfridge, P., 1991. "LaSSIE: A Knowledge-Based Software Information System". *Communications of the ACM*, 34:35-49.

Díaz, A., de Pedro, J., Buenaga, M., 1995. *Diseño e implementación de un sistema de selección de componentes software basado en técnicas de indexación automática de texto*, Informe técnico 95-15, Departamento de Informática y Automática, Universidad Complutense de Madrid.

Díaz, J., Cuezva, F., Ibañez, C., Rodriguez, F., Zaccagnini, J. L. & De Pablo, E., 1992. "FORMIP: Herramienta para la formación de personal en el control de procesos". En *Boletín de ADIE*, nº 8, Noviembre.

Doane, S., Kintsch, W., Polson, P., McNamara, D., 1991. "Producing Unix Commands: What Experts Must Know". Tech Report UIUC-BI-CS-91-20, University of Illinois at Urbana-Champaing.

Downton, A., 1991. *Engineering The Human-Computer Interface*. Edit. Mc-Graw Hill.

Draper, S. W., 1984. "The Nature of Expertise in UNIX". En B. Shakel (Eds) *Proceedings of the First Conference on Human-Computer Interaction* INTERACT'84.

Draper, S. W., 1992. "Gloves for the Mind". En Kommers, P., Jonassen, D.H., Mayes, T. (Eds). *Cognitive Tools for Learning*. Edit. Springer-Verlag, Berlin.

Dubs S., Jones M. 1991. *Student Modeling for ITS*. Advanced Computing and Engieneering Departament, Alberta Research Council.

Duffy, T.M., Mehlenbacher B., Palmer, J., 1989. "The evaluation of online help systems: A conceptual model". En E. Barret, (Eds), *The Society of text: Hypertext, Hypermedia and the social construction of information*, MIT Press.

Duffy, T.M., Palmer, J., Mehlenbacher B., 1992. *On Line Help Design and Evaluation*. Ablex Publishing Corporation. Norwood, New Jersey.

Edelson, D., 1992. "When Should A Cheetah Remind You of a Bat? Reminding in Case-Based Teaching". En *Proceedings of the AAAI-92*, San Jose, California.

Eisenstadt, M., Price, B., Domingue, J., 1993. "Software Visualization as a Pedagogical Tool". *Instructional Science* 21, pag 335-365.

Elsom-Cook, M.T., 1988. "Guided discovery tutoring and bounded user modelling". En Self, J. (Eds). *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*. Edit. Chapman and Hall. New York

Elsom-Cook, M.T., O'Maley, C.E., 1990. "ECAL: Bringing the Gap Between CAL and Intelligent Tutoring Systems", *Computers Educations*, Vol. 15, No 1-3

Ercoli, P. & Lewis, R. (Eds), 1988. *Artificial Intelligence Tools in Education*. IFIP Working Conference Proceedings, Edit. North Holland, Amsterdam.

Etzioni, O., Lesh, N., Segal, B. R., 1992. "Building Softbots for Unix". Technical Report 93-09-01. Departament of Computer Science and Engineering. University of Washington, Seattle, USA.

Etzioni, O., Levy, H. M., Segal, B. R. Thekkath, C.A., 1993. "OS Agents: Using AI Techniques in the Operating System Environment". Technical Report 93-04-04.

Departament of Computer Science and Engineering. University of Washington, Seattle, USA.

Feifer, R., Soclof, M., 1991. "Knoledge-Based Tutoring Systems: Changing the Focus from Learner Modelling to Teaching". En *Proceedings of the International Conference of the Learning Sciences*, Evanston, Illinois.

Feifer, R.G., 1992. "Sherlock: An Intelligent Tutoring System for Teaching People How to Learn". In Kopec, D. & Tomshom, R. B. (Eds),*Artificial Intelligence and Intelligent Tutoring Systems: knowledge-basesd Systems for Learning and Teaching*. Edit. Ellis Horwood. New York.

Ferguson, W., Bareiss, R., Osgood, R., Birnbaum, L., 1991. "ASK Systems: An Approach to Story-Based Teaching". En *Proceedings of the International Conference of the Learning Sciences*, Evanston, Illinois.

Fernández-Castro, I., Verdejo, F., Díaz-Ilarraza, 1992. Architectural and Planning Issues in Intelligent Tutoring Systems. Informe técnico FISS-I-56.1-LSI-92. Departamento de Lenguajes y Sistemas Informáticos.UPV.

Fernández-Chamizo, C., González-Calero, P., Hernández-Yáñez, L., Urech-Baqué A., 1995. "Case-Based Retrieval of Software Components". *Expert Systems With Applications*, Vol. 9, No. 3, pp. 397-405.

Fernández-Manjon, B, Buenaga, M., Fernández-Valmayor, A., 1995a. "IHS: An Engineering Step Toward ITS". WCEE'95 *(IFIP World Conference of Computers in Education)*, Birmingham, UK.

Fernández-Manjon, B., Buenaga, M., 1993. "ARGOS: An On Line Intelligent Assistant". *ED-MEDIA 93 (World Conference on Educational Multimedia and Hypermedia)*, Short Paper, Orlando, USA.

Fernández-Manjon, B., Buenaga, M., 1995b. "Internet como herramienta de trabajo en el campo educativo". *Revista ADIE* (Asociación para el Desarrollo de la Informática Educativa), no 4, vol 1, pp. 14-20, España.

Fernández-Manjon, B., Buenaga, M., Fernández-Valmayor, A., 1994. "De los entornos de ayuda inteligente a los tutores inteligentes: el sistema ARGOS". Actas del*II Congreso Iberoamericano de Informatica na Educacao.* vol 2, pag 253-267. Lisboa, Portugal.

Fernández-Manjon, B., Buenaga, M., Fernández-Valmayor, A., Hernández-Yañez, L., 1995c. "The integration of technologies and the reusing of information in IHS". *ED-MEDIA 95 (World Conference on Educational Multimedia and Hypermedia)*, Short Paper, Graz, Austria.

Fernández-Valmayor, A., 1990: *Diseño de una base de conocimientos dinámica y su aplicación en un entorno educativo*, Tesis doctoral, Editorial de la Universidad Complutense de Madrid.

Fernández-Valmayor, A., 1993. "Informe: IFIP WG Working Conference". *Revista ADIE* (Asociación para el Desarrollo de la Informática Educativa), nº 2, pag 62-64, España.

Fikes, R., Cutkosky, M., Gruber, T., Van Baalen, J., 1991."Knowledge Sharing Technology,Project Overview" Knowledge Systems Laboratory, Stanford University.

Finning, T., 1989. "GUMS, A General User Modelling Shell". En Kobsa, A., Whalster, W., (Eds), *User Modelling in Dialog Systems*. Springer-Verlag

Fischer, G., 1987. "A critic for LISP". En *Proceedings of the International Joint Conference on Artificial Intelligence*, pp 177-184.

Fischer, G., 1991. "Supporting Learning on Demand with Desing Environments". En *Proceedings of the International Conference of the Learning Sciences*, Evanston, Illinois.

Fischer, G., 1996. "Making Learning a Part of Life - Beyond the 'Gift Wrapping' Approach to Technology". NSF Symposium Learning and Intelligent Systems. Departament of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder, USA.

Fox, C., 1992. "Lexical Analysis and Stoplists" en (ed.) Frakes, W., Baeza, R., Information Retrieval: Data Structures and Algorithms, Prentice Hall, London.

Frakes, W. & Baeza, R., 1992. *Information Retrieval: Data Structures and Algorithms*, Prentice Hall, London.

Frakes, W., Pole, T., 1994. "An Empirical Study of Representation Methods for Reusable Software Components". *IEEE Transactions on Software Engieniering*, vol 20, n° 8, pag 617-630.

Frasson, C. & Gauthier (Eds), 1992. *Intelligent Tutoring Systems, Proceedings of ITS'92,*, Edit. Springer-Verlag, Berlin.

Genesereth , M. R., 1982. "The role of plans in intelligent teaching systems". En Sleeman, D & Brown, J.S. (Eds) *Intelligent Tutoring Systems*, Academic Press Inc.

Gil, Y., 1994. "Knowledge Refinement in a Reflective Architecture". *Proceedings of Twelfth National Conference of Artificial Intelligence*, Seattle, WA.

Guindon, R., 1988. "How to Interface to Advisory Systems? User Request Help with a Very Simple Language", (ed.) Soloway, E., Frye D., *Proceedings CHI'88*, ACM Press, New York.

Girardi, M., Ibrahim, B., 1993. "A Software Reuse System based on Natural Language Specifications". *Proceedings of the 5th International Conference on Computing and Information (ICCI'93)*, Sudbury, pp. 507-511.

Girardi, M., Ibrahim, B., 1994. "Automatic Indexing of Software Artifacts". *Proceedings of the 3rd International Conference on Software Reuse*, Rio de Janeiro.

Gonzalez, A., Dankel, D., 1993. *The Engieneering of Knowledge-based Systems*. Prentice-Hall International, New Jersey.

Gonzalez-Calero, P., (en prensa). Un enfoque basado en conocimiento de la sintesis de programa a partir de componentes reutilizables. Tesis doctoral. UCM.

Gonzalez-Calero, P., Fernández-Manjón, B., Fernández-Chamizo, C., 1996. *El sistema Copernico: una herramienta gráfica para la construcción de bases de conocimiento en Loom*. Informe técnico 43/96. Departamento de Informática y Automática. UCM.

Gruber, T., 1990. "The Role of Standard Knowledge Representation for Sharing Kowledge-Based Technology". *Technical Report No. KSL 90-53*. Kowledge System Laboratory. Stanford University. California

Gruber, T., 1992. "A traslation Approach to Portable Ontology Specifications". *Technical Report KSL 92-71*. Knowledge Systems Laboratory, Stanford University, California.

Gruber, T., Olsen G., 1994. "An Ontology for Engineering Mathematics". In Doyle, J., Torasso, P., Sandewall, E., (Eds) *Fourth International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, Bonn, Germany.

Guha, R., Lenat D., 1994. "Enabling Agents to Work Together", *Communications of the ACM*, Vol.37, No.5.

Gutierrez Serrano, J., 1994. *INTZA: un Sistema Tutor Inteligente para Entrenamiento en Entornos Industriales*. Tesis doctoral. Facultad de Informática, Universidad del País Vasco. San Sebastian.

Harmon, P., 1987. "Intelligent Job Aids: How AI Will Change Training in the Next Five Years". En Kearsley, G. (Eds), *Artificial Intelligence & Instruction*. Edit. Addison-Wesley. Massachusetts. USA.

Hartley, J. R. & Pilkington, R., 1988a. "Software Tools for Supporting Learning: Intelligent On-Line Help Systems" In Ercoli, P. & Lewis, R. (Eds), *Artificial Intelligence Tools in Education*. IFIP Working Conference Proceedings, Edit. North Holland, Amsterdam.

Hartley, J. R. & Smith, M. J., 1988b. "Question answering and explanation giving in on-line help systems". En Self, J. (Eds). *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*. Edit. Chapman and Hall. New York

Hayes-Roth, F., & Jacobstein, N., 1994. "The State of Knoledge-Based Systems". *Communications of the ACM*. pp. 27-39, Vol. 37, No 3.

Hecking, M., Kemke, C., Nessen, E., Dengler, D., Gutmann, M., Hector, G., 1988. *The SINIX Consultant - A Progress Report*. Memo Nr. 28. University of Saarlndes, Saarbrücken, Germany.

Heinsohn, J., Kudenko, D., Nebel, B., and Profitlich, H., 1994. "An empirical analysis of terminological representation systems". *Artificial Intelligence*, vol. 68, no. 2, pp. 367-398.

Hernández, L., 1989. *Un entorno para la creación de software educativo. Metodología de diseño y desarrollo*. Tesis doctoral. Universidad Complutense de Madrid.

Hernández, L., Vaquero, A. & Fernández, C., 1990. "Object Oriented Authoring Systems Construction". En Oñate, E., Suárez, B., Owen, D., Schrefler, B., Kroplin, B. & Kleiber, M. (Eds). *Computer Adided Training in Science and Technology*. Proceedings of ICCATST, Barcelona, Spain.

Hollan, J., Hutchins, E., Weitzman, L., 1987. "STEAMER: An Interactive, Inspectable, Simulation-Based Training System". En Kearsley, G. (Eds), *Artificial Intelligence & Instruction*. Edit. Addison-Wesley. Massachusetts. USA.

Höök, K., 1995. "Adapting explanations to the User's Task", SICS Research Report R95008, SICS, Sweden.

Ibrahim, B., 1993. "Au-delà de la convivialité: Les logiciels auto-éducatifs". *Informatique/Informations* (22): 21-23. Switzerland.

Ibrahim, B., 1994. "Pedagogical Value of the World-Wide Web". Tech. Report. Computer Science Departament, University of Geneva, Switzerland.

Jerrams-Smith, J., 1989. "An Attemp to incorporate Expertise about Users into an intelligent interface for Unix". *International Journal of Man-Machine Studies*, vol 31, pp 269-292, september.

Jonassen D. H., 1990a. "Hypermedia-based Intelligent Tutoring System: A Proposal". Tech Report, University of Colorado

Jonassen, D.H. & Grabinger, R.S. 1990b. "Problems and Issues in Designing Hypertext/Hypermedia for Learning". In Jonassen, D.H.& Mandl, H. (Eds). *Designing Hypermedia for Learning*. Edit. Springer-Verlag, Berlin.

Jonassen, D.H. & Mandl, H. (Eds), 1990c. *Designing Hypermedia for Learning*. Edit. Springer-Verlag, Berlin.

Jones, J., Millington, M., 1988. "Modelling Unix Users with an Assuption-based Truth Maintenance System: Some Preliminary Findigs". In Kelleger, G. (Ed) Reason Maintenance Systems and Their Applications, Ellis Horwood, Chichester

Jones, J., Virvou, M., 1991. "User Modelling and Advice Giving in Intelligent Help Systems for Unix". *Information and Software Technology*, vol 33, n° 2, pp 121-133.

Jonhson, V. M., 1993. Investigations into Database Management System Support for Expert System Shells. Tech Report KSL, Kowledge System Laboratory. Stanford University. California.

Jonhson, W. L., 1986. *Intention-Based Diagnosis of Novice Programing Errors*. Edit. Morgan Kaufmann Publishers Inc. Los Altos, California.

Jonhson, W. L., 1994. "Dynamic (Re)Generation of Software Documentation". *Proceedings of the Fourth Systems Reengineering Technology Workshop*, APL Research Center Report RMI-94-003, Johns Hopkins University Applied Physics Laboratory, pp. 57-66.

Kaplan, C., Fenwick, J., Chen, J., 1993. "Adaptive Hypertext Navigation Based On User Goals and Context". En *User Modeling and User-Adapted Interaction*, n 3, pp. 193-220.

Kass, R., 1989. "Student Modelling In Intelligent Tutoring Systems - Implications for User Modelling". En Kobsa, A., Whalster, W., (Eds), *User Modelling in Dialog Systems*. Springer-Verlag

Kass, R., Finin, T., 1991. "General User Modeling: A Facility to Support Intelligent Interaction". In Sullivan, J. & Tyler, S. (Eds) *Intelligent User Interfaces* , Edit. ACM Press, Addison-Wesley, New York.

Kay, J., 1995. "The um Toolkit for Cooperative User Modelling". En *User Modeling and User-Adapted Interaction*, n 4, pp. 149-196.

Kearsley, G. (Eds), 1987a. *Artificial Intelligence & Instruction.* Edit. Addison-Wesley, Massachusetts, USA.

Kearsley, G., 1987b. "Computer-Aided Instruction, Intelligent", en (ed.) Shapiro, S., *Encyclopaedia of Artificial Intelligence*, Edit. Willey, New York, pag.154-159.

Kearsley, G., 1988.*Online Help Systems: Design and Implementation.* Ablex Publishing Corporation, Norwood, New Jersey, USA.

Kearsley, G., 1996. *Explorations in Learning & Instruction: The Theory Into Practice Database.* School of Education and Human Development , George Washington University, Washington. (Accesible Internet, *URL, http://www.gwu.edu/~tip*)

Kemke, C., 1987. "Representation of Domain Knowledge in an Intelligent Help System". In *Proceedings INTERACT'87,* 2nd IFIP Conference on Human-Computer Interaction, Amsterdam: Elsevier Science Publisher, pp 215-220.

Kobsa, A., 1993. "User Modeling: Recent Work, Prospects and Hazards". Bericht Nr. 26/93 (WIS-Bericht 4). Universidad de Konstanz.

Kobsa, A., 1994. *User Modeling and User-Adapted Interaction,* Tutorial of the XIII Summer Courses. Universidad del Pais Vasco. San Sebastian.

Kobsa, A., Whalster, W., (Eds), 1989. *User Modelling in Dialog Systems.* Springer-Verlag.

Kobsa, A., Wolfgaang, P., 1995. "The User Modeling Shell System BGP-MS". En *User Modeling and User-Adapted Interaction,* n. 4, pp. 59-106.

Kok A.J., 1991. "A Review and Synthesis of User Modelling in Intelligent Systems", *The Knowledge Engineering Review*, 6(1).

Kommers, P., Jonassen, D.H., Mayes, T. (Eds), 1992. *Cognitive Tools for Learning.* Edit. Springer-Verlag, Berlin.

Kopec, D. & Tomshom, R. B. (Eds), 1992. *Artificial Intelligence and Intelligent Tutoring Systems: Knowledge-Based Systems for Learning and Teaching.* Edit. Ellis Horwood. New York.

Kopec, D., Brody, M., Shi, C. C., & Wood, C., 1992. "Towards an Intelligent Tutoring System with Aplication to Sexually Transmited Diseases". In Kopec, D. & Tomshom, R. B. (Eds),*Artificial Intelligence and Intelligent Tutoring Systems: knowledge-basesd Systems for Learning and Teaching.* Edit. Ellis Horwood. New York.

Kozma, R., 1992. "Constructing Knowledge with Learning Tool". En Kommers, P., Jonassen, D.H., Mayes, T. (Eds), *Cognitive Tools for Learning.* Edit. Springer-Verlag, Berlin.

Kramer B.M & Mylopoulos J., 1987. "Knowledge Representation", en (ed.) Shapiro, S., *Encyclopaedia of Artificial Intelligence*, Edit. Willey, New York, pag.882-890.

Krueger, C., 1992. "Software Reuse", *ACM Computing Surveys*, Vol.24, No.2

Kuah, B., Shneiderman, B., 1992. "Providing Advisory Notices for Unix Command Users: Design, Implementation, and Empirical Evaluations". Tech Report UMCP-CSD CS-TR-3007. University of Maryland at College Park, USA.

Landow, G., 1990. "Popular Fallacies About Hypertext". In Jonassen, D.H.& Mandl, H. (Eds). *Designing Hypermedia for Learning*. Edit. Springer-Verlag, Berlin.

Lawler, R., Yazdani, M. (Eds), 1987. *Artificial Intelligence and Education. Vol 1. Learning Envieronments and Tutoring Systems.* Edit. Ablex Publishing. New Jersey.

Lebowitz, M., 1987. "Memory Organization Packets", en (ed.) Shapiro, S., *Encyclopaedia of Artificial Intelligence*, Edit. Willey, New York, pag.591-592.

Leffler, S., McKusick, M., Karels, M., Quaterman, J., 1989. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Edit. Addison-Wesley, Massachusetts, USA.

Legget, J., Schnase, J., & Kacmar, C., 1990. "Hypertext for Learning". En Jonassen, D.H. & Mandl, H. (Eds). *Designing Hypermedia for Learning*. Edit. Springer-Verlag, Berlin.

Lenat, D., Guha, R., 1990. "CYC: towards programs with Common Sense", *Communications of the ACM*, Vol.33, No.8, pag.30-49.

Lesgold, A., 1987. "Education Applications", en (ed.) Shapiro, S., *Encyclopaedia of Artificial Intelligence*, Edit. Willey, New York, pag.267-272.

Lewis, M., Milson, R. & Anderson, J.R., 1987. "The Teacher´s Apprentice: Designing an Intelligent Authoring System for High Schol Matematics". En Kearsley, G. (Eds). *Artificial Intelligence & Instruction*. Edit. Addison-Wesley. Massachusetts. USA.

Lindig, C., 1995. "Concept-Based Component Retrieval", Proc. IJCAI-95 Workshop on Formal Approaches to the Reuse of Plans, Proofs, and Programs, Montreal.

Linster, M., Musen, M., 1992. "Use of KADS to create a conceptual model of the ONCOCIN Task". *Knowledge Adquisition*, n. 4, pp. 55-87.

ISX, 1991. LOOM User's Guide. ISX Corporation.

Maarek, Y. & Berry, D., 1991a. "An Information Retrieval Approach For Automatically Constructing Software Libraries", *IEEE Trans. Software Engineering*, Vol.17, No.8.

Maarek, Y., 1991b. "Software Library Construction from an IR Perspective", *ACM SIGIR Forum*, Vol.25, No.2, pag.8-19.

MacGregor, R., 1991. "The Evolving Technology of Classification-based Knowledge Representation Systems". En Sowa, J.F. (Ed.) Principles of Semantic Networks, Morgan Kaufmann, CA.

Maddix, F., 1990. *Human-Computer Interaction: Theory and Practice.* Edit. Ellis Horwood. London.

Maes, P., 1994. "Agents that Reduce Work and Information Overload." *Communications of the ACM*, Vol. 37, No. 7.

Maida, A., 1987. "Frame theory", en (ed.) Shapiro, S., *Encyclopaedia of Artificial Intelligence*, Edit. Willey, New York, pag.302-312.

Major, N., Reichgelt, H., 1992. "COCA: A Shell for Intelligent Tutoring Systems". En (ed.) Frasson, C. & Gauthier, *Intelligent Tutoring Systems, Proceedings of ITS'92*, Springer-Verlag.

Mamone, S., 1990. "An Expert System for Unix Problem Resolution". *SIGART Bulletin*, Volume 1, Number 2, July, pag 8-11. ACM Press.

Marchionini, G., 1991. "Psychological Dimensions of User-Computer Interfaces". ERIC Digest ED337203 The Educational Resources Information Center, Syracuse, N.Y.

Marchionini, G., 1995. *Information Seeking in Electronic Environments*. Cambridge University Press.

Maybury, M., (Eds), 1993. *Intelligent Multimedia Interfaces*. AAAI Press/The MIT Press, Menlo Park, California

Mayes, T., Kibby, M., Anderson, T., 1990. "Learning About Learning from Hypertext". In Jonassen, D.H.& Mandl, H. (Eds), *Designing Hypermedia for Learning*. Edit. Springer-Verlag, Berlin.

Mayfield, J., (en prensa). "Two-Level Models of Hypertext". En R. D. Semmel (ed.), *Advances in Software Engineering and Knowledge Engineering*, World Publishing.

Mayfield, J., Nicholas, C., 1993. "SNITCH: Augmenting Hypertext Documents with a Semantic Net". International Journal of Intelligent and Cooperative Information Systems 2(3):335-351.

MCC, 1995. *CYC: Common Sense Knowledge Base*. Microelectronics & Computer Technology Corporation. (Accesible Internet, *URL, http://www.mcc.com/projects/cyc/cyc.html*)

McCalla, G., 1992. "The Central Importance of Student Modelling to Intelligent Tutoring". En Costa, E. (Eds), *New Directions for Intelligent Tutoring Systems*. Vol. 91, pp 107-131, NATO ASI Series, Springer-Verlag, Berlin.

Merril, M., 1987. "An Expert System for Instructional Design". *IEEE Expert* "Expert systems that teach", Vol 2, No 2.

MetaCard, 1995. MetaCard 1.4. MetaCard Corporation (Accesible Internet, *URL, http://www.metacard.com*).

Michel, S., 1991. *Hypercard, Manual de referencia*, Edit McGraw-Hill, Madrid.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., Miller, K., 1990. *Five Papers on WordNet*, Cognitive Science Laboratory, Princeton University, CSL Report 43.

Montero Antón, J., 1993. *Generación de unidades didácticas y presentaciones con HyperCard*. Proyecto de fin de carrera, E.U.I.T. de Telecomunicación, U.P.M., Madrid.

Musen, M. A., 1992. "Dimensions of Knowledge Sharing and Reuse". *Computers and Biomedical Research*, 25, pag. 435-467. USA.

Nessen, E., 1989. "SC-UM: User Modelling in the SINIX Consultant". *Applied Artificial Intelligence*, vol 3, no 1, 33-44.

Nielsen, J., 1990. "The Art of Navigating through Hypertext", *Communications of the ACM*, Vol.33, No.3, pag. 296-310.

O'Shea, T., Self, J., 1985. *Enseñanza y Aprendizaje con Ordenadores.* Anaya Multimedia, Madrid.

O'Sullivan D., McElligott, A., Sutcliffe, R., 1995. "Aumenting the Princeton WordNet with a Domain Specific Ontology". Tech Report, University of Limerick, Ireland.

Oñate, E., Suárez, B., Owen, D., Schrefler, B., Kroplin, B. & Kleiber, M. (Eds), 1990. *Computer Adided Training in Science and Technology.* Proceedings of ICCATST, Barcelona, España.

Orwant, J., 1995. "Heterogeneous Learning in the Doppelganger User Modeling System". *User Modelling and User-Adapted Interaction.* Vol 4, no. 2, pp 107-130.

Papert, S., 1980. *Mindstorms: Children, Computers and Powerful Ideas*, Basic Books Inc., New York.

Papert, S., 1987a. "Computers In Education: Conceptual Issues", en (ed.) Shapiro, S., *Encyclopaedia of Artificial Intelligence*, Edit. Willey, New York

Papert, S., 1987b. "Microworlds: transforming education", en Lawler, R., Yazdani, M. (Eds) *Artificial Intelligence and Education. Vol 1. Learning Envieronments and Tutoring Systems.* Edit. Ablex Publishing. New Jersey.

Patil, R., Fikes, R., Patel-Schneider, P., Mckay, D., Finin, T., Gruber, T. & Neches, R., 1992. "The DARPA Knowledge Sharing Effort: Progress Report". Knowledge Systems Laboratory, Stanford University, USA.

Pilkington, R., 1992. "Question-answering for Intelligent On-line Help: The Process of Intelligent Responding". *Cognitive Science*, 16 (4), 455-491.

Pitts, G., Regian, W., 1992. "A Fuzzy Logic-Based Intelligent Tutoring System (ITS)", *Proceedings IFIP-92*, vol II, North-Holland, Amsterdam.

Pressman, R., 1993. *Ingeniería del Software. Un enfoque práctico*, McGraw-Hill.

Prieto-Díaz, R., 1990. "Domain Analysis: An introduction", *ACM SIGSOFT*, Vol.15, No.2.

Prieto-Diaz, R., 1991. "Implementing Faceted Classification for Software Reuse", *Communications of the ACM*, Vol.34, No.5.

Prieto-Díaz, R., Freeman, P., 1987. Classifying Software for Reusability, *IEEE Software*, Vol.4, No.1

Puerta, A., 1992. "The Study of Models of Intelligent Interfaces". Tech Report KSL 92-65, Knowledge System Laboratory, Stanford University, California, USA.

Puerta, A., 1994. *Model-Based Interface Development*. Tutorial of the XIII Summer Courses. Universidad del Pais Vasco. San Sebastian.

Quilici, A., 1989. "Detecting and Responding to Plan-Oriented Misconceptions". En Kobsa, A., Whalster, W., (Eds), *User Modelling in Dialog Systems*. Springer-Verlag

Quillian, M. R., 1967 "Word Concepts: A Theory and Simulation of Some Basic Semantic Capabilities" en *Readings in Knowledge Representation*, Brachaman, R.J. y Levesque H.J. (Eds). Morgan Kaufmann Publishers Inc., CA. 1985

Rich, C., Waters, R. C., 1986. *Readings in Artificial Intelligence and Software Engineering*. Morgan Kaufmann Publishers, Inc., Los Altos, California.

Rich, E., Knight, K., 1994. *Inteligencia Artificial* Edit. McGraw-Hill, Inc., 2ª Edición, Madrid.

Rich, E., 1989. "Stereotypes and User Modelling". En Kobsa, A., Whalster, W., (Eds), *User Modelling in Dialog Systems*. Springer-Verlag

Riesbeck C.K., Schank R.C., 1989. *Inside Case-Based Reasoning*, Erlbaum, Hillsdale, NJ.

Rojas Guzman, G., 1991. *Un sistema tutorial inteligente aplicado a la enseñanza de la geometría*. Tesis doctoral. Universidad Politécnica de Madrid. Madrid.

Rolston, D.W., 1988. *Principles of Artifical Intelligence and Expert Systems Development*. Edit. Mc-Graw Hill, New York.

Salton, G. & McGill, M.J., 1983. *Introduction to Modern Information Retrieval*, McGraw-Hill, New York.

Salton, G., 1986. "Another Look at Automatic Text-Retrieval Systems", *Communications of the ACM*, Vol.29, No.7, pag.648-656.

Salton, G., Buckley, C., 1988. "Improving Retrieval Performance by Relevance Feedback". Technical Report TR88-898, Departament of Computer Science, Cornell University, USA.

Salton, G., Buckley, C., 1989. "On the Automatic Generation of Content Links in Hypertext ". Technical Report TR89-993, Departament of Computer Science, Cornell University, USA.

Santesmases, J., Solé, J., Vaquero, A., Guillén, J., 1969. "Realización española de un sistema de enseñanza automática". *Revista de Automática*, 3-4.

Scardamalia, M. & Bereiter, C., 1993. "Technologies for Knowledge-Building Discourse". En *Proceedings of the International Conference of the Learning Sciences*, Evanston, Illinois.

Scardamalia, M., 1991. "An Architecture for the Social Construction of Knowledge". En *Communications of the ACM*, pp 37-41, Vol. 36, n 5.

Schank R., Cleary C., 1994. *Engines for Education*, Lawrence Erlbaum Ass., New Haven. . (Accesible Internet, *URL, http://www.* ils.nwu.edu)

Schank, R., 1991. "Where's the AI?". *AI Magazine*, Winter, pag 38-49.

Self, J. (Eds), 1988a. *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction*. Edit. Chapman and Hall. New York

Self, J., 1988b. "Student Models: What Use Are They?" In Ercoli, P. & Lewis, R. (Eds), *Artificial Intelligence Tools in Education.* IFIP Working Conference Proceedings, Edit. North Holland, Amsterdam.

Self, J., 1990a. "Bypassing the intractable problem of student modelling", in C. Frasson and G. Gauthier (eds.), *Intelligent Tutoring Systems: at the Crossroads of Artificial Intelligence and Education*, 107-23, Norwood, N.J.: Ablex.

Self, J., 1990b. "Theoretical foundations for intelligent tutoring systems" *Journal of Artificial Intelligence in Education*, 1(4), 3-14.

Self, J., 1994. "The role of student models in learning environments". *Transactions of the Institute of Electronics, Information and Commnication Engineers* E77-D(1), 3-8.

Selker, T., 1992. "A Framework for Proactive Interactive Adaptative Computer Help". PhD Thesis, University of New York. (También como Tech Report RC 17597, T.J. Watson Research Center, IBM, NY.)

Selker, T., 1994. "Coach: A Teaching Agent that Learns". *Communications of the ACM*, Vol.37, No.7, pag.92-99.

Seoane, J., 1993. "Cibernáutica". *Revista Informática y Automática* (AEIA). Vol 26 Num. 4. Madrid.

Shneiderman, B., 1989b. "Reflections on Authoring, Editing,and Managing Hypertext". En E. Barret, (Eds), *The Society of text: Hypertext, Hypermedia and the social construction of information*, MIT Press.

Shneiderman, B., Kearsley, G., 1989a. *Hypertext Hands-On!: An Introduction to a New Way of Organizing and Accessing Information*. Addison-Wesley Publishing Company, Massachusetts.

Sleeman, D & Brown, J.S. (Eds), 1982. *Intelligent Tutoring Systems*, Academic Press Inc.

Smith, J & Weiss, S., 1988. "Hypertext", *Communications of the ACM*, Vol.31, No.7, pag.816-819.

So, B., Travis, L., 1994. "A Step Toward an Intelligent UNIX Help System: Knowledge Representation of UNIX Utilities". Informe técnico 1230/94. Computer Sciences Department. University of Wisconsin-Madison. USA.

Sokolnicki, T., 1991. "Towards knowledge-based tutors: a survey and appraisal of Intelligent Tutoring Systems". *The Knoledge Engineering Review*, Vol 6:2, 59-95.

Sowa, J. (Ed) *Principles of Semantic Networks: Exploration in the Representation of Knowledge*. Edit. Morgan Kaufmann, San Mateo, California.

Speel, P., 1995. *Selecting Knowledge Representation Systems*. Doctoral Dissertation, Twente University, Neederland.

Sullivan, J. & Tyler, S. (Eds), 1991. *Intelligent User Interfaces* , Edit. ACM Press, Addison-Wesley, New York.

Sun, 1989. *SunOs Reference Manual v.4.1.* Sun Microsystems Europe.

Sun, 1992. *AnswerBook, Solaris X.X* CD-ROM SMCC Versión A. Sun Microsystems Europe.

Sutcliffe, A., Old, A., 1987. "Do User Know They Have User Models? Some Experiences in the Practice of User Modelling". Bullinger, H., Shackel, B., (Eds) *Proceedings Interact'87*, Elsevier Science Publishers B. V., pp 35-41.

Tansley, D., Hayball, C., 1993. *Knowledge-Based Systems Analysis and Design. A KADS Developer's Handbook*. Prentice-Hall, New York.

Tattersall, C., 1992. "A New Arquitecture for Intelligent Help Systems". En (ed.) Frasson, C. & Gauthier, *Intelligent Tutoring Systems, Proceedings of ITS'92*, Springer-Verlag.

tcsh, 1991. Pagina manual UNIX, tcsh - C Shell With File Name Completion and Command Line Editing.

Thimbleby, H., 1992. "Heuristics for Cognitive Tools". En Kommers, P., Jonassen, D.H., Mayes, T. (Eds), *Cognitive Tools for Learning*. Edit. Springer-Verlag, Berlin.

TMC 91. *WAIStation reference manual v.0.57*, Thinking Machines Corporation, Cambridge, Massachusetts.

Tyler, S., Schlossberg, J., Gargan, R., Cook, L., Sullivan, J. 1991"An Intelligent Interface Architecture for Adaptative Interaction". En Sullivan, J. & Tyler, S. (Eds) *Intelligent User Interfaces* , Edit. ACM Press, Addison-Wesley, New York.

Vaquero Sanchez, A., Fernández Chamizo, C. 1987. *La informática aplicada a la enseñanza*, Edit. Eudema, Madrid.

Vaquero, A., Fernández Chamizo, C., Sanchez, J.M., Troya, J.M., Hernández, L., 1986. "SIETE: Sistema Informatizado en Español para el desarrollo de Temas de Enseñanza", *Revista de la Real Academia de Ciencias Exactas, Físicas y Naturales*, tomo LXXX, pag. 473-476.

Verdejo, F., 1992. "A Framework fo Instructional Planning adn Discourse Modelling in Intelligent Tutoring Systems". En Costa, E. (Eds), *New Directions for Intelligent Tutoring Systems*. Vol. 91, pp 146-170, NATO ASI Series, Springer-Verlag, Berlin.

Verdejo, F., 1995. "An experimental project for network-mediated professional specialization: distance learning scenario design and support system". Betty Collis and Gordon Davies (Eds) *Adults in Innovative Learning Situations*, pp. 81-89, Elsevier Science B.V., Amsterdam.

Wan, D., Johnson, P.M., 1994. "Experiences with CLARE: a Computer-Supported Collaborative Learning Environment". *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, Chapel Hill, North Carolina, USA.

Wang, H. & Kusniruk, A., 1992. "The UNIX Tutor", in (ed.) Frasson, C. & Gauthier, *Intelligent Tutoring Systems, Proceedings of ITS'92*, Springer-Verlag.

Wasson B., Akselsen S., 1992. "An Overview of On-line Assistance: from On-line Documentation to Intelligent Help and Training". *The Knowledge Engineering Review*, pp 289-322, vol 7(4).

Weida, R. A., 1991. "Kowledge Representation and Reasoning with Definitional Taxonomies". Technical Report CUCS-047-91. Departament of Computer Science. Columbia University. USA.

Weida, R. A., 1995. "Closing the Terminology". *1995 International Workshop on Description Logics*. Columbia University. USA.

Wenger, E., 1987. *Artificial Intelligence and Tutoring Systems*, Morgan Kaufman Publishers Inc., Los Altos, CA.

Whalster, W., Kobsa, A., (Eds), 1989. "User Models in Dialog Systems". En Kobsa, A., Whalster, W., (Eds), *User Modelling in Dialog Systems*. Springer-Verlag.

Wielinga, B., Schreiber, A., Breuker, J., 1992. "KADS: A Modelling Approach to Knowledge Engieering". *Knowledge Adquisition*, n. 4, pp. 5-53.

Wilensky, R., Arens Y., Chin D., 1984. "Talking to Unix in English: an overview of UC", *Communications of the ACM*, Vol. 27, No.6.

Wilensky, R., Chin, D. N., et al., 1989. "The Berkeley UNIX Consultant Project". Informe técnico UCB//CSD-89-520. University of California, Berkeley. USA.

Winkels, R., 1992a. *Explorations in Intelligent Tutoring and Help*. IOS Press, Amsterdam.

Winkels, R., Breuker, J., 1992b. "What's in an ITS?. A Functional Decomposition. En Costa, E. (Eds), *New Directions for Intelligent Tutoring Systems*. Vol. 91, pp 57-68, NATO ASI Series, Springer-Verlag, Berlin.

Winkels, R., Breuker, J., den Haan, N., 1991. "Principles and Practice of Knowledge Representation in EUROHELP". En En *Proceedings of the International Conference of the Learning Sciences*, Evanston, Illinois.

Winston, P & Horn, B., 1984. *LISP*. Addison Wesley Publishing Company, Reading MA.

Wood, M. & Sommerville, I., 1988. "An Information Retrieval System for Software Components", *ACM SIGIR Forum*, Vol.22, Nos. 3/4, pag.11-27.

Woolf, B., 1987a. "Theoretical Frontiers in Building a Machine Tutor". En Kearsley, G. (Eds), *Artificial Intelligence & Instruction*. Edit. Addison-Wesley. Massachusetts. USA.

Woolf, B., 1992. "Hypermedia in Education and Training". In Kopec, D. & Tomshom, R. B. (Eds),*Artificial Intelligence and Intelligent Tutoring Systems: knowledge-basesd Systems for Learning and Teaching*. Edit. Ellis Horwood. New York.

Woolf, B., Blegen, D., Jansen, J., 1987b. "Teaching a Complex Industrial Process". en Lawler, R., Yazdani, M. (Eds) *Artificial Intelligence and Education. Vol 1. Learning Envieronments and Tutoring Systems*. Edit. Ablex Publishing. New Jersey.

Woolf, B., Cunningham, P., 1987c. "Multiple Knowledge Sources in Intelligent Teaching Systems". *IEEE Expert "Expert systems that teach"*, Vol 2, No 2.

Wright, J.R., Weixelbaum, G. T., *et al*, 1993. "A Knowledge-Based Configurator that Supports Sales, Engeneering, and Manufacturing at AT&T Network Systems". *AI Magazine*, 13(3), pp 69-80.

Yazdani, M., 1987. "Intelligent Tutoring Systems: An Overview", en Lawler, R., Yazdani, M. (Eds) *Artificial Intelligence and Education. Vol 1. Learning Envieronments and Tutoring Systems*. Edit. Ablex Publishing. New Jersey.