HoloVM: Uma máquina virtual com suporte à Concorrência, Mobilidade e Blackboards

Alex Sandro Garzão[1], Jorge Luis Victória Barbosa[2]
[1,2]*Universidade do Vale do Rio dos Sinos (Unisinos)*
Av. Unisinos, São Leopoldo, RS, Brasil
{ alexgarzao@bol.com.br, barbosa@exatas.unisinos.br }

Resumo

Este artigo propõe uma máquina virtual com suporte à concorrência, mobilidade e blackboards, realizando processamento simbólico. Após explicações sobre o que é o Holoparadigma e a Hololinguagem, será demonstrada a estrutura desta máquina virtual, juntamente com o projeto de um compilador e um montador assembler que serão utilizados para auxiliar na validação e uso desta máquina.

1. Introdução

Atualmente existem diferentes propostas e implementações de máquinas virtuais. Porém, apesar destas inúmeras opções (por exemplo, JVM [8], WAM [4] e MOZART [6]), nenhuma conseguiu suprir as necessidades propostas pelo Holoparadigma, algumas por não terem recursos essenciais a este paradigma e outras pelo baixo desempenho apresentado. A JVM (*Java Virtual Machine*), mesmo não suprindo estas necessidades, proporcionou subsídios suficientes para uma rápida prototipação do ambiente de desenvolvimento, bem como a explicitação de vários recursos inerentes deste novo paradigma de software. Este ambiente, denominado HoloJava [15], traduz um programa Holo para a linguagem Java.

Porém, mesmo tendo a JVM suprido parte destas necessidades, ainda assim não houve a possibilidade de implementar alguns recursos importantes. Assim surgiu a proposta da HoloVM, uma especificação de máquina virtual com suporte nativo à concorrência (entes concorrentes e ações concorrentes), mobilidade e *blackboards*, proporcionando assim uma melhor utilização destes recursos e tornando a execução dos programas mais eficiente. Além disso, está sendo definido e implementado um compilador para a Hololinguagem e um montador para a Holoassembler (ambos abordados na seção 4).

A seção 2 apresenta uma introdução ao Holoparadigma e a Hololinguagem. Por sua vez, a seção 3 contém uma descrição da HoloVM. As seções 4, 5 e 6 abordam, respectivamente, o ambiente para execução de programas, sua implementação e a conclusão do artigo.

2. Holoparadigma e Hololinguagem

O Holoparadigma é um novo paradigma de software que integra os conceitos de paradigmas básicos e estimula a exploração automática do paralelismo [1].

A semântica do Holoparadigma estabelece a utilização de duas unidades de modelagem: o ente e o símbolo. O ente é a principal abstração do Holoparadigma, enquanto que o símbolo é utilizado para descrever os entes e suas relações. Dessa forma, o símbolo é considerado como um átomo de informação neste paradigma.

Existem dois tipos de entes:

- Elementar: ente atômico que não possui níveis de composição;
- Composto: ente formado pela composição de outros entes.

Um ente elementar é organizado em três partes: interface, comportamento e história. A interface descreve suas possíveis relações com os demais entes; o comportamento contém ações que implementam sua funcionalidade; e a história é um espaço de armazenamento compartilhado no interior de um ente. Um ente composto (Figura 1) possui a mesma organização do elementar, porém, suporta a existência de outros entes na sua composição (entes componentes). Cada ente possui uma história que fica encapsulada no seu interior, e no caso dos entes compostos, ela é compartilhada pelos entes componentes. A Figura 2 mostra um ente composto de três níveis e exemplifica os níveis de encapsulamento da história. Os entes componentes participam do desenvolvimento da história compartilhada e sofrem os reflexos das mudanças históricas. Sendo assim, podem

existir vários níveis de encapsulamento da história, porém, eles acessam somente a história no seu nível.

Um ente assemelha-se a um objeto [2] do paradigma orientado a objetos. Do ponto de vista estrutural, a principal diferença consiste na história, a qual atua como uma forma alternativa de comunicação e sincronização. Além disso, existe maior enfoque na composição e suporte implícito da mobilidade. Um ente composto assemelha-se a um grupo. Neste caso, a história atua como um espaço compartilhado vinculado ao grupo (ente composto). O Holoparadigma propõe a utilização do processamento simbólico como a base para o tratamento de informações.

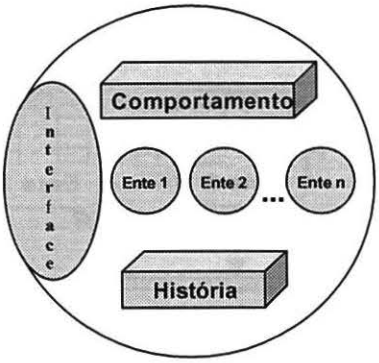

Figura 1. Ente composto

A mobilidade é a capacidade que permite o deslocamento de um ente. No âmbito do Holoparadigma existem dois tipos de mobilidade:

- Mobilidade Lógica: relaciona-se com o deslocamento a nível de modelagem, ou seja, sem considerações sobre a plataforma de execução. Neste contexto, um ente se move quando cruza uma ou mais fronteiras de entes (Figura 3);
- Mobilidade Física: relaciona-se com o deslocamento entre nodos de uma arquitetura distribuída. Neste contexto, um ente se move quando desloca-se de um nodo para outro (Figura 4).

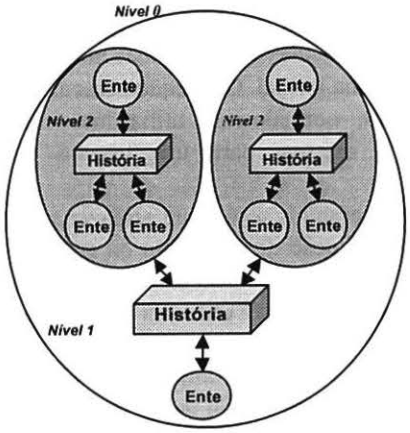

Figura 2. Ente com três níveis

O modelo de coordenação dos entes compostos assemelha-se a um *blackboard* [3]. Neste caso, os *Knownledge Sources* (KSs) são entes e o *blackboard* é a história. Este modelo utiliza invocação implícita, ou seja, o *blackboard* realiza a comunicação e o sincronismo entre KSs. Em Holo, os entes influenciam outros entes através da história (invocação implícita), mas também podem trocar informações diretamente (invocação explícita).

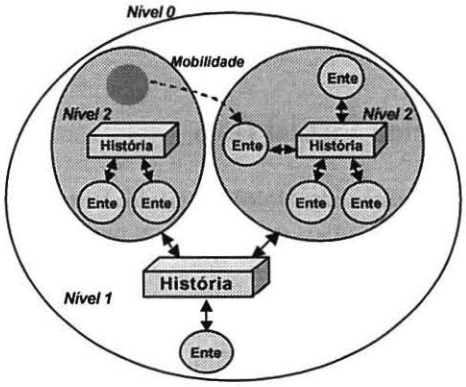

Figura 3. Mobilidade lógica

Para uma melhor exploração deste paradigma, foi criada a Hololinguagem, uma linguagem de programação que implementa os conceitos propostos pelo Holoparadigma.

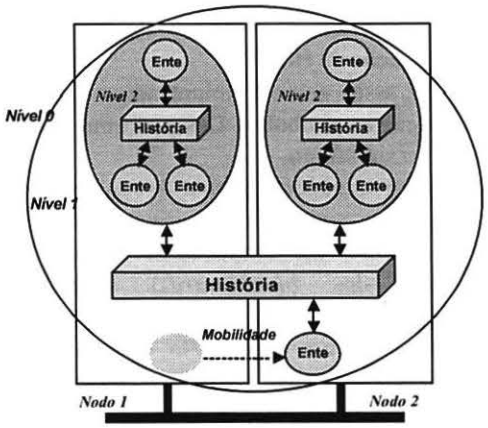

Figura 4. Mobilidade física

Um programa em Holo é composto de descrições de entes. Na Figura 5 é mostrada a descrição de um ente: na interface são inseridos os cabeçalhos das ações que podem ser acessadas por outros entes; no comportamento são descritas as ações que suportam a funcionalidade de um ente; a história é uma área de memória compartilhada pelas ações descritas no comportamento e pelos entes componentes.

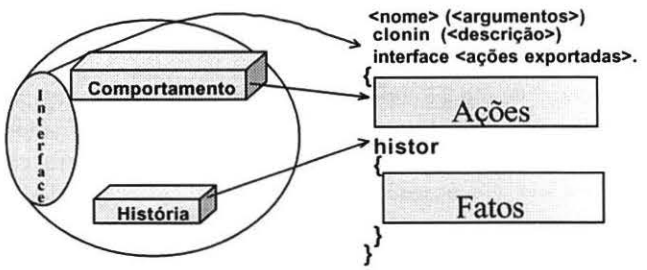

Figura 5. Descrição de um ente

3. Máquina Virtual Holo

A especificação da HoloVM (Holo Virtual Machine) está focada em recursos que não foram possíveis de serem implementados utilizando a JVM. Dessa forma, haverá meios para uma melhor exploração desse paradigma, propiciando novas idéias e melhorias.

Dentre os projetos que foram pesquisados, é possível citar o WAM, Oz [7], Mozart e JVM. Algumas das estruturas da HoloVM foram baseadas na implementação da JVM [8,9,10,11].

A idéia básica de uma máquina virtual é criar uma abstração de hardware sobre um sistema operacional. Desta forma, a implementação da HoloVM visa criar esta abstração de forma a capacitar que um *bytecode* Holo (seção 3.4) seja executado nas plataformas onde exista uma implementação da HoloVM.

A HoloVM, assim como o Holoparadigma, possui como unidade básica o símbolo. Dessa forma, ela realiza processamento simbólico.

Pode-se citar o suporte nativo à concorrência inter-entes e intra-entes, mobilidade lógica e inserção/remoção de código nos entes em tempo de execução (através da implementação dos *blackboards*) como recursos implementados na HoloVM.

Alguns recursos propostos pelo Holoparadigma não estão sendo implementados. Porém, a implementação atual pode ser facilmente ajustada para incorporar novas funcionalidades.

A seguir há uma descrição de tópicos importantes da HoloVM.

3.1. Concorrência na HoloVM

A unidade básica de execução na máquina virtual (fluxo de execução) denomina-se *ByteCodeExecutor* (BCE). O BCE, por definição, é uma *thread* de execução (Figura 6). Cada ente, implicitamente, possui seu próprio fluxo de execução de forma a possibilitar a exploração da concorrência inter-entes (entes concorrentes). As ações podem ser invocadas de forma síncrona ou assíncrona. Quando invocadas de forma assíncrona, estas ações possuem seu próprio fluxo de execução, possibilitando assim a exploração da concorrência intra-entes (ações concorrentes). Os BCEs compartilham a mesma história e cada BCE possui sua própria pilha de operandos e pilha de controle (ambas abordadas na próxima seção). Cada ente possui sua própria história e pode realizar invocações implícitas para a história compartilhada ou explícitas para outros entes. Dessa forma, além de ser utilizada para troca de informações, a história compartilhada pelos entes também pode ser utilizada para sincronizá-los. As ações podem trocar informações através do uso da história ou através de invocações explícitas.

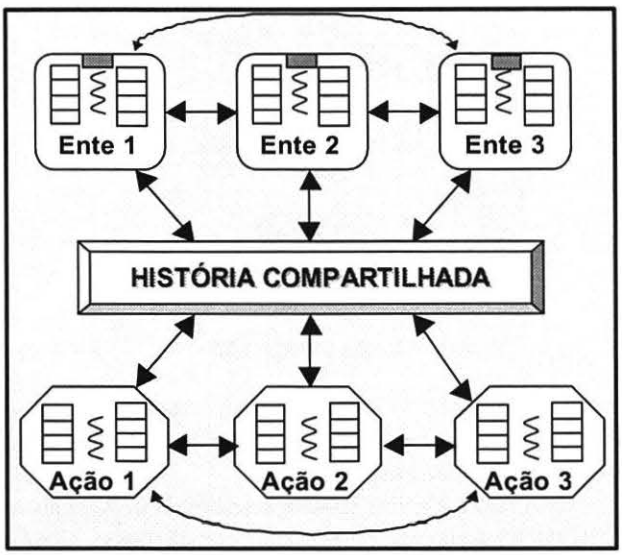

Figura 6. Concorrência na HoloVM

3.2. Pilhas da HoloVM

A idéia de uma máquina de pilha, contendo uma *OperandStack* (pilha de operandos) e uma *ControlStack* (pilha de controle) mostrou-se uma forma eficaz de obter-se uma máquina virtual rápida e de fácil implementação. Desta forma, estas pilhas são utilizadas no processamento interno da máquina virtual. Conseqüentemente, o conceito de registradores, normalmente utilizados em máquinas reais, somente existem para uso interno da própria HoloVM.

A pilha de operandos serve para armazenar parâmetros necessários à execução de uma ação ou ente, juntamente com as informações de retorno destas ações. A pilha de controle, em contrapartida, é utilizada apenas para controle interno do BCE. É onde são armazenadas informações como endereço de retorno de uma ação (quando esta ação foi invocada por outro ente ou ação), símbolos locais de uma ação, entre outros. Não há *opcodes* específicos para

acessar esta pilha, ficando seu uso restrito à máquina virtual.

3.3. Acesso aos *Blackboards*

Baseando-se na premissa que o comportamento de um ente é a implementação de alguns dos conceitos existentes em um *blackboard*, ao fazer uma afirmação para um comportamento, na verdade, está sendo inserida uma ação neste comportamento. Da mesma forma, é possível retirar uma ação deste comportamento ou aguardar que ela seja criada. Em suma, algumas características existentes nos *blackboards* estão implementadas na HoloVM e poderão ser utilizadas tanto para a invocação da história, do comportamento ou da interface de um ente.

O acesso aos *blackboards* é feito através de *opcodes* específicos para esta função. Como exemplo cita-se a existência de *opcodes* para afirmar ou consultar informações em um *blackboard*, podendo estes opcodes serem bloqueantes ou destrutivos.

3.4. *Bytecode* Holo

Atualmente, estão definidos 56 *opcodes*, dos quais 18 estão implementados. A estrutura atual da HoloVM, assim como a JVM, suporta até 255 *opcodes* distintos. Isso se deve ao fato da máquina virtual utilizar um *byte* (8 bits) para especificar um *opcode válido*.

Dentre as categorias de *opcodes* que foram definidas, pode-se salientar *opcodes* para controle do fluxo de execução, controle da pilha de operandos, manipulação de variáveis, matemáticos, lógicos, booleanos, condicionais, interação com *blackboards*, mobilidade, concorrência e para ações específicas da Hololinguagem.

A Figura 7a mostra um trecho de código na Linguagem Holo. Este código calcula uma expressão matemática, armazena o resultado em uma variável chamada X, e logo após mostra o conteúdo desta variável. Na Figura 7b encontra-se a tradução para Holo Assembler (Linguagem assembler parcialmente descrita na seção 4.1). Por sua vez, a Figura 7c mostra a utilização da pilha de operandos após a execução de cada *opcode*.

3.5. Arquivo com *bytecodes* Holo

Após a inicialização da máquina virtual, ocorrerá a carga do arquivo contendo o *bytecode* Holo. Similar ao arquivo class da JVM, foi criado o arquivo *HVM* (Holo Virtual Machine) para a HoloVM.

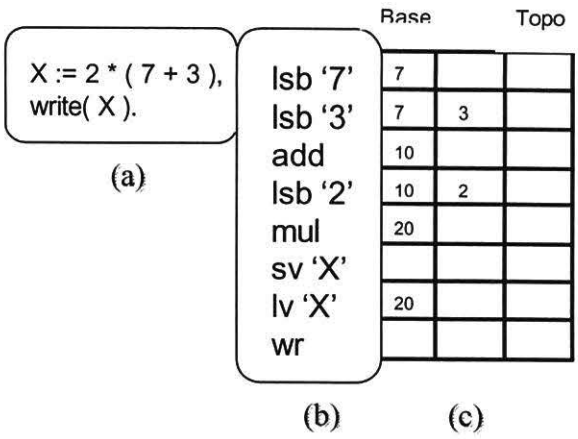

Figura 7. Exemplo de código

A Figura 8 mostra uma descrição deste arquivo, onde são guardadas as definições dos entes contidos em um programa Holo. Tanto a organização baseada em registros quanto a estrutura da *ConstantPool* foram herdadas da JVM. Há uma seção chamada *FileHeader* (cabeçalho) contendo uma assinatura que identifica o tipo de arquivo, a versão da HoloVM (visando compatibilidade com outras versões), o número de constantes e o número de entes (entende-se por constante qualquer *token* que esteja em um programa holo que não faça parte diretamente do léxico da linguagem, como mensagens aos usuários e números). Na seção chamada *ConstantPool* (grupo de constantes) há a definição de cada uma destas constantes. Na seção intitulada *BeingsDescription* (descrição de entes) há a definição dos entes, indicando seu nome, suas ações e, dentro de cada ação, os parâmetros de cada ação e seus *opcodes*.

4. Ambiente de Execução

Com a implementação da HoloVM, houve também a necessidade de ferramentas para ajudar na sua validação e testes. Assim surgiu a especificação do HoloASM e do HoloCompiler. Essas ferramentas são indispensáveis, visto que sem elas é difícil validar e verificar a potencialidade desta nova máquina virtual.

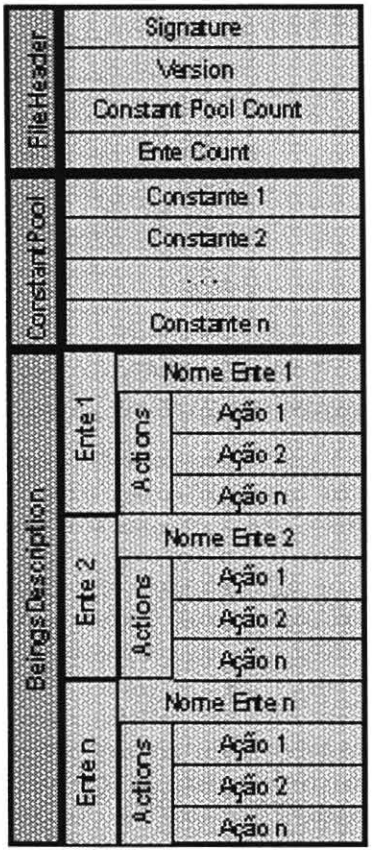

Figura 8. Estrutura do arquivo hvm

4.1. HoloASM

O HoloASM (Holo Assembler) auxilia na validação da HoloVM. Ele é capaz de ler um programa em Holoassembler e traduzi-lo para um arquivo *hvm*. A sintaxe de um programa em Holoassembler foi baseada parcialmente na Linguagem JASM (Java Assembler). As figuras 9, 10 e 11 mostram três exemplos de arquivos em HoloAssembler. O exemplo 1 exibe a mensagem "Hello world !!!", o exemplo 2 executa a clonagem de dois entes e o exemplo 3 exibe os números de 1 à 10.

```
lsb 'Hello World !!!'
wr
wrln
ret
```

Figura 9. Exemplo 1: Hello World !!!

4.2. HoloCompiler

Apesar do HoloASM ser de grande utilidade para validar a HoloVM, ele se torna uma ferramenta pouco produtiva quando comparada aos recursos disponíveis em um compilador de alto nível [12,13].

```
lsb 'Ente1'
clone
lsb 'Ente2'
clone
ret
```

Figura 10. Exemplo 2: Clonagem

Neste contexto surgiu o HoloCompiler, uma especificação de um compilador (Figura 12) para a Linguagem Holo. Este compilador traduz um programa Holo para HoloAssembler, podendo assim ser utilizado o HoloASM para finalizar a conversão para um arquivo *hvm*.

```
;; Quantos := 10
lsb 10      ;; carrega 10 na pilha
sv Quantos  ;; armazena em Quantos

;; for X := 1 to Quantos do
lsb_1       ;; carrega 1 na pilha
sv X        ;; armazena em X
:inicio_for
lv X        ;; carrega X
lv Quantos  ;; carrega Quantos
le
ifnot goto proximo_commando

;; writeln( X )
lv X        ;; carrega X
wr          ;; escreve valor
wrln        ;; salto de linha

;; finalização do for
lv X        ;; carrega X
inc         ;; incrementa
sv X        ;; armazena em X
goto inicio_for
:proximo_commando
ret
```

Figura 11. Exemplo 3: Comando FOR

5. Implementação

A implementação da HoloVM, do HoloASM e do HoloCompiler utilizam a Linguagem C++, sob a licença GPL, utilizando o compilador gcc no ambiente Linux. As bibliotecas necessárias na implementação da HoloVM, como acesso a *blackboards*, concorrência e mobilidade, estão em desenvolvimento. Por enquanto, não foi utilizada nenhuma biblioteca de código externa ao projeto. Tanto para o HoloASM como para o HoloCompiler, os analisadores léxicos, sintáticos e semânticos [12,13] utilizam o gerador de parsers chamado YAPG. Este gerador de parsers possui uma gramática muito parecida com o JavaCC [14], porém gera código para a Linguagem C++.

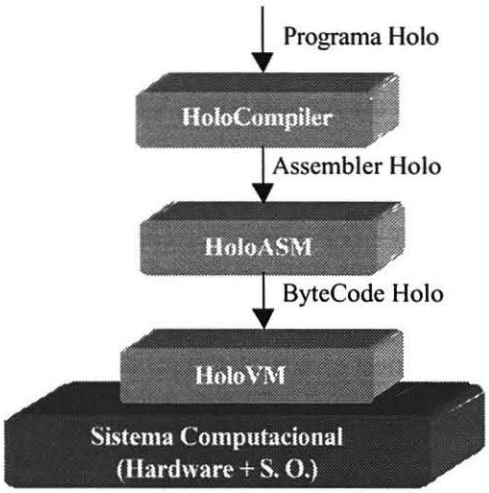

Figura 12. Ambiente para execução de programas

6. Conclusão

A máquina virtual proposta neste artigo, apesar de conter estruturas simples, proporcionará a verificação na prática de conceitos envolvidos no Holoparadigma que não foram testados até o presente momento. Foi visto também que recursos como concorrência, mobilidade e *blackboards* podem coexistir em uma máquina virtual com processamento simbólico.

Até o presente momento, tem-se a definição da estrutura da HoloVM, algumas rotinas de controle e parte da implementação das rotinas principais da máquina virtual. É possível executar *opcodes* envolvendo pilha, laços de controle, operadores booleanos e condicionais. Tem-se também parte da implementação do HoloASM. A definição da máquina virtual e o conjunto de *opcodes* necessita de correções que visam aumentar o desempenho da máquina virtual.

Como atividades futuras, destaca-se a implementação das rotinas para acesso a *blackboards*, a implementação dos *bytecodes* restantes, o término da implementação do HoloASM, a validação da HoloVM com o HoloASM e o início da implementação do HoloCompiler. Estas atividades concretizarão a proposta e indicarão os novos caminhos a serem seguidos.

Referências Bibliográficas

[1] Barbosa, Jorge L. V., and Geyer, Cláudio F. R., "Um Modelo Multiparadigma para Desenvolvimento de Software Paralelo e Distribuído", WSCAD, São Pedro, janeiro 2000.

[2] Sebesta, Robert W., "Concepts of Programming Languages", Addison Wesley, 1999. 670p.

[3] Vraned, Sanja, and Stanojevic, Mladen., "Integrating Multiple Paradigms within the Blackboard Framework", IEEE Transactions on Software Engineering, v.21, n.3, march 1995.

[4] Aït-kaci, Hassan., "Warren's Abstract Machine - A Tutorial Reconstruction", Cambridge, MIT Press, 1991.

[5] "Programming Languages for the Java Virtual Machine", Disponível em:

<http://grunge.cs.tu-berlin.de/~tolk/ vmlanguages.html>. Acesso em: novembro 2001.

[6] Roy, Peter V. et al. Mobile Objects in Distributed Oz. ACM Transactions on Programming Languages and Systems, New York, v.19, n.5, p.804-851, September 1997.

[7] Smolka, Gert. The Oz Programming Model. In: Computer science today, 1995. Proceedings... Berlin: Springer-Verlag, 1995. p. 324-343. (Lecture Notes in Computer Science, v.1000).

[8] Lindholm, Tim; Frank Yellin. The Java virtual machine specification. Addison Wesley, 1999. 473p.

[9] Tucker, Adrew, "Reading Java Class Files in C++", C/C++ User Journal, April 1998.

[10] Moss, Karl; "How Can I Measure Java Code Performance ?", Dr. Dobb's Journal, October 2000.

[11] Yourst, Matt T.; "Inside Java Class Files", Dr. Dobb's Jounal, January 1998.

[12] Aho, Alfred V.; Ravi Sethi; Jeffrey D. Ullman. "Compilers : principles, techniques, and tools", Addison Wesley, 1988, 796p.

[13] Price, Ana Maria; Simão Sirineo Toscani. "Implementação de linguagens de programação : compiladores", Sagra Luzzatto, 2001, 194p.

[14] "JavaCC – The Java Parser Generator", Disponível em: <http://www.webgain.com/products/java_cc/>. Acesso em: novembro 2001.

[15] Barbosa, Jorge Luis Victória; Du Bois, André; Pavan, Altino; Geyer, Cláudio Fernando Resin. HoloJava: Translating a Distributed Multiparadigm Language into Java. In: Conferência Latinoamericana de Informática, 27., 2001, Mérida, Venezuela. Proceedings... Mérida: Universidad de Los Andes, septiembre 2001. CD