Generación de visualizaciones de *software* usando ANIMALSCRIPT

Guido Rößling, Bernd Freisleben

Department of Electrical Engineering and Computer Science

University of Siegen

roessling@acm.org, freisleb@informatik.uni-siegen.de

Traductor: Cristóbal Pareja Flores
Departamento de Sistemas Informáticos y Programación
Universidad Complutense de Madrid
cpareja@sip.ucm.es

Resumen

Hay muchas herramientas *software* para visualización disponibles para el público. Sin embargo, muchas de ellas sólo tratan áreas de aplicación específicas, obligando a los usuarios a cambiar entre distintas herramientas. El paquete de visualización ANIMAL, con su lenguaje de visualización ANIMALSCRIPT incorporado, es una herramienta de animación de algoritmos que incluye un editor gráfico, y puede usarse para visualizaciones de *software* cualesquiera. Este artículo introduce los conceptos básicos de ANIMAL y ANIMALSCRIPT así como algunas prestaciones especiales, como apoyo multilingüe. ANIMALSCRIPT es también fácil de adaptar y extender con otras características más avanzadas.

1 Introducción

La creciente popularidad de la visualización de algoritmos y programas ha llevado al desarrollo de diversas herramientas, muchas de las cuales están diseñadas para tratar una cierta área de la visualización. La ventaja de implementar esas herramientas específicas es que pueden ser altamente especializadas y así ofrecer altas prestaciones. Sin embargo, normalmente estos sistemas resultan inútiles fuera de su campo específico. Por ejemplo, una herramienta desarrollada para visualizar algoritmos de codificación *run-length* (*RLE*) [4] es muy útil en este área, pero no para otros contenidos. Uno de los campos que ha mostrado un interés creciente por la visualización es la enseñanza. Aquí, un buen número de profesores se ha interesado en usar en sus clases animaciones de algoritmos previamente generadas, en vez de tener que efectuar las operaciones ellos mismos en la pizarra. Sin embargo, si el sistema usado para la visualización es muy específico, tendrán que usar alternativamente la herramienta y el sistema usual de presentación. Esto resulta más incómodo aún en conferencias, donde los oradores que desean mostrar el uso de un sistema de este tipo han de alternar entre la herramienta de visualización y la elegida para la presentación, como *PowerPoint.*

Al disponer de una sola herramienta que sea lo bastante genérica para mostrar la presentación y manejar el contenido presentado, se ahorra el tiempo de los cambios entre dichas herramientas. Y lo que es más importante, el presentador se ahorra el aprendizaje de distintas herramientas. Pero el sistema ha de ser lo bastante genérico

como para cubrir los requisitos típicos de las presentaciones y las operaciones propias del campo de aplicación. Es difícil que un solo sistema pueda ofrecer apoyo para visualizar cualquier clase de *software* que los usuarios puedan desear. Por consiguiente, es beneficioso disponer de una herramienta que sea extensible dinámicamente por los usuarios para atender sus necesidades específicas sin tener que leer y modificar largos fragmentos de código subyacente. Este artículo presenta el lenguaje de guión (*script language*) ANIMALSCRIPT, que extiende las capacidades del sistema de visualización ANIMAL [7]. ANIMALSCRIPT ofrece las operaciones básicas requeridas para visualizar *software*, pero también se puede adaptar fácilmente para tratar los requisitos específicos de otro contexto. Por ejemplo, ANIMALSCRIPT facilita la colocación relativa de objetos y los aspectos multilingües.

El artículo está organizado como sigue. El apartado 2 presenta algunas herramientas específicas para una cierta área de uso. El apartado 3 hace un breve repaso de los conceptos de visualización subyacentes en ANIMAL y de cómo se generan las visualizaciones. El apartado 4 presenta las capacidades de ANIMALSCRIPT, mostrando varias instantáneas del sistema. En el apartado 5 se mencionan algunas posibilidades para adaptar y extender las capacidades de ANIMAL y ANIMALSCRIPT. En el apartado sexto concluye el artículo, y se mencionan áreas de investigación futura.

2 Clasificación de herramientas disponibles

Con la creciente popularidad de la visualización del *software*, han surgido muchos sistemas en los últimos años. Puesto que no podemos cubrir todas las herramientas existentes, nos concentraremos en algunas que son representativas dentro de su categoría. Las herramientas se pueden clasificar según el tipo de entrada requerida para generar una visualización: movimientos de ratón en una interfaz gráfica de usuario (GUI), código fuente, llamadas a los métodos de una biblioteca especializada o a las órdenes de un lenguaje de guión.

Normalmente, las herramientas clásicas de presentaciones como *PowerPoint* de Microsoft o *StarImpress* de Sun esperan que el usuario dé todos los contenidos de la visualización manualmente a través de una GUI. La ventaja de este enfoque es que el usuario tiene un control absoluto sobre todos los elementos, al menos dentro de las posibilidades de la herramienta. Sin embargo, al estar obligados a hacerlo todo manualmente, la generación tampoco se puede automatizar. Si hiciera falta producir una segunda visualización sobre el mismo tema pero con valores cambiados, el usuario habría de partir de cero. Más aún, los paquetes de presentaciones están a menudo mejor dotados para generar imágenes visuales sensacionales en presentaciones cortas que para atender realmente a las necesidades que se exigen al presentar la semántica de procesos dinámicos, como algoritmos o estructuras de datos. Así, ofrecen una variedad razonable de modos de añadir elementos nuevos, pero pueden estar limitados a la hora de modificar los elementos visibles. Más todavía, el concepto subyacente a la presentación es una secuencia de dispositivas; esto supone que la siguiente diapositiva está inicialmente vacía, sin poder reflejar el contenido de la última, lo cual es inapropiado para visualizar algoritmos que efectúan diversas operaciones a lo largo de una colección de "diapositivas". En resumen, aunque las herramientas de presentaciones son muy útiles para ese cometido, no lo son tanto para mostrar el comportamiento de sistemas o algoritmos.

Los sistemas de visualización basados en el código fuente, como *DDD* [10], *Jeliot* [2], *Jeliot 2000* [1] y *WinHIPE* [5], presentan automáticamente una visualización de dicho

código subyacente. Esto también supone que es muy fácil generar automáticamente visualizaciones: el usuario sólo tiene que proporcionar dicho código fuente o cambiar los valores de los parámetros según sus necesidades específicas. Pero estos sistemas tienen algunas limitaciones. En primer lugar, el lenguaje de programación que se puede usar es fijo para el sistema. Por ejemplo, *DDD* trata programas en código máquina generado normalmente a partir de *C* o *C++,* mientras que *Jeliot* y *Jeliot 2000* trabajan con código fuente en Java, y *WinHIPE* está restringido a la programación funcional. Si la herramienta no soporta el lenguaje de programación usado en el contexto de la presentación, no puede usarse. Por otra parte, como la visualización se genera automáticamente, las posibilidades del usuario para cambiar las imágenes son limitadas. Por ejemplo, es improbable que el sistema permita deducir qué representación de una estructura de datos dada es la más apropiada semánticamente a partir del código subyacente: por ejemplo, una pila implementada con una lista enlazada podría no ser mostrada como una pila sino como una lista. Por otra parte, los presentadores pueden desear añadir texto explicativo, o adoptar un punto de vista más abstracto del algoritmo subyacente, como es el seudocódigo, para evitar confundir a los usuarios con los detalles de implementación. Ninguna de estas operaciones es normalmente soportada por estos tipos de herramientas. Finalmente, si el tema de la presentación no está disponible como un algoritmo, las herramientas no pueden usarse en absoluto.

Herramientas como *JAL* [8], de Silicon Graphics, se basan en llamadas a métodos de una biblioteca de programación específica (API) para generar la visualización. La facilidad de uso de la biblioteca, así como el punto hasta el que son soportadas las operaciones, depende del estado de la biblioteca. Se pueden señalar dos exigencias efectivas: el usuario ha de ser lo bastante experto en programación para escribir un programa que invoca correctamente los métodos de la biblioteca, y el lenguaje en que la librería está implementada debe encajar con el usado para el código subyacente. Si se dan ambos requisitos en una API, puede ser muy útil para generar visualizaciones; de lo contrario, la biblioteca es inadecuada y por tanto inaplicable.

Los sistemas que admiten datos de entrada en un formato específico, como un lenguaje de guión, ofrecen una solución parcial al problema de adecuación. La herramienta no se ocupa de cómo se genera la entrada de datos – manualmente por el usuario, o automáticamente por el código subyacente. Así pues, el usuario no tiene que ser un experto en programación. Sin embargo, los algoritmos implementados en cualquier lenguaje pueden ser modificados para generar los guiones apropiados. Por este hecho, las herramientas que admiten un guión como entrada se han popularizado en los últimos años: *JSamba* [9], *JAWAA* [6] y sistemas como *JellRep* y *JFlap* [3], específicos para contenidos de construcción de compiladores. El lenguaje de guión subyacente a *JSamba* es genérico, mientras que muchas otras herramientas ofrecen un lenguaje orientado a un contexto específico. Por ejemplo, *JAWAA* está orientado a algoritmos y estructuras de datos, y así ofrece elementos como árboles y grafos; sin embargo, es menos útil para presentaciones porque no soporta otras estructuras como listas de elementos. Por otra parte, con frecuencia los sistemas no permiten retroceder uno o más pasos en la visualización.

En general, los usuarios deben elegir el tipo de herramienta más apropiada para sus intenciones. Han de reflexionar sobre las posibilidades y limitaciones de cada enfoque, especialmente si también desean reutilizar la herramienta en otras aplicaciones menos específicas. Los sistemas basados en guiones ofrecen un equilibrio razonable entre las facilidades de la generación y las de automatización. Y

si el lenguaje de guión no es demasiado especializado, también permiten su reutilización en otros contextos.

En lo que sigue, examinaremos el lenguaje de guión ANIMALSCRIPT. Como ANIMALSCRIPT está implementado en un sistema de visualización, primero consideramos las características centrales de dicho sistema.

3 El sistema de visualización ANIMAL

El sistema de visualización ANIMAL [7] considera una visualización como una secuencia de pasos discretos. Cada paso puede contener un número arbitrario de transformaciones. Cada una trabaja sobre un número arbitrario de objetos a la vez. ANIMAL ofrece los elementos primitivos básicos polígono, línea poligonal, arco y texto; para facilitar su uso inicial para visualizaciones sobre introducción a la informática, también ofrece listas de elementos. Cada una de esas primitivas contiene unas pocas propiedades ajustables por el usuario. Algunas de ellas son comunes a todas las primitivas: color y profundidad, para resolver conflictos de solapamiento. Otras propiedades dependen de la primitiva considerada. Por ejemplo, un polígono puede tener un color de relleno, mientras que esto carece de sentido para una línea poligonal. Por otra parte, se puede asociar un puntero al principio o al fin de las líneas poligonales, lo que es inadecuado para los polígonos. Obsérvese que el número de nodos de una poligonal no está limitado en ANIMAL.

ANIMAL ofrece transformaciones para cambiar el estado de visibilidad de cualquier objeto, rotándolo, moviéndolo o cambiando sus propiedades de color. Esas operaciones están implementadas dependiendo del tipo de objeto sobre el que se ejecutan. Por ejemplo, cuando se mueve un elemento de una lista, el usuario puede elegir entre mover un elemento como está, ajustando sólo uno o más apuntadores, o mover el elemento con los apuntadores fijos. Se pueden definir movimientos complejos, siguiendo una línea poligonal o un arco arbitrarios. Los cambios de color pueden aplicarse a todas las propiedades de color de un objeto dado; así pues, es posible tener un polígono con un color de relleno distinto del color de su contorno. Todas las operaciones pueden temporizarse especificando su aparición y duración.

Los pasos individuales de visualización son normalmente elegidos usando botones como los de un aparato reproductor de vídeo, con la posibilidad de retroceder en la visualización. El autor de una visualización puede también indicar que un cierto paso avance automáticamente hasta el siguiente tras un cierto lapso de tiempo. Y también puede disponer de una regla graduada para mostrar el porcentaje de la visualización ya mostrado y ajustar el estado actual. Finalmente, el autor puede etiquetar los pasos, que se van recogiendo en una ventana separada, y así pueden usarse para saltar hasta el paso asociado a una etiqueta desde cualquier lugar de la visualización con un simple clic del ratón.

Para que ANIMAL sea útil a una audiencia lo más amplia posible, hemos seguido el principio de “mantenerlo simple”. ANIMAL ofrece una pequeña GUI que permite a los usuarios editar cualquier presentación usando el ratón, con prestaciones estándar como rejillas y posibilidades de arrastrar y soltar. El número de primitivas gráficas y transformaciones se ha limitado intencionadamente. Los usuarios que deseen una interfaz más potente pueden desplazarse al lenguaje de guión predefinido ANIMALSCRIPT una vez que se han familiarizado con el uso de ANIMAL. La interfaz gráfica y todos los mensajes de salida pueden traducirse a otro idioma mediante un

simple clic del ratón. Por el momento, sólo podemos proporcionar apoyo en alemán e inglés. El apoyo para otros lenguajes como español, francés e italiano requiere sólo traducir un archivo de texto, sin recompilación. La figura 1 enseña la ventana principal de ANIMAL después de que haber cambiado el lenguaje. Nótese que la traducción afecta a los menús y a sus elementos, a los textos explicativos y a las combinaciones abreviadas de teclas.

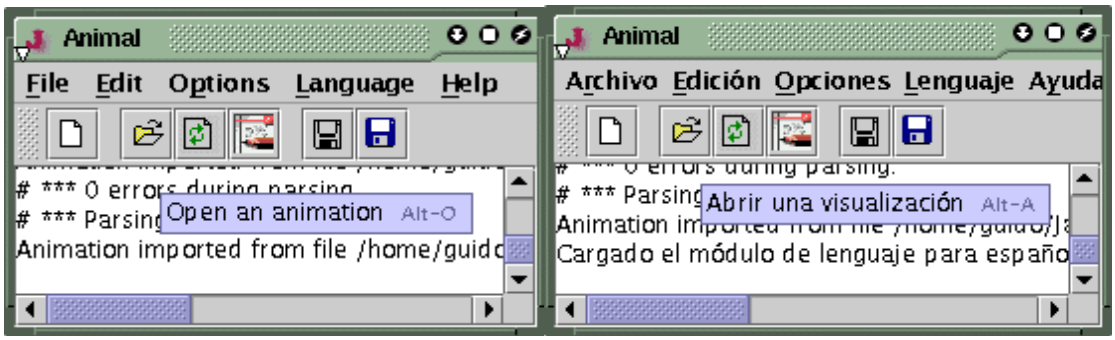

Fig. 1: Efecto de traducir el GUI de ANIMAL sobre los títulos, mensajes y combinaciones de teclas

ANIMAL también ofrece una interfaz para filtros de importación, y proporciona algunos filtros de exportación. Actualmente, sólo se pueden importar visualizaciones de *JSamba*, y pueden exportarse como películas en formato Quicktime, unos pocos formatos más de imágenes y en XML. Otros formatos son relativamente sencillos de implementar si está disponible la necesaria definición del formato objeto destino. Todos los filtros están registrados en un archivo de configuración que se puede actualizar cuando se incluye un nuevo filtro. La adición de nuevos filtros de importación o exportación no requiere acceso al código de base de ANIMAL, ni precisa recompilación.

4 Características escogidas de ANIMALSCRIPT

ANIMALSCRIPT ofrece un catálogo de prestaciones extenso, que no podemos cubrir aquí en su totalidad. Por consiguiente, empezaremos con una breve introducción a ANIMALSCRIPT y luego nos concentraremos en dos características especiales: colocación de objetos relativos y aspectos multilingües. ANIMALSCRIPT contiene órdenes para manejar todas las características de ANIMAL relacionadas en el apartado precedente. También incluye órdenes específicas para objetos particulares frecuentes, como cuadrados en vez de polígonos en general. Al igual que otros lenguajes para visualización [6, 9], ANIMALSCRIPT está organizado por líneas: cada una contiene exactamente una orden o comentario. Se ha de dar un nombre a cada elemento gráfico usado durante una visualización para permitir su uso en transformaciones ulteriores. Esos nombres pueden ser intercambiados mediante una orden `swap`, lo cual facilita mucho las transformaciones al usuario.

Algunas de las características avanzadas de ANIMALSCRIPT van más allá de las operaciones disponibles en el GUI de ANIMAL. Por ejemplo, basta una simple orden de guión para enlazar o desunir elementos de una lista. Otras órdenes especiales predefinidas en ANIMALSCRIPT facilitan el manejo de *arrays* que incluyan apuntadores a componentes, sobreescribir valores en los elementos e intercambiar los

contenidos de dos componentes de un *array*. También se da apoyo para usar el código o seudocódigo con flexibilidad, manejando el sangrado y resaltando la línea del código en curso, o su contexto, como es por ejemplo el bucle en que está encerrado.

La principal diferencia entre ANIMALSCRIPT y los lenguajes de guión usados en otros sistemas como *JAWAA* y *JSamba* reside en las posibilidades de ubicación relativa de elementos, el apoyo multilingüe y la emisión de diagnósticos. La ubicación relativa de elementos en ANIMALSCRIPT libera al autor de registrar y mantener la posición de cada objeto en cada momento, permitiéndole un acceso directo a su caja circundante, el mínimo rectángulo que contiene el elemento entero. Los usuarios tienen las siguientes posibilidades para especificar la posición de cada elemento:

- Coordenadas absolutas, como (400, 200).
- Ubicación relativa a un objeto previamente definido. El punto de referencia puede tomarse el centro de un objeto, o cualquiera de los ocho puntos cardinales típicos de su caja circundante.
- Ubicación relativa a un nodo cualquiera de una línea poligonal o de un polígono previamente definido.
- Ubicación relativa a la última coordenada definida, lo que resulta útil para operaciones como "avanzar 20 *píxeles*".
- Ubicación relativa a una posición, previamente definida mediante cualquiera de las técnicas anteriores.

La figura 2 muestra estas aplicaciones y el código ANIMALSCRIPT requerido para producirlas. Obsérvese que el cuadrado relleno se ha situado con coordenadas absolutas. La posición de la caja vacía se ha especificado mediante un desplazamiento desde la esquina sudeste del cuadrado relleno, y el texto de la parte superior derecha se ha colocado relativo al nodo del cuadrado. El último nodo usado en la visualización fue la esquina superior izquierda del texto "*relative to last node*", provocando que el texto de la parte centro izquierda de la pantalla se coloque en posición relativa a la esquina superior izquierda del texto.

Como se mencionó antes, ANIMAL ofrece un método para traducir todas las entradas de la GUI. ANIMALSCRIPT da un paso más permitiendo al usuario dar la traducción

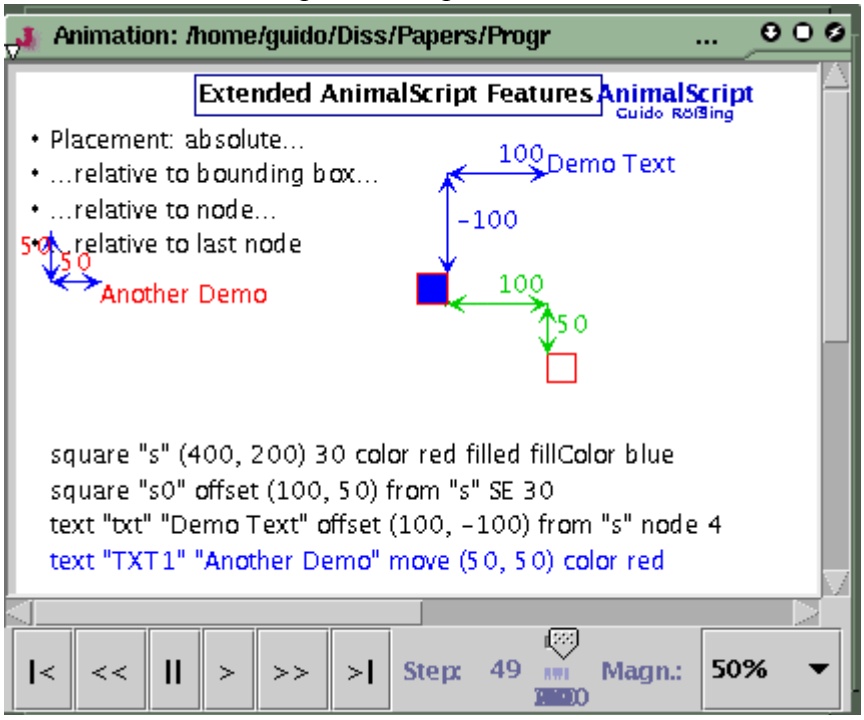

Fig. 2: Colocación de objetos usando ANIMALSCRIPT: coordenadas absolutas, relativas a la caja circundante y al

de cada elemento textual. Los idiomas soportados en la visualización deben enumerarse al principio del archivo de visualización. Entonces, a cada pieza de texto se le puede dar una traducción prefijada según el código del lenguaje. Al cargar la visualización, se pregunta al usuario por el lenguaje que va a usar.

Se observa que la traducción de un texto rara vez tiene la misma longitud que su correspondiente en el idioma original; por ejemplo, los textos en alemán suelen ser más largos que sus equivalentes en inglés. Por tanto, no basta con dar una simple traducción: por ejemplo, los elementos que supuestamente se van a colocar a la derecha de un mensaje traducible tendrían que disponer de un cierto margen, o bien deberían poder solaparse con el propio mensaje. Este problema puede resolverse usando la ubicación relativa alineada con la caja circundante del elemento traducido: después de que el texto se ha leído y traducido al lenguaje escogido, se conoce su posición y tamaño, de forma que los elementos y efectos que dependen de su colocación se pueden determinar correctamente. Una consecuencia de esto es que los usuarios que trabajan en un contexto internacional no están obligados a generar visualizaciones separadas de acuerdo con la audiencia destinataria. En vez de eso, pueden encapsular las traducciones en la misma visualización y elegir el lenguaje de acuerdo con el contexto idiomático en que están en un momento dado. La figura 3 muestra un ejemplo de visualización con una versión en inglés y otra en español. Obsérvese el cambio de anchura en el rótulo superior, que usa coordenadas relativas para encajar bajo la cabecera.

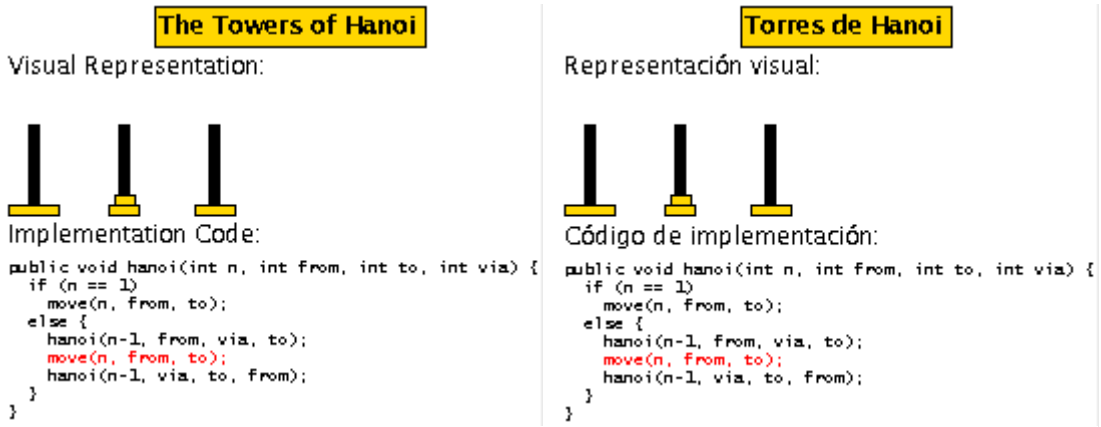

Fig. 3: Ejemplo de visualización usando el apoyo multilingüe de ANIMALSCRIPT

La generación de diagnósticos en ANIMALSCRIPT es de gran ayuda, especialmente para usuarios principiantes. La funcionalidad de la orden `echo` de ANIMALSCRIPT aporta información sobre la posición (esquina superior izquierda) o la caja circundante de los objetos, sobre los números de las componentes de una agrupación de objetos o de los objetos visibles en un momento dado, sobre comentarios de texto cualesquiera, o sobre la sintaxis de una orden de ANIMALSCRIPT. Algunas de esas aplicaciones pueden verse en la figura 4.

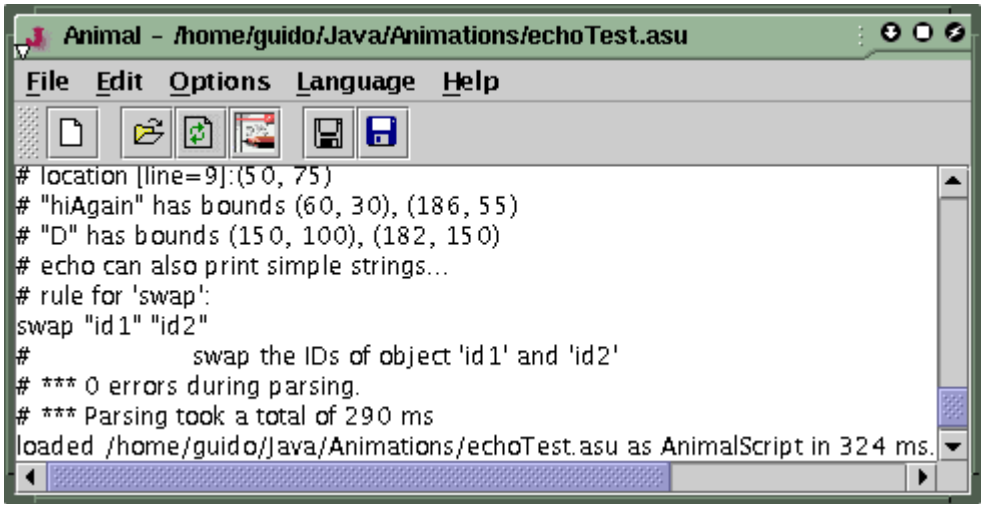

Fig. 4: Ejemplo de aplicaciones de la orden echo

5 Extensión de ANIMALSCRIPT

Nosotros creemos que ANIMALSCRIPT contiene diversas prestaciones beneficiosas para los usuarios, y que permiten el uso de la herramienta en un contexto genérico. Sin embargo, somos conscientes de que hay muchas áreas de aplicación para las cuales sería de ayuda apoyo específico. Por ejemplo, actualmente no hay soporte para algunas estructuras de datos como los grafos o para aplicaciones teóricas como las máquinas de Turing o simulación de autómatas de estados finitos. Como no podemos esperar satisfacer los deseos de todos los usuarios, hemos adaptado la estructura del *parser* de ANIMALSCRIPT para que sea fácilmente extensible.

Los usuarios que deseen proporcionar una cierta extensión de ANIMALSCRIPT deberían comprobar primero si ya hay una implementación disponible de dicha extensión por un tercero. De lo contrario, nosotros podemos indicarles los sencillos pasos que han de seguir para extender ANIMALSCRIPT. Para facilitar esta tarea en lo posible, ofrecemos un analizador sintáctico que traduce un subconjunto de las reglas BNF a código Java útil para analizar sintácticamente las órdenes y extraer las propiedades. Entonces, el usuario sólo tendrá que proporcionar algunas partes del código, especialmente la conversión de las propiedades en un conjunto de objetos primitivos y transformaciones de ANIMAL.

Las componentes del analizador sintáctico de ANIMALSCRIPT se cargan dinámicamente a partir de un archivo de configuración. Por tanto, los usuarios que han implementado o cargado una extensión sólo tienen que actualizar el archivo de configuración. No es necesario ningún cambio en el sistema ANIMAL ni en ANIMALSCRIPT, ni tampoco se requiere recompilación alguna. Esto también es válido para el proceso de extender la funcionalidad de ANIMALSCRIPT, que no requiere modificaciones o recompilación del código existente. La única excepción a esta regla afecta a las características ya implementadas, tales como la reparación de errores.

6 Conclusiones

En este artículo, hemos repasado algunas características de ANIMAL y ANIMALSCRIPT.

ANIMAL es útil para visualizaciones de *software* genéricas y no requiere ningún conocimiento o habilidad especiales del usuario debido a su editor gráfico. ANIMALSCRIPT añade una mayor funcionalidad para los usuarios que desean preparar la generación de animaciones mediante guiones, como es usual en otros sistemas [6, 9]. Actualmente, ANIMALSCRIPT sólo soporta un número limitado de operaciones. Sin embargo, esas operaciones incluyen características avanzadas que no se encuentran usualmente en muchas aplicaciones comparables, tales como la ubicación relativa de objetos, el apoyo multilingüe y la emisión de diagnósticos. Para atender a una audiencia lo más amplia posible, tanto el GUI de ANIMAL como las visualizaciones de ANIMALSCRIPT se pueden traducir fácilmente a otros idiomas. Especialmente cuando se usa en conjunción con la colocación relativa de objetos, los autores sólo han de proporcionar las traducciones sin tener que adaptar las ubicaciones de los objetos.

Esperamos encontrar usuarios dispuestos a ayudarnos a añadir nuevas características a ANIMALSCRIPT una vez documentados sobre cómo se puede llevar a cabo. Muchos cambios no requieren modificar el código de ANIMALSCRIPT. Otras posibles extensiones de ANIMALSCRIPT son el soporte para imágenes, estructuras de datos y figuras específicas para informática, y las características típicas que ofrecen las herramientas de presentación. En nuestra experiencia, generar una visualización mediante guiones es mucho más rápido que visualmente, una vez que el usuario se ha familiarizado con los conceptos subyacentes y las órdenes habilitadas. Esto es también válido para presentaciones de conferencias, a pesar de que muchos paquetes de presentaciones se apoyan fuertemente en editores gráficos. Pese a la carencia de órdenes para listas de elementos en ANIMALSCRIPT, hemos usado ANIMAL en conjunción con ANIMALSCRIPT como la principal herramienta para presentaciones de conferencias sin problemas o dificultades.

Referencias

[1] M. Ben-Ari (2000): *An Extended Experiment with Jeliot 2000.* First International Program Visualization Workshop, Porvoo, Finlandia.

[2] J. Haajanen, M. Pesonius, E. Sutinen, J. Tarhio, T. Teräsvirta, P. Vanninnen (1997): *Animation of User Algorithms on the Web*. IEEE Symposium on Visual Languages Proceedings (VL'97), Capri, Italia, págs. 360-367.

[3] T. Hung, S. H. Rodger (1997*): Increasing Visualization and Interaction in the Automata Theory Course*. 31st SIGCSE Technical Symposium on Computer Science Education, Austin, Texas, págs. 6-10.

[4] S. Khuri, H.-C. Hsu (2000): *Interactive Packages for Learning Image Compression Algorithms.* 5th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2000), Helsinki, Finlandia, págs. 73-76.

[5] F. Naharro-Berrocal, C. Pareja-Flores y J. Á. Velázquez-Iturbide, *Automatic generation of algorithm animations in a programming environment*, 30th ASEE/IEEE Frontiers in Education Conference, Volume II, Stipes Publishing L. L. C., 2000, págs. 6-12 (sesión S2C).

[6] W. C. Pierson, S. Rodger (1998): *Web-based Animation of Data Structures Using JAWAA*. 29th SIGCSE Technical Symposium on Computer Science

Education, Atlanta, Georgia, págs. 267-271.

[7] G. Rößling, M. Schüler, B. Freisleben (2000): *The ANIMAL Algorithm Animation System*. 5th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2000), Helsinki, Finlandia, págs. 37-40.

[8] Silicon Graphics (1998): *JAL Algorithm Animation*. Available online at http://reality.sgi.com/austern/java/demo/demo.html

[9] J. Stasko (1998): *Samba Algorithm Animation System*. Available online at http://www.cc.gatech.edu/gvu/softviz/algoanim/samba.html

[10] A. Zeller (2000): *Dynamic Display of Data Structures Using DDD*. First International Program Visualization Workshop, Porvoo, Finlandia.

Guido Rößling nació en Darmstadt, Alemania, y se licenció en Informática en 1996, en la Universidad Tecnológica de Darmstadt. Desde 1996, es un investigador a tiempo completo en la Universidad de Siegen. Desde 1997, es también profesor de *Informática I/II* y *Programación* en la Berufsakademie de Mannheim. Actualmente trabaja en su tesis, diseñando una herramienta de visualización extensible y configurable dinámicamente. Sus intereses en investigación incluyen la utilización eficaz de multimedia en la enseñanza, animaciones de algoritmos y generación automática de representaciones de datos. Es miembro de ACM, IEEE, AACE y GI.