

VAST: Visualizador de Árboles Sintácticos. Evaluación de usabilidad

Francisco J. Almeida-Martínez
Lenguajes y Sistemas Informáticos I
Universidad Rey Juan Carlos
Tulipán s/n
28933 Móstoles
francisco.almeida@urjc.es

Jaime Urquiza-Fuentes
Lenguajes y Sistemas Informáticos I
Universidad Rey Juan Carlos
Tulipán s/n
28933 Móstoles
jaime.urquiza@urjc.es

Resumen

Esta comunicación describe VAST, una herramienta educativa que se ha diseñado para ser utilizada en clases de compiladores y procesadores del lenguaje. La versión actual permite generar y visualizar los árboles sintácticos y su proceso de construcción. Las principales ventajas de VAST son: independencia del generador de parsers utilizado, permite que los estudiantes visualicen el comportamiento de los parsers que desarrollan y consta de una interfaz diseñada para manejar cómodamente árboles sintácticos muy grandes. Describimos diferentes maneras educativas de utilizar VAST, así como una evaluación de usabilidad.

1. Introducción

Asignaturas como *Procesadores de Lenguajes* o *Compiladores* son tradicionalmente complejas para los estudiantes. Su enseñanza se suele estructurar en las fases de análisis léxico y sintáctico, traducción dirigida por sintaxis y, si se trata de asignaturas de compiladores, las tablas de símbolos, análisis semántico, generación de código intermedio y final. Las fases de análisis léxico y sintáctico se basan claramente en la teoría de lenguajes formales. Sin embargo, la fase de traducción dirigida por sintaxis, y su utilización en compiladores como análisis semántico y generación de código intermedio, no tienen una relación tan clara con la teoría de lenguajes formales, sin embargo, necesitan por parte del alumno la comprensión de la estructura sintáctica subyacente.

La complejidad del diseño de parsers ha disminuido gracias a las herramientas de generación automática de traductores, como la pareja *Lex* y *Yacc* [11]. Estas herramientas están orientadas a un uso profesional lo que puede dificultar el aprendizaje de los alumnos.

La fase de análisis léxico está muy ligada con la teoría subyacente y a las herramientas de generación automática correspondientes. Sin embargo, en el caso del análisis sintáctico esta relación parece cercana. Existen conceptos en la asignatura cuya interpretación en este tipo de herramientas es muy complicado, pues o necesita de un experto (tablas acción e ir-a) o bien simplemente no los implementan (Árbol Sintáctico (AS)).

En este artículo presentamos VAST¹[2], un software educativo que permite la visualización de los AS de forma casi independiente de la herramienta generadora utilizada. En esta nueva versión VAST tiene soporte para analizadores LL(1), visualización de la pila, opciones de configuración, mejora del interfaz y diferentes distribuciones según las características del árbol. Además, describimos una evaluación de usabilidad y otra de observación que hemos hecho de la herramienta.

El resto de la comunicación se estructura como sigue. En la sección 2 describimos los trabajos relacionados, en la sección 3 describimos la implementación de VAST. En la sección 4 detallamos las diferentes visualizaciones que ofrece. En la sección 5 describimos su uso educativo, en la sección 6 describimos las eva-

¹<http://www.lite.etsii.urjc.es/vast>

luaciones realizadas hasta el momento. Y para terminar, en la sección 7 explicamos nuestras conclusiones así como las líneas de trabajo futuro.

2. Trabajos relacionados

Existen herramientas que tratan de cubrir el hueco existente entre la teoría y la práctica del análisis sintáctico, pero o lo hacen de forma parcial o demasiado particular.

En un extremo encontramos una herramienta educativa como JFlap [17]. Ésta representa una opción válida a nivel de fundamentos teóricos, además su diseño persigue objetivos específicamente pedagógicos. Sin embargo, JFlap no permite que los alumnos generen sus propios parsers.

En el otro extremo encontramos herramientas que visualizan el proceso de reconocimiento con un enfoque más práctico. Hemos encontrado nueve herramientas de este tipo.

ICOMP[3], VISICLANG[15], APA²[10] y TREE-VIEWER[18] no permiten generación de parsers. Por otro lado, VCOCO[16] genera analizadores pero las visualizaciones que ofrece están más orientadas a usuarios avanzados. Por último, GYACC[12], CUPV[9], LISA[13] y ANTLRWorks[4] presentan visualizaciones más elaboradas, permitiendo además la generación de analizadores. Sin embargo, todas ellas dependen de una notación propia, o de un sistema con el que están íntimamente ligadas. Este enfoque restringe su uso educativo ya que hace al generador y las visualizaciones totalmente dependientes.

Al no existir ninguna herramienta que contemple todos los tipos fundamentales de analizadores y visualice sus distintas dimensiones – algoritmo de funcionamiento, árbol sintáctico – es probable que el profesor tenga que usar más de una, cambiando de notación, visualización y organización. En este contexto, los estudiantes tienen que aprender cómo utilizar herramientas diferentes: notación, proceso de construcción, interpretación de los mensajes de salida

²Realmente esta herramienta no tiene nombre, hemos escogido el acrónimo del título del artículo donde se describe

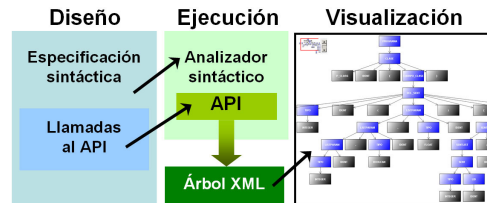


Figura 1: Esquema de funcionamiento de VAST

-conflictos, matriz de transición o conjunto de items-. Además, el profesor tiene que dedicar tiempo para familiarizarse con los diferentes entornos, y para planear su integración en el curso. Esto puede hacer más difícil su uso en entornos educativos [14].

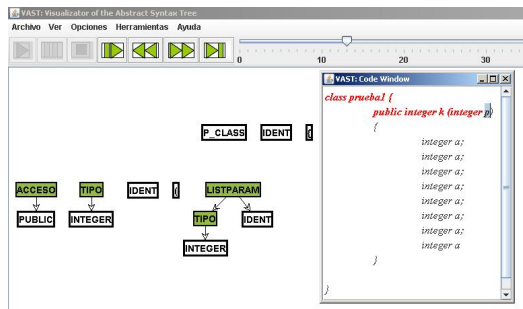
Por lo tanto, nuestros esfuerzos se centrarán en cubrir el hueco entre la teoría y la práctica, visualizando el árbol sintáctico, su proceso de construcción, la recuperación de errores, ofreciendo un interfaz adecuado para su manipulación, pero además haciéndolo de forma casi independiente³ del generador de analizadores sintácticos que se utilice.

3. Implementación de VAST

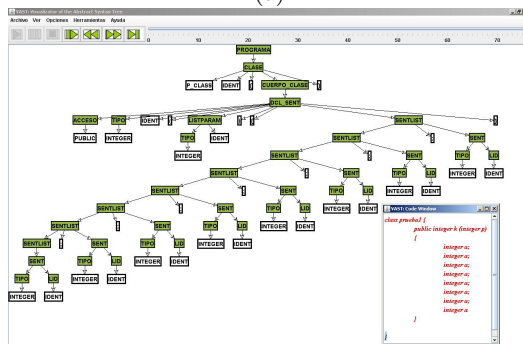
El diseño de VAST permite separar la visualización de AS de su proceso de generación. Para ello VAST ofrece una API (VASTAPI) que sirve para la creación del AS y una interfaz gráfica (VASTVIEW) para su visualización. VASTAPI se relaciona con el procesamiento del lenguaje transformando llamadas del estilo "Se ha aplicado la reducción/derivación $X ::= Z$ " en la información que posteriormente VASTVIEW representará de forma visual. En concreto VASTAPI ofrece un método llamado *addProduction* (*antecedente*, *consecuente*) que permite comunicar qué producción se ha aplicado. Con este diseño se logra que VAST sea independiente del generador de parsers que se utilice.

VASTAPI genera una representación intermedia en XML del contenido del AS. Actual-

³Decimos casi independiente porque VAST está desarrollado en Java, y por lo tanto el generador de analizadores también deberá generar código Java



(a)



(b)

Figura 2: Pasos de la animación del reconocimiento de una cadena: en el paso (a) se muestra un estado intermedio de reproducción y en el (b) se ha creado completamente el AS

mente puede trabajar tanto con analizadores LR(1) como con LL(1). Gracias al funcionamiento interno de VASTAPI, el usuario no tiene que especificar de qué tipo de analizador se trata, ya que esto se realiza automáticamente.

En el desarrollo de VASTAPI se ha utilizado JDOM [8]. Nótese, que en el caso de los analizadores LR(1) el proceso de creación del AS es inverso al de creación del XML, por ello no se hará efectiva su creación hasta haber aplicado todas las reducciones.

En la figura 1 se puede observar el esquema básico de implementación y utilización de VAST.

4. Trabajando con VAST

En esta sección explicamos el uso básico de VAST. Primero, el usuario tiene que introducir su propia especificación usando los métodos de VASTAPI. Una vez que se ha generado el parser, su ejecución producirá la información necesaria para que VASTVIEW visualice el AS y anime su proceso de construcción.

4.1. Generación de las visualizaciones

El objetivo principal de VAST es permitir la generación y manipulación del AS de manera independiente del generador de parsers utilizado. La representación en XML creada por VASTAPI sirve de fuente de datos para VASTVIEW que visualizará el AS. Las llamadas a VASTAPI deben insertarse como parte de las acciones asociadas a las reglas gramaticales.

4.2. Visualización del AS con VASTVIEW

La representación gráfica que se ha elegido para la visualización del AS es la estructura arborea resultante del reconocimiento de la cadena de entrada. Además, durante el reconocimiento VAST puede distinguir entre los diferentes tipos de nodos que componen el AS: nodos terminales (T), nodos no terminales (NT) y nodos de error (NE). Véase la figura 3.

Los nodos *T* serán los nodos hojas del árbol, los nodos *NT* serán aquellos nodos internos al AS y los nodos *NE* representarán los puntos en los que el analizador se recupera de un error sintáctico. La representación de estos últimos sólo aparecerá si el usuario utiliza la recuperación de errores. La visualización de los nodos *NE* permite determinar el lugar exacto en el que se ha recuperado un error y la cantidad de elementos reconocidos. Los elementos que se han reconocido antes de producirse un error sintáctico aparecerán como nodos hijos del error correspondiente.

En la versión actual de VAST la recuperación de errores se consigue de una forma más eficiente con los generadores de analizadores LR(1), ya que de los generadores de analiza-

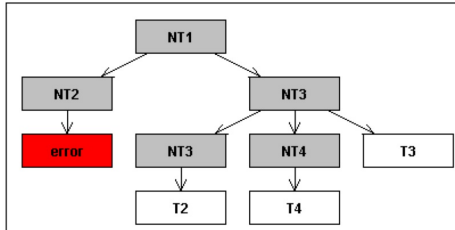


Figura 3: Ejemplo de representación de símbolos terminales (Tx), no terminales (NTx) y nodos de recuperación de errores

dores LL(1) utilizados, ninguno realiza de manera similar la recuperación de errores.

Probablemente, los AS generados por los parsers creados por los estudiantes son grandes y sin ninguna estructura prefijada (simetría, anchura o altura). Por ello, VASTVIEW está formada por conjunto de interfaces gráficas especialmente diseñadas para manejar este tipo de árboles.

Las interfaces, ofrecen diferentes formas de distribuir la visualización dependiendo de la apariencia del árbol. Existe una distribución para los árboles verticales y horizontales, además, cuando un AS no puede clasificarse como horizontal/vertical VASTVIEW ofrece una representación flotante.

Cada interfaz contiene una vista global y detallada del AS, junto con la funcionalidad de zoom, agregación de algunas partes del AS y animación de su proceso de construcción. La vista global y detallada permiten que el usuario pueda manipular fácilmente el AS. La vista global señala la parte visible del AS en la vista detallada. Las funcionalidades de la vista detallada dan al estudiante la posibilidad de examinar el AS con zoom, agregación y desplazamiento, todo esto sincronizado con la vista global. El zoom en la vista detallada permite a los estudiantes fijar su atención en determinadas partes del AS y ajustar el nivel de detalle. La agregación de subárboles -acciones de expandir/resumir nodos- permite que los estudiantes obtengan una representación del AS en la que sólo son visibles las partes de interés. Por último, las barras de desplazamiento

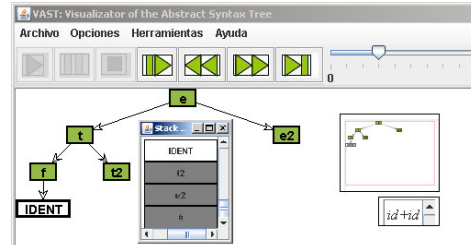


Figura 4: Interfaz de VASTVIEW

permiten que los estudiantes puedan desplazarse por la vista detallada (véase la figura 4). El desplazamiento puede realizarse con las barras de desplazamiento de la vista detallada o bien moviendo el área señalada en la vista global.

Además de la visualización del AS, VASTVIEW visualiza también la pila asociada a este tipo de autómatas, incluyendo opciones de configuración donde cambiar colores, tamaños, fuentes etc, tanto para el AS como para la pila.

4.3. Animación de la construcción del AS

La animación del proceso de construcción del AS se realiza usando diferentes estados intermedios generados y ordenados por VASTAPI. La animación de este proceso ayuda a los estudiantes a entender cómo la cadena de entrada se reconoce por medio de desplazamientos y reducciones -conexiones de un nodo existente con otro nodo no terminal-, junto con la recuperación de errores.

Para reproducir una animación, VASTVIEW consta de unos controles VCR, junto con una barra de desplazamiento que permite desplazarse rápidamente a un estado intermedio de la animación. Durante el proceso de construcción, el AS cambia su forma, área y contenido. Por ello, la interfaz podría adaptarse a cada estado usando la política "best-fit", cambiando la localización de los nodos y otras propiedades de la representación gráfica. Hemos decidido mantener esas propiedades a menos que los estudiantes las cambien. Todos los nodos mantendrán su localización desde su creación hasta el final del proceso de construcción. Esto

evitará que los estudiantes se distraigan, por ejemplo, mientras buscan la nueva localización de un nodo ya existente.

La animación del proceso de construcción del AS está sincronizado con la visualización de la pila para analizadores LR(1) y LL(1) (véase la figura 2). Además, se incorpora otra visualización donde se permite ver el histórico de la pila, es decir, los distintos estados por los que ha ido pasando esta durante la reproducción del proceso de construcción.

5. Uso educativo de VAST

VAST permite que los estudiantes y profesores observen y manipulen el AS y su proceso de construcción. Desde el punto de vista del profesor, VAST puede usarse de forma independiente del generador de parsers utilizado, tanto para LL(1) como para LR(1). Por esta razón, el profesor sólo debe dedicar tiempo una vez a aprender a generar visualizaciones con VAST. Además, la representación gráfica y la interfaz de visualización del AS es siempre la misma, lo que evita que los alumnos tengan que aprender una nueva para cada técnica de análisis.

Desde el punto de vista de los estudiantes, además de ver el comportamiento del parser desarrollado por el profesor, pueden desarrollar sus propias visualizaciones y comprobar el comportamiento de los parsers que han creado. En un campo relacionado [7] se ha mostrado que un uso más activo de las visualizaciones por parte de los alumnos podría mejorar su proceso de aprendizaje.

6. Evaluación de VAST

En esta sección describimos en la medida de lo posible la evaluación realizada con VAST. Describimos los estudiantes que participaron en la evaluación, el diseño experimental de la evaluación y las tareas realizadas en la evaluación. Para más detalle existe un informe técnico disponible en [1].

6.1. Asignatura

La evaluación se realizó en el desarrollo de la asignatura Procesadores del Lenguaje de Ingeniería Informática en la Universidad Rey Juan Carlos. Participaron 59 alumnos de forma incentivada con un aumento del 2% en la nota final sólo si aprobaban el examen.

Los alumnos realizaron un test de estilos de aprendizaje [6] cuyos resultados establecen que el estilo de aprendizaje de los alumnos es eminentemente activo y visual.

6.2. Diseño del experimento

La evaluación se ha diseñado como un experimento controlado más un estudio de observación. Como se trataba de evaluar una herramienta educativa hemos utilizado guías de un verdadero estudio experimental [5]. Se trata de una evaluación controlada porque se dividió la asignatura en dos grupos, el grupo de tratamiento, que trabajaba con VAST, y el grupo de control, que utilizaba ANTLRWorks.

La creación de los grupos fue parcialmente aleatoria, ambos estaban equilibrados en la medida de lo posible, utilizando las puntuaciones de un pretest. Los estudiantes se clasificaban usando las puntuaciones del pretest pero la asignación de un grupo (tratamiento/control) era aleatoria.

En este caso, la variable independiente era la herramienta utilizada: VAST para el grupo de tratamiento o ANTLRWorks para el grupo de control.

Las variables dependientes permitían saber la opinión de los alumnos sobre cuatro aspectos diferentes: facilidad de uso, mejora de aprendizaje, calidad de la herramienta y satisfacción del estudiante.

6.3. Tareas

Todas las tareas realizadas por los estudiantes debían estar documentadas con visualizaciones y explicaciones textuales. Las tareas consistían en tres ejercicios sobre conceptos de los analizadores LL(1).

En el grupo de control los estudiantes debían implementar la gramática utilizando el

editor de ANTLRWorks. Para ver las visualizaciones estáticas debían utilizar el intérprete. Sin embargo, si querían ver el proceso de animación tenían que usar el depurador. En ambos casos era necesario especificar la cadena de entrada, la regla de comienzo y el final de línea de la plataforma.

Por otro lado, en el grupo de tratamiento los estudiantes debían codificar la especificación utilizando un editor de propósito general. Luego debían introducir las llamadas a VASTAPI y generar el parser con ANTLR. Para cada ejecución del parser, éste generaba una representación XML que era visualizada con VASTVIEW.

En el primer ejercicio se pedía que los estudiantes cambiasen la precedencia de los operadores binarios de una gramática. La gramática original era:

```
S := F N
N ::= + F N | - F N | * F N | / F N |
F ::= id | cte | ( S )
```

La gramática resultante tenía que dar la mayor precedencia a * y /, luego una intermedia al - y la menor al +. Además, en todos los casos la asociatividad era por la derecha.

En el segundo ejercicio, los estudiantes tenían que generar dos cadenas de entrada con errores sintácticos generados por: el símbolo de comienzo de la regla y el siguiente terminal esperado.

Por último, en el tercer ejercicio se preguntaba por la recuperación de errores en modo pánico. Para cada no terminal, se debía generar una cadena errónea, de tal forma que los símbolos de sincronización perteneciesen al conjunto SIGUIENTE del no terminal en cuestión.

6.4. Resultados de la evaluación

Los resultados de la evaluación los podemos dividir en observaciones por los profesores y respuesta de los cuestionarios de los alumnos.

Durante el experimento, los profesores observaban cómo trabajaban los estudiantes con las herramientas, encontraban errores y funcionalidades útiles y otras que no lo eran tanto.

El grupo de control trabajaba con ANTLRWorks, un entorno cerrado que encapsula el editor y la generación de visualizaciones. A los estudiantes les gustaban el editor y como diferenciaba entre reglas léxicas y sintácticas. Sin embargo, detectaron algunos posibles errores: ruta del fichero con blancos y final de línea plataforma. En este último, los alumnos que usaban Mac tenían que usar el final de línea de Windows. Algunos estudiantes tuvieron problemas con el debugger de ANTLRWorks, ya que el puerto que requería libre no estaba disponible.

El grupo de tratamiento utilizaba ANTLR para generar el parser y VAST para generar y visualizar el AS. Los estudiantes utilizaron editores estándares para especificar la gramática e incluir las llamadas a VASTAPI. Por último, utilizaron dos herramientas para compilar/ejecutar el parser, algunos javac/java desde la consola y otros Eclipse IDE. A los estudiantes les gustaban las animaciones generadas por VAST y prestaban especial atención al orden de construcción del AS. Detectaron un error en el algoritmo de dibujo de árboles. Se tuvieron serios problemas durante la compilación debido a la versión del JDK, pues las llamadas a VASTAPI no se compilaban correctamente. Algunos lograron solucionar este problema, otros decidieron utilizar Eclipse y otros decidieron trabajar fuera del laboratorio.

Otro problema que se detectó, fue el número de ventanas abiertas de forma simultánea: el editor, la consola, la ventana de compilación y VASTVIEW. Nótese que los estudiantes no encontraron que esta situación fuera realmente un problema, sino que se trata de un punto donde introducir una mejora.

Los resultados de los cuestionarios muestran que la opinión de los estudiantes sobre ANTLRWorks y VASTVIEW es bastante similar, entorno a 4 (en una escala de 1 a 5). La única diferencia que se detectó fue la ayuda al aprendizaje del comportamiento de la pila (ANTLRworks 2.38, VASTVIEW 3.46). Sin embargo, la opinión de los estudiantes acerca de VASTAPI es peor que la de ANTLRworks, tanto en facilidad de uso (ANTLRworks 4.36, VASTAPI 3.15) como en satisfac-

ción (ANTLRWorks 4.09, VASTAPI 3.45).

7. Conclusiones

Hemos presentado la herramienta educativa VAST que permite la visualización de AS. Hemos identificado un hueco entre los conceptos que se imparten en la teoría y las herramientas generadoras utilizadas en la práctica. Pensamos que este hueco se puede llenar con la visualización del AS y su proceso de construcción. Además, dicha visualización puede ayudar a la enseñanza/comprensión de la traducción dirigida por la sintaxis.

Hemos analizado muchas herramientas que permiten a los usuarios generar sus propios parsers. Sin embargo, estas requieren de un experto para comprender correctamente las estructuras generadas o están muy ligadas al entorno de ejecución. De hecho, cada herramienta posee su propia sintaxis, mensajes de error o distintas estructuras internas.

VAST ha sido desarrollado para solucionar estos problemas visualizando el AS y su proceso de construcción casi de forma independiente del generador de parsers utilizado. Por esta razón, un profesor puede elegir un generador basándose en sus propios criterios -algoritmo de parseo, sintaxis- y utilizar VAST para visualizar los AS. Hay que tener en cuenta que VAST se desarrolló en dos partes por separado: la API de generación -VASTAPI- y la interfaz de visualización -VASTVIEW-.

VASTAPI se diseñó para construir el AS y generar un fichero XML con su representación. VASTVIEW se encarga de interpretar el fichero XML y visualizar el AS y su proceso de construcción. En consecuencia tenemos dos niveles de independencia: una entre el generador de parsers y VASTAPI, y la otra entre VASTAPI y VASTVIEW. Hasta el momento hemos desarrollado VASTAPI con Java, por esta razón no logramos completamente la independencia con el generador de parsers. Sin embargo, simplemente exportando nuestra API a otros lenguajes o creando un nivel más de independencia seremos capaces de utilizar VAST en otros entornos de desarrollo.

Hemos evaluado la usabilidad de VAST en

un escenario real. Los resultados para la interfaz de visualización, VASTVIEW, han sido positivos, además los alumnos piensan que: es fácil de usar, les ayuda al aprendizaje y tiene una buena calidad. También hemos observado que a los alumnos les ha gustado las capacidades de visualización y animación de VASTVIEW.

ANTLRWorks obtuvo unos resultados similares. Sin embargo, observamos que algunos aspectos como la elección de la regla de comienzo, o el hecho tener dos herramientas diferentes de visualización (intérprete y depurador) podían ser confusos para los estudiantes. Esto se debe a la orientación profesional de la herramienta.

En cuanto a VASTAPI, la opinión de los alumnos es ambigua. Por un lado, piensan que la API es bastante fácil de usar. Por otro lado, la puntuación en la facilidad de uso es peor que la de VASTVIEW y ANTLRWorks.

8. Trabajos futuros

Nuestras líneas de trabajo futuro se basan en los resultados de la evaluación. Hemos obtenido pistas de cómo diseñar una herramienta para la visualización del AS. Por un lado, la interfaz de visualización es adecuada para los estudiantes evitando características profesionales. Sin embargo, debería incorporar otras funcionalidades como la especificación de la gramática, explicaciones textuales de las diferentes acciones realizadas y navegación avanzada a lo largo del proceso de parseo, permitiendo a los estudiantes seleccionar partes de la cadena de entrada y mostrar los correspondientes estados.

Por otro lado, hemos detectado que la generación de las visualizaciones se realiza en muchos pasos: edición de la gramática, adaptación de la gramática, generación del parser, compilación del parser, edición de la cadena de entrada, ejecución del parser y visualización. Aunque no debería ser un problema para los profesores, la integración de estos pasos mejorará la interacción de los estudiantes con la herramienta. Debido a esto, hemos planteado una integración global basada en dos

integraciones funcionales: edición-generación-compilación y ejecución-visualización.

La primera integración deberá adaptar automáticamente la especificación de la gramática, generar el código fuente del parser y compilarlo. La segunda deberá permitir la edición de la cadena de entrada, ejecutar el parser y visualizar el AS usando la misma interfaz. De esta forma, la visualización del parser se adaptará al proceso normal de su desarrollo: generación y ejecución.

La adaptación de la gramática se realizará con desarrollos específicos para cada herramienta generadora. Desde el punto de vista de los estudiantes, hacer la adaptación de la gramática transparente, es más importante que perder independencia con el generador de parsers. Desde el punto de vista el profesor, se mantiene la independencia con la herramienta generadora porque se podrá modificar la gramática manualmente.

Por último, ampliaremos VAST para visualizar la traducción dirigida por la sintaxis.

9. Agradecimientos

Este trabajo ha sido financiado por el proyecto TIN2008-04103/TSI del Ministerio de Ciencia e Innovación de España.

Referencias

- [1] F. Almeida-Martínez and J. Urquiza-Fuentes. Teaching ll(1) parsers with vast - a usability evaluation. Technical report, http://www.dlsi1.etsii.urjc.es/doc/DLSI1-URJC_2009-01.pdf, 2009.
- [2] F. Almeida-Martínez, J. Urquiza-Fuentes, and J. Velázquez-Iturbide. Visualización de Árboles sintácticos en la enseñanza de procesadores de lenguajes. 2008.
- [3] K. Andrews, R. R. Henry, and W. K. Yamamoto. Design and implementation of the uw illustrated compiler. *SIGPLAN Not.*, 23(7):105–114, 1988.
- [4] J. Bovet. Antlrworks: The antlr gui development environment, 2008.
- [5] M. K. Cohen L., Manion L. Research methods in education. 2001.
- [6] S. L. Felder R. *Learning and teaching styles in engineering education. Engr. Education*, 78(7):674–681, 1988.
- [7] D. S. Hundhausen C. and S. J. A meta-study of algorithm visualization effectiveness. *J. of Vis. Lang. and Comp.*, 13(3):259–290, 2002.
- [8] J. Hunter. Jdom. <http://www.jdom.org/>, 2008.
- [9] A. Kaplan and D. Shoup. CUPV a visualization tool for generated parsers. *SIGCSE Bull.*, 32(1):11–15, 2000.
- [10] S. Khuri and Y. Sugono. Animating parsing algorithms. *SIGCSE Bull.*, 30(1):232–236, 1998.
- [11] J. R. Levine, T. Mason, and D. Brown. *Lex & Yacc*. O'Reilly, second edition, 1992.
- [12] M. E. Lovato and M. F. Kleyn. Parser visualizations for developing grammars with yacc. *SIGCSE Bull.*, 27(1):345–349, 1995.
- [13] M. Mernik and V. Zumer. An educational tool for teaching compiler construction. *IEEE Transactions on Education*, 46(1):61–68, Feb. 2003.
- [14] T. Naps, G. Röfling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and J. Velázquez-Iturbide. Iticse 2002 working group report: Exploring the role of visualization and engagement in computer science education. *SIGCSE Bull.*, 35(2):131–152, june 2002.
- [15] D. Resler. Visiclang—a visible compiler for clang. *SIGPLAN Not.*, 25(8):120–123, 1990.
- [16] R. D. Resler and D. M. Deaver. VCOCO: a visualisation tool for teaching compilers. *SIGCSE Bull.*, 30(3):199–202, 1998.
- [17] S. Rodger. Learning automata and formal languages interactively with JFLAP. *SIGCSE Bull.*, 38(3):360–360, 2006.
- [18] S. R. Vegdahl. Using visualization tools to teach compiler design. *J. Comput. Small Coll.*, 16(2):72–83, 2001.