Un Sistema Inteligente para el Aprendizaje de Fundamentos de Programación Orientada a Objetos

Jaime Gálvez-Cordero, Félix Gómez-Cordero,
Eduardo Guzmán, Ricardo Conejo

Departamento de Lenguajes y Ciencias de la Computación.
Universidad de Málaga.
{jgalvez,guzman,conejo}@lcc.uma.es

Resumen En este trabajo se presenta una herramienta Web para aprender fundamentos de programación orientada a objetos. Se trata de un sistema inteligente cuyo modelo del alumno se ha construido siguiendo la técnica de Modelado Basado en Restricciones. Los alumnos aprenden mientras que resuelven problemas a través de una interfaz visual. En estos problemas los alumnos han de implementar una o más clases. El sistema elige dinámicamente el problema más adecuado al nivel de conocimiento de cada individuo, y en cada problema, se van mostrando ayudas y refuerzos adaptados también a su estado de conocimiento.

1. Introducción

Los sistemas inteligentes para el aprendizaje utilizan técnicas de Inteligencia Artificial en conjunción con teorías de aprendizaje, obtenidas a partir de estudios psicológicos e investigaciones realizadas en el campo de la educación, con el fin de mejorar el proceso de aprendizaje del alumno. Mediante estos sistemas se ofrece un entorno de instrucción que se adapta a las capacidades y necesidades de aprendizaje del estudiante, siendo incluso un método más efectivo que la instrucción tradicional profesor-alumno [2]. Aunque existen diversas teorías de aprendizaje, casi todas tienen en común tres fases [3]. En la primera, el aprendizaje conceptual, se adquiere el conocimiento teórico de forma declarativa [11]. La segunda fase consiste en practicar el conocimiento adquirido, transformándolo en procedimental [1]. La tercera fase ocurre de manera ocasional y tiene lugar al perfeccionar el conocimiento anterior mediante la deducción de nuevo conocimiento.

La motivación de este trabajo radica en la dificultad que tienen los alumnos que cursan la asignatura de Elementos de Programación del primer curso de las tres titulaciones técnicas de la E.T.S.I. de Telecomunicación de la Universidad de Málaga, para aprender los Fundamentos de la Programación Orientada a Objetos (FPOO), más concretamente la abstracción de datos. El primer contacto que tienen estos alumnos con la programación es haciendo uso del paradigma imperativo, y es en la asignatura de Elementos de Programación donde se les introduce por primera vez el concepto de programación orientada a objetos. El objetivo

principal de este trabajo es ayudar a paliar este problema proporcionando a los estudiantes una herramienta que permita practicar sus conocimientos, mediante ejercicios que se adapten a su nivel. La idea es asistir a los alumnos mientras éstos resuelven problemas. Según [6], existen evidencias psicológicas de que el *refuerzo* inmediato tras un error, es la acción pedagógica más efectiva; ya que para el alumno, es más fácil localizar y analizar el estado mental que le llevó a ese error e identificar carencias en su conocimiento, que esperar a recibir un refuerzo por parte del profesor. El refuerzo inmediato, a su vez, también reduce la frustración que puede sufrir el alumno debida a carencias en su conocimiento.

El artículo se organiza de la siguiente manera: las características generales del sistema se detallan en la siguiente sección. Seguidamente, se explican las partes principales de éste: Inicialmente la interfaz que ofrece tanto a alumnos como a profesores (sección 3), la forma de representar el conocimiento (sección 4), el modelado del alumno empleado (sección 5), y se describe el funcionamiento del módulo pedagógico (sección 6). En la sección 7 se describen algunos sistemas similares. Finalmente, se analizan las contribuciones y las líneas futuras de trabajo.

2. Características generales del sistema

Dentro de la arquitectura de este sistema se pueden distinguir las siguientes partes principales: Una **interfaz** Web que permite al alumno, de forma visual, resolver problemas. También se ha desarrollado una herramienta de autor a través de la cual los profesores pueden añadir problemas e incluir su solución. El **modelo del dominio** contiene la representación, realizada por los expertos en la materia, del conocimiento que se va enseñar. Esta parte es la encargada de la generación de problemas, así como de la evaluación de las soluciones realizadas por el estudiante. El **modelo del alumno** reúne los conocimientos del estudiante y se usa para adaptar el proceso de enseñanza a las capacidades de aprendizaje y al nivel estimado de cada alumno. Por último, el **módulo pedagógico** define las estrategias de enseñanza que establecen cómo mostrar la información al alumno durante la resolución de problemas, y de seleccionar el problema que debe ser presentado al alumno en cada momento.

El sistema tiene dos modos de funcionamiento permitiendo al estudiante, por un lado, resolver los ejercicios que le propone el sistema, los cuales son adaptados teniendo en cuenta los errores más frecuentes del alumno y la gravedad de éstos, y por otro lado, resolver sus propios ejercicios, para así poder detectar y corregir sus propios errores.

3. La interfaz

La herramienta creada posee una interfaz Web, de forma que los alumnos pueden aprovechar las facilidades que hoy en día proporciona Internet para utilizar este sistema desde cualquier lugar y en cualquier momento.

A través de ella, cuando el alumno aplica los FPOO para resolver un problema, puede insertar declaraciones y sentencias (mediante un mecanismo de selección y arrastre). Estos problemas han de ser escritos en el pseudolenguaje creado para la asignatura. Aunque las sentencias que se ofrecen son un subconjunto del lenguaje completo necesario para desarrollar cualquier programa, éstas son las precisas para realizar los ejercicios que se plantean a los estudiantes que comienzan a aprender FPOO.

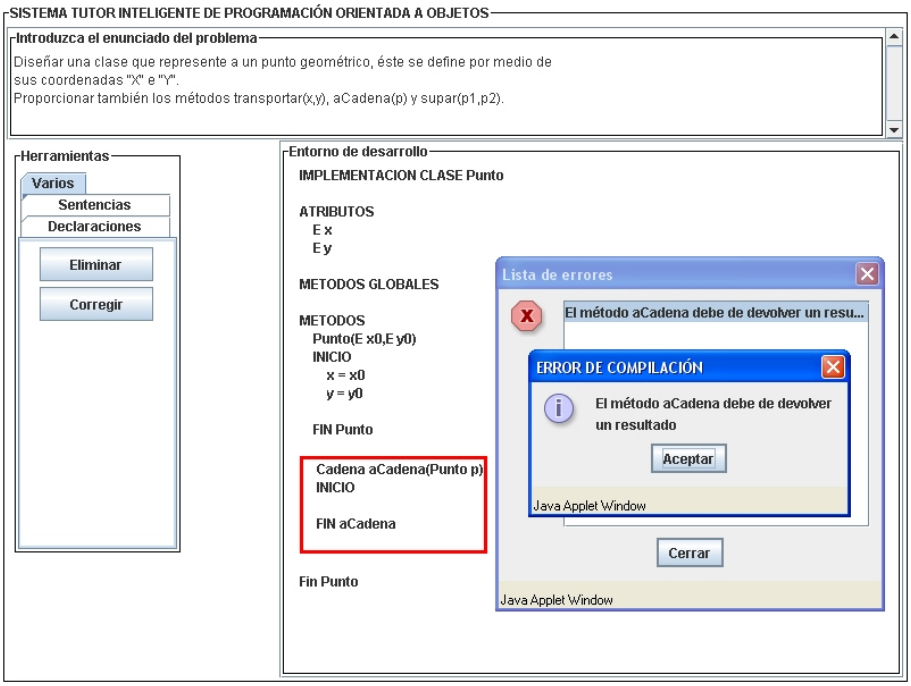

Figura 1. Aspecto de la interfaz del sistema.

La figura 1 muestra el aspecto de la interfaz del sistema. Como se puede apreciar, se trata de una interfaz visual, la cual se ha intentado que sea sencilla e intuitiva. Las partes en las que ésta se estructura se enumeran a continuación:

- El enunciado del problema, ubicado en la parte superior.
- La barra de herramientas (marco inferior izquierdo de la figura), que se compone de tres pestañas: i) Sentencias, que contiene un botón por cada una de las sentencias que ofrece el pseudolenguaje. ii) Declaraciones, con los botones que permiten crear clases, definir sus atributos o sus métodos, y definir variables. iii) Varios, en la que se incluyen un botón para eliminar código y otro para compilarlo. Nótese que los elementos de las dos primeras pestañas son arrastrables sobre el entorno de desarrollo. Cada vez que se

desea añadir código al programa que se está realizando, se debe de hacer por medio del arrastre del botón correspondiente. Cuando el arrastre se produce, el programa que se está realizando indica con cuadros verdes las posibles zonas en las cuales se puede soltar el código que se desea añadir.

- El entorno de desarrollo. Es la parte de la aplicación en la cual se añade el código del programa que se desea realizar. El código que en ella se muestra es interactivo; a la hora de añadir código las diferentes partes del programa se resaltan para que se pueda seleccionar aquélla en la que éste se desea añadir. El código que ya se ha añadido al programa también se puede editar por medio de un doble clic. Cuando una parte del código se edita, se lanza una ventana que muestra el contenido a editar.

Durante la resolución de un problema se ofrece al alumno, en todo momento, el conjunto completo de sentencias existentes en el sistema, tanto si son correctas, como si no. Con esto se consigue evaluar si la intención del alumno a la hora de programar en pseudolenguaje es correcta o no. Además, la interfaz evita el uso de ventanas desplegables y elementos para auto-completar código. Se ha considerado que esta característica no es adecuada cuando el usuario está aprendiendo, ya que muchas veces ayuda al alumno a evitar posibles errores que el alumno podría cometer, y por tanto dificulta el poder aprender de ellos.

4. Representación del conocimiento

Para que el módulo pedagógico pueda inferir el conocimiento del alumno a partir del código que ha escrito, es necesario representar la solución escrita mediante información que pueda ser procesada y evaluada. Concretamente, la representación se realiza de forma declarativa mediante una serie de hechos, los cuales se van obteniendo simultáneamente a la escritura del código a través del espacio de trabajo del sistema.

Cada elemento del código tiene asociado uno o más hechos que pueden estar a su vez relacionados con otros mediante referencias. Esta relación está determinada por la situación de los elementos en el código: métodos de una clase, parámetros de un método, tipos, etc. Por ejemplo, si una variable está definida dentro de un método, existirán referencias entre el hecho que representa a la variable y el que representa al método. De esta forma, una solución propuesta por un alumno estará representada por un conjunto de hechos relacionados entre sí formando una estructura de árbol.

Hay que distinguir tres tipos de hechos: a) Los predefinidos por el sistema, que representan los tipos y sentencias propias del pseudolenguaje. b) Los incluidos por el profesor junto con el enunciado del problema. Éstos representan algoritmos predefinidos que el alumno debe utilizar para construir la clase correspondiente. c) Los que se van añadiendo dinámicamente conforme el alumno va elaborando la solución al problema, a través del espacio de trabajo. Éstos podrán verse modificados o eliminados dependiendo de si el alumno cambia o elimina el código asociado, afectando también a aquellos hechos que estén involucrados.

Los hechos a su vez, son representados mediante plantillas. Teniendo en cuenta la asociación entre código-hechos-plantillas existirán diferentes tipos de plantillas correspondientes a las diferentes sentencias del código: clases, métodos, atributos, tipos básicos, parámetros, operadores aritméticos, llamadas a métodos, etc. Todas estas plantillas tienen en común una serie de campos: un identificador unívoco, el nombre de la plantilla, una referencia a la plantilla en la que se define la sentencia asociada, y otros campos que opcionalmente pueden hacer referencia a operadores o tipos implicados en la expresión. Por ejemplo, un método tendrá un campo adicional para mantener las referencias a sus parámetros.

5. Modelado del alumno

Para poder adaptar el proceso de aprendizaje del alumno es necesario disponer de un modelo preciso de éste. Existen varios estudios que definen esta tarea como un problema inherentemente inmanejable [12]. No obstante, asumiendo que es complejo elaborar un modelo que se acople con exactitud a la realidad, aproximaciones a ésta también pueden considerarse útiles desde el punto de vista del aprendizaje.

En este trabajo, para esta tarea se ha utilizado, principalmente, el *Modelado Basado en Restricciones* (MBR) (en inglés, *Constraint Based Modeling*), propuesto en la teoría de Ohlsson [9]. Esta técnica de modelado propone el aprendizaje a partir de los errores que comete un alumno, generando refuerzos que le ayuden a aprender de ellos. Según esta técnica, el dominio posee unos principios básicos que deben ser cumplidos por todas las soluciones. Dicho de otro modo, estos principios definen una serie de restricciones que no pueden ser violadas por ninguna solución. Por tanto, lo importante en el MBR no es lo que hace el alumno, sino el estado de la solución que esté elaborando. La principal ventaja del MBR es su simplicidad de cómputo, ya que el modelo del estudiante se reduce a una correspondencia de patrones, permitiendo que se puedan expresar de forma compilada como redes Rete [4]. La elaboración del modelo del alumno, empleando esta técnica, consiste en identificar las restricciones que éste ha violado.

Para manejar las reglas que representan las restricciones se debe usar un motor de reglas. En este trabajo se ha optado por JESS (Java Expert System Shell) [5], que puede ser incorporado por las aplicaciones escritas en el lenguaje Java. JESS se caracteriza por tener un tamaño muy reducido y una gran velocidad de ejecución, hecho más que interesante cuando se desarrolla una aplicación que funciona a través de Internet. Este motor de reglas utiliza una versión mejorada del algoritmo Rete para procesar sus reglas mediante semejanza de patrones de una alta eficiencia.

El modelo del alumno también almacena las acciones realizadas por éste, información que, junto con las restricciones violadas, será usada por el módulo pedagógico para determinar las debilidades en su conocimiento y, de esta forma, poder actuar en consecuencia para corregirlas. Además, para determinar cuál es problema más adecuado para él, se infiere y almacena su nivel de conocimiento.

Éste se representa mediante un valor comprendido en el intervalo [0,10], distinguiendo tres categorías o subniveles: *bajo*, *medio* y *alto*, que dividen al intervalo anterior en tres partes de igual tamaño. El cálculo del nivel se realiza después de la última compilación del problema que se esté resolviendo, y consiste en comparar la puntuación obtenida por el alumno en los últimos problemas (puede configurarse el número de problemas), y el resto de estudiantes de su mismo nivel. Si la puntuación es superior, entonces se aumenta de nivel, en otro caso, continúa en el mismo.

6. Módulo pedagógico

Este módulo asiste al alumno durante su proceso de aprendizaje de dos formas diferentes. Por una parte, seleccionando de forma adaptativa el ejercicio que debe resolver, y una vez que el alumno aborda la realización del problema, detectando sus deficiencias aplicando el MBR.

6.1. Selección Adaptativa de Problemas

Para seleccionar el problema más adecuado, se utiliza el nivel de conocimiento de éste, la puntuación de las reglas y la dificultad del problema.

La puntuación de una regla representa el nivel de error que supone violar la restricción asociada. Se encuentra en el intervalo (0,10], siendo un error más grave cuanto mayor es la puntuación. Este intervalo se divide en tres partes iguales representando diferentes niveles de gravedad: *no importante*, *medio* y *grave*.

Las puntuaciones iniciales han sido asignadas por los expertos (profesores de la asignatura) que se encargaron de recopilar las restricciones necesarias y que conocen la gravedad de cada una. Posteriormente, la puntuación de cada regla se recalcula con cada compilación. Para ello se incrementa en uno el número de veces que se ha infringido la regla y se compara con la media de las demás reglas. Cuanto mayor sea la frecuencia del error cometido, mayor será su gravedad. Así la puntuación $p_r(t+1)$ de la regla r tras la compilación $t+1$ se puede expresar como:

$$p_r(t+1) = p_r(t) \times \frac{\delta}{\overline{p(t)}} \tag{1}$$

donde δ es el valor medio de la categoría a la que pertenece la regla, es decir, 1,65 si el grupo al que pertenece la regla es *no importante*, 5 si *medio*, y 8,35 si *grave*. $\overline{p(t)}$ se calcula aplicando la siguiente fórmula, donde N es el número total de reglas.

$$\overline{p(t)} = \frac{\sum_{i=1}^{N} p_i(t)}{N} \tag{2}$$

La dificultad de un problema también se representa mediante un valor del intervalo (0,10]. Para determinarla, cada vez que el alumno realiza la compilación,

se cuantifican y almacenan los errores cometidos sumando la puntuación de las reglas correspondientes. Los errores repetidos en diferentes compilaciones no se tienen en cuenta en este cálculo con el fin de evitar la acumulación de errores en la puntuación total (suma de puntuaciones de cada compilación).

Al finalizar un problema e, su dificultad d_e se calcula mediante la comparación de la puntuación media del problema (ecuación 4), obtenida de todos los alumnos (M) que lo han resuelto (donde t_{ei} es la puntuación del alumno i en el problema e), con la puntuación media de todos los problemas existentes (ecuación 5):

$$d_e = \frac{\overline{t_e}}{\overline{P_{TP}}} \times 5 \qquad (3) \qquad \overline{t_e} = \frac{\sum_{i=1}^{M} t_{ei}}{M} \qquad (4) \qquad \overline{s} = \frac{\sum_{j=1}^{E} d_j}{E} \qquad (5)$$

Una vez determinados estos parámetros, para seleccionar el problema más adecuado, hay que contemplar dos características: el nivel del alumno, para presentarle un problema conforme a éste; y sus deficiencias, para poder saber qué problema las puede tratar mejor. La selección del problema se realiza en varios pasos: primero se seleccionan todos los problemas del mismo nivel que posee el alumno, y se ordenan de mayor a menor dificultad. Si no existen problemas en el mismo nivel, es necesario ordenar de la forma adecuada el resto de problemas para seleccionar uno que sea apropiado, para ello, se ordenan de menor a mayor, si el alumno tiene un nivel *bajo*, o de mayor a menor, si posee cualquiera de los otros dos niveles.

6.2. Resolución Asistida de Problemas

Siguiendo con la metodología del MBR, los hechos generados a partir del código escrito por el alumno son utilizados para determinar cuáles son las deficiencias de su conocimiento. Este proceso consiste en introducirlos en el motor de reglas JESS, que realizará un proceso de inferencia, tal y como se describe en el algoritmo Rete [4], determinando los errores cometidos.

Inicialmente el motor contendrá hechos generados automáticamente, correspondientes a los tipos y métodos predefinidos. Además, al iniciar el motor, se añaden a éste, por un lado, funciones necesarias para la evaluación de las reglas de inferencia, utilizadas para realizar comprobaciones, como por ejemplo, ver si el formato del nombre de una función es correcto; y por otro lado, restricciones.

Las restricciones, son reglas de inferencia en las que el antecedente representa situaciones erróneas y el consecuente, básicamente, informa de ellas. Si el alumno comete cierto error, el antecedente de la regla asociada se evaluará positivamente, y se realizará el consecuente. En el sistema se han definido restricciones de varios tipos de forma que se puedan detectar las diferentes carencias en el conocimiento del alumno. Las *restricciones sintácticas* se relacionan con la infracción de reglas de la gramática, considerando también las recomendaciones de estilo que se realizan en la mayoría de los lenguajes, con relación al uso de mayúsculas al principio de las clases, nombres de las variables, etc. Aunque supuestamente son sólo recomendaciones, se ha optado por considerarlas errores

para fomentar la utilización de un convenio en el nombrado desde el comienzo del aprendizaje. Las *restricciones de visibilidad* abarcan todos los errores derivados del desconocimiento de las reglas de ámbito de clases, métodos, variables, parámetros, etc. Las *restricciones de tipos* buscan incoherencias entre los tipos de las operaciones aritméticas, llamadas a métodos, devolución de variables, etc., que realiza el estudiante. Dentro de este tipo de restricciones, también se consideran errores como el no completar los operadores de alguna sentencia. Las *restricciones sobre el formato de los nombres* abordan los errores relacionados con el nombre de variables locales, parámetros, métodos, nombres de clases, etc., así como posibles dependencias que existan entre ellos. Por último, las *restricciones sobre referencias perdidas* abarcan todos los errores que pueden tener lugar como consecuencia de la eliminación de código al borrar la declaración de alguna variable, atributo o método, que había sido utilizada en el código, existiendo por tanto una referencia a una parte de código que ya no existe. En total se han elaborado 71 restricciones, de las cuales 15 son de visibilidad, 13 de sintaxis, 9 de tipos, 21 sobre el formato de los nombres, y 13 sobre referencias perdidas.

Para dar una idea más clara sobre el funcionamiento de las restricciones, se muestra un ejemplo de una de las reglas utilizadas, perteneciente a la categoría de restricciones sintácticas, y que comprueba que la declaración de las variables locales se realiza en el cuerpo de un método.

```
(defrule declared-var-wrong-sintax
    (status (compiling TRUE))
    (local-var  (id ?id) (parent-id ?p-id))
    (test (eq (is-valid-id ?p-id) TRUE))
    (test (neq (fact-slot-value (fact-id ?p-id) template) imple-method))
    =>
    (bind ?ids (create$ ?id))
    (bind ?error-text-easy (create$ "Una declaración de una variable local
                sólo puede hacerse directamente en el cuerpo del método"))
    (bind ?error-text (create$ "Sintaxis incorrecta"))
    (bind ?rule-name (create$ (rule-name)))
    (throw-error ?rule-name ?ids ?error-text-easy ?error-text)
    (printout t "VARIABLE DECLARADA FUERA DE UN MÉTODO" crlf))
```

En el antecedente se establecen las condiciones que definen un error en los conceptos del alumno: se verifica que la variable sea local mediante el predicado *local-var*, que tenga un valor válido (predicado *is-valid-id*), y por último se comprueba que haya una situación de error con respecto a la declaración de la variable, que se daría si el hecho "padre"(correspondiente a *?p-id*), en el que ha sido definida dicha variable, no es un método. En el consecuente de la regla se realiza el tratamiento del error definiendo distintos niveles de detalle de información y lanzando el error para que sea manejado por el motor de reglas. La posible información a mostrar se adapta dinámicamente al nivel de conocimiento del alumno, pudiendo ser muy precisa o aproximándose prácticamente a la que

daría cualquier compilador actual si el nivel es alto. Un ejemplo de estos errores se puede observar en la figura 1.

7. Trabajos relacionados

Este sistema utiliza la teoría de Ohlsson sobre el MBR. Dentro de los sistemas educativos que emplean el MBR, los más populares son SQL-Tutor y Kermit. Ambos se centran en el dominio de las bases de datos. Según sus autores, SQL-Tutor [8,7] intenta guiar al alumno adaptándose a sus necesidades y habilidad de aprendizaje, al igual que lo haría un tutor humano. Este sistema se centra únicamente en el aprendizaje de la sentencia 'select'. La primera versión del sistema fue desarrollado en 1997.

El sistema KERMIT [10], fue desarrollado en el 2002, y tiene como objetivo la enseñanza del diseño conceptual de bases de datos mediante el modelo Entidad-Relación. Al igual que SQL-Tutor, se basa en el modelado de restricciones, verifica los errores cometidos, y presenta ayuda ante ellos, procurando adaptarse a las necesidades del alumno. Pero además, intenta conocer hasta qué punto la solución del alumno es correcta comparándola con una solución ideal elaborada por un experto. Actualmente contempla cerca de 100 restricciones, diferentes niveles del alumno, y utiliza una interfaz gráfica para realizar el modelo Entidad-Relación.

8. Conclusiones y Trabajos Futuros

En este artículo se ha presentado un sistema inteligente para ayudar a los alumnos a aprender FPOO. Un prototipo del sistema puede ser probado en la dirección Web *http://www.lcc.uma.es/fpoo* (con usuario y contraseña 'demo'). Esta herramienta permite realizar un proceso de aprendizaje personalizado y adaptado a las capacidades de cada estudiante. Para ello se consideran las soluciones realizadas previamente por éste, así como también, las realizadas por el resto de alumnos sobre el problema en cuestión. También las ayudas e información se muestran de acorde al nivel estimado. De esta manera, el sistema es capaz de reforzar las deficiencias del alumno, lo que la convierte en una herramienta de gran ayuda para impartir las clases de la asignatura de Elementos de Programación. Además, el sistema está diseñado de forma que permite al profesor en cualquier momento añadir nuevos ejercicios.

El resultado de este trabajo es una herramienta funcional que se ha comenzado a utilizar este curso como complemento a las clases presenciales. Sin embargo, existen todavía diversos aspectos que se podrían mejorar. Aunque el sistema se centra en enseñar FPOO, habría que estudiar si es factible permitir la inclusión también de sentencias de selección e iteración. Para esto habría que identificar y elaborar las restricciones necesarias para esas sentencias. En este sentido el principal problema es que el añadir estas sentencias dificulta aún más el proceso de determinación de si el código escrito resuelve el problema indicado en el enunciado. La línea de trabajo más inmediata es el desarrollo de un modelo basado

en datos estadísticos que sustituya a los heurísticos que emplea actualmente el sistema y que se basan en la información proporcionada por los expertos. Desde el punto de vista técnico, otras mejoras que se pueden realizar son: la elaboración de herramientas que permitan un análisis exhaustivo de los datos recopilados; la mejora de las herramientas de gestión de los problemas y de las restricciones definidas por los expertos; etc.

Referencias

1. Anderson, J.: 1983, *The Architecture of Cognition*. Cambridge, MA: Harvard University Press.
2. Bloom, B.S.: 1984, *The 2-sigma problem: The search for methods of group instruction as effective as one-to-one tutoring*. Educational Researcher, 13, pp. 4–16.
3. Ferrero, B. (2004). *DETECTive: un entorno genérico e integrable para diagnóstico de actividades de aprendizaje*. Tesis doctoral. Universidad del País Vasco.
4. Forgy, C. L. (1982). *Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem*. Artificial Intelligence 19, pp. 17–37.
5. Friedman-Hill, E.J. (1997). *JESS, The Java Expert System Shell*. SAND–98-8206, November.
6. Mitrovic, A. (2002). *SINT - A Symbolic Integration Tutor*. Conf. on Intelligent Tutoring Systems. LNCS 2363. pp. 587–595.
7. Mitrovic, A. (2003). *An intelligent SQl tutor on the Web*. Int. J. Artificial Intelligence in Eucation, vol. 13, no. 2-4, pp 173–197.
8. Mitrovic, A., Martin, B. & Mayo, M. (2002). *Using Evaluation to Shape ITS Design: Results and Experiences with SQL-Tutor*. Int. J. User Modeling and User-Adapted Interaction, vol. 12, no. 2-3, pp. 243-279.
9. Ohlsson, S. (1994). *Constraint-based Student Modeling*. In Student Modelling: the Key to Individualized Knowledge-based Instruction. Springer. pp. 167-189.
10. Suraweera, P. and Mitrovic, A. (2002). *KERMIT: a Constraint-based Tutor for Database Modeling*. In: S. Cerri, G. Gouarderes and F. Paraguacu (eds.) Proc. 6th Int. Conf on Intelligent Tutoring Systems, LCNS 2363, pp. 377–387.
11. VanLehn, K.: 1996, *Conceptual and Meta Learning During Coached Problem Solving*. In: C. Frasson, G. Gauthier, A. Lesgold (eds.): Intelligent Tutoring Systems. New York: Springer-Verlag, pp. 29–47.
12. Zukerman, I. and D. Albrecht (2001). *Predictive Statistical Models for User Modeling*. User Modeling and User–Adapted Interaction 11, pp. 5–18.