Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial
ISSN: 1137-3601
revista@aepia.org
Asociación Española para la Inteligencia Artificial
España

Gómez Martín, Marco Antonio; Gómez Martín, Pedro Pablo; González Calero, Pedro A.
Aprendizaje Activo en Simulaciones Interactivas
Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, vol. 11, núm. 33, 2007, pp. 25-36
Asociación Española para la Inteligencia Artificial
Valencia, España

Aprendizaje Activo en Simulaciones Interactivas

**Marco Antonio Gómez Martín, Pedro Pablo Gómez Martín,
Pedro A. González Calero**

Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid
C/ Prof. José García Santesmases, s/n
28040 Madrid (España)
{marcoa, pedrop}@fdi.ucm.es, pedro@sip.ucm.es

Resumen

Las simulaciones por ordenador en entornos inmersivos se han utilizado tradicionalmente como herramientas de enseñanza sobre todo en dominios donde el entrenamiento en el mundo real resulta demasiado costoso o peligroso. El trabajo que aquí presentamos se enmarca dentro de una línea emergente en la aplicación de los entornos inmersivos de enseñanza que no busca replicar en el entorno virtual las sensaciones del mundo real sino favorecer la motivación del estudiante y el aprendizaje en el contexto de una actividad de resolución de problemas.
En este artículo presentamos el sistema JV^{2}M, un entorno de aprendizaje para enseñar la máquina virtual de Java y la compilación de lenguajes orientados a objetos a través de una metáfora que utiliza el lenguaje de los videojuegos. En JV^{2}M la actividad del estudiante se centra en la resolución de ejercicios con la colaboración de un agente virtual pedagógico, cuyo comportamiento viene dado por una representación explícita de conocimiento del dominio. Además del funcionamiento y arquitectura del sistema, se describen aquí las herramientas de autoría que soportan su uso.

Palabras clave: Educación, enseñar haciendo, agente pedagógico, compilación, Java.

1. Introducción

La aproximación de enseñanza conocida como *aprender haciendo* (*learning-by-doing*) promueve un aprendizaje basado en la experiencia, herencia de los días de los aprendices de trabajos artesanales como el de zapatero o carpintero. Aquellos días, los expertos observaban como los no versados en la materia iban resolviendo las tareas cada vez más complicadas que les asignaban. La misma técnica se utiliza en otras áreas como al enseñar a pilotar un avión o a controlar una central nuclear. Sin embargo, en esos casos los sistemas que se manejan son mucho más caros y críticos, por lo que es arriesgado dejar el control en manos de alguien inexperto. Los entornos virtuales generados por ordenador solucionan el problema. Basta con construir *simuladores* de las máquinas y recrear situaciones donde el estudiante actúa como si lo hiciera en el mundo real, desarrollando así las habilidades necesarias en un entorno seguro y controlado. Por ejemplo, [10] muestra un escenario simulado para entrenar a los oficiales de acciones tácticas, y [9] embarca al estudiante en un navío virtual para enseñarle a manejar un compresor de alta presión. También es posible crear entornos que no tienen su contrapartida en el mundo real. Por ejemplo, [3] crea un ordenador virtual donde el usuario tiene que ejecutar un programa, recogiendo las instrucciones de la memoria principal, cargándolas en la CPU y ejecutándolas.

Los simuladores más simples únicamente se encargan de recrear el entorno real en un entorno virtual donde el que el usuario puede interactuar. Estos

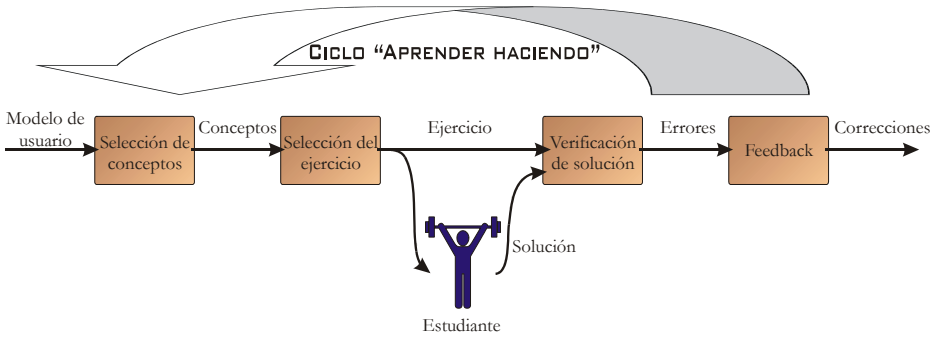

Figura 1: Ciclo aprender haciendo

sistemas siguen requiriendo un tutor humano que observa las acciones que el estudiante realiza en el simulador para corregirle. Existen sistemas más avanzados, conocidos como ITSs (*Intelligent Tutoring Systems*), que son simuladores a los que se le añade un módulo inteligente que hace las veces de "profesor", registrando todos los movimientos que hace el estudiante y corrigiéndole [10].

Nosotros proponemos el uso de esta aproximación de "aprender haciendo" para enseñar compilación de lenguajes orientados a objetos, en particular compilación de Java. El sistema resultante proporciona una simulación metafórica de la máquina virtual de Java (JVM, [7]), donde los estudiantes se enfrentan a ejercicios de compilación. Utilizando las ideas de los agentes pedagógicos (como [3], [6] y [9]) nuestro entorno de aprendizaje se completa con JAVY, un personaje que ayuda al usuario en su proceso de aprendizaje. La interacción con el entorno es similar al de las aventuras gráficas en tres dimensiones. Como este método se suele asociar al ámbito de los juegos de ordenador, se espera que tenga el efecto positivo de maximizar la motivación mientras el estudiante está resolviendo los ejercicios.

El sistema resultante se llama JV^{2}M, dando la idea de que es *dos veces virtual*, ya que es una representación *virtual* de la máquina *virtual*.

La sección siguiente describe nuestra visión del ciclo de aprender haciendo. La sección 3 introduce las principales características de JV^{2}M, para pasar en la sección 4 a describir su arquitectura. La sección 5 muestra el conocimiento del sistema y cómo el agente pedagógico lo utiliza durante la resolución de un ejercicio. Tras la descripción de las herramientas para la construcción de ese conocimiento en la sección 6, pasamos a explicar un ejemplo de ejecución del sistema en la sección 7. El artículo termina con unas conclusiones y trabajo futuro.

2. Ciclo de aprender haciendo

La técnica de aprender haciendo ataca el problema de la enseñanza de habilidades enfrentando a los estudiantes con ejercicios valiosos pedagógicamente. Nuestro modelo de aprender haciendo se puede ver como un ciclo (Figura 1). Consta de cinco pasos:

1. *Selección de los conceptos a practicar*: se eligen los conceptos del dominio. Si queremos que el aprendizaje tenga éxito, la selección no debe hacerse de forma arbitraria, sino dependiendo del modelo del usuario. Esta selección abre las puertas al uso de distintas estrategias pedagógicas.
2. *Selección del ejercicio*: utilizando los conceptos seleccionados, se escoge de la base de ejercicios (o se crea automáticamente) uno que los ponga en práctica.
3. *Resolución del ejercicio*: el alumno resuelve (o, al menos, intenta resolver) el problema al que se enfrenta.
4. *Verificación de la solución*: las acciones del usuario son comparadas con la solución válida, para evaluar su corrección.
5. *Feedback*: utilizando las conclusiones alcanzadas en el paso anterior, se proporcionan al alumno consejos o explicaciones. Por ejemplo, el sistema puede explicar cuestiones relativas a los errores cometidos, o incluso aclarar algún conocimiento teórico.

En el caso de los ITS, normalmente las tres últimas etapas se realizan *a la vez*. El sistema a menudo proporciona al alumno ayuda contextualizada durante la resolución de un ejercicio (etapa 3 del ciclo), por lo que necesita estar comprobando continuamente la solución parcial (etapa 4) y proporcionar las explicaciones *antes* de que de su respuesta sea completa (etapa 5).

3. Descripción general de JV^{2}M

Con las ideas anteriores de simuladores e ITSs, hemos desarrollado el sistema JV^{2}M. JV^{2}M es un sistema para enseñar el proceso de compilación de lenguajes de alto nivel, es decir, *cómo el código fuente se traduce a código objeto*. En realidad no nos centramos en el trabajo del analizador léxico y sintáctico, sino en cómo se realiza (de manera intuitiva) la generación del código de, por ejemplo, la evaluación de las expresiones o de las estructuras de control. Para conseguir aprender a realizar la traducción, el estudiante necesita saber tanto el lenguaje fuente como el objeto. Hemos elegido Java como lenguaje fuente y el lenguaje interpretado por la máquina virtual de Java (JVM) como lenguaje objeto. Éste es conocido como los *bytecodes* de Java, y debe entenderse como el *lenguaje ensamblador* de la JVM. Debemos asumir que el alumno conoce Java, pero no obligamos a que conozca el código objeto de la JVM. Por eso, nuestro sistema *también lo enseña*, lo que lo convierte en un sistema educativo centrado en dos aspectos: *conceptos de alto nivel* relacionados con los procesos de compilación, y *conceptos de bajo nivel* relacionados con las instrucciones y estructuras de la JVM.

Para eso, cada ejercicio está compuesto del código Java y el código objeto al que el estudiante tendrá que llegar traduciendo (compilando) el primero. El sistema contiene una colección de estos ejercicios. En cada ejecución del ciclo de la Figura 1 se selecciona automáticamente, en la segunda fase, uno de ellos dependiendo de los objetivos pedagógicos que se deseen cubrir.

Una vez decidido el siguiente ejercicio, cualquier aplicación educativa basada en la aproximación de aprender haciendo, lo presenta al estudiante para que lo resuelva. Por ejemplo, un sistema basado en Web pediría al usuario que introdujera el código objeto generado, posiblemente tecleando una a una cada instrucción máquina. Nuestra aproximación es distinta: el estudiante *ejecuta* el código, en vez de escribirlo. Más concretamente, el código que debería ser el resultado de la compilación es ejecutado en un entorno virtual en tres dimensiones que recrea una metáfora de la JVM. Como modelo subyacente de esa metáfora, el sistema implementa una máquina virtual, que permite al entorno presentar el estado de sus estructuras en cada momento, y que debe verse como el *simulador* del sistema, igual que en una aplicación para enseñar a pilotar un avión se debe simular su física. Las estructuras de la máquina virtual son representadas mediante diferentes objetos y personajes. El usuario también aparece en el mundo 3D, representado mediante un avatar que habita en ese mundo virtual (Figura 9).

Para la interacción entre el usuario y el entorno virtual, hemos utilizado las ideas seguidas por muchos juegos de entretenimiento, en particular, nos hemos basado en el sistema utilizado en las *aventuras gráficas* como las desarrolladas por LucasArts Entertainment Company LLC. En éstas, el usuario puede pasear por el entorno y cuando se acerca a algún objeto con el que puede interactuar, aparecen sobreimpresionadas las acciones que puede realizar con ellos. Existen cuatro acciones básicas: *mirar*, *coger*, *usar* y *usar con*. Aunque el significado concreto de cada una de ellas varía dependiendo del objeto con el que se esté tratando, las ideas son las siguientes:

- *Mirar*: sirve como primer nivel de ayuda, especialmente para aquellos objetos que no puedan ser representados con suficiente detalle. Cuando se utiliza la acción *mirar* sobre uno de ellos, el avatar del usuario proporciona su descripción. Por ejemplo, al realizar la acción sobre la representación 3D metafórica de la pila de operandos (un componente básico de la JVM), el usuario podrá ver al propio avatar decir "es la pila de operandos del *frame*[1] actual".
- *Coger*: el avatar posee un *inventario*, donde guarda los objetos que va recogiendo. Esta acción sirve precisamente para eso. Sin embargo, la *semántica real* de la acción puede variar sensiblemente de unos casos a otros. Por ejemplo, si se ejecuta la acción sobre la pila de operandos, mencionada previamente, en vez de "coger" la pila entera, se "desapilará" el último valor.
- *Usar*: el objeto seleccionado es utilizado de un modo particular. Por ejemplo, "usar" un avatar conlleva el comienzo de una conversación, mientras que "usar" una palanca significa accionarla.
- *Usar con*: es la única acción compuesta disponible. El avatar puede tener en la mano uno de los objetos del inventario, y usarlo con otro objeto del entorno. Como antes, la semántica concreta dependerá de la combinación de objetos que se haga. Por ejemplo, utilizar un operando con la pila de operandos significará *apilar* ese valor.

[1] *Frame es el registro de activación en la terminología de la JVM*

El inventario es, como hemos dicho, donde se almacenan los objetos recogidos del entorno. Éstos objetos simbolizan operandos, constantes, referencias a clases y otros elementos intermedios necesarios para ejecutar las instrucciones sobre la metáfora.

Además, el entorno está también habitado por un agente pedagógico llamado JAVY que es el responsable de la fase de *feedback*. Al igual que ocurre en muchos ITS, en vez de esperar al final de la resolución del ejercicio, JAVY proporciona ayuda contextualizada acerca de la situación actual. Cuando el usuario pide ayuda, el agente utiliza su conocimiento para darle distinta información dependiendo del ejercicio y su estado de ejecución. El estudiante puede incluso pedir a JAVY que termine el ejercicio mientras él le observa.

El alumno puede realizar preguntas utilizando también el tipo de interacción que aparece en muchas de las aventuras gráficas, como *Grim Fandango* de LucasArts. En particular, el usuario se acerca a JAVY y es él, *el agente*, quién decide qué preguntas son válidas en ese momento en función del contexto, y las presenta al estudiante. Éste elige una de ellas, y obtiene la respuesta. A continuación se comienza un nuevo ciclo en la conversación, volviéndose a presentar nuevas preguntas que encajen con la explicación que acaba de proporcionarse. Más adelante se muestra el conocimiento del sistema para llevar a cabo este proceso.

4. Arquitectura

El sistema, a grandes rasgos, tiene cuatro elementos: el motor gráfico, el simulador del mundo, el entorno virtual y el agente pedagógico.

En los siguientes apartados se hace una descripción de cada uno de ellos.

4.1 Motor gráfico

Dentro de este módulo, colocamos las rutinas encargadas de presentar la imagen tridimensional en pantalla y la gestión de la entrada del usuario. Constituye la parte más concreta de la aplicación, que lidia con el *hardware* subyacente utilizando librerías gráficas y llamadas al sistema operativo.

El sistema hace uso de Nebula [8], un motor gráfico de libre distribución, para el dibujado del entorno virtual que, además, se encarga de gestionar la entrada del usuario, abstrayendo los distintos dispositivos y plataformas, generando eventos unificados que distribuye al resto del sistema.

4.2 Simulador

El simulador es el encargado de guardar el estado actual del mundo en el que el estudiante está inmerso, de manera independiente de la metáfora elegida. Todas las acciones que tanto el usuario como el agente realizan sobre la metáfora visual son ejecutadas en realidad *sobre el simulador*. Esa ejecución provocará los cambios de estados que posteriormente serán propagados al entorno virtual.

Cualquier cambio en el simulador conlleva la modificación de su apariencia en entorno virtual. Para facilitarlo, sus elementos están programados para ser capaces de avisar ante cambios de estado, algo que resulta muy útil para el módulo siguiente.

Es aquí donde se almacena el estado de ejecución del ejercicio. Para ello, implementa una máquina virtual de Java reducida, que permite cargar clases, crear objetos, y realizar todas aquellas operaciones que el sistema puede enseñar.

También se considera parte del estado del mundo el contenido del inventario. Recordemos que el inventario son todos los objetos que el estudiante o el agente han ido recogiendo por el mundo y que deben utilizar para resolver el ejercicio.

4.3 Entorno virtual. *Object Interface Manager*

Este módulo es el que gestiona la representación visual de cada uno de los objetos del simulador que aparecen en la metáfora. Cuando los objetos del simulador cambian, avisan a sus representaciones visuales, para que alteren su apariencia.

El módulo recibe el nombre de *gestor de objetos de interfaz* (OIM, *Object Interface Manager*), y los objetos que controla son conocidos como *objetos OIM*. Éstos pueden agruparse en dos categorías:

- *Objetos representando una estructura de la JVM*: por ejemplo, el usuario puede ver un edificio por cada una de las clases Java cargadas en la JVM, o una pila de cajas en el *frame* actual que representa la pila de operandos.
- *Objetos con los que se puede interactuar*: por ejemplo, los personajes que habitan el entorno. JAVY es uno de estos "objetos", ya

que el estudiante puede acercarse y hablar con él (bajo la primitiva "*usar*"). Hay otros objetos interactivos que no representan una estructura determinada en el simulador, sino que aparecen como consecuencia de la metáfora elegida. Por ejemplo, la especificación de la JVM no habla explícitamente de la existencia de una unidad aritmético-lógica (ALU). Sin embargo, su existencia se da por supuesta para la ejecución de muchas de las instrucciones. Debido a ello, hemos decidido incluir en la metáfora un avatar que hace las veces de una ALU, constituyendo un objeto interactivo que no tiene su equivalente en una JVM real.

Aunque la primera categoría de objetos no es modificada directamente por los avatares (estudiante o JAVY), son dinámicos, debido a que pueden cambiar cuando el personaje utiliza los objetos interactivos de la segunda categoría. Por ejemplo, la metáfora elegida no permite realizar ninguna acción directa sobre la pila de frames. Sin embargo, cuando se crea un nuevo frame, el objeto que representa esa estructura de la JVM cambia su apariencia en el entorno, por lo que debe considerarse un objeto dinámico.

Por otro lado, existen entidades que pertenecen a ambas categorías. Por ejemplo, la ya nombrada pila de operandos, puede ser manipulada a través de la metáfora, además de representar una estructura de la JVM.

Existen dos entidades del entorno virtual *especiales*. Son los dos avatares que representan al estudiante y a JAVY. Su singularidad se da porque son las únicas capaces de *interactuar* con el resto. El esquema interno recuerda mucho al de los Sims [4]. Cada objeto dinámico es responsable de conocer cómo actúan los avatares sobre él y no al contrario. Cuando un avatar quiere realizar una operación sobre uno de ellos, utiliza un método de ese objeto indicando que es él quien quiere ejecutarlo.

Cuando tiene lugar una de estas ejecuciones, comienza un proceso que terminará con el cambio en el estado del mundo y en la metáfora presentada por pantalla. En particular, un avatar *ejecuta* una acción sobre un objeto OIM interactivo. Éste *no* cambia su estado directamente, sino que llama a la función apropiada en el simulador. Por ejemplo, si el objeto interactivo es el que gestiona la creación de nuevos frames, llama a la implementación de la JVM albergada en el simulador, para que cree un nuevo frame. El simulador, al cambiar su estado, *avisa* al objeto encargado de la representación del objeto cambiante (pila de frames), para que actualice su representación en el entorno virtual.

4.4 Agente pedagógico

JAVY es el agente pedagógico que ayuda a resolver los ejercicios. En el apartado anterior hemos hablado someramente del avatar que lo representa en el entorno virtual. Existe un componente aparte del sistema en el que se implementa su inteligencia o, dicho de otro modo, la gestión del agente. Desde él salen las órdenes hacia el avatar, para que se mueva e interactúe en el mundo virtual. La implementación está dividida en tres módulos: el módulo de percepción, el de movimiento y el módulo cognitivo.

La base de las acciones que este componente trata son lo que conocemos como *microinstrucciones*, En el apartado 5.1 daremos más detalles sobre ellas. Por ahora basta entender que una microinstrucción es la unidad mínima de acción que puede realizarse en el entorno con repercusión sobre la JVM subyacente.

Módulo de percepción

Es el responsable de detectar las acciones que se realizan en el entorno virtual. Para eso, recibe mensajes cada vez que algún atributo del mundo cambia.

El responsable del envío de esos mensajes es el módulo OIM, explicado anteriormente. OIM contiene un generador de mensajes que es utilizado por cualquier objeto OIM que quiere informar a las capas de percepción de los posibles agentes.

El cometido principal es la *traducción* entre las acciones que han sido ejecutadas sobre los objetos de la metáfora y las *microinstrucciones* que el módulo cognitivo entiende. Por ejemplo, se traduce la acción de *usar* un objeto del inventario con la pila de operandos, por la microinstrucción *push*.
La capa de percepción también es avisada cuando el estudiante quiere iniciar una conversación con JAVY. Cuando la conversación empieza, es también esta capa la responsable de *recoger* las frases dichas por el estudiante y enviárselas al módulo cognitivo.

Módulo de movimiento

Cuando el agente quiere ejecutar alguna acción en el mundo, utiliza este módulo para mover al actor y que la operación se realice.

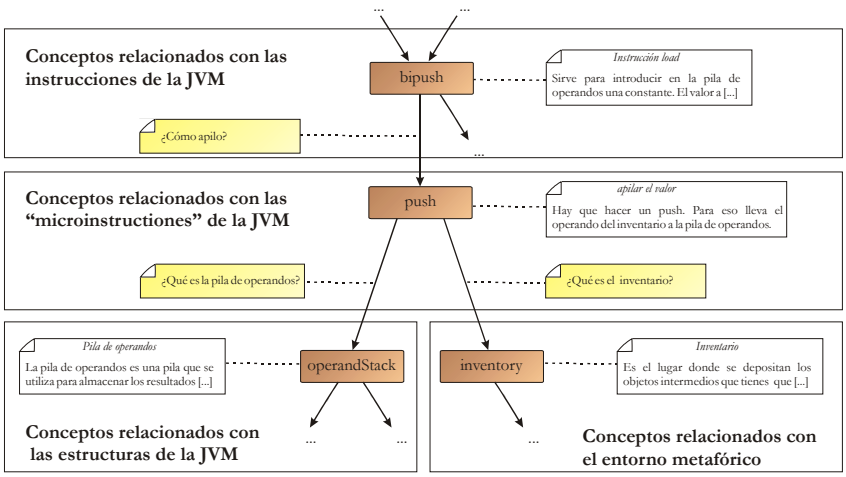

Figura 2: Fragmento detallado de la jerarquía conceptual

El cometido es el inverso al de la capa de percepción. Debe traducir las órdenes provenientes del módulo cognitivo en acciones sobre el entorno virtual. Para eso, debe ser capaz de encontrar el objeto OIM que ejecuta esa operación e invocarlo.

Módulo cognitivo

Es el que toma todas las decisiones de JAVY, según la información que le llega desde la capa de percepción. Cuando decide entrar en acción, utiliza la capa de movimiento para que el avatar que lo representa se anime e interactúe con el entorno virtual.

El módulo cognitivo está implementado en una hebra aparte. Esta decisión se ha tomado en base a la necesidad de tiempo real en la parte de actualización del entorno virtual. Utilizando una hebra auxiliar no imponemos restricciones de tiempo de respuesta en el razonamiento del agente.

Las decisiones que el agente toma son muy dependientes del conocimiento que éste maneja para comprobar la validez de los pasos que está siguiendo el estudiante. En el siguiente apartado detallamos cómo almacenamos el conocimiento del sistema en JV^2M.

5. Conocimiento del sistema

Como hemos dicho, el sistema enseña el proceso de compilación de Java. Su primera fuente de conocimiento es un conjunto de ejercicios Java, que son indexados para su selección en la segunda fase del ciclo esquematizado por la Figura 1. Internamente, cada ejercicio contiene el código Java y el código compilado. Cada ejercicio es diseñado por un tutor humano, que lo anota dando indicaciones de cómo se ha llevado a cabo el proceso de compilación. Los ejercicios hacen posible enseñar los que previamente denominábamos *conceptos de "alto nivel"*, en otras palabras, el proceso de la compilación.

Dado que los estudiantes *ejecutan* las instrucciones del código compilado, el sistema no solo debe averiguar si el alumno está compilando el código Java correctamente, sino también si está ejecutando bien el código objeto resultante. Para ello, el sistema tiene que conocer el modo de ejecución de instrucción de la JVM y, además, ser capaz de dar explicaciones si el alumno comete fallos. Este conocimiento es *compartido por todos los ejercicios*, pues muchas instrucciones de la JVM serán utilizadas en un sinfín de ellos. La información sobre la ejecución es también construida por tutores humanos, y constituye lo que denominábamos domino de *"bajo nivel"* (modelo de ejecución de la JVM).

El punto de unión entre ambos niveles es la tercera fuente de conocimiento, a la que llamamos jerarquía conceptual y que almacena todos los conceptos del dominio que se deben aprender. Contiene explicaciones *generales* de todos los conceptos (sin enmarcarlas en un problema o instrucción de la JVM específica).

Los siguientes subapartados describen cada una de estas tres partes, empezando con la última.

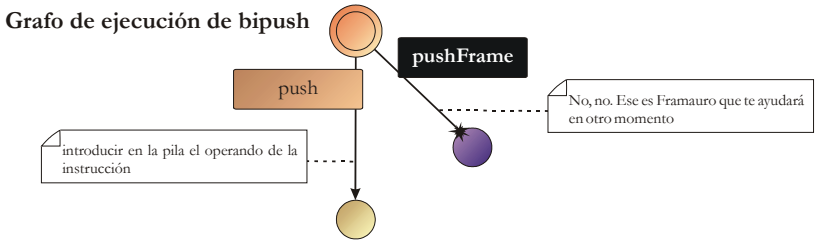

Figura 3: Grafo de ejecución sencillo con un camino erróneo

5.1 Jerarquía conceptual

La jerarquía conceptual es una red de conceptos relacionados. Cada concepto tiene un *nombre*, una *descripción textual* que la detalla, y algunas *conexiones* con otros conceptos relacionados.

JAVY utiliza la descripción textual para explicar cada concepto cuando el estudiante pide ayuda. En caso de preguntar por ellas, JAVY le "dice" ("lee") esas descripciones, y ofrece al usuario la posibilidad de *preguntar* por los conceptos relacionados con el actual a través de las *conexiones* que aparecen en la jerarquía. En lugar de utilizar un generador de lenguaje natural capaz de construir esas preguntas, las aristas están etiquetadas con el propio texto de la pregunta.

La Figura 2 ejemplifica la idea. Por ejemplo si se le pregunta a JAVY sobre el concepto `push` (en el centro de la imagen), da la explicación que aparece en la etiqueta de ese concepto: "Hay que hacer un push. Para eso, lleva el operando del inventario a la pila de operandos". Después de eso, se comprueban los enlaces, y se le permite al usuario preguntar "¿Qué es la pila de operandos?" y "¿Qué es el inventario?". El ciclo de la conversación se repite utilizando otro concepto, dependiendo de la selección del usuario.

Los conceptos se organizan dentro de cinco posibles categorías, dependiendo de a qué hagan referencia. La primera categoría engloba a todos aquellos conceptos que se refieren a la *compilación* de Java, como por ejemplo `arithmeticExpression` o `whileInstruction`. Hacen referencia, por tanto, sobre todo al dominio *de alto nivel* enseñado por el sistema.

La segunda se refiere a instrucciones de la JVM, como `returnInstruction` o `bipush`, la tercera incluye las *microinstrucciones* identificadas en la JVM y que veremos más adelante como `push` o `popFrame` y la cuarta se refiere a las *estructuras* de la JVM como `operandStack`. Todas ellas hablan del dominio *de bajo nivel* enseñado por el sistema.

Finalmente, hay un quinto grupo que contiene conceptos relacionados con la *metáfora* realizada entre la JVM real y el entorno virtual recreado en el sistema. El sistema no tiene como objetivo enseñar esos conceptos, pero se incluyen para que el sistema sea capaz de explicarlos y así *ayudarle* a utilizar la propia aplicación. El ejemplo más claro es `inventory`, que explica lo que es el inventario, algo claramente relativo a la propia dinámica de uso de la aplicación, pero no a la compilación ni a la ejecución de código en la máquina virtual.

La separación de los conceptos en categorías *no* es utilizada por JAVY, sino que sirve para añadir información específica a algunos de ellos que *no* es necesaria en los demás. Es útil, también, como un modo *ad hoc* de detectar errores en las relaciones internas del conocimiento, como veremos después.

5.2 Grafos de ejecución

Obviamente la aplicación debe almacenar en algún sitio de qué forma se ejecutan las instrucciones de la JVM. Hemos analizado todas ellas e identificado los diferentes pasos (*microinstrucciones*) que se necesitan para ejecutarlas. Entendemos por microinstrucción a una *manipulación atómica de una estructura de la JVM*. Ya hemos dicho que la jerarquía conceptual enumera, entre sus conceptos, aquellos que se refieren a instrucciones y a microinstrucciones y cada uno tendrá, como es natural, una descripción general sobre lo que hace.

El sistema almacena además un *grafo de ejecución* para cada instrucción. Cada nodo representa un estado de la ejecución de esa instrucción, mientras que las aristas son los pasos que hacen avanzar el proceso mediante una microinstrucción. Las aristas también guardan una explicación que indica por qué esa microinstrucción *es válida en ese momento*. Con esto tenemos que la jerarquía mantiene una descrip-

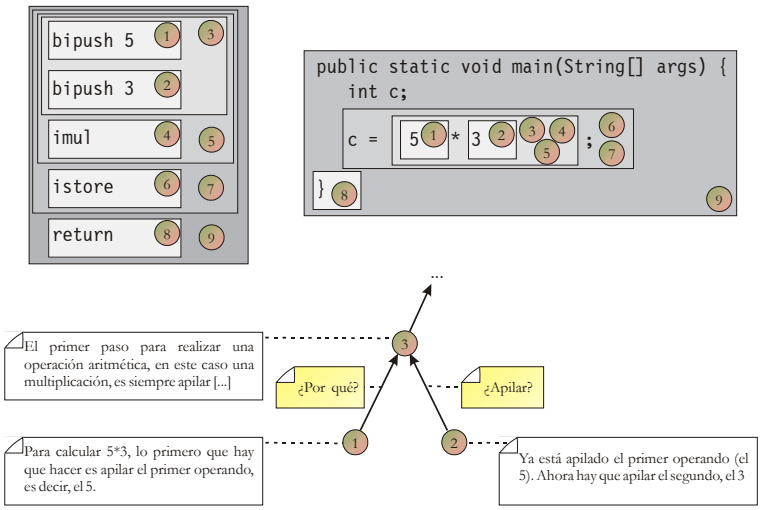

Figura 4: Fragmento detallado de un ejercicio

ción general de cada microinstrucción, y el grafo de ejecución almacena una explicación contextualizada a la instrucción donde se utiliza. El control (somero) de problemas en el conocimiento comentado previamente encaja en este punto, pues se obliga que sólo existan grafos de ejecución para aquellos conceptos que pertenezcan a la categoría de *instrucciones*, y que sólo puedan colocarse en las aristas conceptos marcados como *microinstrucciones*. De esa forma se evita que se deslicen errores que coloquen, por ejemplo los conceptos `whileInstruction` o `inventory` como microinstrucción.

En la Figura 3 aparece el grafo de ejecución de una instrucción sencilla (`bipush`), en la que únicamente se requiere la ejecución de una microinstrucción (`push`). En la arista en la que aparece, se aprecia que está etiquetada con la explicación: "introducir en la pila el operando de la instrucción". Si el usuario intenta realizar otro paso, JAVY utiliza ese texto y lo mezcla con una plantilla para construir su consejo: "Si yo fuera tú intentaría *introducir en la pila el operando de la instrucción*".

En caso de que el estudiante siga confundiéndose, JAVY busca el concepto de la microinstrucción válida en la jerarquía conceptual, recupera su campo *nombre* (en este ejemplo "apilar el valor", como aparece en la etiqueta asociada a él en la Figura 2), y lo aplica a otra plantilla, para decir "Para *introducir en la pila el operando de la instrucción*, hay que *apilar el valor*".

Los grafos de ejecución, además, pueden guardar explícitamente caminos erróneos con explicaciones especializadas sobre los errores más comunes, de forma que JAVY pueda proporcionar consejos. En la Figura 3 se aprecia que `pushFrame` no es una microinstrucción válida, y, si se ejecuta, el agente diría "No, no. Ese es Framauro que te ayudará en otro momento". En ese caso, si el alumno se equivoca ejecutando esa microinstrucción concreta, en lugar de dar el consejo genérico construido con la plantilla se proporciona esta explicación más específica.

Aunque en la figura no se muestra, los grafos de ejecución contienen información adicional sobre aquellos objetos que deben *manipular*. Por ejemplo, la instrucción `bipush` del ejemplo tiene *un operando* con el valor a apilar. Es necesario que ese requerimiento quede marcado de forma explícita en algún sitio; por otro lado, se necesitará una *coherencia* de forma que el sistema obtenga automáticamente ese operando y se lo haga llegar al avatar del jugador colocándolo automáticamente en su inventario.

Por su parte, la microinstrucción `push` (consistente en, efectivamente, apilar el valor en la pila de operandos) debe hacer referencia al objeto que hay que apilar. Esto repercute en el último aspecto que dejábamos abierto al hablar de la jerarquía conceptual. Aquellos conceptos que se encuentran dentro de la categoría de *instrucción* o *microinstrucción* poseen información adicional sobre su "prototipo", consistente en el número de objetos que reciben "de entrada" (manipulan) o generan "de salida" (obtienen, como por ejemplo al extraer un dato de la pila de operandos).

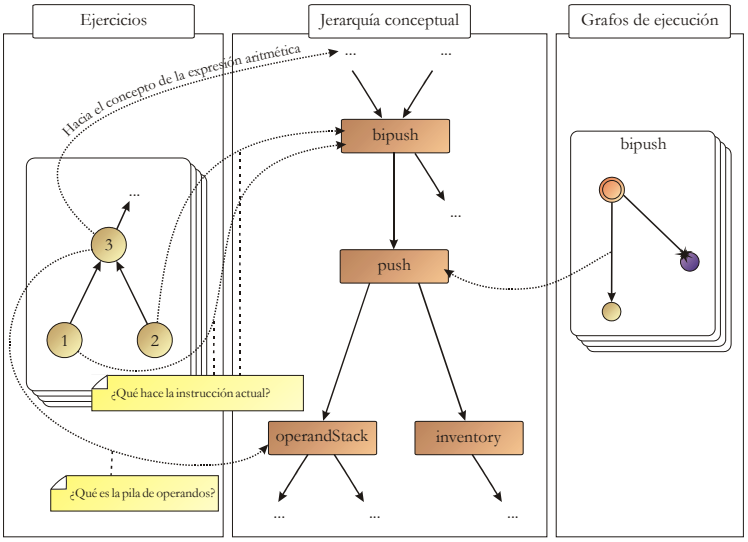

Figura 5: Visión general del conocimiento de JAVY

La información sobre el prototipo se extrae de la jerarquía conceptual; los grafos de ejecución relacionan los objetos obtenidos en la ejecución de unas microinstrucciones con los utilizados en las posteriores, de modo que pueda asegurarse que el usuario está obteniendo y utilizando correctamente los diferentes valores intermedios.

5.3 Ejercicios

Cada ejercicio contiene el código fuente de un programa Java y su versión compilada. Los ejercicios también son creados utilizando una herramienta de autoría que ayuda a organizar en bloques jerárquicos tanto el código fuente como el código objeto. Esta jerarquía genera una estructura en forma de árbol que normalmente es bastante similar al árbol sintáctico creado por el compilador. El tutor que diseña un determinado ejercicio añade una explicación a cada bloque (nodo del árbol), que aclara por qué esa sección se ha compilado de esa forma.

Las aristas que unen los nodos del árbol también son enriquecidas con frases (más bien preguntas) que son utilizadas por JAVY para guiar la conversación. La Figura 4 muestra un fragmento de un ejercicio. Por claridad, hemos etiquetado cada bloque del árbol con un número. Cuando se está ejecutando la primera instrucción de la JVM (`bipush 5`, estaríamos en el nodo 1), el usuario puede preguntar al agente por qué se ha traducido el primer operando de la expresión `5*3` con esa instrucción. En este caso, JAVY contestará con "Para calcular 5*3, lo primero que hay que hacer es apilar el primer operando, es decir, el 5" (como aparece en la etiqueta de dicho nodo). Utilizando la arista que une los bloques 1 y 3 del árbol, el alumno puede preguntar "¿Por qué?", lo que lo permitirá ir ascendiendo en la jerarquía y obteniendo información cada vez más general y referida a mayores porciones de código.

5.4 Juntando todas las piezas

Los tres bloques de conocimiento analizados no están aislados, sino relacionados utilizando enlaces adicionales. Esto los convierten en una buena fuente de información que JAVY usará para "dialogar" con el usuario. Por ejemplo, los bloques de código de los ejercicios suelen estar enlazados con nodos de la jerarquía conceptual, de forma que los bloques pequeños se relacionan con el concepto de la instrucción de la JVM que contienen, y los grandes se relacionan con los conceptos de compilación.

Todos estos enlaces adicionales también están etiquetados con preguntas que el usuario puede hacer cuando el estado de la conversación está localizado en el nodo del que sale la arista. La Figura 5 muestra la estructura general.

6. Herramientas para la construcción del conocimiento

En la sección anterior se ha realizado una descripción sobre la estructura del conocimiento poseído

por el sistema. Todo ese conocimiento está especificado en varios ficheros XML con una DTD específica adaptada a las necesidades de la aplicación. Esos archivos con conocimiento son creados por *tutores* expertos en el dominio, y con ellos se alimenta el sistema para enseñar al alumno.

Aunque tradicionalmente XML suele anunciarse como de fácil comprensión (lectura) para los humanos, en la práctica escribir manualmente un fichero XML de cierta longitud resulta bastante farragoso y propenso a errores. Para facilitar la labor de los tutores se han desarrollado (en Java) varias herramientas de autoría específicas para el conocimiento necesario, que general los XML interpretados posteriormente por la aplicación educativa.

Dado que el punto central de todo conocimiento del sistema está en la jerarquía conceptual, la primera herramienta que se desarrolló, conocida como JaCo, es la encargada de facilitar su especificación.

Los grafos de ejecución se crean con otra herramienta diferente denominada JaMOn. Aunque no se ha entrado en demasiado detalle, existe una fuerte dependencia entre la jerarquía conceptual y los grafos de ejecución que originaron que ambas herramientas terminaran fusionándose en una sola que recibió el nombre mixto de JaCoMOn.

Se espera que el conocimiento creado con ella (la jerarquía conceptual y los grafos de ejecución) sea relativamente *estable* a lo largo del tiempo, pues sirve como *marco* para los ejercicios en sí mismos (código Java y su versión compilada). Éstos son creados por una tercera herramienta conocida como Jade.

6.1 JaCo

JaCo es la herramienta utilizada para construir la jerarquía conceptual. Permite crear los conceptos y establecer la información de cada uno de ellos. Ésta incluye el nombre y la explicación del concepto tal y como se muestra en las figuras previas. También mantiene información adicional como un comentario, cuya única utilidad es ser un lugar de anotaciones *para el autor* del conocimiento (JAVY no lo utiliza nunca), o una *descripción corta* útil para *recordar* el significado de un concepto sin necesidad de explicarlo completamente.

JaCo también permite establecer las relaciones entre los conceptos, creando aristas etiquetadas con la pregunta realizada por JAVY para permitir pasar de un concepto a otro durante la conversación.

6.2 JaMOn

JaMOn se utiliza para escribir los grafos de ejecución utilizando, también, una herramienta visual. Se encarga de mantener la *coherencia* con la jerarquía conceptual asociada, controlando que sólo se utilizan conceptos *instrucción* y *microinstrucción* en los lugares correctos, y que se hace un uso correcto de los *prototipos* de cada una de ellas.

6.3 Jade

Jade se utiliza para escribir los ejercicios. El tutor debe proporcionar el código Java del ejercicio y su versión compilada, así como relaciones entre los bloques de ambos códigos y explicaciones de cada uno de ellos. Como ayuda, la herramienta recibe el código en Java y *compila* automáticamente el código invocando a `javac`. Utilizando la información *de*

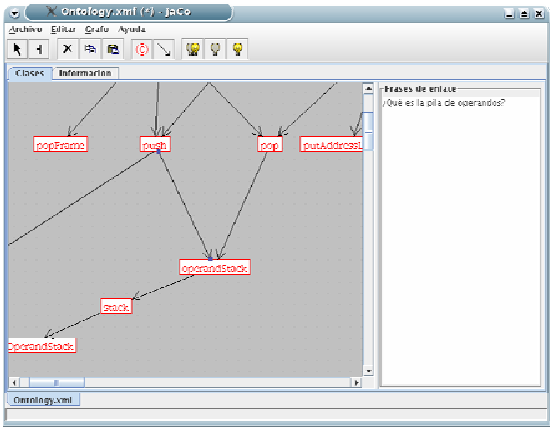

Figura 6: JaCo en ejecución

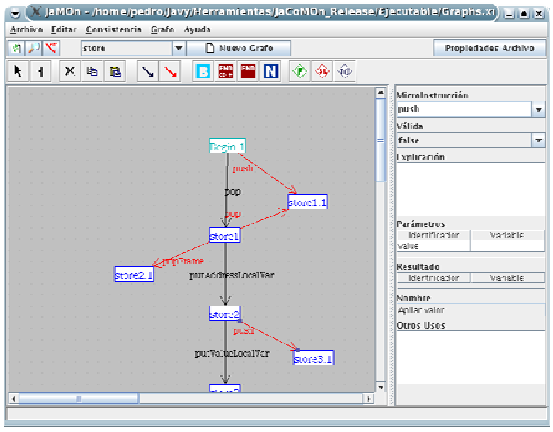

Figura 7: JaMOn en ejecución

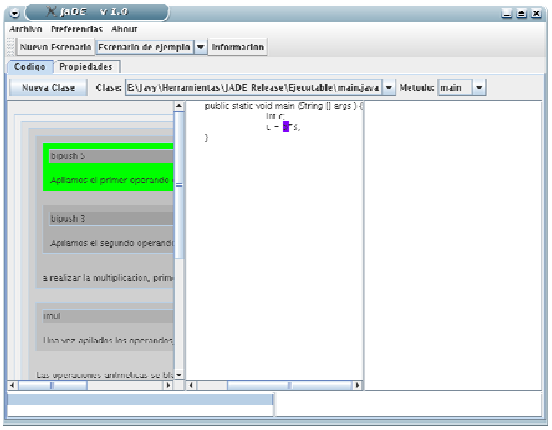

Figura 8: JaCo en ejecución

depuración incluida en los `.class` generados, construye una primera tentativa de relación de bloques que, posteriormente, el tutor podrá mejorar.

7. Ejemplo detallado

En esta sección mostramos un pequeño ejemplo de ejecución del sistema[2]. El estudiante se enfrenta al ejercicio mostrado en la Figura 4. Utilizando el teclado, es posible consultar el código Java que hay que compilar. En este ejemplo, al principio del ejercicio, se debe ejecutar la instrucción `bipush 5`. Dado que ésta está relacionada con el número `5` de la expresión del código Java, éste aparece destacado al principio del ejercicio (ver Figura 9).

La instrucción "`bipush 5`" tiene un parámetro que se añade automáticamente al inventario del usuario como un objeto virtual. Imaginemos que el usuario lo selecciona e intenta dárselo a Framauro, un personaje auxiliar utilizado para manipular la pila de frames. Esa acción se corresponde con la microinstrucción `pushFrame`. Como mostrábamos en el grafo de ejecución en la Figura 3, JAVY interpreta este paso como inválido y da una explicación adaptada a este error, obteniéndola del grafo: "No, no. Ese es Framauro que te ayudará en otro momento". Ante este error, supongamos que el usuario se rinde, y pregunta a JAVY. En el primer paso o ciclo de la conversación, el agente permite al usuario hacer siempre las mismas preguntas generales:

- ¿Qué se supone que tengo que hacer ahora?

[2] *Video disponible en http://gaia.fdi.ucm.es/grupo/projects/javy/*

Figura 9: El sistema en funcionamiento

- ¿Por qué tengo que hacer esa instrucción?
- Déjalo. Ya me las apaño.

La última frase siempre estará disponible, para permitir al usuario terminar el diálogo. Las otras dos aparecen siempre que se inicia una conversación. Sin embargo, cada vez enlazarán a diferentes partes del conocimiento, dependiendo del contexto.

En concreto, la primera frase se relaciona con la microinstrucción que tiene que ejecutarse a continuación. Si se selecciona, JAVY utiliza la descripción asociada y la plantilla descrita en la Sección 5.2 para decirle al usuario: "Para *introducir en la pila el operando de la instrucción* hay que *apilar el valor*".

Después de esto, el agente construye una nueva pregunta válida, utilizando otra plantilla y el *nombre* de la microinstrucción de la jerarquía conceptual: "Cuéntame algo más sobre eso de *apilar el valor*". Esta frase enlaza con el concepto de la jerarquía conceptual de esa microinstrucción.

Si el estudiante la selecciona, JAVY consulta su fuente de conocimiento para dar la explicación asociada (que, como muestra la Figura 2, es: "Hay que hacer un push. Para eso, lleva el operando del inventario a la pila de operandos"). En ese momento, el alumno podrá ir preguntando por los conceptos más profundos de la jerarquía conceptual, según se ha explicado en la Sección 5.1.

Esto debe dejar claro que la primera pregunta disponible al principio de toda conversación proporciona información sobre los *conceptos* que hacen referencia a instrucciones de la JVM y sus estructuras, y que dependerá de la instrucción y microinstrucción

actual. Por ejemplo, no se hablará nada de orientación a objetos, si se está en `bipush 5`.

Por otro lado, la segunda pregunta posible al iniciar la conversación ("¿Por qué tengo que hacer esta instrucción?") enlaza siempre con la información del ejercicio, más concretamente con el bloque más profundo que contiene la instrucción actual. Imaginemos que el usuario ya ha ejecutado la primera instrucción y se encara a la segunda, `bipush 3`. Si el estudiante se acerca al agente para iniciar la conversación, ahora la segunda pregunta enlazará con el bloque más profundo con esa instrucción, es decir, el etiquetado en la Figura 4 con el número 2. Cuando el estudiante la selecciona, JAVY utiliza la información adjunta: "Ya está apilado el primer operando (el 5). Ahora hay que apilar el segundo, el 3".

En ese momento, el usuario tiene disponibles *dos* preguntas. La primera enlaza con el *bloque padre* del código, que aparece en la Figura 4 (etiqueta 3). Esta conexión permite al estudiante *navegar* en el árbol de bloques, consiguiendo información sobre el proceso de compilación, es decir, sobre los *conceptos de alto nivel*. La segunda pregunta disponible enlaza con la *jerarquía conceptual* utilizando la conexión que se muestra en la Figura 5, por lo que la pregunta es "¿Qué hace la instrucción actual?" que termina en el concepto `bipush`.

8. Conclusiones y trabajo futuro

En este artículo hemos descrito JV^{2}M, un entorno virtual para la enseñanza de la compilación de Java y la estructura de su máquina virtual. Este entorno está habitado por JAVY, un agente pedagógico que acompaña al alumno a lo largo de toda la resolución del ejercicio para apoyarle proporcionándole ayuda cuando la necesite. El texto describe con cierto detalle el conocimiento almacenado por el sistema y su uso utilizando un ejemplo.

Sin embargo, el éxito de la aplicación requiere la disponibilidad de una gran base de ejercicios para poder recuperar y presentar el ejercicio más adecuado en cada momento. Actualmente nos estamos centrando en el uso de razonamiento basado en casos (CBR [1]) y en particular en su fase de adaptación para reducir la cantidad de ejercicios que el tutor humano tiene que crear manualmente e inyectar en el sistema [5]

Por último, aunque las explicaciones de JAVY son lo bastante contextualizadas, su generación está demasiado fijada de antemano en la base de conocimiento. Estamos planteándonos utilizar el sistema como un banco de pruebas para utilizar CBR conversacional [2] que guíe la conversación con JAVY.

Referencias

[1] A. Aamodt y E. Plaza. Case-Based Reasining: Foundational Issues, Methodological Variations, and System Approaches. *AI Communication*, 7(1):39-59, 3 1994.

[2] D. W. Aha, L. Breslow, y H. Muñoz-Avila. Conversational Case-Based Reasoning. *Applied Intelligence*, 14(1):9-32, 2001.

[3] W. Bares, L. Zettlemoyer, y J. C. Lester. Habitable 3D Learning Environments for Situated Learning. En *Proceedings of the Fourth International Conference on Intelligent Tutoring Systems*, páginas 76-85, San Antonio, TX, Agosto 1998

[4] EA Games. "Los Sims". Información disponible en http://thesims.ea.com/

[5] P. P. Gómez Martín, M. A. Gómez Martín, B. Díaz Agudo y P. A. González Calero. Opportunities for CBR in Learning-by-doing. En *Proceedings of the 6th International Conference on Case-Based Reasoning, ICCBR 2005*, páginas 267-281, Chicago, IL, Agosto 2005

[6] J. C. Lester, S. A. Converse, S. E. Kahler, S. T. Barlow, B. A. Stone y R. Bhogal. The Persona Effect: Affective Impact of Animated Pedagogical Agents. En *Proceedings Human Factors in Computing Systems (CHI'97)*, páginas 359-366, Atlanta, Marzo 1997

[7] T. Lindholm y F. Yellin. *The Java Virtual Machine Specification, 2ª edición*. Addison-Wesley, Oxford, 1999

[8] Radon Labs GmbH Game Development. Nebula Device 2004. Disponible en http://www.radonlabs.de/

[9] J. Rickel y W. L. Jonson. Animated Agents for Procedural Training in Virtual Reality: Perception, Cognition and Motor Control. *Applied Artificial Intelligence*, 13(4):343-382, 1999

[10] R. H. Stottler. Tactical Action Officer Intelligent Tutoring System (TAO ITS). En *Proceedings of the Industry/Interservice, Training, Simulation & Education Conference (I/ITSEC 2000)*, Noviembre 2000