Un entorno educativo integrado para la visualización de métodos algorítmicos y estructuras de datos: de la especificación algebraica a la implementación

Clara Segura Isabel Pita Rafael del Vado Ana Isabel Saiz Pablo Soler

Departamento de Sistemas Informáticos y Computación

Universidad Complutense de Madrid

C/ Profesor José García Santesmases s/n, 28040 Madrid

{csegura,ipandreu,rdelvado}@sip.ucm.es, {anussita,ileras}@gmail.com

Resumen

El alto nivel de abstracción necesario para la enseñanza de "estructuras de datos" y "métodos algorítmicos" dificulta su comprensión por parte del alumnado. Con el fin de abordar de una forma eficaz esta situación, en los últimos años se ha desarrollado e implantado en la Facultad de Informática de la Universidad Complutense de Madrid un innovador sistema de enseñanza interactiva acorde con las nuevas enseñanzas del *Espacio Europeo de Educación Superior* (EEES). En este trabajo presentamos las dos principales aportaciones de este sistema. En primer lugar se describe la herramienta *Vedya* para la visualización de estructuras de datos y algoritmos. En segundo lugar, el sistema *Maude* para la ejecución de especificaciones algebraicas de tipos abstractos de datos mediante *Eclipse*, con el cual es posible estudiar desde el nivel más abstracto de una especificación software hasta su implementación concreta en *Java*, permitiendo así el proceso de auto-aprendizaje del alumno.

1. Motivación

El estudio de las "estructuras de datos" y de los "métodos algoritmos" constituye uno de los aspectos esenciales del curriculum de todo ingeniero informático. Sin embargo, el alto nivel de abstracción necesario para la enseñanza de estos temas dificulta a menudo su comprensión por parte del alumnado. Con el fin de abordar de una forma eficaz esta situación, en los últimos años se ha desarrollado e implantado en la Facultad de Informática de la Universidad Complutense de Madrid un innovador sistema de enseñanza interactiva acorde con las nuevas enseñanzas del EEES y el modelo de aprendizaje centrado en el alumno. En este trabajo presentamos las dos principales aportaciones de este sistema: por un lado, la herramienta de visualización *Vedya* mediante la cual es posible proporcionar al alumno un sistema integrado de aprendizaje tanto de las principales estructuras de datos como de los métodos algorítmicos más relevantes. Por otro lado, el sistema *Maude* para la ejecución de "especificaciones algebraicas" de tipos abstractos de datos a través del lenguaje de especificación formal que proporciona este sistema.

Gracias al entorno de programación *Eclipse* se ha conseguido un sistema totalmente integrado de utilidad práctica tanto para alumnos como profesores, que permite pasar desde el nivel más abstracto de una estructura de datos, proporcionado por su especificación algebraica en *Maude*, hasta su implementación concreta en un lenguaje de programación moderno como es *Java*. Todo este proceso de aprendizaje puede ser guiado y tutorizado de una manera completamente autónoma mediante la utilización de la herramienta *Ved-*

ya, mediante la cual es posible consultar la documentación relativa a cada una de las especificaciones algebraicas, distinguir entre el comportamiento de la estructura y sus diferentes implementaciones a través del uso de diferentes vistas, o consultar información sobre el coste de las diferentes implementaciones propuestas.

2. La herramienta Vedya

Vedya es un entorno interactivo para el aprendizaje de estructuras de datos y esquemas algorítmicos. Cubre las estructuras de datos más comunes: pilas, colas, árboles binarios de búsqueda, árboles AVL, colas de prioridad y tablas. También proporciona otros tipos de datos como un "consultorio médico". Con respecto a los esquemas algorítmicos cubre los principales métodos de resolución [4]: divide y vencerás, método voraz, programación dinámica, vuelta atrás y ramificación y poda. Todas las estructuras de datos y métodos algorítmicos enseñados en las asignaturas afines se hallan así integrados en el mismo entorno. *Vedya* permite trabajar sobre distintas estructuras de datos y con varias secuencias de operaciones sobre la misma estructura mediante un sistema de multiventanas y multipestañas.

Actualmente existen dos versiones de la herramienta. La primera contiene todas las estructuras de datos y esquemas algorítmicos mencionados anteriormente mientras que la nueva versión ofrece por el momento un subconjunto en un entorno visual más atractivo. Esta última se puede encontrar en `http://www.fdi.ucm.es/profesor/csegura/`.

Existen varias opciones de uso de la herramienta. La principal es la ejecución interactiva pero también es posible crear simulaciones que se ejecutan automáticamente, visualizar tutoriales y resolver tests dentro del mismo entorno. También incorpora un conjunto de animaciones que muestran cómo se usan las estructuras de datos para resolver determinados problemas. En la Fig. 1 se muestra un ejemplo de la ventana principal para los árboles binarios de búsqueda. El panel central se usa para representar la estructura. En el caso de los árboles se ofrecen facilidades gráficas para expandir o contraer el árbol, o para moverse por la pantalla de forma que se puedan ver las partes escondidas de árboles muy grandes. En el lado izquierdo figura la lista de operaciones que se pueden ejecutar. Las operaciones parciales no permitidas aparecen deshabilitadas. En el panel de la derecha se visualiza la secuencia de acciones realizadas. A continuación podemos seguir ejecutando acciones, podemos retroceder en la secuencia para visualizar estados previos o podemos usar las facilidades de simulación (botones estándar para ejecutar, parar, avanzar y retroceder de la parte superior) para ejecutar de manera continuada la secuencia desde el punto deseado. Obsérvese que en la parte superior siempre se muestra la última acción realizada.

Existen dos tipos de vistas: la del comportamiento de la estructura para comprender de manera intuitiva su funcionamiento; y una o varias vistas de implementación, por ejemplo estática o dinámica. En la Fig. 2 se muestra la vista de comportamiento de una cola, la representación de su implementación estática basada en un vector circular y la implementación dinámica basada en punteros.

Adicionalmente, el entorno proporciona documentación que incluye la especificación algebraica, el código de las implementaciones habituales y el coste de las operaciones en dichas implementaciones.

En la parte superior de la pantalla, un menú facilita el manejo del sistema. Permite crear una nueva estructura, abrir una previamente guardada o guardar el estado de la actual. También permite ejecutar las operaciones de la estructura, usar las facilidades de simulación y cambiar la velocidad de las animaciones.

La ventana principal para la ejecución de los algoritmos es similar. Se han implementado los algoritmos basados en divide y vencerás de búsqueda binaria en un vector y quicksort; los algoritmos que resuelven el problema de la mochila (en su versión fraccionaria o no) basados en programación dinámica, método voraz

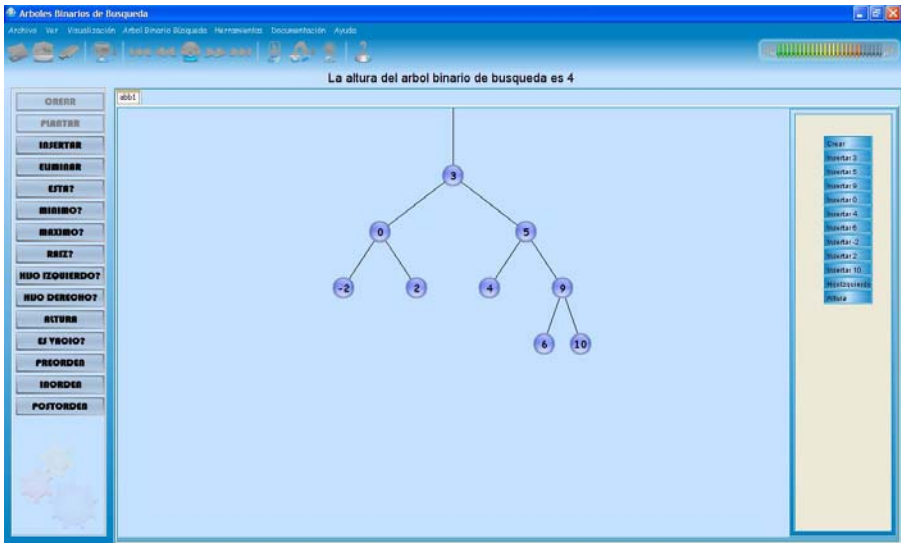

Figura 1: Ventana principal para los árboles binarios de búsqueda en *Vedya*

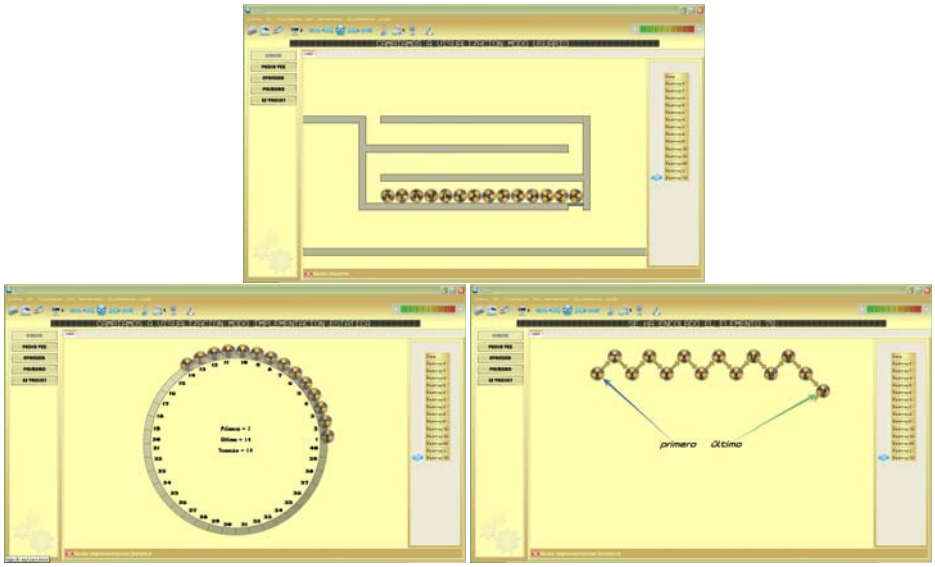

Figura 2: Vistas de comportamiento e implementación de las colas

y ramificación y poda; así como el algoritmo voraz de Dijkstra para calcular los caminos mínimos en un grafo.

Como se ha comentado anteriormente *Vedya* contiene tutoriales sobre tipos de datos (pilas, colas, árboles binarios de búsqueda, árboles rojinegros, colas de prioridad y árboles 2-3-4) y animaciones de algoritmos que muestran el uso de una estructura de datos para resolver un problema (evaluación de una expresión en forma posfija, transformación de una expresión de forma infija a posfija, recorrido en anchura de un árbol, comprobación de palabras palíndromas). También, se dispone de animaciones sobre grafos: árbol de recubrimiento con coste mínimo mediante los algoritmos de Prim y Kruskal, y cálculo de los caminos mínimos mediante el algoritmo de Dijsktra.

Finalmente, *Vedya* ofrece la herramienta *Vedya-test* para resolver tests, que se puede ejecutar de manera independiente. Dicha herramienta permite a los profesores crear, modificar o borrar preguntas de una base de datos, y crear tests a partir de ellas. Los estudiantes visualizan los tests, los resuelven y pueden consultar las soluciones correctas. Las preguntas están agrupadas por temas en la base de datos pero se pueden mezclar preguntas sobre distintas estructuras de datos en el mismo test.

3. Ejecución de especificaciones algebraicas en Maude

Para la ejecución de las especificaciones algebraicas se ha hecho uso del lenguaje *Maude* [3], basado en la *lógica de reescritura*. El lenguaje se encuentra disponible para *Linux* y *MacOs* en `http://maude.cs.uiuc.edu`, y también existen extensiones para su ejecución en *Windows* en `http://moment.dsic.upv.es`.

Las especificaciones se pueden ejecutar en *Eclipse* mediante "plugins" especiales desarrollados en el DSIC de la UPV y en el DLCC de la UMA. Este entorno, mostrado en la Fig. 3, facilita la utilización a los alumnos al integrar el editor de texto con los comandos de ejecución del sistema. A la izquierda se presentan los proyectos desarrollados; en la parte central se encuentra el editor, sobre el cual está el panel de ejecución del sistema; y en la parte inferior la consola donde se muestra el resultado de la ejecución. En la parte derecha se pueden abrir otras ventanas que permiten definir diferentes opciones del sistema y hacer depuración.

En *Maude* el elemento básico de una especificación es un "módulo". El lenguaje permite la definición de *módulos funcionales* utilizados para definir los tipos de datos; *módulos de sistema* utilizados para definir sistemas de reescritura; y *módulos orientados a objetos*, que permiten utilizar una sintaxis de clases, objetos y mensajes. Un módulo funcional para las colas presenta el siguiente aspecto:

```
fmod COLA{X :: TRIV} is
  sort Cola{X} .
  op error : -> Cola{X} .
  op error : -> X$Elt .
  op cola-vacia : -> Cola{X} .
  op pedir-vez : Cola{X} X$Elt -> Cola{X} .
  op avanzar : Cola{X} -> Cola{X} .
  op primero : Cola{X} -> X$Elt .
  op es-cola-vacia? : Cola{X} -> Bool .
  var C : Cola{X} .
  var E : X$Elt .
  eq avanzar(cola-vacia) = error .
  ceq avanzar(pedir-vez(C,E)) = cola-vacia
    if es-cola-vacia?(C) .
  ceq avanzar(pedir-vez(C,E)) =
    pedir-vez(avanzar(C),E))
    if not es-cola-vacia?(C) .
  eq primero(cola-vacia) = error .
  ceq primero(pedir-vez(C,E)) = E
    if es-cola-vacia?(C) .
  ceq primero(pedir-vez(C,E)) = primero(C)
    if not es-cola-vacia?(C) .
  eq es-cola-vacia?(cola-vacia) = true .
  eq es-cola-vacia?(pedir-vez(C,E)) = false
endfm
```

El lenguaje permite importar otros módulos, definir varios *tipos de datos*, definir las *operaciones* sobre los tipos y las *ecuaciones* que definen el comportamiento de las operaciones. Los módulos pueden ser parametrizados, utilizandose "teorías" para definir los parámetros

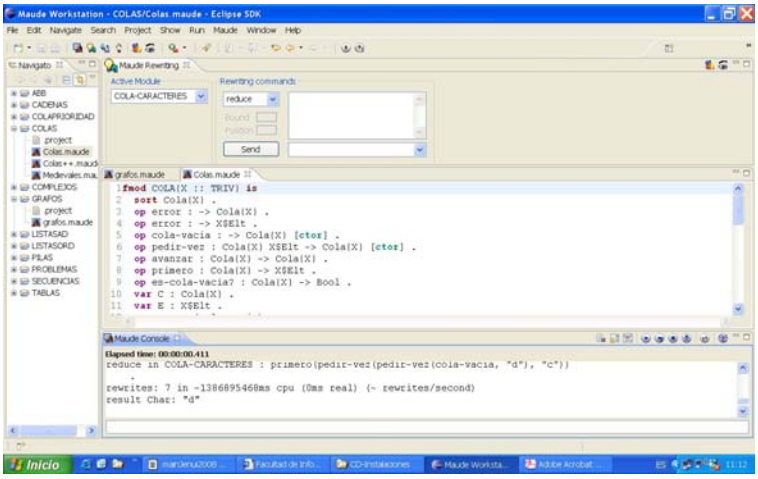

Figura 3: Ejecución en *Maude* bajo el entorno *Eclipse*

y "vistas" para relacionar el parámetro formal con el parámetro real. El sistema tiene predefinidos los tipos abstractos de datos más utilizados, así como las teorías y vistas más comunes:

```
view Int from TRIV to INT is
  sort Elt to Int .
endv
fmod COLA-ENTEROS is
  protecting COLA{Int} .
endfm
```

Como puede observarse la sintaxis es parecida a la utilizada en varios textos de especificaciones algebraicas de tipos de datos [4, 5].

Para ejecutar una especificación el alumno introduce el texto en el editor; a continuación ejecuta el sistema *Maude* mediante los botones existentes en la consola e introduce el módulo. El sistema detecta los errores de sintaxis existentes y los muestra en la consola. Una vez que el módulo ya no tiene errores el alumno puede reducir términos utilizando las ecuaciones del módulo. Para ello puede usar el cuadro de comandos existente en la parte superior de la pantalla, o bien escribir directamente el comando en el editor e introducirlo en el sistema. Por ejemplo, para obtener el `primero` de una cola podemos reducir el término: `red primero(pedir-vez(pedir-vez(cola-vacia, 5),4))`. Este término se debe reducir sobre el módulo instanciado de las colas con la teoría de los números enteros `INT`. En nuestro ejemplo ese módulo se denomina: `COLA-ENTEROS`.

La posibilidad de reducir términos de forma automática permite a los alumnos realizar una prueba inicial de sus especificaciones detectando muchos de los errores cometidos al definir las operaciones mediante ecuaciones. Otra ventaja aún mayor que se obtiene al ejecutar las especificaciones es que el alumno comprende la diferencia entre el *módulo parametrizado* y el *módulo instanciado* al poder reducir términos sobre distintos módulos. Por ejemplo, se podría declarar un nuevo módulo `COLA-CARACTERES` sobre el que reducir términos del tipo `red primero(pedir-vez(pedir-vez(cola-vacia, 'a'),'c'))`.

Durante el curso se han realizado prácticas de las estructuras de datos incluidas en *Vedya*, las cuales pueden encontrarse en `http://www.fdi.ucm.es/profesor/csegura/`.

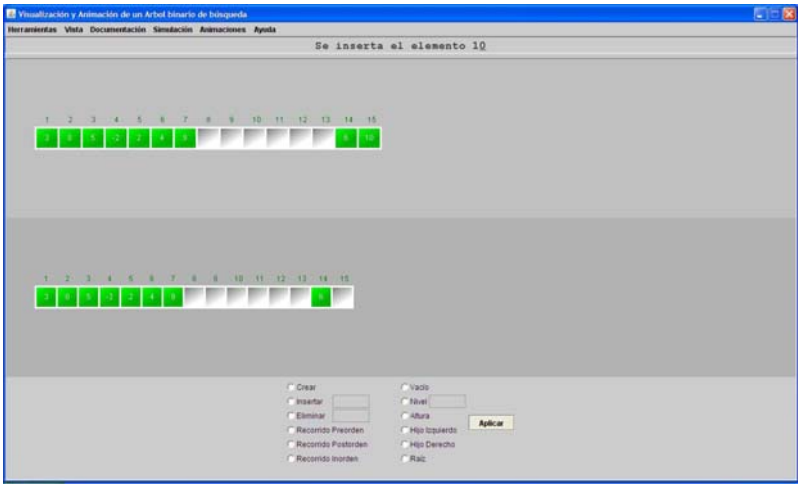

Figura 4: Implementación estática de árboles binarios de búsqueda en *Vedya*

También se propusieron otros ejemplos de tipos de datos, como las "colas medievales" o un "consultorio médico" [4]. En todas ellas se propone definir un tipo de datos paramétrico e instanciarlo con distintas teorías. Las prácticas se completan con diferentes términos que el alumno debe reducir sobre alguno de los módulos instanciados para probar la especificación, así como propuestas de pequeños cambios en algunas operaciones, o bien definiciones erróneas para que las detecten. Al tratarse de estudiantes de segundo curso se han utilizado solo algunas de las facilidades del lenguaje. En cursos superiores en que los alumnos tienen más conocimientos puede utilizarse toda la riqueza del lenguaje (relación de subtipos, lógica de pertenencia, sistemas de reescritura).

4. De la especificación algebraica a la implementación

La herramienta *Vedya* se convierte en un instrumento pedagógico de elevado interés práctico cuando se trata de dirigir todo el proceso de auto-aprendizaje de las principales estructuras de datos, desde su especificación algebraica hasta sus posibles implementaciones, dentro de un entorno tan potente como es el que se ha descrito en la sección precedente.

Los alumnos obtienen una primera toma de contacto de la estructura de datos que van a estudiar a través de la utilización de *Vedya*. Por ejemplo, si su aprendizaje se centra en los *árboles binarios de búsqueda*, comenzarán accediendo a la sección correspondiente de la herramienta, donde podrán experimentar libremente y por sí mismos con cada una de las operaciones ofrecidas por esta estructura (véase la Fig. 1). Con el fin de afianzar este conocimiento intuitivo, el alumno dispone adicionalmente de la posibilidad de utilizar la herramienta *Vedya-test*.

Una vez que el alumno ha obtenido una idea clara del comportamiento informal de la estructura de datos, puede pasar a trabajar con el sistema *Eclipse*. El primer paso consistirá en plasmar formalmente ese conocimiento intuitivo que ha obtenido a través de la utilización de *Vedya* en una especificación algebraica concreta escrita en *Maude*. Con el fin de facilitar este difícil salto en su auto-aprendizaje, el alumno puede utilizar, de forma interactiva, la documentación que se incluye en el manual

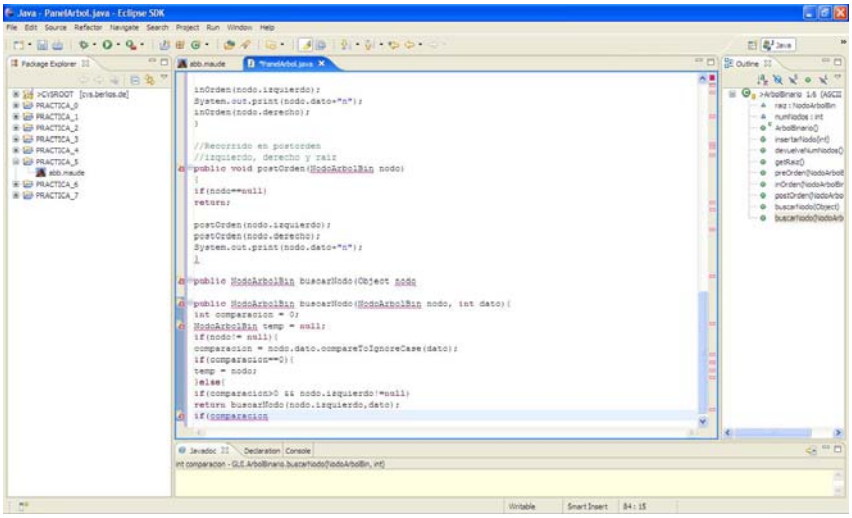

Figura 5: Implementación de árboles binarios de búsqueda en *Java* utilizando *Eclipse*

de la herramienta *Vedya*.

Una vez introducida esta especificación en el sistema *Eclipse* (véase la Fig. 3), ya puede pasar a ejecutar en *Maude* pequeñas pruebas, con el propósito de comprobar si coincide o no con la noción intuitiva e informal de la estructura de datos de la que inicialmente partía. La experimientación así realizada le permitirá alcanzar el nivel de abstracción necesario que en toda especificación formal de una componente software se requiere y que tan difícil es de adquirir en la enseñanza de la Informática, siempre sobre la base inicial de un conocimiento intuitivo y experimental.

Obtenida la especificación algebraica de la estructura de datos, el siguiente paso consiste en desarrollar una implementación en un lenguaje moderno de programación como es *Java*, a través de las facilidades proporcionadas por el entorno de programación en *Eclipse*. En esta ocasión, el alumno puede hacer uso de la especificación algebraica que acaba de construir como si se tratara de un auténtico "manual de instrucciones". La principal ventaja de nuestra metodología es que la especificación se comporta como un *prototipo* de la estructura de datos a implementar, de forma que el alumno consigue averiguar cual es su comportamiento exacto en todos aquellos casos de duda que le puedan surgir durante el proceso de diseño, incluso antes de que pueda compilar su programa. Con el fin de poder guiar de una forma más específica el paso de la especificación a la implementación, el alumno puede hacer uso de nuevo de la herramienta *Vedya*. En esta ocasión, puede acceder desde el menú de opciones a la parte que se correspondería con la implementación de la estructura de datos que está estudiando (véase la Fig. 4). Allí puede experimentar con distintas posibilidades de implementación.

Una vez que el alumno se ha familiarizado con las distintas implementaciones de su estructura, está ya suficientemente preparado para decidir adecuadamente una *representación* en el lenguaje *Java* (véase la Fig. 5). La posibilidad de haber entendido y evaluado previamente distintas implementaciones mediante *Vedya* también le permite al alumno adquirir un conocimiento claro del *coste algorítmico* de la codificación de cada una de las operaciones epecificadas en función del ti-

	2002/03	2003/04	2004/05	2005/06	2006/07
No presentados	57.6 %	45.3 %	42.3 %	64.7 %	50.8 %
Aprobados	15.3 %	22.2 %	20.2 %	18.2 %	30.1 %
Suspensos	27.1 %	32.5 %	37.5 %	17.1 %	18.9 %

Figura 6: Comparativa de resultados académicos.

po representante elegido, de forma que este sea también un criterio decisivo en el momento de diseñar su implementación. Aquí juega un papel importante la parte de *Vedya* dedicada a los *métodos algorítmicos*, pues permite al alumno adquirir una buena metodología.

5. Conclusiones

En este trabajo se ha descrito la utilización de un entorno educativo innovador para la enseñanza interactiva de estructuras de datos y métodos algorítmicos a través de la herramienta de visualización *Vedya* y del lenguaje *Maude* con su entorno de programación bajo *Eclipse*.

En los últimos años han surgido numerosos trabajos sobre visualización de estructuras de datos y algoritmos. Por ejemplo, en [1] se presenta una herramienta del mismo estilo. Sin embargo, muchos carecen de una interfaz gráfica de usuario común a las estructuras de datos y algoritmos o bien solamente se pueden ejecutar en algunos sistemas operativos. En este sentido, *Vedya* es algo más que una simple herramienta de ejecución de estructuras de datos como se ha mostrado en la Sección 4.

La aplicación de este sistema integrado permite a los alumnos adquirir la capacidad de implementar correcta y eficientemente una estructura de datos con respecto a su especificación formal, haciendo uso en su diseño de esquemas algorítmicos adecuados. Como consecuencia, se consigue dotar a los alumnos de una metodología completa y profesional de desarrollo software de gran utilidad en la enseñaza actual de la Informática.

Durante el curso 2006/07 se ha realizado un estudio detallado de la aplicación en el aula del entorno educativo propuesto a través del Campus Virtual de la UCM, en la asignatura de *Estructura de Datos y de la Información* de segundo curso de la Ingeniería Informática. La Fig. 6 muestra la mejora de los resultados académicos (ver [2] para más detalles).

Las fases de aprendizaje aquí descritas pueden ser aplicadas en otras asignaturas de planes de estudios que incluyan el aprendizaje de estructuras de datos y algoritmos.

Referencias

[1] Tao Chen and Tarek Sobh. A tool for data structure visualization and user-defined algorithm animation. In *Frontiers in Education Conference*, 2001.

[2] C. Segura et al. Interactive Learning of Data Structures and Algorithmic Schemes. In *ICCS 2008: Advancing Science through Computation*. Springer, LNCS 5101, To appear, 2008.

[3] M. Clavel et al. *All about Maude. A High Performance Logical Framework*. Springer, LNCS 4350, 2006.

[4] N. Martí, Y. Ortega, and A. Verdejo. *Estructuras de datos y métodos algorítmicos. Ejercicios resueltos*. Colección Práctica, Pearson, Prentice Hall, 2003.

[5] R. Peña. *Diseño de programas. Formalismo y abstracción*. Prentice Hall, 2005.