Razonamiento Basado en Casos: Una Visión General

Laura Lozano
Universidad de Valladolid
Estudiante 4º Informática
e-mail: laurloz@lab.fi.uva.es

Javier Fernández
Universidad de Valladolid
Estudiante 4º Informática
e-mail: javifer@lab.fi.uva.es

Prólogo

Basado en el trabajo de [Aamodt & Plaza], [Kolodner], [Mitra & Basak], [Watson] ó [Sankar & Simon], entre otros, este trabajo pretende ser una síntesis del apasionante tema del Razonamiento Basado en casos, permitiendo al lector tener una visión general del mismo a la par que cierta profundización en aspectos clave. Además, se proporciona un ejemplo propio, empleando la herramienta ReMind.

1. Introducción al Razonamiento basado en casos

1.1. Definición y características

En los últimos años, el Razonamiento Basado en casos ha experimentado un rápido crecimiento desde su nacimiento en Estados Unidos. Lo que sólo parecía interesante para un área de investigación muy reducida, se ha convertido en una materia de amplio interés, multidisciplinar y de gran interés comercial.

El Razonamiento Basado en Casos, no es más que otro paradigma de resolución de problemas, pero son precisamente las diferencias con el resto de los acercamientos de la inteligencia artificial las que lo hacen tan especial. En lugar de confiar únicamente en el conocimiento general del dominio del problema, o realizar asociaciones a lo largo de relaciones entre descripciones del problema y conclusiones, este paradigma es capaz de utilizar conocimiento específico de experiencias previas, es decir, situaciones de un problema concreto (casos). Un problema nuevo (al decir nuevo nos referimos a nunca antes tratado) es resuelto cuando se encuentra un caso pasado similar y se reutiliza en la situación del problema nuevo.

Una segunda diferencia, no por ello menos importante, es un acercamiento al aprendizaje incremental, sostenido, ya que se guarda una experiencia nueva cada vez que se resuelve un problema, pasando a estar disponible para futuros problemas desde ese mismo momento.

Con las pocas nociones vistas hasta ahora ya podríamos aventurarnos a dar una primera definición para el Razonamiento Basado en Casos: Resolver un problema nuevo recordando una situación similar previa y reutilizando su información y conocimiento.
Si bien el paradigma, como técnica de inteligencia artificial, es novedoso, el Razonamiento Basado en Casos es bien conocido entre los psicólogos muchos años

atrás. Pensemos en un médico que examinando un nuevo paciente recuerda un caso parecido unas semanas atrás; encuentra una similitud importante de los síntomas y decide asumir que posee la misma enfermad y le trata de la misma manera, pues el tratamiento resultó efectivo en la ocasión anterior.
De la misma manera, a un broker de la bolsa le recomiendan una aparente buena inversión pero ciertos síntomas del mercado le recuerdan que hace un año con una situación parecida perdió mucho dinero y decide no invertir en esta ocasión.

En los dos ejemplos anteriores, podemos observar como se ha recordado un caso anterior para aplicarlo a un nuevo problema. En la terminología CBR, podemos definir un caso como una situación de un problema. De esta manera, una situación previamente experimentada, que ha sido capturada y aprendida de manera que pueda ser reutilizada para resolver futuros problemas, se denomina un *caso previo*, *caso almacenado* ó *caso guardado*. Así, un *caso nuevo* ó un *caso sin resolver* no es más que la descripción de un problema nuevo a resolver (donde "resolver" puede ser desde justificar o criticar una solución propuesta, a interpretar el problema, generar un conjunto de soluciones posibles ó generar expectativas de datos observados).

El Razonamiento Basado en Casos sugiere un modelo de razonamiento que incorpora los aspectos ya mencionados de resolución de problemas, entendimiento y aprendizaje e integra todo ello en meros procesos de memoria. En resumen, estas son las premisas subyacentes al modelo:

- La referencia a casos pasados es interesante y de gran utilidad para tratar situaciones que -vuelven a darse-. La referencia a situaciones similares es necesaria a menudo para tratar la complejidad de una nueva situación. Por ello, recordar un caso para usarlo en un problema futuro (e integrar ambos) es, necesariamente, un proceso de aprendizaje.
- Debido a que las descripciones de los problemas son, a menudo, incompletas, es necesario una etapa de entendimiento o interpretación., ya que no puede llevarse a cabo un razonamiento, una resolución adecuada de una nueva situación, si ésta no se entiende con cierta completitud. Se puede considerar que esta etapa es a la vez un prerrequisito y una parte del ciclo de razonamiento, pues el entendimiento de las situaciones mejora conforme progresa el razonador. No obstante, cualquier forma de razonamiento necesita que la situación sea elaborada con suficiente detalle y representada con suficiente claridad y con el vocabulario apropiado para que el razonador reconozca el conocimiento que necesita (sea conocimiento general o casos) para razonar a partir de él.
- La práctica demuestra que no suele existir un caso pasado exactamente igual que un caso nuevo. Por ello, es muy usual el tener que adaptar la solución pasada para que se ajuste a la nueva situación.
- El aprendizaje es una consecuencia natural del razonamiento. Si se halla un nuevo procedimiento en el curso de la resolución de un problema complejo y su ejecución resulta positiva, entonces se aprende el nuevo procedimiento para resolver esta nueva clase de situaciones.
- La revisión de la solución propuesta y el análisis de la revisión son dos partes necesarias para completar el ciclo de razonamiento/aprendizaje. Este análisis de la revisión (habitualmente llevada a cabo por un agente externo, léase humano) puede conllevar una reparación de posibles fallos.

Estas premisas sugieren que la calidad de un razonador basado en casos depende de:

1. La experiencia que tiene.
2. La habilidad para entender situaciones nuevas en términos de experiencias pasadas.
3. Su capacidad de adaptación.
4. Su capacidad de evaluación y reparación.
5. Su habilidad para integrar nuevas experiencias en su memoria adecuadamente.

1.2. Tareas y sistemas representativos

Las aplicaciones CBR se clasifican principalmente en dos tipos: tareas de clasificación y tareas de síntesis.

En las tareas de clasificación, un caso nuevo se *empareja* (matching en su terminología original) con aquellos de la base de casos para determinar que tipo, clase o caso es. La solución del caso que mejor ajusta es el que se reutiliza.

La mayoría de las herramientas CBR disponible dan un soporte aceptable para las tareas de clasificación, que suelen estar relacionadas con la recuperación de casos. Existe una gran variedad de tareas de clasificación, como por ejemplo:

- Diagnosis: Médica o de fallos de equipos
- Predicción: Pronóstico de fallos de equipos o actuación sobre el stock de un mercado.
- Valoración: análisis de riesgos para bancos o seguros o estimación de costes de proyectos.
- Control de procesos: Control de fabricación de equipos.
- Planificación: Reutilización de planos de viaje ó planificadores de trabajo.

Las tareas de síntesis intentan crear una nueva solución combinando partes de soluciones previas. Éstas son inherentemente complejas a causa de las restricciones de los elementos usados durante la síntesis.

Los Sistemas CBR que realizan tareas de síntesis deben realizar adaptación y son normalmente sistemas híbridos que combinan CBR con otras técnicas. Algunas de las tareas que realizan estos sistemas son:

- Diseño: La creación de un nuevo artefacto adaptando elementos de otros existentes.
- Planificación: La creación de nuevos planes a partir de otros previos.
- Configuración: La creación de nuevos planificadores a partir de otros previos.

Por norma general, los sistemas que implementan tareas de síntesis son más difíciles de construir que los que implementan tareas de clasificación.

1.2.1 Tareas de clasificación

Las tareas de clasificación son habituales en el mundo de los negocios e incluso cualquier día de nuestra vida. Se pueden reconocer cuando se necesita emparejar un objeto o evento con otros en una librería en la cual se puede inferir una respuesta.

Algunos ejemplos de tareas de clasificación en el mundo de los negocios son:

- ¿Qué tipo de casa es esta? (de lujo, chalet, adosado, cabaña, apartamento, etc.)
- ¿Qué tipo de tratamiento debe darse al paciente? (observación, esteroides, antibióticos, etc.)
- ¿Hay petróleo bajo esta tierra? (es posible, es imposible, muy probablemente, no es probable)
- ¿Cuánto tiempo llevará acabar este proyecto? (3 meses, 6 meses, 1 año, etc.)

Si bien alguna clasificación no tiene valores discretos, sí es posible dar un cierto rango. Por ejemplo, una casa de lujo no tiene porqué ser la que vale exactamente 1 millón de Euros, pero sí las que pueden valer entre 1 y 3 millones de Euros.

Habitualmente, las clasificaciones de las preguntas se refieren a resultados, esto es, un resultado es normalmente un atributo del caso y es por lo que se clasifican los casos.
Podemos aplicar CBR fácilmente a problemas de clasificación puesto que pueden consistir en:

- La recuperación de un amplio conjunto de casos similares, por ejemplo aquellos en los que el antibiótico fue el tratamiento.
- Recuperar el mejor ajuste de este conjunto, quizás para sugerir penicilina como antibiótico específico.
- Adaptar la solución, por ejemplo alterando la dosis para diferentes edades o pesos de los pacientes.
- Almacenar el resultado del nuevo caso para un futuro uso.

Como vemos, las tareas de clasificación son fáciles de implementar porque se ajustan al ciclo CBR, los casos tienden a ser más fáciles de representar y recuperar, y los algoritmos de recuperación utilizados en la mayoría de las herramientas CBR son clasificadores.

1.2.2 Tareas de síntesis

Estas tareas son comunes en el comercio pero difíciles de implementar. Esto se debe a que es más fácil ajustar un artefacto a un conjunto de artefactos prototípicos que construir un artefacto a partir de una especificación.

Las tareas de clasificación simplemente requieren reconocimiento de las características mientras que las tareas de síntesis requieren colocar las características correctas en el orden y lugar correcto.

Los sistemas de síntesis operan en dominios de diseño o planificación para intentar simplificar el proceso creativo produciendo un diseño o plan que se sabe que es bueno para producir el plan final a partir de él.

Para los diseñadores, esto es más rápido que empezar un diseño desde una hoja en blanco. Se asume que modificar un buen diseño o plan inicial es más fácil que crear uno desde el principio.

En muchas circunstancias esto es cierto, sin embargo hay muchas situaciones en que se debe empezar desde cero sin tener referencia de ningún ejemplo pasado. Por ejemplo, muchos cohetes espaciales se diseñan desde cero para no cometer los mismos errores.

Las razones por las que los sistemas de síntesis son difíciles de construir son:

- La representación de un caso de un plan o diseño es compleja y altamente estructurada con muchas dependencias entre características. Los casos no se almacenan en un medio único y homogéneo, por tanto la recuperación de casos en más difícil.
- Las herramientas CBR tienden a no soportar indexación o recuperación de representaciones de casos altamente estructurados.
- La adaptación es a menudo un requisito clave en las tareas de síntesis.

1.2.3 Sistemas Representativos

Estudiaremos los siguientes sistemas CBR:

- CHEF: planificador basado en casos.
- JULIA: diseñador basado en casos.
- CASEY: programa de diagnósticos basado en casos.
- HYPO: programa interpretativo basado en casos.
- PROTOS: programa de clasificación basado en casos.

Los modelos de memoria empleados en cada uno de estos sistemas son explicados en detalle en la sección 3.1.

CHEF

CHEF es un planificador basado en casos que toma como entrada una conjunción de submetas que necesita lograr para conseguir un plan como salida. Su dominio es la creación de recetas. Las recetas son vistas como planes. Las recetas proporcionan la secuencia de pasos a seguir para preparar un plato. Por tanto la entrada de CHEF son las metas que se pueden conseguir con las recetas (por ejemplo, incluye pescado, sofreír, sabor salado) y la salida es una receta (plan), que puede obtener esas metas.

Como planificador basado en casos, CHEF crea sus planes a partir de viejos planes que funcionaron en situaciones similares y los modifica para adaptarlos a la nueva situación. Por tanto el primer paso en la creación de un plan es recuperar una vieja receta que cumpla el mayor número posible de metas de la entrada. Para recordar esta clasificación indexa los planes por las metas que logra. Ternera con brócoli es indexado por varias metas, entre ellas, incluye carne, incluye una verdura fresca, sofreír y lograr sabor salado.

El siguiente paso es adaptar el viejo plan a la nueva situación. Esto se hace en dos etapas. Primero, se reinstancia el viejo plan, es decir, crea una instancia en la que

sustituye los nuevos objetos por los del viejo plan. Por ejemplo, si está creando una receta de pollo con guisantes desde la receta de ternera con brócoli, sustituye ternera por pollo y brócoli por guisantes. Para poder hacer esto, necesita conocer los roles que desempeñan los objetos en la vieja receta. CHEF tiene un conocimiento bastante limitado sobre esto y para saber qué objetos sustituir y por cuáles busca las similitudes entre los objetos del viejo plan y los del nuevo y sustituye los objetos del nuevo plan por los más similares del viejo plan. Por ejemplo, si el pollo y la ternera están definidos como carne, los sustituye.

En la segunda etapa, aplica *criticas de objeto* para adaptar el viejo plan a la nueva situación. Un ejemplo es, el pato no debe tener grasa antes de sofreírlo. Se expresaría de la siguiente manera:

Después del paso: deshuesar el pato
hacer: quitar la grasa al plato
porque: en este momento el pato tiene grasa

Esta crítica está asociada al objeto pato, y cada vez que se usa el pato en una receta se dispara la crítica. Si hay un paso de deshuesar el pato en la receta que se está creando, se añadiría detrás un paso indicando que se debería de quitar la grasa al pato.

Las *críticas de objeto* generalmente añaden pasos de preparación especiales a la receta (deshuesar, quitar la grasa…). Las críticas son la forma en que CHEF codifica el conocimiento sobre procedimientos especiales asociados al uso de objetos de su dominio. Su uso durante la adaptación muestra la interacción entre el uso de experiencia y de conocimiento general de los sistemas CBR.

Después de la reinstanciación y la aplicación de críticas, CHEF obtiene un plan completo. Para validar el plan lo ejecuta en un simulador muy parecido al mundo real y si es correcto lo almacena, si no crea una explicación causal de porqué no funcionó el plan y lo usa como índice para reparar el plan.

Los *TOPs* (*paquetes de organización temática*) son estructuras que encierran conocimiento general del dominio Relacionan el conocimiento acerca de cómo interaccionan entre sí los objetivos del caso considerado.
En CHEF, se usan para caracterizar de un modo abstracto las interacciones entre las etapas de una receta. Así, el sistema incluye un *TOP* EFECTOS LATERALES: CONDICIÓN INVALIDADA : CONCURRENCIA. CONCURRENCIA significa que dos objetivos han sido llevados a cabo a través de una sola acción. EFECTO LATERAL: CONDICIÓN INVALIDADA significa que ha tenido lugar un efecto lateral: intentar satisfacer uno de los objetivos impide la satisfacción del otro. Este *TOP* caracteriza, por ejemplo, el fallo que ocurre cuando se cocina brécol con ternera y el brécol se reblandece.

Los *TOPS* son importantes porque capturan conocimiento a veces independiente del dominio de resolución de problemas. En CHEF, una solución al problema es dividir la acción en dos etapas de modo que los objetivos no interfieran entre sí (cocinar los ingredientes en dos etapas). Otra solución basada en un *TOP* sería añadir otra acción que contrarrestara el efecto de la anterior (añadir algo que absorbiera el agua que reblandece el brécol). Es importante observar que estas estrategias son bastante

generales y podrían constituir sugerencias interesantes para la reparación de planes en otros muchos dominios.

CHEF emplea tres tipos de estrategias de reparación: (1) dividir y reformular los pasos del plan, (2) alterar el plan para que se ajuste a la solución esperada o (3) buscar un plan alternativo.

En primer lugar CHEF elige la estrategia de reparación buscando casos similares en el TOP que sugieran una reparación y después chequea la posibilidad de aplicar las condiciones para cada plan reparado. Así encuentra el más apropiado. Aplica el plan de reparación apropiado, resolviendo el plan fallido. CHEF realiza otra etapa más antes de acabar, actualiza su conocimiento del mundo para poder anticiparse y no repetir el error que acaba de cometer. Para ello, encuentra unos predictores del tipo de fallo y lo usa como índice para advertir de la posible ocurrencia del fallo. Evidentemente estos índices son extraídos de la explicación previa del fallo, aislando la parte de la explicación que describe características del medio responsables del fallo.

La siguiente figura muestra la arquitectura de CHEF. Los rectángulos representan sus unidades funcionales, los círculos sus fuentes de conocimiento y los óvalos las entradas y salidas de cada proceso funcional.

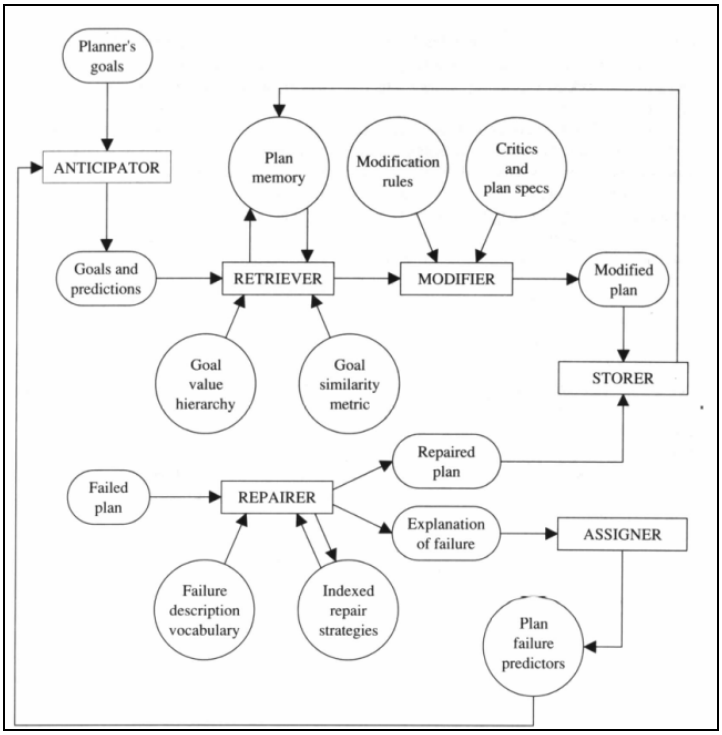

Figura 1. Arquitectura Funcional de CHEF

CASEY

CASEY es un sistema de diagnóstico basado en casos. Como entrada toma una descripción de su nuevo paciente, incluyendo muestras normales y presentándolas como síntomas. Su salida es una explicación causal de los desordenes que tiene el paciente.

CASEY diagnostica los pacientes aplicando heurísticas de adaptación y emparejamiento basado en el modelo independientes del dominio y son tan precisas como el modelo del dominio en el que se aplican. La diagnosis la realiza en dos pasos: primero busca en la memoria casos y utiliza reglas de evidencia basadas en el modelo para determinar cuales de los casos que ajustan parcialmente son suficientemente similares al nuevo problema para proporcionar una diagnosis precisa. Después aplica reglas de reparación basadas en el modelo (estrategias de adaptación) para adaptar el diagnóstico antiguo a la nueva situación.

JULIA

Julia es un diseñador basado en casos que opera en el dominio de planificación de comidas.

Como en otros dominios de diseño los problemas se describen en términos de las restricciones que tienen que cumplir, y las soluciones describen la estructura y un artefacto que cumple la mayor cantidad de restricciones posibles. Los problemas son muy extensos y en general no pueden resolverse encontrando un caso anterior que se pueda adaptar para resolver el problema. Habitualmente los problemas se dividen en partes que se resuelven independientemente. Obviamente se debe tener en cuenta las dependencias entre las partes en que se descompone el problema. Para ello JULIA refuerza el razonamiento basado en casos con un proceso de propagación de restricciones, usando las restricciones como índices del diseño. Las restricciones provienen de diversas fuentes, unas del conocimiento general del artefacto a diseñar (ingredientes incompatibles, por ejemplo) y otras del nuevo problema (los ingredientes que quiere emplear el usuario).

JULIA debe tener un conocimiento general del tipo de artefacto a diseñar almacenándolo en prototipos de objetos, es decir, los prototipos de objetos se corresponden con los diferentes tipos de comida que conoce, por ejemplo, cenas americanas, cenas europeas, comidas de catering, etc. Cada prototipo tiene una estructura específica y relaciones entre sus partes. JULIA usa los prototipos de comida para proporcionar *frameworks* como soluciones cuando los casos no son la opción correcta.

Además de posibles fallos en el proceso de diseño puede verse afectado cuando el cliente introduce nuevas restricciones, lo cual ocurre de manera habitual en la vida real. La adaptación se puede realizar bien por sustitución o bien cambiando la estructura de la solución, en ambos casos guiado por restricciones.

HYPO

HYPO es un razonador interpretativo que trabaja en el dominio de la ley. HYPO toma como entrada una situación legal y como salida crea un argumento para sus clientes legales. HYPO puede crear argumentos igual de buenos tanto para la defensa como para la acusación. El dominio particular en el que tiene más experiencia es el de los casos de revelación. Típicamente en estos casos, un secreto de una compañía se desvela por otra compañía de la competencia y ésta saca provecho de la situación, por ejemplo comercializando rápidamente un nuevo producto. Pero no todos los casos de revelación

de secretos son ilegales y la tarea del programa es decidir, para cualquier caso, si es o no legal y crear un argumento que avale la decisión.

El proceso de razonamiento de HYPO consta de varios pasos:

- Analizar el caso para determinar cuales son sus factores relevantes de entre todos sus descriptores.
- Recuperar casos que compartan estos factores.
- Separar los casos anteriores en los que apoyan a la defensa y en los que apoyan a la acusación de la nueva situación.
- Seleccionar los casos que comparten el mayor número de características con la nueva situación.
- HYPO crea los argumentos, llamados argumentos de tres capas. Se elige el caso que más ajusta de los de la defensa y el que más ajusta de la acusación. Se buscan diferencias entre estos dos casos y se eligen casos que den soporte a estas diferencias para crear *argumentos* y *contrargumentos*. Este es el principal paso de este proceso, se analizan diferencias y similitudes entre casos antiguos y nuevos y entre varios casos antiguos.
- El análisis realizado en el proceso de argumentación es empleado para justificar la decisión tomada por HYPO.
- Se crean casos hipotéticos que nunca han ocurrido y se usan para validar el análisis realizado y determinar el alcance del análisis.

PROTOS

PROTOS implementa clasificación y adquisición de conocimiento basado en casos. Dada una descripción de una situación u objeto la clasifica por su tipo. Cuando PROTOS clasifica un elemento de forma incorrecta, su *consultor experto* le informa del error y del conocimiento que debe usar para clasificarlo correctamente.

El dominio de PROTOS es el de los desordenes auditivos. Dada una descripción de síntomas y los resultados de algunas pruebas de un paciente, PROTOS determina que problema auditivo tiene el paciente. El consultor experto es un otorrino experto. Puede emplearse para otras tareas como el reconocimiento del estado emocional de un agente.

Para hacer la clasificación de una situación u objeto, PROTOS busca el objeto o situación, ya conocido, que mejor ajusta a la nueva situación u objeto y le asigna la clasificación del que mejor ajusta al nuevo.

El proceso llevado a cabo por PROTOS es, primeramente, determinar la categoría del nuevo problema buscando, en las categorías de problemas auditivos que conoce, aquella cuyas características importantes coinciden con las características importantes del nuevo problema. A continuación, verifica esta hipótesis intentando ajustar su nuevo caso a ejemplares de la categoría de hipótesis para ver si puede encontrar un buen emparejamiento. Si lo encuentra el proceso acaba. Si no, utiliza los resultados de este proceso de ajuste para seleccionar una hipótesis mejor. Este proceso se guía por su conocimiento sobre los tipos de errores de clasificación más comunes en este dominio.

Cuando se hace una clasificación incorrecta debido a la fuerte coincidencia del caso con los casos de la categoría, PROTOS añade enlaces de diferencia (el modelo de memoria

de PROTOS es categoría-ejemplar y es explicado en detalle en la sección 3.1) desde la categoría incorrecta a la correcta para evitar cometer el mismo error en el futuro.

El proceso de adquisición de conocimiento es dirigido por los fallos del proceso de clasificación. Cuando no es capaz de identificar correctamente la categoría de una entrada, establece una conversación con el consultor experto con el objetivo de añadir nuevo conocimiento y revisar las conexiones de su memoria. La comunicación entre el experto y PROTOS se realiza mediante un vocabulario fijado de relaciones causales, taxonómicas, funcionales y correlacionales.

2. Razonamiento basado en casos: descripción general

2.1. El ciclo del razonamiento basado en casos

Un ciclo de vida CBR está formado esencialmente por los cuatro procesos siguientes (ver figura 2):

1. Recuperar el caso ó casos pasados más similares (RETRIEVE). Esto es, retomar la experiencia de un problema anterior que se cree es similar al nuevo.
2. Reutilizar la información y conocimiento de este caso ó casos recuperados para resolver el nuevo problema (REUSE). Esto es, copiar o integrar la solución del caso ó casos recuperados.
3. Revisar la solución propuesta (REVISE)
4. Guardar la nueva solución una vez ha sido confirmada ó validada (RETAIN). Se guardan aquellas partes de la experiencia de una manera tal que sea útil para resolver problemas futuros.

Es decir, un problema nuevo se resuelve recuperando uno o más casos previos (ya experimentados), reutilizando el caso de una manera u otra, revisando la solución propuesta, y guardando la nueva experiencia incorporándola a la base de conocimiento existente (*base de casos*).

A este respecto hay que hacer varias aclaraciones:

- Los cuatro procesos no son tareas únicas, es decir, cada uno de ellos implica llevar a cabo una serie de tareas más específicas, que serán descritas en el apartado siguiente.
- Si bien se ha dado a entender que en el proceso de reutilización se lleva a cabo toda la problemática de adaptación del caso ó casos recuperados para el nuevo problema, en muchas aplicaciones prácticas las fases de reutilización y revisión apenas se distinguen, y muchos investigados hablan de *fase de adaptación*, que combina ambas. No obstante, la adaptación es quizá uno de los frentes más abiertos en los sistemas CBR debido a su complejidad.

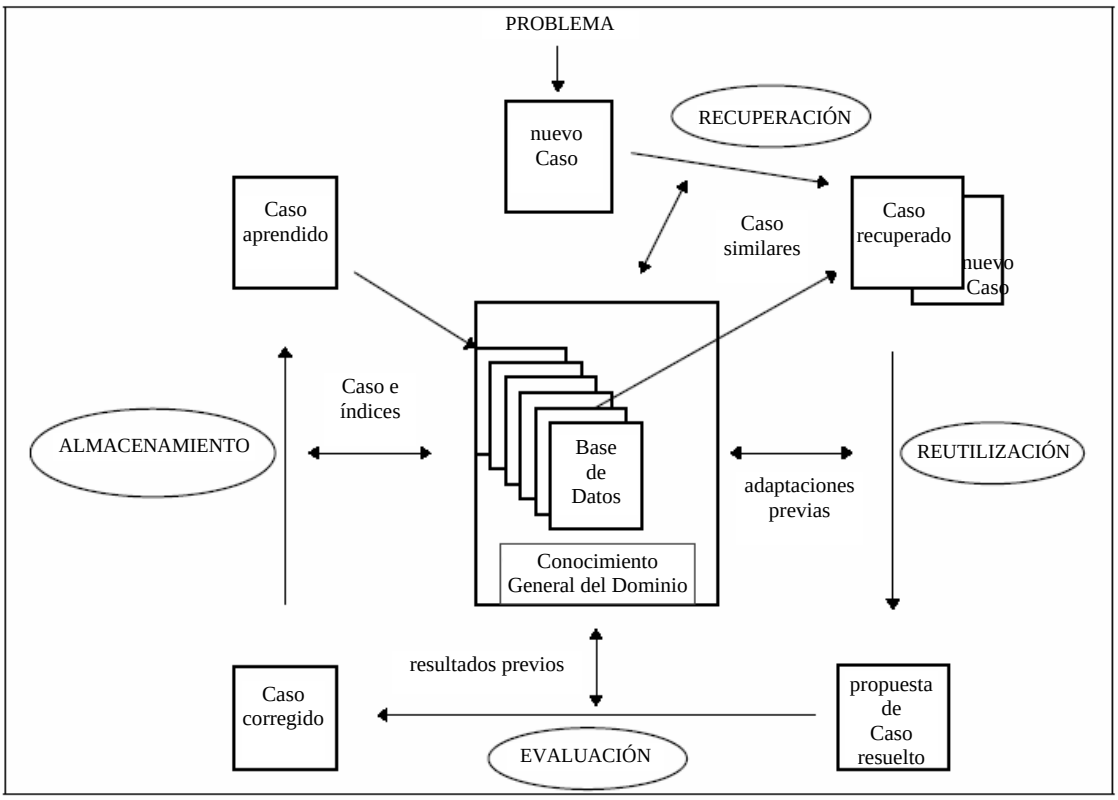

Figura 2. Ciclo básico de un sistema CBR

En la figura podemos observar lo comentado sobre el ciclo CBR. Una descripción de un problema define un nuevo caso, que se usa en la RECUPERACIÓN de un caso de entre la colección de casos pasados. El caso recuperado se combina con el nuevo caso para, a través de la REUTILIZACIÓN, proponer un caso que sea una solución al problema inicial. En una fase de EVALUACIÓN se verificará el éxito de la solución, por ejemplo siendo comprobada en el mundo real (habitualmente por un agente humano), y se reparará si falla. Durante el ALMACENAMIENTO, la experiencia útil se guarda para una futura reutilización, y el caso base se actualiza por un nuevo *caso aprendido*, o por una modificación de algunos casos existentes.

Además, la figura nos muestra cómo el conocimiento general juega un papel importante en el ciclo, dando soporte a los procesos CBR. Dependiendo del tipo de método CBR, este soporte puede variar, desde una consideración débil (ó nula) a un soporte muy fuerte. Se trata, por tanto, de un conocimiento general, frente al conocimiento específico encerrado en los casos. Por ejemplo, en el diagnóstico médico por analogía con casos de pacientes previos, el conocimiento general podría consistir en un modelo de anatomía junto con relaciones causales entre estados patológicos, formulados quizá a modo de un conjunto de reglas.

2.2. Jerarquía de tareas

La visión orientada a procesos del ciclo de vida CBR nos da una buena idea de la secuencia de pasos que se siguen, pero para ver realmente el mecanismo que subyace es necesario tomar una visión orientada a tareas, donde cada paso, o subproceso, se ve como una tarea que el razonador CBR tiene que conseguir. Las tareas se agrupan por las metas del sistema, y una tarea particular se realiza aplicando uno ó varios métodos.

En la figura 3, las tareas se muestran con los nombres de los nodos en negrita, mientras que los métodos están en cursiva. Los enlaces entre nodos de tarea (líneas continuas) representan varias descomposiciones de la tarea implicada.

Por ejemplo, la tarea de más alto nivel que es la resolución del problema y aprendizaje de la experiencia se descompone en cuatro tareas, que se corresponden con los cuatro procesos de la figura 2, recuperación, reutilización, revisión y almacenamiento. Es necesario llevar a cabo las cuatro tareas para poder alcanzar el objetivo de más alto nivel. A su vez, la tarea de recuperación se descompone en las tareas de identificar, buscar, emparejar y seleccionar. Y así sucesivamente.

Los métodos de cada tarea (líneas discontinuas) indican diferentes formas de llevar a cabo la tarea. Un método especifica un algoritmo que identifica y controla la ejecución de la subtarea particular, utilizando el conocimiento e información necesarios. Por ejemplo, para la tarea de más alto nivel (recordemos que es la resolución de un problema y el aprendizaje de la experiencia), el método para llevarla a cabo es el razonamiento basado en casos.

La figura es completa en cuanto a las particiones que realiza, es decir, el grado de descomposición de las tareas en subtareas pretende ser suficiente para llevar a cabo la misma. Por supuesto, la figura no muestra ninguna estructura de control en las subtareas, si bien se da a entender cierto secuenciamiento en cuanto que se han colocado antes (referente a posición en la página) las subtareas que aquellas que las siguen.

Sin embargo, la figura no es completa en cuanto a los métodos, es decir, uno de los métodos que se indican puede ser suficiente para resolver la tarea, o puede que se deban combinar varios de ellos, o en el caso más extremo se deban utilizar otros métodos aquí no indicados.

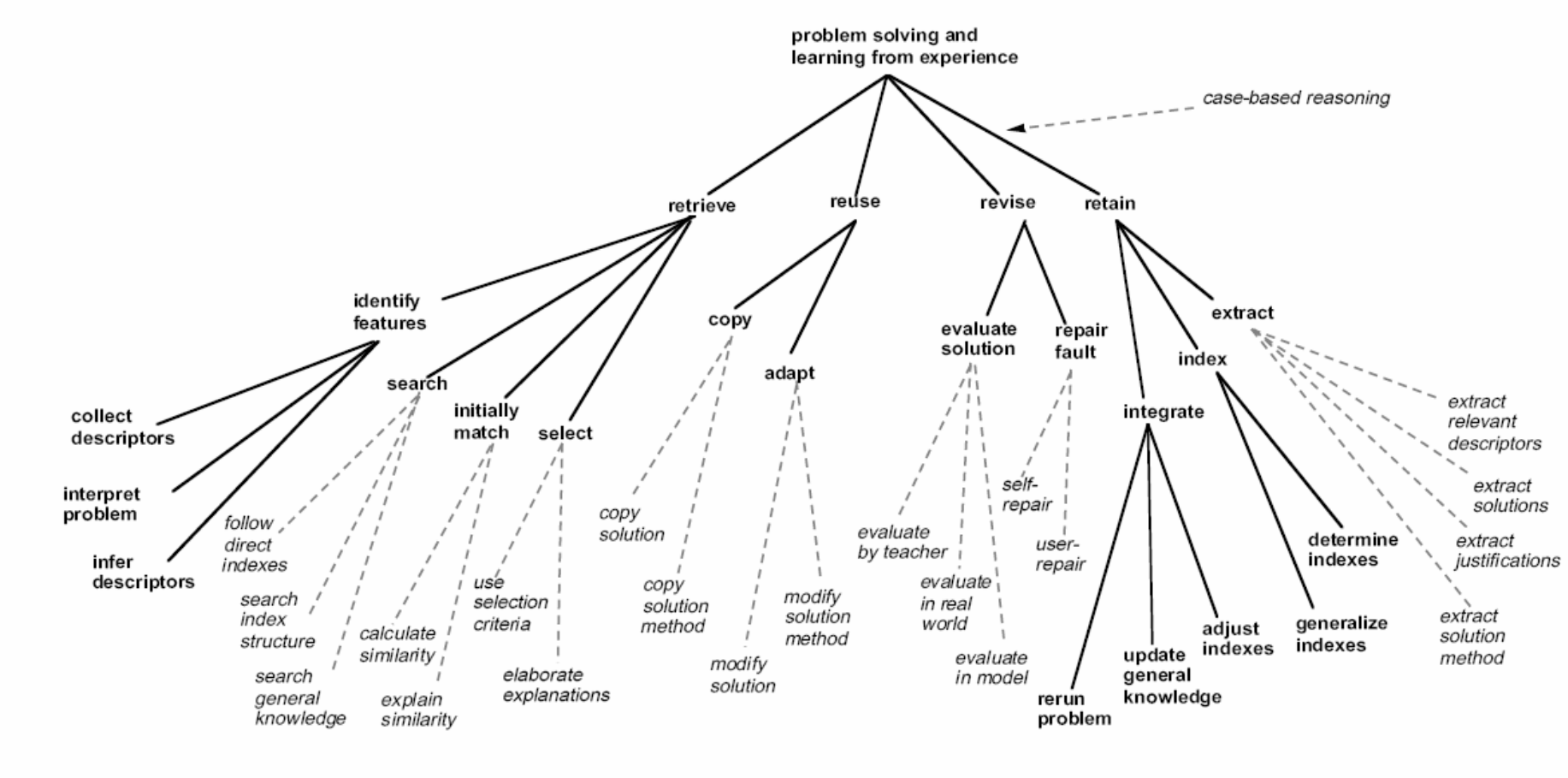

Figura 3. Una descomposición tarea-método de CBR

2.3. Representación del conocimiento y métodos básicos

Los métodos básicos de representación del conocimiento son (1) las bases de datos, tanto relacionales como orientadas a objetos, (2) la lógica de predicados y (3) los sistemas basados en reglas, que explicaremos brevemente en este apartado.

Las bases de datos imponen una estructura a los datos que almacenan. Esta estructura hace que el significado no esté en un único nivel sino que se encuentre contenido en las relaciones entre datos. En las BD relacionales los datos se almacenan una única vez para que sea consistente, las relaciones entre datos se definen usando tablas relacionales y claves primarias y foráneas. Podemos realizar consultas para obtener información y modificar los datos de una tabla sin tener que alterar otra.

En las BD orientadas a objetos, ciertos registros individuales son almacenados como instancias de clases. La clasificación de las entidades del mundo real en jerarquías de objetos y el uso de herencia proporcionan a estas BD una potencia de representación del conocimiento mayor que las BD relacionales.

Otra representación mucho más potente del conocimiento es la lógica de predicados.
En este enfoque declarativo representamos hechos y relaciones entre hechos en forma de reglas IF-THEN que nos permiten inferir nuevo conocimiento.

Los sistemas basados en reglas permiten definir reglas IF-THEN con una serie de ventajas:

- Las reglas se entienden fácilmente
- Las reglas son independientes
- Las reglas encapsulan pequeñas porciones de conocimiento que conjuntamente permiten modelar un problema complejo.
- Las reglas pueden colocarse en cualquier orden en un programa. En estos sistemas el conocimiento se representa como hechos y las reglas permiten manipular los hechos.

A pesar de parecer simple, el proceso se complica porque puede que más de una regla pueda ser aplicada en un momento dado, y la aplicación de una regla hace aplicable muchas más. Por tanto estos sistemas necesitan una estructura de control para decidir la siguiente regla a aplicar y como encadenar reglas.

Existen dos mecanismos de aplicación de las reglas:

- forward Chaining: los nuevos hechos son inferidos de hechos existentes
- backward Chaining: Por ejemplo, si tenemos una regla IF A THEN B, aplicar backward chaining significaría usar el conjunto de reglas existentes para determinar que cosas deben ser ciertas para que B sea cierto. Esta técnica es empleada para probar una hipótesis.

A pesar de la potencia de estos sistemas tienen ciertas limitaciones. (1) Es extremadamente difícil obtener un conjunto de reglas correcto. Este problema es conocido como *cuello de botella de elicitación del conocimiento* y hace muy difícil la recolección de conocimiento en ciertos dominios. (2) Además estos sistemas no

funcionan adecuadamente cuando se carece de un modelo explícito de conocimiento. Estas dos limitaciones impiden construir sistemas de soporte de decisión útiles.

En este punto podríamos pensar que entonces las BD son una tecnología adecuada para la tarea de recuperación de soluciones conocidas de problemas: pueden almacenar gran cantidad de información, mantienen relaciones entre elementos, y proporcionan un rápido acceso a la información. Pero hay que tener en cuenta que antes de recuperar una solución hay que identificar el problema y esto no es soportado eficientemente por las BD convencionales. Para encontrar el problema correcto y su solución conocida debemos de ajustar el nuevo problema a todos los de la base de datos. Sin embargo la representación de los problemas reales es compleja, hay que incluir muchas características, y la representación de la BD puede no ser adecuada. Las BD son excelentes para ajustes exactos pero muy pobres para ajustes parciales.

Para recuperan un emparejamiento de problema necesitamos un método que nos permita:

- describir el problema como lo vemos, en lenguaje natural.
- encontrar el mejor emparejamiento del problema.
- validar el ajuste.
- proporcionar una solución.
- adaptar la solución del problema que empareja a nuestro problema actual.

Esta es la funcionalidad proporcionada por los razonadores basados en casos.

3. La biblioteca de casos

Un caso se puede considerar un registro de una experiencia previa o un problema. La información que se almacena sobre una experiencia depende tanto del dominio como del propósito para el que el caso se usa.

En un sistema CBR se incluirá información acerca de la especificación del problema y atributos del medio que describen el entorno del problema. La descripción del problema debe incluir:

- Las metas que se deben conseguir para resolver el problema.
- Las restricciones de estas metas.
- Las características de la situación del problema y las relaciones entre sus partes.

Otra información muy importante que se debe almacenar es la descripción de la solución, que será usada cuando nos encontremos en una situación similar.

Dependiendo de cómo el sistema razone con los casos, esta descripción puede incluir únicamente los hechos que llevan a la solución o información sobre pasos adicionales en el proceso de obtención de la solución. Además la descripción de la solución también podrá incluir:

- Las justificaciones de las decisiones tomadas en la solución.
- Soluciones alternativas que no fueron elegidas y porqué.
- Soluciones no admisibles y porqué fueron rechazadas.

- Expectativas acerca del resultado de la solución.

También es importante incluir una medida del éxito si en la base de casos se han logrado soluciones con diferentes niveles de éxito o fracaso.

Además también se debe incluir información acerca del resultado:

- Si el resultado cumple o no con las expectativas y una explicación.
- Si el resultado fue un éxito o un fracaso.
- Una estrategia de reparación.
- Qué podría haberse hecho para evitar el problema
- Un puntero al siguiente intento de solución (resultado de aplicar la reparación)

El conocimiento que almacena un caso es específico, no abstracto, y todo el conocimiento relacionado (es decir, conocimiento aplicable en una circunstancia concreta) se encuentra cerca en la base de casos, esto quiere decir que el conocimiento que necesitemos para resolver un problema específico lo encontraremos agrupado en unos pocos casos o incluso en uno.

En los sistemas CBR la base de casos es la memoria de casos almacenados. A la hora de construir la memoria debemos tener en cuenta los siguientes aspectos:

- La estructura y representación de los casos.
- El modelo de memoria usado para organizar los casos.
- Los índices empleados para identificar cada caso.

3.1. Representación de casos y modelos de memoria

Los casos pueden representar distintos tipos de conocimiento y pueden ser almacenados en diferentes formatos. Esto dependerá del tipo de sistema CBR, por ejemplo los casos pueden representar personas, objetos, diagnósticos, planes, etc.

En muchas aplicaciones prácticas los casos se representan como dos conjuntos desestructurados de pares *atributo-valor* que representan el problema y las características de la solución.

Sin embargo, es muy difícil decidir exactamente qué representar. Por ejemplo en un sistema CBR para diagnóstico médico, un caso podría representar el historial médico completo de un paciente o solamente una visita. En caso de ser una única visita, el caso podría ser un conjunto de síntomas junto con el diagnóstico y el tratamiento prescrito. En este caso no se considera la evolución del paciente lo cual sería útil para resolver un problema futuro similar. Si el caso representa el historial médico completo del paciente, se puede incorporar información sobre la evolución del paciente desde una visita a la siguiente. Sin embargo este formato dificulta la búsqueda de un conjunto particular de síntomas para obtener un tratamiento.

En una representación orientada al objeto, el caso sería como muestra la siguiente figura.

Figura 4. Caso de diagnóstico en representación objeto

Una de las ventajas del razonamiento basado en casos es la flexibilidad que ofrece respecto a la representación. Se puede elegir la implementación adecuada dependiendo del tipo de información a representar, variando desde un simple boolean, un número, datos dependientes del tiempo, relaciones entre datos, ficheros, frames, redes semánticas, etc.

A la hora de elegir una representación para un caso se deben tener en cuenta los siguientes factores:

- La estructura interna del caso, si está formado por subcasos o componentes.
- El tipo y estructura de la información que va a describir el caso debe estar disponible o ser posible crearlo.
- El lenguaje ó *shell* en el que se va a implementar el sistema CBR. La elección del lenguaje ó *shell* también estará influenciada por la disponibilidad de los mismos y los conocimientos de que disponga el desarrollador.
- El mecanismo de búsqueda e indexación que se vaya a emplear. El formato del caso debe permitir la recuperación de casos de forma eficiente.
- La forma en la que los casos están disponibles o son obtenidos. Por ejemplo si se crea una base de casos a partir de una colección existente de experiencias pasadas, la facilidad con la que se puedan traducir estas experiencias a un formato adecuado es importante para el sistema CBR.

Independientemente de la representación que elijamos siempre debemos tener en cuenta que la información que almacene un caso debe ser relevante tanto para el propósito del sistema como para asegurar que siempre será elegido el caso más apropiado para solucionar un nuevo problema en un determinado contexto.

En muchos sistemas CBR no se necesitan almacenar todos los casos existentes, sino que se sigue un criterio para decidir que casos almacenar y cuales descartar.
Por ejemplo, si disponemos de dos casos muy similares para una misma situación sólo almacenaremos uno de ellos, o podríamos crear un caso artificial que fuese una generalización de dos o más casos concretos.

Una vez elegida la representación de los casos, la elección del modelo de memoria es el siguiente paso. Existen principalmente dos estructuras de memoria, plana y jerárquica.

En una estructura *plana* de la base, los casos se almacenan secuencialmente, mientras que en una estructura *jerárquica* los casos se agrupan en categorías para reducir el número de casos a buscar en una consulta.

Dependiendo del método de búsqueda que apliquemos a estas dos estructuras obtenemos una serie de estructuras de memorias derivadas que explicaremos a continuación.

Utilizaremos el siguiente ejemplo para ilustrar las estructuras de memoria: supongamos que nuestros casos van a representar frutas y que las características que queremos almacenar son el color, el sabor y la temporada.

3.1.1 Memoria plana, búsqueda secuencial (HYPO)

Los casos se almacenan secuencialmente en una lista simple, un arreglo o un fichero.
Para lograr una recuperación eficiente, se indexan los casos de la base. En este método los índices se eligen para representar los aspectos importantes del caso y la recuperación involucra comparar las características consultadas con cada caso de la base de casos.

Este tipo memoria presenta la ventaja de que añadir nuevos casos resulta muy “barato” (rápido y fácil de implementar). No ocurre así con la recuperación de casos, ya que resulta muy lento cuando el número de casos en la base es alto.
Nuestra base de frutas sería de la siguiente manera:

Figura 5. Casos de la base de frutas

Por ejemplo, considerando que el sabor es más importante que el color y la temporada y que `ROJA y NARANJA` pueden emparejarse parcialmente, ante el caso de entrada `ROJA, ÁCIDA, INVIERNO`, el sistema operaría de la siguiente manera:

Caso comparado	Resultados de la comparación		
	color	*sabor*	*temporada*
níspero	parcial	si	no
naranja	parcial	si	si
manzana	no	no	si
albaricoque	parcial	no	no

Figura 6. Resultados de la comparación

3.1.2 Memoria jerárquica, búsqueda secuencial:

- Redes de características compartidas:

Los casos se almacenan en un árbol o en un grafo acíclico directamente. El grafo subdivide el espacio de casos de acuerdo con los atributos que les caracterizan (*clustering*). Las características comunes ocupan nodos del grafo de donde cuelgan los casos que las comparten. La mayoría de los sistemas tienen alguna especie de umbral relativo al número de casos que debe compartir el valor de una característica, de modo que se justifique su posición en un nodo.

Este procedimiento es computacionalmente más caro, ya que la base requiere un mayor espacio de almacenamiento y su actualización resulta más compleja ya que requiere generalmente modificar una parte de la red. Por otra parte, el diseño y mantenimiento de la red óptima es costoso. Los casos recuperados dependen del contenido de los nodos: la jerarquía establecida debe responder a la importancia de las características.

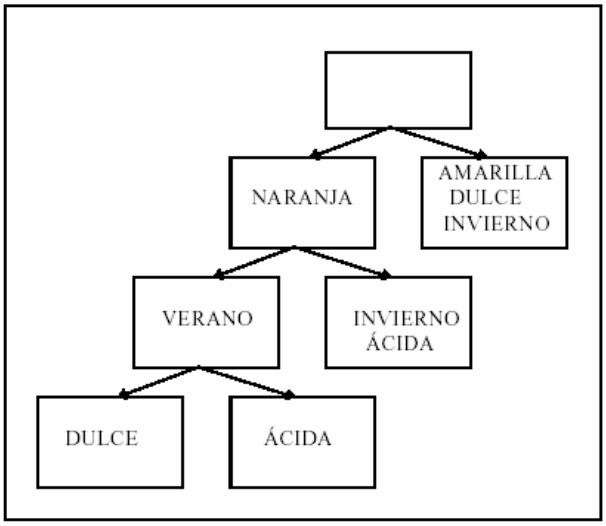

Figura 7. Subdivisión de casos

El algoritmo de búsqueda del caso que mejor se ajusta al de entrada es también sencillo:

```
Inicializar N = nodo padre
      Repetir hasta que N sea un caso:
            Encontrar el nodo bajo N más similar a la entrada.
Devolver N.
```

Asumiendo de nuevo que ROJA y NARANJA son parcialmente equiparables y considerando la entrada ROJA VERANO DULCE:

Característica	Valor	Resultado de la comparación
COLOR	NARANJA	parcial

Figura 8

Ir a los nodos bajo NARANJA

TEMPORADA	VERANO	si

Figura 9

Ir a los nodos bajo VERANO

SABOR	DULCE	si

Figura 10

El caso recuperado como más similar a la entrada es el CASO 4 (albaricoque), obtenido de forma mucho más eficiente que si se hubiera utilizado una memoria plana.
Si se considera la característica SABOR más importante que la característica TEMPORADA y esta a su vez es más importante que la característica COLOR, el diseño de la base de casos debería ser:

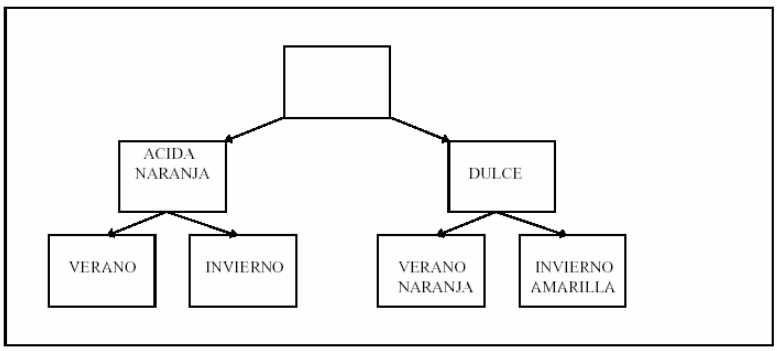

Figura 11. Nueva subdivisión cambiando importancias

- Redes de discriminación con prioridades (CHEF):

Los tipos de sistema anteriores no resuelven el problema de búsqueda cuando la entrada está incompleta. En las redes de discriminación con prioridades cada nodo contiene una pregunta para la cual los subnodos correspondientes ofrecen respuestas alternativas. Las preguntas más importantes se formulan primero, situándose más arriba en la jerarquía.

Así como las redes de características compartidas, las redes de discriminación subdividen el conjunto de casos y comparten la mayoría de las ventajas y desventajas de esta técnica. La formulación de preguntas puede implementarse más eficientemente que el emparejamiento en cada subnodo. Al mismo tiempo, separar los atributos de sus valores particulares hace más fácil identificar qué atributos han resultado más útiles para la caracterización del caso. Las desventajas que plantean son las mismas que las de las redes de características compartidas: algunos casos significativos pueden ignorarse si la ordenación de las preguntas no es la óptima.

Así por ejemplo, si se asume `SABOR` más importante que `TEMPORADA` y esta característica a su vez más importante que `COLOR`, las preguntas correspondientes se formularán según este orden de importancia. El algoritmo de búsqueda resulta:

```
Inicializar N = nodo padre
      Repetir hasta que N sea un caso:
            Formular pregunta en N sobre la entrada
            Asignar N = subnodo conteniendo la respuesta
Devolver N
```

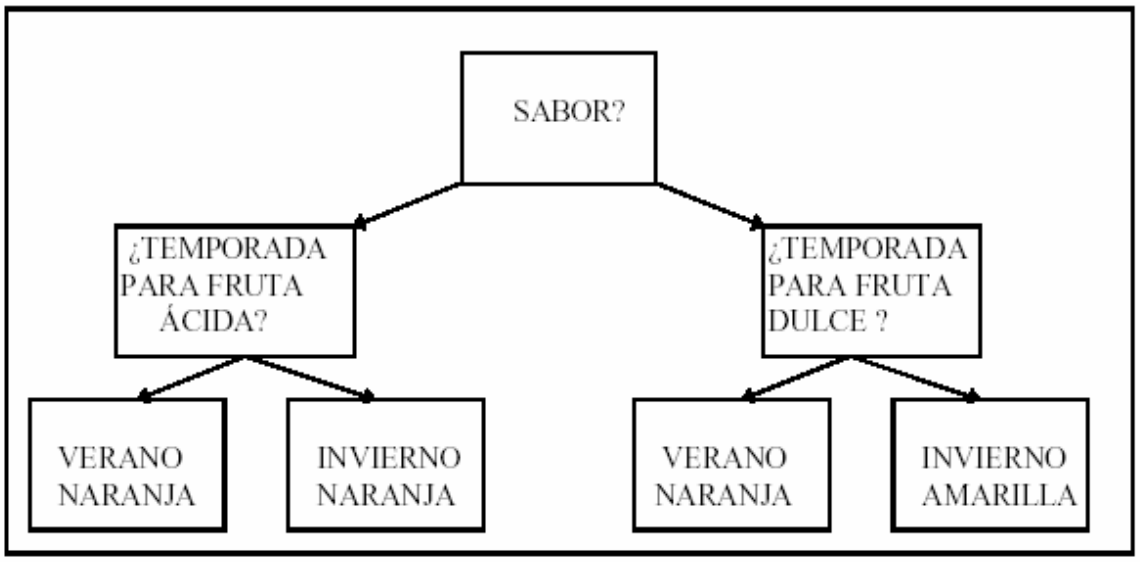

Figura 11. Subdivisión de casos

Considérese la entrada `ÁCIDA INVIERNO NARANJA`. El algoritmo operaría del modo siguiente:

```
¿SABOR? –> ÁCIDA
¿TEMPORADA? –> INVIERNO
Devolver ÁCIDA INVIERNO NARANJA
```

- Redes de discriminación redundante o modelo de memoria dinámica (CYRUS, JULIA) :

Las redes de discriminación redundantes resuelven este problema organizando los casos mediante varias redes de discriminación, cada una con una ordenación diferente de las preguntas. La variedad más común de las redes redundantes incluye en su organización las propiedades de las redes de características compartidas para mantener su tamaño bajo control. Explicaremos más en detalle este modelo de memoria dinámica por tratarse del más comúnmente identificado con los sistemas RBC, desarrollado a partir de la teoría general de *MOPs* (del inglés, *Model Organisation Packets*: paquetes de organización de memoria) de Schank. En este modelo la memoria de casos es una estructura jerárquica de episodios generalizados.

La idea básica es organizar los casos que comparten propiedades similares en una estructura más general (el episodio generalizado o MOP).
Un episodio generalizado (GE) contiene tres tipos de objetos: normas, casos e índices.
Las normas son características comunes a todos los casos indexados bajo un GE. Los índices son las características que nos diferencian entre los casos de un GE.
Un índice puede apuntar a un GE más específico o directamente a un caso. Un índice consta de dos términos: un nombre y un valor.

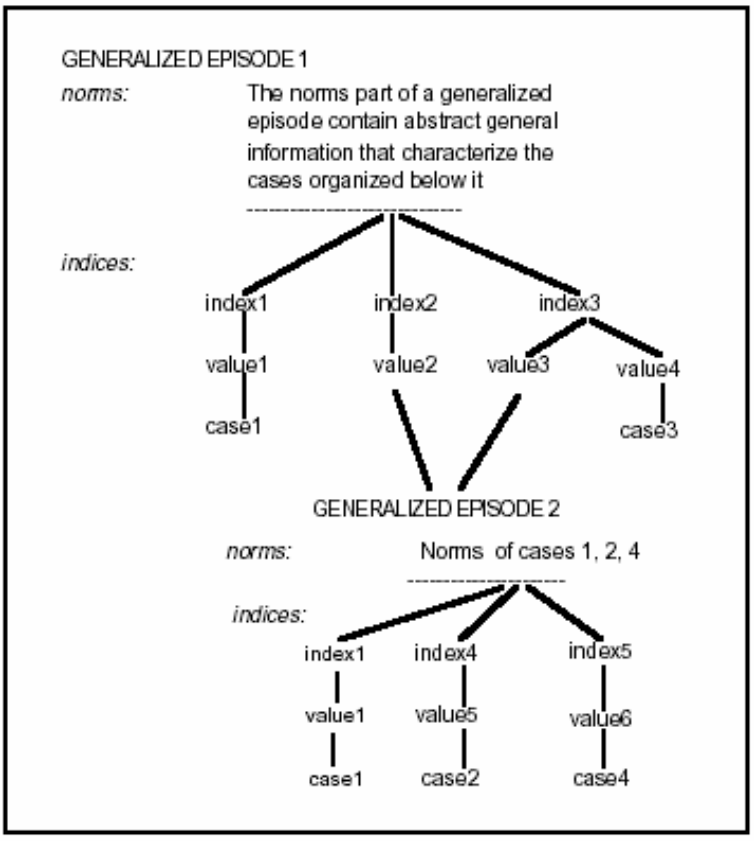

Figura 12. Episodio Generalizado

La figura muestra un complejo GE con los casos y GE más específicos que engloba.
La memoria de casos es una red discriminatoria, de manera que un nodo o es un GE (conteniendo las normas), o un nombre de índice, o un valor de índice o un caso.

Cada par índice-valor apunta desde un GE a otro GE o a un caso. Un valor de índice apunta a un único caso o a un único GE.

El esquema de indexación es redundante, puesto que existen múltiples caminos hasta un caso concreto o un GE. Esto se aprecia en la figura en la indexación del caso 1.

Cuando un nuevo caso es dado se busca el mejor emparejamiento y el nuevo caso se coloca en la red comenzando por el nodo raíz.

El procedimiento de búsqueda para el almacenamiento de casos es similar al que se utiliza en la recuperación de casos. Cuando una o más características del caso ajustan con una o más características de un GE, el caso es distinguido por el resto de sus características.

Normalmente se encuentra el caso que más características tiene en común con el caso de entrada[1]. Durante el almacenamiento de un nuevo caso, cuando una característica del nuevo caso se ajusta a la de un caso existente se crea un nuevo episodio generalizado.
Los dos casos se distinguen indexándolos con diferentes índices bajo el GE.
Si durante el almacenamiento de un caso, dos casos (ó dos GE) finalizan bajo el mismo índice, automáticamente se crea un nuevo GE. Por tanto, la memoria de casos es dinámica en el sentido de que las características que comparten dos casos son dinámicamente generalizadas en un GE y los casos son indexados bajo ese GE por sus características diferenciadoras.

Para la recuperación de un caso, se busca el GE que tiene más normas en común con la descripción del problema. Una vez encontrado se recorren los índices bajo el GE para encontrar el caso que contiene más características adicionales en común con el problema.

El almacenamiento de un nuevo caso se realiza del mismo modo, con el proceso adicional de la creación dinámica de GE.

Puesto que la estructura de índices es una red discriminatoria, un caso (ó un puntero a un caso) es almacenado bajo un índice que le diferencia del resto de casos. Esto puede hacer crecer de forma explosiva el número de índices a medida que aumenta el número de casos.

La mayoría de los sistemas que usan este esquema de indexación ponen algunos límites a la hora de seleccionar índices para casos. En CYRUS, por ejemplo, sólo se permite un pequeño vocabulario de índices.

CASEY almacena una gran cantidad de información en sus casos. Además para todas las características observadas, almacena la explicación causal para el diagnóstico encontrado, así como una lista de estados en el modelo de ataque al corazón, para los cuales había evidencia en el paciente. Estos estados, denominados estados causales generalizados, son los índices primarios de los casos.

---

[1] Este criterio de similitud se usa para ilustrar el método pero podría ser otros como priorizar el emparejamiento de un subconjunto de características o valorar la similitud de otra manera.

El criterio de valoración de similitudes se puede emplear para guiar la búsqueda, por ejemplo identificando qué índices se siguen primero si se hace una elección .

El principal rol de un episodio generalizado es representar una estructura indexada para el emparejamiento y recuperación de casos.
Las propiedades dinámicas de esta organización de memoria podrían ser vistas como un intento de construir una memoria que integra conocimiento de episodios específicos con conocimiento generalizado de los mismos episodios.

Por tanto, esta organización del conocimiento es adecuada para el aprendizaje tanto de conocimiento generalizado como de conocimiento específico y es un modelo admisible , aunque simplificado, del razonamiento y aprendizaje humano.

La siguiente figura muestra algunas ramas de la red de discriminación redundante correspondiente a nuestro ejemplo.

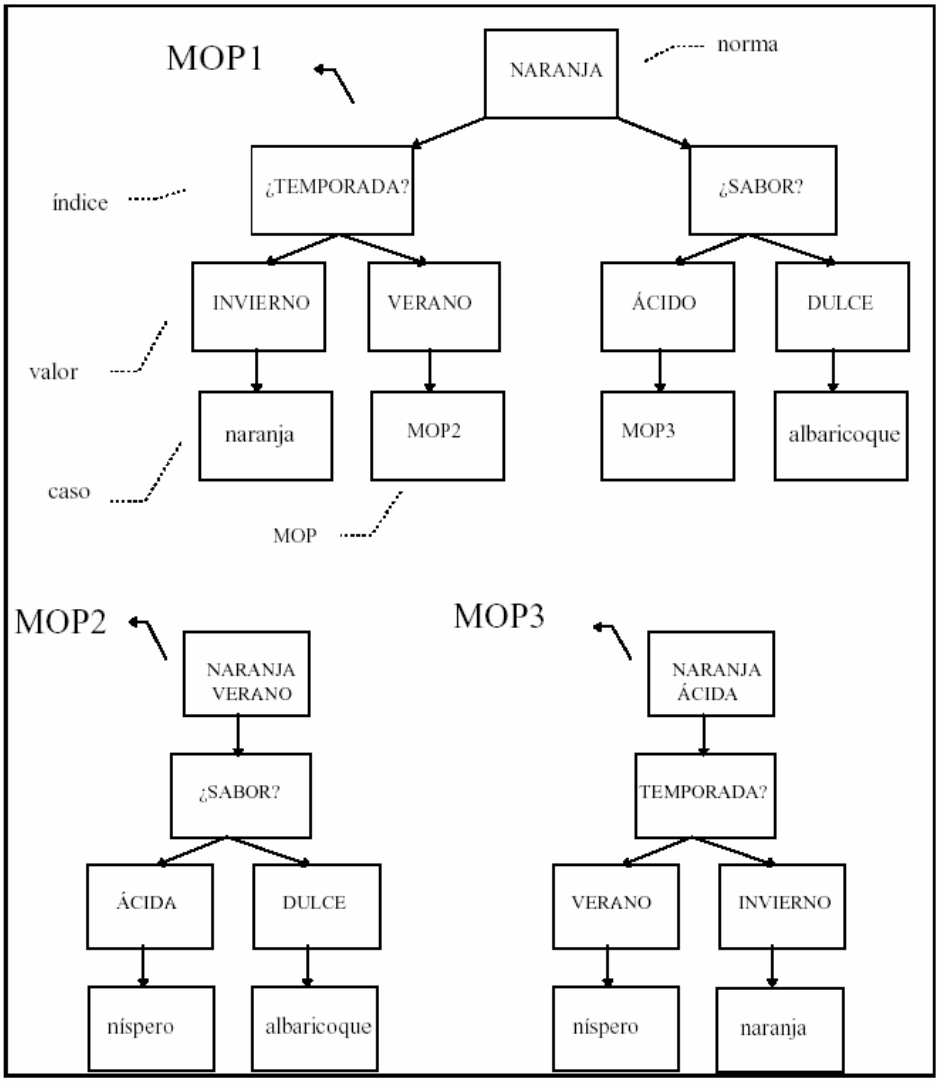

Figura 13. Red discriminatoria redundante

El algoritmo de búsqueda aplicable es el siguiente:

```
Algoritmo de discriminación en paralelo en cada red
Devolver el conjunto unión de los casos encontrados
```

```
Comparar la entrada con cada elemento del conjunto para
identificar los mejores casos
```

Consideremos la entrada `NARANJA INVIERNO`:

```
Preguntas en el nivel 1:
¿TEMPORADA?
Respuesta: naranja: TEMPORADA = INVIERNO
```

Consideremos ahora la entrada `NARANJA VERANO ÁCIDA`:

```
Preguntas en el nivel 1:
¿TEMPORADA?
Respuesta MOP2: TEMPORADA = VERANO
¿SABOR?
Respuesta MOP3: SABOR = ÁCIDA
Preguntas en el nivel 2:
bajo MOP2 ¿SABOR?
Respuesta níspero: SABOR = ÁCIDA
bajo MOP3 ¿TEMPORADA?
Respuesta níspero: TEMPORADA = VERANO
```

- Modelo categoría-ejemplar (PROTOS):

En este modelo los casos se denominan *ejemplares*. Las bases psicológicas y filosóficas de este método son la visión de que el "mundo real", los conceptos naturales, deberían ser definidos de forma extensiva. Además, se asignan diferentes importancias a las distintas características al definir la pertenencia de un caso a una categoría. Cualquier intento de generalizar un conjunto de casos debe hacerse cautelosamente.
Esta visión de representar los conceptos es la base de este modelo de memoria.

La memoria de casos está empotrada en una estructura de red de categorías, casos índices y punteros. Cada caso se asocia con una categoría. Un índice puede apuntar a un caso o a una categoría. Existen tres tipos de índices:

- Enlaces de característica: apuntan desde los descriptores el problema (características) a los casos o categorías (llamados *recuerdos*).
- Enlaces de caso: apuntan desde las categorías a sus casos asociados (llamados *enlaces de ejemplar*).
- Enlaces de diferencia: apuntan desde los casos a los casos vecinos que sólo se diferencian en un número pequeño de características.

Normalmente, una característica se describe por un nombre y un valor. Los ejemplares de una categoría son almacenados de acuerdo al grado en que son prototipos de la categoría. La siguiente figura ilustra parte de esta estructura de memoria.

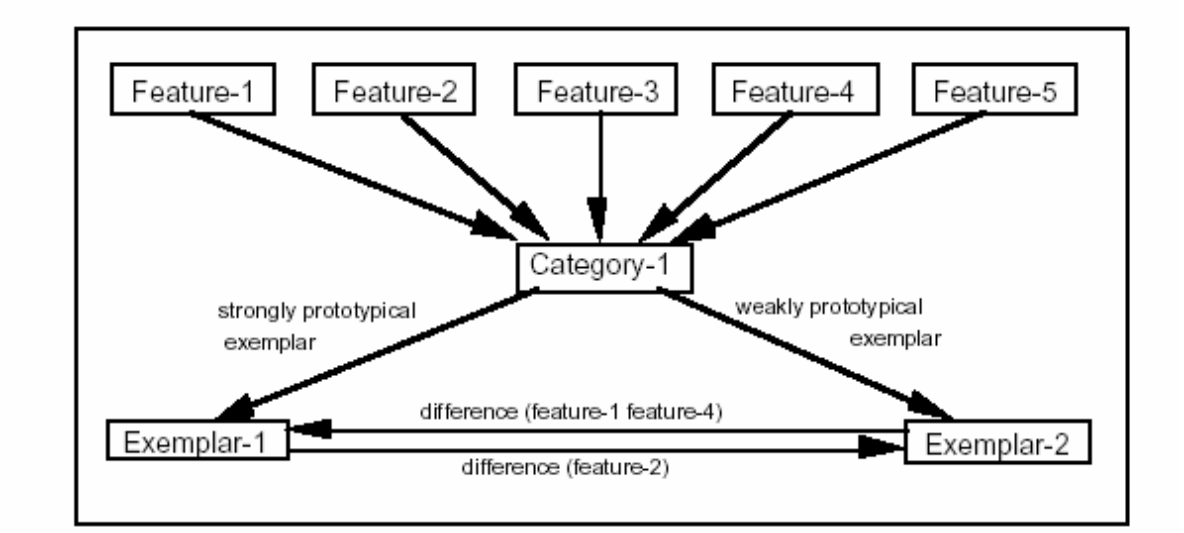

Figura 14. Composición de una categoría

Los índices anónimos son recuerdos desde las características a una categoría. Dentro de la organización de la memoria las categorías son entrelazadas en una red semántica la cual también contiene las características y estados intermedios (subclases de los conceptos *objetivo*) pero referenciadas de otra forma.

Esta red representa la base del conocimiento general del dominio, el cual da soporte explicativo para algunas tareas del CBR. Por ejemplo, un mecanismo clave de emparejamiento de casos es un método llamado "*emparejamiento de patrones basado en conocimiento*".

La búsqueda de un caso en la base de casos que ajuste a una descripción de entrada, se realiza combinando las características de entrada de un problema en un puntero al caso o categoría que comparte la mayoría de las características. Si un recuerdo apunta directamente a una categoría, se recorren los enlaces a sus casos más y estos casos son devueltos. Como ya dijimos, el conocimiento general del dominio es usado para permitir el emparejamiento de características semánticamente similares. Un nuevo caso es almacenado buscando un emparejamiento de caso, y estableciendo los índices de características apropiados. Si en el emparejamiento se encuentra un caso que apenas se diferencia del caso de entrada, puede que el nuevo caso no sea almacenado, o que se fusionen ambos casos siguiendo enlaces taxonómicos en la red semántica. La estructura para nuestro ejemplo sería la mostrada en la siguiente figura.

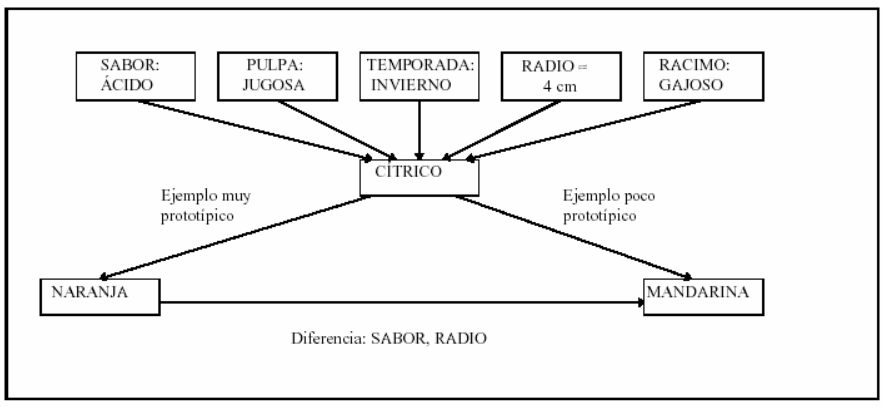

Figura 15. Estructura para el ejemplo de las frutas

3.1.3 Memoria plana, búsqueda paralela

Los casos se almacenan como vectores de características en una memoria de contenido *direccionable* en un hardware paralelo, que permite la búsqueda paralela. La recuperación y emparejamiento de casos se lleva a cabo en una única etapa y añadir nuevos casos resulta poco costoso ya que no hay necesidad de subdividir la base mediante índices. El principal problema de estas memorias es su coste hardware elevado y la limitada sensibilidad al contexto del procedimiento de equiparación, al tratarse sólo características superficiales. Una variación de este método utiliza recuperación múltiple para derivar características no superficiales (este tema se tratará más adelante).

3.1.4 Memoria jerárquica, búsqueda paralela

Utilizan asimismo hardware paralelo. Operan almacenando un par *atributo-valor* en cada procesador. Los casos se ordenan jerárquicamente en una red de características compartidas para evitar la necesidad de un excesivo número de procesadores.
La principal ventaja de la búsqueda paralela está en que la recuperación no está restringida por el orden de las características ni la elección de los índices. Su inconveniente principal radica en su alto coste.

La elección del modelo de memoria depende de los siguientes factores:

- La representación usada en la base de casos.
- El propósito del sistema CBR. Por ejemplo una estructura jerárquica es adecuada para un sistema dedicado a resolver problemas de clasificación.
- El número y complejidad de los casos almacenados. A medida que el número de casos aumenta en la base, un modelo de memoria de búsqueda secuencial (estructura plana), hace aumentar el tiempo de recuperación.
- El número de características usadas para el emparejamiento de casos durante una búsqueda.
- La existencia de casos suficientemente similares como para agruparlos.
- El conocimiento que se tiene sobre el dominio específico. Esto influye en la habilidad para determinar si los casos son similares. Si se dispone de muy poco conocimiento sobre el dominio es posible que la agrupación de casos sea incorrecta.

En conclusión, se asume que los casos tienen dos componentes: la especificación del problema y la solución. Normalmente, la especificación del problema suele consistir en un conjunto de atributos y valores. Los atributos de un caso deben definir el caso de forma única y deben ser suficientes para pronosticar una solución. La representación puede ser una estructura de datos plana o una compleja jerarquía de objetos.

3.2 Indexación de casos

La indexación de casos consiste en asignar índices a los casos para una futura recuperación y comparación. Los índices de un caso determinarán en que contexto el caso será recuperado, por tanto, los índices de un caso deben reflejar las características importantes del caso y los atributos que influyen en el resultado del caso y describir las circunstancias en las que se espera que un caso será recuperado.

Por ejemplo, si una comida resulta un fracaso porque un invitado vegetariano no podía comer carne, será conveniente indexar tal comida por las características vegetariano y carne. Del mismo modo, si un diagnóstico resultó especialmente problemático por la presencia de un conjunto de síntomas atípicos, tal caso debería indexarse mediante los citados síntomas.

Los índices deben ser lo suficientemente abstractos como para permitir recuperar un caso en todas las circunstancias en las que es útil pero no demasiado abstracto. Cuando los índices de un caso son demasiado abstractos, el caso puede ser recuperado en muchas situaciones, o se requiere demasiado procesamiento para el emparejamiento de casos.

Los índices principalmente tienen tres aplicaciones:

- Índices físicos reales en una red de discriminación (redundante o no) guían y dirigen la búsqueda a través de la memoria de casos (es el uso habitual en la mayoría de las memorias de casos)
- Apuntando a características relevantes de los casos como criterio de decisión a la hora de escoger el mejor caso.
- Dentro de funciones de evaluación sirven para asignar los pesos de más alto valor a las características más importantes.

Criterios de indexado:

- En cada característica disponible. Sólo aplicable cuando el sistema contempla una pocas características o lleva a cabo la recuperación en paralelo.
- Por un conjunto de características fijo, bien definido.
- Por las características que se han mostrado predictivas en el pasado.
- Por las diferencias entre lo que está en memoria y el caso que está siendo indexado. De este modo se controla la redundancia de la red.
- Por las características que predigan un fallo o un éxito
- Después de analizar la razón del fallo, la explicación se generaliza y se indexa con la combinación de características que se considera responsable del fallo.
- Por las características que se consideran útiles para conseguir un objetivo o desarrollar alguna tarea. Un razonador sabe lo que implica cada una de sus decisiones. La explicación de esta implicación se generaliza por técnicas de aprendizaje basado en explicación y el indexado se hace a través de la combinación de características que son responsables de la consecución de los objetivos o del desarrollo de la tarea.

4. Recuperación de casos

La tarea de recuperación comienza con una descripción del problema (parcial), y finaliza cuando se encuentra un caso anterior, que es el que mejor ajusta.
La tarea de recuperación se divide en varias etapas: Identificación de características, Emparejamiento inicial, Búsqueda y Selección.
La tarea de identificación consiste en proporcionar un conjunto de descriptores de problema relevantes, el objetivo de la tarea de emparejamiento es devolver un conjunto de casos suficientemente similares al nuevo caso –dado un criterio de similitud de algún tipo y la tarea de selección elige entre esos casos el que mejor se ajusta. A continuación

estudiaremos con más detalle las cuatro etapas de la recuperación así como los principales criterios de similitud y algunos de los algoritmos más utilizados.

4.1 Algoritmos de recuperación

En el apartado 3.1 explicamos en detalle los distintos modelos de memoria y para cada modelo vimos un pequeño esquema del algoritmo de recuperación empleado, en este apartado veremos más en profundad las dos principales clases de algoritmos existentes.

Los procesos involucrados en la recuperación de un caso de una base de casos son muy dependientes del modelo de memoria y de los procedimientos de indexación usados.

Los algoritmos de recuperación utilizados varían desde una búsqueda del vecino más cercano hasta el uso de agentes inteligentes.

Los algoritmos de recuperación más investigados son: K-vecinos más cercanos, árboles de decisión y sus derivados. Estas técnicas necesitan de una medida de similitud para determinar la proximidad entre casos por ejemplo la distancia *euclídea*, de *hamming* o de *levenshtein.*

1. K-Vecinos más cercano: el caso recuperado es aquel en el que la suma de los pesos de las características que ajustan con las del caso nuevo es mayor que la de otros casos que ajustan. Esto quiere decir, que si los pesos de las características son iguales, un caso que ajusta con n características será recuperado antes que otro caso que empareja con k características, siendo k<n. Se puede asignar un peso mayor a las características consideradas más importantes.

   Una típica función de evaluación empleada para encontrar el vecino que mejor empareja es la mostrada a continuación

$$similarity(Case_I, Case_R) = \frac{\sum_{i=1}^{n} w_i \times sim(f_i^I, f_i^R)}{\sum_{i=1}^{n} w_i}$$

   Donde w_i es el peso de importancia de una característica, *sim* es la función de similitud de las características y f_i^I y f_i^R son los valores para una característica *i* de la entrada y del caso recuperado respectivamente.
   La siguiente figura muestra un esquema simple del ajuste en el vecino más cercano. El caso3 es el seleccionado como el vecino más cercano porque *similarity*(*NC, case3*)> *similarity*(*NC, case1*) and *similarity*(*NC, case3*)> *similarity*(*NC, case2*).

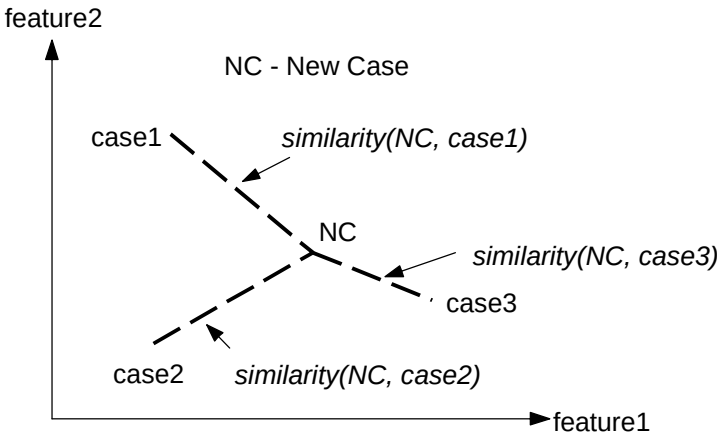

Figura 16. Estructura vecino más cercano

2. Algoritmos inductivos: La inducción es una técnica desarrollada por los investigadores de aprendizaje de máquinas para extraer reglas o construir árboles de decisión a partir de datos pasados. En los sistemas CBR la base de casos es analizada por un algoritmo de inducción que determina que características son las que mejor diferencian los casos entre sí para producir un árbol de decisión que clasifica (o indexa) los casos. El algoritmo inductivo más utilizado es ID3.

   Este enfoque es muy útil cuando se requiere un único caso como solución y cuando la característica del caso depende de otras.

   En las siguientes figuras mostramos el árbol de decisión generado para los datos de la tabla. La tarea es predecir el estado de un préstamo a partir de las características del solicitante del préstamo (ingresos, estado de su trabajo y la devolución).

Nº de caso	Estado del préstamo	Ingresos mensuales	Estado del trabajo	Devolución
Caso 1	Bueno	$2000	Empleado	$200
Caso 2	Muy malo	$4000	Empleado	$600
Caso 3	Muy bueno	$3000	En paro	$300
Caso 4	Malo	$1500	Empleado	$400

Figura 17. Cuatro casos de préstamos

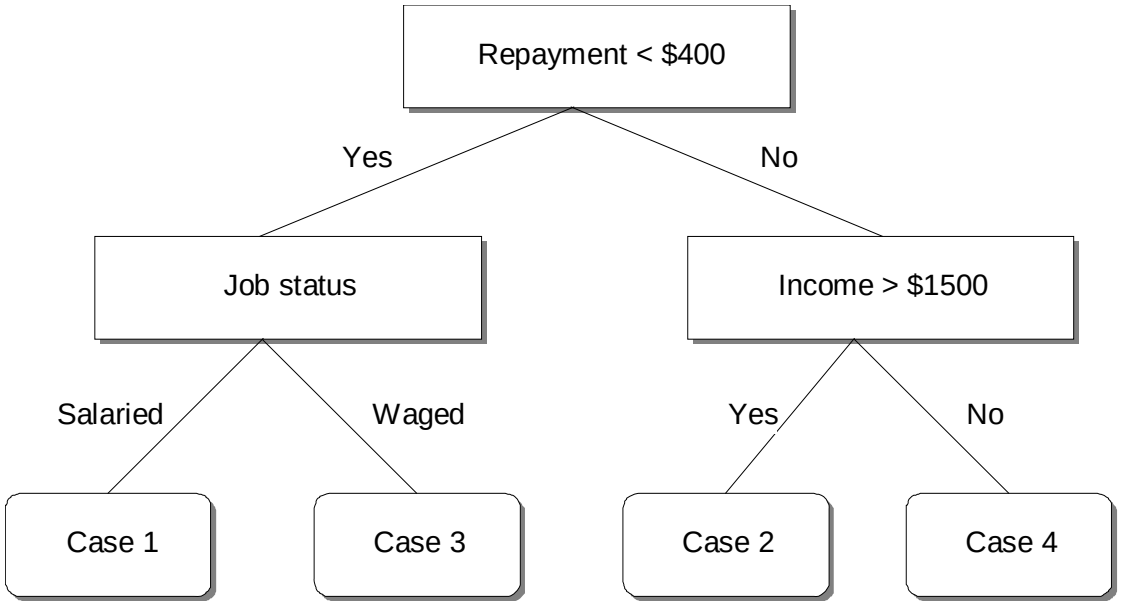

Figura 18. Características del solicitante

Nº de caso	Estado del préstamo	Ingresos mensuales	Estado del trabajo	Devolución
Caso X	?	$1000	Empleado	$600

Figura 19. Un caso Objetivo

Si partimos del caso mostrado en la tabla anterior como objetivo, para determinar el estado del préstamo de este caso el algoritmo recorre el carbol de decisión y busca el caso de la base de casos que mejor ajusta. Para la devolución del préstamo el algoritmo primero selecciona la rama izquierda. Después, llega hasta el nodo (Income>$1500) y selecciona la rama izquierda basándose en el mes de ingreso. De esta manera podemos predecir que el caso que mejor ajusta es el caso 4. Esto sugiere que la concesión del préstamo es mala porque la salida del caso 4 es mala.

Vecino más cercano vs. Inductivo

El vecino más cercano es una técnica muy simple que proporciona una medida de cómo de similar es un caso objetivo a un caso de la base. Pero su principal desventaja es la velocidad de recuperación ya que para encontrar el caso que mejor ajusta, el caso objetivo debe compararse con cada caso de la base de casos. Esto quiere decir que una comparación de similitud (distancia) debe calcularse para cada característica indexada. Esto hace que el algoritmo sea ineficiente la a medida que aumenta la base de casos.

Por ejemplo para una base de 100 casos con 10 características indexadas habría que realizar 1000 cálculos de similitud. Si el tamaño de la base aumenta a 10000 casos tendríamos que realizar 100000 cálculos de similitud.

Una solución parcial es calcular, antes de la recuperación, una posición para cada caso de la base en el espacio n-dimensional y usarlo como índice. En tiempo de ejecución se identifican los casos que no ajustan por estar a una cierta distancia del caso objetivo. De esta manera se consigue un tiempo de recuperación aproximadamente constante, a pesar del tamaño de la base. De todas formas hay que tener en cuenta que el cálculo de los índices es un proceso que consume recursos y habría que recalcularlos cada vez que un nuevo caso es añadido a la base.

El método inductivo depende de la *preindexación* ya que el árbol de decisión es construido antes de que la recuperación comience. Esto es también un proceso que consume recursos para una base de casos grande y tiene que repetirse el proceso cada vez que un nuevo caso es añadido a la base. Pero los tiempos de recuperación son extremadamente rápidos y es más lento a medida que aumenta la base.
Sin embargo la principal desventaja de las técnicas inductivas es que si los datos de un caso se pierden o son desconocidos no se puede recuperar el caso. El vecino más próximo es menos sensible a pérdidas.

La elección de un método u otro dependerá de la experiencia. En general es preferible el vecino más cercano sin ningún *preíndice* hasta el tiempo de recuperación. Si el tiempo de recuperación es crítico es preferible el método inductivo o *preindexar* la base de casos.

En algunas herramientas CBR se emplean ambas técnicas: se usa indexación inductiva para recuperar un conjunto de casos que ajustan y vecino más cercano para hacer un ranking de casos en el conjunto basándose en su similitud con el caso objetivo.

4.2 Etapas y criterios de similitud

4.2.1 Criterios de similitud

Algunos enfoques recuperan un caso anterior basándose en similitudes sintácticas superficiales de los descriptores del problema (CYRUS), otros enfoques intentan recuperar los casos basándose en similitudes semánticas (PROTOS, CASEY).

Para emparejar casos basándonos en las similitudes semánticas y la relativa importancia de las características, se necesita un conjunto extensivo del conocimiento general del dominio para dar una explicación de porque dos casos emparejan y como de fuerte es el emparejamiento.

La similitud sintáctica, a veces llamada "*conocimiento-pobre*" presenta ventajas en dominios donde es muy difícil o imposible adquirir conocimiento general.
Por otro lado, los métodos orientados a la semántica, "*conocimiento-intensivo*"[2] son capaces de usar el significado contextual de la descripción del problema en el emparejamiento, en dominios donde el conocimiento general está disponible.

[2] Los métodos sintácticos también pueden contener un montón de conocimiento general del dominio, implícito en sus métodos de emparejamiento. La distinción entre conocimiento-pobre y conocimiento-intensivo está relacionada con la representación explícita del conocimiento del dominio. Es decir, se refiere al conocimiento generalizado del dominio, ya que los casos también contienen conocimiento explícito, pero éste es conocimiento específico del dominio.

Un factor a tener en cuenta cuando elegimos una estrategia de recuperación es el propósito de de la tarea de recuperación. Si el propósito de la recuperación es adaptar el caso recuperado para una futura reutilización, esto puede llevarse a cabo en el método de recuperación. La "*recuperación para adaptación*" ha sido utilizada en el razonamiento por analogías y en la resolución de problemas de diseño.

4.2.2. Etapas

4.2.2.1 Identificación de características

La identificación del problema puede simplemente consistir en el reconocimiento de sus descriptores de entrada. Pero en los métodos de conocimiento intensivo se requiere un enfoque más elaborado, se intenta comprender el problema dentro de su contexto. Los descriptores desconocidos pueden ser omitidos o explicados por el usuario.
En PROTOS si una característica de entrada es desconocida se pide al usuario una explicación, que sirve para enlazar la característica en la red semántica existente (estructura de categorías).

Para entender el problema es necesario filtrar el ruido de los descriptores del problema, para inferir las características relevantes, para comprobar si los valores de las características tienen sentido dentro del contexto, para generar expectativas de otras características, etc.

Se pueden obtener descriptores adicionales a los dados como entrada, usando un modelo del conocimiento general o recuperando una descripción del problema similar de la base de casos y usar las características de ese caso como características esperadas.
La comprobación de expectativas puede ser hecha en el modelo del conocimiento (casos y conocimiento general) o preguntando al usuario.

4.2.2.2 Emparejamiento inicial

La tarea de encontrar un buen emparejamiento suele dividirse en dos subtareas: Un proceso de emparejamiento inicial en el cual se recuperan un conjunto de posibles candidatos y un proceso más elaborado para seleccionar la mejor del conjunto.

La búsqueda de un conjunto de casos posibles de emparejamiento se hace usando los descriptores del problema (características de entrada) como índices a la memoria de casos, bien sea de forma directa o indirecta.

Existen tres formas de recuperar un caso o un conjunto de casos: (1) Siguiendo los índices directos desde las características del problema, (2) buscando una estructura de índices o (3) buscando un modelo del conocimiento general del dominio.

Un dominio dependiente pero con un sistema de medición global de similitudes es usado para valorar similitudes de un ajuste superficial.
Los sistemas basados en memoria dinámica usan el segundo enfoque pero también emplean el conocimiento general del dominio en la búsqueda.

PROTOS utiliza las estrategias 1 y 3, los punteros directos son usados para proporcionar un conjunto de candidatos los cuales son considerados emparejamientos posibles por el uso de conocimiento general.

Los casos pueden ser únicamente recuperados desde las características de entrada o también desde las características inferidas de la entrada.
Los casos que emparejan con todas las características de entrada son buenos candidatos *dependientes de la estrategia*, pero los casos que emparejan con una parte de las características del problema (de entrada o inferidas) pueden ser también recuperados.

A menudo se ejecutan pruebas para comprobar la relevancia de un caso recuperado, sobre todo si el caso recuperado sólo emparejaba con un subconjunto de características. Por ejemplo, una prueba simple puede ser comprobar si la solución recuperada es conforme con el tipo de la solución esperada del nuevo problema.

Se necesita un método para valorar el grado de similitud como por ejemplo métricas de similitud basadas en las similitudes superficiales del problema y las características del caso.

La valoración de similitudes puede ser más orientada al conocimiento-intensivo, intentando entender el problema en profundidad y usando objetivos, restricciones, etc., para guiar el emparejamiento.

Otra opción es asignar un peso a los descriptores del problema de acuerdo con su importancia par la caracterización del problema, durante la fase de aprendizaje.

En PROTOS, cada característica de un caso almacenado tiene asignado un grado de importancia para la solución del caso.

4.2.2.3 Selección

Del conjunto de casos similares, se selecciona el que mejor empareja. La selección podría hacerse durante el proceso de emparejamiento inicial pero es más habitual hacerlo posteriormente a partir de un conjunto de posibles candidatos. El mejor emparejamiento suele determinarse evaluando cual es el emparejamiento inicial más próximo. Para ello, se generan explicaciones, para justificar las características que no son completamente idénticas, basándose en el conocimiento de la red semántica.
Si el caso seleccionado no se ajusta lo suficiente, se hará otro intento para conseguir un mejor emparejamiento, siguiendo los enlaces de diferencia a los casos relacionados más cercanos.
Esta subtarea es más elaborada que la de recuperación aunque no en todos los sistemas se distinguen.

El proceso de selección genera consecuencias y expectativas para cada caso recuperado e intenta evaluar las consecuencias y justificar las expectativas. Esto lo hace usando modelo de conocimiento general del dominio del propio sistema o preguntando al usuario para confirmarlo u obtener información adicional. Normalmente los casos son clasificados de acuerdo a alguna métrica o a criterios de clasificación.

Los métodos de selección de conocimiento-intensivo generan explicaciones que soportan este proceso de clasificación y el caso cuya explicación es lo más próxima al nuevo problema es seleccionado.

Otras propiedades de un caso que son consideradas en algunos sistemas CBR son:

- importancia relativa y propiedades discriminatorias de sus características.
- El grado en el que un caso es un prototipo dentro de su clase
- Enlaces de diferencia a los casos relacionados.

5. Reutilización de casos: métodos de adaptación

La reutilización de casos es el proceso de transformar una solución recuperada en una solución apropiada para el problema actual. Se ha llegado a argumentar que la reutilización es el paso más importante del razonamiento basado en casos, ya que incorpora inteligencia a lo que, de no ser así, sería un mero proceso de reconocimiento de patrones.

A este respecto hay que considerar:

- Se habla de *reutilización transformacional* cuando lo que se reutiliza es la solución del caso previo recuperado. Debemos tener en cuenta que, si bien es lo más habitual, no siempre es necesario modificar la solución recuperada, sino que podría usarse como solución al problema actual sin ninguna modificación.
- Se habla de *reutilización derivacional* cuando lo que se reutiliza es el método que construyó la solución del caso previo recuperado. De la misma manera, no siempre es necesario modificar esa secuencia de pasos que se siguió para obtener la solución recuperada, sino que podría usarse sin ninguna modificación.
- Cuando se recupera más de un caso, se puede obtener una solución a partir de los múltiples casos, o se puede optar por presentar varias soluciones.

A la hora de elegir entre seguir una u otra estrategia para la reutilización, se debe tener en cuenta:

- De media, ¿qué grado de similitud va a tener el caso recuperado con el caso actual?
- Generalmente, ¿en cuántas características diferirán los casos entre sí?
- ¿Existen reglas conocidas que puedan utilizarse para llevar a cabo la reutilización de un modo más eficiente?

5.1 Métodos de Sustitución

Los métodos de sustitución dan valores apropiados para la nueva situación dependiendo de los valores de la solución previa. Se puede sustituir sólo un componente de la solución, varios de ellos, o todos los componentes de una solución.

5.1.1 Reinstanciación

La reinstanciación se usa cuando la similitud entre el caso actual y el recuperado (ó recuperados) es obvia y no hay otros tipos de restricciones o requerimientos impuestos para la solución. En otras palabras, el framework actual y el recuperado son netamente similares aunque ciertos roles tengas valores diferentes.
El proceso a seguir sería tomar la solución recuperada y actualizar los roles que difieran por los nuevos.

Ya hemos visto un ejemplo de reinstanciación en CHEF. Si tenemos almacenada una receta de pollo al whisky y deseamos una receta de pavo al brandy, el sistema puede ser capaz de recuperar la receta almacenada y darse cuenta que se debe actuar de igual manera, salvo que donde antes era pollo ahora es pavo y donde antes era whisky ahora brandy. Los roles "comida" y "licor" son los mismos en los dos casos, pero tienen valores diferentes.

Con este pequeño ejemplo podemos darnos cuenta que muy posiblemente la mayor complejidad está en que el sistema pueda llegar a inferir la gran similitud de los casos. Por ello, un razonador que pretenda poder emplear el método de la reinstanciación debe conocer:

- Correspondencias entre roles para el caso previo y el nuevo caso
- Un *framework* abstracto para la solución previa

Con ello, cuando encuentre que los roles del nuevo caso son iguales pero con valores distintos al caso recuperado, sustituirá los valores en 3 pasos:

1. Abstrae el *framework* del problema y solución previos.
2. Computa correspondencias entre roles del problema nuevo y el previo
3. Instancia el *framework* del problema y solución previos basándose en las correspondencias del paso anterior.

En este punto, es de mencionar que si bien es normal que existan diferencias entre autores en cuanto a número o forma de llevar a cabo métodos, también se ha constatado una diferencia entre autores con respecto a la propia clasificación de métodos. Según algunos, por ejemplo [Aamodt & Plaza], el método de Reinstanciación en realidad debería denominarse de "*Copia*" y debe ser distinguido del resto, que sí son considerados de Adaptación. Otros autores, [Sankar & Simon], sí consideran la reinstanciación un método de adaptación pero no lo consideran un método de sustitución.
Aquí seguiremos la visión de [Kolodner], que además de considerarlo de adaptación, lo considera también de sustitución.

5.1.2 Ajuste de Parámetros

Este método se emplea para interpolar los valores de una solución previa a una nueva solución. Es decir, dada una solución previa y un caso nuevo que difiere en algún grado del caso previo, la solución previa se modifica para la extensión en que difieren los dos casos.

Por ejemplo, en un posible sistema de justicia, un nuevo crimen semejante pero más cruel que uno previo ya almacenado, requiere una sentencia más dura.

Para realizar los cambios en los parámetros de la solución previa, se debe tener en cuenta qué grado de diferencia hay con el nuevo caso y en qué consisten estas diferencias. Esto se lleva a cabo en dos fases:

1. Se comparan los descriptores del problema previo y el nuevo y se extraen las diferencias.
2. Se aplican heurísticas de ajuste especializadas a la solución previa para crear la nueva solución. Las mencionadas heurísticas capturarán las relaciones entre características del problema y parámetros de la solución.

Es de mencionar que el ajuste de parámetros es una interpolación, es decir, no se debe confundir con otros métodos que sí crean verdaderas nuevas soluciones. Este método sólo construye una nueva solución a partir de una existente ajustando sus parámetros.

5.1.3 Búsqueda Local

Así como la reinstanciación sustituía un conjunto completo de roles, hay situaciones en las que sólo es necesaria una sustitución de una pequeña parte de una solución previa, para que se adapte perfectamente al nuevo caso. Un ejemplo sencillo sería, en el mencionado sistema de cocina, sustituir lasaña por lasaña vegetariana.

Se denomina búsqueda Local al proceso de buscar en una jerarquía abstracta (en los entornos de un concepto), algo relativamente similar que pueda ser sustituido.

Se suele usar cuando una solución previa es casi correcta para el caso nuevo, y puede ajustarse con alguna sustitución menor. Así, para recorrer esa jerarquía mencionada, este método busca primero en los hermanos del elemento previo para ver si alguno sirve como sustitución. Si no tiene éxito busca en los primos, y así sucesivamente.

La búsqueda local debe restringirse puesto que de otro modo sería ineficiente. Además se suele incorporar ciertas guías de movimiento hacia arriba y hacia abajo en la jerarquía.

Para entender el método veamos este ejemplo: imaginemos un sistema de guiado en rutas (al estilo PLEXUS). Le preguntamos al sistema como coger el metro en la ciudad de Madrid. Este sistema recuerda que tiene almacenado cómo coger el metro en la ciudad de Barcelona, e intenta aplicarlo a Madrid. El primer paso sería comprar un ticket, y para ello indica que se debe comprar el ticket en una ventanilla de la estación. Sin embargo, en Madrid no hay ventanillas (supongamos) porque las han sustituido por cajeros automáticos. El sistema buscaría entre múltiples hermanos: comprar tickets por internet, comprar tickets vía telefónica… hasta que encuentre una opción que pueda aplicar, como comprar tickets en cajero automático. Quizá los subpasos que deba aplicar ahora sean distintos, pero al sustituir la diferencia se aplicarán.

Una variante de este método es la *Memoria de Consulta*, útil cuando un simple método de búsqueda local no es capaz de predecir adecuadamente las relaciones entre conceptos, lo que conlleva una búsqueda ineficiente e incluso sin garantías de éxito.

Para estos casos, la variante de la memoria de consulta construye una descripción parcial de un elemento que sería una buena sustitución e intenta buscarlo. El método particular de búsqueda dependerá de la organización de la memoria (si por ejemplo fuera indexada, se utilizarían los índices como guía).

Otra variante es la *Búsqueda Especializada*, que va un paso más allá de la memoria de consulta: le da a la memoria instrucciones de cómo encontrar el elemento que se necesita. Evidentemente, este método funciona con heurísticas, que deberán ser descritas adecuadamente e incluso pueden agruparse por ciertos criterios.

Un ejemplo de heurística sencilla sería, en el caso del sistema de cocina, para encontrar qué utensilios de cocina se puede haber usado para cocinar un alimento para una cierta cultura (mediterránea, oriental, etc.), búsquese en qué recetas puede aparecer el alimento, de entre estas, se seleccionan las recetas de la cultura deseada y para estas últimas qué utensilios se usan.

También existe otra variante más, denominada trivialmente *Sustitución Basada en Casos***.** Esta variante simplemente intenta recuperar otro caso que se ajuste lo más posible a esa pequeña parte en que se difiere. Digamos que es una visión recursiva de la sustitución.

5.2 Métodos de Transformación

Los métodos de sustitución son apropiados cuando ya existe una base de casos aproximadamente sólida. En otro caso, los métodos de transformación son una alternativa muy viable, pues generan (derivan) una nueva solución basándose en las restricciones y características de la solución requerida. Un caso típico en el que se usan se observa cuando se intenta buscar una sustitución pero no se consigue ninguna que satisfaga los requerimientos exactamente. Entonces se utilizará parte o toda la solución recuperada más similar para generar una nueva solución.

5.2.1 Transformación de Sentido Común

Este método se aplica cuando alguno de los elementos de la solución previa no cumplen las restricciones de la nueva situación (y por supuesto no existe sustitución posible). La acción que se lleva a cabo es transformar la solución usando *reglas de sentido común*, reglas que utilizan el conocimiento acerca de la importancia relativa y funciones de diferentes componentes.

Básicamente, las reglas de sentido común no son más que un conjunto reducido de heurísticas que, empleando dicho conocimiento mencionado, determinan qué transformaciones hay que aplicar a la solución para que funcione adecuadamente en la nueva situación, como por ejemplo suprimir cierto componente de una solución o sustituir otros.

Debemos considerar ciertas premisas:

1. Los sistemas deben de ser capaces de identificar el componente que se debe *reparar* de un elemento candidato a ser transformado.
2. Las distintas representaciones deben separan componentes primarios de secundarios
3. Así mismo, deben mantener relaciones interna entre los elementos de los problemas que se están resolviendo

Volviendo al sistema de cocina, si se solicita una receta de pizza cuatro quesos y recuperamos una receta de pizza de chorizo con queso mozzarella, podemos ver que el candidato a ser sustituido es el chorizo por los otros 3 quesos, pero de esta manera la receta recuperada perdería todo su significado si quitamos el componente chorizo, ya que parece ser que éste es el componente primario de la receta original.
Es papel de las heurísticas definir qué puede ser sustituido.

Algún ejemplo de heurísticas (reglas de sentido común):

- Borrar componente secundario: Puede borrarse un componente secundario que no juegue un papel necesario.
- Sustituir un componente: Reemplazar cualquier componente por otro que puede llevar a cabo las mismas funciones.
- Añadir componente: Añadir componentes necesarios, asegurando que sus efectos no interfieren en las funciones necesarias de otros componentes.
- Ajustar la cantidad de un componente: Hacer hasta que se consiga lo requerido.

5.2.2 Reparación Guiada por Modelo

Mientras que la Transformación de Sentido Común confía en el conocimiento extraído del funcionamiento regular del mundo (por ello se denomina de Sentido Común), la Reparación Guiada por Modelo es un método de transformación que confía en el conocimiento de conexiones causales de algún tipo de sistema o situación.

Un ejemplo muy común se da en ciertos tipos de enfermedades; debido que una elevada presión sanguínea y ciertas arritmias se conoce que son resultado de tener arterias taponadas, la “elevada presión sanguínea” puede ser reemplazada por “arritmia” en una solución previa en la que se trataba con “arritmia” como resultado de “arterias taponadas”.

La reparación guiada por modelo es una colección de heurísticas que acceden a modelos causales para transformar una solución previa de tal modo que se ajuste a la nueva situación.

Para el ejemplo anterior, la heurística utilizada se denomina sustituir-evidencia: si dos estados proveen la misma evidencia para un tercer estado, se pueden sustituir entre sí. Y el modelo causal empleado es el que explica el comportamiento cardiaco.

Como en el método de ajuste de parámetros, este método debe evaluar las diferencias entre la descripción del problema previo y el nuevo. Se trataría de un proceso es 3 pasos:

1. Comparar el problema nuevo y viejo y extraer las diferencias
2. Evaluar las diferencias usando el modelo causal disponible, caracterizando las diferencias.
3. Para cada diferencia, aplicar a la solución previa la heurística de reparación guiada por modelo apropiada, creando una nueva solución.

Aunque parezca similar, podemos observar dos diferencias importantes entre el método de ajuste de parámetros y la reparación guiada por modelo:

1. El ajuste de parámetros sólo puede ajustar valores y cantidades de valores. Sin embargo, la reparación guiada por modelo puede transformar la estructura de una solución previa o reparar algún componente.
2. Las heurísticas del ajuste de parámetros están muy especializadas en ciertos tipos de parámetros en ciertos tipos de dominios. Aunque parezcan modelar qué cosas funcionan, no se apoyan en un modelo sólido. Las heurísticas de reparación basada en modelo pretenden ser de propósito general, y están basadas en nuestro conocimiento causal.

5.3 Heurísticas de Adaptación y Reparación de propósito particular

Los métodos presentados hasta este momento son básicamente de propósito general., e independientes del dominio. Podemos denominarlos *métodos débiles* de adaptación. Sin embargo, la experiencia en inteligencia artificial nos enseña que, allí donde estos métodos se utilizan, la aplicación de conocimiento específico del dominio, suele producir mejores resultados con menos esfuerzo.

Es decir, las heurísticas de adaptación de propósito particular, cuando existen, proveen excelentes guías para llevar a cabo la adaptación. Estas heurísticas guían tres tipos distintos de adaptación:

- Adaptación de dominio específico: Referente a reglas de sustitución y transformación específicas para algunos dominios de problemas.
- Modificación estructural: En este caso se modifica una solución insertando un nuevo elemento en la estructura. A menudo se combina con modificaciones en la estructura de propósito general.
- Reparación de propósito general: Recordemos que la reparación es una adaptación que se lleva a cabo cuando la evaluación de una solución ha encontrado algún fallo. La información que se acompaña en el informe del fallo, puede considerarse de propósito específico, y desde luego es una buena guía para saber qué debe cambiarse en la solución para que funcione.

Quizá lo más difícil sea asegurar un criterio de aplicabilidad de estas heurísticas. Una posibilidad es indexar las heurísticas por las características que se puedan dar con mayor probabilidad. Esto es, en algún paso del proceso se accederá a través de sus índices, y si se cumplen las condiciones de aplicabilidad de una heurística, ésta se lleva a cabo.

Un ejemplo podría ser, en el sistema de cocina, asegurar un pre-proceso de algunos alimentos. Esto es, creamos heurísticas que nos digan qué condiciones se deben dar para cocinar un plato. Podemos decir por ejemplo, que en las recetas en que se cocine pato, este debe estar antes desplumado adecuadamente, y escribimos la heurística como: "cuando se cocine pato, desplumarlo antes de cocinarlo". Y antes de comenzara cocinar una receta que incluya pato, se comprobaría que se cumple la heurística y se lleva a cabo.

Otra posibilidad es agrupar las heurísticas por función. Por ejemplo, podemos agrupar las heurísticas que sirvan para ajuste temprano de parámetros, otras para anticipación a problemas potenciales, etc.

5.4 Repetición Derivacional

En los métodos anteriores, se usa una solución previa para fijar una solución al nuevo problema. Sin embargo, hay ciertas situaciones en las que es más apropiado re-computar una nueva solución o una parte de una solución usando los mismos medios que se usaron para computar la solución previa. Denominamos a este método repetición derivacional (ó reutilización derivacional, según autores)

Ejemplos típicos que se dan en la propia vida real son muchas de las lecciones en el colegio, instituto e incluso universidad. Primeramente el profesor nos enseña la teoría y hace algunos ejercicios. Para comprobar que los alumnos lo han entendido, plantea otra serie de cuestiones muy parecidas a los ejercicios vistos en clases. Estos, obviamente se pudieran resolver directamente con la teoría, pero lo más cómodo es tomar un ejercicio ya realizado y aplicar a la nueva cuestión los mimos pasos que se utilizaron para resolverlo.

Para poder aplicar este método, los casos almacenados deben mantener información a cerca del método usado para resolver el problema, incluyendo una justificación de los operadores usados, submetas consideradas, alternativas generadas, caminos de búsqueda fallidos, etc. Cuando se recupera un caso, se reinstancia adecuadamente para la nueva situación y se repite, para el nuevo contexto, el plan que se llevó a cabo.

La experiencia no dice que este método se suele emplear para computar ciertas partes de la solución, mientras que para el resto se utiliza alguno de los métodos anteriormente descritos. En concreto, ciertos valores intermedios de la solución se computan de esta manera (los que se consideren más indicados para aplicar este método).

5.5 Otros métodos

Como se ha indicado, puesto que hay casi tantas clasificaciones de métodos como autores, se ha seguido la línea marcada por [Kolodner]. Otro punto de vista interesante y novedoso, lo aporta [Mitra & Basak], que realiza tres tipos de clasificaciones dependiendo del requerimiento de conocimiento del dominio para formular reglas de adaptación, la capacidad de adaptación de las reglas y el tipo de requerimiento de conocimiento del dominio. Quizá lo más interesante es que introduce algún método que difiere de los anteriores y que mencionaremos brevemente.

- Adaptación usando algoritmos genéticos: En un algoritmo genético, la base de casos forma la población inicial de los genotipos. Primero, el algoritmo recupera casos con un emparejamiento parcial de la base de casos con la ayuda de ciertos requerimientos de diseño especificados. A continuación, los casos recuperados se *mapean* en una representación de genotipo para que se apliquen ciertos operadores de mutación. Finalmente, los nuevos genotipos generados se mapean en los correspondientes fenotipos/casos infiriendo valores para los atributos y añadiendo el contexto del nuevo diseño.
  La colección de atributos que se emplean para describir un caso viene a ser la colección de genes que forman un genotipo.

- Adaptación con redes Bayesianas: En este método, el conocimiento del dominio del problema se codifica usando *m* atributos $A_1, \ldots, A_m$, donde un atributo A_i tiene n_i valores posibles, $a_{i1}, \ldots, a_{in}$. El mundo no es más que una serie de vectores binarios donde la función característica $f_i(a_{ij})$ es 1 si A_i tiene un valor a_{ij}, 0 en otro caso. Un caso se codifica como un vector $(P_k(a_{11}),\ldots P_k(a_{1ni}),\ldots,P_k(a_{m}n_{m}))$, donde $P_k(a_{ij})$ la probabilidad de que el atributo A_i tenga el valor a_{ij} en la clase k, es decir, $P_k(a_{ij}) = P(A_i = a_{ij} \mid C = c_k)$ ya que cada caso se identifica unívocamente por un c_k. La entrada de la red Bayesiana es la definición de las distribuciones de probabilidades iniciales para los valores de cada atributo, es decir $c_0 = P_0(a_{11}), \ldots, P_0(a_{1ni}),\ldots., P_0(a_{m1}),\ldots P_0(a_{m}n_{m})$. Las componentes de vector de salida son probabilidades de la forma $P(A_i = a_{ij} \mid C_0 = c_0)$.

6. Herramientas para el Razonamiento Basado en Casos

Algunos científicos cognitivos argumentan que el auge de este paradigma se debe a su naturaleza intuitiva puesto que, como ya se ha mencionado, el Razonamiento Basado en Casos linda con el razonamiento humano.
Al mismo tiempo, los vendedores de software pueden argumentar que el auge se debe precisamente a que, en no demasiado tiempo pero eficientemente, se han desarrollado herramientas CBR que llevan la teoría a la práctica. A continuación se proporciona un breve resumen de las herramientas más destacadas.

6.1 ART* Enterprise

ART* Enterprise es un producto de la empresa Brightware, una antigua división de Inference Corporation, uno de los primeros desarrolladores de herramientas de inteligencia artificial. ART* Enterprise se comercializa como una aplicación de desarrollo orientada al objeto y ofrece una variedad de paradigmas de representación, incluyendo:

- Un lenguaje de programación procedural
- Herencia múltiple, polimorfismo y encapsulación de objetos.
- Reglas
- Casos

Incluye una interfaz gráfica de construcción, control de versiones, y posibilidad de incorporar datos de repositorios en la mayoría de formatos DBMS propietarios para desarrollar aplicaciones cliente servidor. Además, ART* Enterprise proporciona un

soporte multi-plataforma para la mayoría de sistemas operativos actuales. Así como para plataformas hardware (por ejemplo, se puede desarrollar en una plataforma aplicaciones para otras).

El componente CBR en ART* Enterprise es básicamente el mismo que en CBR3 (se ampliará en su consiguiente apartado), si bien se añade cierta funcionalidad de acceso directo a CBR que lo hace incluso más controlable (manejable) que CBR3. Incluso podemos crear representaciones de casos jerárquicas complejas y controlar estrechamente el indexado de casos y las estrategias de recuperación.

No obstante, también tiene sus inconvenientes, como por ejemplo que los atributos de los casos en general son pares atributo:valor planos y no se ofrece soporte para indexado inductivo.

6.2 Case-1

Case-1 es una herramienta CBR desarrollada por Astea Internacional, y toma muchas ideas de CBR3.

Los casos se representan como texto de forma libre que describe un problema, un conjunto de cuestiones predispuestas que pueden confirmar o rechazar un caso, y un conjunto de soluciones. Además, los casos pueden ser manipulados por cualquier persona, sin necesidad de experiencia en programación, y se almacenan en una base de datos relacional.

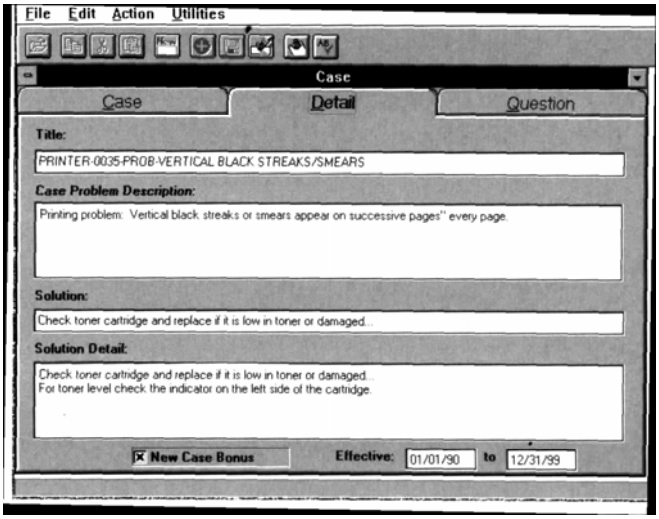

Figura 20. Interfaz de desarrollo de Case-1

Quizá no sea una herramienta tan madura como CBR3, ni mejora sustancialmente su funcionalidad, pero es una herramienta a seguir, sobre todo porque se puede integran muy bien con otros productos, como el PowerHelp de Astea.

6.3 CaseAdvisor

Comercializada por Sententia Software, de nuevo esta herramienta se basa en los productos CBR de la compañía Inference.

El componente CaseAdvisor Authoring es una herramienta muy sencilla, con un entorno amigable que nos permitirá introducir detalles como el nombre del caso, su descripción ó la solución. Junto con el componente CaseAdvisor Problem Resolution (el motor de recuperación de casos) y el CaseAdvisor WebServer (utilidad para Web) forman los tres componentes básicos del sistema.

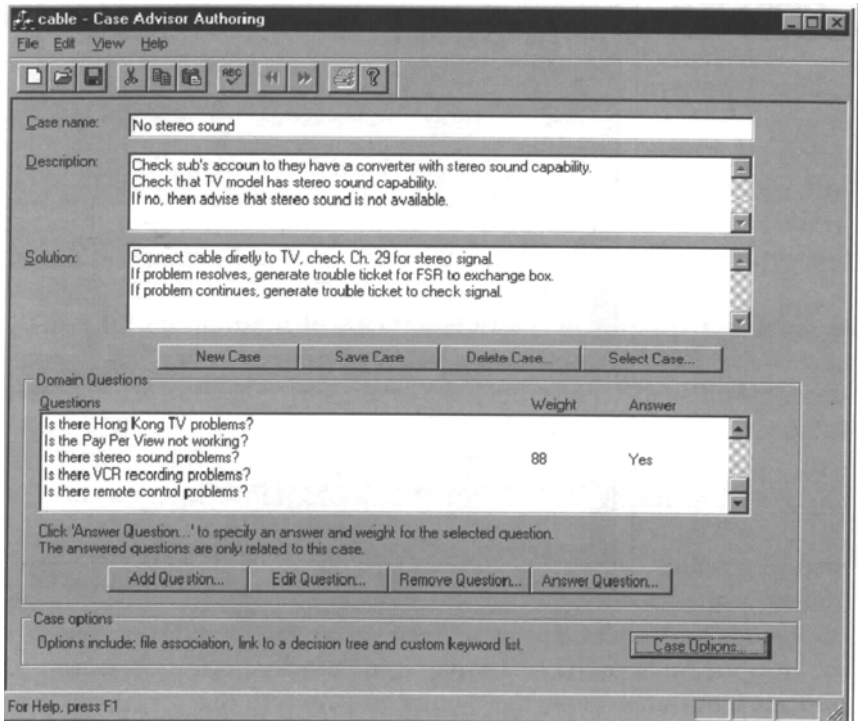

Figura 21. Interfaz de desarrollo de CaseAdvisor

Junto a los anteriores detalles, también se pueden proporcionar cuestiones para confirmar el caso, sin la posibilidad de poder utilizarse para rechazarlo como en CBR3. A las cuestiones se les asigna un peso de 0-100.

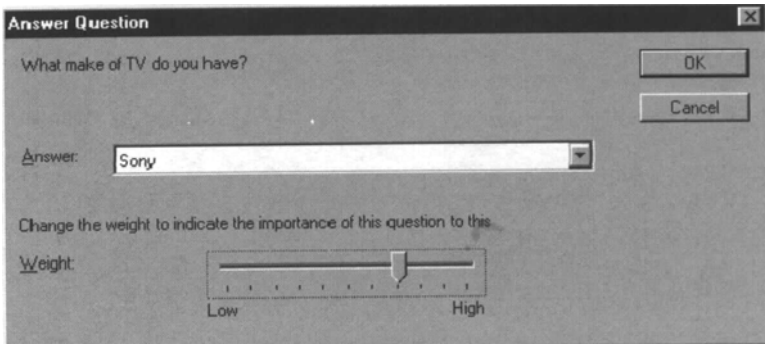

Figura 22. Sopesado de cuestiones de CaseAdvisor

Los Casos pueden tener añadidos como ficheros de texto o incluso multimedia que se abre con el navegador (Netscape)

Además, CaseAdvisor permite al usuario crear árboles de decisión de diagnóstico sencillos. Estos pueden utilizarse en conjunción con el sistema CBR en cualquier punto durante una diagnosis para guiar al usuario a través de la serie de pasos del diagnóstico. Los árboles de decisión se creamos usando una interfaz textual muy sencilla.

Precisamente, un inconveniente de este producto reside en que esta característica hubiese sido más aprovechable si se acompañase de una representación gráfica de los árboles de decisión.

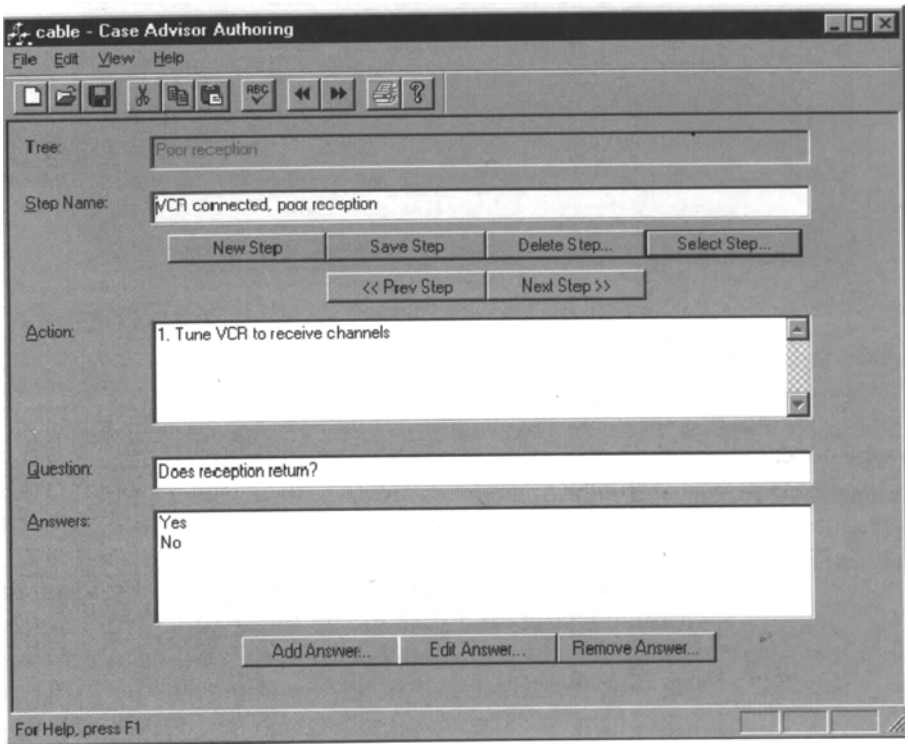

Figura 23. Editor del árbol de decisión de CaseAdvisor

Una vez más, la interfaz del sistema de recuperación está dividida en tres secciones, descripción del problema, cuestiones y solución. El usuario introduce una descripción al problema y CaseAdvisor recupera los casos que más se ajustan. Se muestran las preguntas para ayudar al usuario a que escoja un caso particular del conjunto de recuperados. Posteriormente el usuario deberá confirma la diagnosis con un nivel de puntuación

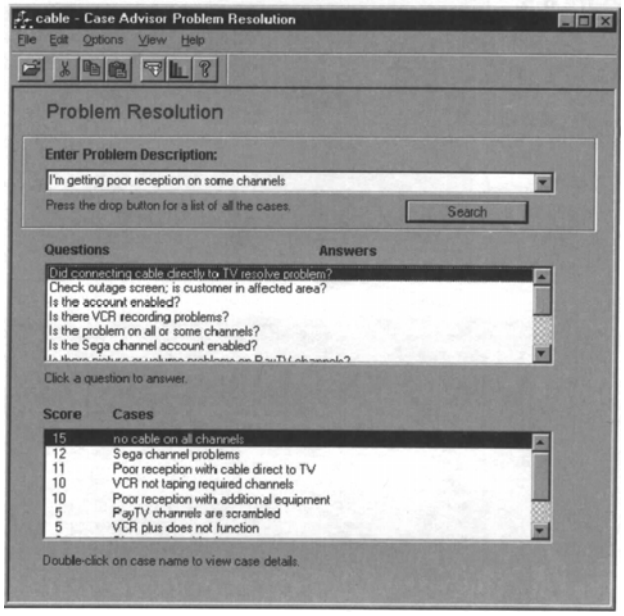

Figura 24. Interfaz de Resolución de Problemas de CaseAdvisor

6.4 CasePower

CasePower es una herramienta específica para construir hojas de estilo Microsoft Excel que puedan ser analizadas usando CBR. CasePower provee la funcionalidad básica de CBR y se enfoca principalmente a aplicaciones numéricas.

Los datos simbólicos se representan en jerarquías ordenadas que se traducen en rangos numéricos.

CasePower usa recuperación del vecino más próximo y reduce el tiempo de búsqueda calculando, fuera de línea, un índice para cada caso. Posteriormente, calcula el índice del nuevo problema y lo compara con los índices de la base de casos.

La adaptación se lleva a cabo usando formulas Excel y macros.

6.5 CBR3

Desarrollado por Inference Corp., los productos de la familia CBR3 son, para muchos, los más exitosos y mayormente trabajados.

La familia de herramientas CBR3 está formada por:

- CBR Express, entorno de desarrollo y *authoring* para bases de casos.
- CasPoint, motor de búsqueda en memoria eficiente para bases de casos desarrolladas usando CBR Express.
- Generator, herramienta que automatiza la creación de bases de casos a partir de una colección de ficheros de texto, como ASCII, Ms Word, etc.
- Tester, herramienta que provee una variedad de métricas para desarrolladores de base de casos que usan CBR3
- CasePoint WebServer, herramienta que soporta la funcionalidad de CasePoint en Internet.

CBR3 usa una estructura de registros simple que son almacenados en una base de datos relacional. Los casos incluyen un título, una descripción, un conjunto de cuestiones con un cierto peso, y un conjunto de acciones. Los casos pueden compartirse a través de la red de una organización.

La siguiente figura muestra un informe de caso de ejemplo de CBR3 que diagnostica fallos en punteros de láser.

```
BEGIN CASE CASE11
  TITLE
     Ink cartridge is damaged, causing black stains.
  DESCRIPTION
     Stains appear as small, round, black dots that
     occur on front or back of page.
     Sometimes wide inconsistent stains appear.
  QUESTIONS
     Are you having print quality problems?
       ANSWER  : Yes
       SCORING : (-)
     What does the print quality look like?...
       ANSWER  : Black Stains
       SCORING : (default)
     Does cleaning the printer with cleaning paper
     remove problem?
       ANSWER  : No
       SCORING : (default)
  ACTIONS
     Check toner cartridge and replace if it is low in
     toner or damaged...
  BROWSE TEXT
  CREATION    29/7/91 14:19:22
  LAST_UPDATE 29/7/91 14:19:22
  LAST_USED   29/7/91 14:19:22
  STATUS ACTIVE
END CASE
```

Figura 25. Caso CBR3 exportado como informe

Sin embargo, los desarrolladores no usan CBR3 programando, sino a través de la interfaz de CBR Express, cercana a usuarios sin experiencia en programación.

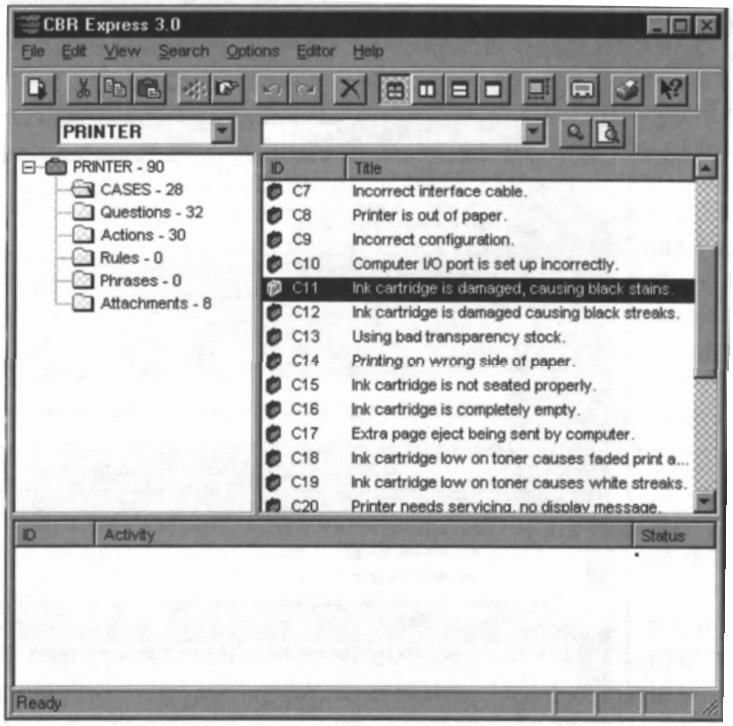

Figura 26. Interfaz de desarrollo de CBR Express

CBR3 usa recuperación del vecino más próximo, buscando un ajuste entre la consulta del usuario en texto libre y el título y descripción de los casos en la base de casos. De hecho, uno de los puntos fuertes de CBR3 es su habilidad para tratar texto en formato libre.

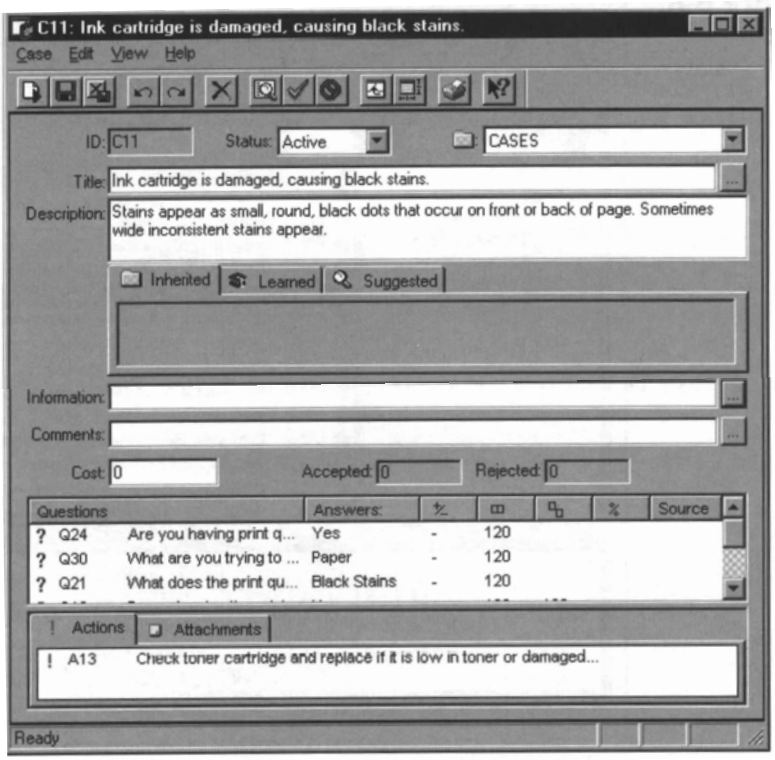

Figura 27. Editor de Casos de CBR Express

Los casos pueden heredar características de otros casos. Podemos decir un subconjunto de casos donde todos ellos cumplen ciertas propiedades.

Como se ha indicado, para la búsqueda en la base de casos se usa CasePoint, que examina una entrada de texto de usuario en formato libre y trata de ajustarlo con títulos y descripciones de casos almacenados, obteniendo un conjunto de casos posibles junto con su valor de exactitud y una serie de cuestiones. Las respuestas a estas cuestiones permiten reducir el número de casos y guiar una solución más precisa.

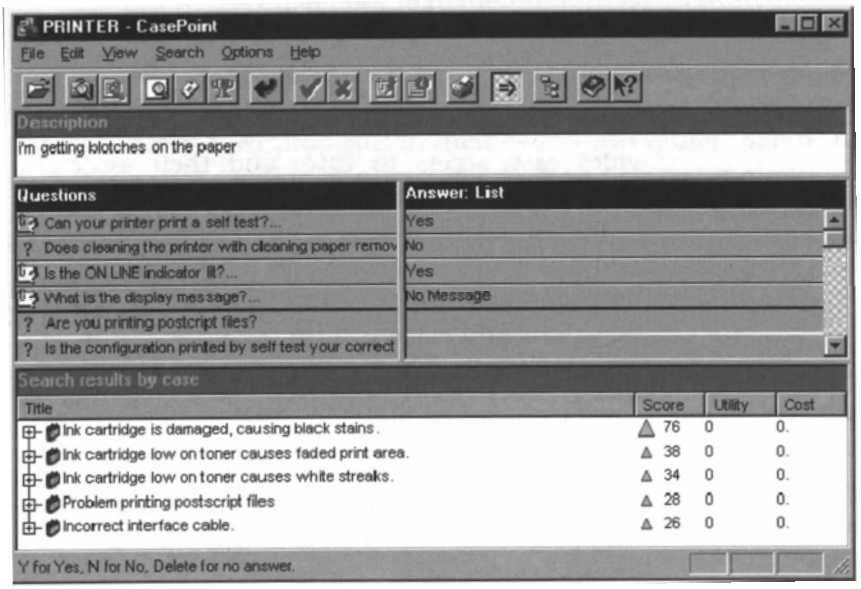

Figura 28. Interfaz CasePoint

Si no se halla una solución satisfactoria, se usa el concepto de *caso sin resolver***,** lo que significa que se guarda la trascripción de consulta a ficheros ó incluso se puede mandar un mensaje o e-mail al administrador del sistema para que verifique el resultado.

CasePoint también permite utilizar reglas que identifiquen palabras claves en el texto de consulta para discriminar más la búsqueda y presentar las soluciones ordenando por utilidad (cómo de útil se considera que puede ser la solución) o por coste (cómo de costosa se considera que puede ser la solución).

6.6 ReCall

ReCall es una herramienta de la compañía Isoft y está codificada en C++.

Esta herramienta ofrece una combinación de recuperación de vecino más cercano y recuperación por inducción, usando una representación orientada a objetos con mecanismos de herencia, descriptores tipados, *facets*, *demons* y relaciones entre objetos. Por tanto, los casos particulares de una base de casos no son más que instancias.

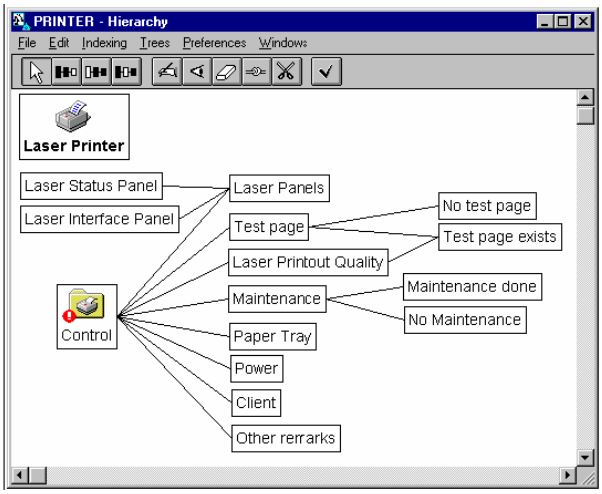

Figura 29. Jerarquía de Objetos en ReCall

Esto permite al usuario trabajar muy cómodamente, puesto que puede especificar conocimiento complejo estructurando módulos más sencillos, y dejar ciertas descripciones o características incompletas puesto que pueden derivarse del sistema de herencia. Para una mejor organización y para ganar eficiencia en la recuperación, se complementa el sistema con un sistema de índices de herencia múltiple, permitiendo la comparación de casos.

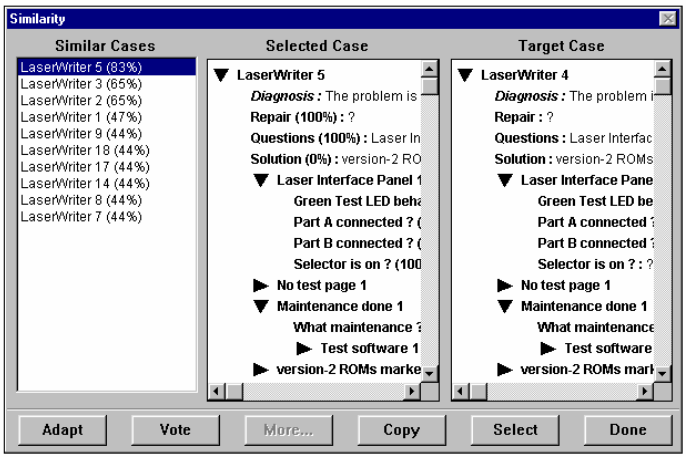

Figura 30. Comparando Casos en ReCall

ReCall provee diferentes métodos para analizar automáticamente la base de casos y devolver una selección de índices así como su organización. No obstante esta automatización, llevada a cabo con métodos inductivos, no es obligatoria y los desarrolladores experimentados pueden impone su propia organización.

ReCall soporta dos métodos diferentes de adaptación: el primero de ellos es el considerado por defecto y se basa en un principio de *voting* , y para el segundo se utilizan reglas de adaptación definidas por el usuario.

6.7 ReMind

ReMind es una herramienta que inicialmente fue desarrollada con el soporte del famoso programa DARPA de Estados Unidos por la empresa Cognitive Systems.

Los casos en ReMind se representan como pares *atributo:valor*.

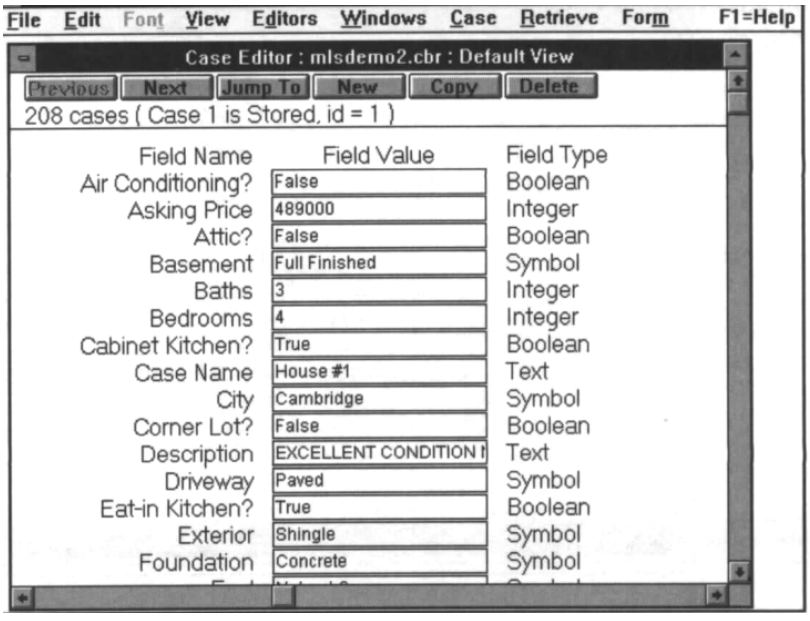

Figura 31. Interfaz de desarrollo ReMind

ReMind ofrece varias posibilidades de recuperación:

- por el vecino más cercano, con ayuda de la importancia (peso) definido por el usuario para las características de cada caso.
- Por patrones, soportando consultas SQL simples
- inductiva, bien automáticamente sin interacción del usuario o bien se trata de una recuperación inductiva guiada por conocimiento, esto es, el usuario crea un modelo cualitativo para guiar el algoritmo de inducción con un cierto trasfondo de conocimiento.
- Los modelos cualitativos (*Q-models*) se crean gráficamente para indicar qué conceptos (propiedades de los casos) dependen de otros. Al mismo tiempo, podemos dotar de importancia (peso) a estas dependencias.
  Cuando el algoritmo de inducción se guíe por estos modelos, los árboles de decisión serán un mejor reflejo de la relación causal de los conceptos en los casos.

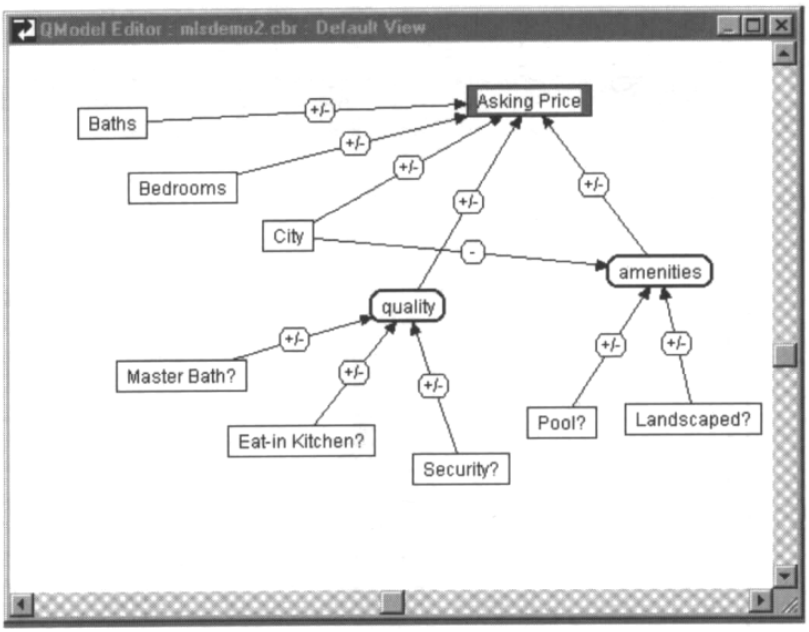

Figura 32. Un Q-model en ReMind

Otra de las ventajas de usar una recuperación inductiva es la posibilidad de poder explicar porqué se recupera un conjunto de casos y no otro.

Con respecto al proceso de adaptación, simplemente reseñar que se utilizan ciertas fórmulas de adaptación que ajustan los valores basándose en las diferencias entre el caso recuperado y el caso nuevo (básicamente utiliza un método de sustitución).

Además, es reseñable la posibilidad de importar bases de casos de Bases de Datos. No obstante, un punto negativo es tratamiento de texto en formato libre, no tan bueno como alguno de las aplicaciones ya mencionadas.

6.7.1 Ejemplo propio en ReMind: Venta de Móviles

Para probar muchos de los puntos que se han expuesto en los apartados anteriores, así como para ampliar la información disponible sobre ReMind y extrapolarla al resto de herramientas CBR, se ha creado un ejemplo propio usando ReMind. Se ha implementado un sistema de **Venta de Móviles**.

Nuestro sistema está dirigido a un supuesto vendedor de móviles que recibe del cliente cierta información de las características del móvil que desea. El vendedor introducirá en ReMind las características más reseñables y aplicará un proceso de recuperación para obtener los móviles más similares al deseo del cliente. Además, gracias a la adaptación, podrá crear un nuevo prototipo de móvil que podrá almacenar en la base de datos para, por ejemplo, hacerse una idea del móvil prototípico deseado por sus clientes y realizar nuevos pedidos.

Un caso almacenará información sobre un móvil, distribuida en distintos campos. Para ello, se han considerado las características de los móviles más destacables en la actualidad. La siguiente figura muestra la edición de un campo:

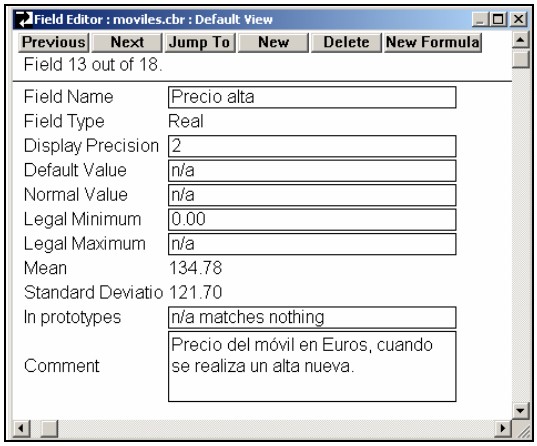

Figura 33. Edición de un campo

Cada campo tiene unos atributos comunes y otros que dependen del tipo del campo. Todos los campos deben tener:

- Un nombre, que lo identifica.
- Un tipo. Los tipos posibles son: Entero, Real, Texto, Símbolo (es como un enumerado), Caso (una referencia a otro caso), Fecha, Booleano ó Listas de los anteriores.
- un valor por defecto, que es el valor que aparecerá nada más editar un hipotético nuevo caso.
- Un valor normal, que es el valor que se toma como índice para el cluster cuando el caso no especifica un valor para este campo (n/a). Si no se indica, se toma un valor medio.
- Un Prototipo, que le indica a ReMind cómo tratar los campos no introducidos (n/a).
- Un comentario textual, que nos ayuda a comprender el uso del campo.

Además están los atributos que dependen del tipo del campo. Por ejemplo en la figura anterior, al ser de tipo Real, podemos definir la precisión (nº de decimales que se tratan), el mínimo y el máximo de los valores. La media y la desviación estándar son valores calculados por ReMind automáticamente.

Nuestro sistema tiene almacenados 37 casos que corresponden con 37 modelos de móviles. La siguiente figura nos muestra el editor de casos con un caso almacenado:

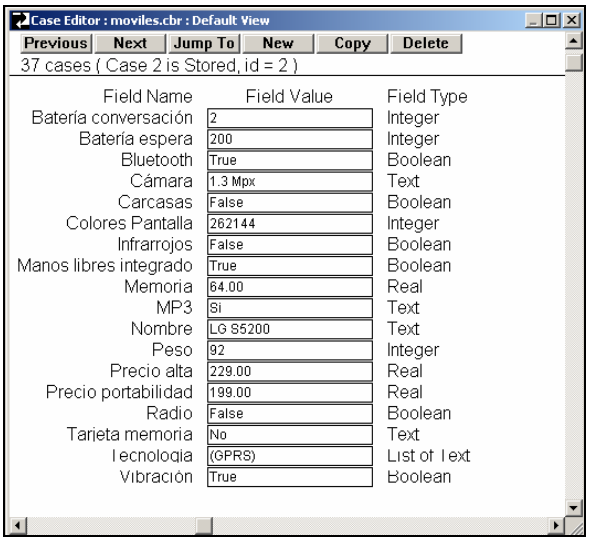

Figura 34. Editor de casos

Para poder emplear los distintos métodos de recuperación y adaptación hay que introducir cierta información. La siguiente figura muestra el editor de importancia.

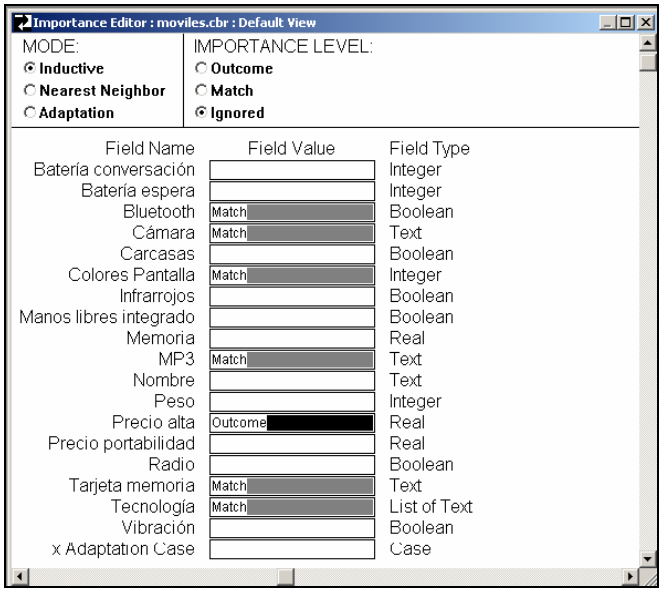

Figura 35. Editor de importancia para Inducción

El modo indica qué método deseamos parametrizar. En la figura anterior, se introduce qué queremos emparejar en el método inductivo (*Match*), qué queremos ignorar (*Ignored*) y qué campo es para qué parámetro se enfoca el *clustering* (*Outcome*). Se han seleccionado para emparejar las características más reseñables, cuyos valores

determinan en buena medida el precio del móvil (esto se indicará en el *Q-Model*). Para *Outcome*, como sólo permite uno, se ha optado por el Precio alta, muy de interés para el cliente.

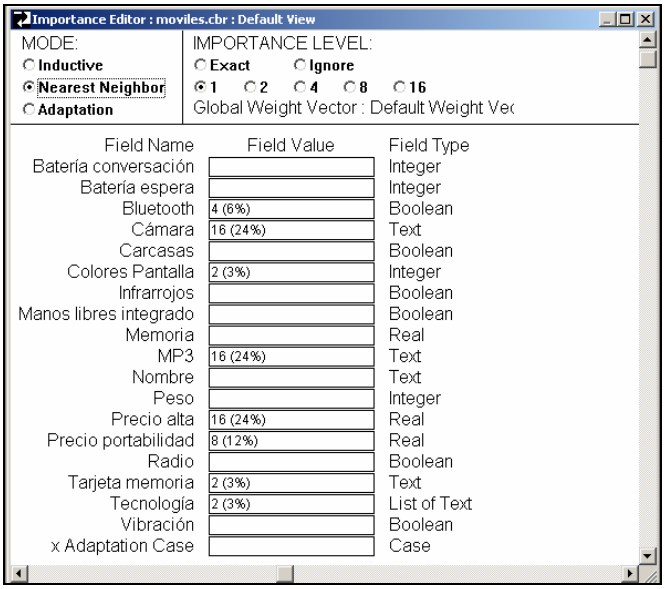

Figura 36. Editor de importancia para Vecino más Cercano

Para el caso de vecino más cercano, se especifica el vector de pesos de los campos, dotando de más importancia (de 1-16) a las características más reseñables, así como si se desea ignorar o que el ajuste sea exacto.

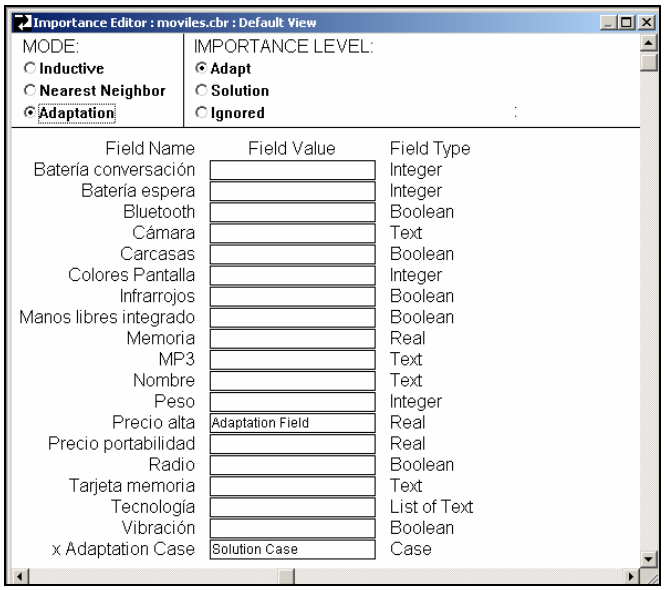

Figura 37. Editor de importancia para Adaptación

Para el caso de la adaptación, se especifica qué campo se debe adaptar, ignorar o sí es la solución (sólo para parámetros de tipo Case).

Con el editor de Cluster podemos generar automática o manualmente los clusters, es decir, un conjunto de índice que hace más eficiente la recuperación inductiva.

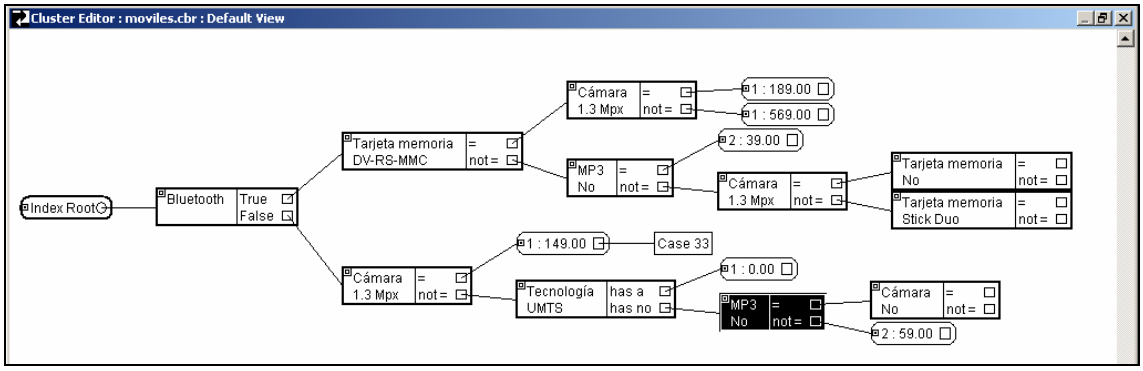

Figura 38. Editor de *Clusters*

También creamos un *Q-Model* que guíe la inducción.

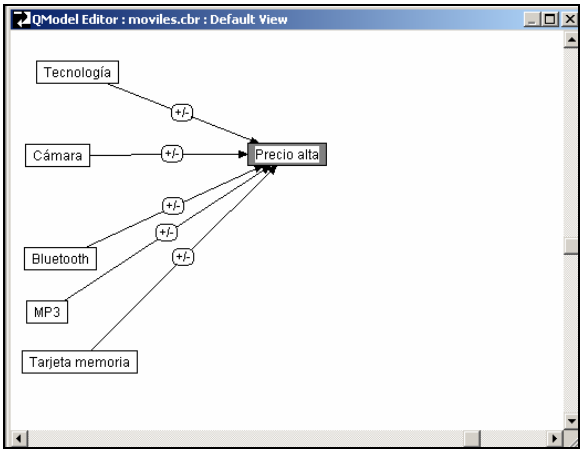

Figura 39. Editor de *Q-Model*

La figura anterior nos indica que ciertos campos, Tecnología, Cámara, BlueTooth, MP3 y Tarjeta Memoria, influyen positiva o negativamente en el Precio alta. En estos modelos también existe la posibilidad de indicar que ciertos campos solo influyen positiva o negativamente e incluso de añadir nodos virtuales.

A la hora de recuperar un caso por vecino más cercano seleccionamos el vector de pesos (normalmente el de por defecto) y aparecen los casos recuperados. Esto se muestra en la siguiente figura

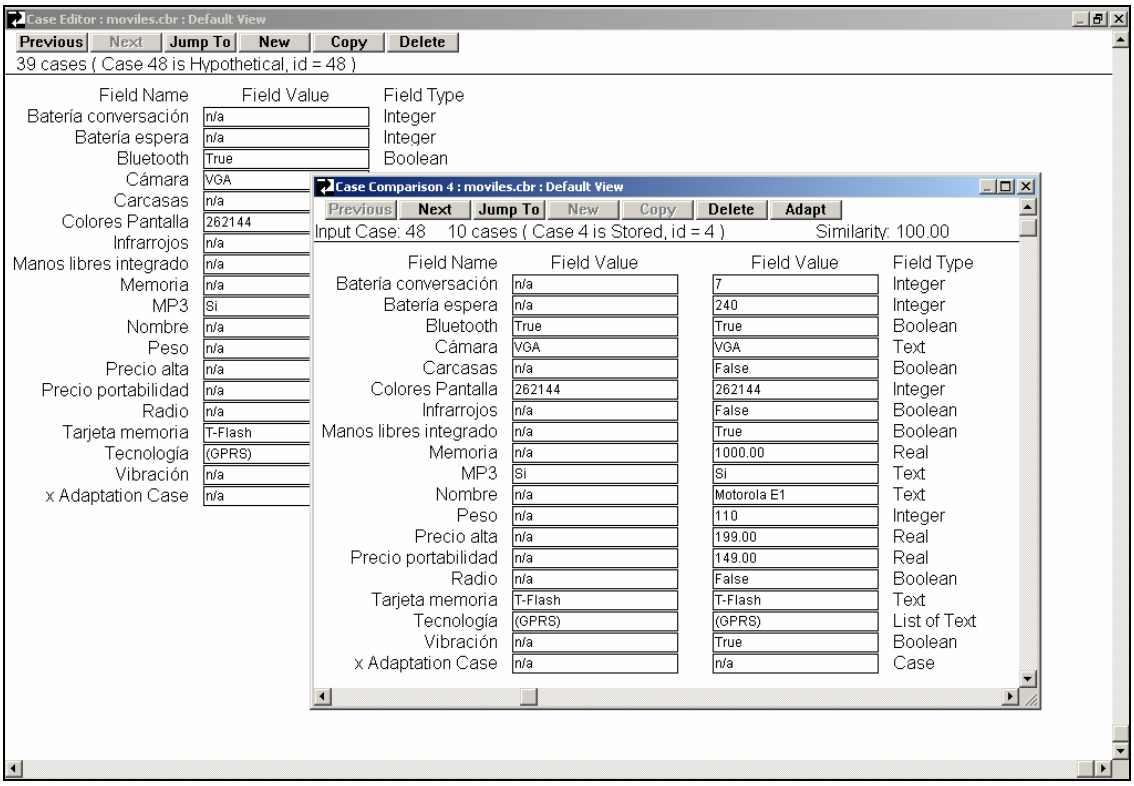

Figura 40. Recuperación Vecino más Cercano

El vendedor podrá adaptar el caso que desee de entre los recuperados, guardando en *x Adaptation Case* una referencia al caso del que se adaptó.

De igual manera, si seleccionamos recuperación por inducción, ReMind nos proporcionará los casos recuperados guiándose por los clusters y el Q-Model. Además de poder comparar y adaptar el caso que se desee, nos da información de los pasos seguidos.

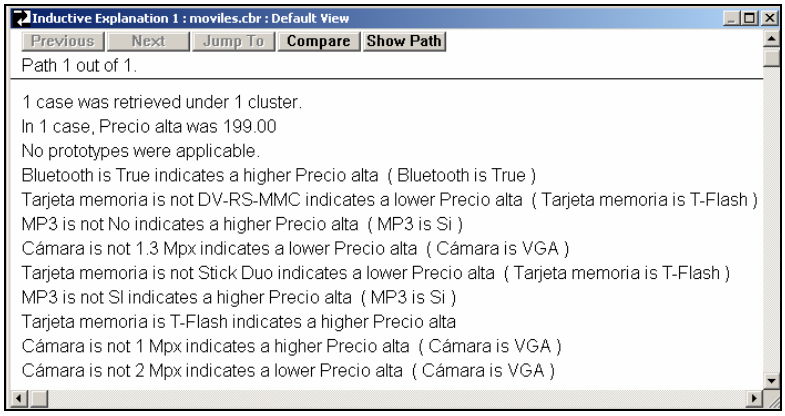

Figura 41. Explicación de camino en recuperación por inducción

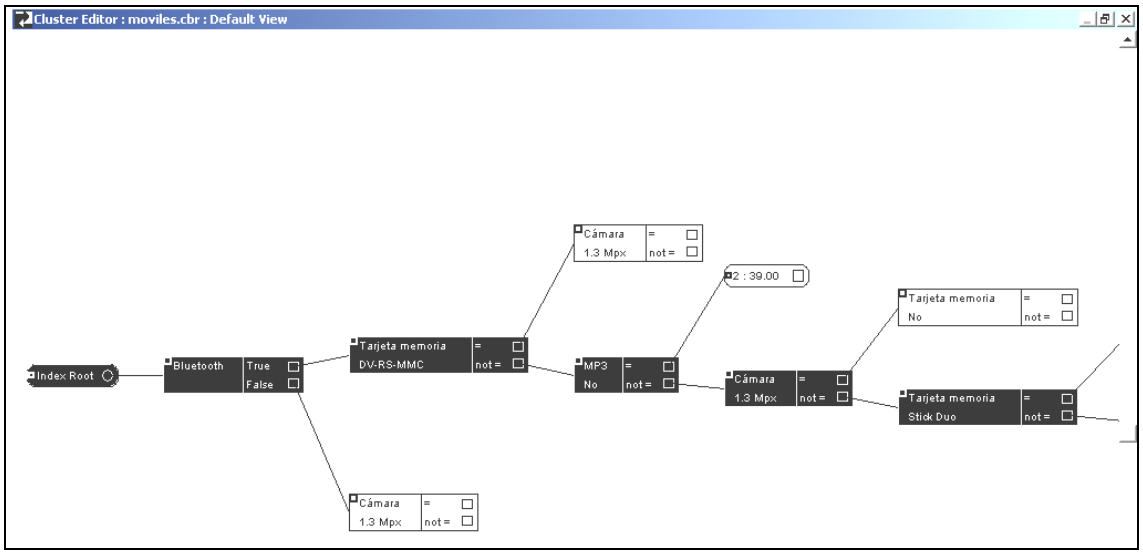

Figura 42. Gráfico de camino en recuperación por inducción

Otra posibilidad es crear un patrón y recuperar los casos que se ajusten a él. Por ejemplo, creamos el patrón *Amplia Memoria* que recuperará todos los casos cuyo campo Memoria sea > de 5 (indicando 5 MB).

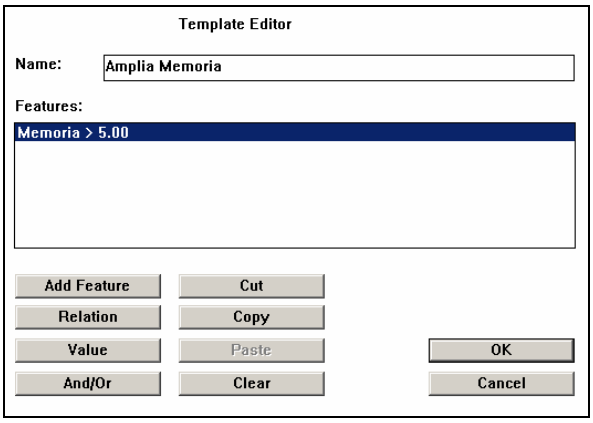

Figura 43. Editor de Patrones

7. Ventajas e inconvenientes

El razonamiento basado en casos provee numerosas ventajas:

1. Reduce la tarea de adquisición de conocimiento. Eliminando la necesidad de extraer de un modelo o de un conjunto de reglas, como en los sistemas basados en modelo/reglas, la tarea de adquisición de conocimiento por parte de un razonador basado en casos consiste, básicamente, en una colección relevante de experiencias/casos y su representación y almacenamiento.

2. Evita la repetición de errores del pasado. En los sistemas que guardan tanto los fallos como los éxitos, así como las causas de los fallos, se utiliza la información acerca de qué causó el fallo en el pasado para predecir posibles fallos futuros. El sistema puede incluso alertar al razonador para que tome las acciones necesarias para no repetir errores.

3. Provee gran flexibilidad en el modelado del conocimiento. En contraste con los sistemas basados en modelos, donde muchos problemas no se resuelven por considerarlos fuera de su ámbito ó no se entiende el problema a poco que los datos estén incompletos, los sistemas CBR usan las experiencias pasadas como dominio de conocimiento y dan una solución razonable, previa adaptación, de este tipo de problemas.

4. Permite al razonador proponer soluciones en dominios que no son del todo entendidos por el razonador. En situaciones donde existe poco conocimiento para construir un modelo causal del dominio ó para derivar un conjunto de heurísticas, un razonador basado en casos sí puede desarrollar un pequeño conjunto de casos que le permitan funcionar.

   Hay dominios que son imposibles de entender completamente, a menudo porque dependen del comportamiento humano impredecible, la economía por ejemplo. Otros, porque simplemente no se entienden todavía, por ejemplo cómo funcionan algunos medicamentos y enfermedades. Sin embargo, el razonamiento basado en casos nos permite tomar ciertas premisas y predicciones basándonos en lo que funcionó en el pasado.

5. Permite hacer predicciones del posible éxito de una solución propuesta. Cuando la información se almacena teniendo en cuenta el nivel de éxito de las soluciones previas, el razonador basado en casos puede ser capaz de predecir el éxito de una solución propuesta para el problema actual. Obviamente, el razonador tendrá en cuenta no sólo esos niveles de éxito almacenados sino las diferencias entre el caso ó casos recuperados y la situación actual.

6. Aprende con el tiempo. Esto que parece obvio, por ejemplo en la vida de un ser humano, no lo es en la mayoría de los sistemas de resolución de problemas. Sin embargo, como ya se ha citado, a medida que los razonadores basados en casos son más usados, encuentran más situaciones de problemas y crean más soluciones. Tras las evaluaciones, determinando un nivel de éxito en las soluciones, los casos se añaden al sistema y el sistema tendrá más variedad de situaciones y más grado de refinamiento y éxito.

7. Se propone soluciones a problemas rápidamente. En dominios que requieren un gran procesamiento para crear una solución de la nada (por ejemplo recorriendo grafo), el razonamiento basado en casos puede reducir ese tiempo considerablemente tomando una solución temprana y modificándola adecuadamente.

8. Provee un medio de justificación. Así como un médico puede justificar un tratamiento a un paciente en sus experiencias similares previas, el razonamiento basado en casos puede dar un caso previo y su solución (con éxito) para convencer al usuario, o justificar, una solución propuesta al problema actual. Si el usuario deseara una medida de calidad de la solución, el sistema podría cuantificar cuanto éxito tuvo el caso pasado y qué grado de similitud hay con el caso actual y el pasado.

9. Se puede utilizar para muchos propósitos, como crear planes, diagnosis, argumentación de puntos de vista, etc., de formas muy distintas, dependiendo básicamente de los métodos de recuperación y adaptación que implementen.

10. Es un reflejo del razonamiento humano, ya que nosotros usamos una forma de razonamiento basado en casos. Esto es una gran ventaja a la hora de poder entender el funcionamiento del sistema, así como la justificación de una solución propuesta por un sistema basado en casos.

Por supuesto, también hay una serie de desventajas:

1. Puede haber una tendencia a usar los casos previos ciegamente, confiando en la experiencia previa sin validarla en la nueva situación.
2. Los casos previos pueden predisponer demasiado al razonador a la hora de resolver un nuevo problema.
3. Las personas, especialmente los principiantes, a menudo no se acuerdan del conjunto de casos más apropiados cuando están razonando.

Confiar en experiencias previas sin validarlas puede generar soluciones y evaluaciones ineficientes o incorrectas. La recuperación de casos inapropiados puede costar un tiempo considerable o llevar a errores muy costosos, que podrían ser evitados por métodos más incrementales.

Ciertos sistemas basados en casos son los sistemas de ayuda o apoyo de toma de decisiones, que aumentan la memoria humana proveyendo los casos apropiados, al tiempo que siguen dejando al usuario humano un margen de razonamiento. Podemos hacer más seguros estos y otros sistemas evitando los comportamientos negativos descritos.

Referencias y Bibliografía

[Aamodt & Plaza] A. Aamodt, E. Plaza (1994); Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Communications. IOS Press, Vol. 7: 1, pp. 39-59.

[Sankar & Simon] Sankar K. Pal, Simon C. K. Shiu (2004); FOUNDATIONS OF SOFT CASE-BASED REASONING. Editorial JOHN WILEY & SONS, INC.

[Watson] Ian Watson1 (1997); Applying Case-Based Reasoning: Techniques for Enterprise Systems. Esitorial Morgan Kaufmann

[Kolodner] Kolodner, J. L. (1993). *Case-Based Reasoning*. Editorial Morgan Kaufmann.

[Mitra & Basak] Rudradeb Mitra, Jayanta Basak; Methods of Case Adaptation (2005): A Survey. INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 20, 627–645

y simples referencias a:

ReCall: http://www.alice-soft.com/html/prod_recall.htm
Angeles Manjarrés Riesco; Razonamiento Basado en Casos. UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA (ESPAÑA)