Herramientas de generación de procesadores de lenguajes para código móvil seguro.

J. Aguirre, M.Arroyo, N. Florio, J. Felippa, G. Gomez, F.Bavera, P. Caymes Scutari, D. Nordio[1].

RESUMEN

En este trabajo se presentan las líneas del grupo de investigación del Departamento de Computación de la UNRC constituido por los autores. Los aspectos fundamentales son la creación de modelos y herramientas de última generación para la generación de procesadores de lenguajes incluyendo la generación de Código Móvil Seguro. Se presenta el trabajo ya realizado que ha permitido construir un prototipo de ambiente de generación de procesadores de lenguajes basado en un modelo que soporta una integración más fuerte que las usadas, de las gramáticas de atributos y los esquemas de traducción, los trabajos realizados sobre evaluación concurrente de atributos y su inserción en el prototipo. Se analiza la nueva técnica para obtener Código Móvil Seguro (CMS) basado en la generación conjunta del código y de una prueba de seguridad – Proof Carrying Code PCC – y el proyecto de integrar en el ambiente un módulo que genere PCC basado en tipos dependientes. Es de destacar que el gran desarrollo de las redes de computadoras y de las actividades que se soportan sobre ellas y la aparición de lenguajes y herramientas, como Java, que soportan la migración de código entre distintas plataformas, ha puesto al CMS en un lugar relevante de tecnología informática.

INTRODUCCIÓN

La especificación de lenguajes formales y el desarrollo de sus correspondientes procesadores, compiladores, intérpretes o interfaces con el usuario, ha ocupado un lugar preponderante en las Ciencias de la Computación, con muy importantes consecuencias sobre la industria de software. Pese a los grandes avances realizados en el área, aún quedan aspectos importantes por resolver en la problemática de la generación de procesadores de lenguajes y en particular en la de los compiladores de compiladores [Aho86][App98][Fra95][Wai92].

A su vez el gran desarrollo de las redes de computadoras y de las actividades que se soportan sobre ellas, ha impulsado el uso de código móvil. Por otra parte la aparición de lenguajes y herramientas, como Java, que soportan la migración entre distintas plataformas ha hecho que se torne fundamental poder contar con métodos y herramientas que garanticen la seguridad del entorno local en el que se ejecuta código migrado del exterior.

El campo de trabajo del grupo aborda aspectos de las problemáticas citadas, pudiéndose agrupar sus actividades en dos líneas: A) Modelos y herramientas que soportan la generación de procesadores de lenguajes y B) Generación de código móvil probado (Proof Carrying Code). Como resultado instrumental se pretende obtener un prototipo de ambiente de generación de procesadores de lenguajes que implemente los resultados obtenidos, del cual existe una primera versión *-japlage-* [Agu01]

[1] Universidad Nacional de Río Cuarto. e-mails: jaguirre@dc.exa.unrc.edu.ar, marroyo@dc.exa.unrc.edu.ar, nflorio@dc.exa.unrc.edu.ar, jfelippa@dc.exa.unrc.edu.ar, ggomez@dc.exa.unrc.edu.ar, pancho@dc.exa.unrc.edu.ar, pcaymesscutari@dc.exa.unrc.edu.ar, nordio@dc.exa.unrc.edu.ar.

Este trabajado ha sido realizado en el marco de proyectos subsidiados por la SECyT de la UNRC y por la Agencia Córdoba Ciencia.

Descripción de las líneas de trabajo

A continuación se da una descripción sucinta de las dos líneas fundamentales:

A) Modelos y herramientas que soportan la generación de procesadores de lenguajes.

Los modelos teóricos que sustentan los lenguajes de especificación usados actualmente por los generadores de procesadores de lenguajes – y en particular de compiladores de compiladores – son las gramáticas de atributos y los esquemas de traducción. La herramienta más usada, yacc – originalmente desarrollada dentro del entorno de desarrollo de Unix y luego migrada a prácticamente todas las plataformas – brinda un mecanismo que resulta de una débil integración de ambos modelos. Con esta herramienta se puede trabajar con un conjunto restringido de esquemas de traducción – aquellos que no alteran el orden de los símbolos no terminales en la traducción–, a los cuales se agregan las facilidades de las gramáticas con atributos a izquierda – L-attributed grammars. Este modelo no permite resolver problemas que requieran gramáticas que escapen a este último tipo, tales como el de la verificación del sistema de tipos de Ada, en estos casos el usuario debe recurrir a la generación de un árbol sintáctico, usando las facilidades brindadas por yacc, y luego construir su propio evaluador que implemente las estrategias de recorrido necesarias. Por otra parte, como yacc está basado en un analizador ascendente –LALR(1)– la herencia debe ser emulada por el usuario, mediante la introducción de símbolos demarcadores y esta técnica conduce en muchos casos a que la gramática modificada deje de pertenecer a la clase LALR(1) quedando fuera del alcance de la herramienta. En los últimos años la aparición de JAVA, con la seguridad de su sistema de tipos fuerte y la posibilidad de realizar implementaciones independientes de la plataforma y de generar código móvil a través de redes, ha hecho que aparezcan nuevas herramientas de generación implementadas sobre él. Sin embargo, todas ellas siguen imponiendo al usuario restricciones sobre el tipo de la gramática de atributos a utilizar.

El grupo ha construido con un modelo teórico - Esquemas de Traducción con Atributos (ATS)- y un modelo de implementación [Agu98], que brinda una combinación más potente de los formalismos antes mencionados y se ha implementado un prototipo de ambiente *Japlage* [Agu01], que implementa los modelos desarrollados.

El sistema de tiempo de ejecución –RTS- de los procesadores generados según el modelo de implementación está compuesto por tres tipos de procesos:

1. El parser, proceso secuencial que a lo largo del análisis sintáctico inicializa los procesos de los otros dos tipos.
2. Los procesos de traducción, iniciados por un *scheduler* que determina el orden en que deben ser ejecutados. Estos procesos pueden correr en paralelo con el analizador sintáctico y los procesos del tercer tipo.
3. Los procesos de evaluación de atributos que son ejecutados concurrentemente con todos los procesos del sistema, incluso con los del mismo tipo, con la única restricción de la dependencia entre atributos.

Los procesos de tipo 3 se comunican entre sí y con los procesos de tipo 2 mediante objetos compartidos en los que se alojan los valores de los atributos. Cuando un proceso hace referencia a un atributo aún no evaluado, se bloquea hasta que el valor esté disponible, la figura 1 muestra la vinculación entre estos procesos.

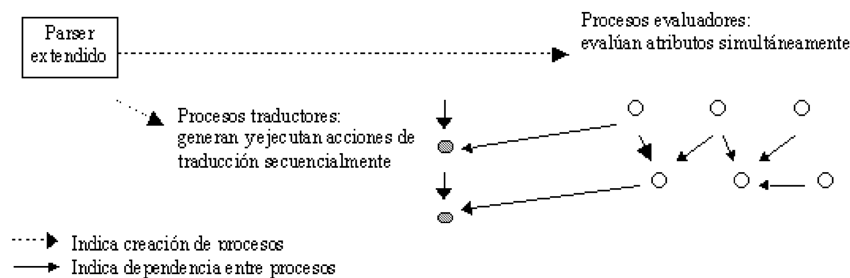

Figura 1: Implementación de un ATS: componentes y su interrelación

Este tipo de evaluación llamado "bajo demanda" [Sar93] genera un importante costo adicional debido a la nutrida comunicación producida. El grupo ha comenzado a trabajar en la optimización de la evaluación concurrente de Gramáticas de Atributos, obteniendo un algoritmo de partición del grafo de dependencias que permite generar procesos con comunicación mínima para gramáticas de atributos de tipo NC(1) [Arr00]. Así mismo se ha extendido este algoritmo a las gramáticas NC(1) condicionales [Boy96] [Arr01].

Asimismo se pretende realizar un estudio sobre implementaciones con paralelismo real. Para lo cual se ha configurado un cluster de PC's en principio basado en la plataforma BeoWulf para emular una arquitectura paralela de alto rendimiento con bajo costo.

B) Generación de código móvil probado (Proof Carrying Code, PCC).

El avance hacia la sociedad informatizada y el consecuente uso masivo de las redes públicas de computadoras, impone la utilización de código que se mueve de un equipo a otro. Resulta por lo tanto de primordial importancia que se pueda garantizar que la ejecución del código importado no dañe la integridad y/o privacidad del ambiente local en el que se ejecuta, mediante el uso de código seguro. Un importante ejemplo de software inseguro es el código que se importa mediante browsers de Internet y un claro ejemplo de software invasor lo constituyen los virus que usan ese código como medio de propagación.

La técnica de Proof Carrying Code (PCC), iniciada por Necula y Lee [Nec96][Fei97] permite a un productor de software proveer un programa conjuntamente con la prueba formal de su seguridad. El destinatario puede especificar su política de seguridad mediante un sistema de axiomas para razonar sobre ella. La demostración construida por el productor debe basarse en dichos axiomas y será ejecutada en el ambiente local antes de que el código gane el control. El código importado sólo será ejecutado si la prueba resulta exitosa.

PCC se basa en los mismos métodos formales que la verificación de programas [Pon99], pero con la ventaja de que la demostración de condiciones de seguridad es mucho menos compleja que la demostración de la corrección de programas. La demostración del productor no asegura la corrección del programa pero sí que su ejecución no violará, bajo ninguna circunstancia, ninguna de las condiciones de seguridad impuestas. Al usarse PCC no es necesario verificar la autenticidad de la identidad del productor, ya que la validez de la prueba constituye una evidencia incontrovertible de su seguridad.

Los lenguajes de código intermedio constituyen un elemento capital en el diseño de compiladores. Ellos permiten modularizar el proceso descomponiéndolo en dos fases: el pasaje del len-

guaje fuente al lenguaje intermedio y la generación del código objeto a partir se este último. Los lenguajes intermedios también son centrales en el PCC, si la prueba se realiza sobre ellos, el desarrollo puede usarse para una diversidad tanto de lenguajes de programación, desde los cuales se genera dicho código intermedio, como de plataformas para las cuales se genera código a partir de él [App00].

En el proyecto se propone usar un sistema de tipos que corresponde a una avanzada tecnología, a efectos de disponer de la mayor información estática en la fase de la compilación correspondiente a la generación de código intermedio. Para ello se introducirá un nuevo lenguaje de código intermedio de última generación basado en los tipos dependientes (Dependent types). El uso de Tipos Dependientes en lugar de un sistema de tipos tradicional para el lenguaje intermedio, presenta la ventaja de que estos sistemas son pequeños pero altamente poderosos. Usando Tipos Dependientes se puede implementar un único formalismo para la generación de código intermedio y para la construcción de la prueba.

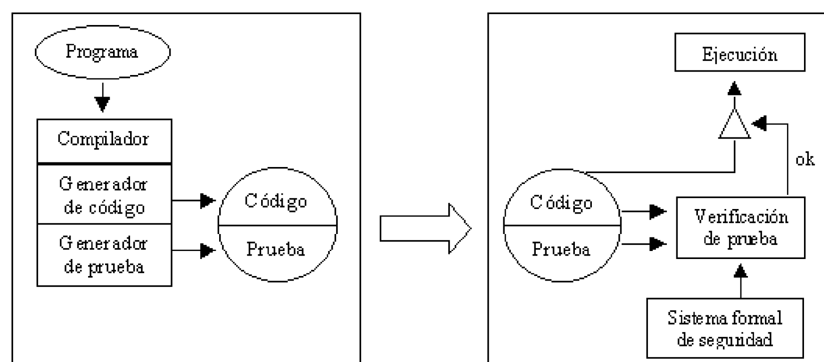

Figura 2: Visión global de Proof-Carrying Code

Productos que se espera obtener.

Como resultados esperados se espera producir tres herramientas computacionales:

- Una herramienta de última generación para la producción de procesadores de lenguajes formales.
- Un evaluador de atributos paralelo eficiente para la familia más amplia de gramáticas de atributos (Well Formed Attributed Grammars). El evaluador será integrado a la herramienta mencionada anteriormente.
- Una herramienta para la generación de código PCC integrada al ambiente de desarrollo de procesadores de lenguajes.

Referencias

[Agu98] J. Aguirre, V. Grinspan, Marcelo Arroyo, "Diseño e Implementación de un Entorno de Ejecución, Concurrente y Orientado a Objetos, generado por un Compilador de Compiladores", Anales ASOO 98 JAIIO, pp. 187-195, Buenos Aires1998.

[Agui 01] JACC un entorno de generación de procesadores de Lenguajes, J. Aguirre, M. Arroyo, J. Felippa, G. Gomez. Anales del CACIC 2001, pp. 145-355, Calafate 2001, ISBN 987 96 288-6-1

[Aho86] A.V. Aho, R. Sethi, J.D. Ullman, “Compilers: Principles, Techniques, and Tools”, Addison Wesley, 1986.

[App00] A. W. Appel, A. P. Felty, “A Semantic Model of Types and Machine Instructions for Proof-Carrying Code”, 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.(POPL '00), pp. 243-253, January 2000.

[App98] A. W. Appel, “Modern Compiler Implementation in Java”. Cambridge University Press, ISBN: 0-521-58388-8, 1998.

[Arr00] M. Arroyo, N. Florio, J. Aguirre, “Un generador de evaluadores de gramáticas de atributos NC(1) de máximo paralelismo sin sincronización entre procesos”, UNRC, Anales del CACIC 2000 pp. 523-526, Ushuaia 2000.

[Arr01] M. Arroyo, N. Florio, J. Aguirre, “Gramáticas de atributos NC(1) condicionales”, UNRC, Anales del CACIC 2001 pp. 333-344, Ushuaia 2001, ISBN 987 96 288-6-1

[Boy96] J. T. Boyland, “Conditional Atribute Grammars”, ACM Transactions on Programming Languages and Systems, Vol. 18, No. 1, January 1996, pages 73-108.

[Fei97] J. Feigenbaum, P. Lee. “Trust Management and Proof-Carrying Code in Secure Mobile-Code Applications”, DARPA, Workshop on Foundations for Secure Mobile Code. 1997.

[Fra95] C. Fraser, “Retargetable C Compiler : Design and Implementation”, Benjamin Cummings Publishing Company, 1995

[Nec96] G. Necula, P. Lee. “Safe Kernel Extensions Without Run-Time Checking”. Second Symposium on Operating Systems Design and Implementation. Seatle. 1996.

[Pon99] C. Pons, R.Giandini, G. Baum, M. Felder, “A dynamic logic theory describing the UML”. Position paper for the OOPSLA´99 workshop on Rigorous Modeling and Analysis with the UML, Denver, Colorado, USA, November 1999.

[Sar93] J. Saravia, P. Henriques, “Concurrent Attribute Evaluation”, tecnical report, Departamento de Informática, Univ. de Minho, Portugal, 1993.

[Wai92] Waite, Carter. “An Introduction to Compiler Construction”. HaperCollins College Publishers. ISBN: 0-673-39822-6. 1992.