Construcción automatizada de talleres de programación: PEPIM.

Miguel SANTANA
Laboratoire de Génie Informatique - Institut IMAG
BP 68, 38402 St Martin d'Heres Cédex, Grenoble, FRANCIA

Escuela de Graduados - Pontificia Univ. Católica del Perú
Apartado 1761, Lima, PERU

RESUMEN

El ritmo acelerado de aparición de nuevos computadores estos últimos años ha creado una fuerte demanda en lo que respecta a la realización de herramientas de base para estas máquinas. Debido a esta demanda, las investigaciones sobre la automatización de la construcción de estas herramientas han cobrado un mayor auge, tanto a nivel universitario como de casas de software.

Este artículo describe el diseño y la implementación de PEPIM, un sistema de producción automática de cadenas de desarrollo de software (talleres de programación) para una máquina dada, a partir de su descripción en un lenguaje apropiado. La realización de un taller completo puede ser efectuada en pocas semanas, lo que constituye una mejora considerable en relación a la construcción manual del mismo; por otro lado, las modificaciones del taller (nuevos componentes, nuevas versiones, cambios en las especificaciones, etc.) son realizadas con mucho mayor facilidad y, por supuesto, a un costo menor.

Este proyecto ha sido realizado por el Laboratorio LGI conjuntamente con la empresa Silicone, una casa de software francesa pequeña pero sumamente dinámica. El producto elaborado representa el estado del arte actual en el área y ha despertado muchas expectativas entre los constructores de computadores, entre ellos: Texas Instruments, Motorola, NEC y Thomson.

I. INTRODUCCION.

El ritmo acelerado de aparición de nuevos computadores durante estos últimos años, ha creado una fuerte demanda en lo que respecta a la realización de herramientas de base para estas máquinas. Esta demanda se caracteriza por tres puntos principales:

a) La construcción de estas herramientas debe poder hacerse lo más rápidamente posible, debido a la importancia del factor tiempo en el mercado de computadores. En la mayor parte de los casos, la disponibilidad de las herramientas debe preceder la aparición de la máquina; de esta manera, los primeros usuarios (compañías que comercializan software, clientes importantes, etc.) pueden desarrollar con anticipación sus productos o aplicaciones.

b) La primera versión de las herramientas debe permitir un desarrollo cruzado: la programación se hará sobre una máquina diferente (llamada máquina de desarrollo) pero a destinación del nuevo procesador o computador. El desarrollo del software de base, de las primeras aplicaciones y de la mayor parte de los programas de test del hardware se hacen con la ayuda de esta versión. El término primera versión es en realidad un eufemismo para designar todas las versiones de desarrollo cruzado que serán necesarias para adaptarse a la evolución de la máquina.

c) El conjunto de herramientas, que constituyen lo que llamaremos taller de programación, debe ser lo más completo posible: compiladores de lenguajes de alto nivel, ensamblador, macro-procesador, debugger, editor de referencias (linkage editor), simulador, etc. Es raro hoy en día que los usuarios potenciales de una nueva máquina queden satisfechos con un ensamblador y un loader absoluto.

Para tratar de satisfacer esta demanda, las compañías de software se han visto obligadas a desarrollar nuevos métodos y herramientas más adaptados a la producción de este tipo de software. Sin embargo, los resultados obtenidos no han sido verdaderamente los esperados: costos muy altos, plannings demasiado largos, características técnicas pobres, número restringido de herramientas, etc.

El objetivo del proyecto PEPIM (Producción de Entornos de Programación Independientes de la Máquina) es de resolver este problema, a través de un nuevo enfoque: la automatización de la construcción de este tipo de herramientas.

El artículo propuesto presenta el diseño y la implementación de PEPIM. El primer capítulo constituye una descripción general de este sistema: orígenes, objetivos, contexto de realización, características generales. El capítulo

siguiente discute los aspectos relativos a la producción automática de los diferentes elementos del taller: informaciones necesarias, tratamiento de estas informaciones, construcción propiamente dicha. En el tercer capítulo presentamos la arquitectura general de PEPIM, describiendo de manera sucinta las características de cada uno de sus componentes. Finalmente, el cuarto capítulo analiza los resultados obtenidos con nuestro enfoque y los compara con los de productos similares.

II. DESCRIPCION GENERAL DE PEPIM.

II.1 ORIGEN DEL PROYECTO.

El Laboratorio LGI trabaja desde hace muchos años en el área de automatización de la construcción de compiladores, en el marco del proyecto GEMME [Santana 83, Santana 84, Hochain 86, Santana 86, ...]. Este proyecto nació como una respuesta al retraso del área de generación de código en compilación y ante la multiplicación de los nuevos computadores y lenguajes de programación al final de los años setenta. En ese entonces, numerosos proyectos se volcaron al estudio de este problema, entre los cuales cabe destacar: PQCC [Leverett 80], el método de generación Graham/Glanville [Graham 82, Crawford 82] y las proposiciones de Ganapathi [Ganapathi 82]. Sin embargo, pocos proyectos llegaron a resultados concretos, como fue el caso de GEMME; efectivamente, el primer prototipo de GEMME interesó bastante al Centro de Telecomunicaciones de Francia, quien decidió incorporarlo como parte del ambiente de programación CONCERTO [Concerto 86]. Gracias a esta colaboración, pudimos continuar el desarrollo de GEMME hasta llevarlo a un nivel de producto prácticamente comercial.

El proyecto PEPIM nace como una prolongación de GEMME. Este nuevo proyecto retiene las ideas más interesantes de GEMME, al mismo tiempo que mejora ciertos de sus aspectos y amplía sus objetivos. Se trata en lo esencial de aplicar las ideas desarrolladas para GEMME a un área más vasta: la construcción de todas las herramientas de base de un computador (compiladores, ensamblador, editor de referencias o linkage editor, cargador o loader, debugger, etc).

II.2 CONTEXTO DE TRABAJO.

Desde un inicio, PEPIM ha sido desarrollado en estrecha colaboración entre el Laboratorio LGI y la empresa Silicone. Este tipo de colaboración tiene la ventaja de unir los recursos propios de una empresa privada a los conocimientos (el know-how) de un laboratorio de investigación; sin embargo, también tiene su lado negativo, ligado a los imperativos de planificación y de comercialización que son los de toda empresa.

Por otro lado, cabe destacar el interés de realizar un producto de avanzada, sabiendo que va a ser comercializado y, por lo tanto, usado por un gran número de usuarios. Este hecho compensa con creces el ritmo diferente de trabajo.

II.3 OBJETIVOS.

El objetivo principal de PEPIM es la automatización total de la construcción de lo que se conoce como cadena de desarrollo o taller de programación de una máquina. Nuestra inquietud nace de la dificultad y el tiempo que toma la realización de uno de estos talleres; la automatización debe resolver ambos problemas, disminuyendo de manera importante los costos de producción.

Esta automatización debe cumplir sin embargo con ciertos imperativos:

a) Eficiencia de las herramientas obtenidas por medio de las herramientas de producción automática. Somos conscientes que esta eficiencia no será de ninguna manera igual a la de las herramientas construídas manualmente pero tampoco queremos obtener tiempos de ejecución o espacios de memoria inmensos y, por lo tanto, prohibitivos.

b) Facilidad de uso. La construcción de un nuevo taller no tiene porque ser una tarea dificultosa, aunque los usuarios sean, por lo general, especialistas. El sistema debe tratar de ayudar al usuario a todo nivel: diseño, realización, verificación, nuevas versiones, etc.

c) Modularidad del taller final. Los componentes que serán incluídos en un taller dependen de lo que ya existe en la máquina destinación (aquella para la cual se construye el taller); así, por ejemplo, puede suceder que sólo sea necesario construir un compilador C, dado que la máquina destinación ya cuenta con un ensamblador, un editor de referencias y un cargador. Por otro lado, las herramientas construídas con PEPIM deben poder ser integradas armoniosamente entre ellas y sobre todo con otras herramientas, completamente ajenas a PEPIM.

Otro objetivo importante de nuestro proyecto es la calidad del código generado a través de los compiladores construídos con PEPIM. Esta calidad debe ser igual y, en la medida de lo posible, superior a la de los compiladores de producción disponibles en el mercado; en este sentido, conservamos los mismos objetivos de GEMME.

En cuanto a las máquinas aceptadas, nos hemos limitado a la familia de máquinas a registros con posibilidades de extendernos hacia las máquinas a pila. En este aspecto, somos un poco más ambiciosos que GEMME y pensamos con ello poder cubrir casi el 99% de las máquinas existentes.

II.4 CARACTERISTICAS GENERALES.

El sistema PEPIM está compuesto de un conjunto de herramientas de producción automática de traductores, que permiten construir compiladores y un ensamblador para una máquina dada; el sistema también comporta otras herramientas, que son completamente independientes de la máquina destinación. La construcción de un nuevo taller se hace creando el (o los) traductores apropiados y ensamblándolos con el resto de herramientas. Ciertos componentes del taller pueden ser reemplazados por las herramientas del constructor de la máquina (ensamblador, editor de referencias, cargador), para guardar una mayor compatibilidad con el sistema operativo del mismo.

El sistema PEPIM ofrece además un entorno interactivo que permite la creación o la modificación de una descripción de máquina a un nivel de abstracción superior al del lenguaje de descripción propiamente dicho. Este entorno ofrece una interfase mucho más agradable, así como numerosas facilidades: formularios simplificados de descripción, deducción automática de ciertas informaciones, inicialización a partir de la descripción de alguna otra máquina, generación de programas de test, etc.

El conjunto de herramientas que pueden ser construídas por medio de PEPIM es bastante completo: ensamblador, compilador e intérprete C, compilador Pascal (aún no disponible), debugger de alto nivel, editor de referencias, cargador; es decir, prácticamente todo el software de base para una nueva máquina o procesador. Todas estas herramientas se integran armoniosamente y constituyen un verdadero taller de programación. Cabe indicar que el usuario puede construir únicamente los componentes que le faltan y enriquecer así su taller de programación: las técnicas desarrolladas permiten tomar en cuenta las características de los elementos existentes e inclusive su uso al interior de las herramientas generadas por PEPIM; así, por ejemplo, un compilador C generado por PEPIM producirá un programa objeto compatible con el editor de referencias del sistema operativo de la máquina.

III. CONSTRUCCION AUTOMATIZADA DE TALLERES.

III.1 ANALISIS DEL PROBLEMA.

La idea de base del proyecto consiste en tratar de separar las informaciones que dependen de la máquina destinación y aquellas que son independientes de la misma. De esta manera, las herramientas pueden ser descompuestas en dos partes: una común a todas las máquinas (parte algorítmica) y otra específica a cada máquina (parte datos). Todas las características de la máquina de destinación serán reunidas en un conjunto de variables y de tablas, que conformarán la parte datos de las herramientas; la

construcción de una nueva herramienta necesitará por lo tanto la escritura de esta segunda parte.

Este enfoque permite reducir de manera apreciable los costos de producción, puesto que una parte importante de las herramientas es realizada una sola vez.

Sin embargo, la construcción manual de la parte datos constituye una tarea bastante larga y fastidiosa, además de poco fiable; es por ello que hemos preferido que esta tarea sea efectuada automáticamente por PEPIM, a partir de una descripción de la máquina destinación en un lenguaje apropiado.

Esta descripción contiene un cuerpo común al conjunto de herramientas del taller y una serie de anexos propios a cada una de las herramientas; cabe indicar que ciertas herramientas no poseen un anexo propio, sea porque las informaciones del cuerpo de la descripción les son suficientes, sea porque son completamente independientes de la máquina de destinación; es el caso del analizador C, de su intérprete, del linkage editor, etc. En cambio, todos los traductores poseen un anexo específico.

III.2 CONSTRUCCION DE UN ENSAMBLADOR.

La construcción de un ensamblador necesita principalmente dos tipos de informaciones:

a) La sintaxis del lenguaje ensamblador. Una parte de esta sintaxis es fija e idéntica para todos los ensambladores construídos con PEPIM; se trata esencialmente de la representación de las unidades lexicales (constantes, identificadores, comentarios, etc.) y de la disposición de las informaciones en una línea fuente. Otra parte de las informaciones debe ser descrita, pues depende de la máquina destinación:
   - representación de los modos de direccionamiento,
   - mnémonicos de las instrucciones,
   - tipos de secciones y como van a ser direccionadas.

   Por otro lado, el usuario tiene la posibilidad de definir ciertas directivas o completar algunas otras; sin embargo, la mayor parte de las directivas son fijas.

b) El formato de cada instrucción, el cual incluye:
   - el número de operandos,
   - los modos de direccionamiento que acepta la instrucción para cada operando,
   - el formato binario de la instrucción.

En ciertos casos, un tercer tipo de información puede ser necesario: el formato bajo el cual debe ser generado el código objeto; esto sucede cuando se quiere usar un editor de referencias que ya existe y no aquel que es propio a PEPIM.

Todas estas informaciones forman parte de la descripción de la máquina destinación. Sin embargo, algunas de ellas son comunes a la generación del ensamblador y de los compiladores: modos de direccionamiento, familias de modos y sintaxis del lenguaje de ensamblaje.

Estas informaciones son transformadas por PEPIM, luego de una verificación, en tablas que serán incorporadas a un esqueleto de ensamblador para dar nacimiento al ensamblador deseado. Estas tablas son generadas bajo la forma de un programa C normal y, luego de ser compiladas, linkeditadas con el resto del ensamblador.

III.3 CONSTRUCCION DE UN COMPILADOR.

La principal dificultad para la construcción automática de un compilador reside en la parte generación de código del mismo. Efectivamente, el uso de herramientas de construcción automática de analizadores se ha vuelto banal estos últimos años, dada la fiabilidad y la facilidad de uso alcanzadas por estas herramientas [Johnson 75, Aho 86]; por el contrario, las herramientas e inclusive los métodos de construcción automática de generadores de código son más bien escasos. Dado que PEPIM constituye una prolongación del proyecto GEMME, fue natural que escogiéramos el método y las herramientas desarrollados en éste [Santana 86].

Las características principales del método de generación de GEMME son las siguientes:

a) Se trata de una solución integrada, que toma en cuenta todos los aspectos de la generación de código. Sin embargo, la selección de instrucciones ha sido privilegiada por el hecho de constituir la fuente de optimizaciones más rica.

b) La selección de instrucciones está basada en una búsqueda exhaustiva; por esta razón, el código generado es siempre el mejor a nivel local. Por otro lado, los algoritmos correspondientes son más simples y sobre todo más generales.

c) Todos los datos relativos a la máquina destinación son conservados en tablas; de esta manera, los mismos algoritmos pueden ser usados para máquinas diferentes: basta con cambiar las tablas.

d) En razón de su fuerte interacción, las diferentes tareas de la generación son hechas simultáneamente (asignación de registros, implantación de variables, traducción de tipos, administración de temporarios, selección de instrucciones, etc).

El generador de código resultante puede ser visto como un intérprete dirigido por las diferentes tablas de descripción de la máquina destinación:
- tabla de objetos de la máquina (representación de los datos, tipos existentes, registros, descripción de la memoria, etc),
- tabla de modos de direccionamiento y de sus clases,
- tabla de operaciones que indica las diferentes maneras de traducir cada operación del lenguaje fuente,
- sintaxis del lenguaje ensamblador que debe ser generado.

El generador descompone el programa en curso de compilación en operaciones simples, que son sometidas al mecanismo de selección de instrucciones, elemento central de nuestro método de generación. Este mecanismo efectúa un recorrido exhaustivo de las instrucciones que pueden ser usadas para traducir la operación y genera la mejor secuencia de instrucciones que encuentra; este recorrido exhaustivo nos permite aprovechar al máximo los modos de direccionamiento y el juego de instrucciones de la máquina.

Como se puede observar, el generador es tan sólo un esqueleto, que tiene que ser completado con las tablas indicadas para obtener un verdadero generador de código. La construcción se hace exactamente de la misma manera que en el caso de un ensamblador: la descripción es transformada en tablas C, que luego de ser compiladas son linkeditadas con el esqueleto.

Gran parte de las informaciones necesarias para construir un compilador son compartidas con el ensamblador.

III.4 CONSTRUCCION DE OTROS COMPONENTES.

Los otros componentes de PEPIM pueden ser considerados independientes de la máquina destinación. Así, por ejemplo, el analizador C necesita muy pocas informaciones sobre la máquina, al igual que el intérprete C: tamaño de cada tipo de base, disposición de los bits en una palabra. Por su lado, el editor de referencias y el cargador de PEPIM son completamente independientes de la máquina destinación; si el usuario decide usar otro editor de referencias, el formato objeto que éste usa deberá ser descrito, para que el ensamblador pueda hacer uso de él.

Todos estas herramientas han sido realizadas de manera a ser totalmente independientes de la máquina destinación, sea retrasando el tratamiento correspondiente hasta una etapa posterior, sea tomando ciertas hipótesis de funcionamiento. Por consiguiente, estas herramientas son las mismas en todos los talleres construídos con PEPIM. En realidad, existen ciertas diferencias, debidas en su mayor parte a problemas de interface, tanto con el usuario como con el sistema operativo; sin embargo, estas diferencias no son conceptuales y, por lo tanto, sin importancia.

IV. ARQUITECTURA DE PEPIM.

IV.1 ORGANIZACION GENERAL.

El sistema PEPIM está organizado en dos niveles:

a) Generación de herramientas, que permite construir una herramienta dada, a partir de una descripción de la máquina destinación y de un esqueleto o parte algorítmica de esta herramienta (ver capítulo III).

b) Taller de programación, constituído por las herramientas generadas por el nivel anterior y por aquellas que son independientes de la máquina destinación.

Esta organización está ilustrada por la figura 1.

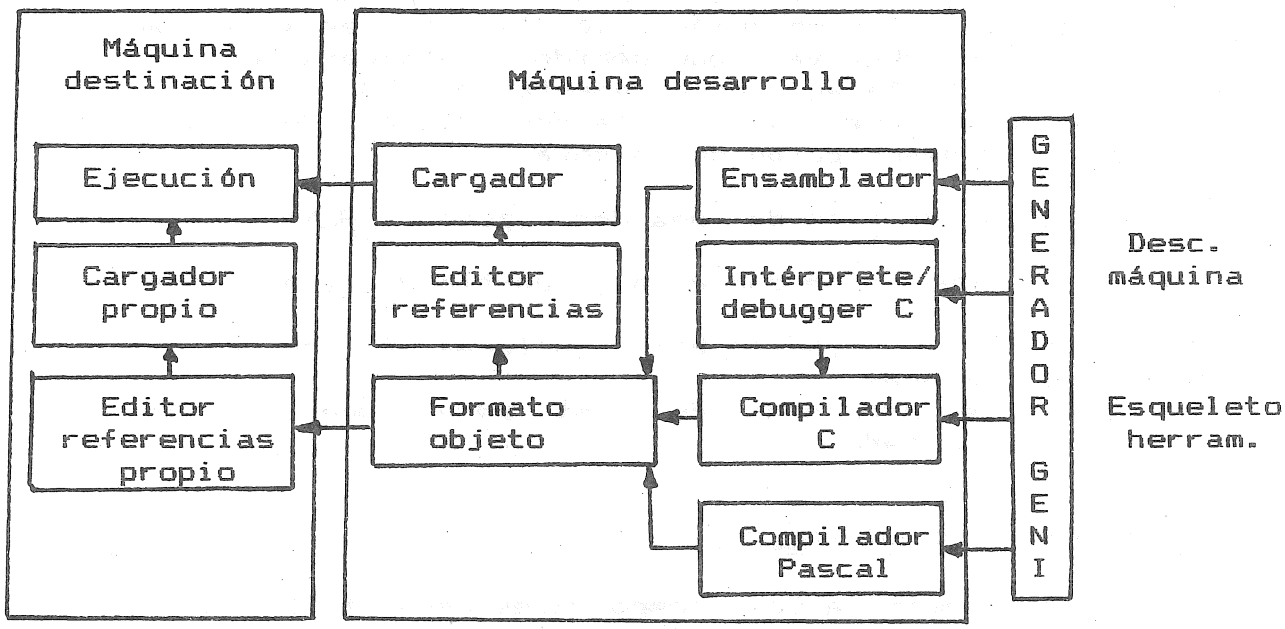

Figura 1. Organización general de PEPIM.

IV.2 SISTEMA DE AYUDA PARA CONSTRUIR UN TALLER.

La descripción de la máquina destinación puede ser hecha en un lenguaje que hemos definido para este efecto (Lenguaje de Descripción o LD). Sin embargo, esta tarea puede resultar sumamente tediosa, por la cantidad de modos de direccionamiento e instrucciones de una máquina, y poco fiable, por el gran número de opciones e informaciones puntuales que requiere nuestro sistema. Es por ésto que decidimos tomar un enfoque radicalmente opuesto, que consiste en ofrecer un sistema interactivo de ayuda para la construcción del taller: el sistema GENI (GENerador Interactivo).

GENI ofrece numerosas facilidades para la confección de una descripción:
- formularios de descripción simplificados,
- menús para escoger las opciones,
- valores por defecto o deducidos a partir de otros (algunos tienen que ser confirmados por el usuario),
- jerarquización de la descripción (acceso secuencial y/o directo a las rúbricas),
- verificación de la coherencia entre las diferentes rúbricas,
- indicación de las partes incompletas,

etc.
Durante el trabajo de edición, GENI hace uso de la división en cuerpo principal y anexos de la descripción, tanto para guiar al usuario como para pedirle sólo las informaciones estrictamente necesarias.

Al final de cada sesión, GENI produce el programa equivalente en lenguaje LD, representación bajo la cual es almacenada la descripción. Cabe observar que debido a esta característica, el usuario puede trabajar directamente en LD, si lo desea; igualmente, puede hacer uso de la descripción de otra máquina para comenzar la descripción de la suya.

Aparte de la edición de una descripción, GENI ofrece otros servicios:
- construcción de una herramienta, a partir de la descripción en curso,
- listado de la descripción bajo su forma LD, con una tabla de referencias cruzadas,
- generación de programas de test para poner a punto la descripción realizada.

IV.3 COMPONENTES DE UN TALLER.

La figura 2 muestra los componentes de un taller que pueden ser construídos a través de PEPIM. Esta misma figura pone en relieve las relaciones entre estos componentes, así como las diferentes representaciones que puede tomar un programa.

Es importante destacar la estrecha relación que existe entre el compilador y el intérprete/debugger C. Estas dos herramientas están articuladas alrededor de una representación intermediaria común, que permite tanto la interpretación del programa como su traducción en lenguaje assembler. El intérprete permite ejecutar el programa del usuario, ayudándolo en su puesta a punto con mecanismos como:
- visualización de la función en curso de ejecución e indicación de la instrucción ejecutada,
- trazado de variables o de funciones,
- breakpoints asociados a instrucciones o funciones, con posibilidad de ser activados únicamente bajo ciertas condiciones (contadores, cierto valor de una expresión, identidad de la función que hace el llamado, etc),

- ejecución paso a paso,
- evaluación de expresiones y funciones desde el lenguaje de comandos,
- acceso a la pila del programa C,

etc.

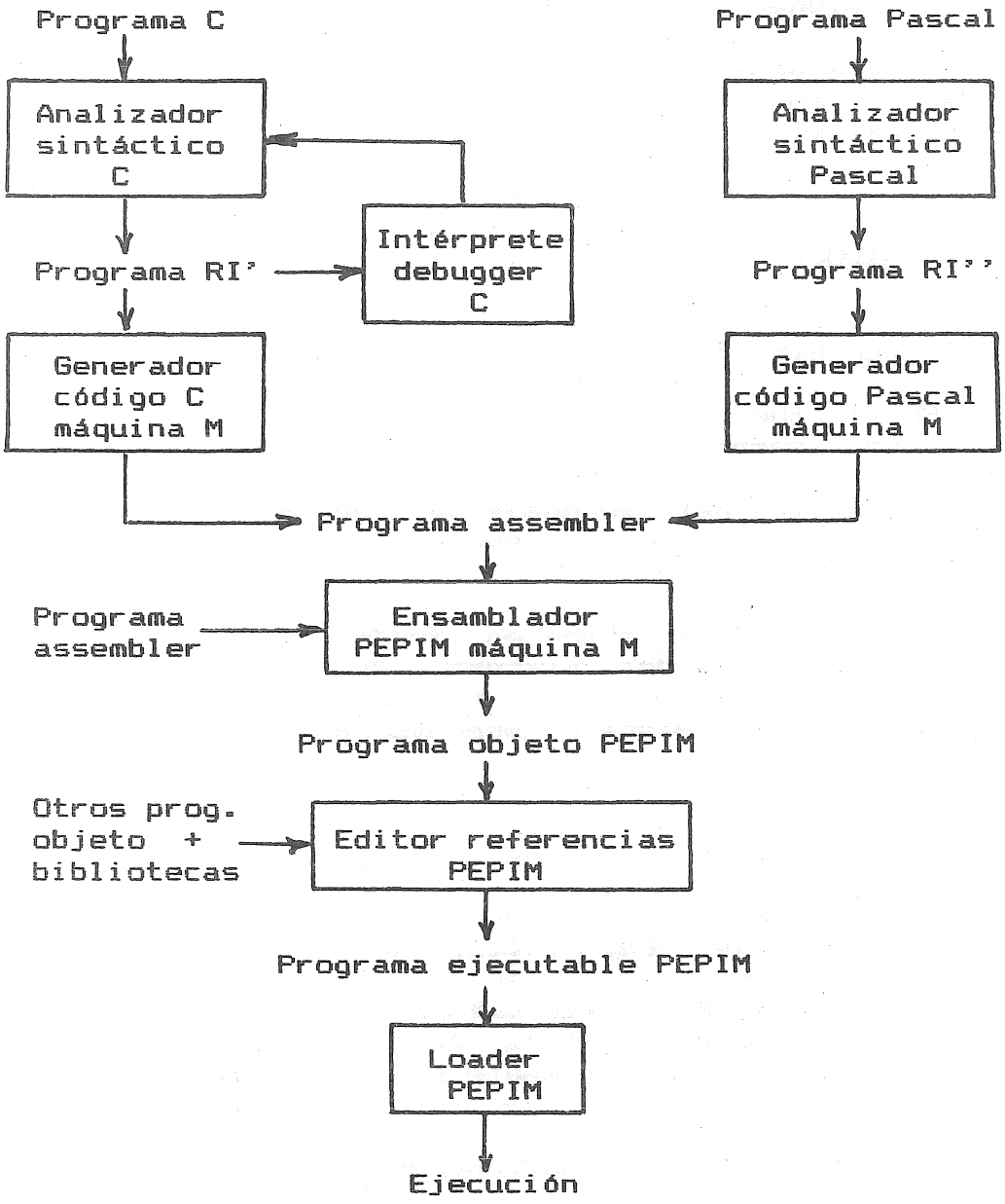

Figura 2. Taller de programación.

Como se puede observar, el intérprete/debugger trabaja a nivel de programa fuente, lo que le permite efectuar todo tipo de verificaciones sobre el comportamiento del programa: control de tipos, apuntadores con direcciones incorrectas, pasaje de parámetros incorrecto, etc. Por otro lado, el usuario manipula directamente los objetos que él ha definido (variables, instrucciones, funciones), sin necesidad de conocer ni direcciones ni el lenguaje assembler o binario de la máquina destinación. Cabe destacar igualmente el cuidado con el que ha sido realizada la interfase de esta herramienta, dada su importancia en el taller.

V. RESULTADOS OBTENIDOS.

V.1 ESTADO ACTUAL DEL SISTEMA.

El proyecto PEPIM fue iniciado en el mes de Setiembre de 1985, cuando se comenzaron a realizar las primeras especificaciones: arquitectura del sistema, generador de ensambladores, definición del lenguaje LD. Desde entonces, una gran parte del sistema ha sido implementado, aunque aún faltan ciertos elementos.

Nuestra estrategia ha consistido en desarrollar primero los componentes más importantes del sistema, de manera a poder mostrarlos a los clientes potenciales, y a incrementar su número y calidad progresivamente. Hasta el momento han sido realizados y entregados:

- ensamblador,
- analizador sintáctico C,
- intérprete/debugger C,
- generador de código C,
- editor de referencias (link) y cargador (loader),
- compilador del lenguaje LD, incluyendo los anexos que corresponden a las herramientas enumeradas.

Por otro lado, el sistema GENI está siendo desarrollado en este momento y debe ser entregado a fines del año 1987; una parte importante de GENI funciona actualmente y es usada internamente por el LGI y Silicone.

Varias herramientas han sido construídas con los componentes disponibles: ensambladores para el 68000, el 8086 y el 6502, un compilador C para el procesador Texas C25 y para el 68020, etc.

V.2 PORTABILIDAD DEL SISTEMA.

La portabilidad es un aspecto importante en un sistema como PEPIM, puesto que se trata de un sistema de desarrollo cruzado que tiene que ser instalado en diferentes máquinas de desarrollo, según la evolución del mercado. Es por esta razón

que nuestra implementación ha sido cuidadosamente realizada, con el objetivo de alcanzar la mayor portabilidad posible.

PEPIM ha sido implementado en C, por tratarse del lenguaje más portable de nuestros días. Por otro lado, hemos tratado de evitar todas las construcciones C que puedan resultar conflictivas en un transporte y previsto los problemas inevitables de representación de datos.

El sistema ha sido desarrollado en un VAX 785 bajo el sistema UNIX Berkeley 4.3; desde hace algunos meses, migramos hacia un MicroVAX II bajo el sistema VMS, sin mayores problemas. Para cerciorarnos de la portabilidad de nuestro sistema, hemos efectuado algunas experiencias de transporte de componentes; entre éstas, cabe destacar el transporte de la parte de generación de ensambladores hacía tres máquinas diferentes (Micromega, Sun 3 y PC-IBM), sin encontrar dificultades mayores.

El primer transporte del conjunto del sistema será hacia los PC-IBM y compatibles, a partir del mes de Octubre de 1987.

V.3 PERFORMANCES ALCANZADAS.

La eficiencia de las herramientas construídas con PEPIM constituye uno de nuestros principales objetivos: ésta debe mantenerse dentro de lo razonable y de ninguna manera constituir un freno al uso de las herramientas.

Para verificar el cumplimiento de este objetivo, hemos realizado dos tipos de comparaciones:

a) Ensambladores generados por PEPIM contra ensambladores entregados por el constructor mismo. En este caso, las cifras son bastante dispares pero permiten formarse una idea: los ensambladores UNIX (as) son en promedio dos veces más rápidos que los nuestros, mientras que el MASM o el ASM-86 del PC-IBM son prácticamente equivalentes a nuestro ensamblador 8086.

b) Generador de código PEPIM contra generador de código GEMME. Dado que PEPIM constituye en cierto modo una mejor implementación de GEMME, esta comparación nos permite observar la influencia de las mejoras introducidas así como del cambio de lenguaje de desarrollo (C en vez de Lisp); por otro lado, las medidas tomadas para evaluar GEMME nos permiten hacer comparaciones indirectas con otros compiladores. Nuestras primeras medidas indican un alza de aproximadamente 50% de la eficiencia respecto de GEMME; cabe indicar que las evaluaciones de éste señalaban un tiempo entre dos a tres veces mayor al de los compiladores de producción.

VI. CONCLUSIONES.

El sistema presentado en este artículo representa una respuesta a las necesidades crecientes en el área del desarrollo de software de base. El diseño de este sistema incorpora técnicas bastante avanzadas y no por ello pierde su calidad de producto utilizable industrialmente.

El anuncio de PEPIM atrajo el interés de numerosos constructores de computadores, algunos de los cuales ya han concretizado este interés a través de contratos de colaboración o de desarrollo. Entre estas compañías podemos citar Texas Europe (compilador C para su procesador C25), Thomson (taller completo para su nuevo procesador en tratamiento de señales) y SFENA (compilador C para un procesador especializado en la aviación). Otras compañías están negociando en estos momentos contratos similares (NEC y Motorola).

Diferentes desarrollos han sido previstos durante el año 1988, para consolidar y completar nuestro sistema. La primera prioridad ha sido dada a la optimización y mejora del sistema actual: mayor eficiencia, transporte hacia otras máquinas de desarrollo (PC-IBM, Sun y Apollo), mejora de la interfase de GENI, etc. Después de esta etapa, nos gustaría agregar nuevas herramientas:

- compilador Pascal,
- intérprete/debugger Pascal,
- editores sintácticos C y Pascal,

y, en un futuro más lejano, un generador de simuladores.

BIBLIOGRAFIA

[Aho 86]
A. Aho, R. Sethi, J. D. Ullman, "Compilers. Principles, techniques and tools", Addison-Wesley, 1986

[Concerto 86]
"CONCERTO: Atelier de logiciel. Présentation technique", CNET Lannion, Perros-Guirec, Febrero 1986.

[Coussediere 85]
L. Coussedière, M. Santana, "Atelier logiciel: Spécifications générales", TR DC-068-01, LGI/Silicone, Grenoble, Octubre 1985.

[Crawford 82]
J. Crawford, "Engineering a production code generator", Sigplan 82 Symposium on Compiler Construction, Boston, Junio 1982, p. 205-215.

[Ganapathi 82]
M. Ganapathi, C. Fischer, "Description driven code generation with attribute grammars", 9th Annual Symposium on Principles of Programming Languages, Albuquerque, Enero 1982, p. 108-119

[Graham 82]
S.L. Graham, R.R.Henry, R.A. Schulman, "An experiment in table driven code generation", Sigplan 82 Symposium on Compiler Construction, Boston, Junio 1982, p. 32-43

[Hochain 86]
J.C. Hochain, M. Santana, "Génération de code multi-machine", Jornadas CONCERTO, Perros-Guirec, Febrero 1986

[Johnson 75]
S.C. Johnson, "YACC: yet another compiler-compiler", TR 32, Computing Science, Bell Laboratories, Murray Hill, 1975

[Leverett 80]
B.W. Leverett y al, "An overview of the Production Quality Compiler-Compiler project", IEEE Computer 13-8, Agosto 1980, p. 38-49

[Santana 83]
M. Santana, "Un système de production automatique de générateurs de code", Facultad de Informática, Universidad de Grenoble, Diciembre 1983

[Santana 84]
M. Santana, J.C. Hochain, "Un sistema de producción automática de generadores de código", IV International Conference on Computer Science, Santiago de Chile, Junio 1984

[Santana 86]
M. Santana, "Notre expérience avec le générateur de code multicible GEMME", 2das Jornadas JISI, Túnez, Abril 1986