Sistemas informáticos
Curso 2009-2010

Aprendizaje Interactivo de Estructuras de Datos: de la Especificación Algebraica a la Implementación

Luis Jiménez Paniagua

José Marcos Barrio

Rubén Fuentes Iglesias

Dirigido por Rafael del Vado Vírseda

Departamento de Sistemas Informáticos y Computación

FACULTAD DE INFORMATICA

UNIVERSIDAD COMPLUTENSE DE MADRID

Agradecimientos

Queremos mostrar nuestro agradecimiento a las siguientes personas, sin cuyo apoyo no hubiésemos logrado llevar a buen puerto este proyecto:

- *A nuestro profesor Rafael del Vado, por su inestimable ayuda y su humor, que hicieron mucho más llevadero este trabajo.*

- *A nuestros familiares, cuya comprensión ha resultado valiosísima en los muchos momentos difíciles que nos hemos encontrado.*

Página de autorización

Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Madrid, 2 de Julio de 2010

Fdo. Luis Jiménez Paniagua

Fdo. José Marcos Barrio

Fdo. Rubén Fuentes Iglesias

ÍNDICE

Resumen del proyecto

En castellano

Este es un proyecto fresco e innovador que representa el desarrollo de una herramienta educativa, incorporando una nueva estructura de datos, montículos, y adaptando ideas de otras herramientas, basándose en algunas ya existentes las cuales han sido desarrolladas en años pasados en la asignatura de Sistemas Informáticos, Vedya y Vedya-Test, a través de una sola herramienta capaz de controlar la evolución pedagógica del alumno en la asignatura de “Estructura de Datos y de la Información”.

El propósito de esta herramienta es ayudar al alumno a comprender la estructura de los montículos, a que comprenda su especificación a través de Maude y su implementación a través de eclipse.

En inglés

This is a fresh and innovative project that represents the development of an educational tool, adding a new data structure, heaps, and adapting ideas from other tools, based on some already existing ones that have been developed at Sistemas Informáticos some years before, called Vedya and Vedya-Test, through a unique tool capable of controlling the pedagogical evolution of the student when at “Estructuras de Datos y de la Informacion”.

The aim of this tool is helping students to understand the heap structure, its specification through Maude and its implementation through Eclipse.

Palabras Clave

- *Montículos*

En computación, un montículo (heap en inglés) es una estructura de datos del tipo árbol con información perteneciente a un conjunto ordenado. Los montículos máximos tienen la característica de que cada nodo padre tiene un valor mayor que el de todos sus nodos hijos, mientras que en los montículos mínimos, el valor del nodo padre es siempre menor al de sus nodos hijos.

- *Especificación*

En informática una especificación es una definición formal o semi-formal de un sistema informático ya sea un programa informático, un componente de software, una librería o cualquiera otro tipo de software.

- *Implementación*

Un lenguaje con reflexión proporciona un conjunto de características disponibles en tiempo de ejecución que, de otro modo, serían muy difícilmente realizables en un lenguaje de más bajo nivel. Algunas de estas características son las habilidades para:

- *Descubrir y modificar construcciones de código fuente (tales como bloques de código, clases, métodos, protocolos, etc.) como objetos de "categoría superior" en tiempo de ejecución.*
- *Convertir una cadena que corresponde al nombre simbólico de una clase o función en una referencia o invocación a esa clase o función.*
- *Evaluar una cadena como si fuera una sentencia de código fuente en tiempo de ejecución.*

- *Aprendizaje*

Es la adquisición de nuevos conocimientos, conductas, habilidades, valores o pensamientos, a partir de determinada información percibida.

- *Educación*

El proceso multidireccional mediante el cual se transmiten conocimientos, valores, costumbres y formas de actuar.

- *Conocimiento*

Es el conjunto organizado de datos e información que permiten resolver un determinado problema o tomar una decisión.

Estructura de datos

Es una colección de datos cuya organización se caracteriza por las funciones definidas utilizadas para almacenar y acceder a elementos individuales de datos. Las estructuras de datos pueden descomponerse en los elementos que la forman. La manera en que se colocan los elementos dentro de la estructura afectará la forma en que se realicen los accesos a cada elemento.

1. Nuevas tecnologías en educación

Con independencia de la filosofía, la psicología o cualquier otra consideración académica, es indudable que las nuevas tecnologías que progresivamente se han ido aplicando a la educación han estimulado algunas de las tendencias anteriormente discutidas. Por ejemplo, el advenimiento de los sistemas multimedia de alta fidelidad y de realidad virtual naturalmente llevan a sus entusiastas a argumentar sobre los beneficios del aprendizaje a través de la "inmersión en una situación" que es una variación de la perspectiva situacionista. Similarmente, la disponibilidad de redes de alta velocidad permite un grado de trabajo distribuido y en colaboración que previamente era inalcanzable, lo que conduce a discusiones acerca de las virtudes intrínsecas del "aprendizaje social" mediado por la tecnología.

Indudablemente, como también se ha señalado en múltiples ocasiones, la aplicación de los sistemas educacionales traerá consigo nuevos objetivos docentes y nuevos métodos de enseñanza. Habrá que plantearse:

- *Planes de estudio orientados a nuevos objetivos de enseñanza y aprendizaje.*
- *Nuevos métodos para poder evaluar a los alumnos en nuevos entornos de trabajo.*
- *Nuevos estándares en métodos de enseñanza.*
- *Cambios en la formación de los educadores.*
- *Cambios en los mecanismos de gestión académica que faciliten el tratamiento individualizado de las necesidades, entre las que se enmarcan las relacionadas con la accesibilidad.*
- *Reorientación de los currículos, introduciendo la adquisición de capacidades y no tanto la medición de conocimientos adquiridos.*
- *Nuevos métodos de seguimiento de la actividad docente que faciliten su gestión y ayuden a anticipar problemas de aprendizaje.*
- *Nuevos paradigmas de aprendizaje que supongan la integración efectiva de la formación en la actividad laboral.*

Desde una visión positiva de las potencialidades de Internet y del resto de las aplicaciones, como herramientas de cambio de la práctica educativa, este estudio, en la medida de lo posible, quiere ayudar a entender que la

incorporación de las tecnologías a la educación depende de muchos factores, entre los que resultan esenciales la formación y la actitud de los docentes, así como la voluntad de la comunidad educativa de perseguir una educación más flexible e integradora, más cercana al mundo exterior y más centrada en las individualidades del alumno.

1.1. Objetivos de los Sistemas de Educación

Los sistemas interactivos de enseñanza/aprendizaje surgen con el objetivo de servir de complemento a la enseñanza tradicional y mitigar así algunas de las carencias encontradas en el método tradicional. Tal es la importancia de este tipo de sistemas que su desarrollo constituye ya un área de investigación importante en la que trabajan investigadores de diversos campos, principalmente procedentes de las ciencias de la computación, psicología y educación. Como objetivos concretos se muestran las ventajas del uso de los sistemas interactivos de enseñanza/aprendizaje (SIEA) para dar respuestas a problemas planteados. El desarrollo de estos sistemas debe basarse tanto en la naturaleza del conocimiento que debe ser aprendido como en el conocimiento del propio proceso de aprendizaje.

Entre los objetivos de los sistemas de educación destacan los siguientes:

- *Fomentar la participación del alumno en el proceso de aprendizaje.*
- *Incrementar el flujo y la calidad de la información recibida.*
- *Incentivar el aprendizaje significativo y activo.*
- *Modificar las estructuras de pensamiento del alumno.*
- *Ayudar a descubrir las naturalezas de las materias.*
- *Fomentar el aprendizaje colaborativo*
- *etc.*

1.2. Panorama histórico

En el desarrollo de estos sistemas se consideran diversas etapas.

Se distingue la etapa inicial de los llamados CAI (del inglés Computer Aided Instruction). Los tradicionales programas CAI se centraban en representar la estructura de la materia a enseñar y en transmitir dicha estructura siguiendo métodos tradicionales de enseñanza. Estos programas pueden considerarse, como los descendientes evolutivos de los libros, ya que al igual que ellos, están organizados estáticamente de tal forma que contienen tanto el dominio de conocimiento como el conocimiento tutorial de los maestros, como expertos humanos.

En los libros se encuentran algunas ayudas, tales como: secciones, capítulos, índices, tablas y figuras; son las herramientas con las que se facilita la presentación de un tema, y los autores tienen conocimiento tanto en el dominio de lo que hay que comunicar como en la escritura del propio libro.

Un libro puede propiciar varios niveles de lectura, proporcionar referencias cruzadas entre secciones y suministrar glosarios adecuados, siempre y cuando el autor lo haya indicado en el texto. Un libro no puede responder a preguntas inesperadas del lector. Tampoco puede modificarse "al vuelo" el conocimiento presentado, de tal forma que se adapte a un lector específico. Las limitaciones propias del medio de impresión no permiten mucha flexibilidad y dinamismo en el acceso al conocimiento contenido en el libro.

Los autores de programas CAI hacen lo mismo: piensan con antelación en acciones educacionales, anticipan las circunstancias que requieren decisiones y escriben el código apropiado que permita capturar tales decisiones. Por consiguiente, los programas CAI tradicionales aprovechan la experiencia tutorial de los maestros expertos y directamente reflejan esta habilidad en el comportamiento de los programas. Esta circunstancia hace potente el enfoque de los sistemas CAI, pero, también es uno de sus principales defectos ya que pocos maestros pueden anticipar todos los errores de concepto que un estudiante puede llegar a adquirir. Además, es prácticamente imposible realizar programas de aplicaciones que contengan todas las decisiones imaginables. Aún cuando es verdad que los programas CAI, una vez que son desarrollados y probados pueden ser usados por un gran número de personas, también es cierto que son difíciles de modificar.

Las principales desventajas de los programas CAI se pueden resumir de la siguiente manera:

- *La calidad de los programas de aplicaciones está estrechamente relacionada, en muchos sistemas de tutorial dirigido, con la habilidad del autor para anticipar (e incluir en ellos) tantas respuestas del estudiante como sea posible y entonces poder especificar la trayectoria de tutorial que resulte más apropiada.*

- *Una estrategia de enseñanza, trasladada a líneas de código, no se ajusta a las necesidades específicas del estudiante.*

- *La autonomía que ofrecen los sistemas de tutorial menos dirigido representan un obstáculo para aquellos estudiantes que no tienen la posibilidad de aprovechar esta autonomía.*

- *En los sistemas CAI no hay una definición explícita del objetivo de la enseñanza, ni el sistema se comporta dinámicamente adaptando su funcionamiento a las características del usuario para alcanzar dicho objetivo.*

- *Los programas CAI, a menudo, son desventajosos en lo referente al costo, dado que desarrollar y mantener tales programas implica invertir recursos de considerable magnitud.*

A continuación surgen los ***Sistemas Tutoriales Inteligentes (STI)*** *(también aparecerán como ITS, del inglés Intelligent Tutoring Systems), considerados los primeros sistemas basados en la aplicación de técnicas del campo de la Inteligencia Artificial (IA) a la educación. Estos sistemas seguían incidiendo en la utilización de métodos tradicionales de enseñanza (en los que el alumno seguía presentando una actitud más o menos pasiva). El alumno se enfrenta al proceso de aprendizaje mediante una guía de acciones previamente prefijadas que se corresponden con un modelo de tutorización individual "uno a uno". No obstante, es de destacar que en estos sistemas tomaban un papel relevante dos elementos que desde entonces están presentes en la mayoría de las aplicaciones:*

- ***El Modelo del Estudiante.*** *Se utiliza para representar lo que el sistema supone que el estudiante ha aprendido.*

- ***El Modelo Pedagógico.*** *Contiene el conocimiento referido a la forma de gestionar el propio proceso de aprendizaje.*

El gran problema de estos sistemas es que estaban más centrados en el conocimiento que se quería transmitir que en el propio proceso de aprendizaje de dicho conocimiento. En cualquier caso, supusieron un avance considerable

al establecer la necesidad de introducir el conocimiento y de modelar el comportamiento.

*Con la intención de superar los problemas detectados en los STI surgieron los Entornos de **Aprendizaje Interactivos (EAI)** y los **Micromundos**. En este tipo de entornos, la inteligencia ya no está centralizada en un tutor, que actúa para garantizar que lo que aprende el alumno se corresponde con el modelo de un experto, en su lugar, se proporcionan diferentes herramientas que fomentan la investigación sin un riguroso control externo. En otras palabras, se realiza un planteamiento constructivista del aprendizaje frente a la formulación conductista de etapas anteriores. En estos sistemas predomina el uso de imágenes, vídeo y otras representaciones gráficas.*

Más adelante, además del desarrollo de sistemas mixtos que incorporan diversos paradigmas, se produce una paulatina división del campo en diversas áreas de interés: aprendizaje colaborativo, modelado del usuario, interfaces adaptativos, aprendizaje a través de la web, estandarización de contenidos y cursos, etc. Para poder integrar los diversos trabajos realizados en cada uno de estos campos, se plantea la necesidad de introducir mayor metodología y la de concretar formatos y estándares de control y gestión de este tipo de aplicaciones.

1.3. Métodos de Enseñanza/Aprendizaje

Actualmente, para poder proporcionar un comportamiento inteligente en los sistemas interactivos de E/A, el campo de la lA se plantea el diseño de estos sistemas desde la perspectiva de ofrecer contribuciones a la solución de los problemas educacionales ya puestos de manifiesto con el desarrollo de STIs y EAIs referentes a las siguientes cuestiones:

- *¿Cuál es la naturaleza del conocimiento?*
- *¿Cómo debe ser aprendido?*
- *¿Los tutores deben instruir, tutorizar, guiar o entrenar a los estudiantes?*
- *¿Cuáles son las medidas de efectividad?*

Se debe responder, por tanto, a las dos preguntas básicas: la naturaleza del conocimiento y la del propio proceso de aprendizaje.

1.3.1. Naturaleza del conocimiento

Se han planteado diversos debates y determinado multitud de posturas sobre la naturaleza del conocimiento en sucesivas etapas a lo largo de la historia.

Se puede mencionar la dicotomía planteada entre el constructivismo y el instruccionismo u objetivismo (por ejemplo, sistema SPENGELS).

El primero se usa en favor del estilo de pedagogía centrado en el estudiante ejemplificado en los EAls. De acuerdo con este punto de vista, el conocimiento debe ser construido por el aprendiz pieza por pieza, de modo que estos sistemas se ven corno las herramientas ideales para potenciar esta construcción auto-guiada. El proyecto más "extremo" en este sentido es LOGO. Aunque parecen más adecuados sistemas intermedios en los que el sistema no debería contener conocimiento correcto apriorístico pero debería intentar descubrirlo en un esfuerzo conjunto con el estudiante. El sistema PEOPLE-POWER ilustra este enfoque.

*El **situacionismo** comparte ciertos principios con el construccionismo pero pone un mayor énfasis en que el conocimiento construido no existe en la memoria sino que emerge de la interacción con el entorno. Frente a la ambivalencia entre constructivismo y situacionismo, en las secciones siguientes se muestran algunos de los nuevos métodos que recientemente han despertado el interés de los tecnólogos educacionales.*

*El **conexionismo** es otro punto de vista del conocimiento que se presenta en contraste con la versión de procesamiento de símbolos del objetivismo. Establece que el conocimiento está implícitamente representado en los pesos y enlaces entre un gran número de nodos modelados en redes neuronales.*

En definitiva, la naturaleza del conocimiento influye en la naturaleza del aprendizaje.

1.3.2. Paradigma de aprendizaje

El diseño de sistemas educacionales refleja una visión bastante ecléctica de la naturaleza del aprendizaje. Muchos tipos diferentes de circunstancias y actividades pueden conducir al aprendizaje y muchos de ellos se han soportado, hasta cierto punto, dentro de estos sistemas. Así, se distinguen los siguientes paradigmas de aprendizaje:

- *Aprendizaje basado en casos: La idea del aprendizaje basado en casos, derivada del campo de razonamiento basado en casos en Inteligencia Artificial, es que los estudiantes aprenden de historias (casos) si se les presentan en el punto preciso en que están interesados en conocer la información que la historia les trasmite. Así, en lugar de aprender reglas abstractas para aplicar a situaciones, los estudiantes explotan las analogías encontradas en casos similares para sintetizar sus propias reglas de decisión.*

- *Aprendizaje orientado a fallos: los sistemas como GREATERP son sistemas basados en la asunción de que el aprendizaje está orientado por fallos, esto es, que la ocurrencia de fallos proporciona la oportunidad de aprender. Este conocido tutor de Lisp está basado en ACT que es esencialmente una teoría objetivista que pone énfasis en el almacenamiento de esquemas en memoria.*

- *Aprendizaje a través de la experimentación: Un escenario estándar es un entorno para la resolución de problemas donde los estudiantes realizan experimentos guiados por el sistema en su interpretación. Por ejemplo, QUEST que proporciona una simulación gráfica de circuitos que permite a los estudiantes entender los principios que gobiernan su comportamiento realizando operaciones de localización de problemas (del inglés troubleshooting).*

- *Aprendizaje basado en simulación: Se basan en el éxito de los simuladores de vuelo. Estos dispositivos proporcionan un ilimitado y muy barato tiempo de práctica para perfeccionar habilidades que serían muy costosas, lentas y peligrosas de adquirir usando un avión real.*

- *Aprendizaje a través de diálogo: Se describe un entorno con el que los estudiantes intercambian argumentos durante un debate, mientras el sistema actúa como un árbitro, usando directrices de diálogo tomadas de la teoría de juegos para determinar la validez de los movimientos.*

- *Aprendizaje reflexivo: Se basa en la experiencia de la clarificación de ideas que puede derivarse de la discusión con compañeros de estudio que tampoco tienen una comprensión absoluta de la materia bajo estudio pero son capaces de formular preguntas que provocan una reestructuración del pensamiento.*

- *Aprendizaje como una actividad social: Hay propuestas de "diseño socio-tecnológico" que ponen énfasis en el diseño de sistemas en el entorno socio-tecnológico en que habrán de usarse.*

- *Aprendizaje visual: La visualización con frecuencia permite a los aprendices y profesionales obtener representaciones de un gran número de*

datos, para "generar intuiciones" y sugerir hipótesis para posteriores comprobaciones. En algunos casos abren completamente nuevos campos de estudio y definen nuevos objetivos curriculares al tiempo que proporcionan nuevos métodos de aprendizaje (por ejemplo, el campo de fractales).

- *Aprendizaje en colaboración: Las tecnologías para colaboración basadas en computadores proporcionan nuevos métodos cooperativos de trabajo y aprendizaje.*

2. Sistemas de Enseñanza / Aprendizaje

El desarrollo de un Sistema Interactivo de Enseñanza/Aprendizaje conlleva una serie de etapas en las que se definen y desarrollan los diferentes componentes que, de forma integrada, constituyen el sistema.

En primer lugar, se desea que se comprenda la funcionalidad y la estructura, con sus diferentes módulos, que definen los sistemas de E/A.

También se desea aclarar que una de las funcionalidades básicas del sistema debe ser la adaptación

Para alcanzar estos objetivos, en primer lugar, se analizan los componentes que constituyen la estructura básica de un SIEA. Se insiste en la importancia de la representación del conocimiento del dominio y en la esencia del funcionamiento de estos sistemas: el modelado del conocimiento de la instrucción, concretado en el modelo pedagógico, y el modelado del proceso de aprendizaje del alumno, asociado al resto de la información de interés en el correspondiente modelo del alumno.

Partiendo del concepto de memoria y aprendizaje, se continúa presentando una serie de paradigmas de desarrollo de aplicaciones asociadas a las distintas etapas que configuran el proceso natural de aprendizaje. Con ello se ofrece una visión global y práctica de los sistemas y del propio proceso.

Una vez se han aclarado los componentes básicos y se ha repasado la complejidad de escenarios de aprendizaje existentes, se propone el estudio detallado del modelado del estudiante.

Otro de los aspectos esenciales es la necesaria capacidad de adaptación que caracteriza estos sistemas.

Finalmente, se introduce la evaluación como una fase esencial en el desarrollo de SIEAs.

2.1. Estructura general de los sistemas de Enseñanza / Aprendizaje

Dentro de los objetivos de los sistemas de educación destaca:

- *Fomentar la participación del alumno en el proceso de aprendizaje.*
- *Incrementar el flujo y la calidad de la información recibida.*
- *Incentivar el aprendizaje significativo y activo.*
- *Modificar las estructuras de pensamiento del alumno.*
- *Ayudar a descubrir las naturalezas de las materias.*
- *Fomentar el aprendizaje colaborativo.*

2.1.1. Modelo pedagógico

Es una de las partes esenciales del sistema. En ella se materializa el conocimiento estratégico sobre el proceso de instrucción. La representación de este conocimiento puede ser reutilizada en diferentes dominios.

Aun siendo una parte esencial del sistema, muchas aplicaciones tienen una funcionalidad muy limitada. La razón es la dificultad de recoger con precisión dicho conocimiento. El objetivo sería crear una especie de sistema experto que fuera capaz de resolver cualquier tipo de eventualidad en el proceso de aprendizaje (por ejemplo, determinar y representar todos los errores que podría cometer un alumno y las acciones que el tutor puede realizar para remediar dichos errores). Esa postura, adoptada en la mayoría de los STIs, se transformó paulatinamente en propuestas centradas en entornos interactivos, donde el control sobre el proceso de aprendizaje se relajaba notablemente y se ponía mayor énfasis en la libertad de indagación por parte del alumno. Esto hacía perder en cierto modo la confianza en la validez de estos sistemas como mecanismos para la educación autónoma, con lo que se vio la necesidad de incluir algún modelo pedagógico, aunque fuera mínimo.

*En el modelo pedagógico se distinguen **decisiones globales**, como la secuencia de etapas en la instrucción y **decisiones locales**, como por ejemplo decidir si se debe o no interrumpir a un estudiante en la realización de una tarea.*

La rigidez o no del modelo pedagógico determinará el grado de control sobre las actividades realizadas. Así se pueden distinguir: control del sistema, control mixto (estudiante y sistema), descubrimiento guiado, asistencia, etc.

En contra de lo que opinan algunos, la idea de que la tecnología desplaza a los docentes está superada. El papel del profesor no solo no pierde importancia sino que se amplia y se hace imprescindible.

Según un estudio experimental que analiza las actitudes de los docentes opina que el uso de las TIC acabará generalizándose entre los profesores. Se supone que esta predisposición implica que comprenden la aportación de las TIC a la mejora de la enseñanza.

Esta idea está muy ligada a la necesidad que tiene el docente de formarse continuamente o de forma permanente, como única vía para poder enfrentarse a las repercusiones educativas de las innovaciones tecnológicas. En este sentido adquieren mucho valor para el profesor los nuevos canales de comunicación que le permiten relacionarse con colegas que pueden ser del mismo centro o incluso del extranjero, con los que puede compartir sus experiencias, sus problemas y sobre todo "estar al día".

La labor del profesor se hace más profesional, más creativa y exigente. Su trabajo le va a exigir más esfuerzo y dedicación.

El nuevo profesor debe crear un entorno favorable al aprendizaje, basado en el dialogo y la confianza. En este ambiente propicio, el docente debe actuar como un gestor del conocimiento y orientar el aprendizaje, tanto a nivel general de toda la clase, como a nivel individual de cada alumno.

La capacidad del profesor va a ser determinante a la hora de enseñar a los alumnos a aprovechar las ventajas de las nuevas herramientas. Sin embargo, y aunque las investigaciones sobre los efectos de las TIC en el aprendizaje no son homogéneas, se han comprobado algunas ventajas que, aunque de forma desigual, pueden favorecer el aprendizaje:

- *Aumento del interés por la materia estudiada.*
- *Mejora la capacidad para resolver problemas.*
- *Los alumnos aprenden a trabajar en grupo y a comunicar sus ideas.*
- *Los alumnos adquieren mayor confianza en sí mismos.*
- *Los alumnos incrementan su creatividad e imaginación.*

2.1.2. Modelo del alumno

El modelo del usuario es una parte fundamental en los sistemas que personalizan las respuestas que proporcionan a sus usuarios. Un modelo de usuario es una representación explícita de las propiedades de un usuario específico. Se utiliza para razonar acerca de las necesidades, preferencias o comportamiento futuro del usuario, así como para diagnosticar fallos en su interacción con el sistema.

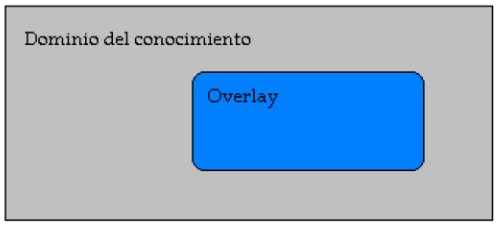
La construcción de este modelo es dinámica y en el caso de los sistemas para la educación debe cubrir todos los aspectos del alumno que puedan repercutir en su aprendizaje.

Aparte de la información menos volátil, como puedan ser datos personales, datos académicos, experiencias previas, etc. el gran problema que se aborda en la definición de este modelo es determinar cuál es el estado de conocimiento del alumno con respecto al conocimiento del dominio que se está aprendiendo.

La determinación del estado de conocimiento del alumno es un problema de diagnóstico. Se trata de establecer, a partir de las evidencias existentes (datos recogidos en la interacción; recordemos que cualquier interfaz es limitada), cuáles son los aciertos y los fallos existentes en dicho estado. Por ejemplo, se pueden codificar de antemano errores típicos que simplifiquen la identificación del problema. Por otro lado, la opción más utilizada es la de establecer una correspondencia entre la red de nodos que representaba el conocimiento aprendido por el alumno y la del conocimiento experto del dominio. Este modelo se denomina ***overlay*** *o* ***modelo superpuesto****. En este caso, el conocimiento del alumno se trata como un subconjunto del conocimiento del experto. Evidentemente, aunque este modelo se haya utilizado para aproximar la solución al problema, no refleja la mayoría de las situaciones de aprendizaje en las que el alumno puede tener conocimiento en cantidad y en calidad muy distinta a la del experto.*

En general, existen varios métodos para representar el conocimiento del alumno, algunos de los más importantes son:

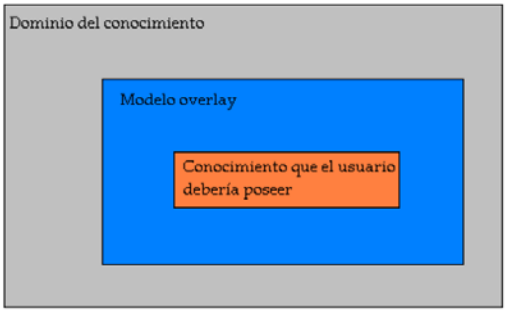
- *Modelo overlay. Aunque ya lo hemos comentado anteriormente, simplificando, los modelos overlay tratan el conocimiento del estudiante como un subconjunto del conocimiento de un experto (ver figura). Es el método más utilizado.*



Modelo overlay

Figura 2.1. Modelo overlay

- *Modelo diferencial. Este modelo es una modificación del overlay. Divide el conocimiento del estudiante en dos categorías (ver figura): conocimiento que el estudiante debería conocer y conocimiento que se supone que el alumno no conoce.*



Modelo diferencial

Figura 2.2. Modelo diferencial

- *Modelo de perturbación. En esta representación se supone que el alumno posee conocimiento potencialmente diferente en cantidad y calidad respecto al de un experto (ver figura). El modelo de perturbación también puede representar el conocimiento y creencias del estudiante más allá del rango del modelo del experto.*

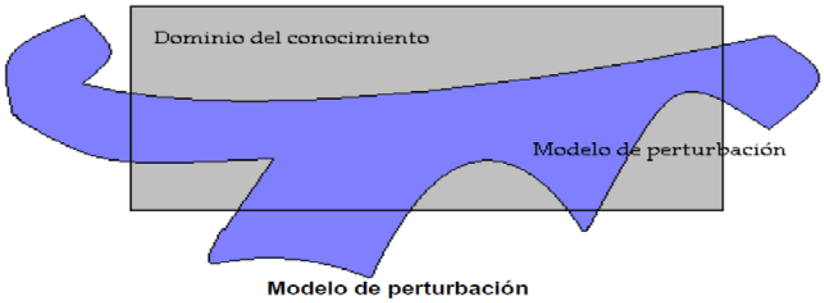


Figura 2.3. Modelo de perturbación

➢ *Modelo de estado vs proceso. Existen otros métodos para la representación del conocimiento del estudiante, los cuales utilizan deducciones para generar predicciones. Como ejemplo, se puede citar el modelo de estado vs proceso, el cual puede ser visto como la capacidad de simular el proceso por el cual un usuario elige consultar material. Con este tipo de modelos debería ser posible predecir el material que el usuario consultará posteriormente.*

2.1.3. Dominio de enseñanza

Es el conocimiento objeto del aprendizaje. Actúa como una fuente de la información que se le presenta al alumno. Por ejemplo, generando explicaciones y respuestas al estudiante. También se utiliza para medir el conocimiento aprendido por el alumno. En este caso hay que establecer un método de comparación entre ambos.

El conocimiento del dominio se puede organizar en una red de nodos en la que se representan conceptos, objetos, relaciones, hechos, reglas, condiciones, acciones, prerrequisitos, objetivos, etc. Este tipo de representación es la que utilizaban los llamados sistemas ASK propuestos por Roger Schank . Los sistemas ASK, son sistemas hipermedia que simulan las situaciones que se presentan en el caso en que un determinado usuario tuviera una conversación con un experto (o con un grupo de expertos). En esta conversación, el usuario se hace una serie de preguntas que el sistema contesta. Sin embargo, por lo general, los participantes en una conversación real influyen en el desarrollo de la misma. Esto ocurre también en este tipo de sistemas, donde el usuario influye en el flujo de la conversación seleccionando

ciertas preguntas, y el sistema influye con las respuestas que proporciona a dichas preguntas.

Este tipo de sistemas combinan numerosos tipos de material dependiendo del dominio en que se apliquen, de manera que se consiga simular mejor una conversación con el experto. En cualquier caso, el material siempre debe estar estructurado de forma que se pueda ir presentando al usuario diferentes opciones entre las que éste debe elegir. La experiencia con comunidades virtuales de enseñanza, nos muestra que el tutor, que en la mayoría de los casos coincide con el administrador de la comunidad, debe construir y gestionar dicha estructura de contenidos.

Los contenidos se presentan mediante una red de conceptos (nodos) prefijados por el tutor y que deben ser aprendidos por el alumno. Los arcos representan la transición de un concepto a otro en función de los conocimientos que el alumno va aprendiendo y sus propios intereses. Dichas transiciones se presentan en forma de preguntas o de opciones que el alumno va eligiendo. En cierto modo, este tipo de propuestas son una concreción de los llamados mapas conceptuales. Un Mapa Conceptual puede definirse como una herramienta de asociación, interrelación, discriminación, descripción y ejemplificación de contenidos, con un alto poder de visualización.

Independientemente de la representación que se elija en el modelado del dominio, un aspecto fundamental en este punto es el de plantear el dilema de cuánto conocimiento hay que representar. Existen diferentes posturas al respecto:

- ***Introducir todo el conocimiento del dominio.*** *Se justifica esta opción en la medida en que se suponga que el sistema y los alumnos tengan que saber resolver el mismo tipo de situaciones.*
- ***Introducir una única guía de estudio en el sistema,*** *de manera que el resto de contenidos se le proporciona al alumno por otro medio alternativo y el sistema irá guiando y aconsejando el estudio a lo largo de dicho material.*
- ***Incrementar el conocimiento del dominio*** *aplicando técnicas de aprendizaje automático (por ejemplo, mediante sistemas aprendices).*

No obstante, además de la elección de una u otra opción hay muchos otros problemas implicados. Por ejemplo, el hecho de que el conocimiento del dominio sea completo o no, no tiene nada que ver con que éste tenga una forma que sea legible para el estudiante.

Debe quedar claro que el conocimiento del dominio es algo dinámico que refleja decisiones, creencias, acciones, etc. Realizadas en el dominio que se está aprendiendo y desde luego no es una base de datos en la que se ha codificado una información conocida.

2.1.4. Aproximación a la adaptación

La esencia de cualquier sistema interactivo de E/A es la de poder adaptarse dinámicamente a las necesidades del alumno. Estos sistemas llevan a cabo una adaptación basada en el modelo del usuario que contiene el estado de conocimiento del estudiante así como sus preferencias y objetivos.

En el caso concreto de los STIs, el objetivo es usar el conocimiento del dominio proporcionado por un experto, el conocimiento que se tiene del alumno y el conocimiento del tutor para ofrecer una enseñanza personalizada y flexible.

Con la evolución de Internet y las posibilidades que ofrece, estos sistemas se han trasladado a Internet y constituyen los llamados sistemas tutoriales inteligentes basados en la Web.

Normalmente, todos estos sistemas se basan en una representación predefinida del conocimiento del dominio y del modelo de estudiante. A pesar de que este tipo de representación predefinida puede proporcionar cierta adaptación, ésta se restringe en su mayoría a un conjunto específico de reglas. Esto implica que la mayoría de situaciones que surgen a lo largo de una interacción con el estudiante deben conocerse de antemano. Como esto es lógicamente muy difícil de definir y actualizar, los sistemas que modifican dinámicamente tanto los modelos del alumno como los de contenidos de acuerdo a las interacciones del alumno, pueden conseguir una mejor adaptación.

2.2. Tipos y filosofías de enseñanza

Una vez descrita la estructura general de los SIEAs es importante destacar que de las múltiples opciones que se tienen, la elección de una u otra depende del enfoque que desde el punto de vista pedagógico se quiera dar al sistema.

2.2.1. Aprendizaje y memoria según Roger Schank

Según Schank la idea clave sobre la memoria está en recordar casos relacionados con la información que se está tratando y para ello nos basamos en la correspondencia en algún aspecto concreto entre la situación actual y la recordada. Para ello, establecemos una etiqueta que identifica dicho aspecto en nuestra memoria de casos. Recordemos que los casos son estructuras de conocimiento sobre situaciones experimentadas en el pasado. Mediante dichas etiquetas se organiza una red de experiencias o casos que se organizan de forma que reflejan precisamente lo que más nos interesa sobre dichas situaciones. Este proceso es dinámico y la reestructuración de dicha red es continua. De esta forma dicha red manifiesta la capacidad natural de aprender ante nuevas experiencias.

Por tanto, la importancia de la memoria en este esquema es que refleja lo que nos interesa de una situación, su significado establece la organización correspondiente. Esta memoria se estructura en una gran red de nodos donde se producen generalizaciones que capturan los aspectos comunes encontrados. Si una generalización se refuta con un caso nuevo contradictorio se construye la explicación del fallo y se recuerda especialmente dicha información para evitar nuevas contradicciones en el futuro.

2.2.2. Estructura de la memoria según Schank

El funcionamiento de la memoria depende de las estructuras mentales que sirven de base para su organización. Evidentemente, no se trata de recordar cada nueva situación sino de establecer una organización que recoja los aspectos relevantes.

Nuestra memoria contiene diferentes tipos de estructuras de organización que categorizan e interrelacionan otras estructuras de memoria. Estas estructuras de organización segmentan la memoria de forma que podemos localizar el elemento adecuado cuando es necesario.

Las estructuras de organización nos ayudan a clasificar y localizar estructuras de niveles inferiores como hechos o casos. A su vez recogen las generalizaciones que hacemos sobre dichos casos más específicos. La forma en que se organizan nos ayuda a hacer predicciones. Por ejemplo, sabemos

que cuando vamos a un examen y nos sentamos en nuestra mesa luego nos entregarán los enunciados correspondientes.

2.2.3. MOPS y Scriptlets

Se proponen dos tipos de estructuras. Por un lado, están los "scripts" o guiones, renombrados en como "scriplets" para resaltar su ámbito limitado. Esto es, recogen lo que sabemos sobre cómo suelen ocurrir los hechos en situaciones típicas. Forman una pequeña parte de una escena o situación, algunos ejemplos son: mirar un menú, lavarse los dientes, aparcar un coche, etc.

Ya que los "scriplets" sólo constituyen una pequeña parte de la experiencia, necesitamos otro tipo de estructuras superiores denominadas MOPS, paquetes de organización de memoria (del inglés memory organization paquets). Estas estructuras dividen las situaciones en escenas. Un MOP para un restaurante contendría escenas como "sentarse", "pedir", "pagar", etc. Dichas escenas apuntan a su vez los "scriplets" que contienen el conocimiento sobre cómo comprendemos y nos desenvolvemos en dichas situaciones.

Esta división de la memoria en dos tipos de estructuras nos permite utilizar lo que hemos aprendido en una tarea en otra distinta. Por ejemplo, podernos "pagar un servicio en un restaurante", "un autobús", "un cine", etc. Sobre dichos "scriplets" los MOPS establecen los matices que distinguen el pago en cada una de las mencionadas situaciones.

Finalmente, resaltar que el motor de esta memoria de casos son las preguntas. Cuando nos enfrentarnos a una situación nueva recordarnos las relacionadas y nos preguntarnos cuáles son las diferencias y los aspectos comunes. Cuando una predicción nos falla nos preguntamos qué causó el fallo y cómo podemos evitarlo en el futuro. Otras veces, cuando nos enfrentamos a un nuevo problema necesitamos elaborar un plan. Es entonces cuando nos preguntamos si ya conocernos un problema relacionado o si podemos descomponerlo en problemas más sencillos. Aunque haya distinto tipo de preguntas todas comparten una esencia común: juegan un papel esencial en el aprendizaje. Apuntan a huecos en nuestras estructuras de memoria que se intentan cubrir.

2.3. Uso y evaluación de los sistemas de educación

La clave para realizar una evaluación apropiada es un buen diseño y desarrollo de experimentos de forma que los factores individuales que se quieran testear puedan separarse fácilmente de otros factores que puedan interferir.

2.3.1. Evaluación de alumnos

La evaluación de los alumnos ha sido siempre una parte importante del sistema de enseñanza y aprendizaje. Por una parte, se necesita saber cuál es el conocimiento adquirido para actuar en consecuencia y, por otra, los propios alumnos necesitan también conocer de manera no subjetiva el conocimiento que han adquirido.

Uno de los mecanismos de evaluación más extendidos, por su facilidad de corrección, es la realización de pruebas de evaluación mediante tests. La realización de éstos tiene la ventaja de sistematizar la evaluación, por lo que ha sido ampliamente usada en aplicaciones de enseñanza asistida por ordenador y en sistemas tutoriales inteligentes.

En los sistemas tradicionales los tests tienen una serie de inconvenientes: las preguntas son las mismas para todos los alumnos, su número es fijo, el tipo de preguntas es muy limitado, etc. Por otra parte, la evaluación que se obtiene tras la realización de un test suele estar basada en el número de respuestas acertadas y no suele considerar la variabilidad en la dificultad de las preguntas, ni otros factores como la probabilidad de acertar una pregunta al azar.

Un intento de solución a dichos problemas son los test adaptativos. Un ejemplo interesante es el sistema SIETTE (Sistema Inteligente de Evaluación mediante Test para TeleEducación). Basándose en la cuantificación probabilista de los resultados de los tests y los posibles errores debidos al azar, junto con una medida de la dificultad de la pregunta, el sistema realiza una generación adaptativa de las preguntas que deberá ir respondiendo el alumno. Para ello se mantiene un registro temporal de la evolución del alumno en el test, que se tiene en cuenta en el proceso de selección de la siguiente pregunta.

Aunque estas investigaciones sean de interés, debemos recordar, que la evaluación basada en la realización de un test condiciona muchas veces el

propio proceso educativo, dado que se puede caer en "enseñar para el test". Entonces se cae en preguntar y centrar el proceso educativo en aquello que pueda ser preguntado objetivamente en un test.

2.3.2. Evaluación del sistema

El término de evaluación empírica (del inglés empirical evaluation) de sistemas se refiere a determinar cómo se comporta un determinado sistema en ciertos experimentos. Por lo general, la evaluación de los SIEAs debería ir encaminada a verificar que el proceso de aprendizaje del alumno se enriquece cuando se utiliza el sistema.

Por otra parte, los sistemas educativos adaptativos se evalúan comparando los resultados que se obtienen aplicando o no adaptación (en base al uso o no de modelos del alumno). Por ejemplo, se puede comparar las acciones previstas para el alumno con las acciones que realmente se realizan. También se puede calcular el porcentaje de errores reconocidos.

Otra cuestión importante en la evaluación del sistema es considerar las características de la persona que se está evaluando. En estos casos los sistemas de educación suelen presentar distintos tipos de material dependiendo del estilo de aprendizaje del alumno.

Una evaluación completa sería aquella que comprendiera la validación tanto de los distintos componentes de un sistema, como por ejemplo, el uso de una determinada técnica de aprendizaje automático para el modelado del usuario, como la validación de todos los componentes en su conjunto. Un ejemplo de esta evaluación sería el sistema ADVISOR. ADVISOR es una arquitectura de aprendizaje compuesta por dos agentes cuyo objetivo es el de centrar el razonamiento de un sistema tutorial inteligente en un sólo módulo. Uno de los agentes es responsable del aprendizaje de un modelo de cómo se comportan los alumnos usando el tutor en distintos contextos. El otro agente coge este modelo del comportamiento del alumno y un objetivo especificando el objetivo educacional que se persigue.

En este proyecto, primero se evaluaron (según los métodos propios de evaluación característicos de los sistemas de aprendizaje automático) los modelos que se construyen describiendo el comportamiento de los alumnos. Una vez validados los componentes que constituyen el sistema ADVISOR, como suele hacerse, se dividieron a los alumnos que participaron en el experimento en dos grupos, uno utilizaba el tutorial sin ADVISOR y otro con él.

3. Sistemas de Tutorización Inteligente

Las primeras aplicaciones de IA en educación desarrollaron STIs. En la mayoría de los casos, como en los precedentes sistemas Cal (del inglés Computer Aided Instruction), se intentaron implementar métodos tradicionales de aprendizaje y enseñanza. La técnica pedagógica de ejercicio y práctica y otras variantes donde los estudiantes resuelven problemas relativamente cortos propuestos por el profesor, tienen un probado éxito en las clases.

3.1. Fundamentos

Los STIs generalmente persiguen objetivos de aprendizaje bien definidos y comúnmente aceptados, como conocimiento factual y habilidades procedurales, reflejo de una perspectiva objetivista del conocimiento y que pueden medirse mediante tests estandarizados. Centrándose en el conocimiento a aprender, los diseñadores de estos sistemas con frecuencia empiezan por especificar este conocimiento tan precisamente como sea posible. Para lo que adoptan la mencionada perspectiva objetivista, que establece que el mundo puede estructurarse de modo completo y correcto en términos de entidades, propiedades y relaciones y que el pensamiento racional consiste en la manipulación de símbolos abstractos vistos como representantes de la realidad. Para conseguir esto aplican las distintas técnicas de representación de conocimiento de la IA (sistemas de producción, marcos, redes semánticas, lógica de predicados, etc.). Con este tipo de desarrollos se ha tratado de mostrar, utilizando métodos tradicionales de aprendizaje, enseñanza y evaluación, que mejoran significativamente la velocidad y calidad del aprendizaje de los alumnos y hasta cierto punto han obtenido éxito en sus pretensiones.

Los STIs intentan capturar un método de enseñanza y aprendizaje ejemplificado por una interacción humana de tutoría uno a uno. Para los investigadores de IA este método de enseñanza ha sido de forma natural el primer objetivo de aplicación. Las versiones ejercicio-y-práctica de la tutoría personalizada son formas de comunicación de conocimiento relativamente bien comprendidas. Este método es ampliamente aceptado tanto por la comunidad educacional como por nuestra cultura natural. Su popularidad se basa en buenas razones. La tutoría uno a uno permite un aprendizaje altamente individualizado y conduce consistentemente a mejores resultados que otros

métodos. Sus desventajas frente a otros métodos analizados se atribuyen principalmente a la inadecuación de las técnicas de evaluación de los resultados del aprendizaje. No obstante, los mencionados problemas de evaluación, la tutorización individualizada siguen formando parte del "estándar de oro" del aprendizaje.

El desarrollo de las ciencias de la computación está generando un abaratamiento en la maquinaria y los programas informáticos. Este avance ha afectado al ámbito educativo produciendo una autentica revolución en los procedimientos clásicos de enseñanza-aprendizaje. Hoy en día existen diferentes sistemas que permiten la tutorización y el aprendizaje casi de cualquier materia académica. La plataforma WebCT, donde se puede gestionar todo un curso a través de Internet, es un claro ejemplo de ello. Sin embargo, este tipo de sistemas tienen algunas limitaciones si los comparamos con lo que haría una persona en una tutoría. El problema es que estos sistemas son una mera plataforma comunicativa que establece un proceso interactivo entre el profesorado y el alumnado. Así pues, aunque plataformas como WebCT permiten que el personal docente pueda poner información y materiales a disposición de sus estudiantes, el sistema no hace nada automáticamente para determinar qué información o materiales serían más apropiados para cada persona en función del nivel de conocimientos que cada uno tiene.

El área de la Inteligencia Artificial en el ámbito de los sistemas expertos ha desarrollado la idea de lo que se conoce como Sistemas de Tutorización Inteligente (STI). Según Xiang (2002), un STI es un tipo de agente inteligente caracterizado por su habilidad para tomar decisiones y la ejecución de acciones sin la intervención de ninguna persona que monitoriza el funcionamiento del sistema. Así pues, un STI tendría que, en primer lugar, evaluar el conocimiento que el aprendiz tiene sobre la materia. A continuación, ha de presentar información acorde a ese nivel (por ejemplo, definiciones, material gráfico o auditivo). Y posteriormente ha de evaluar si la persona ha asimilado el conocimiento nuevo que se pretendía que el usuario hubiese adquirido. En otras palabras, el sistema tiene que generar una representación del conocimiento que tiene el usuario y guiarlo a través de un proceso de enseñanza que consiga que adquiera algún conocimiento o competencia.

Los STI comenzaron a desarrollarse en los años setenta y experimentaron su máximo apogeo a mediados de los años ochenta coincidiendo con el auge de los sistemas expertos, aunque se popularizaron durante la década de 1990. Durante la década de 1970 se pusieron en marcha numerosas investigaciones encaminadas al desarrollo de programas informáticos orientados a la enseñanza bajo el emblema lo que podríamos denominar como Instrucción Facilitada por Computadora (IFC, del inglés Computer Aided Instruction o CAI),

Aprendizaje Asistido por Computadora (AAC, del inglés Computer Assisted Learning o CAL) o Entrenamiento Basado en Ordenador (EBC, del inglés Computer Based Training o CBT). Como señala Wu (1993), el inicio del desarrollo de los primeros STI estuvo marcado por el intento de simular el proceso físico de interacción que se establece entre la persona que tutoriza y la que aprende. En concreto, se pretendía simular el proceso comunicativo que se produce en el tutor y el aprendiz en entornos naturales.

Podríamos ubicar el origen de los sistemas informáticos inteligentes orientados a la enseñanza en el Instituto Tecnológico de Massachussets (MIT). En este sentido, SCHOLAR podría considerarse como el primer STI que, a su vez, estuvo basado una aplicación conocida como ELIZA diseñada para el estudio de la comunicación entre persona-máquina basándose en la Terapia Rogeriana. SCHOLAR se diseñó para enseñar geografía del Continente Americano y supuso un salto cualitativamente sustancial entre los sistemas clásicos Orientados a Marcos Apriorísticos (ad hoc-frame-oriented systems o AFO), donde el proceso de enseñanza se basaba en series de preguntas y respuestas pre-diseñadas que hacían el sistema demasiado rígido y que constreñían la creatividad de la persona que aprende; y lo que se denominó como sistemas Orientados a Estructuras de Información (information – structure-oriented systems o ISO), donde el formato de representación de la información permitía al sistema anticipar las respuestas de la persona que aprende y adaptarse a sus necesidades (Carbonell, 1970).

El Sistema WHY es otro ejemplo de la evolución del los tutores inteligentes al auspicio del MIT. WHY se diseñó para enseñar a interpretar los fenómenos lluviosos en términos causales. El sistema se concibió para hacer reflexionar a los usuarios del sistema en términos socráticos; esto es, desafiaba al educando con preguntas que minaban creencias erróneas reconduciéndolo a la construcción del conocimiento por medio de ejemplos contrafactuales y preguntando por causas generales e intermedias.

Esta idea mecanizada de enseñanza fue introducida alrededor de 1958 por el psicólogo conductista Skinner en el concepto de enseñanza programada. La idea subyacente a la enseñanza programada estuvo condicionada por la filosofía conductista-asociacionista y su objetivo era dirigir el aprendizaje humano bajo condiciones controladas. Los investigadores educativos de la época trataron de implementar estas ideas en un contexto aplicado y se desarrollaron lo que se denominó libros de texto programados. Un libro programado presenta una materia tan estructurada que permite que el usuario vaya asimilando conceptos secuencial y autorreguladamente. Por ejemplo, Dixon (1964) diseñó un libro tras varios años de investigaciones orientado a enseñar los principios de la probabilidad en cursos universitarios de ingeniería.

En una de sus secciones se presentaban preguntas (problemas) en formato de elección múltiple con cuatro alternativas de respuesta. Cada respuesta remitía a otra página y sección donde daba retroalimentación al usuario sobre la idoneidad de su respuesta. Otro formato usado en los libros programados era dividir cada página en dos secciones, una (generalmente la de la derecha) para plantear conceptos o preguntas y la otra para mostrar las respuestas. El usuario tenía que leer las cuestiones mientras mantenía ocultas las respuestas. A continuación, tenía que comparar sus respuestas con las que daba el libro.

Durante la época de máximo esplendor del conductismo, las ciencias de la computación estaban en una etapa primigenia de su desarrollo por lo que el intento de implementar la idea de la enseñanza programada en un dispositivo automático pasó desapercibida. Lo que vino a denominarse como la revolución cognitiva en Psicología relegó a la filosofía conductista, junto a la idea de la formación programada, a un segundo plano como objeto de investigación científica. Hoy día, las computadoras son extremadamente más potentes que entonces, lo que podría abrir paso a una nueva revolución en el campo de la enseñanza programada.

Un STI es un sistema experto; esto es, un mecanismo informático que simula lo que haría un experto humano en un área particular de especialización. En el área educativa, el sistema experto tendría que recoger información sobre el usuario, evaluar el grado de conocimiento que tiene sobre la materia y llevar a cabo un proceso de toma de decisiones sin intervención humana acerca del suministro de ayuda que habría que proporcionar para que la asimilación del contenido fuese óptima.

No obstante, el desarrollo de sistemas expertos orientados a la enseñanza no ha gozado de una gran atención frente a otros campos como los negocios, la industria, la medicina, las aplicaciones militares o la investigación espacial. Por otra parte, los sistemas expertos más utilizados en el desarrollo de este tipo de herramientas han sido los sistemas basados en reglas clásicos. Sin embargo, los sistemas clásicos basados en reglas han sido criticados porque no tienen en cuenta eficientemente la naturaleza incierta de la realidad y porque tienden a generar conclusiones falaces.

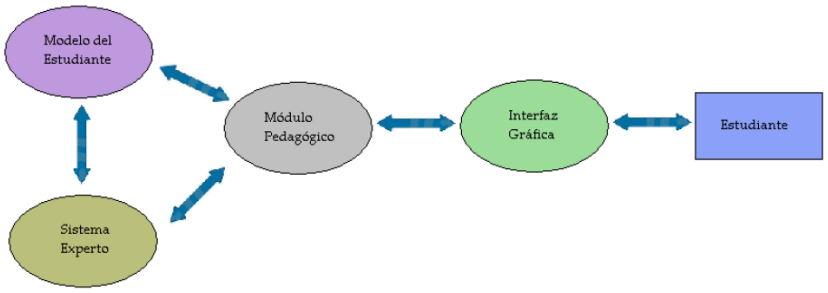
La investigación en Inteligencia Artificial ha desarrollado un nuevo tipo de herramientas que resuelven algunos de los problemas detectados en los sistemas expertos tradicionales. Las redes bayesianas son una generalización de los sistemas expertos clásicos que permiten modelar la realidad tanto cuantitativamente como cualitativamente. Gracias a su habilidad para representar los estados de un sistema de incertidumbres relacionadas, la

utilización de redes bayesianas en el diseño de STI se ha mostrado fructífera y supone un gran avance en el campo de la enseñanza programada.

Con lo expuesto anteriormente el objetivo de esta sección es; en primer lugar, presentar algunas nociones básicas sobre las redes bayesianas. En concreto, se tratará su habilidad para representar gráficamente la estructura cualitativa de la realidad entendida como un conjunto de dependencias condicionales; por otro lado, trataremos el modo en que representan la incertidumbre asociada al modelo por medio de funciones probabilísticas. A continuación, describiremos el funcionamiento de algunos STI basados en redes bayesianas. Por último, propondremos algunos aspectos que consideramos importantes ante el desarrollo de STI con base en los avances recientes en el campo de la psicología del aprendizaje y de la medición psicológica.

3.2. Componentes básicos

Aparentemente, estos sistemas difieren poco de los sistemas Cal que les preceden. En general, ambos se caracterizan por una filosofía común que incluye un gran control del tutor y un formato de tarea de respuesta corta. En ambos sistemas los estudiantes aprenden trabajando en series de cuestiones relativamente breves y en ambos casos el sistema juega exclusivamente el papel de experto en la tarea, controlando la selección de tareas o problemas, mientras que el estudiante es el responsable de resolverlos. El sistema también juega el papel de crítico, y en la mayoría de los STIs es más el sistema que el estudiante quien decide cuándo debe proporcionarse una realimentación crítica. Las diferencias principales entre los STIs y los primeros CAIs no reflejan diferencias en métodos de enseñanza ni filosofías de aprendizaje subyacentes sino logros de la ingeniería y la psicología que permiten al STI tutorizar de un modo "orientado al conocimiento". Al contrario que los primeros sistemas Cal, representan al menos parcialmente el conocimiento y razonamiento de un buen tutor humano individualizado y, por tanto, pueden adiestrar de una forma más detallada que aquellos.



Componentes básicos de un STI

Figura 3.1. Componentes básicos de un STI

El núcleo de un STI es un sistema experto que incluye suficiente conocimiento sobre un área particular para proporcionar respuestas ideales a preguntas y corregir no sólo un resultado final sino cada pequeña etapa de razonamiento intermedia. Esto permite mostrar y modelar una forma correcta de resolver un problema. Con frecuencia, como un tutor humano, puede generar muchos caminos de respuestas diferentes. Los componentes básicos de un STI son (éstos se describirán más en profundidad en los siguientes subapartados):

- *Sistema experto o modelo del conocimiento del dominio: Contiene una representación del conocimiento específico del área de enseñanza en cuestión. El conocimiento del dominio recoge la experiencia operativa de resolver problemas en dicho dominio.*
- *Modelo del estudiante: En los STIs el conocimiento del estudiante se pone en relación con el conocimiento del dominio. Un primer mecanismo muy sencillo e intuitivo es considerar que el conocimiento del alumno se va expandiendo hasta cubrir el conocimiento del experto. Para ello, se establecen medidas de comparación diferentes, según sea el tipo de conocimiento, entre cada uno de los elementos de ambos modelos, el del estudiante y el del dominio. A este modelo se le denomina modelo overlay o modelo superpuesto.*
- *Módulo pedagógico: Contiene reglas similares a las del sistema experto que codifican la experiencia acerca de la propia tutorización, relativa, por ejemplo, a cuándo interrumpir a los estudiantes y qué tipo de información proporcionarles.*
- *Interfaz con el alumno: Es el módulo que sirve de intermediario entre el sistema y el estudiante. Es importante que el interfaz sea amigable y fácil de usar, para que no suponga un escollo añadido en el proceso de aprendizaje del alumno.*

El modo en que entran en acción cada uno de los elementos descritos es el siguiente. El STI, mediante la interfaz gráfica, transmite al alumno los conocimientos representados en el modelo de conocimientos del dominio, siguiendo las directrices pedagógicas especificadas en el módulo pedagógico. Según el alumno va interactuando con el sistema se va actualizando el modelo del estudiante, modificando el grado de conocimientos del alumno, módulos estudiados, etc.

Cada vez que el estudiante comete un error, el STI diagnostica el problema, actualizando el modelo del estudiante, y a continuación intenta remediarlo con un consejo muy detallado acerca de cómo el sistema experto habría operado en esta etapa. Este proceso se repite a cada paso en la evolución hacia la solución completa de un problema. Para que este esquema de funcionamiento sea factible se crea un marco de interacciones entre los distintos módulos. El control lo ejerce el módulo pedagógico en función del conocimiento estratégico y operativo existente. Para ello, interpreta el modelo del estudiante diagnosticado y actúa conforme a una determinada estrategia de instrucción para alcanzar los objetivos declarados en el conocimiento del dominio.

En relación al proceso de diagnóstico de los errores cometidos por el alumno, éste constituye realmente el proceso de modelado del estudiante.

3.3. Entorno de Aprendizaje Interactivo

Estos sistemas surgieron como respuesta a las limitaciones del enfoque adoptado en muchos STIs en la etapa previa. Se trataba de superar algunos de los problemas detectados, como el control excesivo de la acción tutorial ejercida y la gran dependencia del conocimiento del dominio. La teoría de aprendizaje utilizada era el constructivismo. El método de aprendizaje se denomina basado en indagación, también descrito como centrado en el estudiante o construccionista y basado en descubrimientos. El constructivismo pone énfasis en los procesos de estructuración activa del mundo y sostiene que existen múltiples significados o perspectivas para cualquier evento o concepto, en lugar de existir un único significado correcto hacia el cual debe guiarse al estudiante. Todo ello conduce a diseños de sistemas que difieren substancialmente de los STIs. La inteligencia de los EAIs se distribuye entre un conjunto de herramientas en lugar de centralizarse en el tutor. Estas herramientas de computación con frecuencia incluyen video interactivo u otras representaciones gráficas, y permiten a los estudiantes investigar y aprender

libremente, sin un control externo. Esta libertad aporta asimismo beneficios prácticos ya que los EAls no son tan intensivos en conocimiento como los STIs. Los EAls proporcionan una representación explícita de los temas que el estudiante debe investigar pero no necesariamente "conocen todas las respuestas correctas", ni deben incluir modelos de la cognición del estudiante ni deben tomar complejas decisiones pedagógicas. Por otra parte, estos sistemas proporcionan quizá herramientas demasiado potentes, que sobrevaloran la capacidad del alumno para descubrir ideas interesantes o juzgar qué tipo de conocimiento deben construir y, en muchos casos, pueden conducirle a naufragar por un mar de cuestiones sin ningún interés.

Los Micromundos son un tipo particular de EAls que suponen una transición del tutor al concepto de "herramientas educacionales" y del método ejercicio-y-práctica al método de aprendizaje basado en la indagación. También suponen un cambio en los objetivos del aprendizaje. En primer lugar, siguen considerando importante el aprendizaje de conocimiento específico del área, más precisamente, el aprendizaje de la caracterización de patrones de relaciones entre los objetos y propiedades que definen el mundo. Por ejemplo, en el sistema SMITHTOWN, los estudiantes deben aprender acerca de leyes de oferta y demanda, estudiando cambios en los costes de los productos y considerando otros factores que influyan en la oferta o la demanda. En segundo lugar, implícita o explícitamente, muchos Micromundos alientan al estudiante a adquirir por sí mismos habilidades de indagación. Se trata de habilidades genéricas que los estudiantes deben conocer para conducir sus investigaciones virtualmente en cualquier tema. No existe una tarea de análisis riguroso de la indagación comparable a otras habilidades mejor definidas, como integrar o resolver una ecuación cuadrática.

3.4. Sistemas adaptativos

A continuación se mencionarán distintas áreas de desarrollo relacionadas con los SIEAs adaptativos en la web: los sistemas de hipermedia adaptativa, las interfaces adaptativas que utilizan técnicas de aprendizaje automático y, como un caso específico de éstas, los sistemas aprendices. La capacidad de adaptación de estos sistemas, al igual que los STIs, se basa en el mantenimiento de un modelo del usuario que recoja tanto sus gustos, preferencias y datos personales de relevancia, como su evolución en el uso de los distintos recursos disponibles en el web educativo. Los problemas, por tanto, son parecidos, pero las herramientas y los condicionantes del dominio

(multitud de fuentes de información y de canales de comunicación disponibles) permiten aplicar con mucho mayor éxito otro tipo de técnicas, como son las utilizadas en el campo del aprendizaje automático.

La proliferación del uso de Internet en la educación impone algunos cambios lógicos, tanto en el modelo educativo como en los requerimientos que deben cubrir los sistemas que soportan los cursos en la Red.

A pesar de las ventajas evidentes que ofrece Internet, encontramos dos grandes dificultades. En primer lugar cada alumno tiene unas necesidades especiales. En segundo lugar, el carácter estático de los sitios web educativos no permite cubrir de forma adecuada los requisitos cambiantes de los alumnos con necesidades, gustos y preferencias muy diversas.

La solución a este tipo de problemas son los sistemas adaptativos, un área de desarrollo que puede considerarse un caso de estudio dentro de un problema más genérico, la personalización de software.

Una vez estudiado el marco general del problema que se intenta resolver, antes de pasar a analizar los diferentes tipos de problemas relacionados con los sistemas de educación a través de la Web, se presentan, de forma genérica, las tareas y técnicas de adaptación existentes.

En los diferentes tipos de tareas y técnicas hay un denominador común. En todos los casos el sistema utiliza un modelo del usuario que recoge las características más relevantes de éste que permiten realizar la consiguiente adaptación de los recursos utilizados. Se estudia, por tanto, el modelado del usuario.

Una verdadera adaptación sólo tiene lugar si el sistema considera que los gustos, experiencias y necesidades del usuario varían con el tiempo. Esto es especialmente significativo en los sistemas de educación, en los que la curva de aprendizaje condiciona fuertemente los cambios que deben producirse para mantener una buena adaptación. Para resolver este problema se introducen las técnicas de aprendizaje automático para el modelado del usuario.

Una vez descrito el marco general y los elementos necesarios se pasa a caracterizar dos tipos de problemas. El primero, más genérico en su definición, son las interfaces de usuario adaptativas. En estos sistemas se presupone el uso de técnicas de aprendizaje automático. En segundo término se analizan los tipos de problemas, los métodos y las técnicas que definen el campo de la hipermedia adaptativa. En especial, se estudiarán los sistemas de educación adaptativos que utilizan dichas técnicas.

3.4.1. Personalización del software

Los procesos que guían el aprendizaje y los nuevos modelos de enseñanza pueden beneficiarse del uso intensivo de los recursos ofrecidos por Internet. La variedad de información y servicios ofertados y, sobre todo, los canales de comunicación alternativos que pueden establecerse entre los distintos protagonistas, producirán cambios considerables en los modelos de enseñanza aplicados, especialmente en el modelo de enseñanza a distancia. Frente a estas ventajas, se observa que la propia variedad y dispersión de las fuentes y servicios disponibles en la web educativa, unida a la naturaleza dispar del alumnado (más acusada en la enseñanza a distancia) dificultan el aprovechamiento de este medio. Para paliar estos problemas y otros relacionados se aconseja el desarrollo de sistemas en la Web que faciliten un acceso personalizado a dichos recursos.

La personalización de software define un marco más general de problemas en el que la esencia la determina la capacidad de identificar los gustos, necesidades, preferencias, problemas, aptitudes, limitaciones, etc. relativos al usuario del software con el fin de satisfacer dichas características.

La medida del éxito de estos sistemas está en la satisfacción del usuario. Por ello, evidentemente, se trata realmente de aprender todas esas cuestiones sin que por ello el usuario se vea obligado a declararlas explícitamente ni a cambiar continuamente lo que declaró en un momento dado.

Algunas de las motivaciones que podrían señalarse para el desarrollo de este tipo de sistemas son:

- *Una de las tendencias que más claramente definen el mercado en la actualidad (coches, ropa, canales de televisión, contratos telefónicos, etc.) es la oferta de productos personalizados a un bajo coste. Parece contradictorio que los programas capaces de adaptarse a las necesidades y preferencias del usuario sean prácticamente inexistentes.*
- *Dada la velocidad con que surgen los nuevos entornos de trabajo y sus respectivas versiones cabría preguntarse si no sería mejor intentar desarrollar programas que se adaptaran a nuestros gustos y necesidades, antes que obligarnos a adaptarnos continuamente a sus exigencias.*
- *Debido a la universalidad, accesibilidad y las innumerables ventajas para transmitir de una forma rápida y eficiente cualquier tipo de contenidos y, sobre todo, los servicios de comunicación alternativos ofertados en Internet, el*

uso de la red está introduciéndose en todo tipo de actividades. Especialmente en la educación a distancia. Sin embargo, el usuario de estos servicios se siente abrumado por la cantidad de información disponible. Puede tener problemas en encontrar la información deseada, en contactar con las personas que realmente le interesan, en no recibir cierto tipo de correos, en ser avisado sólo de las cuestiones que él considera relevantes. En definitiva, en obtener servicios personalizados.

- *Desde el punto de vista científico, una de las cualidades más definitorias de la inteligencia es la capacidad de aprender. Por tanto, ¿hasta qué punto podemos decir que un programa es "inteligente" si no es capaz de adaptarse a las necesidades del usuario que lo está utilizando?.*

- *El éxito de la telefonía móvil y los dispositivos electrónicos portátiles de tamaño reducido requiere especialmente el diseño de interfaces que se adapten a las necesidades de cada usuario y no le hagan perder el tiempo o hagan poco menos que inmanejable un sistema genérico con demasiadas opciones o una navegación tediosa para llegar a la información deseada. Podrían citarse otras muchas motivaciones, pero parece evidente que esta área de desarrollo de software está teniendo un interés creciente.*

3.4.2. Modelado automático del usuario

Si el objetivo de los sistemas es adaptarse al usuario en función de su comportamiento, parece natural aplicar las técnicas de aprendizaje automático para recoger los datos referidos a dicho comportamiento y así predecir su forma de actuar en el futuro. Sin embargo, para llevar a efecto este planteamiento hay que considerar una serie de problemas que comentaremos enseguida.

Los modelos del usuario guardan información sobre: intereses del usuario (a corto o a medio plazo), planes de los usuario e intenciones a corto plazo, conocimiento del dominio, preferencias, habilidades, creencias (acerca del sistema, de la materia, etc.), etc.

Según la manera en que se considera y se recoge esta información distinguimos los siguientes tipos de modelo de usuario: Modelos explícitos y Modelos implícitos o automáticos (predefinidos o construidos por el sistema respectivamente), Modelado a corto y a largo plazo (útil para una sesión o a lo largo de todas las interacciones).

Toda esta información puede recogerse, bien de forma explícita (mediante formularios de preguntas presentados a los usuarios) o bien de forma implícita observando el comportamiento del usuario (qué opciones ha elegido, o qué datos ha enviado, etc.). Si para construir el modelo, el usuario y el sistema deben colaborar entonces estamos realizando modelado de usuario cooperativo.

La representación de esta información depende de cómo se utilizará en el sistema, lo más usual son los sistemas que utilizan modelos representados en forma de pares atributo-valor o reglas.

4. CBR : Concepto

El Razonamiento Basado en Casos (CBR) es una rama de la inteligencia artificial que se preocupa por el estudio de los mecanismos mentales necesarios para repetir lo que se ha hecho o vivido con anterioridad, ya sea por uno mismo, o ya sea por casos concretos recopilados en la bibliografía o en la sabiduría popular. Los diversos casos son del tipo "Si X, entonces Y" con algunas adaptaciones y críticas según las experiencias previas en el resultado de cada una de dichas reglas.

Las características principales del CBR son:

- *CBR es un razonamiento en base a la experiencia: usa ejemplos previos como punto de partida para el razonamiento.*
- *CBR resuelve nuevos problemas recuperando y adaptando soluciones de problemas previos.*
- *El razonamiento basado en casos (CBR) es una buena técnica de decisión, a largo plazo.*
- *Cuanto mayor sea la base de casos, más acertada será la decisión que se tome. No está pensado para una implantación inmediata.*

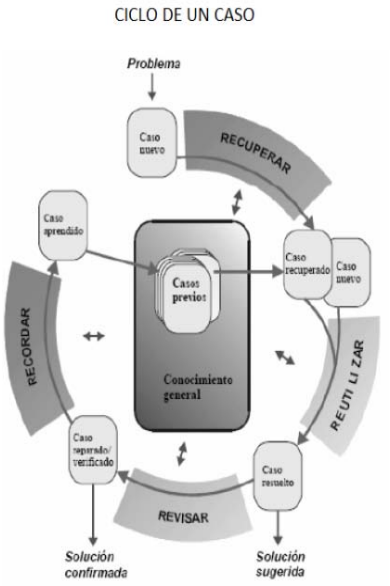


Figura 4.1. Ciclo de un caso

Ventajas

- *Se proponen soluciones rápidamente*
- *No se necesita conocer completamente el dominio*
- *Los casos son útiles para conceptos mal definidos*
- *Se resaltan las características importantes*

Desventajas

- *Los casos viejos pueden ser pobres*
- *Los casos más apropiados pueden no ser recuperados*
- *Puede necesitar conocimiento para realizar la adaptación*

4.1. Representación de los casos

Los casos contienen:

- *El problema que describe el estado del mundo cuando ocurrió el caso*
- *Una descripción de la solución encontrada*
- *Un resultado describiendo el estado del mundo después de que ocurrió el caso*

La representación puede hacerse de muy diversas maneras:

- *Frames*
- *Objetos*
- *–…*

Se ha de tener en cuenta:

- *La funcionalidad*
- *La facilidad de adquisición de la información representada en el caso*

4.2. Indexación

Indexar un caso consiste en identificar una o más características, que describen una situación concreta, para utilizarlas como índices durante la recuperación.

Características de los índices:

- *Predictivos*
- *Únicos*
- *Lo suficientemente concretos para poder ser reconocidos en el futuro*

4.3. Recuperación de los casos

- *El mecanismo de recuperación debe permitir recuperar un caso aunque no exista una combinación perfecta en base a similitud*
- *Usando métricas de similitud, no todas las características tienen la misma importancia*

5. Introducción al sistema Vedya

El sistema de aprendizaje que presentamos en este trabajo extiende la herramienta Vedya desarrollada en proyectos de Sistemas Informáticos anteriores mediante la incorporación de un Tutor Inteligente. Gracias a él es posible adquirir conocimientos sobre la estructura de datos conocida como montículo ("heap" en inglés), básica en el estudio y uso de las colas de prioridad.

El sistema consta de tres partes diferenciadas: una primera parte, en la cual se le presentan al alumno una pantalla para que pueda experimentar con la estructura de datos en tres vistas diferentes:

- *de usuario, con una representación en forma de árbol, que se aproxima más al enfoque clásico de los montículos (como árboles semicompletos);*
- *estática, que permite trabajar con el montículo implementado mediante el uso de una estructura estática (array) de elementos;*
- *dinámica, que permite centrarse en una visión inspirada en los punteros (como los usados en C), aunque de una forma mucho más amigable y fácil de entender para el usuario.*

La segunda y tercera parte se entrecruzan. La segunda parte consiste en una serie de test, que le serán presentados al usuario a medida que acaba satisfactoriamente las pruebas anteriores (que podrían ser el uso de las vistas de los montículos de la primera parte, o la realización de otros tests, así como el haber demostrado suficientes conocimientos en la programación de esta estructura en Java y Maude). Estos tests están orientados al afianzamiento de los conocimientos cubiertos por los ejercicios que se hayan realizado anteriormente.

En cuanto a la tercera parte, su desarrollo es crucial para demostrar los conocimientos adquiridos en el aprendizaje de los montículos. Se trata de una pequeña sesión de programación orientada a la realización de un diseño de la estructura de montículos. Primero, se realizará una especificación en Maude; y después, una implementación en Java.

Todo ello hace de Vedya una herramienta que puede resultar de gran ayuda a los estudiantes de ingeniería informática (y, por qué no, a cualquiera que quiera acercarse a conocer el campo de las estructuras de datos). Los montículos son una de las más importantes estructuras de datos, y Vedya puede convertirse en un importante refuerzo.

Veamos ahora algunos aspectos adicionales de la herramienta Vedya.

5.1 Herramienta Vedya Test

Esta importante herramienta permite la creación de los tests que se van a emplear en la herramienta Vedya.

Es importante disponer de esta herramienta porque permite la creación de tests de una forma sencilla y rápida. Además, permite guardar las preguntas en forma de base de preguntas, lo que resulta muy útil a la hora de exportar un test al lugar donde se necesita.

En la herramienta de Vedya, los tests que aparecen han sido creados gracias a la herramienta Vedya Test. Ha habido que adaptar partes de la herramienta para permitir poder leer estos tests desde dentro del sistema, y permitir su uso desde Vedya. Así, se consigue de una forma mucho mejor llegar al objetivo de que la herramienta pueda servir por sí sola como un sistema de tutorización inteligente.

5.2 Lenguaje Maude

El lenguaje Maude ofrece un lenguaje orientado a la especificación formal de tipos abstractos de datos mediante el uso de términos algebraicos que permite representar, formalmente, las estructuras de datos que se estudian en la asignatura en la asignatura de Estructura de Datos y de la Información.

Maude es un lenguaje extremadamente potente y orientado a programación declarativa, siendo capaz de soportar un álgebra de operaciones de composición de módulos extensible. Algunas de sus aplicaciones más interesantes son las de metalenguaje, en las cuales Maude es usado para crear entornos ejecutables para distintas lógicas, demostraciones de teoremas, lenguajes y modelos de computación. Se puede modelar una gran cantidad de estructuras usando este lenguaje, e incluso su potencia expresiva le lleva a poder expresar otros aspectos que los puramente referidos a dichas estructuras.

El lenguaje Maude, por sus características, se ajusta perfectamente a la tarea de especificar los montículos, así como otras estructuras de datos, como se ha indicado anteriormente. Sin embargo, y a pesar del gran peso que tiene Maude en las especificaciones, se ha optado por no incluir el sistema Maude completo en Vedya, sino que se ha incluido un pequeño procesador de

lenguaje que permite comprobar que se ha realizado una especificación que cumple con unos objetivos mínimos que son exigibles a cualquier alumno que esté aprendiendo montículos, debido a que son unas operaciones completamente esenciales.

Estas operaciones serán comentadas más tarde.

5.3 Lenguaje JAVA

Constituye este lenguaje uno de los más importantes lenguajes de programación desarrollados en los últimos años.

Java es un lenguaje de programación desarrollado por la empresa Sun Microsystems hacia el año 1995. Tomó mucha de su sintaxis de C y C++, pero dispone de un modelo de objetos más simple y carece de algunas características engorrosas que hacían de estos lenguajes algo difícil o lioso de utilizar muchas veces, como pueden ser las características de trabajo a bajo nivel o el uso de punteros.

La filosofía que impregna Java fue definida por Sun Microsystems de la siguiente forma: "Write once, run anywhere" (escribir una vez, ejecutar en cualquier sitio). Se trata de la forma en la que Java actúa: lejos de ser un lenguaje compilado como C o C++, se trata de un lenguaje interpretado. ¿Cómo se consigue esto?

La compilación de Java genera un bytecode, que es independiente de cualquier plataforma. Este bytecode es interpretado después, durante la ejecución del programa, por la máquina virtual de Java. Esta máquina virtual se encarga de ejecutar los programas escritos en Java, y permite el funcionamiento de dichos programas. Así pues, lo único que hace falta para ejecutar es esa máquina virtual. Disponiendo de ella, los programas pueden ser ejecutados en cualquier plataforma.

En el sistema Vedya, al igual que se ha hecho con el sistema Maude, se ha decidido por no incluir un compilador de Java, ni la herramienta Eclipse, sino que se ha decidido incluir un pequeño procesador de lenguaje capaz de verificar que se ha realizado una correcta implementación de los montículos en Java. Se darán más detalles de esto después.

6. Vedya: Montículos. De la especificación a la implementación

6.1 Motivación, objetivos y diseño

6.1.1 Motivación

Los montículos resultan, como se ha dicho anteriormente, una de las más importantes estructuras de datos que se pueden encontrar en el mundo de la informática. Siendo también una de las estructuras más difíciles de entender y, también, de aprender. Por ello, este año se ha decidido, en la asignatura de Sistemas Informáticos, diseñar una herramienta basada en el sistema Vedya para conseguir acercar a la gente en general, y a los alumnos de ingeniería informática de la Universidad Complutense en particular, esta importante estructura de datos.

Debido a que, por lo general, el estudio de esta estructura es duro y difícil (lo sabemos bien), hemos hecho un proyecto en el cual la visualización de los montículos sea mucho más amigable, y le permita al alumno experimentar por sí mismo con las operaciones que pueden ser llevadas a cabo con los montículos. De esta forma, deseamos que su estudio no sea tan dificultoso, y se pueda convertir en algo, por qué no, mucho más ameno.

6.1.2 Objetivo

El objetivo ha sido conseguir hacer de la herramienta Vedya, en principio destinada a la ayuda a la comprensión de las distintas estructuras de datos y de los métodos algorítmicos, un sistema que se comporta de forma similar a un tutor inteligente, capaz de ayudar y de enseñar la estructura de datos conocida como montículos.

Así, la herramienta pasa de ser algo pasivo a algo más activo, con capacidad no sólo para enseñar el funcionamiento de las estructuras de datos, sino que permite familiarizarse con ellas (aunque, en este caso, sean sólo los montículos), permitiendo además afianzar los conocimientos y practicar no sólo la mera teoría, sino también la práctica.

6.1.3 Diseño

6.1.3.1 Casos de Uso

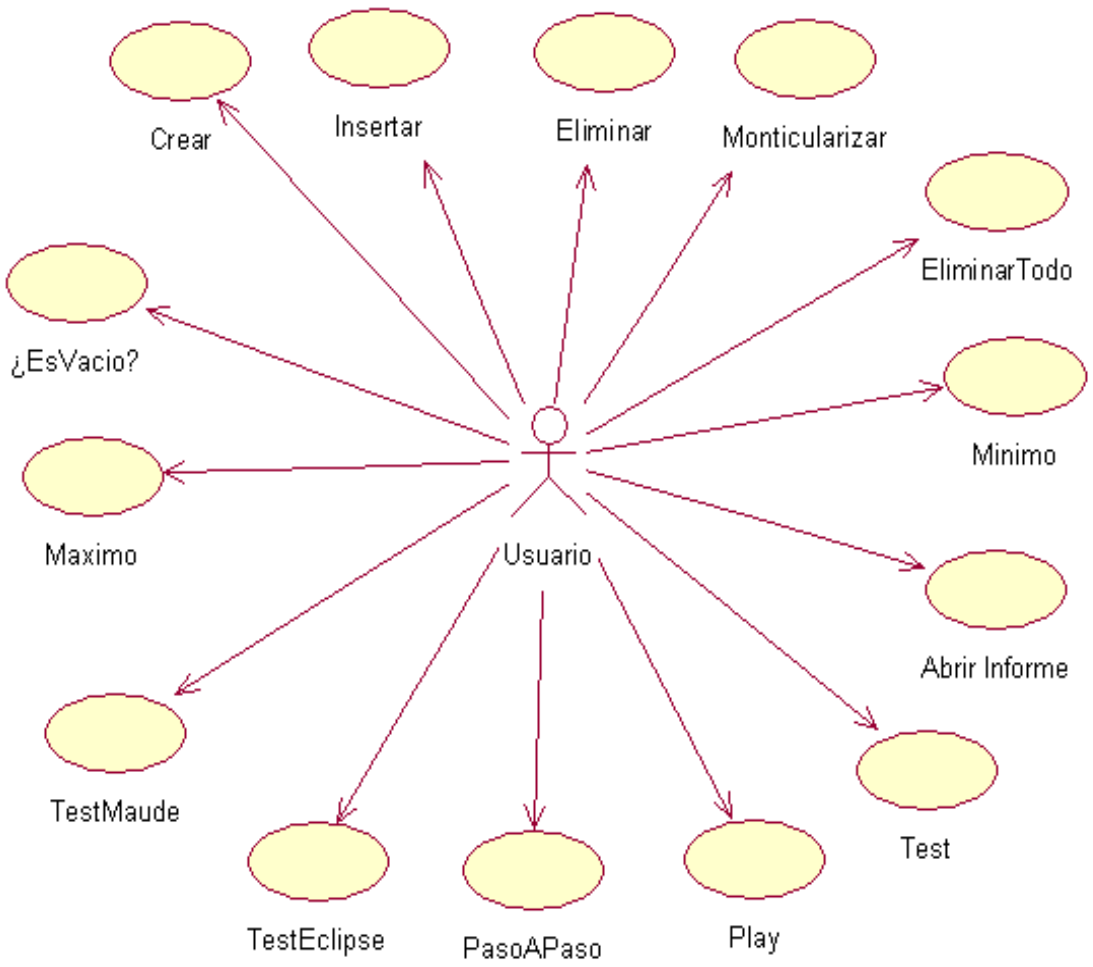


Figura 6.1. Casos de uso generales

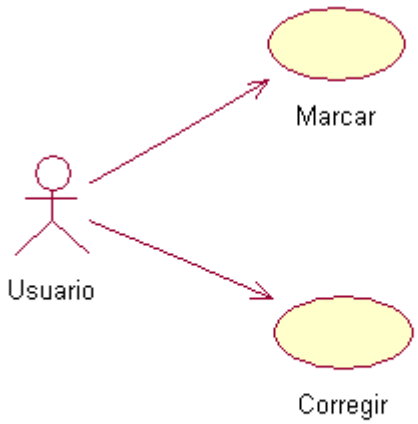


Figura 6.2. Casos de uso correspondientes al primer test

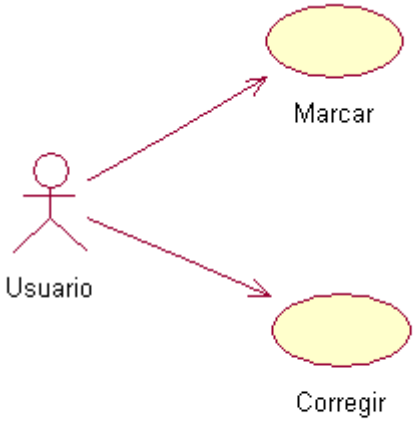


Figura 6.3. Casos de uso correspondientes al test de Maude

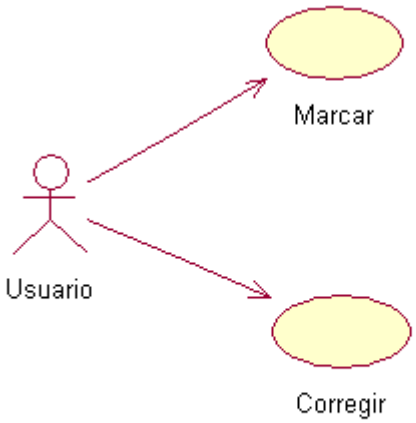


Figura 6.4. Casos de uso correspondientes al test de Eclipse

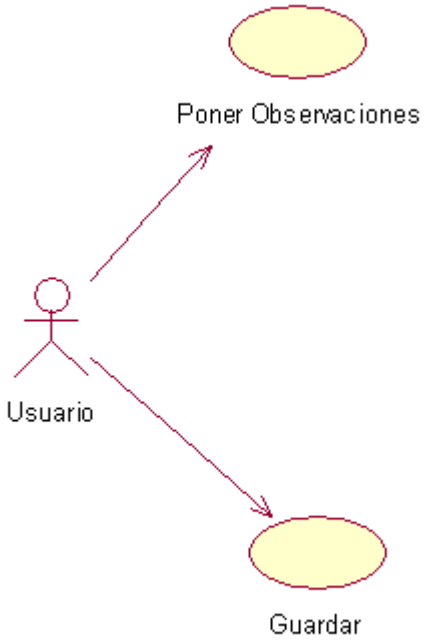
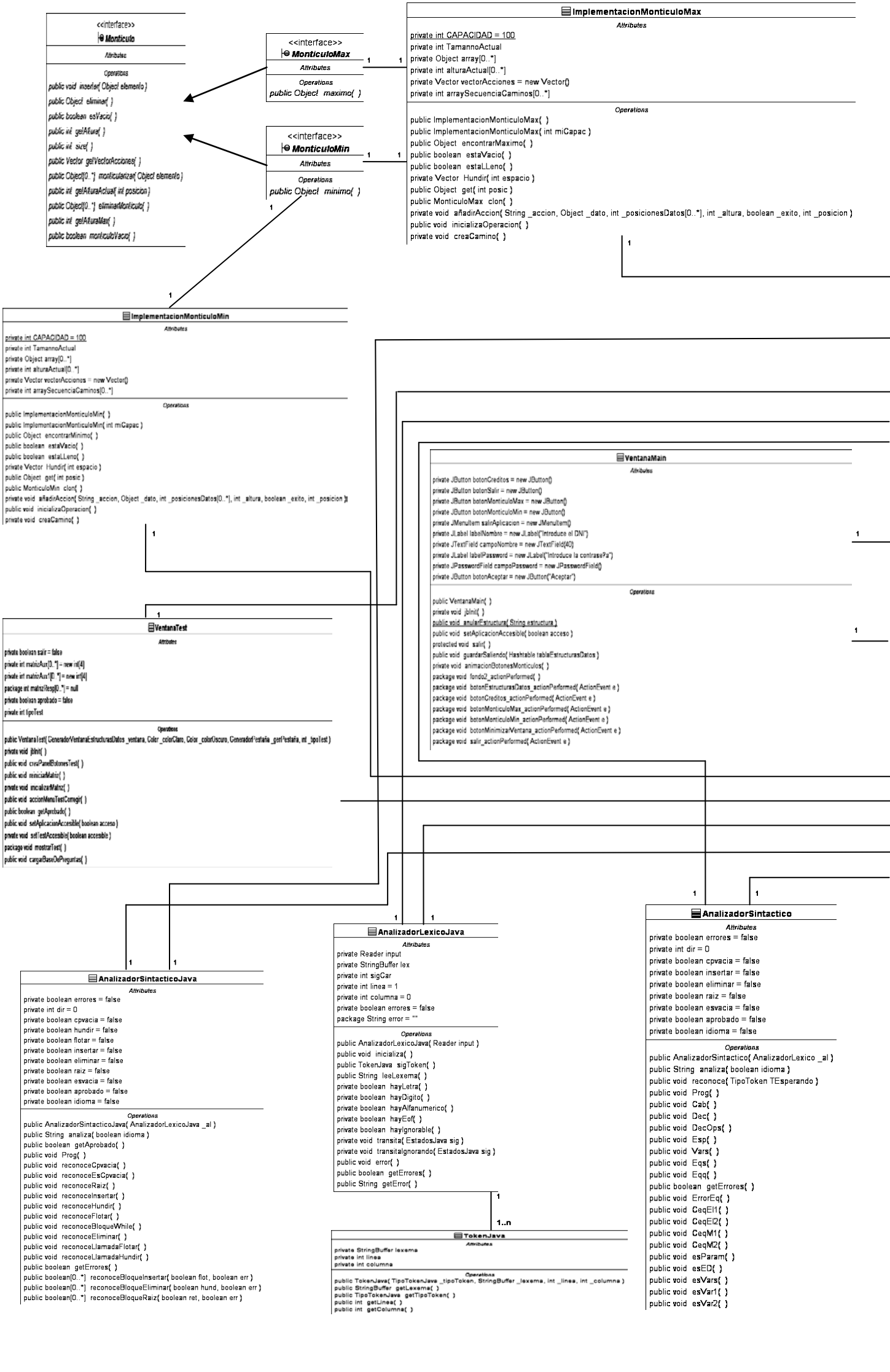
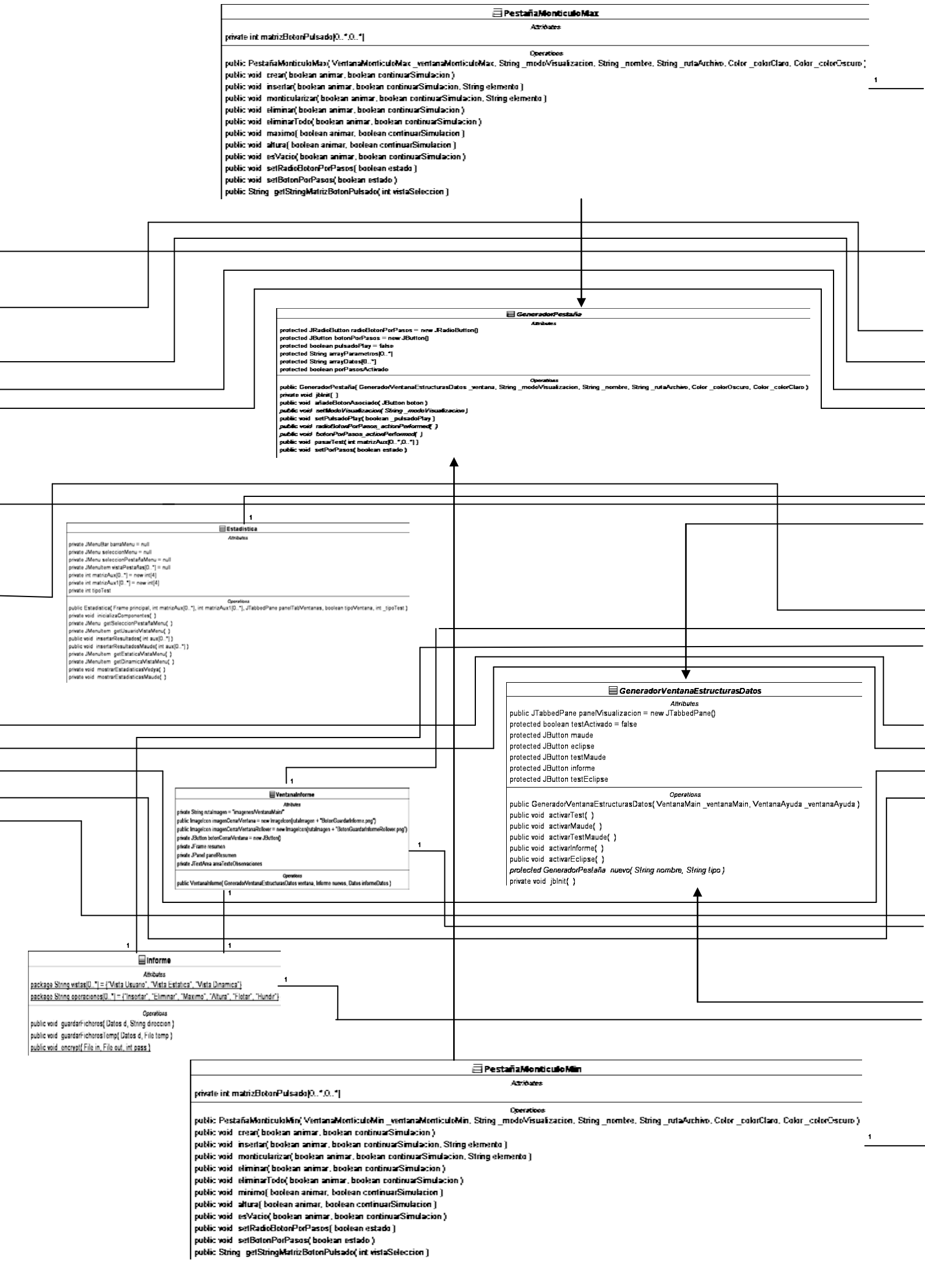


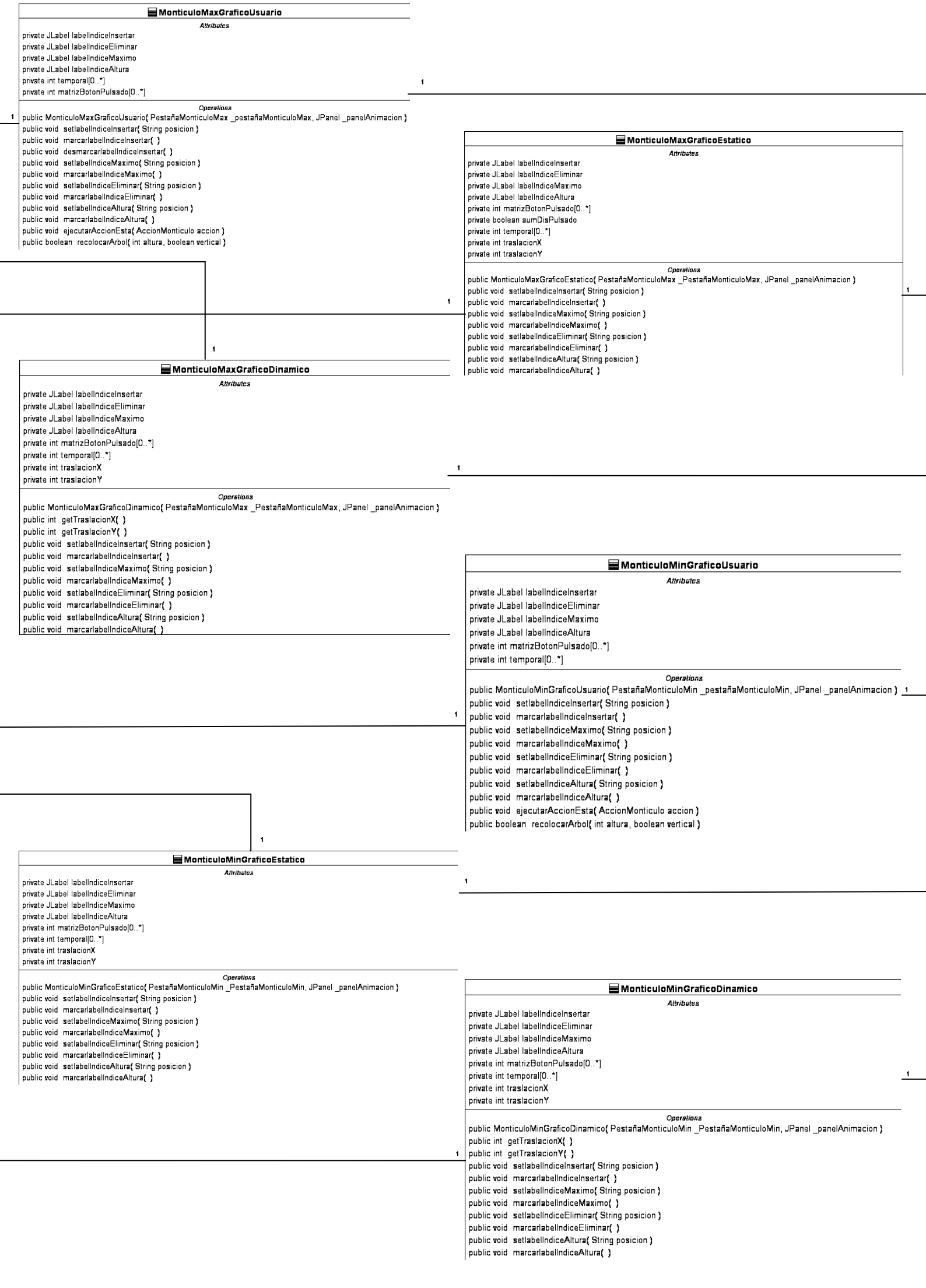
Figura 6.5. Casos de uso correspondientes a la ventana del Informe

6.1.3.2 Diagrama de clases


«interface»
Monticulo
Atributos
Operations
public void insertar(Object elemento)
public Object eliminar( )
public boolean esVacio( )
public int getAltura( )
public int size( )
public Vector getVectorAcciones( )
public Object[0..*] monticularizar(Object elemento)
public int getAlturaActual(int posicion)
public Object[0..*] eliminarMonticulo( )
public int getAlturaMax( )
public boolean monticuloVacio( )
«interface»
MonticuloMax
Attributes
Operations
public Object maximo( )
«interface»
MonticuloMin
Attributes
Operations
public Object minimo( )
1
1
1
1
1
1
ImplementacionMonticuloMax
Attributes
private int CAPACIDAD = 100
private int TamannoActual
private Object array[0..*]
private int alturaActual[0..*]
private Vector vectorAcciones = new Vector()
private int arraySecuenciaCaminos[0..*]
Operations
public ImplementacionMonticuloMax( )
public ImplementacionMonticuloMax(int miCapac)
public Object encontrarMaximo( )
public boolean estaVacio( )
public boolean estaLLeno( )
private Vector Hundir(int espacio)
public Object get(int posic)
public MonticuloMax clon( )
private void añadirAccion(String _accion, Object _dato, int _posicionesDatos[0..*], int _altura, boolean _exito, int _posicion)
public void inicializaOperacion( )
private void creaCamino( )
1
ImplementacionMonticuloMin
Attributes
private int CAPACIDAD = 100
private int TamannoActual
private Object array[0..*]
private int alturaActual[0..*]
private Vector vectorAcciones = new Vector()
private int arraySecuenciaCaminos[0..*]
Operations
public ImplementacionMonticuloMin( )
public ImplementacionMonticuloMin(int miCapac)
public Object encontrarMinimo( )
public boolean estaVacio( )
public boolean estaLLeno( )
private Vector Hundir(int espacio)
public Object get(int posic)
public MonticuloMin clon( )
private void añadirAccion(String _accion, Object _dato, int _posicionesDatos[0..*], int _altura, boolean _exito, int _posicion)
public void inicializaOperacion( )
private void creaCamino( )
1
1
VentanaMain
Attributes
private JButton botonCreditos = new JButton()
private JButton botonSalir = new JButton()
private JButton botonMonticuloMax = new JButton()
private JButton botonMonticuloMin = new JButton()
private JMenuItem salirAplicacion = new JMenuItem()
private JLabel labelNombre = new JLabel("Introduce el DNI")
private JTextField campoNombre = new JTextField(40)
private JLabel labelPassword = new JLabel("Introduce la contraseña")
private JPasswordField campoPassword = new JPasswordField()
private JButton botonAceptar = new JButton("Aceptar")
Operations
public VentanaMain( )
private void jbInit( )
public void anularEstructura(String estructura)
public void setAplicacionAccesible(boolean acceso)
protected void salir( )
public void guardarSaliendo(Hashtable tablaEstructurasDatos)
private void animacionBotonesMonticulos( )
package void fondo2_actionPerformed( )
package void botonEstructurasDatos_actionPerformed(ActionEvent e)
package void botonCreditos_actionPerformed(ActionEvent e)
package void botonMonticuloMax_actionPerformed(ActionEvent e)
package void botonMonticuloMin_actionPerformed(ActionEvent e)
package void botonMinimizarVentana_actionPerformed(ActionEvent e)
package void salir_actionPerformed(ActionEvent e)
1
1
VentanaTest
Attributes
private boolean salir = false
private int matrizAux[0..*] = new int[4]
private int matrizAux1[0..*] = new int[4]
package int matrizResp[0..*] = null
private boolean aprobado = false
private int tipoTest
Operations
public VentanaTest(GeneradorVentanaEstructurasDatos _ventana, Color _colorClaro, Color _colorOscuro, GeneradorPestaña _genPestaña, int _tipoTest)
private void jbInit( )
public void creaPanelBotonesTest( )
public void reiniciaMatriz( )
private void inicializarMatriz( )
public void accionMenuTestCorregir( )
public boolean getAprobado( )
public void setAplicacionAccesible(boolean acceso)
private void setTestAccesible(boolean accesible)
package void mostrarTest( )
public void cargaBaseDePreguntas( )
1
1
1
1
1
1
AnalizadorSintacticoJava
Attributes
private boolean errores = false
private int dir = 0
private boolean cpvacia = false
private boolean hundir = false
private boolean flotar = false
private boolean insertar = false
private boolean eliminar = false
private boolean raiz = false
private boolean esvacia = false
private boolean aprobado = false
private boolean idioma = false
Operations
public AnalizadorSintacticoJava(AnalizadorLexicoJava _al)
public String analiza(boolean idioma)
public boolean getAprobado( )
public void Prog( )
public void reconoceCpvacia( )
public void reconoceEsCpvacia( )
public void reconoceRaiz( )
public void reconoceInsertar( )
public void reconoceHundir( )
public void reconoceFlotar( )
public void reconoceBloqueWhile( )
public void reconoceEliminar( )
public void reconoceLlamadaFlotar( )
public void reconoceLlamadaHundir( )
public boolean getErrores( )
public boolean[0..*] reconoceBloqueInsertar(boolean flot, boolean err)
public boolean[0..*] reconoceBloqueEliminar(boolean hund, boolean err)
public boolean[0..*] reconoceBloqueRaiz(boolean ret, boolean err)
1
1
AnalizadorLexicoJava
Attributes
private Reader input
private StringBuffer lex
private int sigCar
private int linea = 1
private int columna = 0
private boolean errores = false
package String error = ""
Operations
public AnalizadorLexicoJava(Reader input)
public void inicializa( )
public TokenJava sigToken( )
public String leeLexema( )
private boolean hayLetra( )
private boolean hayDigito( )
private boolean hayAlfanumerico( )
private boolean hayEof( )
private boolean hayIgnorable( )
private void transita(EstadosJava sig)
private void transitaIgnorando(EstadosJava sig)
public void error( )
public boolean getErrores( )
public String getError( )
1
1..n
TokenJava
Attributes
private StringBuffer lexema
private int linea
private int columna
Operations
public TokenJava(TipoTokenJava _tipoToken, StringBuffer _lexema, int _linea, int _columna)
public StringBuffer getLexema( )
public TipoTokenJava getTipoToken( )
public int getLinea( )
public int getColumna( )
1
1
AnalizadorSintactico
Attributes
private boolean errores = false
private int dir = 0
private boolean cpvacia = false
private boolean insertar = false
private boolean eliminar = false
private boolean raiz = false
private boolean esvacia = false
private boolean aprobado = false
private boolean idioma = false
Operations
public AnalizadorSintactico(AnalizadorLexico _al)
public String analiza(boolean idioma)
public void reconoce(TipoToken TEsperando)
public void Prog( )
public void Cab( )
public void Dec( )
public void DecOps( )
public void Esp( )
public void Vars( )
public void Eqs( )
public void Eqq( )
public boolean getErrores( )
public void ErrorEq( )
public void CeqEl1( )
public void CeqEl2( )
public void CeqM1( )
public void CeqM2( )
public void esParam( )
public void esED( )
public void esVars( )
public void esVar1( )
public void esVar2( )


PestañaMonticuloMax
Atributes
private int matrizBotonPulsado[0..*,0..*]
Operations
public PestañaMonticuloMax(VentanaMonticuloMax _ventanaMonticuloMax, String _modoVisualizacion, String _nombre, String _rutaArchivo, Color _colorClaro, Color _colorOscuro)
public void crear(boolean animar, boolean continuarSimulacion)
public void insertar(boolean animar, boolean continuarSimulacion, String elemento)
public void monticularizar(boolean animar, boolean continuarSimulacion, String elemento)
public void eliminar(boolean animar, boolean continuarSimulacion)
public void eliminarTodo(boolean animar, boolean continuarSimulacion)
public void maximo(boolean animar, boolean continuarSimulacion)
public void altura(boolean animar, boolean continuarSimulacion)
public void esVacio(boolean animar, boolean continuarSimulacion)
public void setRadioBotonPorPasos(boolean estado)
public void setBotonPorPasos(boolean estado)
public String getStringMatrizBotonPulsado(int vistaSeleccion)
1
GeneradorPestaña
Atributes
protected JRadioButton radioBotonPorPasos = new JRadioButton()
protected JButton botonPorPasos = new JButton()
protected boolean pulsadoPlay = false
protected String arrayParametros[0..*]
protected String arrayDatos[0..*]
protected boolean porPasosActivado
Operations
public GeneradorPestaña(GeneradorVentanaEstructurasDatos _ventana, String _modoVisualizacion, String _nombre, String _rutaArchivo, Color _colorOscuro, Color _colorClaro)
private void jbInit( )
public void añadeBotonAsociado(JButton boton)
public void setModoVisualizacion(String _modoVisualizacion)
public void setPulsadoPlay(boolean _pulsadoPlay)
public void radioBotonPorPasos_actionPerformed( )
public void botonPorPasos_actionPerformed( )
public void pasarTest(int matrizAux[0..*,0..*])
public void setPorPasos(boolean estado)
1
Estadistica
Atributes
private JMenuBar barraMenu = null
private JMenu seleccionMenu = null
private JMenu seleccionPestañaMenu = null
private JMenuItem vistaPestañas[0..*] = null
private int matrizAux[0..*] = new int[4]
private int matrizAux1[0..*] = new int[4]
private int tipoTest
Operations
public Estadistica(Frame principal, int matrizAux[0..*], int matrizAux1[0..*], JTabbedPane panelTabVentanas, boolean tipoVentana, int _tipoTest)
private void inicializaComponentes( )
private JMenu getSeleccionPestañaMenu( )
private JMenuItem getUsuarioVistaMenu( )
public void insertarResultados(int aux[0..*])
public void insertarResultadosMaude(int aux[0..*])
private JMenuItem getEstaticaVistaMenu( )
private JMenuItem getDinamicaVistaMenu( )
private void mostrarEstadisticasVedya( )
private void mostrarEstadisticasMaude( )
GeneradorVentanaEstructurasDatos
Atributes
public JTabbedPane panelVisualizacion = new JTabbedPane()
protected boolean testActivado = false
protected JButton maude
protected JButton eclipse
protected JButton testMaude
protected JButton informe
protected JButton testEclipse
Operations
public GeneradorVentanaEstructurasDatos(VentanaMain _ventanaMain, VentanaAyuda _ventanaAyuda)
public void activarTest( )
public void activarMaude( )
public void activarTestMaude( )
public void activarInforme( )
public void activarEclipse( )
protected GeneradorPestaña nuevo(String nombre, String tipo)
private void jbInit( )
1
VentanaInforme
Atributes
private String rutaImagen = "imagenes/VentanaMain"
public ImageIcon imagenCerraVentana = new ImageIcon(rutaImagen + "BotonGuardarInforme.png")
public ImageIcon imagenCerraVentanaRollover = new ImageIcon(rutaImagen + "BotonGuardarInformeRollover.png")
private JButton botonCerraVentana = new JButton()
private JFrame resumen
private JPanel panelResumen
private JTextArea areaTesteObservaciones
Operations
public VentanaInforme(GeneradorVentanaEstructurasDatos ventana, Informe nuevo, Datos informeDatos)
1
1
1
1
1
Informe
Atributes
package String vistas[0..*] = {"Vista Usuario", "Vista Estatica", "Vista Dinamica"}
package String operaciones[0..*] = {"Insertar", "Eliminar", "Maximo", "Altura", "Flotar", "Hundir"}
Operations
public void guardarFichoros(Datos d, String direccion)
public void guardarFichorosTemp(Datos d, File temp)
public void encrypt(File in, File out, int pass)
1
PestañaMonticuloMin
Atributes
private int matrizBotonPulsado[0..*,0..*]
Operations
public PestañaMonticuloMin(VentanaMonticuloMin _ventanaMonticuloMin, String _modoVisualizacion, String _nombre, String _rutaArchivo, Color _colorClaro, Color _colorOscuro)
public void crear(boolean animar, boolean continuarSimulacion)
public void insertar(boolean animar, boolean continuarSimulacion, String elemento)
public void monticularizar(boolean animar, boolean continuarSimulacion, String elemento)
public void eliminar(boolean animar, boolean continuarSimulacion)
public void eliminarTodo(boolean animar, boolean continuarSimulacion)
public void minimo(boolean animar, boolean continuarSimulacion)
public void altura(boolean animar, boolean continuarSimulacion)
public void esVacio(boolean animar, boolean continuarSimulacion)
public void setRadioBotonPorPasos(boolean estado)
public void setBotonPorPasos(boolean estado)
public String getStringMatrizBotonPulsado(int vistaSeleccion)
1


MonticuloMaxGrafico
Attributes
private JLabel labelFlechaNO
private JLabel labelFlechaNE
private JLabel labelFlechaSO
private JLabel labelFlechaSE
private Vector vectorAcciones
private int indiceVectorAcciones = 0
Operations
public MonticuloMaxGrafico(PestañaMonticuloMax _pestañaMonticuloMax, JPanel _panelAnimacion)
public void dibujaMonticuloMax(Iterator iterador, int numElementosMonticulo)
public void ejecutarAcciones( )
public void ejecutarAccionCrea(AccionMonticulo accion)
public void ejecutarAccionAñade(AccionMonticulo accion)
public void ejecutarAccionElimina(AccionMonticulo accion)
public void ejecutarAccionIntercambia(AccionMonticulo accion)
public void ejecutarAccionEsVacio(AccionMonticulo accion)
public void ejecutarAccionCambiar(AccionMonticulo accion)
public void ejecutarAccionMaximo(AccionMonticulo accion)
public void ejecutarAccionAltura(AccionMonticulo accion)
public void setMatrizBoton(int matriz[0..*])
public void setMatrizBotonAntigua(int matriz[0..*])
VentanaMonticuloMax
Attributes
private JRadioButtonMenuItem barraHerramientasVisualizacionModoUsuario
private JRadioButtonMenuItem barraHerramientasVisualizacionModoImplementacionEstatica
private JRadioButtonMenuItem barraHerramientasVisualizacionModoImplementacionDinamica
private JButton botonMonticuloMaxCrear
private JButton botonMonticuloMaxInsertar
private JButton botonMonticuloMaxEliminar
private JButton botonMonticuloMaxMaximo
private JButton botonMonticuloMaxAltura
private JButton botonMonticuloMaxEsVacio
private JButton botonMonticuloMaxEliminarTodo
private JButton botonMonticuloMaxMonticulizar
Operations
public VentanaMonticuloMax(VentanaMain _ventanaMain, VentanaAyuda _ventanaAyuda, String nombre, String grupo)
public Object[0..*] ejecutaOperacion(MonticuloMax monticuloMax, String operacion, String tipo, int numeroLinea)
public void accionVisualizacionModoUsuario( )
public void accionVisualizacionModoImplementacionEstatica( )
public void accionVisualizacionModoImplementacionDinamica( )
protected void menuMonticuloMaxCrear_actionPerformed( )
protected void menuMonticuloMaxInsertar_actionPerformed( )
protected void menuMonticuloMaxEliminar_actionPerformed( )
protected void menuMonticuloMaxMaximo_actionPerformed( )
protected void menuMonticuloMaxAltura_actionPerformed( )
protected void menuMonticuloMaxEsVacio_actionPerformed( )
protected void menuMonticuloMaxEliminarTodo_actionPerformed( )
protected void menuMonticuloMaxMonticulizar_actionPerformed( )
public void añadirTestInforme(int matrizAux[0..*], int tipo)
public void muestra_estadisticas(int matrizAux[0..*], int matrizAux1[0..*], int tipoTest)
MonticuloMinGrafico
Attributes
private JLabel labelFlechaNO
private JLabel labelFlechaNE
private JLabel labelFlechaSO
private JLabel labelFlechaSE
private Vector vectorAcciones
private int indiceVectorAcciones = 0
Operations
public MonticuloMinGrafico(_pestañaMonticuloMin, JPanel _panelAnimacion)
public void dibujaMonticuloMin(Iterator iterador, int numElementosMonticulo)
public void ejecutarAcciones( )
public void ejecutarAccionCrea(AccionMonticulo accion)
public void ejecutarAccionAñade(AccionMonticulo accion)
public void ejecutarAccionElimina(AccionMonticulo accion)
public void ejecutarAccionIntercambia(AccionMonticulo accion)
public void ejecutarAccionEsVacio(AccionMonticulo accion)
public void ejecutarAccionCambiar(AccionMonticulo accion)
public void ejecutarAccionMinimo(AccionMonticulo accion)
public void ejecutarAccionAltura(AccionMonticulo accion)
public void setMatrizBoton(int matriz[0..*])
public void setMatrizBotonAntigua(int matriz[0..*])
VentanaMonticuloMin
Attributes
private JRadioButtonMenuItem radioMenuVisualizacionModoUsuario
private JRadioButtonMenuItem radioMenuVisualizacionModoImplementacionEstatica
private JRadioButtonMenuItem radioMenuVisualizacionModoImplementacionDinamica
private JButton botonMonticuloMinCrear
private JButton botonMonticuloMinInsertar
private JButton botonMonticuloMinEliminar
private JButton botonMonticuloMinMinimo
private JButton botonMonticuloMinAltura
private JButton botonMonticuloMinEsVacio
private JButton botonMonticuloMinEliminarTodo
private JButton botonMonticuloMinMonticulizar
Operations
public VentanaMonticuloMin(VentanaMain _ventanaMain, VentanaAyuda _ventanaAyuda, String nombre, String grupo)
public Object[0..*] ejecutaOperacion(MonticuloMin monticuloMin, String operacion, String tipo, int numeroLinea)
public void accionVisualizacionModoUsuario( )
public void accionVisualizacionModoImplementacionEstatica( )
public void accionVisualizacionModoImplementacionDinamica( )
protected void menuMonticuloMinCrear_actionPerformed( )
protected void menuMonticuloMinInsertar_actionPerformed( )
protected void menuMonticuloMinEliminar_actionPerformed( )
protected void menuMonticuloMinMinimo_actionPerformed( )
protected void menuMonticuloMinAltura_actionPerformed( )
protected void menuMonticuloMinEsVacio_actionPerformed( )
protected void menuMonticuloMinEliminarTodo_actionPerformed( )
protected void menuMonticuloMinMonticulizar_actionPerformed( )
public void muestra_estadisticas(int matrizAux[0..*], int matrizAux1[0..*], int tipoTest)
public void añadirTestInforme(int matrizAux[0..*], int tipo)


MonticuloMaxGraficoUsuario
Attributes
private JLabel labelIndiceInsertar
private JLabel labelIndiceEliminar
private JLabel labelIndiceMaximo
private JLabel labelIndiceAltura
private int temporal[0..*]
private int matrizBotonPulsado[0..*]
Operations
public MonticuloMaxGraficoUsuario(PestañaMonticuloMax _pestañaMonticuloMax, JPanel _panelAnimacion)
public void setlabelIndiceInsertar(String posicion)
public void marcarlabelIndiceInsertar( )
public void desmarcarlabelIndiceInsertar( )
public void setlabelIndiceMaximo(String posicion)
public void marcarlabelIndiceMaximo( )
public void setlabelIndiceEliminar(String posicion)
public void marcarlabelIndiceEliminar( )
public void setlabelIndiceAltura(String posicion)
public void marcarlabelIndiceAltura( )
public void ejecutarAccionEsta(AccionMonticulo accion)
public boolean recolocarArbol(int altura, boolean vertical)
MonticuloMaxGraficoEstatico
Attributes
private JLabel labelIndiceInsertar
private JLabel labelIndiceEliminar
private JLabel labelIndiceMaximo
private JLabel labelIndiceAltura
private int matrizBotonPulsado[0..*]
private boolean aumDisPulsado
private int temporal[0..*]
private int traslacionX
private int traslacionY
Operations
public MonticuloMaxGraficoEstatico(PestañaMonticuloMax _PestañaMonticuloMax, JPanel _panelAnimacion)
public void setlabelIndiceInsertar(String posicion)
public void marcarlabelIndiceInsertar( )
public void setlabelIndiceMaximo(String posicion)
public void marcarlabelIndiceMaximo( )
public void setlabelIndiceEliminar(String posicion)
public void marcarlabelIndiceEliminar( )
public void setlabelIndiceAltura(String posicion)
public void marcarlabelIndiceAltura( )
MonticuloMaxGraficoDinamico
Attributes
private JLabel labelIndiceInsertar
private JLabel labelIndiceEliminar
private JLabel labelIndiceMaximo
private JLabel labelIndiceAltura
private int matrizBotonPulsado[0..*]
private int temporal[0..*]
private int traslacionX
private int traslacionY
Operations
public MonticuloMaxGraficoDinamico(PestañaMonticuloMax _PestañaMonticuloMax, JPanel _panelAnimacion)
public int getTraslacionX( )
public int getTraslacionY( )
public void setlabelIndiceInsertar(String posicion)
public void marcarlabelIndiceInsertar( )
public void setlabelIndiceMaximo(String posicion)
public void marcarlabelIndiceMaximo( )
public void setlabelIndiceEliminar(String posicion)
public void marcarlabelIndiceEliminar( )
public void setlabelIndiceAltura(String posicion)
public void marcarlabelIndiceAltura( )
MonticuloMinGraficoUsuario
Attributes
private JLabel labelIndiceInsertar
private JLabel labelIndiceEliminar
private JLabel labelIndiceMaximo
private JLabel labelIndiceAltura
private int matrizBotonPulsado[0..*]
private int temporal[0..*]
Operations
public MonticuloMinGraficoUsuario(PestañaMonticuloMin _pestañaMonticuloMin, JPanel _panelAnimacion)
public void setlabelIndiceInsertar(String posicion)
public void marcarlabelIndiceInsertar( )
public void setlabelIndiceMaximo(String posicion)
public void marcarlabelIndiceMaximo( )
public void setlabelIndiceEliminar(String posicion)
public void marcarlabelIndiceEliminar( )
public void setlabelIndiceAltura(String posicion)
public void marcarlabelIndiceAltura( )
public void ejecutarAccionEsta(AccionMonticulo accion)
public boolean recolocarArbol(int altura, boolean vertical)
MonticuloMinGraficoEstatico
Attributes
private JLabel labelIndiceInsertar
private JLabel labelIndiceEliminar
private JLabel labelIndiceMaximo
private JLabel labelIndiceAltura
private int matrizBotonPulsado[0..*]
private int temporal[0..*]
private int traslacionX
private int traslacionY
Operations
public MonticuloMinGraficoEstatico(PestañaMonticuloMin _PestañaMonticuloMin, JPanel _panelAnimacion)
public void setlabelIndiceInsertar(String posicion)
public void marcarlabelIndiceInsertar( )
public void setlabelIndiceMaximo(String posicion)
public void marcarlabelIndiceMaximo( )
public void setlabelIndiceEliminar(String posicion)
public void marcarlabelIndiceEliminar( )
public void setlabelIndiceAltura(String posicion)
public void marcarlabelIndiceAltura( )
MonticuloMinGraficoDinamico
Attributes
private JLabel labelIndiceInsertar
private JLabel labelIndiceEliminar
private JLabel labelIndiceMaximo
private JLabel labelIndiceAltura
private int matrizBotonPulsado[0..*]
private int temporal[0..*]
private int traslacionX
private int traslacionY
Operations
public MonticuloMinGraficoDinamico(PestañaMonticuloMin _PestañaMonticuloMin, JPanel _panelAnimacion)
public int getTraslacionX( )
public int getTraslacionY( )
public void setlabelIndiceInsertar(String posicion)
public void marcarlabelIndiceInsertar( )
public void setlabelIndiceMaximo(String posicion)
public void marcarlabelIndiceMaximo( )
public void setlabelIndiceEliminar(String posicion)
public void marcarlabelIndiceEliminar( )
public void setlabelIndiceAltura(String posicion)
public void marcarlabelIndiceAltura( )
1


HiloMonticuloMaxUsuario
Attributes
private boolean exito
private int tramo = 0
Operations
public HiloMonticuloMaxUsuario(MonticuloMaxGraficoUsuario _monticuloMaxGraficoUsuario, String _accion, ElementoGraficoMonticuloUsuario _elemento, int _altura, int _velocidadAnimaciones, int _maximo, boolean _seguir)
public HiloMonticuloMaxUsuario(MonticuloMaxGraficoUsuario _monticuloMaxGraficoUsuario, String _accion, boolean _exito, int _velocidadAnimaciones)
public HiloMonticuloMaxUsuario(MonticuloMaxGraficoUsuario _monticuloMaxGraficoUsuario, String _accion, ElementoGrafico _vectorElementos[0..*], int _velocidadAnimaciones, int _maximo, boolean _seguir)
public HiloMonticuloMaxUsuario(MonticuloMaxGraficoUsuario _monticuloMaxGraficoUsuario, int _altura, boolean _recolocarVertical, int _velocidadAnimaciones)
public void finish( )
public void run( )
1
HiloMonticuloMaxEstatico
Attributes
private boolean exito
private int tramo = 0
Operations
public HiloMonticuloMaxEstatico(MonticuloMaxGraficoEstatico _monticuloMaxGraficoEstatico, ElementoGraficoMonticuloEstatico _elemento, String _accion, int _posicionYFinal, int _velocidadAnimaciones, int _maximo, boolean _seguir)
public HiloMonticuloMaxEstatico(MonticuloMaxGraficoEstatico _monticuloMaxGraficoEstatico, String _accion, ElementoGraficoMonticuloEstatico _elemento, int _altura, int _velocidadAnimaciones, int _maximo, boolean _seguir)
public HiloMonticuloMaxEstatico(MonticuloMaxGraficoEstatico _monticuloMaxGraficoEstatico, String _accion, boolean _exito, int _velocidadAnimaciones)
public HiloMonticuloMaxEstatico(MonticuloMaxGraficoEstatico _monticuloMaxGraficoEstatico, String _accion, ElementoGrafico _vectorElementos[0..*], int _velocidadAnimaciones, int _maximo, boolean _seguir)
public void finish( )
public void run( )
1
HiloMonticuloMaxDinamico
Attributes
private boolean exito
private int tramo = 0
Operations
public HiloMonticuloMaxDinamico(MonticuloMaxGraficoDinamico _monticuloMaxGraficoDinamico, String _accion, ElementoGrafico _vectorElementos[0..*], int _velocidadAnimaciones, int _maximo, boolean _seguir)
public HiloMonticuloMaxDinamico(MonticuloMaxGraficoDinamico _monticuloMaxGraficoDinamico, String _accion, ElementoGraficoMonticuloDinamico _elemento, int _altura, int _velocidadAnimaciones, int _maximo, boolean _seguir)
public HiloMonticuloMaxDinamico(MonticuloMaxGraficoDinamico _monticuloMaxGraficoDinamico, String _accion, boolean _exito, int _velocidadAnimaciones)
public void finish( )
public void run( )
1
HiloMonticuloMinUsuario
Attributes
private boolean exito
private int tramo = 0
Operations
public HiloMonticuloMinUsuario(MonticuloMinGraficoUsuario _monticuloMinGraficoUsuario, String _accion, ElementoGraficoMonticuloUsuario _elemento, int _altura, int _velocidadAnimaciones, int _maximo, boolean _seguir)
public HiloMonticuloMinUsuario(MonticuloMinGraficoUsuario _monticuloMinGraficoUsuario, String _accion, boolean _exito, int _velocidadAnimaciones)
public HiloMonticuloMinUsuario(MonticuloMinGraficoUsuario _monticuloMinGraficoUsuario, String _accion, ElementoGrafico _vectorElementos[0..*], int _velocidadAnimaciones, int _maximo, boolean _seguir)
public HiloMonticuloMinUsuario(MonticuloMinGraficoUsuario _monticuloMinGraficoUsuario, int _altura, boolean _recolocarVertical, int _velocidadAnimaciones)
public void finish( )
public void run( )
1
HiloMonticuloMinEstatico
Attributes
private boolean exito
private int tramo = 0
Operations
public HiloMonticuloMinEstatico(MonticuloMinGraficoEstatico _monticuloMinGraficoEstatico, ElementoGraficoMonticuloEstatico _elemento, String _accion, int _posicionYFinal, int _velocidadAnimaciones, int _maximo, boolean _seguir)
public HiloMonticuloMinEstatico(MonticuloMinGraficoEstatico _monticuloMinGraficoEstatico, String _accion, ElementoGraficoMonticuloEstatico _elemento, int _altura, int _velocidadAnimaciones, int _maximo, boolean _seguir)
public HiloMonticuloMinEstatico(MonticuloMinGraficoEstatico _monticuloMinGraficoEstatico, String _accion, boolean _exito, int _velocidadAnimaciones)
public HiloMonticuloMinEstatico(MonticuloMinGraficoEstatico _monticuloMinGraficoEstatico, String _accion, ElementoGrafico _vectorElementos[0..*], int _velocidadAnimaciones, int _maximo, boolean _seguir)
public void finish( )
public void run( )
1
HiloMonticuloMinDinamico
Attributes
private boolean exito
private int tramo = 0
Operations
public HiloMonticuloMinDinamico(MonticuloMinGraficoDinamico _monticuloMinGraficoDinamico, String _accion, ElementoGrafico _vectorElementos[0..*], int _velocidadAnimaciones, int _maximo, boolean _seguir)
public HiloMonticuloMinDinamico(MonticuloMinGraficoDinamico _monticuloMinGraficoDinamico, String _accion, ElementoGraficoMonticuloDinamico _elemento, int _altura, int _velocidadAnimaciones, int _maximo, boolean _seguir)
public HiloMonticuloMinDinamico(MonticuloMinGraficoDinamico _monticuloMinGraficoDinamico, String _accion, boolean _exito, int _velocidadAnimaciones)
public void finish( )
public void run( )

Figura 6.6. Diagrama de clases

Siguiendo la lectura del diagrama de clases, la clase principal es la clase VentanaMain. Esta clase constituye el núcleo del programa, pues desde ella serán llamadas muchas de las otras clases presentes.

Como puede comprobarse, en la clase VentanaMain se encuentran conexiones a las clases VentanaMonticuloMax y VentanaMonticuloMin. Aparece un uno en la línea de la que une las clases. El uno de la figura, aquí y en otras líneas, representa la relación (en este caso, sólo puede haber un objeto de la clase de este tipo). Si apareciese 1..n, significa que muchos objetos de la clase pueden estar relacionados con sólo uno de la otra clase (por ejemplo, la clase AnalizadorLexicoJava puede reconocer varios tókens como era de esperar).

Como se ve en el diagrama, las clases VentanaMonticuloMax y VentanaMonticuloMin heredan de GeneradorVentanaEstructurasDatos (si las clases heredan de otra, la clase hija es el inicio de una flecha, y la padre su final). Las clases VentanaMonticuloMax y VentanaMonticuloMin tienen cada una la oportunidad de abrir varias pestañas del tipo apropiado: VentanaMonticuloMax abre pestañas PestañaMonticuloMax y VentanaMonticuloMin abre pestañas PestañaMonticuloMin. Ambas clases, PestañaMonticuloMax y PestañaMonticuloMin heredan de la clase GeneradorPestaña. La clase VentanaMonticuloMax también contiene la representación de un montículo de máximos como se puede apreciar; en este caso, se trata de ImplementacionMonticuloMax. De la misma forma, la clase VentanaMonticuloMin también posee la implementación del montículo de mínimos como ImplementacionMonticuloMin. La clase ImplementacionMonticuloMax implementa la interfaz MonticuloMax, y la clase ImplementacionMonticuloMin implementa la interfaz MonticuloMin. Ambas interfaces heredan de la interfaz Monticulo, que define un montículo genérico.

Para la representación gráfica, existen las clases MonticuloMaxGrafico (especializada por VentanaMonticuloMax) y MonticuloMinGrafico (especializada por VentanaMonticuloMin). MonticuloMaxGrafico está relacionada con MonticuloMaxGraficoEstatico, MonticuloMaxGraficoDinamico y MonticuloMaxGraficoUsuario, que son las clases encargadas de dibujar los montículos en cada una de las vistas de la herramienta. De la misma forma, MonticuloMinGrafico está relacionada con MonticuloMinGraficoEstatico, MonticuloMinGraficoDinamico y MonticuloMinGraficoUsuario, que cumplen la misma función que sus homólogas de máximos. Cada una de las clases que especializan MonticuloMaxGrafico y MonticuloMinGrafico, existen una serie de

clases encargadas del dibujo correcto en la pantalla de los montículos. Son éstas HiloMonticuloMaxUsuario, HiloMonticuloMaxEstatico e HiloMonticuloMaxDinamico para las animaciones a la hora de dibujar los montículos de máximos en cada una de sus tres vistas, e HiloMonticuloMinUsuario, HiloMonticuloMinEstatico e HiloMonticuloMinDinamico para el mismo propósito de antes, salvo que para montículos de mínimos.

La clase encargada de los tests, VentanaTest, está incluida en VentanaMonticuloMax y VentanaMonticuloMin. También en estas clases se encuentran incluidas AnalizadorSintactico y AnalizadorSintacticoJava, así como AnalizadorLexico y AnalizadorLexicoJava. AnalizadorLexico se encarga de hacer un análisis léxico del archivo de Maude, mientras que AnalizadorSintactico se encarga del análisis sintáctico del fichero. De igual forma, AnalizadorLexicoJava se encarga del análisis léxico del fichero de Eclipse (Java), y AnalizadorSintacticoJava del análisis sintáctico del fichero. Para poder realizar el análisis léxico de los archivos, se necesita una clase que reconozca los tókens. En este caso, se ha representado la clase TokenJava en el diagrama, que es la encargada de realizar la tarea en Java.

Por último, VentanaInforme está unida a las clases VentanaMonticuloMax y VentanaMonticuloMin, pues estas últimas son las encargadas de llamarla. VentanaInforme está unida a la clase Informe, llamada por la clase anterior para poder generar el informe. Y como puede apreciarse, también esta última clase está relacionada con VentanaMonticuloMax y VentanaMonticuloMin.

6.1.3.3 Distribución de las clases

Las clases se distribuyen en los siguientes paquetes:

***ventanas:** es el paquete en el que se encuentran las clases principales que muestran por pantalla la actividad que se está desarrollando con la herramienta Vedya.*

Incluye las clases MainProyecto, VentanaMain, VentanaTest, VentanaGuardarSalir, VentanaGuardarSalirIndividual, GeneradorMenuTest, VentanaAyuda, VentanaBBDD, VentanaDatos, VentanaElementoAsociado, VentanaError2, VentanaPanelError, VentanaPregunta, VentanaSeleccionar, VentanaVerPregunta.

ventanas.estructurasDatos: *este paquete contiene el esqueleto de las posteriores pantallas que van a aparecer cuando se use el sistema. Contiene las clases GeneradorVentanaEstructurasDatos, PanelDatos, GeneradorPestaña, GeneradorMenu y GeneradorBarraHerramientas.*

ventanas.estructurasDatos.ventanas: *este paquete contiene las clases que se encargan de visualizar la ventana en la cual se visualizarán después las estructuras de datos y los tests del sistema. Incluye las clases VentanaMonticuloMax y VentanaMonticuloMin.*

ventanas.estructurasDatos.pestañas: *este paquete es el encargado de permitir mostrar los gráficos del sistema, las preguntas de los tests,etc., porque constituye la parte sobre la cual se dibujan todas las partes del sistema concernientes a la estructura de montículos. Incluye las clases PestañaMonticuloMax y PestañaMonticuloMin.*

utilidades: *este paquete contiene algunas clases de utilidad para la herramienta, como puede ser los botones de la parte derecha de la ventana principal de Vedya de operaciones. Contiene las clases Pareja, PanelRecorrido, JPanelRespuesta, JPanelPregunta, JPanelGradiente, JLabelGradiente, JButtonGradiente, FiltroURL, FileViewVEDYA y FileChooserPreviewVEDYA.*

utilidades.estructurasDatos: *es el paquete encargado de permitir las acciones de las animaciones que se pueden ver en la herramienta. Incluye las clases AccionMonticulo, AccionMonticuloMax y AccionMonticuloMin.*

implementacion.estructurasDatos.interfaces: *es el paquete encargado de proporcionar las funciones básicas para la implementación de los montículos. Contiene las clases Monticulo, MonticuloMax y MonticuloMin.*

implementacion.estructurasDatos.excepciones: *es un paquete que contiene excepciones definidas "ad hoc" para la herramienta. Contiene las clases EmptyException, FullException y OrdenException.*

implementacion.estructurasDatos: *es el paquete en el que se encuentran la implementación de los montículos para poder ser utilizados por la herramienta (se trata de una implementación que usa la herramienta, no de la que deberá ser después implementada). Incluye las clases ImplementacionMonticuloMax e ImplementacionMonticuloMin.*

graficos.estructurasDatos.usuario.elementosGraficos: *este paquete contiene aquellas clases necesarias para la visualización de los elementos de los montículos en la vista de usuario (como las bolas o las rayas negras que dibujan el árbol). Contiene las clases NexoPadreHijo y ElementoGraficoMonticuloUsuario.*

graficos.estructurasDatos.usuario: *este paquete contiene las clases encargadas de dibujar el gráfico que representa a los montículos en la vista de usuario, y crea los hilos que después serán utilizados en las animaciones. Contiene las clases MonticuloMaxGraficoUsuario y MonticuloMinGraficoUsuario.*

graficos.estructurasDatos.estatica.elementosGraficos: *este paquete contiene los elementos que se van a mostrar en la visión gráfica de los montículos (en este caso, las bolas). Contiene la clase ElementoGraficoMonticuloEstatico.*

graficos.estructurasDatos.estatica: *este paquete contiene las clases encargadas de dibujar el montículo en su visión estática, así como de crear los hilos que después ejecutarán las animaciones. Contiene las clases MonticuloMaxGraficoEstatico y MonticuloMinGraficoEstatico.*

graficos.estructurasDatos.dinamica.elementosGraficos: *este paquete contiene aquellos elementos necesarios para poder dibujar los montículos en la visión dinámica de los montículos (como las bolas, las flechas, indicador de primer elemento o el final de la estructura). Contiene las clases ElementoGraficoMonticuloDinamico, JLabelFlecha, JLabelFlecha45Grados, Puntero y PunteroMonticulo.*

graficos.estructurasDatos.dinamica: *este paquete contiene las clases encargadas de dibujar el montículo en su visión dinámica, y también de crear los hilos que ejecutarán las animaciones para esta vista. Contiene las clases MonticuloMaxGraficoDinamico y MonticuloMinGraficoDinamico.*

graficos.estructurasDatos: *es el paquete en el que se encuentran la generalización de las clases que crean los gráficos de todas las vistas de la aplicación, y que han de ser heredadas para poder cumplir con su cometido en estas tres vistas. Contiene las clases MonticuloMaxGrafico y MonticuloMinGrafico.*

graficos: *este paquete contiene las clases para la correcta visualización de los elementos que forman los elementos de los montículos (las bolas) en las tres vistas de la aplicación. Contiene las clases ElementoGrafico y JLabelElemento.*

generarEstadisticas: *es el paquete en el que se encuentran la clase Estadistica, cuyo propósito es permitir al alumno conocer sus estadísticas tras realizar el test y permitirle hacerse una idea de su evolución durante el uso de la herramienta Vedya. Genera una ventana que ofrece los resultados (éstos serán grabados en el informe cuando éste se genere).*

generadorInforme: *es el paquete en el que se encuentran todo lo referente al informe que será generado después para poder llevar a cabo un estudio sobre la aplicación y cómo influye en el estudio. Contiene las clases Datos, Informe y VentanaInforme.*

comprobarUsuario: *es el paquete en el que se encuentran las clases necesarias para la correcta identificación del usuario que va a usar el programa. Debido a que el informe que se va a generar se refiere a un alumno en particular, su identificación es muy importante. Contiene las clases datosUsuario y leerFichero.*

animacion.estructurasDatos.usuario: *es el paquete en el que se encuentran las clases encargadas de ejecutar las animaciones que se pueden*

ver en la vista de usuario de los montículos. Contiene las clases HiloMonticuloMaxUsuario y HiloMonticuloMinUsuario.

animacion.estructurasDatos.estatica: *es el paquete en el que se encuentran las clases encargadas de ejecutar las animaciones que se pueden ver en la vista estática de los montículos en la herramienta. Contiene las clases HiloMonticuloMaxEstatico y HiloMonticuloMinEstatico.*

animacion.estructurasDatos.dinamica: *es el paquete en el que se encuentran las clases encargadas de ejecutar las animaciones que se pueden ver en la vista dinámica de los montículos en la herramienta. Contiene las clases HiloMonticuloMaxDinamico y HiloMonticuloMinDinamico.*

animacion.estructurasDatos: *es el paquete en el que se encuentran las clases que permiten otras animaciones auxiliares que no sean las referidas a los montículos, que constituyen las animaciones principales (por ejemplo, mover la flecha que aparece en la parte derecha de la ventana de montículos). Contiene las clases HiloAvanzaRegistro y HiloPanelDatos.*

animación: *es el paquete en el que se encuentra la clase responsable de la animación que aparece al principio (el movimiento de los botones al principio). Contiene la clase HiloPosicionBoton.*

test: *es el paquete en el cual están almacenadas todas aquellas clases que contribuyen a que Vedya incorpore ahora la funcionalidad de poder plantear tests para ser resueltos por los alumnos. Incluye las clases BaseDePreguntas, Pregunta, Respuesta y Test.*

traductorMaude.tipos: *es el paquete en el que se encuentran las definiciones de tipos que van a ser empleados en el procesamiento de los textos escritos en Maude, y que nombran a las categorías léxicas y sintácticas adoptadas para el análisis del archivo. Incluye las clases TipoError, TipoLiterales y TipoToken.*

traductorMaude.tablaSimbolos: *es el paquete en el cual se encuentran las clases que permiten guardar información sobre las variables y tipos declarados*

en Maude (que sean correctas estas declaraciones). Incluye las clases Id y TablaSimbolos.

traductorMaude.analizadorLexico: *es el paquete en el cual se encuentra el analizador léxico del procesador de lenguaje que se encarga de verificar que el archivo de Maude se encuentra bien escrito. En particular, se encarga de verificar que no existan errores léxicos (nombres equivocados de variables o métodos, variables no declaradas, etc.). Contiene las clases AnalizadorLexico, Estados y Token.*

traductorMaude.analizadorSintactico: *es el paquete en el que se encuentran las clases responsables del análisis sintáctico de Maude, que comprueban que las estructuras de los métodos declarados son correctas, y que los propios métodos declarados son los que debían estar declarados. Incluye las clases AnalizadorSintactico y Error.*

traductorMaude: *es el paquete que incluye a los otros y que es el encargado de pasar el analizador léxico y el sintáctico al archivo Maude para su comprobación. Contiene la clase TraductorMain, que es la encargada de verificar la corrección del archivo.*

traductorJava.tipos: *es el paquete que contiene los tipos principales que serán usados cuando el analizador tipo Java tenga quecorregir un archivo escrito a la manera de Java con la implementación de los montículos. Estos tipos se refieren a los nombres adoptados para las categorías léxicas y sintácticas. El paquete incluye las clases TipoErrorJava, TipoLiteralesJava y TipoTokenJava.*

traductorJava.tablaSimbolos: *es el paquete que contiene las clases encargadas de guardar los nombres dados a variables y métodos dentro del archivo, para después corroborar que estaban bien escritos y que no hay inconsistencias en cuanto a los nombres declarados antes y después en el mismo archivo. Incluye las clases IdJava y TablaSimbolosJava.*

traductorJava.analizadorLexico: *es el paquete que contiene las clases AnalizadorLexicoJava, EstadosJava y TokenJava, y que se encarga del análisis*

léxico del archivo que contiene la implementación de los montículos (con un comportamiento similar al orientado a Maude, pero orientado a Java).

traductorJava.analizadorSintactico: *es el paquete encargado de realizar el análisis sintáctico del archivo que contiene la implementación de los montículos en Java (y que, como en el punto anterior, se comporta de manera similar al analizador sintáctico de Maude, pero orientado a Java). Contiene las clases AnalizadorSintacticoJava y ErrorJava.*

traductorJava: *es el paquete que contiene la clase TraductorMainJava, responsable del análisis del archivo escrito en Java y que se encarga de llamar al analizador léxico y al analizador sintáctico para realizar la verificación de la corrección del archivo.*

6.2 Implementación

Se exponen a continuación las herramientas y opciones que se han elegido a la hora de implementar la herramienta que constituye el objetivo del proyecto de Sistemas Informáticos, así como una breve explicación de los módulos implementados.

6.2.1 Herramientas utilizadas para el desarrollo

Debido a que el sistema Vedya anterior estaba implementado en Java, y a la necesidad de reutilizar el código ya existente para poder conseguir el objetivo fijado en un plazo razonable, se eligió, como lenguaje de programación para realizar el proyecto, el lenguaje Java.

Hemos decidido utilizar la herramienta Eclipse para desarrollar la herramienta. Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma. Este programa típicamente ha sido usado para desarrollar entornos de desarrollo integrado (en inglés integrated development environment, IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son

usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent y Azureus.

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, cubriendo casi todas las áreas de Model Driven Engineering.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

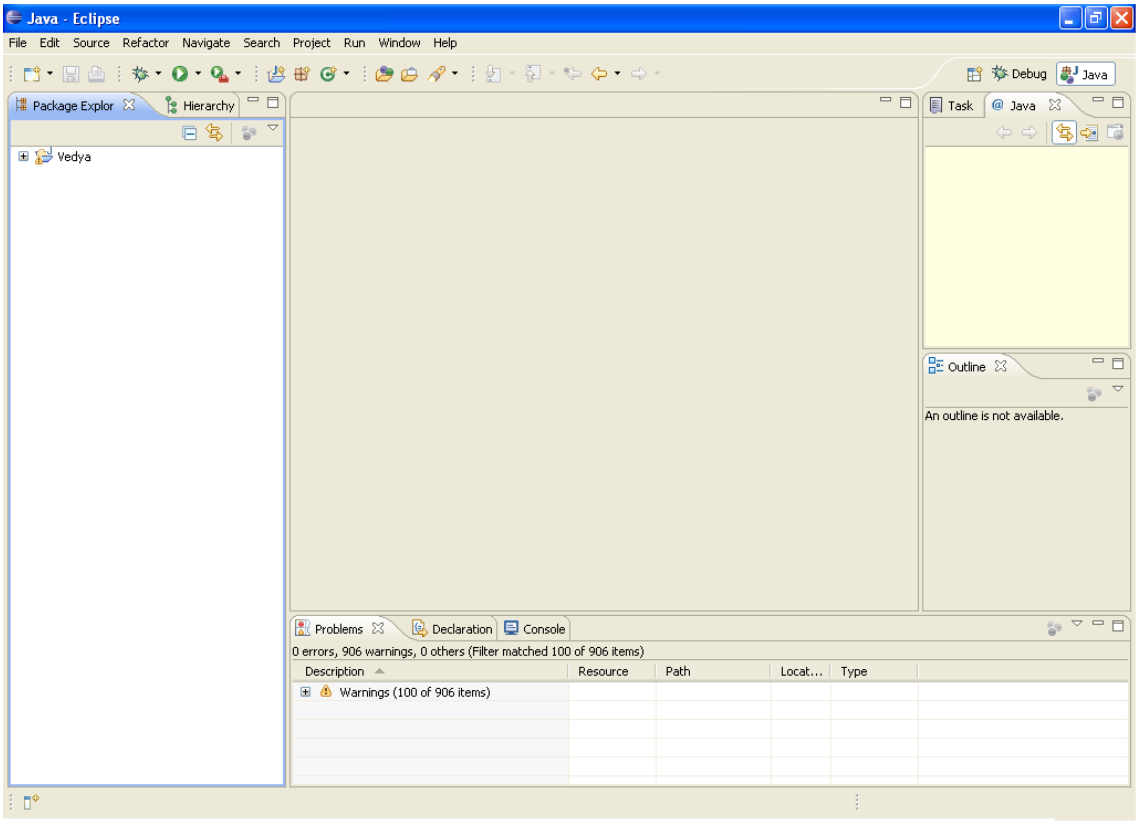


Figura 6.7. Ventana de Eclipse

También hemos utilizado Vedya-Test para poder insertar las preguntas dentro de la herramienta. Vedya-Test es un programa desarrollado anteriormente en la facultad, que facilita el poder hacer preguntas para los tests y presentar éstas de forma muy visual y comprensible. Tiene como ventaja su

gran facilidad de uso, y hay que decir que esta herramienta ha sido utilizada para introducir los tests en la herramienta sin que ello supusiese mucho esfuerzo.

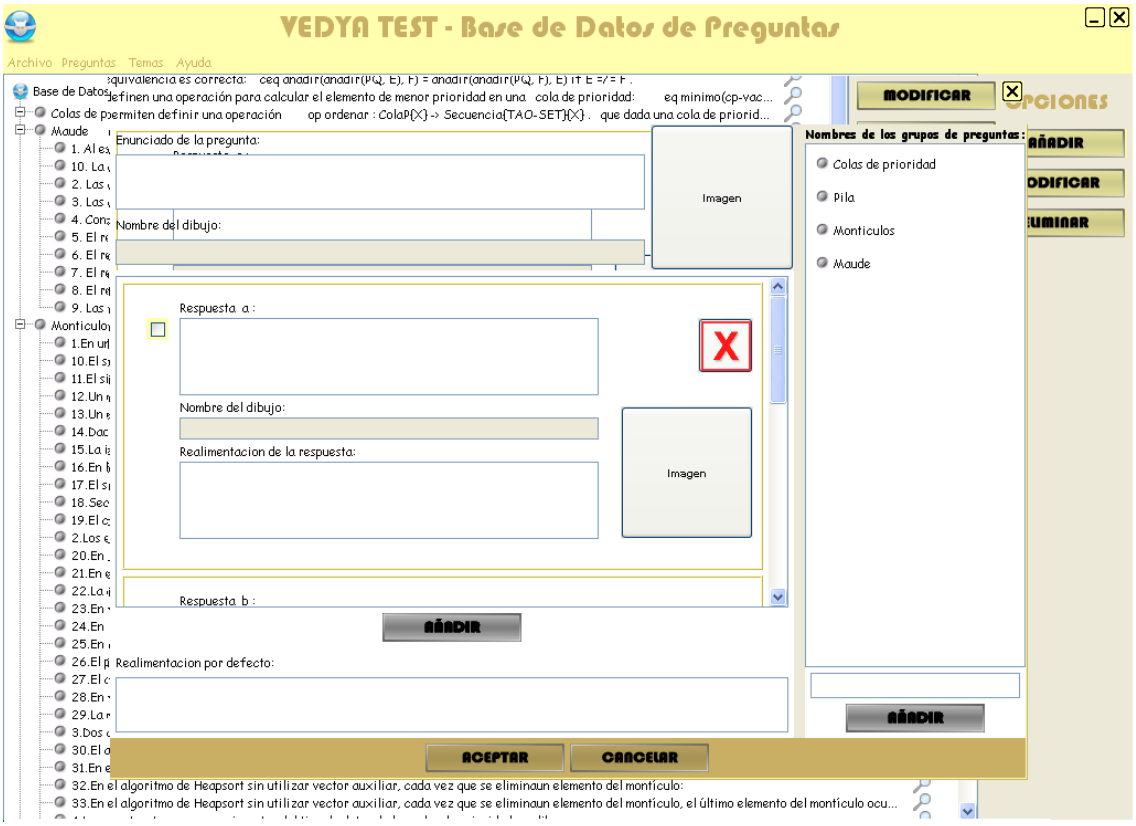


Figura 6.8. Ventana de Vedya-Test

6.2.2. Ventana de login y selección

Ésta es la primera ventana de la aplicación, la que aparece en primer lugar cuando se ejecuta el programa. Se trata de una pantalla que contiene dos campos: uno primero, en el que el alumno deberá escribir su DNI, o el número del documento que le identifique; y uno segundo, que está preparado para que el alumno escriba su contraseña (son válidas tanto EDI10A como EDI10B).

Figura 6.9. Ventana de login de la aplicación

Después de rellenar estos datos, el alumno pasa a pulsar el botón Aceptar, que se encuentra en la misma pantalla. Una vez pulsado este botón, el sistema comprueba que el alumno esté dentro del conjunto de alumnos que están cursando la asignatura de EDI en el curso 2009-2010. Si no lo encuentra, o la contraseña introducida no es válida, el sistema devuelve una ventana de error e impide el acceso al sistema. Por el contrario, si el alumno pertenece al conjunto anterior, y la contraseña es correcta, el sistema muestra otra ventana, llamada ventana de selección.

En esta ventana, el alumno tiene la posibilidad de escoger entre la posibilidad de experimentar con los montículos de máximos o de mínimos. Realizada cualquiera de estas posibilidades mediante la pulsación del botón correspondiente, el sistema pasa a la siguiente ventana.

Figura 6.10. Ventana de selección

6.2.3. Diseño de la ventana principal de la aplicación

La ventana principal de la aplicación es la parte fundamental de todo el proyecto. Su objetivo es permitir a los alumnos interactuar directamente con la herramienta.

La ventana está dividida en 3 paneles. Además, tiene en su parte superior un menú y una barra de herramientas, todas ellas con opciones (algunas implementadas; pero otras, de momento, no tienen utilidad).

Centrándonos en el panel de la izquierda, constatamos que existen diferentes botones. El objetivo de estos botones es poder crear y manipular los montículos, con el objetivo de insertar, borrar, preguntar por la altura y el máximo/mínimo de los elementos que están en él, y poder averiguar si está

vacío (todas estas operaciones fueron descritas y utilizadas en EDI cuando cursamos la asignatura).

El panel central es el corazón de toda la aplicación. En él se van a visualizar: la estructura, las animaciones que darán lugar a la estructura y los distintos tests. Por lo tanto, contiene toda la información útil que se le puede proporcionar a un alumno.

El panel de la derecha contiene todas las operaciones ejecutadas hasta el momento. Su importancia radica en que el alumno puede, de un vistazo, ver cuántas operaciones ha realizado y poder determinar si ya ha empleado la herramienta lo suficiente o necesita comprobar el comportamiento de alguna de las funciones durante más tiempo.

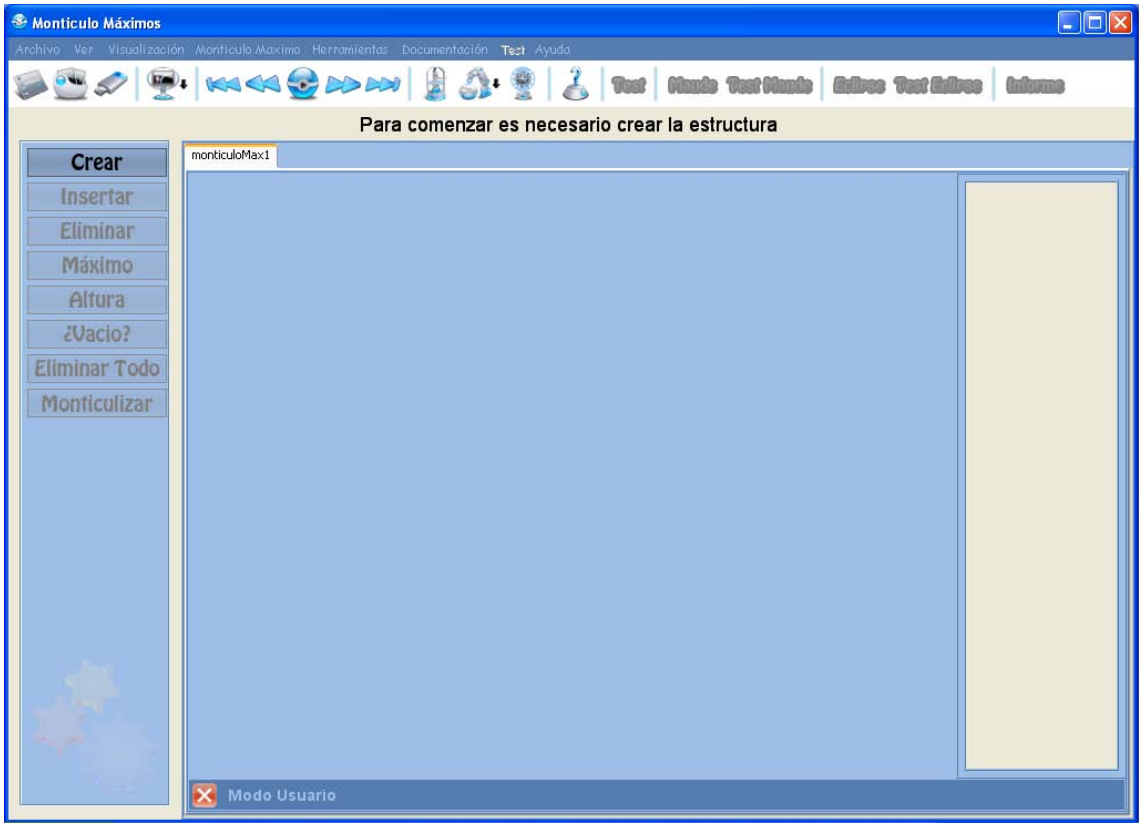


Figura 6.11. Ventana principal de la herramienta

6.2.4. Diseño de las pestañas: montículos y tests

Las pestañas son, realmente, la parte esencial de la parte gráfica de la aplicación. Contienen el dibujo de los montículos que son dibujados sobre estas pestañas.

También contienen los tests que se realizan a los alumnos.

Figura 6.12. Una pestaña de la aplicación

6.2.5. Diseño de la ventana de estadísticas

Cuando en la vista de test se pulsa el botón Corregir, aparece una ventana adicional. Esta ventana proporciona una información de primera mano sobre

los resultados que ha conseguido un alumno en el test, y los logros conseguidos trabajando en la modalidad de montículos.

Presenta en su parte superior un resumen de las veces en las que el alumno ha hecho uso de las funciones que están disponibles para montículos, y en su parte inferior, una relación de las preguntas acertadas y falladas al realizar el test.

La información, presentada de esta forma, proporciona al alumno una visión muy útil sobre su propia evolución en el estudio de los montículos, las funciones que domina y aquellas en las que debería centrarse más y dedicarle más tiempo.

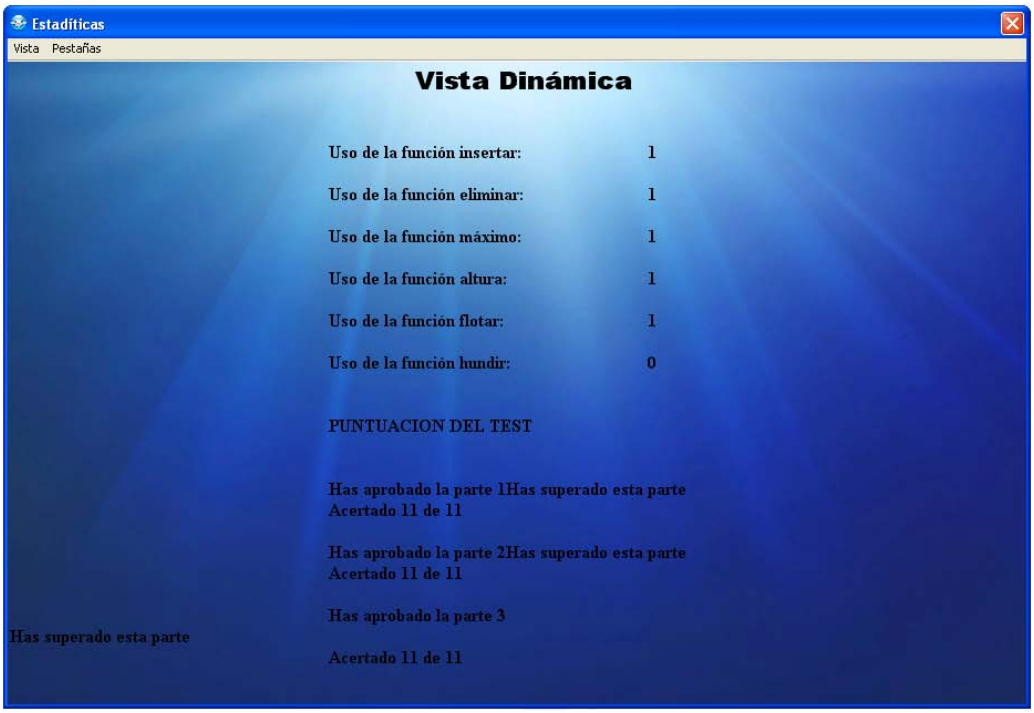


Figura 6.13. Ventana de estadísticas

Existe otra ventana para el caso de que se haya realizado el test que viene después y que está centrado en Maude. No obstante, su diseño es similar a éste.

6.2.6. Diseño del procesador de Maude

El procesador de Maude está compuesto de dos módulos principalmente, un analizador léxico y un analizador sintáctico. El analizador léxico se encarga de comprobar que no existen problemas a la hora de escribir los métodos que se emplean con Maude, ni existen problemas con los nombres de las variables, y el analizador sintáctico se encarga de comprobar que todas las construcciones que deben estar en la forma en la que se escribirían en Maude, se encuentran efectivamente así.

El analizador léxico y sintáctico se pasan a la vez. Así, el analizador sintáctico puede comprobar casi inmediatamente que no ocurran problemas después de pasar el analizador léxico. Para poder llevar a cabo su trabajo, el analizador léxico se sirve de una tabla de símbolos, que recoge todos los nombres declarados en el archivo que han de servir como nombres de identificadores. La tabla de símbolos almacena estos nombres, para su comprobación durante todo el tiempo que dure el proceso de traducción (durante todo el tiempo, puesto que el analizador léxico y el sintáctico, como se ha dicho antes, se pasan a la vez).

El traductor de Maude define, asimismo, una cantidad de tipos y errores que habrán de ser utilizados por el traductor en su tarea. Estos tipos, que permiten trabajar con naturales, booleanos, caracteres, números en coma flotante, también permiten que se generen errores en la traducción del código.

Como ya se ha comentado más arriba en el texto, este traductor de Maude no pretende reemplazar a la herramienta original, sino que su objetivo es introducir a los alumnos en el mundo de Maude y permitirles hacerse una idea de cómo hay que hacer las especificaciones en este lenguaje, para poder enfrentarse después con mayor acierto a una especificación "de verdad". Por esto, el traductor no comprueba cualquier cosa que esté escrita en Maude, sino sólo aquellos archivos que estén escritos de la forma prescrita.

Para el objetivo que nos hemos fijado este año en la asignatura de Sistemas Informáticos, es suficiente con que los alumnos definan el tipo de montículos mínimos (porque es el único que es capaz de reconocer el traductor de Maude).

Después, hay que definir las siguientes operaciones:

- *cp-vacia: esta es la función que define la cola de prioridad vacía.*
- *insertar: esta función permite introducir un elemento dentro de la cola de prioridad, en el lugar correspondiente.*
- *eliminar: esta función permite eliminar el elemento más prioritario de la cola.*
- *vacia: permite averiguar si la cola de prioridad está vacía o no.*
- *minimo: devuelve el elemento más prioritario de la cola de prioridad.*

Con la definición de todas estas operaciones, se consideraría correcto el ejercicio.

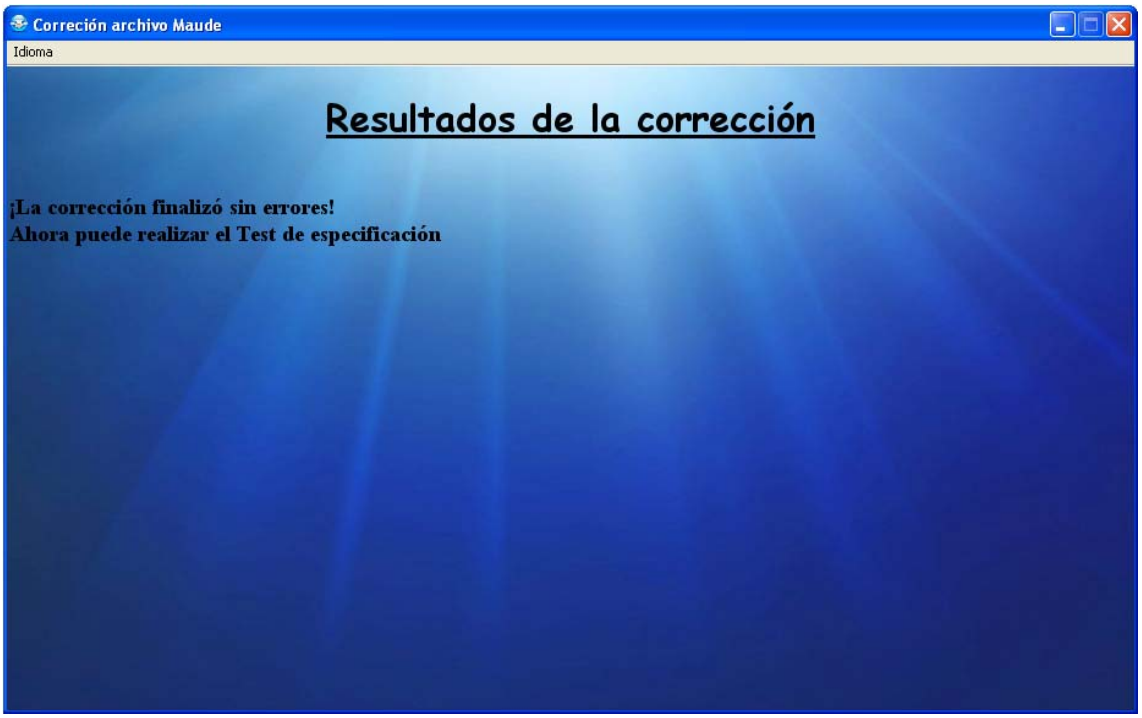


Figura 6.14. Ventana de resultados de Maude

6.2.7. Diseño del procesador de Java

El procesador de Java, al igual que el anterior de Maude, está compuesto de dos módulos, un analizador léxico y un analizador sintáctico. El analizador léxico se encarga de comprobar que no existen problemas a la hora de escribir los métodos que se emplean en Java, ni existen problemas con los nombres de las variables, y el analizador sintáctico se encarga de comprobar que todas las construcciones que deben estar en la forma en la que se escribirían en Java, se encuentran efectivamente así.

Como ocurría antes, el analizador léxico y sintáctico se pasan a la vez, y ambos funcionan igual que en el caso de Maude, aunque con la salvedad de que cada uno ha de comprobar aspectos distintos.

El traductor de Java define, como anteriormente sucedía con Maude, unos determinados tipos y errores que habrán de ser utilizados por el traductor en su tarea. Estos tipos, que permiten trabajar con naturales, booleanos, caracteres, números en coma flotante, también permiten que se generen errores en la traducción del código.

Como ya se comentó anteriormente, este traductor no es un compilador, sino más bien un orientador que indica los problemas que se producirían al intentar programar la estructura de los montículos en Java, y que podría resultar en que aunque el código de las funciones fuese correcto, el archivo final contuviese errores. Estos errores, si bien no serían léxicos ni sintácticos, sí pueden representar errores de concepto. Tales errores son detectados por el traductor, que avisa al usuario de que dichos fallos han sido encontrados y de que necesita repararlos.

Ocurre aquí, al igual que antes, que los montículos de mínimos han sido elegidos para el programa. Por lo tanto, todas las operaciones que deban ser definidas, estarán referidas a esta estructura y no a la correspondiente de máximos.

Después, hay que definir las siguientes operaciones:

- *void cp-vacia(): esta es la función que define la cola de prioridad vacía.*
- *void insertar(int t): esta función permite introducir un elemento t dentro de la cola de prioridad, en el lugar correspondiente.*
- *void eliminar-min(): esta función permite eliminar el elemento más prioritario de la cola.*

- *boolean es-cp-vacia(): permite averiguar si la cola de prioridad está vacía o no.*
- *int minimo(): devuelve el elemento más prioritario de la cola de prioridad.*
- *void flotar(int t): flota un elemento, desde el fondo del montículo, y lo coloca en su posición.*
- *void hundir(int t): hunde un elemento desde la raíz del montículo hasta el lugar en el que debe encontarse el elemento.*

La definición de estos métodos debe incluir una devolución de errores en el caso en que este error hubiese sido declarado en la especificación. Si no hay ningún problema, y tras la definición de estas funciones, el ejercicio se consideraría correcto.

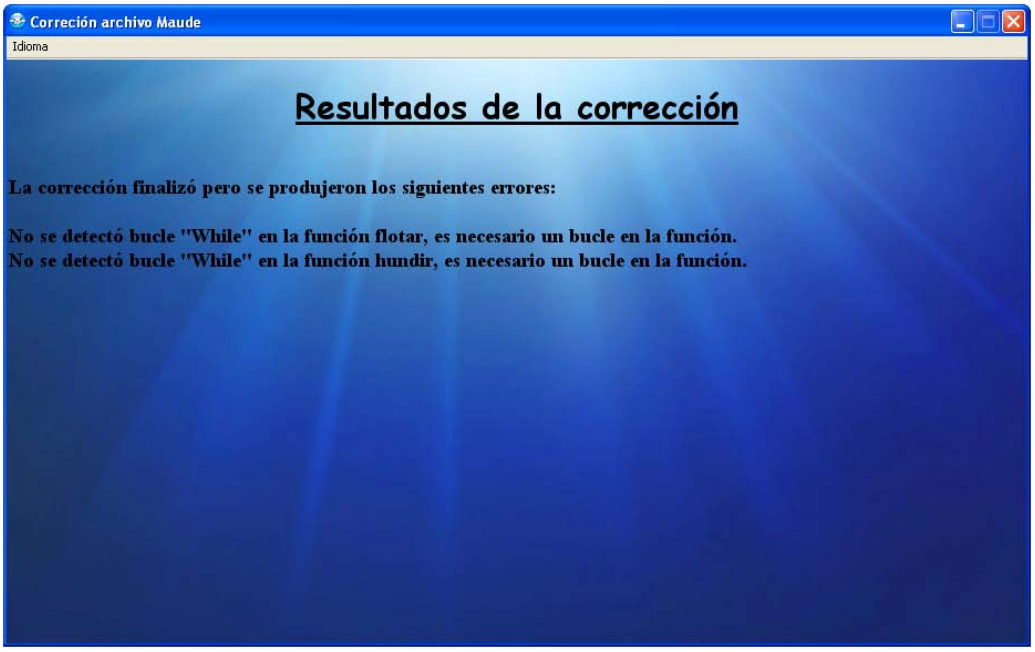


Figura 6.15. Ventana de resultados de Java

6.3 Despliegue del proyecto

Para realizar el despliegue del proyecto, se ha optado por hacer que los alumnos lo puedan descargar y usar. Como el informe que se va a generar está codificado, los datos que se deben enviar al profesor no pueden ser alterados. Así, aunque no se haga un despliegue del proyecto en Internet, sí que los datos van a permanecer inmutables.

El proyecto que puede ser descargado incluye un archivo .jar ejecutable. Este .jar es el proyecto en sí, al cual deben acompañar todas las imágenes que sean necesarias para la correcta visualización del programa.

6.4 Logros y limitaciones

6.4.1. Logros

Entre los logros alcanzados en este proyecto de Sistemas Informáticos, podemos mencionar el que es, acaso, el más importante: convertir una herramienta que era un mero expositor de lo que son estructuras de datos, en una importante herramienta al servicio de la educación de las estructuras llamadas montículos.

Para conseguirlo, se ha cambiado y adaptado mucho trabajo hecho en años anteriores. En cuanto a los resultados del proyecto, ha quedado terminado todo lo que corresponde a la parte gráfica de los montículos. Ciertamente, el aspecto del sistema es muy parecido al del sistema Vedya anterior; pero en líneas generales, y refiriéndonos sólo a la funcionalidad del sistema, la parte gráfica está completamente terminada.

La inclusión dentro del proyecto de los tests también está realizada.

El sistema ofrece, siempre que se pulse el botón de corregir de los tests o el botón de generar informe, un resumen de los resultados obtenidos por el alumno hasta el momento en que pulsó. Consideramos que esta información puede resultar de ayuda al alumno a la hora de comprender qué partes entiende y cuáles deberían ser tomadas más en cuenta.

El traductor de Maude ha quedado completamente terminado, aunque es sólo capaz de reconocer montículos de mínimos. Es capaz de reconocer

completamente construcciones pertenecientes a Maude, e indicar los problemas que existen en cuanto al objetivo perseguido por el proyecto.

El traductor de Java (para la implementación de los montículos en Java) también está acabado; aunque al igual que antes, nos encontramos con que sólo es capaz de reconocer montículos de mínimos. Es capaz de reconocer muchas construcciones del lenguaje Java y determinar si lo escrito podría aceptarse como una buena implementación de montículos.

6.4.2 Limitaciones

La principal limitación consiste en que la herramienta sólo funciona con los montículos. El resto de estructuras de datos no entran dentro de la herramienta, lo cual limita su funcionalidad a una sola estructura.

Siendo más explícitos, podemos numerar las siguientes limitaciones:

- *Montículos. La herramienta sólo es capaz de trabajar con estas estructuras. Las demás estructuras de datos (pilas, colas, árboles…) quedan fuera de la herramienta. Constituiría una interesante ampliación el permitir que todas las estructuras de datos que son estudiadas en la asignatura de EDI pudiesen ser incluidas en la herramienta.*
- *Montículos de mínimos. Tanto el traductor de Maude como el de Java sólo trabajan con montículos, y dentro de esta estructura, sólo pueden trabajar con montículos de mínimos. Una posible ampliación consistiría en permitir trabajar con los dos tipos de montículos, no sólo con uno de ellos. Y también el ampliar las posibilidades de reconocimiento de estructuras de datos a las demás, para poder trabajar con traductores en ambas partes y con todas las estructuras de datos.*
- *Rigidez en las preguntas. Debido a que su intención es dejar muy claros ciertos conceptos, los tests sólo presentan preguntas relacionadas con ellos, impidiendo cualquier otro aspecto educativo. Pensamos que una solución sería poder tener varios tests y que se cargasen aleatoriamente, o bien tener tests que pudiesen cargar preguntas aleatoriamente (siempre dentro de los límites del tema que estos tests tratan). Así, aunque hubiese que repetir el test, éste no sería tan previsible, sino que introduciría novedades (para que los alumnos no respondan "de memoria").*

- *Test de la parte de eclipse. Este test completaría los tests necesarios para que los alumnos de EDI pudieran completar su conocimiento de montículos. Sin embargo, como no se enseña Java en la asignatura de EDI, no se incluye este test.*
- *Rigidez en el flujo de las pruebas. Las pruebas se han pensado para ser realizadas en un determinado orden, y éste es inmutable. Tal vez podría modificarse de tal forma que no resultase un sistema tan rígido, y permitiese más comunicación entre el alumno y el sistema.*
- *Botones de los tests. Estos botones se pueden pulsar varias veces, lo cual es un problema, puesto que sólo deberían ser pulsados una o dos veces, y nunca dos o más consecutivas, debido a que sólo deseamos tener un test a la vez. Sería muy interesante, de cara a posibles expansiones posteriores, que se solucionase este problema.*
- *Ayuda interactiva. El sistema no ofrece en ningún momento ningún tipo de ayuda interactiva, ni ventanas ni etiquetas que pudiesen ayudar a los estudiantes. De cara a posibles expansiones de la herramienta, se podría pensar en incluir algún tipo de ayuda que fuese capaz de indicar a los alumnos aquellos puntos que deberían ser tenidos más en cuenta o que necesita repasar. Pensamos que este tipo de ayuda lo constituirían las etiquetas (pequeños fragmentos de texto que aparecen al pasar el ratón por encima de un botón o una parte de la pantalla), y ventanas de información (que saldrían y le dirían al alumno qué es lo que tiene que hacer al principio de cada una de las partes del sistema).*

Con ello, creemos que el alumno siempre tendría presente lo que tiene que hacer, al ser recordado sobre sus objetivos, resultando un uso mejor y más eficaz de la herramienta.

6.5. Expansión de la herramienta

6.5.1 Motivación

Al ser ésta una herramienta muy novedosa, y sólo habernos podido dedicar un año a ella, podría (y seguramente ocurre) suceder que la herramienta

pudiera adolecer de tener varias carencias. Por ello, se ha decidido escribir esta sección, para mostrar algunas de las características que faltan y que, tal vez, sería bueno que fuesen implementadas en caso de que esta herramienta pudiese ser, en un futuro, ampliada.

6.5.2 Requerimientos

Para poder expandir la herramienta, es necesario que se respete la máxima que ha guiado al proyecto durante todo este tiempo: de la especificación a la implementación. Para ello, todas las estructuras de datos que fuesen incorporadas, deberían seguir el mismo patrón: empezar con una introducción a las distintas estructuras de datos, seguir por un test que trate sobre la estructura aprendida, e implementar dos traductores, uno para reconocer estructuras Maude y orientado a la especificación, y otro para reconocer estructuras de Java y orientado a la implementación. Además, deberían existir sendos tests dedicados a cada una de las dos partes.

Para ello:

- *La parte gráfica de la aplicación puede ser ampliada con más estructuras de datos que permitan un estudio más amplio de dichas estructuras.*
- *Los módulos de traducción de Java y Maude podrían ser ampliados para poder permitir un mayor número de estructuras de datos (incluyendo el caso, ya citado, de poder permitir que entren también los montículos de máximos y no sólo los de mínimos). Sería aconsejable que se mantuviese la estructura que tienen hoy estos dos módulos.*
- *Como no hay posibilidad actualmente de cambiar los tests, una posible ampliación sería añadir un módulo que pudiese elegir aleatoriamente entre distintos tests (pero manteniendo siempre un orden dentro de ellos que permitiese hacer una correcta evaluación). Así, el alumno no tendría que enfrentarse siempre al mismo test.*
- *Las ventanas que reflejan el estado del trabajo realizado por los alumnos deberían conservar sus estructuras actuales, es decir, que reflejen los resultados de los tests realizados las ventanas que aparezcan tras pulsar el botón de corregir en los tests, y que aparezca la ventana con información general cuando sea pulsado el botón de generar informe.*

➢ *Por falta de tiempo, como se indicó más arriba, no es posible trabajar con la herramienta mediante servicios web. Una interesantísima ampliación de la herramienta estaría constituida por la posibilidad de permitir conectar la herramienta a Internet, y permitir al alumno generar y enviar su informe mucho más rápidamente, y además, de forma más segura.*

6.5.3 Trabajo a realizar de cara a la expansión

La ampliación que resulta más urgente en este proyecto es la de permitir que un mayor número de estructuras de datos se beneficien del tutor inteligente. Parece, pues, que entre las primeras operaciones de expansión que se podrían realizar se encontraría la de incluir más ventanas para las otras estructuras que puedan ser añadidas. Simplemente habría que cambiar la ventana principal de la aplicación (que aparece después de la ventana de login) y añadir aquí los botones correspondientes a las otras estructuras. Después, añadir en el paquete ventanas.estructurasDatos.ventanas, así como pestañas dedicadas a las otras estructuras en el paquete ventanas.estructurasDatos.pestañas.

También se puede ampliar el repertorio de estructuras de datos que pueden ser reconocidas antes de ser procesadas por un compilador de Maude o de Java. Para ello, habría que seguir la misma estructura con que ya están escritos los dos traductores de Java y Maude ya escritos, pero adaptados a nuevas estructuras, siempre siguiendo los consejos del profesor. Además, y como ya se indicó atrás, actualmente los traductores de Maude y de Java sólo son capaces de reconocer montículos de mínimos. Podrían hacerse fácilmente otras clases que permitiesen trabajar con montículos de máximos, tomando como referencia aquellas que ya existen para montículos de mínimos.

También, y siempre en el caso de que se ampliase el repertorio de estructuras reconocidas por el sistema, podría añadirse una clase abstracta para representar las ventanas de estadísticas y del informe, permitiendo así que las ventanas específicas dedicadas a mostrar las estadísticas al alumno tuviesen todas el mismo esqueleto y sólo tuviesen que heredarlo.

7. Una demostración de la herramienta

El sistema consta de tres partes diferenciadas: una primera parte, en la cual se le presentan al alumno una pantalla para que pueda experimentar con la estructura de datos en tres vistas diferentes:

- *de usuario, con una representación en forma de árbol, que se aproxima más al enfoque clásico de los montículos (como árboles semicompletos);*
- *estática, que permite trabajar con el montículo implementado mediante el uso de una estructura estática (array) de elementos;*
- *dinámica, que permite centrarse en una visión inspirada en los punteros (como los usados en C), aunque de una forma mucho más amigable y fácil de entender para el usuario.*

La segunda y tercera parte se entrecruzan. La segunda parte consiste en una serie de tests, que le serán presentados al usuario a medida que acaba satisfactoriamente las pruebas anteriores (que podrían ser el uso de las vistas de los montículos de la primera parte, o la realización de otros tests, así como el haber demostrado suficientes conocimientos en la programación de esta estructura en Java y Maude). Estos tests están orientados al afianzamiento de los conocimientos cubiertos por los ejercicios que se hayan realizado anteriormente.

En cuanto a la tercera parte, su desarrollo es crucial para demostrar los conocimientos adquiridos en el aprendizaje de los montículos. Se trata de una pequeña sesión de programación orientada a la realización de un diseño de la estructura de montículos. Primero, se realizará una especificación en Maude; y después, una implementación en Java.

Vamos a describir los pasos que se han de llevar a cabo mientras se trabaja en el sistema Vedya. Como se ha dicho anteriormente, Vedya es un programa que incorpora un tutor inteligente. Así pues, describe una serie de etapas consideradas como de obligado conocimiento antes de poder adquirir un conocimiento más amplio de los montículos.

La herramienta se inicia con una pantalla de login (o registro), en la cual se le pide al usuario su DNI y una clave (datos que serán muy importantes después, cuando haya que tabular estadísticas).

Figura 7.1. Ventana de introducción a Vedya.

Una vez se hayan introducido los datos, y el sistema haya comprobado la corrección de los mismos, aparecerá otra pantalla, en la que el usuario deberá decidir trabajar bien con montículos de máximos, bien de mínimos.

Una vez elegido el tipo de estructura de datos, se abre esta otra pantalla.

En ella, se observan varias partes. Un menú, una barra de tareas, una hueco en la parte principal, y 2 más a los lados de éste. El de la izquierda contiene unos botones, el de la derecha está vacío. Centrémonos en los botones.

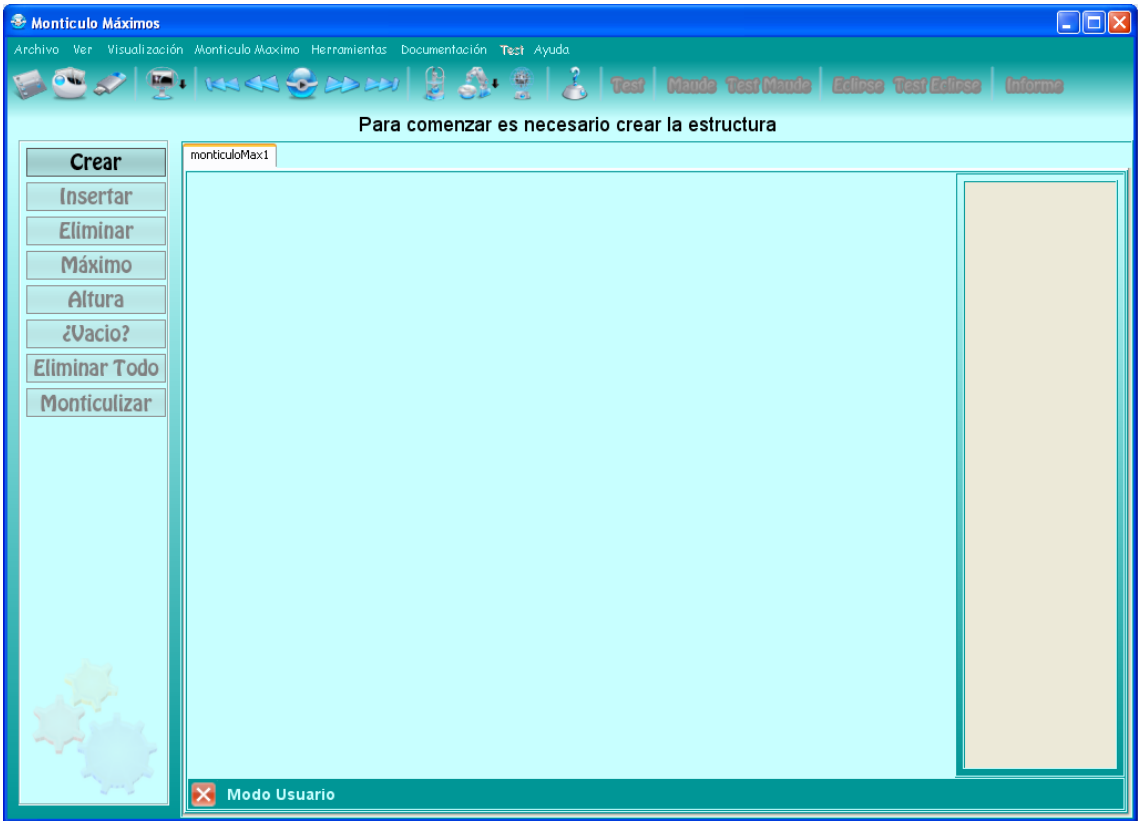


Figura 7.2. Ventana principal de la aplicación.

Los botones son las acciones que se pueden realizar con el programa para los montículos. Como se puede observar, al principio sólo un botón aparece disponible: es el botón de crear, que permite crear la estructura montículo. Una vez pulsado este botón, se iluminan los demás (salvo aquellos que no tienen sentido o no es posible utilizarlos aún porque la función no está disponible), mostrando que ya están operativos. A partir de ahora, el sistema pide lo siguiente:

- *1 ejecución de la función insertar.*
- *1 ejecución de la función eliminar.*
- *1 ejecución de la función altura.*
- *1 ejecución de la función máximo (si se está trabajando con montículos de máximos), o mínimo (si se está trabajando con montículos de mínimos).*

Esto ha de realizarse para cada una de las vistas (de usuario, estática y dinámica de los montículos).

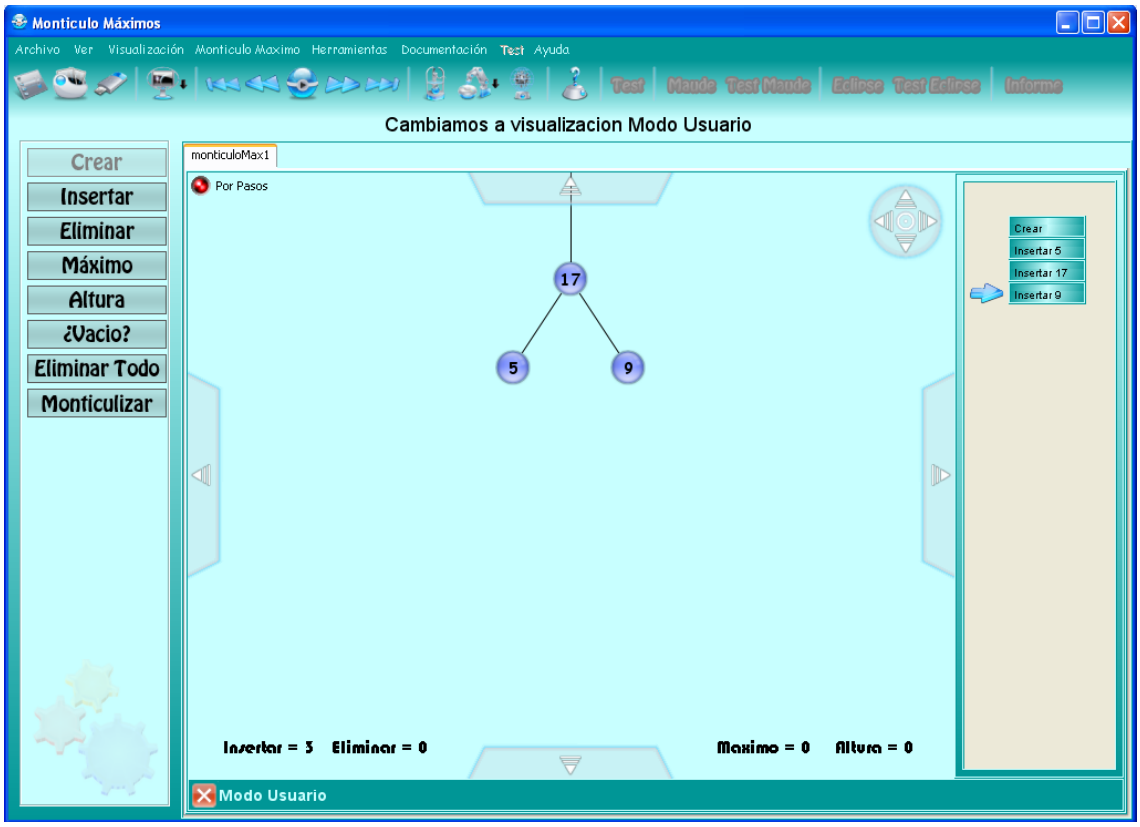


Figura 7.3. Prueba de montículos en visión de usuario.

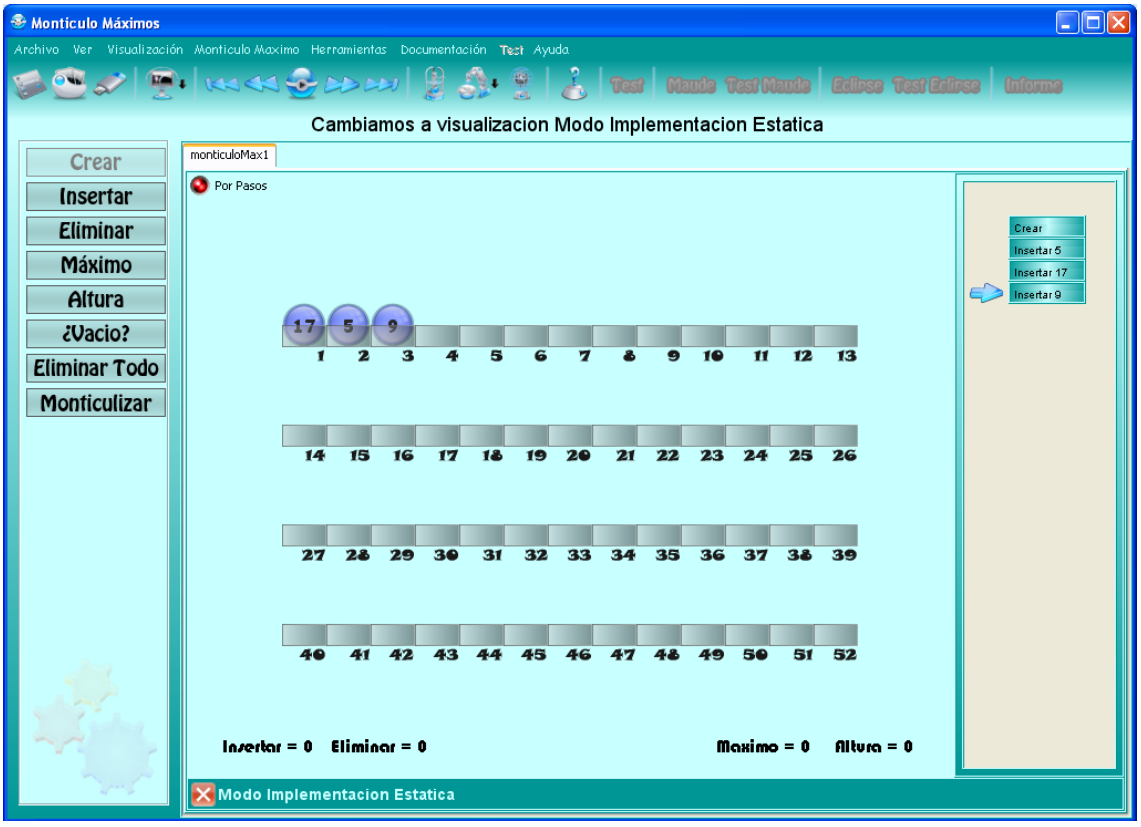


Figura 7.4. Prueba de montículos en visión estática.

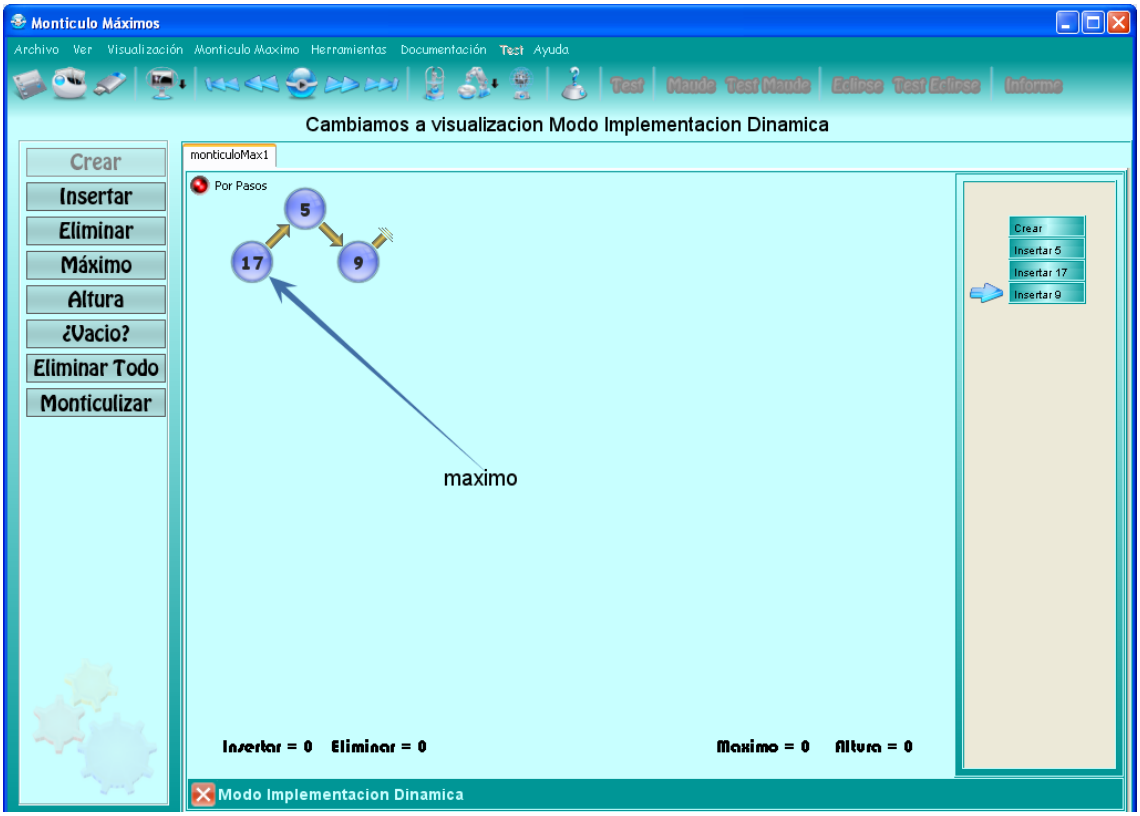


Figura 7.5. Prueba de montículos en visión dinámica.

Una vez realizado todo esto, se desbloquea un botón en la barra de herramientas que dice "Test". Éste test está compuesto por 33 preguntas, y está dividido en 3 partes de 11 preguntas cada una. Para superar el test, hace falta superar cada una de estas tres partes.

Figura 7.6. Botón de test resaltado.

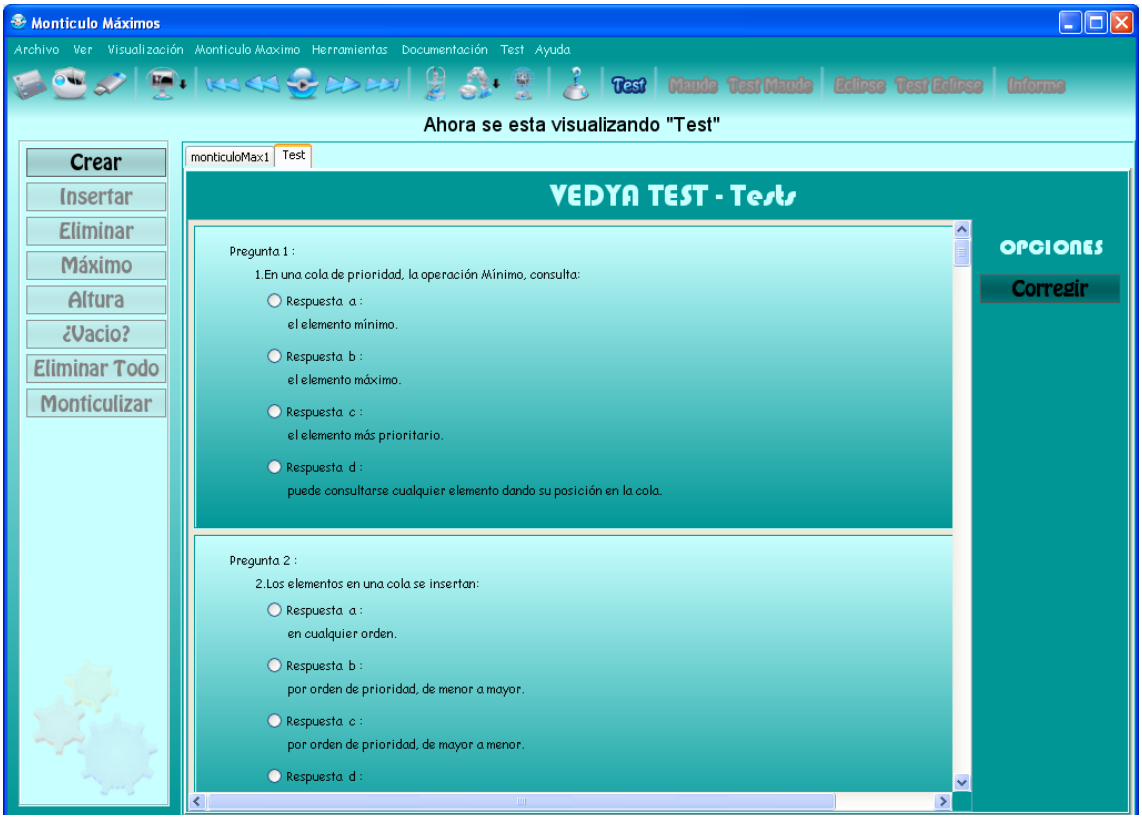


Figura 7.7. Ventana que muestra el test de montículos.

Pulsar el botón Corregir hará que aparezca una nueva ventana, con las estadísticas del test que se haya terminado de hacer.

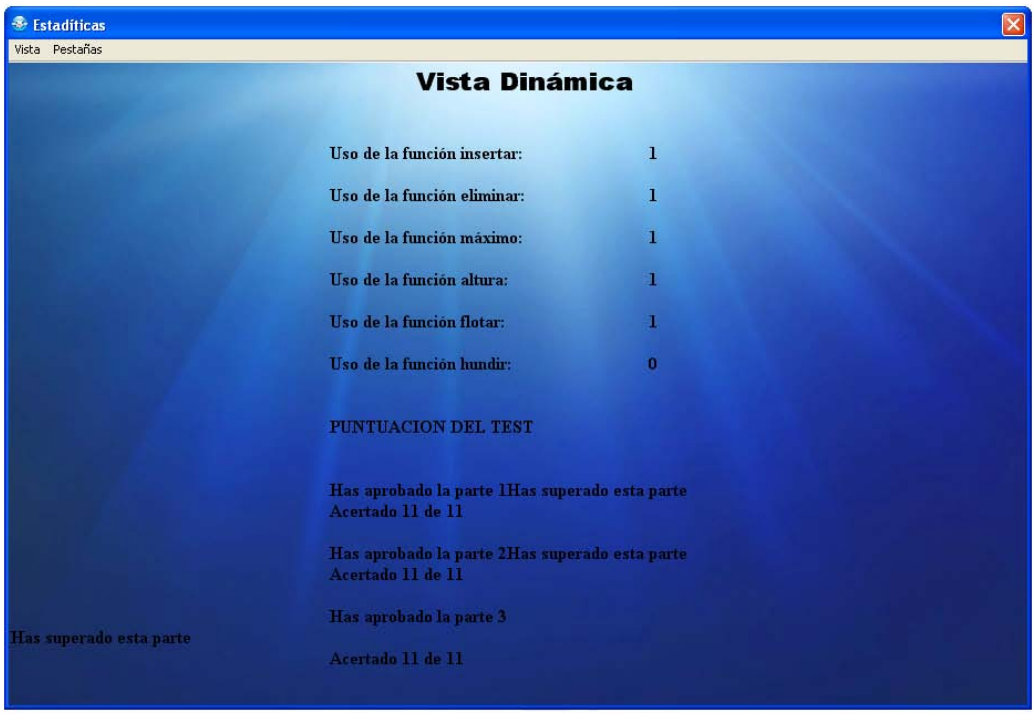


Figura 7.8. Estadísticas de la primera parte de la experiencia.

Una vez superado el test, se activa un botón en la barra de herramientas. Este botón pone "Maude", y al pulsarlo aparece una ventana de selección de archivos.

Figura 7.9. Botón de Maude resaltado.

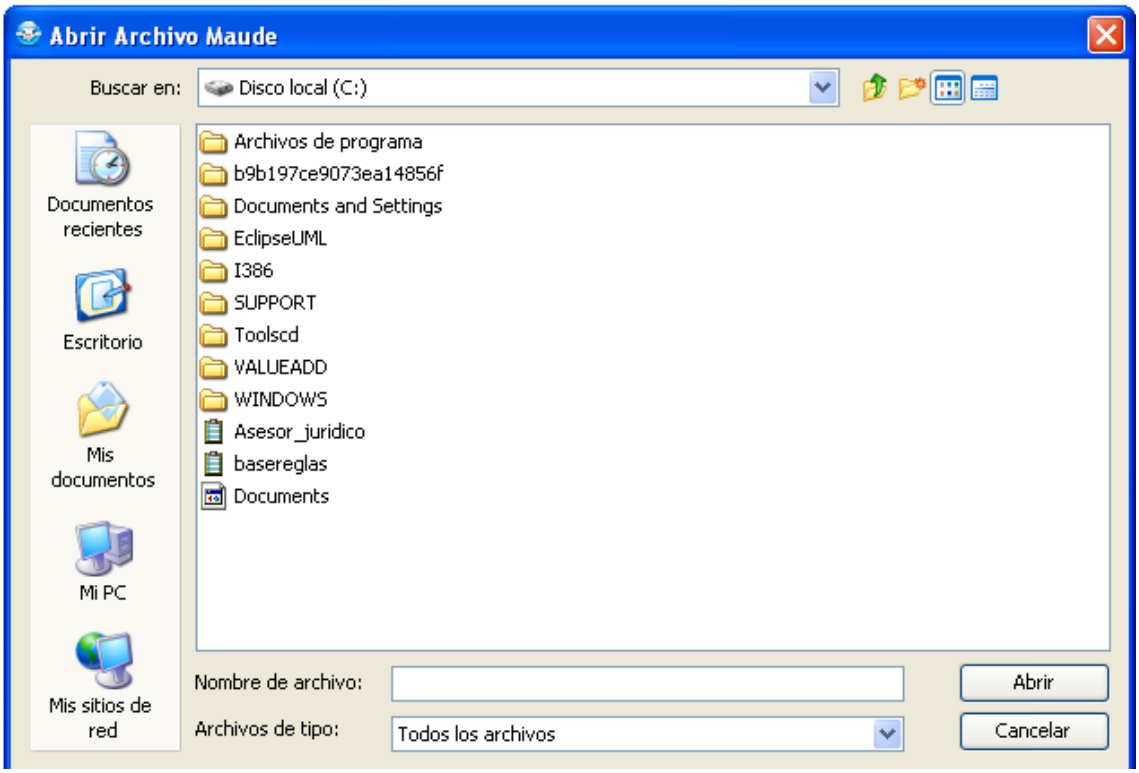


Figura 7.10. Selección del archivo de Maude.

Aquí hay que seleccionar un archivo que contiene una especificación en Maude. Para considerar un archivo como válido, éste debe empezar así (declaración de la estructura):

fmod COLA_PRIORIDAD { X :: TOSET } is
sort ColaP{X} .

Después de esto, en el archivo deben venir recogidas las funciones más importantes que se pueden realizar trabajando con montículos. Estas funciones ya se han definido anteriormente (en el punto 6.2.6). Conviene recordar que los nombres de todas estas funciones son inalterables, puesto que el traductor de Maude busca las compatibilidades en los nombres, y alterar éstos podría llevar a una traducción con fallos.

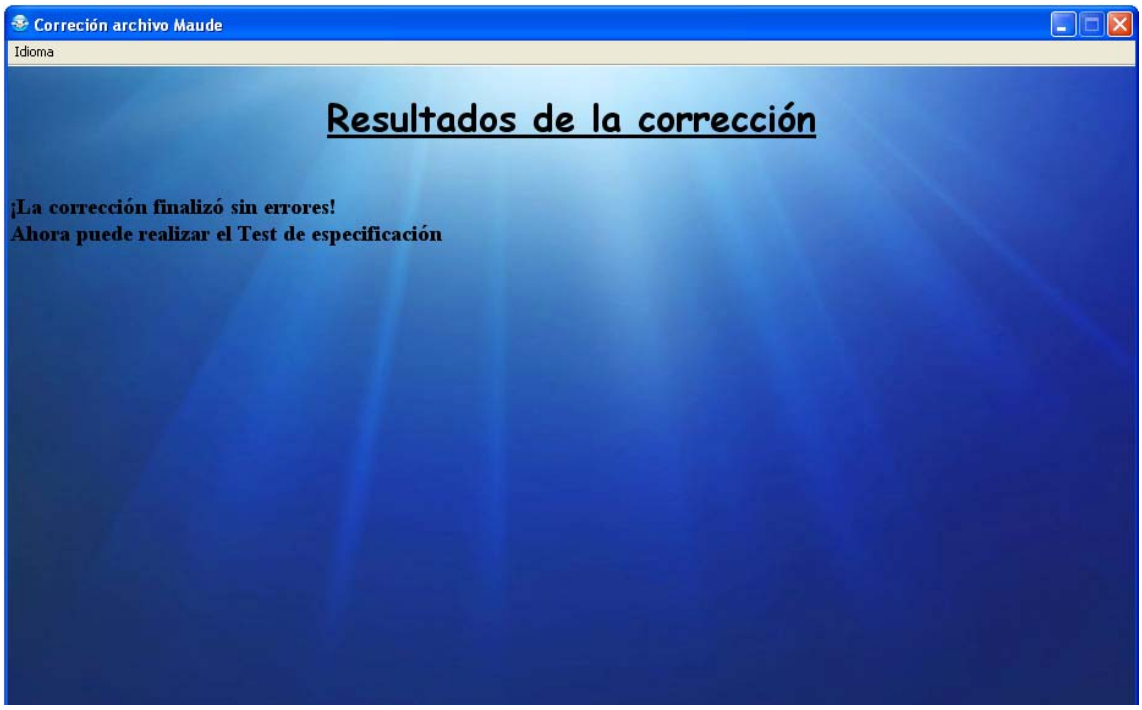


Figura 7.11. Resultados de la corrección del archivo de Maude.

Una vez el sistema nos da el visto bueno al archivo, se desbloquea otro botón en la barra de herramientas, llamado "Test Maude". Este botón nos permite realizar un test al igual que el anterior, solo que las preguntas de éste están centradas en la herramienta Maude. A diferencia del anterior, éste no tiene 3 partes, sino sólo una, la cual debe ser resuelta para que se desactive otro de los botones de la barra de herramientas, llamado "Eclipse".

Figura 7.12. Botón de Test Maude resaltado.

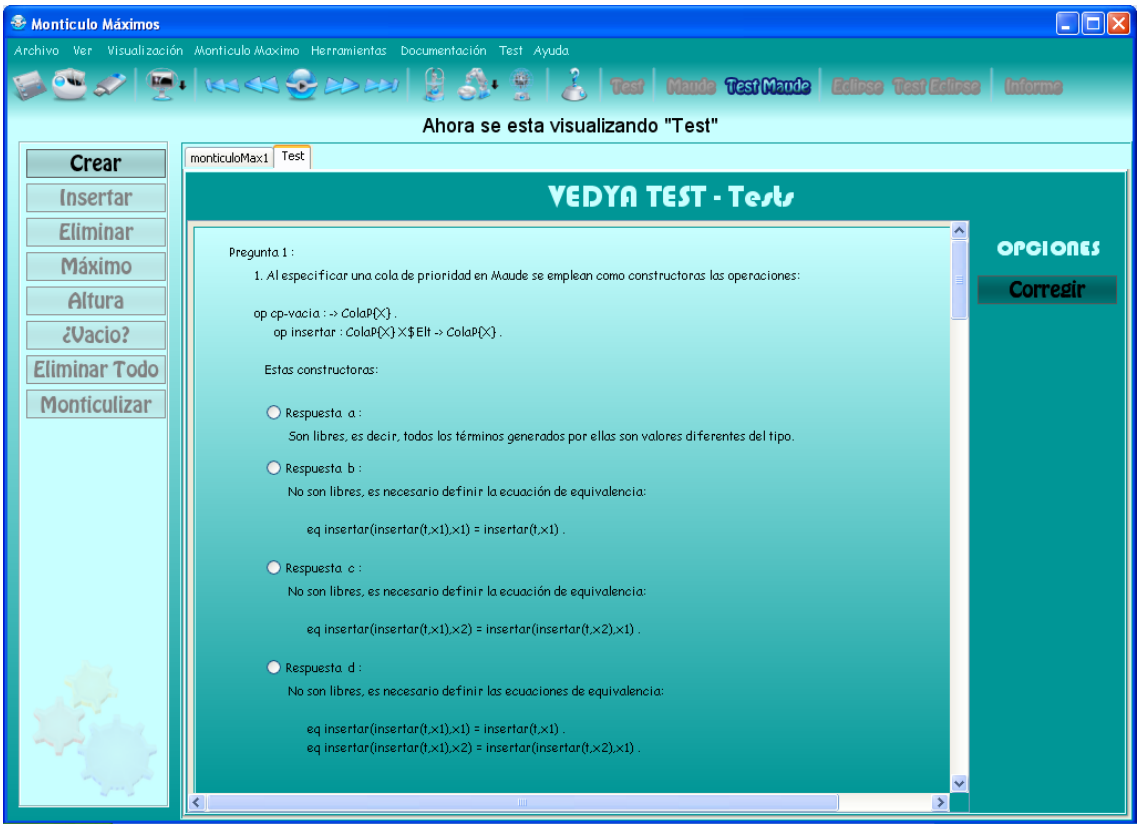


Figura 7.13. Ventana de test correspondiente a la parte de Maude.

Al pulsar sobre el botón Corregir, y al igual que en el caso del test realizado con anterioridad, también aparece una ventana con las estadísticas del último test realizado (en este caso, del test centrado en la especificación de montículos en Maude).

Figura 7.14. Resultados de la corrección del test de Maude.

El botón "Eclipse" funciona exactamente igual que el anterior botón de "Maude", solo que ahora hay que buscar un archivo que contenga una pequeña implementación de los montículos en Java (Eclipse, de ahí el nombre). Éste archivo no debe contener sentencias import ni declaración alguna de clase. En él, sólo se debe encontrar una implementación de las operaciones más importantes de los montículos de mínimos. Los nombres de las funciones (que deben ser escritos así y no pueden ser llamados de otra forma, al igual que en el caso anterior) son los que vienen recogidos en el punto 6.2.7. Es importante resaltar que en este archivo hay que hacer una declaración de errores (como se haría en Java) en aquellas funciones que sean susceptibles de esta declaración (como eliminar-min).

Figura 7.15. Botón Eclipse resaltado.

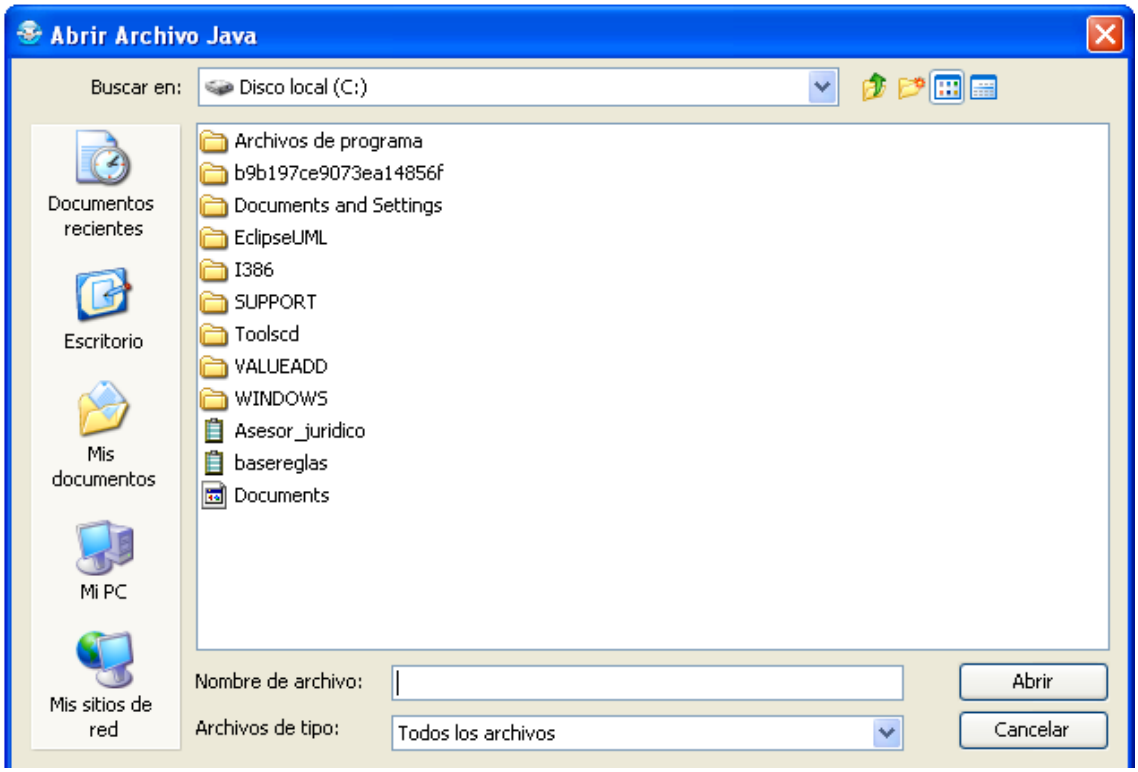


Figura 7.16. Selección del archivo de Eclipse.

Y, después de todo ésto, sólo resta completar un test centrado en preguntas referidas a la implementación de los montículos. Tras completar la anterior implementación de los montículos, y una vez que se ha comprobado que no existían en el archivo errores graves, se activa un botón, "Test Eclipse", que da paso a este test.

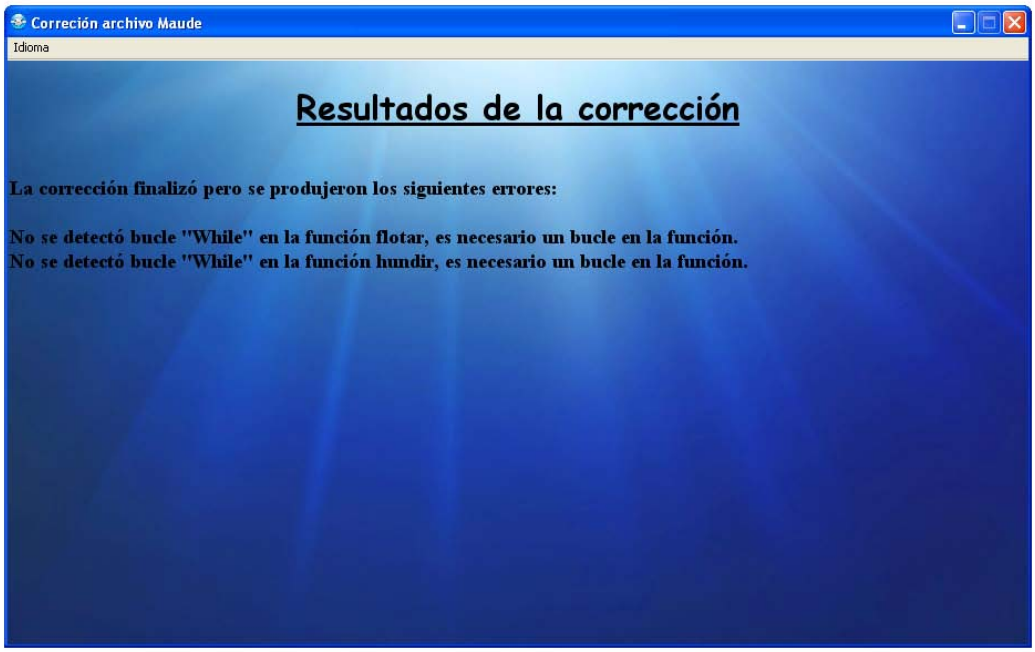


Figura 7.17. Resultados de la corrección del archivo de Eclipse.

Además existe un botón que permite generar un informe con los resultados del alumno, para poder tabular posteriormente los resultados. Al pulsar este botón, aparece una pantalla que tiene los datos de lo que hemos hecho hasta el momento, y abajo una pequeña porción reservada para introducir las observaciones y opiniones sobre el proyecto. Haciendo click en el botón con forma de libros, se acepta y se abre una ventana de selección de archivos, para guardar en una carpeta elegida por el usuario el informe que va a generar el sistema. Una vez seleccionada, aparece en dicha carpeta un informe que contiene los datos mostrados anteriormente, pero codificado.

Figura 7.18. Selección del archivo de Maude.

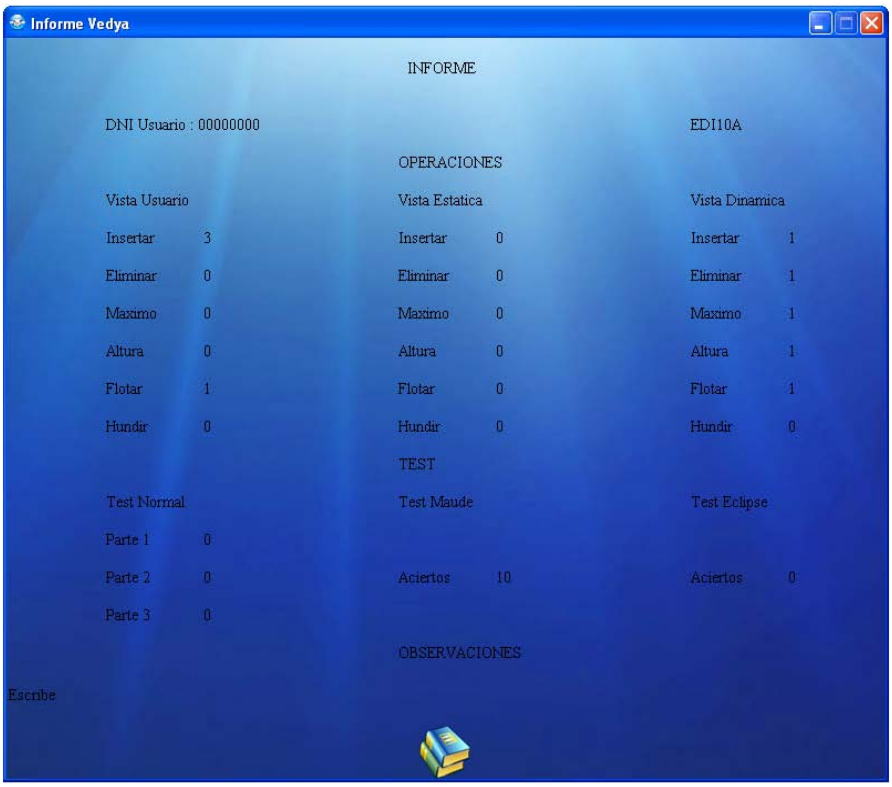


Figura 7.19. Ventana del informe final.

8. Valoración personal

Debemos confesar que, al principio, no pudimos evitar el sentir un cierto recelo en cuanto a este proyecto. Teníamos el código de años anteriores de proyectos similares a éste, pero su enormidad nos llenó al principio de dudas: ¿podríamos ver todas aquellas líneas de código sin perdernos?, ¿podríamos llegar a hacernos una idea general del funcionamiento?

Después, nos surgieron otras: ¿ésto que estamos haciendo es ciertamente lo mejor?, ¿se podría hacer de otra forma?, ¿llegará todo a buen puerto?

Las primeras fueron fáciles de responder, y esta respuesta fue, indudablemente, sí. Para las segundas, sólo una vez terminado hemos comprobado que sí, y si no es lo mejor, sí es una buena solución de todas formas.

Gracias a este proyecto, hemos ampliado las miras que teníamos cuando empezamos esta asignatura: un proyecto, por decirlo así, útil, capaz de llamar la atención, ha sido hecho por nosotros. No podemos evitar sentirnos orgullosos y satisfechos del trabajo realizado.

9. Bibliografía

- *Sistemas interactivos de enseñanza/aprendizaje, editorial Sanz y Torres S.L. , Jesús González Boticario Elena Gaudioso Vázquez.*

- *Las nuevas tecnologías en la educación, Rocío Martín-Laborda, Fundación AUNA.*

- *Paradigmas de aprendizaje, Gonzalo Maldonado Osorio.*

- *Conejo, R., Guzmán, E., Millán, E., Trella, M., Perez-de-la-Cruz, J. L. y Ríos, A. (2004). SIETTE: A web-based tool for adaptive testing. International Journal of Artificial Intelligence in Education, 14, 1-33.*

- *Conejo, R., Millán, E., Perez-de-la-Cruz, J. L. y Trella, M. (2001). Modelado del alumno: un enfoque bayesiano. Revista Iberoamericana de Inteligencia Artificial, 12, 50-58.*

- *Olea, J., y Ponsoda, V. (1996) Tests adaptativos informatizados. En J. Muñiz (Ed.), Psicometría (pp. 729-783). Madrid: Universitas.*

- *Urretavizcaya, M. (2001). Monografía: sistemas inteligentes en el ámbito de la educación. Revista Iberoamericana de Inteligencia Artificial, 12, 2-4.*

- *Wu, A. K. (1993). Paradigms for ITS (intelligent tutoring system). Proceedings of the Conference on Computer, Communication, Control and Power Engineering, TENCON '93. Beijing, China.*

- *Xiang, Y. (2002). Probabilistic reasoning in multiagent systems. A graphical models approach. Cambridge: Cambridge University Press.*

- *A. Aamodt and E. Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Communications. IOS Press, Vol. 7: 1,, pages 39_59, 1994.*

- *Wikipedia. Artículos sobre los lenguajes Maude y Java:*
  Maude: http://es.wikipedia.org/wiki/Maude.
  Java:
  http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java.

Índice de imágenes

ANEXO

From the Algebraic Specification to the Real Implementation: An Educational Environment for the Interactive Learning of Data Structures and Algorithmic Schemes

Rafael del Vado Vírseda
Luis Jiménez Paniagua, José Marcos Barrio, and Rubén Fuentes Iglesias

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid
Facultad de Informática, Madrid, Spain
rdelvado@sip.ucm.es

Abstract— *The high level of abstraction necessary to teach data structures and algorithmic schemes in Computer Science has been more than a hindrance to the students. In order to make a proper approach to this issue, we have developed and implemented an innovative educational environment for the interactive learning of data structures and algorithmic schemes according to the new guidelines of the* European Higher Education Area*. In this paper, we present the new main contributions to this educational environment on Computer Science Education, that is ready for a wider use. First, we describe the tool called* Vedya *for the visualization of data structures and algorithmic schemes. Second, the* Maude *system to execute the algebraic specifications of abstract data types using* Eclipse*, by which it is possible to go from the more abstract level of the algebraic specification to its specific implementation in* Java*.*

Keywords: Computer science education, data structures, algorithmic schemes, algebraic specifications

1. Motivation

The study of "data structures" and "algorithmic schemes" constitutes one of the essential aspects of the academic formation of every engineer in Computer Science. Nevertheless, the high level of abstraction necessary to teach these topics occasionally hinders its understanding to students. In order to make a proper approach to this issue, we have developed and implemented, during the last years, at the Computer Science Department of the Complutense University of Madrid, an innovative interactive learning environment according to the new guidelines of the *European Higher Education Area* and the teaching model focused on the student.

In this paper, we present the two main contributions to this educational environment. On the one hand, the *Vedya* tool, a visualization tool by means of which it is possible to provide the student with a complete learning system of both the main data structures and the more relevant algorithmic schemes. On the other hand, the *Maude* system [3] for the execution of "algebraic specifications" of abstract data types using the language of formal specification provided by this system.

Thanks to the development environment *Eclipse* (`http://www.eclipse.org/`), we have obtained a fully complete system that is useful for the students as well as the professors, that allows to go from the most abstract level of data structures, provided by its algebraic specification in *Maude*, until its specific implementation in a modern programming language as happens with *Java*. All this learning process can be guided and overseen in a completely autonomous way by using the *Vedya* tool, through which it is possible to make enquiries about the documentation related to each of the algebraic specifications, to distinguish between the behavior of the structure and its different implementations through the use of different views or to browse information regarding the cost of the different implementations that have been proposed.

2. The Vedya Tool

Vedya[1] is an integrated interactive environment for learning data structures and algorithmic schemes. It covers the most common data structures: Stacks, queues, binary search trees, AVL trees, priority queues, and sorted and hash tables. Moreover, it also provides other different types of abstract data types, like one for an implementation of a "doctor's office". Concerning the algorithmic schemes, it covers the most common resolution methods [1], [4], [7]: divide and conquer, dynamic programming, backtracking, and branch and bound. All data structures and algorithmic schemes taught in the related study courses are thereby integrated in the same environment: *Vedya* allows the execution of different data structures and several sequences of operations on the same structures at the same time making use of a multi-windows and multi-frame system.

[1] `http://www.fdi.ucm.es/profesor/rdelvado/FECS2010/`

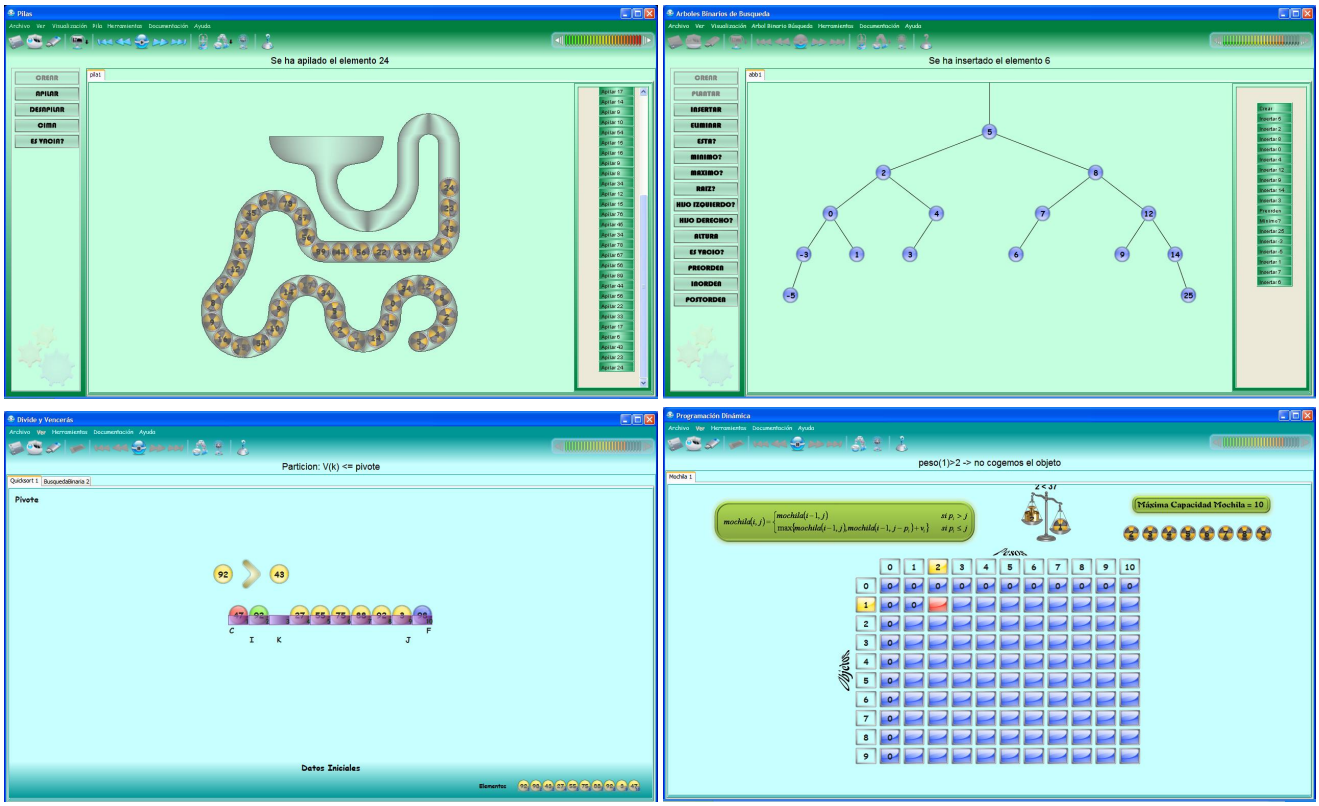


Fig. 1: Data structures and algorithmic schemes in the *Vedya* tool.

Currently, there are two versions of the *Vedya* tool. The first version contains all the data structures and algorithmic schemes mentioned above while the new one offers a subset of them in a more attractive visual environment.

There are several options to use this tool. The main one is the interactive execution, but it is also possible to create simulations that are automatically executed, to visualize tutorials, and to solve tests within the same environment. It also integrates a set of animations that show how data structures are used to solve certain problems. Fig. 6 shows an example of the main windows for data structures (stacks and binary trees). The central panel is used to represent the structure. In the case of linear structures and binary trees, drawing facilities are offered to allow the expansion or contraction of the data structure or to move it over the screen to see the hidden parts. On the left, there is a list of the actions that can be executed. Partial non-allowed actions are disabled. The right panel shows the visualization of the actions that have been already executed. Next, the user may continue executing actions, go up on the sequence of actions to see previous states or she/he may use the stimulation facilities (standard buttons to execute, stop, move forward and move backwards at the top of the screen) to restart the sequence from the beginning. Notice that just above the central panel the result of the last action is shown.

There are two types of views: The one of data structure behavior to intuitively comprehend its operation, and one or several implementation views, either static or dynamic. On Fig. 6 we show the specific behavior view of a stack. Representations of the static implementation based on an array and dynamic implementation based on pointers can be also shown. Furthermore, the environment provides documentation about algebraic specification, the implementation code, and the cost of each implementation. On the top of the screen, there is a menu that facilitates managing the system. We can create a new data structure, open an existing one or save the state of the editing one. We can also execute the operations on the data structure, use the simulation facilities, and change the execution speed of the animations. The main window for the execution of algorithmic schemes looks similar. We have implemented algorithmic schemes based on divide and conquer of binary search and quicksort; algorithmic schemes that solve backtracking problems (in its fractional and non-fractional version) based on dynamic programming; branch and bound, as well as the Dijkstra algorithm to obtain minimum path in a graph.

As has been previously mentioned, *Vedya* is complemented with tutorials on data types (stacks, queues, binary search trees, red-black trees, priority queues, and 2-3-4 trees) and animations of algorithms that show the use of a data

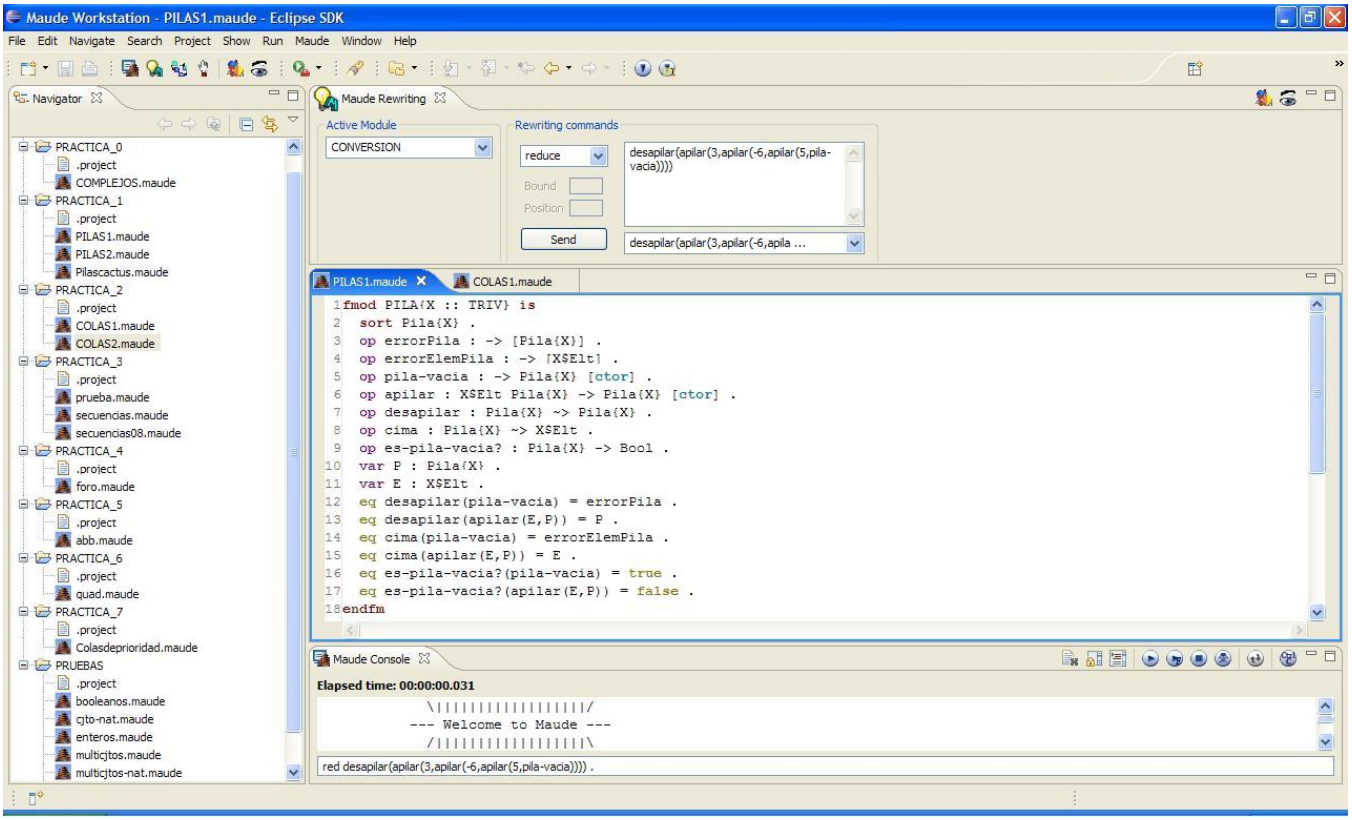


Fig. 2: Integration of *Maude* in *Eclipse* for the execution of algebraic specifications.

structure to solve a problem (evaluation of an expression in postfix form, the transformation of an infix expression to a postfix one, breath-first tree transversal, checking of palindromes). Moreover, there are animations on graphs: To obtain the minimum spanning tree using the Prim and Kruskal algorithms and to compute minimum paths using the Dijsktra algorithm.

Finally, *Vedya* offers the *Vedya-Test* tool to solve tests. This tool can be independently executed and allows teachers to create, modify or delete questions in a database, and to create tests from the database of questions. The student visualizes the tests, solves them and obtains the correct solutions. Questions are grouped by subject-matter on the database, but it is possible to mix questions about different data structures in the same test.

3. Execution of algebraic specifications in Maude

For the execution of algebraic specifications, the language *Maude* [3] based on *rewriting logic* has been used. *Maude* is a high-level language and high-performance system supporting both equational and rewriting computation for a wide range of applications. *Maude* and its formal tool environment can be used in three mutually reinforcing ways: as a declarative programming language, as an executable formal specification language, and as a formal verification system. Moreover, [3] describes the equational specification of the data structures included in the *Vedya* tool now in *Maude* syntax (stacks, queues, lists, binary and search trees, AVL and 2-3-4 trees). The language is available for *Linux* and *MacOS* at `http://maude.cs.uiuc.edu`, and there are also extensions for its execution in *Windows* at `http://moment.dsic.upv.es`.

The specifications can be executed in *Eclipse* (`http://www.eclipse.org/`) by means of special "plugins" developed in the Department of Information Systems and Computation of the Technical University of Valencia and in the Computational Languages and Sciences Department of the University of Málaga. This environment facilitates the student its usage by integrating the text editor with the execution commands of the system (see Fig. 2). On the left, there appear the developed projects; the central part shows the editor and the execution panel of the system is on it; on the inferior part, the control panel that shows the result of the action. On the right part, the user can open other windows that allow the definition of different system options and depuration.

The basic element of a specification in *Maude* is a "module". The language allows defining the functional modules used to define the data types; system modules used to define rewriting systems and modules focused on objects that allow

```
fmod STACK{X :: TRIV} is
  sort Stack{X} .
  op error    : -> Stack{X} .
  op error    : -> X$Elt .
  op empty    : -> Stack{X} .
  op push     : X$Elt Stack{X} -> Stack{X} .
  op pop      : Stack{X} -> Stack{X} .
  op top      : Stack{X} -> X$Elt .
  op isEmpty? : Stack{X} -> Bool .
  var P : Stack{X} .
  var E : X$Elt .
  eq pop(empty) = error .
  eq pop(push(E,P)) = P .
  eq top(empty) = error .
  eq top(push(E,P)) = E .
  eq isEmpty?(empty) = true .
  eq isEmpty?(push(E,P)) = false .
endfm
```

```
fmod QUEUE{X :: TRIV} is
  sort Queue{X} .
  op error    : -> Queue{X} .
  op error    : -> X$Elt .
  op empty    : -> Queue{X} .
  op enqueue  : Queue{X} X$Elt -> Queue{X} .
  op dequeue  : Queue{X} -> Queue{X} .
  op first    : Queue{X} -> X$Elt .
  op isEmpty? : Queue{X} -> Bool .
  var C : Queue{X} .
  var E : X$Elt .
  eq  dequeue(empty) = error .
  ceq dequeue(enqueue(C,E)) = empty
                          if isEmpty?(C) .
  ceq dequeue(enqueue(C,E)) = enqueue(dequeue(C),E))
                          if not isEmpty?(C) .
  eq  first(empty) = error .
  ceq first(enqueue(C,E)) = E if isEmpty?(C) .
  ceq first(enqueue(C,E)) = first(C)
                          if not isEmpty?(C) .
  eq  isEmpty?(empty) = true .
  eq  isEmpty?(enqueue(C,E)) = false
endfm
```

Fig. 3: Algebraic specifications of stacks and queues in *Maude*.

the usage of syntax of classes, objects and messages. The functional modules for stacks and queues are showed with more detail in Fig. 3.

The language allows importing other modules, defining several data types, defining operations on the types and equations that define the behavior of those operations. The modules can be customized, using "theories" to such end in order to define the parameters and "views" to relate the formal parameter to the real parameter. The system has predefined the abstract data types most commonly used, as well as the most common theories and views:

```
view Int from TRIV to INT is
  sort Elt to Int .
endv

fmod STACK-INTEGERS is
  including STACK{vInt} .
endfm

fmod QUEUE-INTEGERS is
  protecting QUEUE{Int} .
endfm
```

As can be observed, the syntax is similar to the one used in several texts of algebraic specifications of data types [8].

In order to execute the specification, the student enters the text in the editor; then, she/he executes the *Maude* system using the existing buttons in the console and enters the module. The system detects existing syntax errors and shows them on the console. Once the module shows no more errors, the student may reduce terms by using the equations of the module. To such end, the student may use the commands chart placed at the top of the screen or she/he may directly write the command in the editor and enter it into the system. For example, in order to obtain the first of a queue, we can reduce the term:

```
red first(enqueue(enqueue(empty, 5),4)).
```

This term must be reduced over the module of the queues using the integer number theory `INT`. In our example, this module is named: `QUEUE-INTEGERS`.

The possibility of reducing terms, in an automatic way, allows the students to carry out an initial test of their specifications by detecting many of the errors committed when defining the operations using equations. Another greater advantage of executing the specifications is that the student comprehends the difference between the parameterized module and the instantiated module by being able to reduce terms on different modules. For example, a new module could be named `QUEUE-CHARACTERS` on which terms of type

```
red first(enqueue(enqueue(empty, 'a'),'c')).
```

can be easily reduced.

Other examples of data types, such as "medieval queues" or a "doctor's office" were also proposed [8]. In all of them, the aim was to define parameterized or instantiated data type with different theories. The practical classes are complemented with different terms that the student must reduce over some type of instantiated modules to prove the specification, as well as proposals to make little changes in some actions or erroneous definitions to detect them.

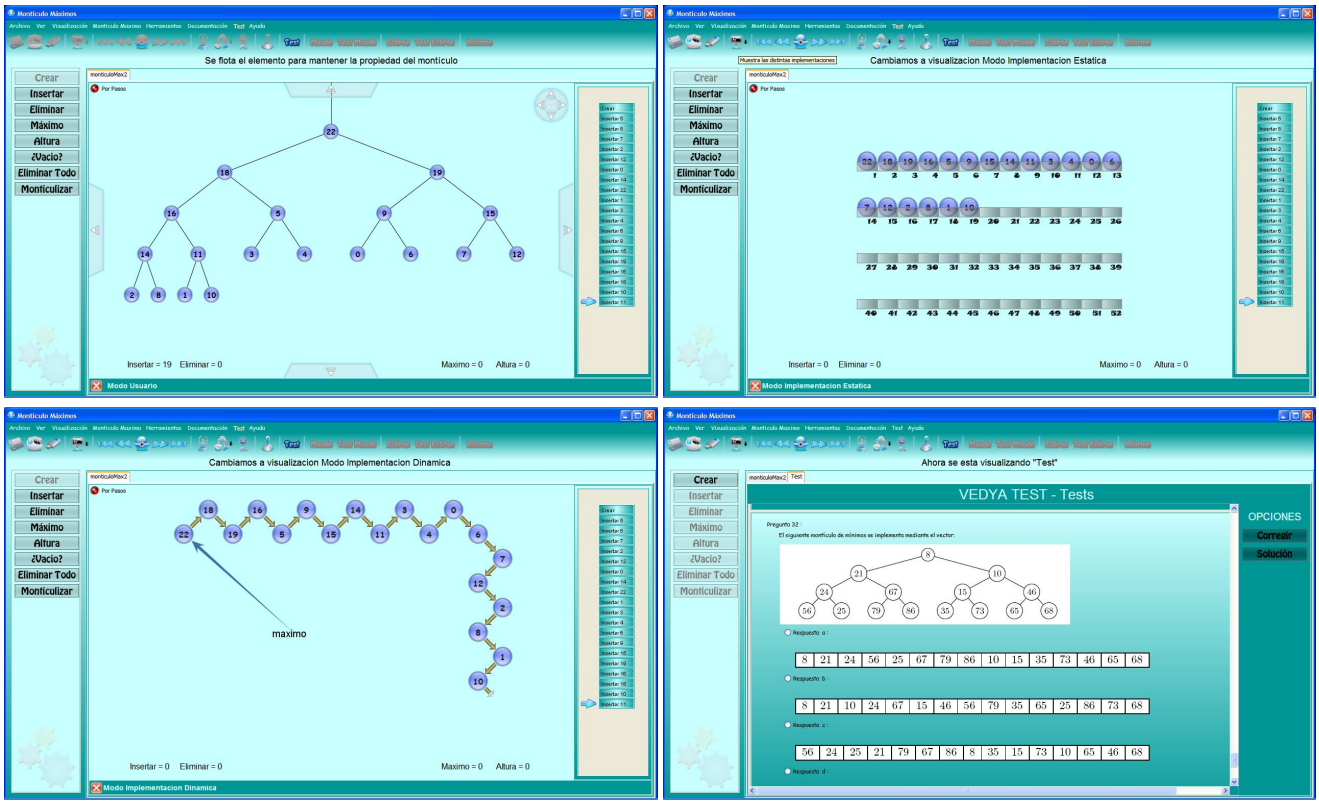


Fig. 4: From the algebraic specification to the real implementation in the *Vedya* tool.

Taking into consideration that students from the second year were involved, just a few of the language facilities have been used. In superior courses where students have more knowledge on the subject, a richer language can be used [3] (e.g., many-sorted equational specifications, order-sorted equational specifications, equational attributes, and membership equational logic specifications).

4. From specification to implementation

The *Vedya* tool turns into a pedagogical instrument of high practical interest since it attempts to address the whole self-learning process of the main data structures, from the algebraic specification in *Maude* until the possible implementations in *Java*, within such a powerful and integrated environment as the one that has been described in the previous section by means of the *Eclipse* system.

The students have their first contact with the data structures that they are going to study by means of the usage of *Vedya*. For example, if their learning of data structures is focused on binary search trees or linear data structures, they will start learning the corresponding section of the tool, where they will be able to experiment, freely and on their own, each one of the actions offered by these structures (see Fig. 6). In order to strengthen and evaluate this intuitive knowledge, the student has, in addition, the possibility of using the *Vedya-Test* tool.

Once the student has a clear idea of the informal behavior of the data structure, she/he may start working on the *Eclipse* system. The first step would be: to formally capture that intuitive knowledge she/he has obtained through the usage of *Vedya* in a specific algebraic specification written in *Maude* syntax. In order to facilitate this difficult step in the student's self-learning, she/he may use, interactively, the documentation that is included in the manual of the *Vedya* tool.

Once the specification (see Fig. 3) is entered into the *Eclipse* system, the student can now go on executing little tests using *Maude*, in order to check whether it coincides with the intuitive and informal notion of data structure from which he initially departed in *Vedya*. Such experience would allow the student to reach the high level of abstraction that is necessary in computer supported education for each formal specification of a software component, always based on the intuitive and experimental knowledge.

Once the algebraic specification of the data structure is obtained, the next step would be to develop an implementation in an object-oriented programming language such as *Java*, by means of the facilities provided by the programming environment in *Eclipse*. This time, the student may use the

	Stacks 1	Stacks 2	Queues	Sequences	BST	AVL	RB	Heaps
Group A (130)	76.4%	82.5%	77.8%	65.6%	82.2%	84.9%	–	86.3%
Group B (59)	78.9%	83.6%	85.0%	63.6%	86.2%	87.7%	90.9%	90.2%
Group C (131)	76.2%	79.8%	73.5%	69.0%	83.5%	–	68.9%	86.8%

	2003/04	2004/05	2005/06	2006/07	2007/08	2008/09
Not attended	57.6%	45.3%	42.3%	64.7%	50.8%	40.2%
Passed	15.3%	22.2%	20.2%	18.2%	30.1%	42.6%
Failed	27.1%	32.5%	37.5%	17.1%	18.9%	17.2%

Fig. 5: Students answering the tests and percentage of correct answers.

algebraic specification that she/he has built, as if dealing with an authentic "instructions manual". The main advantage of our methodology is that the specification behaves now as a prototype of the data structures to be implemented, in a way that the student is able to find out the exact behavior for all those moments of doubt that may appear during the design process, even before she/he is able to compile the program. In order to be able to guide, in a more specific way, the step of specification to implementation, the student may make use again of the *Vedya* tool. This time, the student may access to the part that would correspond with the implementation of data structure that she/he is studying from the options menu (see Fig. 6). From there, she/he may try different implementation possibilities based on arrays or pointers.

Once the student is familiar with the different implementations of the structure, she/he is finally ready to properly decide on a suitable representation in the *Java* language. The possibility of having understood and previously evaluated the different implementations by means of *Vedya* allows the student the possibility to acquire a clear knowledge of the *algorithmic cost* of the chosen implementation in *Java* for each specific operation of the data structure, so that this would also be a decisive criterion at the moment of designing its own implementations. In this part, the "*algorithmic schemes*" part of the *Vedya* tool plays an important role, since it allows the student to acquire a good programming methodology.

5. Evaluation

In order to obtain a detailed evaluation of the usage of *Vedya* and *Maude* in our integrated system, we have proposed several tests related to the behavior, specification, implementation and application of the main data structures offered by the tool. We also collect students' opinion using *Vedya* in the "Data Structures" academic subject at the second year, and in the "Programming Methodology and Technology" subject at the third year, respectively.

The vast majority of our engineering and computer science students have taken an introductory programming course in the first academic year, typically in *Pascal*. Although the learning of the main algorithmic schemes and programming techniques is not a prerequisite to the subject of "Data Structures", many students choose to take it either prior to, or concurrent with, "Programming Methodology and Technology". As a result, although a pseudocode programming language is the assumed language for "Data Structures", many students have enough knowledge about *C++* or *Java* programming languages through the integrated programming laboratories of parallel academic courses and subjects.

Taking into account this profile, skills and background of our engineering and Computer Science students, we have proposed 8 tests in the *Virtual Campus* of the Complutense University of Madrid (`http://www.ucm.es/campusvirtual/CVUCM/`). The number of engineering students registered in the *Virtual Campus* was just over 320 distributed in three groups (130 in group A, 59 in group B, and 131 in group C). Fig. 5 shows the number of the students who answered each of the tests in the corresponding group. We observe that, from the second test on, the number of students becomes stable in a number lightly low to the number of students who access regularly to the *Virtual Campus*. These numbers, though seemingly high, are only between 23 % (75 students of 320) and 37 % (118 of 320) of registered students, which shows the high rate of students giving up in this topic from the beginning.

Fig. 5 also shows the percentage of correct answers in the three groups: In general, it is high, which demonstrates the interest of the students who have taken part. In group B, the percentage is slightly higher than groups A and C; since 85 % of the students who have decided to complete the tests across the *Virtual Campus* of group B are not "new" students of this academic subject. Fig. 5 shows the percentage of students that did not attend the final exam, those who passed, and those who failed during the last six years. We observe that in the last academic courses, in which we have applied the *Vedya* tool, we have reduced by 14% the percentage of students giving up the course with respect to the previous course, and at the same time, we have increased by 12% the percentage of students that passed the exam. The percentage of students that failed the exam

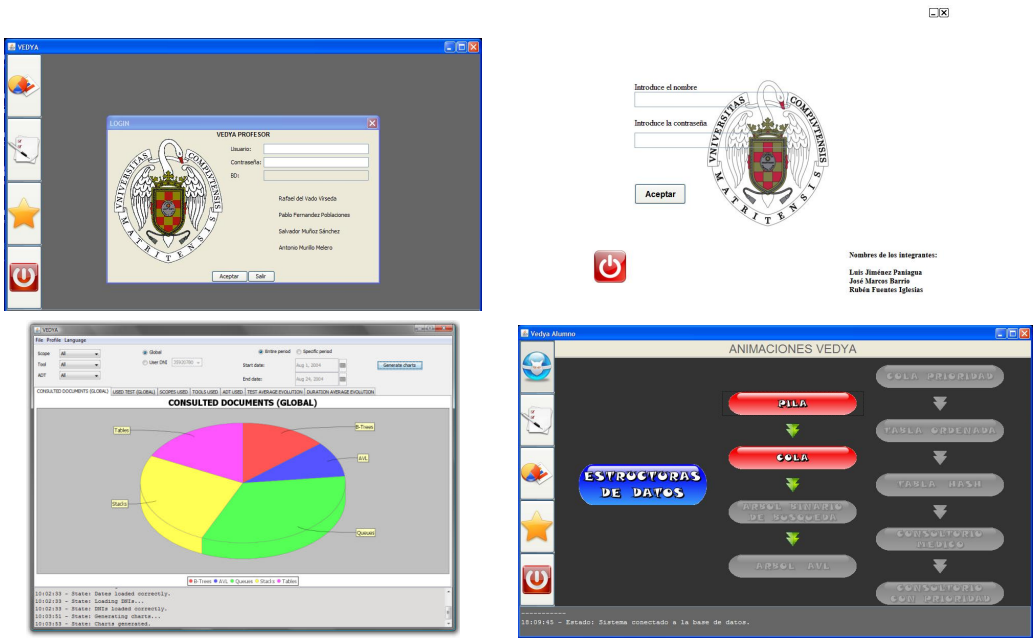


Fig. 6: An Intelligent Tutoring System for the *Vedya* tool.

increased by 2% due to the rise of students attending the exam. Comparing with previous courses (2004 to 2005) the percentage of students that passed has increased between 8% (with respect to the course 2004/05) and 15% (with respect to the course 2003/04).

6. Conclusions

In this paper, we have described the usage of an innovative educational environment for the interactive learning of data structure and algorithmic schemes by means of the visualization tool called *Vedya* and the specification language *Maude* with its programming environment in the *Eclipse* system.

In the last years, many papers on visualization of data structures and algorithms have been written. For example, a tool with a similar style is presented in [2]. Nevertheless, there is a lack in many of them of a graphic user interface of data structures and algorithms or they can only be executed in a few operative systems. In this sense, *Vedya* is something more than a simple tool for the execution of data structure as has been shown in Section 2.

The application of *Vedya* and *Maude* in a complete system as *Eclipse* allows the students the possibility of acquiring the capacity of implementing, correctly and properly, a data structure according to its formal algebraic specification, using in their design, the proper algorithmic schemes. As a consequence, it is possible to provide the students with a complete and professional methodology of software development that is very useful in the current teaching of Computer Science.

During the academic courses 2007/08 and 2008/09, we have carried out a detailed study in classroom on the application of this innovative educational environment by the *Virtual Campus* of the Complutense University of Madrid, in the "Data Structure" and "Programming Methodology and Technology" courses corresponding to the second and third academic courses of Computer Science.

As future work, we plan to design an *Intelligent Tutoring System* in *Eclipse* in order to guide the interactive *self-learning* process of data structures from the algebraic specification to the real implementation.

References

[1] G. Brassard and P. Bratley. *Fundamentals of algorithms*. Prentice Hall, 1996.

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

[3] M. Clavel and et al. *All about maude. A high performance logical framework*. In *How to Specify, Program and Verify Systems in Rewriting Logic*, Springer LNCS, 2006.

[4] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[5] R. del Vado Vírseda. *A visualization tool for tutoring the interactive learning of data structures and algorithmic schemes*. Procs. of the 41st ACM technical symposium on computer science education (SIGCSE 2010), pages 187–191, 2010.

[6] R. del Vado Vírseda. *An interactive tool for data structure visualization and algorithm animation - Experiences and Results*. Procs. of the Second International Conference on Computer Supported Education (CSEDU 2010), 2010.

[7] R. Neapolitan and K. Naimpour. *Foundations of algorithms using C++ pseudocode*. Jones and Bartlett, 2003.

[8] M. Weiss. *Data Structures and Problem Solving Using Java*. Addison-Wesley, 1998.