Ingeniería del Conocimiento. De la Extracción al Modelado de Conocimiento

J. T. Palma, E. Paniagua, F. Martín y R. Marín
Dpto. Ingeniería de la Información y las Comunicaciones
Universidad de Murcia
Facultad de Informática. Campus de Espinardo
{jpalma,paniagua,fmartin,roque}@dif.um.es

Resumen

En el presente trabajo se intentará dar una visión general del estado actual en el que se encuentra la disciplina Ingeniería del Conocimiento. Para comprender la situación actual de esta disciplina es imprescindible un análisis de su evolución histórica, analizando cada uno de los paradigmas en los que se ha ido basando. Este análisis nos permitirá analizar las causas que dieron lugar a la crisis que se produjo en esta disciplina y cuáles fueron las medidas que se articularon para salvarla. Se concluirá el trabajo con una revisión de las metodologías más utilizadas así como análisis del presente y futuro de la Ingeniería del Conocimiento

Palabras claves: Ingeniería del Conocimiento, metodologías, modelado de conocimiento.

1 ¿Qué es la Ingeniería del Conocimiento (IC)?

Pongámonos en la situación de un profano en esta materia e intentemos resolver esta pregunta. Lo primero que haríamos sería consultar el Diccionario de la Real Academia para buscar el significado de las palabras *ingeniería* y *conocimiento* y obtendríamos lo siguiente:

Ingeniería:

1. f. Conjunto de conocimientos y técnicas que permiten aplicar el saber científico a la utilización de la materia y de las fuentes de energía.
2. Profesión y ejercicio del ingeniero.

Conocimiento:

1. m. Acción y efecto de conocer.
2. Entendimiento, inteligencia, razón natural. ...

Atendiendo a estas definiciones podríamos establecer como definición de Ingeniería del Conocimiento "*Conjunto de conocimientos y técnicas que permiten aplicar el saber científico a la utilización del conocimiento (entendimiento, inteligencia o razón natural)*". Evidentemente, esta definición no nos aclararía nada acerca de nuestro objetivo inicial. Para seguir en nuestro empeño no nos queda otra opción que consultar a otras personas, que casi con toda seguridad nos dirigirían hacia el campo de los ordenadores y la informática (la asociación es bastante simple conocimiento → sociedad del conocimiento → nuevas tecnologías → informática). Si empezamos a consultar bibliografía relativa a la informática en busca de nuestro objetivo nos encontraremos con el término *ingeniería del software* (IS en adelante) y junto a él nos podemos encontrar la siguiente definición:

La aplicación de una aproximación sistemática, disciplinada y cuantificable al desarrollo, funcionamiento y mantenimiento del software; en otras palabras, la aplicación de la ingeniería al software [IEEE, 1999].

Esta definición ya nos puede ayudar algo, ya que nos da la visión de que el concepto de ingeniería aplicado a las ciencias de computación (o más ampliamente la visión ingenieril de la informática) va encaminada hacia la sistematización del desarrollo de software. Sin embargo, todavía no estamos en disposición de responder a la pregunta que nos planteábamos al principio. Si establecemos un pequeño paralelismo entre la IS y la IC, podríamos llegar a deducir que si en la IS el elemento central es el software, en la IC el elemento central debe ser el 'software con conocimiento'. Pero ¿qué podríamos entender por 'software con conocimiento'?. Una rápida consulta nos lleva al término sistema experto o sistema basado en conocimiento (SBC en adelante), que se puede definir como: "un sistema software capaz de soportar la representación explícita del conocimiento de un dominio específico y de explotarlo a través de los mecanismos apropiados de razonamiento para proporcionar un comportamiento de alto nivel en la resolución de problemas"[Guida and Tasso, 1994]. En otras palabras los SBC tratan con problemas poco estructurados en los que nos podemos encontrar requisitos subjetivos, entradas inconsistente, incompletas o con incertidumbre y que no pueden ser resueltos aplicando los algoritmos clásicos o la investigación operativa [Alonso *et al.*, 1995].

De esta forma podemos establecer que el objetivo de la IC es el mismo que el de la IS y que no es más que el de transformar el proceso de desarrollo de SBC de un arte a un disciplina ingenieril. Una vez que hemos llegado a este nivel, ya estamos en condiciones de dar una respuesta a la pregunta objeto de esta sección:

la Ingeniería del Conocimiento es la disciplina tecnológica que se centra en la aplicación de una aproximación sistemática, disciplinada y cuantificable al desarrollo, funcionamiento y mantenimiento de Sistemas Basados en Conocimiento. En otras palabras, el objetivo último de la IC es el establecimiento de metodolgías que permitan abordar el desarrollo de SBC de una forma más sistemática.

Un punto que merece la pena aclarar ahora es el significado del concepto *metodología*. A lo largo de este trabajo, entenderemos por metodología un conjunto de métodos y prácticas que permiten desarrollar de una manera efectiva todas aquellas tareas que están implicadas en el diseño, producción, mantenimiento y extensión de un determinado producto, que en nuestro caso es un SBC. Por lo tanto, una metodología indica cómo se deben desarrollar las tareas definidas de acuerdo a un determinado ciclo de vida. Esta última afirmación es de vital importancia por cuanto implica que una metodología necesita de la existencia de un ciclo de vida, y que además supone el punto de apoyo sobre el que se define.

Como se puede apreciar, la IC y la IS son disciplinas completamente paralelas, aunque presenten un desfase temporal de unos 10 años. Muchas de las técnicas de IS han sido adapatadas a la IC con relativo éxito, aunque queda todavía mucho camino por recorrer. Sin embargo, no se debe pensar en la IC como la piedra filosofal que nos permitirá solucionar el desarrollo de aplicaciones para problemas mal estructurados y cuya definición no es completa, más bien la debemos considerar como una herramienta que nos permitirá aislar y delimitar las partes del problema que provocan esa indefinición y como abordar su desarrollo. A pesar de esta, en principio, limitación, la IC tiene que ser analizada desde una perspectiva más amplia, ya que empieza a ser utilizadas en otras disciplinas que en la actualidad están adquiriendo una enorme importancia, como por ejemplo la Gestión del Conocimiento.

A continuación, y para hacernos una idea de cuáles fueron las causas que originaron la necesidad de establecer la IC como una disciplina, haremos una revisión histórica en la que podremos apreciar cómo ha evolucionado esta disciplina desde sus inicios hasta nuestros días.

2 Del Principio hasta la Crisis de la Ingeniería del Conocimiento

Desde sus principios, el principal objetivo de la Inteligencia Artificial (IA, en adelan-

te) era el desarrollo de sistemas que pudieran "pensar"y resolver problemas tan inteligentemente como lo hacían los expertos humanos. Los primeros trabajos que se iniciaron en esta área se centraban en el desarrollo de técnicas generales para la resolución de problemas, como por ejemplo los trabajos de Fikes y Nilsson [Fikes and Nilsson, 1971] en 1971 que dieron como resultado el sistema de planificación STRIPS, o los de Newell y Simons [Newell and Simon, 1988] en 1963 y que desembocaron en el GPS (General Problem Solver). Sin embargo, a finales de la década de los setenta, los investigadores en el campo de la IA reconocieron que los métodos generales de resolución de problemas y las técnicas de búsqueda desarrolladas en los diez años anteriores resultaban insuficientes para resolver ciertos problemas de investigación y desarrollo. Como resultado de esta afirmación, se empezó a considerar que el potencial de un ordenador a la hora de resolver problemas radicaba en el conocimiento que poseía sobre el dominio de la aplicación más que en el mecanismo de inferencia empleado. Por lo tanto, indirectamente se estableció que era más necesario el conocimiento específico sobre el dominio de la aplicación, que un conocimiento general que pudiera ser aplicado sobre varios dominios. Esta postura llevó a los investigadores a afirmar que el conocimiento podía ser adquirido de los expertos y "transferido" a un ordenador a través de una representación que estos pudieran manipular.

El conjunto de ideas que surgieron a finales de la década de los setenta puede ser considerado como el primer punto de inflexión en el desarrollo de los Sistemas Basados en Conocimiento (SBC en adelante) o Sistemas Expertos, dando lugar a sistemas que fueron aplicados en multitud de dominios. Entre estos sistemas se encuentran PROSPECTOR[Duda *et al.*, 1978], XCON [Mcdermott, 1982], MYCIN[Shortliffe, 1976], GUIDON[Clancey, 1979], LES [Scarl *et al.*, 1987] e INTERNIST [Miller *et al.*, 1982].

Estos sistemas eran construidos utilizando el típico ciclo de codificación y corrección de errores, tan utilizado en los laboratorios de investigación de donde surgieron. El conocimiento que poseían estos primeros SBC se obtenía preguntando a los expertos cómo resolvían un determinado problema. Posteriormente, el ingeniero del conocimiento codificaba lo que se recogía del experto en forma de reglas heurísticas (empíricas o asociacionales) que de alguna forma mapeaban las características observables del problema en las conclusiones. Estos SBC estaban dotados de una estructura de control simple y una representación uniforme del conocimiento. El conocimiento era representado en un mismo nivel de abstracción e implícitamente se combinaba conocimiento sobre *cómo* desarrollar una determinada tarea, sobre *qué* se encuentra en el dominio y el *por qué* funcionan las cosas.

El hecho de que en los primeros SBC el conocimiento sobre el problema no estuviera estructurado llevó, a pesar de los prometedores resultados obtenidos, a que la transferencia de éstos al campo comercial a la hora de construir grandes SBC fuera un fracaso en la mayoría de los casos. Aparecían problemas de estimación del tiempo de desarrollo, no se cumplían las expectativas iniciales sobre lo que debería de hacer el producto y, sobre todo, el mantenimiento y verificación de dichos sistemas era una tarea difícil y costosa. Como se indica en [Studer *et al.*, 1998], esta situación es directamente comparable con lo que ocurrió en el desarrollo de los sistemas de información tradicionales en la tan mencionada crisis del software a finales de los sesenta: los medios con los que se desarrollaron los prototipos académicos no pudieron ser extendidos al diseño y mantenimiento de los grandes sistemas de información de larga durabilidad requeridos por las empresas. Por lo tanto, de la misma forma que la crisis del software dio como resultado el establecimiento de la Ingeniería del Software como una disciplina, la situación que se produjo con el desarrollo de los primeros SBC dejó claro la necesidad de establecer planteamientos más metodológicos, dando como resultado el nacimiento de la Ingeniería del Conocimiento como una disciplina más de la IA. El objetivo de esta nueva disciplina, al igual que la Ingeniería del Software, es la de *transformar el proceso de construcción de un SBC de un arte a un proceso de ingeniería. Para llevar a cabo este fin se tenía que revisar todo el proceso de desarrollo y mantenimiento de los SBC, y desarrollar métodos, lenguajes y herramientas especializadas para dicho fin* [Shaw and Gaines, 1992].

3 Buchanan y su Modelo de Ciclo de Vida

Si observamos desde este nuevo punto de vista ingenieril el desarrollo de los primeros sistemas expertos en las décadas de los setenta y ochenta, se puede considerar que el proceso de construcción de un SBC era un proceso de transferencia del conocimiento humano a una base de conocimiento implementada en un ordenador. El conocimiento humano, como ya se mencionó en apartados anteriores era, extraído mediante una serie de entrevistas al experto. Obviamente, este proceso de transferencia descansaba sobre la idea de que el conocimiento requerido por el SBC puede ser recogido e implementado en un ordenador. El paradigma de la Ingeniería del Conocimiento como un proceso de transferencia tiene su culminación en el trabajo de [Buchanan *et al.*, 1983]. En dicho trabajo se presenta el desarrollo de un SBC como un proceso de Adquisición de Conocimiento, entendido éste como la transferencia y transformación de la experiencia en la resolución de problemas desde alguna fuente de conocimiento a un programa. El conocimiento a ser elicitado está formado por una colección de hechos, procedimientos y reglas sobre un dominio concreto. Para este proceso de extracción de conocimiento se necesita una persona, el ingeniero del conocimiento que sirva de intermediario entre el experto y el programa.

Una de las aportaciones más importante del trabajo de Buchanan fue la identificación del proceso de adquisición de conocimiento como un cuello de botella en el proceso de desarrollo de un SBC. Este problema aparece como consecuencia de la falta de "conocimiento"que el ingeniero del conocimiento tiene sobre el dominio de la aplicación, lo que nos lleva a un problema de comunicación con el experto. Para ayudar al ingeniero del conocimiento a solventar este problema, se propone un ciclo de vida para el proceso de adquisición de conocimiento que abarca desde la concepción del sistema hasta su madurez. La metodología propuesta se basaba en el típico ciclo de vida en cascada utilizado en los inicios de la ingeniería del software (figura 1), de la que se puede deducir que el proceso de construcción de un sistema experto se plantea como un proceso de revisión casi constante, que puede implicar la redefinición de los conceptos, de las representaciones o el refinamiento del sistema implementado. Este refinamiento pasa por repetir el ciclo en sus dos últimas etapas para de esta forma ajustar la base de conocimiento y las estructuras de control hasta alcanzar el comportamiento deseado.

La importancia de esta metodología radica en que es el primer intento de planteamiento de una metodología para el desarrollo de SBC que permitía abordar problemas que se pueden presentar en el mundo de la industria. Como se puede observar, la metodología presentada se basaba en el clásico ciclo de vida planteado en el modelo de cascada por la Ingeniería del Software, permitiendo la realización de bucles para completar etapas pasadas. Durante los años siguientes se plantearon numerosas modificaciones a esta metodología para permitir el desarrollo por prototipado [Kahn, 1994] o su adaptación al ciclo de vida en espiral derivado del trabajo de [Boehm, 1988].

Estas metodologías fueron utilizadas con relativo éxito hasta principios de la década de los noventa, fecha en la que se replantea la ingeniería del conocimiento desde la perspectiva de la transferencia hacía la perspectiva del modelado, como un intento de solucionar el problema del cuello de botella que suponía la fase de adquisición de conocimiento. Además del problema que suponía el cuello de botella de la adquisición de conocimiento, se detectaron carencias importantes que intentarían ser resueltas a la luz de nuevas metodologías. Las carencias que hicieron replantear la Ingeniería del Conocimiento se pueden resumir en:

- La generación de explicaciones era complicada debido a la ausencia de separación explícita entre el conocimiento sobre el "cómo", el "qué"y el "por qué", limitándose dichas explicaciones a simples trazas de ejecución.
- El sistema no tenía conciencia de sus propias limitaciones, con lo que tendían a dar respuestas de escasa utilidad en vez de responder con un simple "no se"(tal y como actuaría un experto responsable) a preguntas que pueden estar fuera de su dominio de conocimiento.
- El mantenimiento de dichos sistemas era complicado. Por un lado la validación

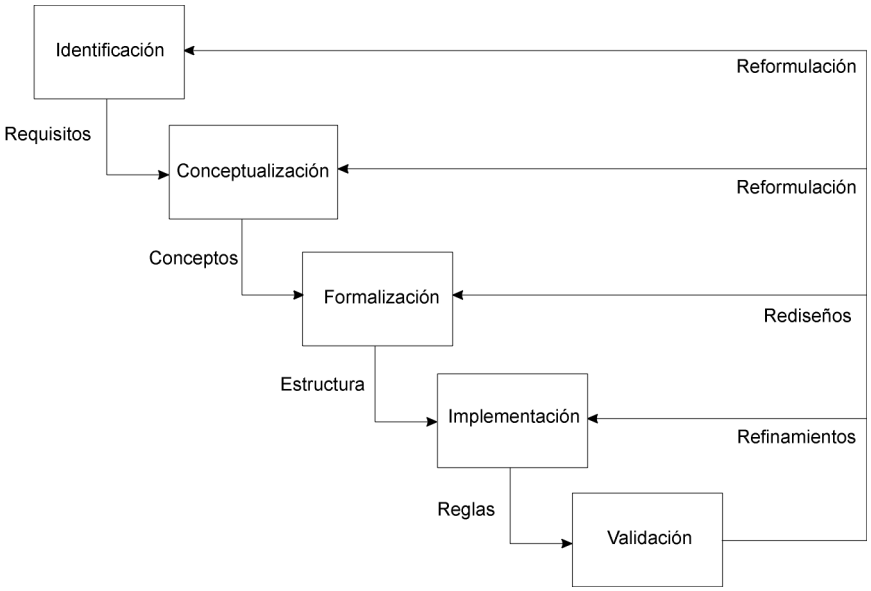

Figura 1: Modelo de Ciclo de Vida propuesto por Buchanan et al.

del conocimiento era una tarea compleja al estar éste desperdigado sobre la base de reglas. Por otro lado, la dispersión del conocimiento provocaba que la extensión de la base de conocimiento (mediante la adición de nuevas reglas) resultara compleja, sobre todo a la luz del trabajo de [Heckerman and Horovitz, 1986] que demostraron que la independencia y modularidad de un conjunto de reglas de una base de conocimiento no era tan evidente, ya que la adición de reglas podían producir interacciones negativas con el resto de las reglas, degradando de esta forma la eficiencia del sistema.

4 El Cuello de Botella de la Adquisición de Conocimiento

Como ya mencionamos en el apartado anterior el término *cuello de botella de la adquisición de conocimiento* fue acuñado por Buchanan [Buchanan *et al.*, 1983] y se utilizó para hacer referencia al hecho de que la adquisición de conocimiento es el punto que plantea una mayor dificultad a la hora de crear una base de conocimiento. En [Musen, 1993] se enumeran las causas principales de esta dificultad radican en los siguientes problemas:

1. **El problema del conocimiento tácito**. Aunque no se sabe exactamente como el conocimiento tácito es codificado psicológicamente en la memoria humana, se da por hecho que la resolución de problemas por parte de los expertos involucra el uso de este tipo de conocimiento. Por lo tanto, no es de extrañar que el conocimiento tácito sea el objeto de estudio de la adquisición de conocimiento. Además, a medida que las personas se vuelven más experimentadas en su área de conocimiento y aplican repetidamente su conocimiento declarativo a determinadas tareas, éste se vuelve tácito, perdiendo de esta forma conciencia de lo que realmente saben. Consecuentemente, el conocimiento que es más idóneo para su incorporación a un SBC, y por lo tanto el objeto de la adquisición de conocimiento, es el tipo de conocimiento sobre el qué los expertos tienen menos conciencia, con lo cual su adquisición se hace una tarea bastante compleja.

2. **El problema de la comunicación**. Este problema radica en que los expertos en una determinada área de aplicación y los ingenieros del conocimiento no utilizan el

mismo lenguaje para comunicarse. Como se afirma en [Musen, 1993], para poder representar el conocimiento relevante en un determinado tema, el ingeniero del conocimiento se debe familiarizar con el domino de la aplicación, debe de aprender el vocabulario utilizado en dicha área, y quizás, una nueva forma de ver los problemas. Por lo tanto, el ingeniero del conocimiento debe esforzarse en comprender el área sobre la que el experto va a hablar, o puede no entender lo que el experto trata de comunicar. Por otro lado, el problema de la comunicación no solo recae en la falta de conocimiento sobre el dominio que posee el ingeniero del conocimiento, sino que también se debe al hecho de que el experto, en la mayor parte de los casos, carece de conocimientos de programación, y por lo tanto, no sabe qué conocimiento es el es idóneo para ser codificado en el ordenador.

3. **El problema de utilizar representaciones del conocimiento**. Además de las barreras lingüísticas y cognitivas, otro componente adicional al cuello de botella de la adquisición del conocimiento radica en el lenguaje elegido para la codificación del mismo, ya que los lenguajes usados para codificar el conocimiento carecen generalmente de la suficiente potencia expresiva. En [McCarthy and Hayes, 1969] se introdujo el término *adecuación epistemológica* como la habilidad de un formalismo de representación del conocimiento para expresar los hechos que una persona conoce sobre algún aspecto del mundo real. Este aspecto es bastante importante ya que hay experimentos que demuestran que la forma en la que es modelado el conocimiento del experto, depende fuertemente del lenguaje de representación utilizado. Por lo tanto, las primitivas ofrecidas por las herramientas de construcción de SBC tiene una influencia bastante fuerte la forma en el que el conocimiento sobre el dominio es adquirido.

Estos problemas se hicieron cada vez más evidentes a medida que se intentaba abordar la construcción de SBC más complejos, dando lugar a que las funcionalidades esperadas no se cumplieran y que tampoco se cumplieran con los plazos establecidos para el desarrollo. Esto evidenciaba el hecho de que se había llegado a un punto en el que el tamaño de los problemas que se intentaban abordar requería de unas metodologías de desarrollo más elaboradas de lo que existían en esos momentos.

5 El Nivel de Conocimiento

Aunque la crisis de la ingeniería del conocimiento era palpable, pocos esfuerzos se realizaron para determinar cuál eran las causas principales de dicha crisis. Fue Newell [Newell, 1982] el primero que analizó en profundidad este problema. Para Newell el principal problema que había que abordar era el establecimiento de la diferencia que existe entre lo que entendemos por conocimiento y su representación, diferencias que en ese momento no estaban claras. El origen de este problema se debía al hecho de que al ser el nivel simbólico el nivel más alto en la jerarquía, tanto el conocimiento como su representación se colocaban en dicho nivel, y por lo tanto, no se podían diferenciar de una forma clara. Para resolver el problema, Newell propone la existencia de un nivel adicional al que le da el nombre de *nivel de conocimiento*. Este nuevo nivel hay que considerarlo realmente como un nivel y como veremos nos va a ayudar a distinguir entre lo que es conocimiento y su representación.

En una rápida aproximación al nivel de conocimiento, podemos decir que:

- El sistema en este nivel es un agente.
- Sus componentes son los objetivos, acciones y cuerpos.
- El medio lo constituye el conocimiento. Por lo tanto, el agente procesará el conocimiento que tiene para determinar que acciones tiene que tomar.
- La ley de comportamiento está determinada por el principio de racionalidad: las acciones son seleccionadas para alcanzar los objetivos, es decir, el agente selecciona las acciones que sabe que le llevan a cubrir los objetivos.

Por lo tanto, al definir un sistema en el nivel de conocimiento lo estamos tratando como si

poseyera conocimiento y objetivos, además de suponer que hará todo lo posible para alcanzar dichos objetivos, en la medida en que su conocimiento se lo permita. En la jerarquía de niveles, el nivel de conocimiento se sitúa sobre el nivel simbólico, con lo que sus componentes (acciones, objetivos y cuerpos) y medio (conocimiento) pueden ser definidos en términos de dicho nivel. La definición estructural de este nuevo nivel es bastante simple, ya que sólo podemos decir que está formado de cuerpos de conocimiento, objetivos y acciones, y que estos están de alguna forma conectados en el sentido de que todos los componentes son tenidos en cuenta en la selección las acciones que hay que tomar.

Hasta la aparición de este nuevo nivel, se consideraba que el nivel simbólico estaba divido en dos subniveles: uno que era propiamente el nivel simbólico y otro que corresponde al nivel de conocimiento, pero la descripción de sistemas que manifiestan un comportamiento inteligente necesita que estos dos niveles sean tratados por separado. La existencia de este nivel queda reflejada en la siguiente hipótesis:

Hipótesis del Nivel de Conocimiento. *Existe un nivel adicional en los sistemas computacionales, que está situado justo encima de nivel simbólico, que está caracterizado por tener el conocimiento como medio y el principio de racionalidad como ley del comportamiento.*

Para describir este nuevo nivel de forma autónoma, es decir, sin hacer referencia alguna a los niveles inferiores, se deben de tratar los siguientes aspectos: la estructura del nivel, constituida por sus componentes y las leyes de composición, las leyes de comportamiento que lo rige, y por último del medio, el conocimiento.

5.1 La Estructura del Nivel de Conocimiento

El agente, es decir, el sistema definido en el nivel del conocimiento, tiene una estructura muy simple. En una primera aproximación podemos decir que un agente está compuesto de un *cuerpo físico*, que le permite interaccionar con el entorno. Este cuerpo físico está formado por un conjunto de *acciones* a través de las cuales se puede realizar dicha interacción. Aunque el cuerpo físico que permite la interacción con el entorno puede ser lo complejo que se quiera, su complejidad no queda reflejada en el sistema descrito en el nivel de conocimiento, ya que este último sólo puede evocar el comportamiento del sistema.

Otro de los elementos que se encuentran en este nivel es el *cuerpo de conocimiento*, que es como una memoria que contiene todo lo que el agente sabe en un determinado instante. Como se puede deducir, el conocimiento es añadido a esta memoria por medio de la ejecución de las acciones. Aunque de algún modo este cuerpo de conocimiento se comporta como una memoria, no tiene ninguna similitud estructural con el concepto de memoria utilizado en los niveles inferiores. Finalmente se puede decir que un agente posee un conjunto de objetivos. Un objetivo es una parte diferenciada del cuerpo del conocimiento que define un determinado estado del entorno. Esta diferenciación del resto del cuerpo del conocimiento les permite desarrollar un papel distinto en el comportamiento que presenta el agente, describiendo de alguna forma las intenciones del mismo.

Un hecho importante que se puede deducir es que en este nivel no existe ninguna ley de composición, a diferencia de los niveles inferiores en los que diferentes leyes de composición definen sistemas con distinto comportamiento. Esta ausencia de estructura es la clave del nivel de conocimiento, en el sentido de que el comportamiento del sistema se define en términos de lo que el agente sabe, y no en la forma de ensamblar sus componentes. Por lo tanto, podemos decir que la estructura interna del sistema se define de una forma en la que no se tiene que conocer nada sobre ella para predecir el comportamiento del mismo, el cuál sólo depende de lo que sabe el agente, lo que quiere y de los medios que tiene para interaccionar con el entorno. Como se verá más adelante, este servirá como punto de partida de la idea de la reutilización del conocimiento, es decir, la posibilidad de definir componentes genéricos de conocimiento que puedan ser aplicables a distintos problemas se puede interpretar como la utilización del mismo nivel de conocimiento para distintas instancias de los niveles inferiores.

5.2 El principio de Racionalidad

Bajo este nombre se esconde la ley que rige el comportamiento de un agente y que permite la predicción de su comportamiento. Este principio puede ser formulado en los siguientes términos:

Principio de racionalidad. *Si un agente tiene conocimiento de que una de sus acciones puede llevarle a la consecución de uno de sus objetivos, entonces el agente seleccionará dicha acción.*

Como se puede observar este principio establece la conexión del conocimiento y los objetivos con el proceso de selección de acciones, sin especificar el mecanismo por el que se lleva a cabo dicha conexión. Esta característica también es distintiva respecto al resto de niveles. En los demás niveles, el comportamiento queda determinado por la forma en que son procesados los componentes. En el nivel de conocimiento el principio de racionalidad nos define la conexión entre las acciones y los objetivos de acuerdo con el conocimiento que posee el agente.

5.3 La Naturaleza del Conocimiento

En las secciones anteriores quedó claro que el medio utilizado en el nivel de conocimiento es el propio conocimiento, es decir, algo que es procesado según el principio de racionalidad. En una definición más formal, podemos decir:

Conocimiento. *Cualquier cosa que puede ser adscrita a un agente y que su comportamiento se puede computar según el principio de racionalidad.*

El conocimiento puede ser descrito totalmente en términos funcionales (qué hace), o en términos estructurales (una serie de objetos que poseen determinadas propiedades y relaciones). Sin embargo, hay que tener en cuenta que el papel que juega el conocimiento no puede ser cubierto directamente por una estructura física, sino que se cubre aproximadamente e indirectamente por distintos sistemas simbólicos.

Un aspecto diferenciador del conocimiento respecto a los medios que podemos encontrar en el resto de los niveles, es el hecho de que el conocimiento no puede ser descrito en términos de los distintos estados en los que pueden encontrarse sus estructuras físicas. Esta característica diferenciadora le da al conocimiento un aspecto abstracto, con lo que no puede ser explicitado con la claridad que lo son el resto de los medios de los niveles inferiores. Sólo puede ser imaginado como resultado de los procesos interpretativos que operan en el nivel simbólico. Por lo tanto, no lo podemos considerar como un conjunto de expresiones simbólicas con una organización estática, hay que considerarlo como compuesto de procesos y estructuras de datos.

Esta visión del conocimiento puede considerarse como la constatación de la existencia del cuello de botella de la adquisición de conocimiento en el proceso de desarrollo de un SBC. El hecho de que sea una entidad abstracta y difícil de detectar, nos puede ayudar a comprender lo difícil del proceso de extracción de conocimiento. Si además, a esto le unimos la situación que existía antes del trabajo de Newell, en la que el nivel de conocimiento se consideraba parte del nivel simbólico, el proceso de construcción de un SBC se convertía en un intento de superar dos barreras: por un lado la dificultad de la detección del conocimiento, y por el otro, el intento de implementarlo directamente en el nivel simbólico, que como dijo Newell, sólo permite reflejar un conjunto bastante reducido de la variedad de todos los posibles comportamientos, resultado de intentar representar un comportamiento indeterminista mediante estructuras deterministas. Newell nos indica que este problema puede ser resuelto representando el conocimiento en un nivel superior, el nivel de conocimiento.

6 KLIC

KLIC (KBS life cycle o Ciclo de Vida para SBC) fue presentada por Guida y Tasso [Guida and Tasso, 1994]. Supuso el primer intento serio en el planteamiento de una metodología para le desarrollo de SBC, ya que no solo se le da importancia al propio proceso de desarrollo sino que también se presta mucha atención a los productos generados en cada una de los procesos. Básicamente, en el nivel más alto de abstracción el ciclo de vida propuesto está compuesto de una serie de *fases* que se corres-

ponden con los procesos principales del ciclo de vida: diseño, producción y mantenimiento. Cada una de estas fases se descompone en una serie de tareas. El modelo planteado es una mezcla entre un modelo ciclo de vida en cascada, las fases se ejecutan en un orden estrictamente secuencial y un modelo en espiral, ya que la ejecución de las tareas puede implicar estructuras de control de tipo bucle, además de estructuras condicionales, de selección y de ejecución paralela.

KLIC se organiza en 6 fases que se agrupan en tres macrofases que se corresponden con los procesos de análisis, desarrollo y mantenimiento:

Fase-0. Análisis de Posibilidades. En esta fase se indentifican qué áreas de la organización bajo estudio se podrían beneficiar del desarrollo de un SBC y se clasifican de acuerdo a su importancia estratégica y táctica, beneficios esperados, costes y complejidad técnica. El producto que se genera en esta fase es el *informe sobre el análisis de posibilidades*, que entre otras cosas incluye un plan maestro para la organización para que esta se beneficie del uso de las tecnologías basadas en el conocimiento.

Fase-1. Análisis de Viabilidad. En esta fase se analiza el dominio de aplicación (posiblemente sugerido por el plan maestro) y se identifica el problema que hay que resolver. A partir de aquí, se analizan los requisitos, se definen los objetivos del proyecto y se describen las especificaciones funcionales, técnicas y el criterio de aceptabilidad. El producto generado en esta fase es el *Informe sobre el análisis de la viabilidad* que además de la información anteriormente comentada, incluye las primeras versiones del diseño técnico, el diseño de la organización y del plan del proyecto.

Fase-2. Construcción del Demostrador. El principal objetivo de esta fase es la construcción de una primera versión y muy limitada del SBC. Esta primera versión nos permitirá entre otras cosas: validar, refinar y reconsidear las decisiones técnicas y el plan del proyecto realizados en la Fase 1, y refinar el análisis de requisitos una vez consultados los usuarios finales. Los productos de esta fase son el *Demostrador* y el *informe sobre el demostrador* en el que se resume las actividades realizadas y los resultados alcanzados.

Fase-3. Desarrollo del Prototipo. El objetivo de esta fase es la de encontrar la solución técnica más apropiada para la aplicación considerada y en base a esta, construir el SBC objeto del proyecto. Los productos generados en esta fase son: el *prototipo*, que es un SBC que de alguna forma cumpla con todas las especificaciones funcionales, el *conjunto de herramientas de desarrollo* que han servido de ayuda para la construcción de la Base de Conocimiento y el *informe sobre el prototipo*, en el que se recogen un resumen de las actividades desarrolladas y los resultados alcanzados. Sin embargo, un hecho bastante importante es que los autores proponen un desarrollo incremental para el prototipo. De esta forma vemos como en un ciclo de vida en cascada se intercalan fases basadas en un ciclo de vida en espiral.

Fase-4. Implementación, Instalación y Entrega del Producto Final. Aquí el objetivo es el desarrollo del SBC completo, el cual tiene que tener el mismo comportamiento que el prototipo pero tiene que estar instalado en un entorno real de operación, verificado con datos reales y, eventualmente, entregado a los usuarios para que lo exploten. Los productos de esta fase son: el *Sistema final*, el *Sistema de soporte para el mantenimiento*, los *manuales de usuario, técnicos y de mantenimiento* y el *Informe sobre el Sistema Final*.

Fase-5. Mantenimiento y Extensión. Esta fase comienza una vez que el producto final (el SBC desarrollado) es entregado a los usuarios finales para su utilización y se extiende durante el resto del ciclo de vida del SBC. En ella se monitoriza la vida operacional del SBC, se realizan intervenciones correctivas, intervenciones adaptativas y se realiza un mantenimiento proactivo y evolutivo. Los productos de esta fase son: las nuevas versiones del SBC, así como los manuales actualizados, y la historia de modificaciones del SBC.

Como podemos apreciar, el trabajo de Guida y Taso supone una de las primeras aproximaciones serias hacia una metodología para el des-

arrollo de SBC. La base de este trabajo reside en el intento de adaptar los resultados alcanzados en la Ingeniería del Software al campo de la Ingeniería del Conocimiento que en esos momentos empezaba ha adquirir un auge importante. Aunque en líneas generales KLIC se basa en el modelo de ciclo de vida en cascada clásico, deja sentadas las bases sobre las que se construirán las nuevas metodologías: (1) se considera el proceso de desarrollo de SBC una tarea compleja y distinta a la del desarrollo de sistemas software tradicionales, y se adopta el ciclo de vida en espiral como el más adecuado para esta fase; y (2) el análisis la viabilidad del proyecto abarca un espectro más amplio que en la Ingeniería del Software, requiriendo un estudio mucho más profundo del entorno organizacional en el que se desarrolla. Como veremos en los siguientes párrafos, estos aspectos serán comunes a las nuevas metodologías basadas en el paradigma del modelado del conocimiento.

7 Los Primeros Pasos hacia las Técnicas de Modelado

En la sección anterior quedó claro que antes de la aparición del trabajo de Newell [Newell, 1982] la discusión en el desarrollo de SBC se realizaba en términos del nivel simbólico, con lo que el debate no se centraba en la relación entre las tareas y el tipo de conocimiento que necesitan. Para centrar la discusión en dichos términos, se plantea que el análisis de los SBC se haga de manera independiente a la implementación. Como el lector, puede suponer esta proposición desemboca en el nuevo paradigma de la Ingeniería del Conocimiento basado en el modelado del conocimiento. Pero como veremos, el camino que se tuvo que recorrer desde que se reconociera la existencia del nivel de conocimiento a la aparición de las primeras metodologías no fue un camino sencillo, sino más bien un proceso en el que se fueron refinando una serie de ideas que fueron surgiendo. Por esto, creemos que es importante comentar dos trabajos que la mayor parte de los autores los sitúan en el puente entre las metodologías modernas y el trabajo de Newell. Se trata de los métodos de limitación de roles y las tareas genéricas, trabajos que se pueden considerar paralelos y que ponen de manifiesto dos de los aspectos más importantes que han llevado al planteamiento de metodologías para el desarrollo de SBC: la idea de limitar los papeles (roles) que puede jugar el conocimiento dentro de un determinado dominio y los conceptos de tareas y métodos genéricos como elementos constitutivos de los SBC.

7.1 Métodos de Limitación de Roles

Los métodos con limitación de roles (MLR, en adelante) [McDermott, 1988] se pueden considerar como uno de los primeros intentos de explotar la reutilización de los métodos de resolución de problemas en el desarrollo de los SBC. McDermott puso de manifiesto que todos los SBC desarrollados hasta la fecha de su trabajo se caracterizaban por tener un conocimiento de control, es decir, un motor de inferencia o método de resolución de problema utilizado (MRP, en adelante) bastante sencillo. Esta simplicidad del conocimiento de control aportaba la flexibilidad suficiente para que éste pudiese ser aplicado a gran cantidad de tareas de distinta naturaleza. Sin embargo, esta flexibilidad obligaba a que parte del conocimiento de control necesario para llevar a cabo tareas concretas se codificara junto con la base de conocimiento, con lo que la selección y secuencia de acciones que se llevaba a cabo dependía del dominio de la aplicación. Obviamente, esta falta de distinción entre el control y el conocimiento específico de la aplicación conllevaba problemas de mantenimiento en las aplicaciones desarrolladas.

Otro problema que surgía al utilizar MRP que obligaban a añadir el conocimiento de control específico de la tarea, al mismo tiempo que se construía la base de conocimiento, es que no se proporcionaba ninguna guía sobre qué tipo de conocimiento debería ser adquirido y cómo éste debería ser codificado. Esto era debido a que los requisitos necesarios sobre el conocimiento específico de la tarea no se podían establecer hasta que no se conociese el conocimiento de control adicional que se necesitaba, y por supuesto, esto no se podía deducir de la definición del MRP.

Para resolver estos problemas, McDermott propuso analizar los MRP utilizados en distintas aplicaciones y redefinirlos llevando hasta sus

últimas consecuencias la diferenciación, tan utilizada en los SBC desarrollados hasta la fecha, del MRP y la base de conocimiento específico de la tarea. Este análisis llevó a las siguientes conclusiones:

- Existían familias de tareas (como por ejemplo el diagnóstico) que podían ser resueltas por métodos cuyo conocimiento de control era lo suficientemente abstracto como para que no se viese influenciada por las características particulares de cualquiera de los miembros de la familia de tareas.
- Al definir un método en los términos anteriores, se indicaba implícitamente qué roles pueden jugar los componentes del conocimiento específico de la tarea. De esta forma, la definición del método proporcionaba una guía lo suficientemente clara de qué conocimiento debe ser adquirido y cómo éste debe ser implementado.
- La definición de los roles que podía jugar el conocimiento específico de la tarea, provocaba una disminución en la diversidad del conocimiento que se tenía que utilizar en la implementación de la tarea.

A estos métodos que proporcionaban las líneas básicas para la adquisición y codificación de conocimiento se les denominó Métodos con Limitación de Roles. Este tipo de métodos constituía una clase bastante importante ya que además de indicar qué conocimiento era necesario adquirir, se podían utilizar en un amplio número de dominios diferentes. Generalmente, un MLR estaba formado por un bucle definido sobre 5 o 10 pasos. Algunos de estos pasos operaban sobre módulos de conocimiento específicos de la tarea, con la condición de que en su definición no se incluyera ningún tipo de conocimiento de control. De esta forma, se conseguía dotar de cierta regularidad al conocimiento específico de la tarea requerido por el MLR. Esta regularidad en el conocimiento hacía tratable la automatización de la adquisición del conocimiento. De esta idea surgieron varias herramientas entre las que cabe destacar MOLE [Eshelman *et al.*, 1993] que se centraba en el MLR de cubrir y diferenciar y que fue aplicado con éxito a numerosos problemas de diagnóstico y SALT [Marcus and McDermott, 1993] que se basaba el MLR de proponer y revisar.

Los MLR propuestos por McDermott fueron utilizados en un gran número de dominios diferentes, pero sin embargo, cuando se intentaba abordar algún problema del mundo real aparecía el problema de cómo precisar si una tarea concreta podía ser resuelta por un MLR determinado (hay que tener en cuenta que este problema está todavía sin resolver). Otros de los inconvenientes que aparecieron se debieron al hecho de que los MLR presentaran una estructura fija, con lo que no se podían utilizar en problemas que requerían la combinación de varios MLR. La principal causa de este problema radicaba en el hecho de que no se había definido de una forma clara la estructura y los elementos constitutivos del conocimiento de control de los MLR, problema que sería resulto de forma paralela por el trabajo de Chandrasekaran que se describirá a continuación.

7.2 Las Tareas Genéricas y Las Estructuras de Tareas

El concepto de tarea genérica (TG, en adelante) [Chandrasekaran, 1983, Chandrasekaran, 1986] [Chandrasekaran and Johnson, 1993] tiene sus orígenes en los trabajos de Gómez y Chandrasekaran [Gómez and Chandrasekaran, 1981], que dieron como resultado la identificación del proceso de clasificación como un elemento común a los problemas de diagnóstico, y en los trabajos de Mittal y Chandrasekaran [Mittal and Chandrasekaran, 1984] que añadieron la abstracción de datos como otro de los elementos comunes a dicho proceso. Estos resultados fueron producto del trabajo realizado en el desarrollo del sistema de diagnóstico MDX [Chandrasekaran and Mittal, 1983] [Chandrasekaran *et al.*, 1979].

El trabajo del grupo de Chandrasekaran se basa en la idea intuitiva de que tienen que existir tipos de conocimiento y requisitos de control que sean comunes al razonamiento de diagnóstico en diferentes dominios, de la misma forma que cabe esperar que la representación del conocimiento asociado a la tarea de diagnóstico se pueda realizar utilizando un vocabulario común. Además, cabe esperar que esta idea se pueda aplicar a otros tipos de razonamiento, como se hizo con el proceso de diseño, y que el resultado obtenido fuera distinto al obtenido con el proceso de diagnóstico. Otra de las ideas que sirvieron de base para este trabajo

fue el hecho de que la experiencia esté constituida por un conjunto organizado de conocimiento (más de lo que podían ofrecer los sistemas basados en reglas), con un comportamiento de control que está indexado por la organización y forma del conocimiento que contienen.

Todos estos antecedentes llevaron a la definición de un TG como una estructura de conocimiento que se describe en el nivel de conocimiento, y que especifica una tarea de utilidad general (por ejemplo, clasificación), un método para llevarla a cabo y las clases de conocimiento que necesita dicho método. En la primera aproximación se identificaron seis TG [Chandrasekaran, 1986]: clasificación jerárquica, valoración de hipótesis, transferencia de información dirigida por conocimiento, ensamblado abductivo, diseño jerárquico por selección de planes y refinamiento y abstracción de estados.

Estas TG pueden ser utilizadas como primitivas a la hora de definir tareas más complejas como el diagnóstico y diseño. Por lo tanto, las TG pueden ser vistas como bloques constructivos para la composición de tareas más complejas. Sin embargo, a pesar de que se admitía la necesidad del análisis en el nivel de tareas y las ventajas que este ofrecía a la adquisición de conocimiento y la Ingeniería del Conocimiento, la definición de las TG presentaban las siguientes desventajas [Chandrasekaran *et al.*, 1992]:

- No queda claro cuál es la diferencia entre tareas y clases de tareas, y no se especifica como están relacionadas las TG complejas con las simples.
- Las TG propuestas están definidas a niveles de complejidad diferentes, es decir, no queda claro cuál es el nivel de abstracción en el que se tienen que definir.

Para solventar dichos problemas se propone una evolución del concepto de TG hacía el concepto de Estructura de Tareas (ET, en adelante). En este nuevo marco se consideran los siguientes elementos:

- **Tareas**. El término tarea se refiere a un tipo de problema o conjunto de instancias de problemas con algo en común, es decir, especifica el problema a resolver y el objetivo a alcanzar. En este punto es importante distinguir entre una instancia de tarea, especificada como un par problema/objetivo concretos (por ejemplo, el diagnóstico de un determinado paciente con unos síntomas específicos), y tareas, que especifican una familia de instancias de un determinado tipo, familias que pueden ser descritas a distintos grados de granularidad. Obsérvese que en la definición de la tarea no se ha incluido nada de cómo llevar a cabo la misma.
- **Métodos y Subtareas**. Los métodos son los medios que nos permiten llevar a cabo una tarea. Un método se define como un conjunto de subtareas que permiten la transformación del estado inicial de una tarea en el estado objetivo. Puede contener información adicional relacionada con la forma en que las subtareas son ordenadas en el tiempo. Las tareas obtenidas pueden ser aplicadas directamente o por medio de otros métodos, y por consiguiente, otras subtareas.
- **Estructuras de Tareas**. La ET es un árbol de tareas, métodos y subtareas aplicadas de forma recursiva hasta que las tareas que se alcancen puedan ser llevadas a cabo utilizando el conocimiento del que se dispone. Esta estructura nos permite especificar el hecho de que para una misma tarea se pueden utilizar más de un método.

El uso de ET nos permite definir procesos de razonamiento en el nivel de conocimiento sin hacer referencia alguna a la implementación, es decir, el nivel simbólico. En las figuras 2-A y 2-B se pueden apreciar las ET que corresponden con la tarea de diagnóstico y la de diseño.

Como conclusión podemos decir que las ET facilita el modelado del conocimiento ya que permite asociar las tareas con los métodos que la desarrollan y el conocimiento necesario para usar dicho método. Otro aspecto que potencia el uso de las ET es el hecho de que permite combinar distintos tipos de métodos para llevar a cabo una tarea. Finalmente, los problemas que surgían del uso de las TG quedan resueltos, ya que mediante la definición separada de los métodos y las tareas queda claro qué conocimiento queda asociado a cada elemento,

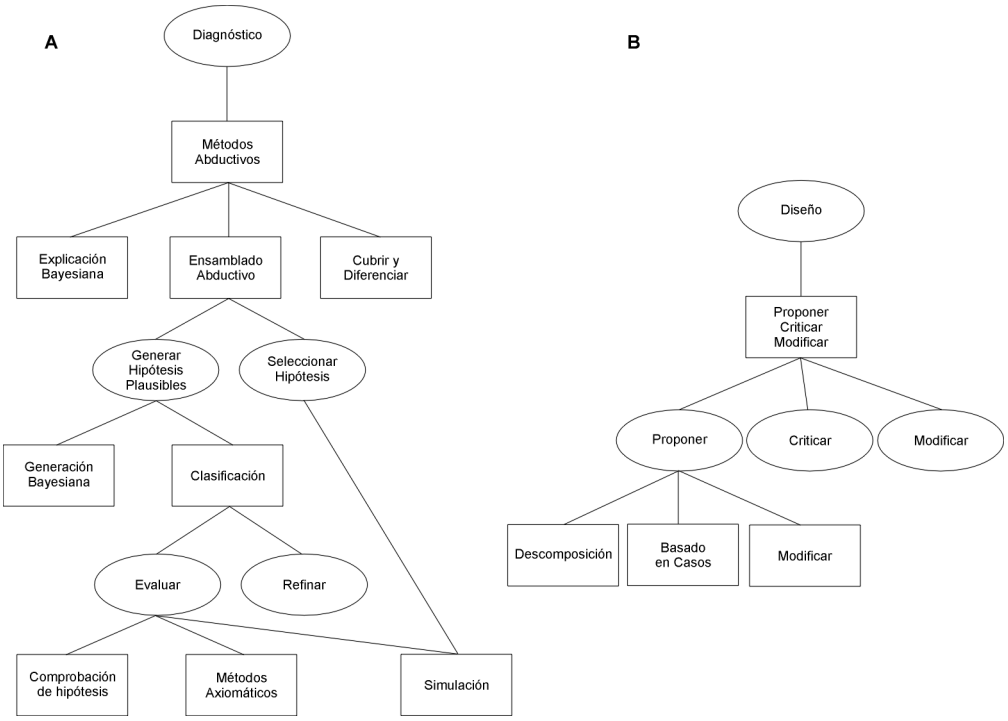

Figura 2: Estructura de Tareas para el Diagnóstico (A) y el Diseño (B). Los círculos representan las tareas y los rectángulos métodos.

y gracias a la ET, queda explicitado la relación entre subtareas y tareas. Además, la propia jerarquía inherente a la ET nos permite dejar claro a qué nivel de abstracción se define cada tarea, y consecuentemente, cuáles son las tareas más generales y cuales las más específicas.

La importancia de este marco de trabajo reside en que es el primer intento serio de descripción de un sistema inteligente en el nivel de conocimiento, permitiendo por tanto la definición de componentes genéricos que pueden ser "reutilizados"en diferentes dominios. Una de las carencias importantes de la aproximación de Chandrasekaran es que no se hace referencia en ningún momento al conocimiento del dominio necesario para las tareas, ya que sólo se describe la estructura de las tareas en el dominio. Por lo tanto, es necesario fundir esta aproximación con la aproximación de McDermott para tener una aproximación que aborde todos los aspectos necesarios para modelar el conocimiento: la estructura de tareas-métodos y la función que juega el conocimiento del dominio.

Aparte de las consideraciones anteriores, el único reproche que se le puede hacer es que no se indica ninguna metodología que dirigiese el proceso de modelado, pero como veremos en las siguientes secciones, los conceptos de tarea, método y su ensamblado en una estructura de tarea-método-descomposición formaron una base lo suficientemente sólida que permitió el desarrollo de metodologías que completasen el camino iniciado por Chandrasekaran.

8 Metodologías Basadas en el Modelado de Conocimiento

Conjuntamente a los trabajos anteriormente citados, otros investigadores realizaron trabajos en la misma línea, entre los que caben destacar a Clancey [Clancey, 1985] que propuso la clasificación heurística como un método genérico de resolución de problemas, Friedlan [Friedland and Iwasaky, 1985] con su refinamiento de esqueletos de planes, y por último, los componentes de experiencia de Steels [Steels, 1990]. Subyacente a todos estos trabajos está la convicción de que existen patrones recurrentes de comportamiento que los inge-

nieros del conocimiento implementan, bien explícitamente o bien implícitamente, en sus sistemas. Además, estos patrones de comportamiento describen los requisitos de conocimiento que necesita el sistema y, por lo tanto, pueden dirigir la adquisición del mismo.

Los trabajos desarrollados en la segunda mitad de la década de los ochenta, y a la vista del trabajo de McDermott[McDermott, 1988], se englobaron dentro de los que se denominaron métodos con limitación de roles. Estos métodos aportaron la ventaja de que proporcionaban una manera de clarificar el conocimiento usado en la resolución del problema y proporcionaban una serie de expectativas que podían dirigir la adquisición del contenido del conocimiento requerido por las aplicaciones [Musen, 1992]. De esta forma, se podía afirmar que si todos los roles de un método de resolución de problemas estaban cubiertos en la base de conocimiento, ésta está completa, en caso contrario, diríamos que está incompleta (si falta algún rol por cubrir) o sobredimensionada (si existen elementos en la base de conocimiento que no están asociados a ningún rol).

Sin embargo, los métodos con limitación de roles presentaban dos problemas importantes. Por un lado, al asumir que el comportamiento del sistema puede ser descrito en términos abstractos, es decir, sin hacer referencia a la implementación, se impide el modelado de los sistemas en los que consideraciones específicas sobre el dominio pueden alterar la estrategia de control del sistema. Por otro lado, al ser bastante genéricos resulta difícil aplicarlos a problemas concretos, ya que dicha generalidad hace que las condiciones requeridas por dichos problemas no se cumplan en el modelo genérico.

Estos problemas provocaron un replanteamiento de estado de la Ingeniería del Conocimiento que dio como resultado la aparición de nuevas metodologías de desarrollo de SBC. Estas nuevas metodologías, que se basaron en las ideas resultantes de los trabajos anteriores, permiten realizar el análisis del sistema en el nivel de conocimiento, ofreciendo la posibilidad de especificar el problema a diferentes niveles de granularidad, con lo cual se potencia la idea de la reutilización de componentes de conocimiento. Por otro lado, estas nuevas metodologías ofrecen un ciclo de vida completo para el proceso de desarrollo (al igual que existe en la ingeniería del software), proporcionado pautas a seguir desde el análisis hasta la implementación final del sistema. A continuación ofreceremos una breve exposición de las tres metodologías más importantes: CommonKADS [Schreiber *et al.*, 1999], MIKE [Angele *et al.*, 1998] [Angele *et al.*, 1996] y PROTÉGÉ-II [Eriksson *et al.*, 1995]. Finalizaremos dicha exposición citando un conjunto de metodologías, que si bien no han llegado al estatus de las anteriores, ofrecen resultados bastante significativos.

8.1 CommonKADS

KADS es una metodología completa para el desarrollo de SBC. Ha sido el resultado de un trabajo que ha durado aproximadamente una década dentro de dos proyectos SPRIT. En sus inicios, KADS se centraba en el problema del cuello de botella de suponía la adquisición de conocimiento, para posteriormente convertirse en una metodología completa para el desarrollo de SBC [Schreiber *et al.*, 1993]. En la actualidad, CommonKADS, nombre que recibe la evolución de KADS, cubre la gestión del proyecto, el análisis organizacional y los aspectos relativos a las Ingenierías del Software y Conocimiento relacionados con el desarrollo de SBC.

En CommonKADS podemos ver reflejadas tres ideas que han emergido, no sólo de la experiencia en la Ingeniería del Conocimiento, sino también del campo de la Ingeniería del Software en general. Estas tres ideas se pueden concretar en tres conceptos: modelado, reutilización y gestión del riesgo.

El principal producto que resulta de la aplicación de CommonKADS es el conjunto de modelos. Este conjunto de modelos se puede considerar una agrupación estructurada de conocimiento que refleja todos aquellos aspectos importantes para que el SBC tenga éxito dentro de un contexto organizacional determinado. Para reflejar los diferentes aspectos del contexto en el cual se quiere implantar el SBC, CommonKADS ofrece seis modelos [de Hoog *et al.*, 1994]: organización, tareas, agentes, comunicación, conocimiento y diseño. Todos estos modelos están relacionados entre sí y pueden ser configurados gracias a unas plantillas que la metodología ofrece para su confección. Los tres primeros modelos describen el

contexto en el cual se va a desenvolver el SBC. Los modelos de experiencia y agentes proporcionan los requisitos de entrada que guiarán la implementación del sistema a través del modelo de diseño. Como se puede deducir, cada una de estas agrupaciones intentan responder a cada una de las preguntas claves en el desarrollo de un SBC: ¿Es un SBC la solución idónea para el problema que se quiere resolver?, pregunta que se puede responder por la descripción del contexto dado por los tres primeros modelos; ¿Cuál es la naturaleza y la estructura tanto del conocimiento como de la comunicación utilizada?, cuya respuesta se puede encontrar en el modelo de experiencia y en el modelo de comunicación; y finalmente, ¿Cómo debe ser implementado el conocimiento?, pregunta que intenta esclarecer el modelo del diseño.

Mención especial al modelo de conocimiento. Este modelo, que está descrito en [Wielinga *et al.*, 1994], describe el conocimiento que tiene un determinado agente y que es relevante para la consecución de una determinada tarea, además de describir la estructura del mismo en función de su uso. Obviamente, este modelo se hace en el nivel de conocimiento, sin hacer referencia a aspectos de implementación. Para poder llevar a cabo este modelado de los distintos papeles que puede jugar el conocimiento, éste está distribuido en tres categorías disjuntas:

- **Conocimiento de tareas**. Describe de una forma recursiva la descomposición de una tarea de alto nivel en varias subtareas. El conocimiento sobre una tarea se divide en dos partes: por una lado la tarea, que sirve para especificar qué es lo que implica la aplicación de la tarea ya que define su objetivo en términos de los roles de entrada y de salida; por otro lado, está el método de la tarea, que define el cómo se lleva a cabo dicha tarea, indicando en qué subtareas se descompone y en qué orden deben de ser procesadas (control).

- **Conocimiento del dominio**. Especifica los hechos y asunciones que necesita el proceso de razonamiento para llevar a cabo su cometido en el dominio de la aplicación. El conocimiento del dominio puede ser estructurado en una serie de modelos del domino que proporcionen una visión coherente de las distintas partes del dominio de la aplicación. Por lo tanto, en este apartado se va a especificar la forma, estructura y contenido del conocimiento relevante para la aplicación. La forma y la estructura constituyen lo que se denomina la ontología del dominio.

- **Conocimiento sobre inferencias**. Describe los procesos primitivos de razonamiento que tienen lugar en una aplicación, así como los roles de conocimiento que son usados por las inferencias. Obviamente, estos roles de conocimientos están relacionados con elementos del conocimiento del dominio. Hay que tener en cuenta, que las inferencias son consideradas primitivas respecto a un modelo de experiencia determinado, ya que en otros modelos de experiencia la misma inferencia puede ser una tarea descomponible.

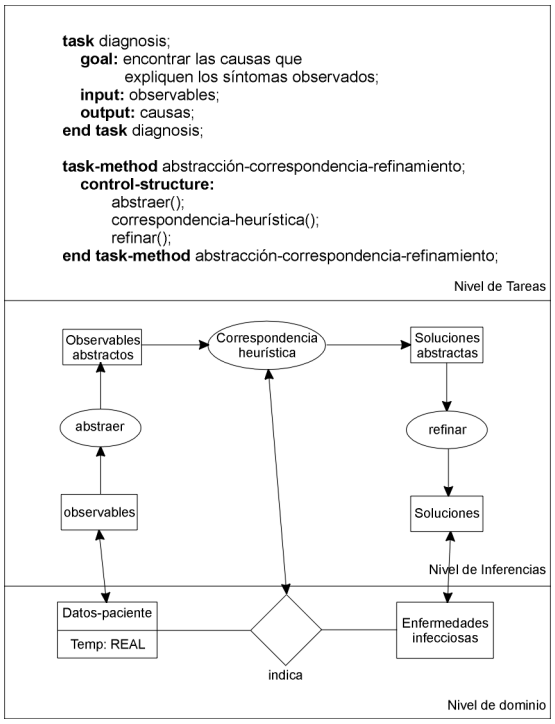

Figura 3: El Modelo de Conocimiento en CommonKADS.

CommonKADS propone el lenguaje CML (Conceptual Modelling Language) [Anjewierden, 1997], para materializar la especificación del modelo de conocimiento. Este lenguaje permite la definición de todos los

elementos anteriormente citados, así como la estructuración del conocimiento en las partes anteriormente citadas. Además propone el uso de una notación gráfica, que permite no sólo la definición de las estructuras de tareas (relación tareas-subtareas), sino que también permite la definición de la ontología y los conceptos del dominio y la definición de la dependencia de los datos entre las inferencias a través de las estructuras de inferencias (figura 3). Aparte de CML, también se propone el uso de $(ML)^2$ [Harmelen *et al.*, 1991] que está más orientado hacia la formalización del modelo. Esta descomposición del nivel de conocimiento en el conocimiento específico del dominio por un lado, y de las tareas y las inferencias por otro, favorece la reutilización del conocimiento en dos niveles. Al haber definido un conjunto de inferencias elementales independientes de la aplicación se permite que estas sean utilizadas en otros procesos de resolución de problemas. En este sentido es importante tener en cuenta el trabajo de [Aben, 1993] donde se catalogan y clasifican el conjunto de inferencias básicas que pueden ser utilizadas en CommonKADS. El otro nivel de reutilización lo establece el conocimiento del dominio, que al ser definido independientemente de las tareas, puede ser reutilizado por otro conjunto de tareas definidas sobre el mismo dominio. Para favorecer la reutilización del conocimiento del dominio CommonKADS permite definir las ontologías a distintos niveles de generalización y distintos puntos de vista, lo que abre el abanico de posibles adaptaciones a otras aplicaciones.

Otro de los aspectos importantes que introdujo CommonKADS fue la definición de un marco de trabajo para la gestión y planificación del proyecto. CommonKADS define un ciclo de vida para el desarrollo del proyecto basado en un modelo en espiral como el propuesto por [Boehm, 1988]. El modelo en espiral que plantea CommonKADS se basa en los siguientes principios [Schreiber *et al.*, 1999]:

- La planificación del proyecto se centra principalmente en los productos y las salidas que tienen que producirse como resultado, más que un conjunto de actividades o fases.

- La planificación se realiza de una forma adaptativa a lo largo de un serie de ciclos en espiral, que están dirigidos por una valoración sistemática de los riesgos del proyecto.

- El control de calidad es una parte más de la gestión del proyecto, ya que la calidad está integrada en el desarrollo del SBC por medio de la metodología.

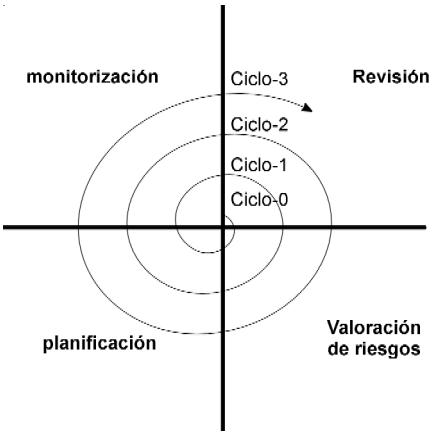

Figura 4: El Ciclo de Vida en CommonKADS.

Estos principios están garantizados por un lado, por el conjunto de modelos, y por otro, por el ciclo de vida en espiral. Este ciclo de vida consta de cuatro fases (figura 4):

- **Revisión**. Es el primer paso de cada ciclo y en el se revisa el estado actual del proyecto y se establecen los objetivos principales que se quieren cubrir en el ciclo en cuestión.

- **Valoración de riesgos**. Las líneas generales del proyecto establecidas en el paso anterior sirven de entradas para esta fase. Su función principal es la identificación y valoración de los principales obstáculos que nos podemos encontrar para la consecución exitosa del proyecto, así como las acciones que se deben tomar para minimizar dichos riesgos.

- **Planificación**. Una vez obtenida una visión clara de los objetivos que hay que cubrir, los riesgos que se pueden presentar y las acciones que hay que tomar, hay que realizar una planificación del trabajo a realizar. En dicha planificación hay que establecer la distribución de la carga del trabajo en términos de qué tareas hay que reali-

zar, una temporalización de dichas tareas, la distribución de los recursos, etc.

- **Monitorización**. Es la última fase del ciclo y está constituida por el desarrollo propiamente dicho. El trabajo realizado en esta fase está controlado y dirigido por el director del proyecto. Para determinar el grado de cumplimiento de los objetivos se requieren reuniones con los agentes implicados en el proyecto (usuarios, administradores, expertos, ..). El resultado de dichas reuniones se utilizará como entrada del proceso de revisión del siguiente ciclo.

Como se puede observar, la metodología CommonKADS abarca todo los aspectos del desarrollo de un SBC, desde los análisis iniciales que sirven para identificar problemas y para establecer la idoneidad de la solución basada en un SBC, hasta la implementación del mismo, proporcionando un marco de trabajo donde llevar a cabo la gestión del proyecto. También hay que resaltar que el modelado del conocimiento posibilita la definición de componentes reutilizables, tanto en el nivel de tareas como en el de conceptualización del dominio, como resultado de la agrupación de las ideas que surgieron después del trabajo de Newell [Newell, 1982]. Todo esto hace que CommonKADS haya sido adoptado no sólo en Europa, donde se ha convertido en un estándar de facto, sino que también empieza a ser utilizado en el resto del mudo.

8.2 MIKE

MIKE (Model-based and Incremental knowledge Engineering) [Angele *et al.*, 1998] [Angele *et al.*, 1996] proporciona una metodología para el desarrollo de SBC que cubre todos los aspectos del proceso, desde la adquisición de conocimiento hasta su diseño e implementación. Al igual que CommonKADS, MIKE se desarrolla por medio de un ciclo de vida en espiral, al que se le han añadido determinados elementos que permiten unir el prototipado con un proceso de desarrollo incremental y sostenible dentro del paradigma del modelado.

El proceso de desarrollo se puede resumir en cuatro fases aplicadas de forma cíclica: *adquisición de conocimiento*, *diseño*, *implementación y evaluación*. Una de las aportaciones principales de esta metodología es la de detallar el proceso a realizar en cada una de las fases (figura 5), poniendo especial hincapié en la fase de adquisición.

De esta forma, la fase de adquisición empieza con la *extracción de conocimiento*, con la que se obtienen descripciones informales sobre el conocimiento del dominio y del proceso de resolución. La extracción de dichas descripciones puede realizarse a través de entrevistas estructuradas con los expertos. El resultado de esta fase se resume en el *modelo de elicitación*, constituidos por los protocolos de conocimiento que quedan almacenados en los *nodos-protocolos*, en donde la información es almacenada en lenguaje natural. En la metodología se propone para la creación de este modelo semiformal, el uso de un sistema hipermedia como MEMO (Mediating Model Organization) [Neubert, 1993].

Una vez terminada la fase de extracción se entra en la *interpretación*. En esta fase, todas las estructuras identificadas en la fase anterior pasan a ser descritas en un lenguaje más fijo y restringido. Esta nueva representación recoge más formalmente toda la información sobre la estructura, es decir, la dependencia entre los datos y las inferencias, dejando la descripción de las inferencias en lenguaje natural. Toda esta información queda recogida en el *modelo de estructura*, que es el resultado de esta fase. Este modelo informal está compuesto de varios contextos:

- **Contexto de actividad**. Está formado por todos los *nodos de actividad*, que representan un paso dentro del proceso de resolución. Junto a estos nodos de actividad se colocan los *nodos de refinamiento*, que indican la descomposición estructural de una actividad determinada. Como podemos ver, este contexto refleja la descomposición jerárquica de tareas que existe en el dominio. El contenido de cada nodo actividad se corresponde con una descripción informal de la especificación de la actividad.

- **Contexto de ordenación**. Sirve para indicar cómo quedan relacionadas las actividades por medio de la especificación del control necesario para llevarla a cabo.

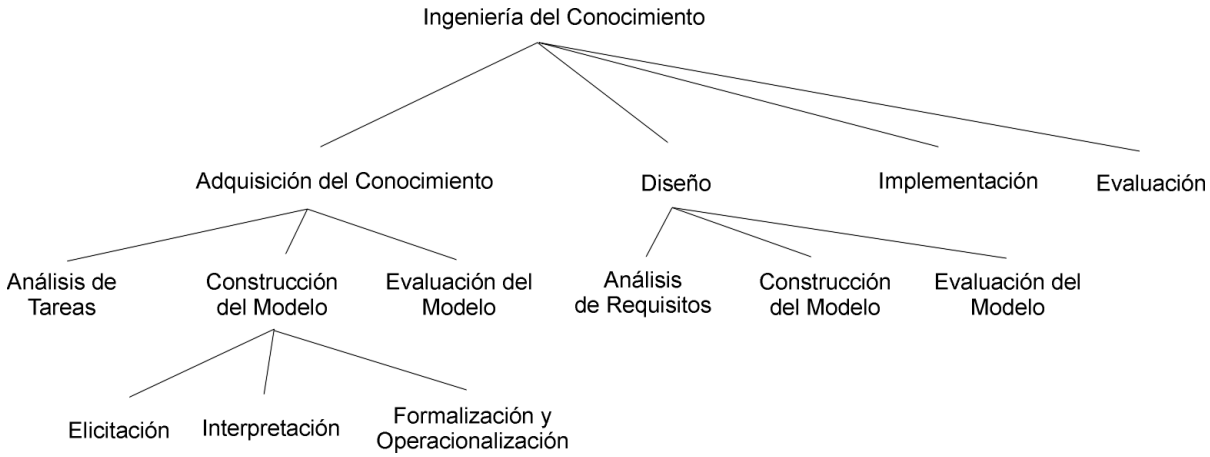

Figura 5: Fases en el Ciclo de Vida en MIKE.

- **Contexto de conceptos**. Está formado por nodos que representan los conceptos del dominio y, por lo tanto, describen la estructura estática del mismo. En este modelo se incluyen enlaces entre los nodos que sirven para indicar como dichos conceptos están organizados mediante asociación, agregación y generalización. En este contexto se propone utilizar un notación gráfica derivada de OMT [Rumbaugh *et al.*, 1991].
- **Contexto de flujo de datos**. Presenta una visión del domino en un nivel concreto de la jerarquía de actividades. En esta visión, los nodos-actividad se enlazan con los nodos-conceptos mediante los *enlaces de flujo de datos*.
- **Contexto de requisitos no funcionales**. Se utiliza para describir cuáles son los requisitos no funcionales (de mantenibilidad y eficiencia) que debe cumplir el sistema.

El modelo de estructura recoge, de esta forma, todos los aspectos funcionales y no funcionales del dominio. Este modelo, junto con el de elicitación, forman las bases para establecer la comunicación entre el experto y el ingeniero del conocimiento, pudiendo incorporar al experto en el proceso de construcción y evaluación del modelo. Además, todos los elementos del modelo de estructura están enlazados con sus correspondientes partes del modelo de elicitación por medio de *enlaces de elicitación*, permitiendo el seguimiento de la evolución de los modelos. Como se puede observar se sigue manteniendo el principio de separación entre el conocimiento del dominio y las tareas del que se desarrollan en el mismo.

Una vez construido el modelo de estructura, se entra en la fase de formalización y operacionalización. En esta fase, se intenta crear un modelo más formal que el de estructura mediante la utilización de un lenguaje formal, KARL (Knowledge Acquisition Representation Languaje) [Fensel, 1995]. La característica principal de este lenguaje es que combina la descripción conceptual de un SBC con una especificación formal y ejecutable. De esta forma, se pueden especificar formalmente, eliminado cualquier vaguedad e imprecisión, las inferencias detectadas en la fase anterior. Al igual que ocurre en los modelos anteriores, los elementos de este modelo se asocian mediante *enlaces de formalización* con los elementos correspondientes en el modelo de estructura. El hecho de que el modelo resultante, el modelo KARL¸ sea ejecutable (bajo ciertas restricciones), permite una evaluación del modelo por parte del experto. Con la obtención del *modelo KARL*, termina la fase de *adquisición del conocimiento quedando identificados*, no sólo la estructura del dominio, sino también los requisitos funcionales y no funcionales.

El modelo formal representado en el modelo KARL es la entrada de la siguiente fase, el *diseño*. En esta fase se consideran los requisi-

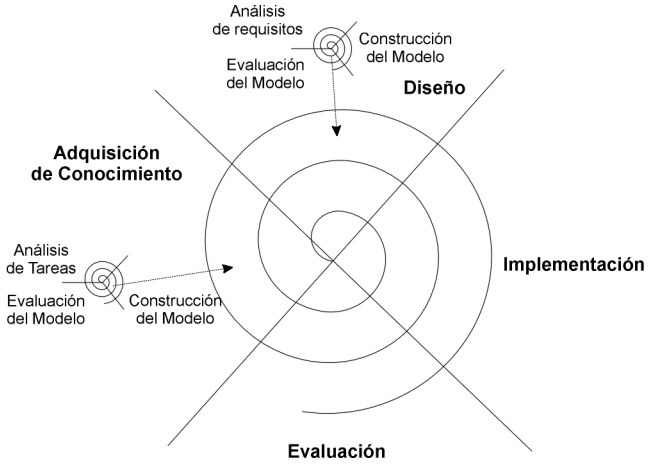

Figura 6: Ciclo de Vida en MIKE.

tos no funcionales (mantenibilidad, eficiencia y restricciones impuestas por la plataforma hardware y software elegida) y se genera un nuevo modelo más detallado, el *modelo del diseño*, especificado en el lenguaje DesignKARL [Landes, 1994], que es una extensión de KARL a la que se le añade la capacidad de expresar algoritmos y estructuras de datos. Esta fase equivale a lo que en la Ingeniería del Software se realiza mediante el diseño detallado. Una vez obtenido el modelo del diseño, se pasa a la fase de *implementación*, en la que se materializa en la plataforma hardware y software elegida. Por último, la última fase del proceso de desarrollo termina con la evaluación, en la que se vuelve a requerir la intervención del experto.

Como ya hemos mencionado, todos estos pasos son realizados de una forma cíclica, basada en el modelo en espiral de Boehm [Boehm, 1988]. De esta forma, los resultados de la evaluación sirven de entrada para la depuración de los modelos en una nueva fase de adquisición de conocimiento. Sin embargo, MIKE aporta una modificación al método clásico de Boehm, ya que las subfases en las que se dividen algunas de las fases anteriores se realizan también siguiendo un modelo en espiral (figura 6).

Como podemos ver, MIKE es una potente metodología que integra formalismos de representación informales y formales con un desarrollo por prototipado. Comparada con CommonKADS, se puede decir que MIKE ofrece una forma sencilla de transición entre el resultado de la adquisición de conocimiento, el modelo de conocimiento y el modelo de diseño, a diferencia de CommonKADS donde estas transiciones no están lo suficientemente especificadas. Sin embargo, a favor de CommonKADS hay que decir que ofrece soporte para un análisis de contexto más amplio, permitiendo evaluar el impacto de la utilización de un SBC en una gran organización. Este análisis de contexto, no está tratado en MIKE, que sólo se centra en el desarrollo del modelo de conocimiento, aunque es uno de los caminos en los que se está trabajando.

8.3 Protégé-II

PROTÉGÉ-II [Puerta *et al.*, 1992] [Eriksson *et al.*, 1995] [Tu *et al.*, 1995] fue el resultado de un proyecto cuyo principal objetivo era resolver las limitaciones del trabajo realizado en la construcción de la herramienta PROTÉGÉ [Musen, 1989a] [Musen, 1989b]. PROTÉGÉ es una metaherramienta que permite a los desarrolladores generar herramientas para la adquisición de conocimiento para distintos dominios de aplicación, pero que se basen en la utilización del método de resolución de problemas con limitación de roles denominado *refinamiento de patrones de planes episódicos*. El uso de un solo método de resolución de problemas hacía imposible abordar problemas que no pudieran ser resueltos por dicho método. Para resolver estas limitaciones se desarrolla PROTÉGÉ-II, que está diseñada para soportar una Ingeniería del Conocimiento orientada a tareas más general que PROTÉGÉ. PROTÉGÉ-II proporciona un entorno para Ingeniería de Conocimiento en el que el desarrollador puede especificar tareas, y puede seleccionar métodos de resolución de una librería de métodos reutilizables. Para permitir el modelado, se distinguen los siguientes elementos:

- **Tareas**. Es un problema en el mundo real, localizado dentro de una organización (*tarea aplicación*), o una especificación abstracta de objetivos independientes del dominio que tienen que ser alcanzados (*clases de tareas, o tarea*). La descripción de una tarea es una especificación abstracta del problema que tiene que ser resuelto por el método. Debe incluir definiciones abstractas de los datos disponibles (entradas) y de las solución (salida) y las relaciones que se deben mantener entre ellos. Aun-

que en la especificación de una tarea no se indica cómo se resuelve, ésta condiciona la selección del método para su resolución.

- **Método**. Un método es un modelo independiente del dominio, que indica cómo se resuelve el problema especificado por la tarea. Además de la especificación de la entrada y la salida de la tarea, el método posee una definición abstracta de los roles que juega el conocimiento del dominio en la resolución del problema. La selección de un método depende de factores que van más allá de la especificación de la tarea, como la disponibilidad de la experiencia, requisitos de espacio y tiempo de computación, y la compatibilidad con el resto de métodos que cooperan en la resolución del problema. Los métodos pueden delegar problemas a subtareas que, a su vez, pueden ser resueltas por otros métodos, formando un árbol de tareas-métodos-subtareas.

- **Mecanismos**. Los mecanismos se pueden considerar como métodos primitivos que no pueden ser descompuestos en otras subtareas. Por lo tanto, pueden ser considerados como cajas negras que no admiten descomposición.

Como se puede deducir, estos elementos se asemejan mucho a las estructuras de tareas propuestas por Chandrasekaran, ya que disponen de tareas que pueden ser resueltas por diferentes métodos, que a su vez pueden delegar en otras subtareas que pueden ser descompuestas. Esta descomposición termina cuando se alcanzan los mecanismos. Esta estrategia de descomposición ayuda a la reutilización de componentes de dos formas: por un lado, los mecanismos, al desarrollar funciones muy definidas, pueden ser aplicables a varios dominios (como las inferencias en CommonKADS); y por otro, la existencia de métodos descomponibles hace más flexible la reutilización de conocimiento, ya que permiten la utilización de métodos y mecanismos alternativos para la resolución de sus subtareas.

Además de estos elementos relacionados con la estructura de tareas-métodos, en PROTÉGÉ-II se distinguen distintos tipos de ontologías. La *ontología del dominio* incluye una descripción declarativa del dominio, y representa la conceptualización del dominio de la aplicación. La *ontología de los métodos* define los conceptos y relaciones que utilizan los métodos, y sirve de marco para especificar sus entradas y salidas. Estas ontologías de los métodos se corresponden con la terminología genérica utilizada por el conjunto de roles del método. Para que los métodos de resolución de problemas puedan ser reutilizables por diferentes dominios, deben ser definidos en términos independientes del dominio. De la misma forma, para que las ontologías que definen los términos y relaciones de un determinado área de aplicación puedan ser reutilizadas por diferentes tareas y métodos, deben de ser definidas de forma independiente de los métodos de resolución. Estos requisitos de reutilización pueden provocar la aparición de diferencias entre la ontología del método y la del dominio en el que se va a utilizar dicho método. Para poder tratar con este problema de la interacción entre estas ontologías, PROTÉGÉ-II introduce el concepto de *ontología de la aplicación*, que sirve para extender la ontología del dominio con conceptos y relaciones propios de la ontología del métodos. De todas formas, aunque se haya definido una ontología intermedia entre el método y el dominio, todavía pueden aparecer problemas cuando se intenta adaptar una ontología de la aplicación con la correspondiente al método elegido. Estos problemas pueden ser:

- **Diferencias terminológicas**, que ocurren cuando los mismos términos y relaciones reciben distintos nombres en las dos ontologías. Para solventar este problema, PROTÉGÉ-II permite definir *correspondencias de renombrado*, que se utilizan para establecer la traducción de términos.

- **Diferencias semánticas**, ocurren cuando el concepto de partida resulta de una transformación semántica del concepto que le corresponde (por ejemplo, la temperatura definida en grados Celsius o en grados Farenheit). La solución de este problema pasa por definir *correspondencias de filtrado*.

- **Diferencias representacionales**. Es la diferencia más compleja que se puede presentar en la estructura de una ontología. Ésta aparece cuando un concepto está definido en una ontología por medio de un conjunto de parámetros con sus valores, y

otra ontología representa el mismo concepto por medio de una jerarquía de subconceptos.

Por lo tanto, y partiendo de la base de que PROTÉGÉ-II tiene una librería de métodos genéricos, la construcción de un modelo de conocimiento se puede hacer en tres pasos: (1) formulación de la ontología del método, incluyendo el conjunto de roles de conocimiento que utiliza, (2) definición de la ontología de la aplicación independientemente de la del método y reutilizando alguna ontología del dominio previamente definida, y (3) formulación de las relaciones de correspondencia entre las dos ontologías (aplicación y método). En la figura 7 se puede ver de forma esquemática las relaciones entre las distintas ontologías.

Aparte de la definición de nuevos modelos por reutilización, una de las características importantes de PROTÉGÉ-II es que facilita la generación de herramientas para la adquisición de conocimiento de forma independiente de los métodos que se vayan a utilizar. La generación de la herramienta de adquisición de conocimiento se hace de manera gráfica y dirigida por la ontología del dominio. En PROTÉGÉ-II se propone un análisis de la adquisición de conocimiento que presenta las siguientes fases:

1. Identificación de los usuarios potenciales de la herramienta de adquisición.

2. Segmentación de la base de conocimiento y determinación de qué parte va a ser adquirida por la herramienta.

3. Definición de un lenguaje en el que expresar el conocimiento (puede ser un lenguaje gráfico).

4. Especificación de la semántica para dicho lenguaje, además de las semántica denotacional que describe la generación de la base de conocimiento.

De esta forma, el análisis de la adquisición de conocimiento se considera una fase más en el desarrollo de una herramienta de adquisición de conocimiento, que una vez terminada, permite el desarrollo de la herramienta, proceso que puede ser simplificado gracias a PROTÉGÉ-II.

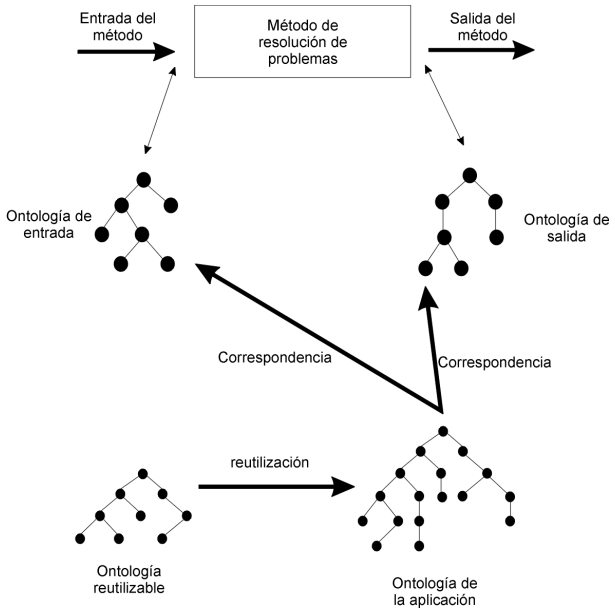

Figura 7: Distintas Ontologías en PROTÉGÉ-II y sus relaciones.

Como se ha descrito, PROTÉGÉ-II define un marco que permite la definición de nuevos modelos de conocimiento por reutilización de componentes definidos en una librería. Este nuevo modelo puede ser creado por reconfiguración de alguno de los componentes de la librería o por combinación de varios componentes de la misma. La diferencia principal de esta aproximación con otras basadas en la reutilización, es que está orientada a la minimización del coste de programación, a diferencia de otras aproximaciones como puede ser la de los *componentes de experiencia* [Steels, 1990]. Sin embargo, una de las debilidades más importantes es que no aporta la definición de un ciclo de vida completo para el desarrollo de aplicaciones al estar más centrado en la fase de adquisición de conocimiento. Esta falta de definición de un ciclo de vida trae consigo un problema importante, y es que al carecer de un análisis de contexto, se complica el uso de la herramienta para grandes aplicaciones que tengan que trabajar en un entorno determinado.

9 Otras Metodologías

Aunque las anteriores metodologías han sido las que más difusión han tenido, existen bastantes trabajos que han dado lugar a metodologías o

marcos de trabajo que han intentado cubrir aspectos concretos de los entornos donde fueron desarrolladas, mostrando su utilidad en bastantes aplicaciones, pero que de algún modo han sido oscurecidas por los resultados y aceptación de las anteriores. En los siguientes párrafos daremos una breve descripción de este conjunto de trabajos.

Una de las metodologías más interesantes es VITAL [Shadbolt *et al.*, 1993] [Domingue *et al.*, 1993] que al igual que CommonKADS se desarrolló con la colaboración de varias empresas y bajo un proyecto ESPRIT. Al igual que CommonKADS se basa en la idea de que el desarrollo de un SBC se ve favorecido por la adopción de una metodología de desarrollo estructurada. Esta metodología produce cuatro productos como resultado de su aplicación: *Especificación de requisitos*, donde se describe la funcionalidad que se esperan del SBC final; *Modelo Conceptual*, equivalente al modelo de experiencia propuesto en CommonKADS; *Modelo de Diseño*, basado en la idea de MIKE en el sentido de que se describe en dos fases: Modelo de diseño funcional, independiente de la implementación, y *Modelo de diseño técnico*, que depende de la implementación; por último, el *Código Ejecutable*, que engloba todo los componentes software que pueden ser objeto de un mantenimiento posterior. Todo estos productos están dirigidos por una metodología para la dirección del proyecto que especifica el ciclo de vida del mismo. Otro aspecto importante es que, al igual que MIKE, ofrece lenguajes formales e informales para representar los distintos niveles de especificación que podemos encontrarnos en el nivel de conocimiento. Por lo tanto, esta metodología que ofrece un ciclo de vida estructurado para la dirección del proyecto que facilita el mantenimiento, depuración y reutilización de los productos obtenidos.

KSM [Cuena and Molina, 1994] [Cuena and Molina, 1997] es otro trabajo que propone un entorno software para ayudar a la construcción, operacionalización y reutilización de modelos de conocimiento. Para definir un modelo hay que hacerlo desde tres perspectivas: (1) la perspectiva del área de conocimiento, que es un descripción modular de las ontologías implicadas en el modelo, (2) la perspectiva de tareas, similar a la capa de tareas de CommonKADS y (3) la perspectiva del vocabulario, que engloba los términos básicos que son compartidos por otros módulos de conocimiento. Al igual que CommonKADS y PROTÉGÉ-II, KSM proporciona elementos computables que implementan los mecanismos básicos de resolución de problemas, que se denominan primitivas de representación. De esta forma, se ofrece una forma sencilla de operacionalizar el modelo en el nivel de conocimiento, ya que sólo haría falta asociar las primitivas de representación con sus respectivas áreas de conocimiento. Sin embargo, al igual que en VITAL, no existe un análisis de contexto, además de no ofrecer directrices para la dirección del proyecto.

Otras aproximaciones que han tenido cierta difusión son: DESIRE (framework for DEsign and Specification of Interacting REasoning components) [Brazier *et al.*, 1997] más orientada hacia los sistemas multiagentes proporcionando modelos genéricos reutilizables para determinados tipos de aplicaciones multiagentes y que permiten su implantación de diferentes dominios. Incluye técnicas de verificación y validación para las propiedades estáticas de sistemas que desarrollan razonamientos complejos. COMMET (The COMponential METhodology) [Steels, 1993] es un entorno de trabajo que soporta el desarrollo de SBC en tres niveles: nivel de conocimiento, nivel de ejecución y nivel de código ofreciendo posibilidad de reutilizar componentes en cada uno de los niveles.

Se pueden citar más marcos de trabajo, pero como podemos observar no aparecen aportaciones importantes que no hayan sido abordadas por otras metodologías. De todas formas, está pequeño barrido puede servir para resaltar la importancia que está adquiriendo la Ingeniería del Conocimiento y que está tomando cuerpo de verdadera ingeniería, ya que tiene metodologías de trabajo estructuradas y bien definidas que han sido aplicadas a numerosos problemas reales con éxito.

10 Conclusiones

Como podemos observar a través de este análisis histórico, la Ingeniería del Conocimiento ha alcanzado la madurez necesaria como para que las metodologías planteadas sean utilizadas a nivel industrial. Sin embargo, a pesar de los

cambios en los paradigmas sobre los que se han sustentado todas las metodologías, existen dos aspectos comunes a todas ellas:

- Todas las metodologías consideran sumamente importante el análisis de viabilidad de la aplicación, entendido este dentro de un contexto organizacional que va más allá del sistema objeto del desarrollo.
- La propia naturaleza de los SBC (el motor de inferencia separado de la base de conocimiento) hace que todas las metodologías planteen la fase de desarrollo siguiendo un modelo de desarrollo incremental, en el que se hace especial hincapié en la extensión gradual de la base de conocimiento.

Partiendo de estos dos aspectos comunes, las distintas metodologías han ido adaptándose a la evolución que se ha ido produciendo en el campo de la ingeniería del software. Así, al igual que en la Ingeniería del Software, la Ingeniería del Conocimiento ha pasado desde el modelo clásico de cascada al modelo de ciclo de vida en espiral apoyado en la gestión de riesgos. También se puede apreciar como una de las principales ventajas que aportaron las técnicas orientadas a objetos en la Ingeniería del Software, como es el hecho de la reducción del salto semántico entre el diseño del sistema y su implementación, permitiendo que las estructuras se mantengan a lo largo del proceso de desarrollo, también se haya trasladado a la Ingeniería del Conocimiento en aspectos tales como el paradigma del modelado y metodologías que se basan en un diseño que preserve la estructura por medio de la reutilización [Benjamins and Aben, 1997]. Esta aproximación de estas dos disciplinas ha llevado a que la metodología más extendida en la Ingeniería del Conocimiento CommonKADS [Schreiber *et al.*, 1999] utilice UML (Unified Modeling Language [Booch *et al.*, 1997]) para la notación gráfica de varios de sus modelos. Sin embargo, la utilización a nivel industrial de las metodologías ha puesto de manifiesto ciertas carencias que empiezan a ser subsanadas a través de distintas extensiones. Entre estas extensiones podemos citar RT-CommonKADS [Palma-Méndez, 1999], en la que se incorporan elementos que hacen posible el modelado de ciertas características propias de los dominios con una fuerte componente de tiempo real, y MASCommonKADS [Iglesias-Fernández, 1997], en la cual se plantea la metodologías desde un punto de vista más orientado hacia el modelado de agentes.

Sin embargo, en la actualidad estamos viviendo un nuevo auge en la disciplina de la Ingeniería del conocimiento. Este auge es debido principalmente a dos causas: el impulso que han sufrido los Sistemas Multiagentes y la Gestión del Conocimiento. La primera de estas causas obedece a la respuesta que intenta dar la Ingeniería del Conocimiento al nuevo paradigma que se abre en la Ingeniería del Software, el Análisis Orientado a Agentes. En este sentido, podemos encontrar un buen número de propuestas, que surgiendo del área de la Ingeniería del Conocimiento se han orientado más al modelado de sistemas multiagentes. Entre estas propuestas podemos citar: ARCHON [Cockburn and Jennings, 1996], aplicada a sistemas industriales, DESIRE [Brazier *et al.*, 1997], MADE [O'Hare and Wooldridge, 1992] o ADEPT [Jennings *et al.*, 1996], aplicada a procesos de gestión de negocio, basada en GRATE [Jennings *et al.*, 1992].

El otro aspecto que ha influido en este nuevo impulso a la Ingeniería del Conocimiento es la Gestión de Conocimiento, entendida esta como *"el conjunto de procedimientos, reglas y sistemas destinados a captar, tratar, recuperar, presentar y transmitir los datos, informaciones y conocimientos dentro de una organización."* [Maestre-Yenes, 2000]. Al ser el conocimiento el principal elemento, cualquier intento de implantar una política de Gestión de Conocimiento en una organización tiene que pasar por una fase de identificación del mismo. Es en este punto donde se encuentran las dos disciplinas, y donde la experiencia adquirida a lo largo de los años por la Ingeniería del Conocimiento ha sido de gran ayuda. Como ejemplo de esta sinergia tenemos los modelos de organización, agentes y tareas de CommonKADS que pueden ser utilizados como punto de partida para la Gestión de Conocimiento [Schreiber *et al.*, 1999].

Como podemos ver, la Ingeniería del Conocimiento es un área en continua expansión, que no sólo se centra en el desarrollo de SBC sino que abarca otras áreas como el Modelado de sistemas Multiagentes y la Gestión del Conocimiento. Sin embargo, todavía existen aspectos

que no están lo suficientemente explotados y racionalizados como es el paso del diseño o modelo a la implementación, campo que está totalmente abierto y en el que los Ingenieros del Software tienen mucho que decir.

Referencias

[Aben, 1993] M. Aben. CommonKADS inferences. Technical Report ESPRIT Project P5248 KADS-II/M2/TR/UvA/041/1.0, Unviersity of Amsterdam, June 1993.

[Alonso *et al.*, 1995] F. Alonso, N. Juristo, J. L. Maté, and J. Pazos. Trends in life-cycle models for SE and KE: Proposal for a spiral-conical life-cycle approach. *International Journal of Software Engineering and Knowledge Engineering.*, 5(3):445–465, 1995.

[Angele *et al.*, 1996] J. Angele, D. Fensel, and R. Studer. Domain and task modelling in MIKE. In A. Sutchliffe, editor, *Domain Knowledge for Interactive System Design*. Chapman & Hall, 1996.

[Angele *et al.*, 1998] J. Angele, S. Decker, R. Perkuhn, and R. Studer. Developing knowledge-based systems with MIKE. *Journal of Automated Software Engineering*, 5(4):326–389, 1998.

[Anjewierden, 1997] A. Anjewierden. CML2. Technical Report 11, University of Amsterdam, http://www.swi.psy.uva.nl/usr/anjo/cml2, 1997.

[Benjamins and Aben, 1997] R. Benjamins and M. Aben. Structure-preserving knowledge-based system development through reusable libraries: A case study in diagnosis. *International Journal of Human-Computer Studies*, 47:259–258, 1997.

[Boehm, 1988] B. W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, pages 61–72, May 1988.

[Bonissone and Johnson, 1983] P. P. Bonissone and H. E. Johnson. Expert system for diesel electric automotive repair. Technical report, General Electrics Co., 1983.

[Booch *et al.*, 1997] G. Booch, I. Jacobson, and J. Rumbagh. The UML specification documents. Technical report, Rational Software Corp., Santa Clara. CA, http://www.rational.com, 1997.

[Brazier *et al.*, 1997] F. M. T. Brazier, B. Dunin-Keplicz, N. Jennings, and J. Treur. DESIRE: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6:67–94, 1997.

[Buchanan *et al.*, 1983] B. G. Buchanan, R. Barstow, R. Bechtal, J. Bennet, W. Clancey, C. Kulikowsky, T. Mitchell, and D. A. Waterman. Constructing an expert system. In F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, editors, *Building Expert Systems*, pages 127–167. Addison Wesley, 1983.

[Chandrasekaran and Johnson, 1993] B. Chandrasekaran and T. R. Johnson. Generic tasks and task structures: History, critique and new directions. In J. M. David, J. P. Krivine, and R. Simmons, editors, *Second Generation of Expert Systems*, pages 222–272. Springer-Verlag, 1993.

[Chandrasekaran and Mittal, 1983] B. Chandrasekaran and S. Mittal. Conceptual representation of medical knowledge for diagnosis by computer: Mdx and related systems. In M. Yovits, editor, *Advances in Computers*, pages 271–293. Academic Press, 1983.

[Chandrasekaran *et al.*, 1979] B. Chandrasekaran, F. Gomez, S. Mittal, and J. Smith. An approach to medical diagnosis based on conceptual structures. In *Proc. of the 6th IJCAI*, pages 134–142, Tokio, Japan, 1979.

[Chandrasekaran *et al.*, 1992] B. Chandrasekaran, T. R. Johnson, and J. W. Smith. Task structure analysis for knowledge modeling. *Communications fo the ACM*, 35(9):124–137, 1992.

[Chandrasekaran, 1983] B. Chandrasekaran. Towards a taxonomy of problem solving types. *AI Magazine*, 4(1):9–17, 1983.

[Chandrasekaran, 1986] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for system desing. *IEEE Expert*, 1(3):23–30, 1986.

[Clancey, 193] W. J. Clancey. GUIDON. *Journal of Computer-Based Instruction*, 10(1-2):8–15, 193.

[Clancey, 1979] W. J. Clancey. Tutoring rules for guiding a case method dialogue. *Intenational Journal of Man-Machine Studies*, 11(1):25–49, 1979.

[Clancey, 1985] W. J. Clancey. Heuristic calssification. *Artificial Intelligence*, 27:289–350, 1985.

[Cockburn and Jennings, 1996] D. Cockburn and N. R. Jennings. ARCHON: A distrubuted artificial intelligence system for industrial applications. In G.. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 319–344. John Willey and Sons, 1996.

[Cuena and Molina, 1994] J. Cuena and M. Molina. KSM: An environment for knowledge oriented design of applications using structured knowledge architectures. In K. Brunnstein and E. Raubold, editors, *Applications and Impacts. Information Processing'94*, volume 2. Elsevier Science (North Holland), 1994.

[Cuena and Molina, 1997] J. Cuena and M. Molina. KSM; an environment for design structured knowledge models. In Tzafestas, editor, *Knowledge-Based Systems: Advanced Concepts, Techniques and Applications*. World Sicientific Publishing Company, 1997.

[de Hoog *et al.*, 1994] R. de Hoog, R. Martilo, B. Wielinga, R. Taylor, C. Bright, and W. Van de Velde. The CommonKADS model set. Technical Report ESPRIT Project P5248 KADS-II/M1/DM.1b/UvA/018/6.0, University of Amsterdam and Lloyd's Register and Touche Ross Management Consultans and Free University of Brussels, June 1994.

[Domingue *et al.*, 1993] J. Domingue, E. Motta, and S. Watt. The emerging VITAL workbench. In *Proceedings of the 7th European Knowledge Acquisition for Knowledge-Based Systems EKAW'93*, pages 320–339, 1993.

[Duda *et al.*, 1978] R. Duda, P. E. Hart, N. J. Nilsson, R. Reboh, J. Slocum, and G. Sutherland. Development of the PROSPECTOR consultation system for mineral explotation. Technical report, Stanford Research Institute, 1978.

[Eriksson *et al.*, 1995] H. Eriksson, T. Shahar, S. W. Tu, A. R. Puerta, and M. A. Musen. Task modelling with reusable problem-solving methods. *Artificial Intelligence*, 79(2):293–326, 1995.

[Eshelman *et al.*, 1993] L. Eshelman, D. Ehret, J. McDermott, and M. Tan. Mole: A tenacious knowledge-acquisition tool. In B. G. Buchanan and D. C. Wilkins, editors, *Readings in Knowledge Acquisition and Learning: Automating the Construction and Improvement of Expert Systems*, pages 253–259. Kaufmann, San Mateo, CA, 1993.

[Fensel, 1995] D. Fensel. *The Knowledge Acquisition and Representation Language KARL*. Kluwer Academic Press, 1995.

[Fikes and Nilsson, 1971] R. Fikes and N.Ñilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.

[Fox and Iwasaky, 1984] M. S. Fox and Y. Iwasaky. ISIS: A knowledge-based system for factory scheduling. *Expert Systems*, 1(1):25–49, 1984.

[Friedland and Iwasaky, 1985] P. Friedland and Y. Iwasaky. The concept and implementation of skeletal plans. *Journal of Automated Reasoning*, 1:161–208, 1985.

[Gómez and Chandrasekaran, 1981] F. Gómez and B. Chandrasekaran. Knowledge organisation and distribution for medical diagnosis. *IEEE Transactions on Systems, Man and Cybernetics*, 11(1):34–42, 1981.

[González *et al.*, 1986] A. J. González, R. L. Osborne, C. Kemper, and S. Lowenfeld. Online diagnosis of turbine generators using artificial intelligence. *IEEE Transaction on Energy Conversion*, 1(2):68–74, 1986.

[Guida and Tasso, 1994] G. Guida and C. Tasso. *Design and Development of Knowledge-Based Systems. From Life Cycle to Methodology*. John Wiley and Sons Ltd., Baffins Lane, Chichester, England, 1994.

[Harmelen *et al.*, 1991] F. Van Harmelen, J. R. Balder, M. Aben, and J. M. Akkermans. $(ml)^2$: A formal language for

KADS conceptual models. Technical Report ESPRIT Project P5248 KADS-II/T1.2/TR/ECN/006/1.0, Netherlands Energy Research Centre ECN and University of Amsterdam, June 1991.

[Heckerman and Horovitz, 1986] D. Heckerman and E. Horovitz. The myth of modularity in rule-based systems. Technical report, Standford University, 1986.

[Holland *et al.*, 1984] J. D. Holland, E. L. Hutchings, and L. Weitzman. An interactive, inspectable, simulation-based training system. *AI Magazine*, 5(2):15–27, 1984.

[IEEE, 1999] IEEE. *IEEE Standards Software Engineering. Customer and Terminology Standards*, volume 1. IEEE, 1999.

[Iglesias-Fernández, 1997] Carlos A. Iglesias-Fernández. *Definición de Una Metodología Para el Desarrollo de Sistemas Multiagentes*. PhD thesis, Dpto. de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, 1997.

[Jennings *et al.*, 1992] N. R. Jennings, E. H. Mamdani, I. Laresgoiti, J. Perez, and J. Corera. GRATE: A general framework for cooperative problem solving. *Journal of Intelligent Systems Engineering*, 1(2):102–114, 1992.

[Jennings *et al.*, 1996] N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, M. E. Wiegand, C. Voudouris, J. L. Alty, T. Miah, and E. H. Mamdani. ADEPT: Managing business processes using intelligent agents. In *Proceedings of the BCS Expert Systems 96 Conference ISIP Track*, pages 5–23, Cambridge, UK, 1996.

[Kahn, 1994] A. F. Kahn. Managing knowledge-based systems development using standard life-cycle techniques. In E. Turban and J. Liebowitz, editors, *Managing Expert Systems*, pages 120–160. Idea Group Publishing, 1994.

[Landes, 1994] D. Landes. DesignKARL. a language for design of knowledge-based systems. In *Proceedings of the 6th International Conference on Software Engineering and Knowledge Engeneering SEKE´94*, pages 78–85, 1994.

[Maestre-Yenes, 2000] P. Maestre-Yenes. *Diccionario de Gestión Del Conocimiento En Informática*. Monografías Y Publicaciones. Gestión Del Conocimiento. Fundación Dintel, 2000.

[Marcus and McDermott, 1993] S. Marcus and J. McDermott. Salt: A knowledge acquisition language for propose-and-revise systems. In B. G. Buchanan and D. C. Wilkins, editors, *Readings in Knowledge Acquisition and Learning: Automating the Construction and Improvement of Expert Systems*, pages 263–281. Kaufmann, San Mateo, CA, 1993.

[McCarthy and Hayes, 1969] K. McCarthy and P. J. Hayes. Some philosophical problems form the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[Mcdermott, 1982] J. Mcdermott. R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 10(1):39–88, 1982.

[McDermott, 1988] J. McDermott. Preliminary steps towards a taxonomy of problem-solving methods. In S. Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–256. 1988.

[Miller *et al.*, 1982] R. A. Miller, H. E. Pople, and J. D. Myers. INTERNIST-i, an experimental computer-based diagnosis consultant for general internal medicine. *New England Journal of Medicine*, 37(8):468–476, 1982.

[Mittal and Chandrasekaran, 1984] S. Mittal and B. Chandrasekaran. Patrec: A knowledge-directed data-base for a diagnostic expert system. *Computer*, 17:51–58, 1984.

[Musen, 1989a] M. A. Musen. *Automated Generation of Model-Based Knowledge-Acquisition Tools*. Pitam, London, 1989.

[Musen, 1989b] M. A. Musen. An editor for the conceptual models of interactive knowledge acquisition tools. *International Journal of Man-Machine Studies*, 31:673–698, 1989.

[Musen, 1992] M. A. Musen. Overcoming the limitations of role-limiting methods. *Knowledge Acquisition*, 4(2):165–170, 1992.

[Musen, 1993] M. A. Musen. An overview of knowledges acquisition. In J. M. David, J. P. Krivine, and R. Simmons, editors, *Second*

Generation of Expert Systems, pages 405–427. Springer Verlag, 1993.

[Neubert, 1993] S.Ñeubert. Model constructrion in MIKE. In *Knowledge Acquisition for Knowledge-BAsed Systems. Proceedings of the 7th European Workshop EKAE'93 (LNCS No. 723)*, pages 200–219. Springer-Verlag, 1993.

[Newell and Simon, 1988] A.Ñewell and H. A. Simon. GPS: A program that simulates human thought. In A. Collins and E. E. Smith, editors, *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*, pages 453–460. Kaufmann, San Mateo, CA, 1988.

[Newell, 1982] Allen Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.

[O'Hare and Wooldridge, 1992] G. M. P. O'Hare and M. J. Wooldridge. A software engineering perspective on multi-agent system design. experiece in the development of MADE. In M. Avouris and Les Gesser, editors, *Distributed Artificial Intelligence: Theory and Praxis*, pages 109–127. Kluwer Academic Publishers, 1992.

[Palma-Méndez, 1999] Jose T. Palma-Méndez. *Ingeniería Del Conocimiento Para Sistemas En Tiempo Real Basados En Conocimiento*. PhD thesis, Facultad de Informática, Universidad de Murcia, 1999.

[Puerta *et al.*, 1992] A. R. Puerta, J. W. Egar, S. W. Tu, and M. A. Musen. A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquistion tools. *Knowledge Acquisition*, 4(2):171–196, 1992.

[Rumbaugh *et al.*, 1991] J. Rumbaugh, M. Blaha, W. Premerlani, and V. F. Eddy. *Object-Oriented Modelling and Design*. Prentice-Hall, New Jersey, 1991.

[Scarl *et al.*, 1987] E. A. Scarl, A. R. Jamieson, and C. I. Delaune. Diagnosis and sensor validation through knowledge of structure and function. *IEEE Transaction on System, Man and Cybernetics*, 17(3):360–368, 1987.

[Schreiber *et al.*, 1993] G. Schreiber, B. J. Wielinga, and J. A. Breuker, editors. *KADS: A Principle Approach to Knowledge-Based System Development*. Academic Press, 1993.

[Schreiber *et al.*, 1999] A. T. Schreiber, J. M. Akkermans, A. A. Anjewierden, R. de Hoog, N. R. Shadbolt, W. Van de Velde, and B. J. Wielinga, editors. *Knowledge Engineering and Management. The CommonKADS Methodology*. MIT Press, Cambridge, Massachusetts. London, England, 1999.

[Shadbolt *et al.*, 1993] N. Shadbolt, E. Motta, and A. Rouge. Constructing knowledge-based systems. *IEEE Software*, 10(6):34–38, 1993.

[Shaw and Gaines, 1992] M. L. G. Shaw and B. R. Gaines. The synthesis of knowledge engineering and software engineering. In P. Loucopoulos, editor, *Advanced Information Systems Engineering (LNCS 593)*. 1992.

[Shortliffe, 1976] E. H. Shortliffe. *Computer-Based Medical Consultations: Mycin*. America Elsevier, New York, 1976.

[Steels, 1990] L. Steels. Components of expertise. *AI Magazine*, 11(2):30–49, 1990.

[Steels, 1993] L. Steels. The componential framework and its role in reusability. In J. M. David, J. P. Krivine, and R. Simmons, editors, *Second Generation of Expert Systems*, pages 273–298. Springer-Verlag, 1993.

[Studer *et al.*, 1998] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, (25):161–197, 1998.

[Tu *et al.*, 1995] S. W. Tu, H. Eriksson, J. Gennari, Y. Shahar, and M. A. Musen. Ontology-based configuration of problem-solving methods and generation of knowledge acquisition tools: Application of PROTÉGÉ-II to protocol-based decision support. *Artificial Intelligence in Medicine*, 7:257–289, 1995.

[Wielinga *et al.*, 1994] B. Wielinga, J. M. Akkermans, H. A. Hassan, O. Olsson, K. Orsvärn, A. Schreiber, P. Terpstra, W. Van de Velde, and S. Wells. Expertise model definition document. Technical Report ESPRIT Project P5248 /KADS-II/M2/UvA/026/5.0, University of Amsterdam and Free University of Brussels and Netherlands Energy Research Centre ECN, June 1994.