ENTORNO DE SOPORTE AL APRENDIZAJE ACTIVO EN PROCESAMIENTO DE LENGUAJE

José Luis Sierra - Alfredo Fernández-Valmayor
Facultad de Informática - UCM

Agradecimientos: Este trabajo está soportado por el proyecto UCM de Innovación y Mejora de la Calidad Docente n.º 35 (año 2006). El trabajo también está soportado parcialmente por los proyectos del MCyT / MEC TIN2004-08367-C02-02 y TIN2005-08788-C04-01.

Palabras clave: Enseñanza del Procesamiento del Lenguaje, Aprendizaje Activo, Prototipado de Procesadores de Lenguaje, Gramáticas de Atributos.

El procesamiento de lenguaje es una competencia central en disciplinas relacionadas tanto con Informática como con Lingüística Computacional. Su carácter constructivo aconseja la adopción de una estrategia activa para su enseñanza, según la cual el alumno aprende definiendo lenguajes y construyendo sus procesadores: los programas que procesan las frases de dichos lenguajes. Bajo esta hipótesis, y en el contexto del programa de Proyectos de Innovación y Mejora de la Calidad Docente 2005-2006 de la UCM, estamos desarrollando un entorno informático orientado a facilitar dicho aprendizaje activo. En este artículo motivamos y describimos dicho entorno.

---

1. INTRODUCCIÓN

Las nuevas tecnologías web y los lenguajes basados en XML [3] han incrementado la importancia del estudio de los procesadores de lenguaje tanto en Ingeniería Informática *(Procesadores de Lenguaje)* como en la Titulación de Lingüística *(Lingüística Informática y Lingüística Computacional)*. El alumno debe poder formular sistemáticamente los diferentes aspectos de un lenguaje (artificial, como en Informática, o natural, como en Lingüística), así como ser capaz de programar sistemáticamente procesadores para dicho lenguaje.

Durante nuestra experiencia docente hemos comprobado la necesidad de promover un aprendizaje activo de la materia, durante el cual el alumno define sus propios lenguajes y programa sus propios procesadores. Para propugnar el desarrollo sistemático de esta tarea, uno de nuestros principales objetivos pedagógicos es concienciar al alumno de la importancia de separar claramente la especificación del lenguaje (vgr., su sintaxis, sus restricciones contextuales y su significado operativo) de la subsiguiente construcción de sus procesadores como programas informáticos. Efectivamente, sin una especificación adecuada que documente el lenguaje procesado, los programas que realizan los procesadores son difíciles de mantener y de modificar. No obstante, este hecho es difícilmente asimilable por parte del alumno, cuya tendencia natural es a concentrarse en la actividad de programación, descuidando los aspectos fundamentales de la especificación formal. La no adecuada asimilación por parte del alumno de la importancia de dicha actividad en el procesamiento de lenguaje puede desvirtuar las habilidades finales adquiridas: aunque el alumno sea capaz de desarrollar procesadores para lenguajes simples, sin dominar las técnicas de especificación formal difícilmente será capaz de gestionar proyectos reales que involucren lenguajes y procesadores más complejos.

El principal motivo de la tendencia del alumno a la programación es el mayor retorno recibido como consecuencia de dicha actividad: un artefacto que puede ejecutarse en un ordenador y que efectivamente procesa lenguaje. Por el contrario, en muchas ocasiones el alumno ve la especificación como una tediosa obligación adicional. El resultado es la producción de especificaciones de baja calidad, que en muchas ocasiones se realizan después de la programación del procesador, a fin de satisfacer los requisitos de evaluación de las actividades propuestas.

Para resolver los inconvenientes descritos hemos planteado un proyecto de Innovación y Mejora de la Calidad Docente (PIMCD 2006/35 *Entorno de Soporte al Aprendizaje Activo en Procesamiento de Lenguaje)* para llevar a cabo el desarrollo de PAG *(Prototyping with Attribute Grammars),* un entorno informático que permita a los alumnos de las materias relacionadas con el lenguaje y su procesamiento el ejecutar directamente especificaciones de lenguaje. Dicho entorno estará inicialmente orientado a facilitar el aprendizaje activo en la materia *Procesadores de Lenguaje,* aunque también esperamos que los resultados puedan aplicarse a materias afines en la titulación de Lingüística *(Lingüística Informática, Lingüística Computacional)*. En este artículo contextualizamos y presentamos el citado entorno.

La estructura del artículo es como sigue. En el apartado 2 motivamos el desarrollo de PAG en base al contexto pedagógico en el que ha surgido el proyecto. En el apartado 3 proporcionamos una descripción de alto nivel del entorno. En el apartado 4 describimos el estado actual del proyecto, las principales conclusiones obtenidas hasta el momento, así como anticipamos algunas posibles líneas de continuación.

2. CONTEXTO PEDAGÓGICO

En este apartado presentamos el contexto pedagógico del trabajo descrito en este artículo. Este contexto se centra, tal y como ya hemos indicado, en la asignatura de *Procesadores de Lenguaje,* en Ingeniería Informática, aunque también esperamos extender la mejora a las asignaturas de *Lingüística Informática* y *Lingüística Computacional* de la Titulación en Lingüística. En la sección 2.1 describimos brevemente la asignatura de *Procesadores de Lenguaje,* así como el papel de la actividad de *prototipado* de procesadores en dicha asignatura. En la sección 2.2 explicamos la realización de dicha actividad en el método docente empleado actualmente, identificando sus ventajas y sus inconvenientes. En base a dicha identificación, en la sección 2.3 establecemos los requisitos para el entorno presentado en este trabajo, y en la sección 2.4 describimos el esfuerzo inicial de búsqueda de soluciones.

2.1. La asignatura de Procesadores de Lenguaje

Procesadores de Lenguaje es una asignatura troncal de nueve créditos del segundo ciclo de Ingeniería en Informática. El principal objetivo pedagógico de esta asignatura es dotar al alumno de las competencias básicas para la descripción sistemática de lenguajes informáticos y para el desarrollo sistemático de sus procesadores (vgr., sus compiladores o sus intérpretes) siguiendo técnicas estándar en el campo [1].

El método pedagógico adoptado en *Procesadores de Lenguaje* propugna un aprendizaje *basado en problemas*. Los alumnos trabajan en grupos para resolver de manera incremental los distintos problemas que surgen en la especificación y construcción de un traductor-intérprete para un lenguaje informático de complejidad no trivial (similar a Pascal [28]). Dicho trabajo se orquesta de acuerdo con el modelo de proceso incremental que normalmente se sigue durante la construcción *real* de un procesador de lenguaje. Para ello comenzamos proponiendo la construcción de un procesador para un lenguaje mínimo muy sencillo (vgr., un lenguaje para escribir expresiones aritméticas), que posteriormente complicamos mediante la introducción sucesiva de distintas características (vgr., variables para almacenar

resultados intermedios, decisiones y repeticiones, definición de elementos de información complejos, procedimientos y funciones definidas por el usuario, etc.).

Como ya hemos indicado anteriormente, para facilitar el mantenimiento y la evolución del lenguaje y de su procesador promovemos una clara distinción entre los aspectos de especificación y de programación. Los alumnos primeramente describen los distintos aspectos del lenguaje y de su procesador (morfología, sintaxis, restricciones contextuales, su traducción a código objeto, la arquitectura de la máquina encargada de ejecutar dicho código y el efecto de cada instrucción sobre dicha máquina) utilizando formalismos de alto nivel independientes de lenguajes concretos de programación. Una vez disponible dicha especificación, proceden a construir el procesador utilizando un método manual sistemático, o bien una herramienta de propósito específico [12]. En este proceso el principal recurso de especificación utilizado es el formalismo de las *gramáticas de atributos,* formalismo propuesto por Donald E. Knuth a finales de los sesenta y ampliamente desarrollado y utilizado desde entonces en el campo [13,19].

Durante la aplicación del método pedagógico descrito se ha observado la ya citada resistencia por parte de los alumnos para asimilar la conveniencia de separar especificación de programación. Para vencer dicha resistencia hemos ensayado las dos siguientes estrategias:

- Solicitar a los alumnos varias realizaciones alternativas de las distintas versiones del procesador utilizando las distintas técnicas explicadas en la asignatura. Dichas técnicas varían desde la programación puramente *manual* en un lenguaje de propósito general (vgr., C++ [27] o Java [2]), hasta el uso de herramientas y lenguajes de propósito específico (vgr., generadores de analizadores léxicos [16], generadores de analizadores sintácticos [9] y generadores de traductores completos [15]) adaptadas a cada una de las técnicas.
- Introducir explícitamente una actividad de *prototipado* rápido.

Mientras que, desde un punto de vista puramente teórico, la exigencia de distintas realizaciones alternativas del procesador debería ayudar al alumno a apreciar cómo el esfuerzo invertido durante la especificación puede posteriormente amortizarse durante la construcción del procesador (efectivamente, la obtención de los programas a partir de especificaciones bien realizadas es un trabajo rutinario, mecánico y en gran medida automatizable), la realidad es que muchas veces se consigue el efecto contrario. Efectivamente, no es extraño ver cómo muchos alumnos (incluso alumnos sobresalientes) se ven desbordados por el trabajo a realizar, y se concentran en la programación, abandonando las actividades de especificación. El resultado es la producción de programas de baja calidad, muchas veces incompletos o incorrectos, así como de especificaciones de peor calidad aún.

La introducción de una actividad de prototipado rápido ha resultado, sin embargo, bastante más fecunda. En esta actividad los alumnos son capaces de obtener una versión previa ejecutable del procesador a partir de la especificación con muy poco esfuerzo, lo que ejerce en los mismos un efecto altamente motivante. Efectivamente, esta actividad ofrece a los alumnos la sensación de que realmente están haciendo algo *útil* mientras especifican, ya que pueden *ejecutar* y probar el procesador que están especificando. Asimismo la actividad les ayuda a detectar de manera muy temprana omisiones y/o errores en la especificación, así como a resolver dichos defectos. El resultado son mejores especificaciones, y, por tanto, programas más acabados y robustos. Desde un punto de vista pedagógico, la actividad ayuda a los alumnos a comprender mejor los conceptos básicos de la asignatura. De hecho, en algunos casos hemos observado que esta estrategia ha conducido de manera natural al ensayo por parte de los alumnos más motivados, y por propia iniciativa, de distintos desarrollos alternativos utilizando las distintas técnicas enseñadas. La siguiente sección examina con más detalle la implementación de esta actividad, así como sus ventajas y sus limitaciones.

2.2. La actividad de prototipado rápido

Durante la actividad de prototipado empleamos el lenguaje de programación Prolog [25] junto con el mecanismo de *Gramáticas de Cláusulas Definidas* (DCGs-*Definite Clause Grammars* [20]) soportado por el mismo. Efectivamente, la notación de las DCGs es prácticamente la misma que la notación de las *gramáticas incontextuales* que se emplean para la descripción de la sintaxis de un lenguaje informático [1]. Asimismo las DCGs soportan *acciones semánticas* que permiten describir cómo llevar a cabo la comprobación de las restricciones contextuales del lenguaje y cómo traducir dicho lenguaje a código ejecutable. Por su parte, los *atributos semánticos* utilizados en el formalismo de las gramáticas de atributos tienen una correspondencia directa con las *variables lógicas* que pueden añadirse a los símbolos de una DCG. Todo ello permite representar directamente los *esquemas de traducción orientados a la traducción descendente* empleados por los alumnos para describir un tipo particular de procesadores: los basados en el análisis descendente de las frases del lenguaje (figura 1).

Figura 1. (a) Fragmento de esquema de traducción orientado a la traducción descendente; (b) fragmento ejecutable correspondiente de una DCG que prototipa el correspondiente traductor

En nuestra experiencia utilizando DCGs Prolog como base de la actividad de prototipado hemos constatado la utilidad de estos artefactos para ayudar a los alumnos a entender los principales conceptos que subyacen a la traducción de lenguaje guiada por la sintaxis. Efectivamente, dado que las DCGs son artefactos más concretos que las gramáticas de atributos, en muchas ocasiones su uso ayuda al alumno a entender algunos de los conceptos más abstractos que subyacen en las especificaciones basadas en tales gramáticas. También nos hemos sorprendido gratamente al comprobar cómo las DCGs han ayudado a muchos alumnos a entender mejor los principales mecanismos subyacentes a la *programación lógica* (el paradigma de programación en el que se basa Prolog, paradigma que nuestros alumnos estudian en la asignatura *Programación Declarativa,* asignatura obligatoria de primer ciclo en *Ingeniería Informática).*

No obstante, las DCGs están lejos de constituir el mecanismo ideal para llevar a cabo el prototipado rápido de los procesadores. Efectivamente, las realizaciones Prolog usuales de estos mecanismos adolecen de las siguientes limitaciones:

- *Limitaciones sintácticas*. La realización más usual de las DCGs en un sistema Prolog no permite tratar cualquier tipo de gramática incontextual, sino únicamente gramáticas que no exhiben *recursividad a izquierdas*. La recursividad a izquierdas es muy útil, sin embargo, en la especificación de muchas características de los lenguajes informáticos. Por ejemplo, la asociatividad a izquierdas de un operador [la propiedad de que, vgr., 5/6/7 debe interpretarse como (5/6)/7 en lugar de cómo 5/(6/7)] se expresa de forma natural mediante una gramática que es recursiva a izquierdas. Si se trata de ejecutar dicha gramática como una DCG en un sistema Prolog usual, dicho sistema entrará en un bucle sin fin, con el consiguiente desconcierto del alumno.
- *Orden de ejecución explícito*. En el estilo usual de programar DCGs, la ejecución de las acciones semánticas depende del orden de aparición de dichas acciones. Por el contrario, las gramáticas de atributos no inducen ningún orden de ejecución, sino que dicho orden es con-

secuencia de las dependencias naturales entre los atributos semánticos para cada frase. De esta forma, mientras que durante la especificación el alumno no tiene por qué pensar en el orden de ejecución de las acciones, durante el prototipado se ve obligado a determinar dicho orden. De no hacerlo así, los prototipos resultantes pueden fallar (vgr., cuando tratan de sumarse dos números que todavía no se conocen), o incluso no terminar (vgr., cuando el sistema Prolog trata de encontrar un valor no negativo para X que, sumado a 5, dé 4 como resultado, y para ello explora sistemáticamente los infinitos números naturales).

Estas limitaciones conducen precisamente a que el prototipo deba basarse en la especificación del procesador mediante un esquema de traducción orientado a la traducción descendente, tal y como hemos indicado anteriormente, en lugar de directamente en las especificaciones de los distintos aspectos del lenguaje y de su traducción mediante gramáticas de atributos, como sería deseable. Efectivamente, con esta aproximación el alumno debe realizar durante el prototipado un trabajo comparable al llevado a cabo durante las fases iniciales de la construcción de un traductor descendente, aun cuando la técnica de programación final elegida sea otra (vgr., traducción ascendente) que no requiera dicho trabajo (figura 2). Este hecho puede desorientar y desmotivar al alumno, y por tanto desvirtuar la naturaleza última de esta actividad. Éste es el motivo que nos ha llevado a plantear el desarrollo de entorno PAG.

Figura 2. (a) Fragmento de especificación basada en gramáticas de atributos; (b) fragmento de la DCG que realiza el correspondiente prototipo del traductor

2.3. Requisitos para un mejor entorno de prototipado de procesadores

A fin de mejorar la actividad de prototipado en nuestra estrategia pedagógica comenzamos planteando una serie de requisitos mínimos para las posibles soluciones:

- *Simplicidad*. La solución deberá mantener la simplicidad y las ventajas de las DCGs. Deberá ser fácilmente asimilable por nuestros alumnos, y deberá tener un impacto mínimo en la planificación actual del curso.
- *Flexibilidad sintáctica*. La solución deberá funcionar con sintaxis incontextuales arbitrarias, incluso con sintaxis ambiguas. Si bien la ambigüedad (la posibilidad de analizar sintácticamente una frase de más de una forma válida) es un fenómeno normalmente a evitar en el diseño de lenguajes informáticos, el poder manejar dicho fenómeno es imprescindible si pretendemos aplicar la mejora al dominio de la Lingüística, ya que los lenguajes naturales son ambiguos por naturaleza.
- *Flexibilidad semántica*. La solución deberá liberar al alumno de la necesidad de especificar explícitamente el orden de realización de las acciones semánticas llevadas a cabo por el traductor. Dicho orden deberá ser consecuencia de las dependencias naturales entre los atributos semánticos asociados con las estructuras sintácticas, tal y como dicta el formalismo de las gramáticas de atributos. En principio, la solución deberá sólo tratar con dependencias no cíclicas, aunque también contemplamos la posibilidad de relajar esta restricción, a fin de poder aplicar la mejora a contextos más generales en Informática (vgr., aprendizaje activo de la especificación denotacional de lenguajes de programación [26]).

– *Facilidad de comprensión*. Los prototipos generados deberán ser fácilmente comprensibles por los alumnos, a fin de permitirles entender, trazar y visualizar su comportamiento durante el procesamiento de distintas frases de ejemplo.
– *Modularidad*. Los prototipos y las especificaciones deberán ser modulares. La modularidad de los prototipos permitirá a los alumnos ejecutar independiente cada uno de sus componentes, a fin de experimentar por separado con los principales aspectos subyacentes al procesamiento de lenguaje. Por su parte, la modularidad de las especificaciones facilitará la determinación incremental de los lenguajes y sus procesadores, lo que se ajusta a la estrategia pedagógica que hemos introducido anteriormente.
– *Portabilidad y facilidad de instalación*. El entorno deberá ser fácilmente portable, así como fácilmente instalable sobre diferentes plataformas. Para tal fin no descartamos poner el mismo disponible a través de la web (vgr., como un *servicio web* al estilo de [21]) con el fin de facilitar su integración en plataformas de *e-learning,* como la actualmente disponible en la UCM [22].

2.4. Búsqueda inicial de soluciones

Una vez fijados los requisitos a satisfacer por la solución procedimos a evaluar distintas alternativas ya disponibles, entre las que cabe destacar:

– Lenguajes de programación que, como *Elegant* [8] o ALE [4], se derivan del formalismo de las gramáticas de atributos o están fuertemente relacionados con el mismo. No obstante, estas alternativas violan el requisito de simplicidad. Efectivamente, la enseñanza y el aprendizaje de cualquiera de estos lenguajes puede requerir una cantidad de tiempo considerable.
– Entornos de desarrollo de procesadores de lenguaje basados en gramáticas de atributos, como FNC-2 [10], Eli [6] o Cocktail [7]. Alguno de estos entornos (vgr., LISA [18]) han sido incluso reconocidos como especialmente adecuados desde un punto de vista pedagógico [17]. No obstante, estos entornos se conciben como herramientas de desarrollo, en lugar de como herramientas de prototipado. Estos sistemas están dotados de lenguajes de especificación sofisticados, que incluyen características que, si bien son muy valiosas durante el desarrollo profesional de procesadores de lenguaje, violan claramente nuestro requisito de simplicidad. Asimismo normalmente están adaptados para tratar con sintaxis no ambiguas, ya que el tratamiento de este tipo de sintaxis puede realizarse de manera más eficiente que el tratamiento de sintaxis generales. Esto viola el requisito de flexibilidad sintáctica. También normalmente se especializan en el tratamiento eficiente de un determinado tipo de dependencias entre atributos, lo que viola el requisito de flexibilidad semántica y, más importante aún, el requisito de facilidad de comprensión, ya que *eficiencia* y *comprensibilidad* son casi siempre características antagónicas. De igual forma, los procesadores generados no son necesariamente modulares, ya que en muchas ocasiones los diferentes aspectos del procesamiento se acoplan para conseguir eficiencia. Por último, también consideramos que el uso de una alternativa de terceros podía afectar a los requisitos de portabilidad y facilidad de instalación.

Una vez finalizado este proceso de evaluación sin encontrar la solución *adecuada* decidimos emprender el desarrollo de nuestra propia solución: PAG. El siguiente apartado introduce las características de alto nivel de dicho entorno.

3. EL ENTORNO PAG

PAG produce prototipos ejecutables a partir de especificaciones basadas en gramáticas

de atributos. En este apartado introducimos el entorno. La sección 3.1 describe brevemente el lenguaje de especificación soportado. La sección 3.2 explica la estructura de los prototipos producidos. Por último, la sección 3.3 describe la arquitectura interna del entorno.

3.1. El lenguaje de especificación

El diseño del lenguaje de especificación de PAG se rige por los requisitos de simplicidad y facilidad de comprensión. De esta forma, hemos tratado de mantener dicho lenguaje lo más cerca posible de la notación para las gramáticas de atributos que utilizamos en nuestras lecciones. También hemos tratado de mantener las ventajas de Prolog como lenguaje de prototipado. Como resultado, el lenguaje de especificación se concibe como un lenguaje de dominio específico embebido en Prolog. Efectivamente:

- Hemos aprovechado las características de adaptabilidad sintáctica de Prolog para definir operadores que permiten escribir de forma natural las producciones y las ecuaciones semánticas de las gramáticas de atributos (figura 3a).
- Las funciones semánticas utilizadas para computar los valores de los atributos semánticos pueden definirse asimismo directamente en Prolog (figura 3b).

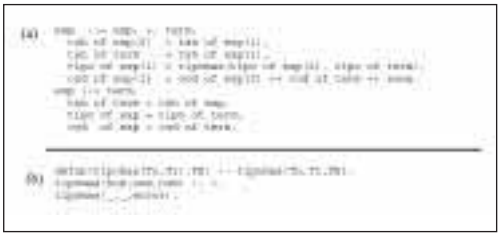

Figura 3. (a) Fragmento de especificación PAG correspondiente con la figura 2 (a); (b) definición de una función semántica

El diseño del lenguaje también obedece al requisito de modularidad. Aquí utilizamos un criterio de modularización sencillo, basado en la separación modular por *aspectos semánticos* propuesta en [11], criterio en el que basamos nuestras presentaciones en la asignatura. De esta forma, sobre una misma sintaxis es posible especificar diferentes gramáticas de atributos, una para cada aspecto semántico involucrado en el procesamiento del lenguaje definido (vgr., construcción de la tabla de símbolos, comprobación de las restricciones contextuales y traducción). Estas gramáticas se unen entonces en base a las reglas sintácticas, para dar lugar a la gramática de atributos final.

Por último, el diseño del lenguaje obedece también a los requisitos de flexibilidad sintáctico-semántica.

Efectivamente, el lenguaje no impone ninguna restricción sobre el tipo de sintaxis utilizada, ni tampoco sobre las posibles dependencias entre atributos (aunque dichas dependencias deberán ser, por lo general, no circulares, como es habitual en traducción).

3.2. La estructura de los prototipos

La estructura de los prototipos generados obedece a los criterios de modularidad y de facilidad de comprensión. Dicha estructura se esquematiza en la figura 4. De acuerdo con dicha figura, cada prototipo consta de:

- Un *analizador sintáctico*. Dicho módulo se encarga de analizar la corrección sintáctica de cada frase procesada, así como de construir el árbol de análisis sin-

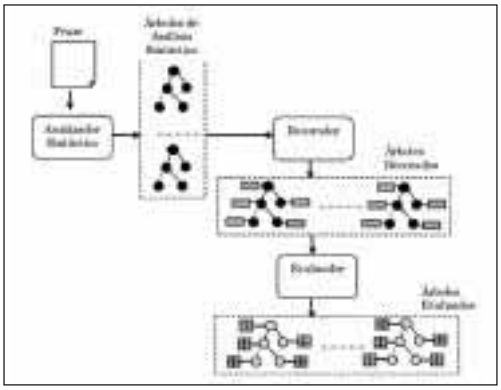

Figura 4. Estructura de los prototipos en PAG

táctico (o los árboles, ya que la sintaxis puede ser ambigua) que refleja(n) su estructura. Este módulo se basa en una implementación simplificada de un algoritmo de análisis sintáctico general: el algoritmo de Earley [5].
– Un *decorador*. Dicho módulo se encarga de asignar qué expresión debe utilizarse para computar el valor de cada atributo semántico asociado con cada nodo del árbol de análisis sintáctico construido.
– Un *evaluador*. Dicho módulo se encarga de evaluar las anteriores expresiones en el orden apropiado para determinar los valores de los atributos semánticos.

La operación de estos prototipos coincide en gran medida con el modelo de procesamiento basado en gramáticas de atributos enseñado a los alumnos. De hecho, un ejercicio habitual que proponemos a los mismos es analizar manualmente varias frases con respecto a una gramática, decorar los correspondientes árboles con las expresiones necesarias para encontrar los valores de los atributos semánticos y evaluar dichas expresiones en el orden apropiado (figura 5).

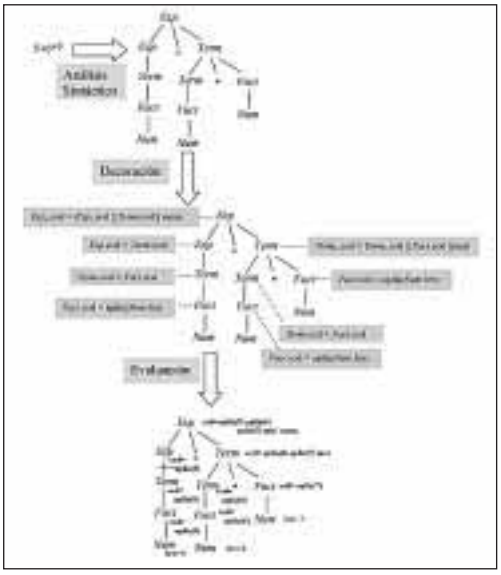

Figura 5. Modelo de procesamiento aplicado a una frase. El procesador genera código para calcular una expresión aritmética

3.3. ARQUITECTURA DEL SISTEMA

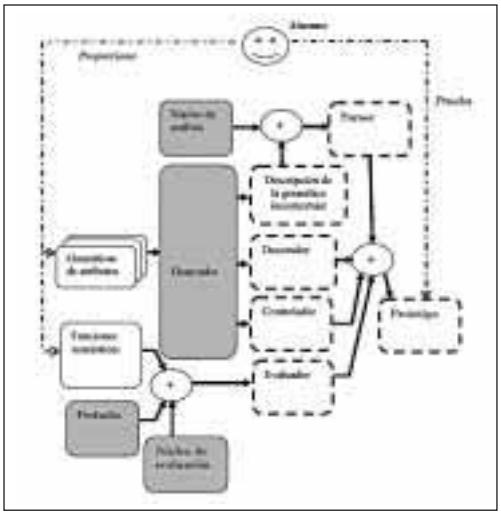

Figura 6. Arquitectura de PAG. Los módulos sombreados representan los componentes predefinidos que conforman el entorno. Los módulos punteados son componentes dependientes de cada especificación y cada prototipo. La combinación de componentes para producir componentes más específicos se representa como (+)

La figura 6 detalla la arquitectura de PAG. De acuerdo con dicha figura, el sistema consta de los siguientes componentes:

– El *núcleo de análisis*. Contiene una implementación simplificada del algoritmo de Earley. Combinada con una representación apropiada de la gramática incontextual subyacente a una especificación, da lugar al analizador sintáctico para el prototipo asociado con dicha especificación.
– El *preludio*. Conjunto predefinido de funciones semánticas usuales que pueden ser utilizadas en cualquier especificación.
– El *núcleo de evaluación*. Contiene la lógica genérica necesaria para evaluar los árboles decorados. Su combinación con las funciones semánticas asociadas a una especificación permite obtener el evaluador del correspondiente prototipo.

– El *generador*. Procesa las especificaciones para producir los prototipos asociados.

La figura 6 también sugiere el funcionamiento de alto nivel del entorno. El alumno proporciona la especificación en términos de:

– Un conjunto de gramáticas de atributos escritas en el lenguaje de especificación de PAG que contienen los distintos aspectos semánticos del procesador.
– La definición de las funciones semánticas referidas desde dichos módulos.

El generador obtiene entonces una gramática de atributos común mediante la *unión* de las gramáticas de atributos de cada uno de los módulos. A partir de dicha gramática produce una descripción de la sintaxis incontextual subyacente que, junto con el núcleo de análisis, da lugar al analizador sintáctico del prototipo. Asimismo produce un decorador específico para el prototipo. Por último, produce un *controlador,* que articula los distintos componentes del prototipo y que permite invocar dicho prototipo como un todo (aunque, como ya hemos indicado anteriormente, el alumno también puede utilizar cada uno de los componentes por separado). La unión de estos componentes con el evaluador obtenido de combinar las funciones semánticas con el núcleo de evaluación permite obtener, por último, el prototipo.

4. ESTADO ACTUAL, CONCLUSIONES Y PERSPECTIVAS DE FUTURO

En este apartado presentamos brevemente el estado actual del proyecto de desarrollo de PAG, así como el trabajo planificado (sección 4.1). También presentamos las principales conclusiones obtenidas con este trabajo (sección 4.2), y adelantamos algunas líneas de trabajo futuro (seccion 4.3).

4.1. Estado actual y trabajo planificado

Actualmente hemos completado el desarrollo de una primera versión de PAG. El resultado se describe en [23]. La versión desarrollada realiza completamente el entorno descrito en el apartado anterior, con la excepción del soporte a especificaciones modulares divididas en múltiples gramáticas de atributos. Siendo dicha versión exploratoria, el entorno presenta, no obstante, distintas carencias que serán resueltas durante el desarrollo de la versión final del mismo:

– La versión final proporcionará soporte completo a especificaciones modulares, siguiendo el modelo simple de modularización por aspectos semánticos al que ya hemos hecho alusión anteriormente.
– Actualmente el generador soporta los mecanismos básicos de generación, pero no ofrece ningún tipo de diagnóstico sobre posibles problemas en la especificación. El generador incluido en la versión final proporcionará al alumno información básica sobre atributos no definidos, que es fundamental a la hora de incrementar el uso de la herramienta.
– El entorno actual no proporciona ningún tipo de soporte para trazar los prototipos, fuera del genérico proporcionado por el sistema Prolog subyacente. En la versión final se incluirán facilidades de traza para cada tipo de componente.
– Por último, en la versión final se mejorará también la eficiencia de algunos aspectos de la implementación.

Asimismo tenemos planificado llevar a cabo evaluaciones preliminares del entorno final con alumnos.

4.2. Conclusiones

El entorno PAG satisface los requisitos planteados para mejorar la actividad de protipado en la construcción de procesadores de lenguaje:

– El lenguaje de especificación es simple, superando incluso en simplicidad al formalismo de las DCGs, y ajustándose estrechamente a la notación de gramáticas de atributos que utilizamos en nuestras clases.

– PAG soporta sintaxis arbitrarias, así como es capaz de tratar dependencias no circulares generales entre atributos semánticos.
– Los prototipos generados en PAG se ajustan estrechamente al modelo de procesamiento de lenguaje basado en gramáticas de atributos que explicamos en nuestras clases. La pérdida de eficiencia que supone la separación explícita de aspectos propugnada por dicho modelo es perfectamente tolerable en este contexto pedagógico, donde el requisito de comprensibilidad impera sobre las posteriores consideraciones de eficiencia.
– Los prototipos son modulares. Efectivamente, el alumno puede utilizar el analizador sintáctico, el decorador o el evaluador aisladamente, o bien puede utilizar todos los componentes en conjunto. Asimismo las especificaciones igualmente son modulares. El mecanismo de modularidad por integración de aspectos semánticos es también consistente con nuestra actual estrategia pedagógica.
– La portabilidad de PAG a diferentes plataformas no debería ser demasiado problemática, ya que PAG es compatible con los sistemas Prolog más habituales para las plataformas más comúnmente utilizadas por los alumnos. Asimismo el sistema es fácilmente instalable, ya que únicamente implica cargar el mismo en el sistema Prolog correspondiente. En cualquier caso, el sistema podrá ofrecerse a través de una interfaz web a fin de mejorar estos aspectos, así como la integración del mismo en otro tipo de entornos y plataformas educativas.

De esta forma, el entorno satisface los requisitos establecidos para la mejora.

4.3. Trabajo futuro

Actualmente estamos trabajando en el desarrollo de la versión final de PAG y en el resto de tareas planificadas. La continuación inmediata del proyecto será implantar la mejora, implantación que llevaremos a cabo en el próximo curso 2006-2007. Como resultado de esta implantación tendremos ocasión de realizar una evaluación exhaustiva del entorno, tanto desde un punto de vista técnico como pedagógico, lo que nos permitirá plantear el desarrollo de una nueva versión mejorada del mismo. Algunas posibles mejoras que actualmente anticipamos son:

– Inclusión de diagnósticos alternativos en el generador (vgr., relativos a la limpieza de la gramática, a la presencia de dependencias entre atributos por la derecha que dificulten la construcción del procesador en *una pasada,* etc.).
– Extensión del entorno para soportar dependencias circulares entre atributos.
– Herramientas de depuración y de traza visual.
– Herramientas de generación automática de documentación, e incluso evolución de PAG a un sistema de *programación literaria* [14] específico para el dominio de procesadores de lenguaje, siguiendo las directrices de nuestros anteriores trabajos orientados al desarrollo documental de aplicaciones ricas en contenidos [24].
– Encapsulamiento del entorno como una aplicación/servicio web.

No obstante, dichas mejoras estarán siempre justificadas por necesidades pedagógicas descubiertas a través del uso real de PAG con alumnos, al igual que el desarrollo de PAG se sustenta en las necesidades de mejora de la actividad de prototipado a las que hemos hecho alusión en este trabajo.

BIBLIOGRAFÍA

1. Aho, A.; Sethi, R., y Ullman, J. D. (1986): *Compilers: Principles, Techniques and Tools*. Adisson-Wesley, Reading, Massachusetts, USA.
2. Arnold, K.; Gosling, J., y Holmes, D. (2005): *The Java Programming Language*. Addison-Wesley, Reading, Massachusetts, USA.

3. BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C. M., y MALER E. (eds.). (2000): Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, www.w3.org.
4. CARPENTER, B., y PENN, G. (2001): *The Attribute Logic Engine User's Guide Version 3.2.1*. University of Toronto.
5. EARLEY, J. (1970): An Efficient Context-free Parsing Algorithm, *Communications of the ACM 13(2),* 94-102.
6. GRAY, R. W.; HEURING, V. P.; LEVI, S. P.; SLOANE, A. M., y WAITE, W. M. Eli. (1992): A Complete, Flexible Compiler Construction System, *Communications of the ACM* 35, 121-131.
7. GROSCH, J., y EMMELMANN, H. (1990): A Tool Box for Compiler Construction, CoCoLab White Paper.
8. JANSEN, P.; AUGUSTEIJN, L., y MUNK, H. (1993): *An Introduction to Elegant. Second Edition,* Philips Research Laboratorios.
9. JOHNSON, S. C. (1978): Yacc: Yet Another Compiler-Compiler. En Kernighan, B. W., y McIlroy, M. D. (eds.) *UNIX Programming Manual Seventh Edition,* Bell Labs.
10. JOURDAN, M., y PARIGOT, D. (1991): Internals and Externals of the FNC-2 Attribute Grammar System, in Ablas, H. Melichar, B. (eds.). *Attribute Grammars, Applications and Systems,* Lecture Notes in Computer Science 545, Springer.
11. KASTENS, U., y WAITE, W. M. (1994): Modularity and reusability in attribute grammars, *Acta Informatica* 31(7), pp. 601-627.
12. KLINT, P.; LÄMMEL, R., y VERHOEF, C. (2005): Toward an Engineering Discipline for Grammarware, *ACM Transactions on Software Engineering and Methodology* 14(3), 331-380.
13. KNUTH, D. E. (1968): Semantics of Context-free Languages, *Mathematical Systems Theory,* 2(2), 127-145. Véase también corrección publicada en 1971, número 5(1), 95-96.
14. — (1984) Literate Programming. *The Computer Journal,* 27(2), 97-111.
15. KODAGAMALLUR, V. (2004): Incorporating Language Processing into Java Applications: A JavaCC Tutorial. *IEEE Software* July/August.
16. LESK, M. E., y SCHMIDT, E. (1978): Lex: A Lexical Analyzer Generator. En Kernighan, B. W., y McIlroy, M. D. (eds.) *UNIX Programming Manual Seventh Edition,* Bell Labs.
17. MERNIK, M., y ŽUMER, V. (2003): An Educational Tool for Teaching Compiler Construction, IEEE Transactions on Education 46(1), 61-68.
18. MERNIK, M.; LENIC, M.; AVDICAUSEVIC E., y ŽUMER, V. (2000): Compiler/Interpreter Generator System LISA, Proc. of the 33rd Hawaii International Conference on Systems Science, Maui, Hawaii, January 4-7.
19. PAAKKI, J. (1995): Attribute Grammar Paradigms – A High-Level Methodology in Language Implementation. *ACM Computing Surveys,* 27(2), 196-255.
20. PEREIRA, F. C. N., y WARREN, D. H. D. (1980): Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence* 13(1-2), 231-278.
21. REBERNAK, D.; CREPINSEK, M., y MERNIK, M. (2006): Web Service for Designing and Implementing Formal Languages. 28th International Conference on Information Technology Interfaces, Dubrovnik, Croatia, junio, 19-22
22. REHBERG, S.; FERGUSON, D.; MCQUILLAN, J.; ENEMAN, S., y STANTON, L. (2004): *The Ultimate WebCT Handbook, a Practical and Pedagogical Guide to WebCT 4.x,* Ultimate Handbooks.
23. SIERRA, J. L., y FERNÁNDEZ-VALMAYOR, A. (2006): A Prolog Framework for the Rapid Prototyping of Language Processors with Attribute Grammars, en Proc. 6th Workshop on Language Description Tools and Applications LDTA 2006 – ETAPS 2006 (también va a ser publicada en *Electronic Notes in Theoretical Computer Science),* Viena, Austria, April 3.
24. SIERRA, J. L.; FERNÁNDEZ-VALMAYOR, A., y FERNÁNDEZ-MANJÓN, B.: A Document-Oriented Paradigm for the Construction of Content-Intensive Applications. *The Computer Journal*. En prensa. Disponible *on-line* en acceso adelantado desde el 13 de abril de 2006 (doi: 10.1093/comjnl/bxl008).
25. STERLING, L., y SHAPIRO, E. (1994): *The Art of Prolog,* The MIT Press, Cambridge, Massachusetts, USA.
26. STOY, J. E. (1977): *Denotational Semantics*: *The Scott-Strachey Approach to Programming Language Theory,* The MIT Press, Cambridge, MA, USA.
27. STROUSTRUP, B. (1987): *The C++ Programming Language 3rd Edition*. Addison-Wesley, Reading, Massachusetts, USA.
28. WIRTH, N. (1971): The Programming Language Pascal, *Acta Informatica* 1, 35-63.