Desenvolvendo Infra-Estrutura para a Criação de Circuitos Transformacionais

Silvia Mara Abrahão[1] Edson Murakami[1] Antonio Francisco do Prado[1] Marcelo Sant'Anna[2]

[1]Departamento de Computação
Universidade Federal de São Carlos
Via. Washington Luiz, 235
04499-610 São Carlos, Brasil
{asilvia, murakami, prado}@dc.ufscar.br

[2]Laboratório de Engenharia de Software
Pontifícia Universidade Católica do Rio de Janeiro
R. Marquês de S. Vicente, 225.
22453-900 Rio de Janeiro, Brasil
santanna@les.inf.puc-rio.br

Resumo

Sistemas de Transformação (STs) são ambientes de engenharia de software que permitem a manipulação estrutural e semântica de software. A partir de experimentos com o ST Draco-PUC e um enfoque voltado para Arquitetura de Software, foi desenvolvido na PUC-Rio o ST SpinOff que demonstra o conceito de Circuitos Transformacionais (CTs). Orientado para o desenvolvimento de transformadores de software baseados em componentes, este ambiente é destinado à construção de partes transformacionais reutilizáveis que podem ser integradas a outros componentes através de um barramento de software. Este artigo apresenta trabalhos que estão sendo desenvolvidos na UFSCar para que o ST SpinOff possa ser empregado sistematicamente.

Palavras-Chave: Sistemas de Transformação, Circuitos Transformacionais, Desenvolvimento de Software Baseado em Componentes, Memória Virtual e Técnicas de Reescrita em Grafos.

1 Introdução

O protótipo SpinOff [14] [15] é uma evolução do sistema de transformação Draco-PUC [8] [9] [10] [11] [16], com ênfase em soluções transformacionais baseadas em componentes. Procurando oferecer uma alternativa à arquitetura monolítica, típica de sistemas transformacionais, o conhecimento transformacional no SpinOff é capturado em um conjunto de componentes interoperáveis. Inspirado em técnicas de *Very Large System Integration* (VLSI), o SpinOff disponibiliza recursos para o projeto e simulação de Circuitos Transformacionais (CTs). Um circuito transformacional é uma representação arquitetônica de um transformador de software. Para que seja possível a manipulação de CTs, são necessárias diversas camadas de serviços, que compõem a infra-estrutura básica das transformações.

Na Figura 1 é apresentada a organização em camadas do SpinOff. A primeira camada é responsável pela manipulação básica de ASTs (*Abstract Syntax Trees*), que são a forma interna primária utilizada pelo SpinOff. A segunda camada, chamada *Mutant*, consiste em um *framework* para o desenvolvimento de interpretadores extensíveis de ASTs. O Mutant é usado para coletar informações ou fornecer semântica operacional a uma AST. Este *framework* é chamado de Mutant, devido sua natureza adaptativa ao fornecer extensibilidade para o desenvolvimento de interpretadores de ASTs. A terceira camada é uma máquina virtual no topo do framework *Mutant*, que fornece a funcionalidade essencial do SpinOff através de uma linguagem interpretada para a manipulação de CTs. Esta linguagem é chamada de *Software Configuration Language* (Scil).

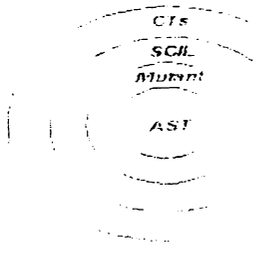


FIGURA 1 - CAMADAS DO SPINOFF

Neste artigo, detalhamos dois trabalhos que vêem sendo desenvolvidos na camada de manipulação básica do protótipo SpinOff. O primeiro trabalho pesquisa modelos de memória virtual para a gerência de ASTs e o segundo investiga como as técnicas de reescrita em grafos podem ser combinadas à manipulação de ASTs. Nas demais seções apresentamos as motivações para tais trabalhos, as abordagens adotadas e alguns dos resultados que foram alcançados até o momento no desenvolvimento destas pesquisas.

Este artigo está organizado nas seguintes seções: a seção 2 apresenta alguns trabalhos relacionados. A seção 3 descreve com mais detalhes o conceito de circuitos transformacionais, bem como sua linguagem de descrição. A seção 4 descreve o modelo de memória virtual utilizado no tratamento das ASTs. A seção 5 descreve a técnica utilizada na análise semântica. Finalmente, as conclusões e trabalhos futuros são apresentados na seção 6.

2 Trabalhos Relacionados

A aplicação de grafos que capturam informações semânticas têm recebido atenção por parte de pesquisadores na área transformacional como é o caso dos projetos DMS [7] e Refine [19]. Motivados por estas mesmas razões, procuram os pesquisadores uma técnica de trabalho mais produtiva. Por ora, os resultados são pontuais em relação a modelos de reescrita em grafos e geração automática de grafos a partir da descrição abstrata da semântica de uma linguagem.

3 Circuitos Transformacionais

A partir dos resultados alcançados pelas ferramentas VLSI, decidiu-se utilizar a idéia de projeto de circuitos eletrônicos para o desenvolvimento de transformadores. Para obter tais resultados, optou-se pelo estilo arquitetural de canais e filtros como estilo básico de trabalho. Os transformadores baseados em componentes que usam esta abordagem são chamados de Circuitos Transformacionais (CTs), dada a similaridade que os circuitos eletrônicos são projetados atualmente. Os componentes básicos de um CT são filtros, objetos, regras e funções, interligados por canais e eventos. Os CTs também possuem uma descrição textual e podem ser manipulados através de editores de composição gráfica.

A partir do conceito-chave de um circuito transformacional, vislumbra-se todo um ramo de atividades e suporte automatizado por ferramentas. Pode-se chamar este ramo de atividades de arquitetura transformacional. A essência da arquitetura transformacional está na concepção, modelagem e prototipação de circuitos transformacionais. Uma linguagem de descrição para circuitos transformacionais permite que a arquitetura de um transformador seja representada, com os objetivos de realizar esboço estrutural, modelagem e prototipação.

Com o intuito de oferecer uma estrutura básica para a construção de transformadores com enfoque em arquitetura, foi criada a linguagem Scil. Esta linguagem pode ser utilizada como modelo fundamental no desenvolvimento de ferramentas de modelagem, prototipação e síntese de CTs. Com Scil pode-se construir um ambiente de execução que operacionaliza as idéias de um CT e torna realizável os conceitos de prototipação dinâmica.

A linguagem Scil é utilizada para descrever configurações de *software* sobre a camada de comunicação do SpinOff. Esta camada é um barramento de *software* responsável pela integração entre os componentes reutilizáveis de diferentes estilos de arquitetura. Os componentes Scil podem ser criados para serem visíveis no barramento SpinOff, e podem ser utilizados por outros desenvolvedores de CTs que compartilham o barramento de comunicação.

Scil pode ser estendida para manipular diferentes estilos de arquitetura, e atualmente suporta objetos, eventos, funções, filtros e canais. Scil é uma linguagem interpretada com sintaxe semelhante a LISP, utiliza ASTs como tipo de dados básico e foi implementada em Java.

3.1 Micro-Componentes

As unidades básicas de composição que são utilizadas no desenvolvimento de CTs são chamadas de micro-componentes. O SpinOff, por *default*, já disponibiliza alguns micro-componentes que imitam clássicos sistemas de transformação baseado em regras. Tais funcionalidades estão disponíveis através dos componentes ***Transform*** e ***sot*** (*Set Of Transforms*). O componente *Transform* encapsula uma regra de reescrita, com um *left-hand-side* (lhs) e um *right-hand-side* (rhs). O componente *sot* encapsula um conjunto de transformações onde as estratégias de navegação e substituição podem ser configuradas.

A Figura 2 mostra um conjunto de micro-componentes independentes e reutilizáveis que podem ser utilizados pelos desenvolvedores na montagem de CTs para um determinado transformador. A atividade de montagem pode ser conduzida através de um editor de composição gráfica ou através de descrições Scil.

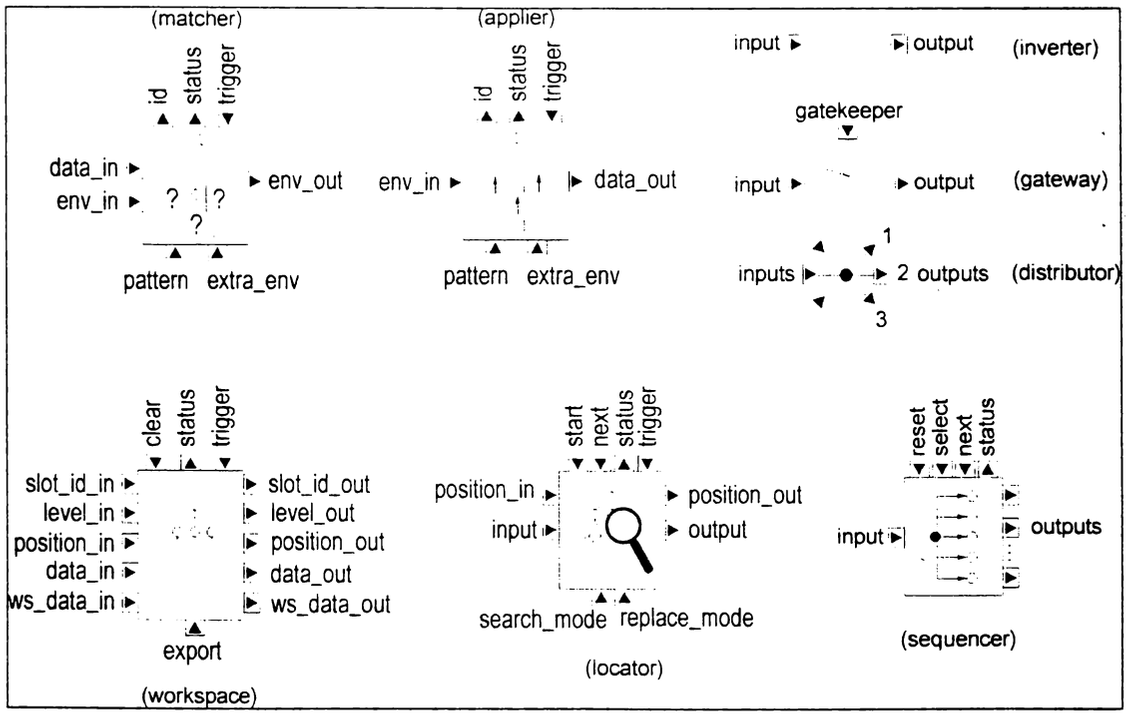


FIGURA 2 – MICRO-COMPONENTES TRANSFORMADORES

O microcomponente `matcher` é responsável pela tarefa de casamento de padrões. O `applier` trabalha juntamente com o `matcher`, e é responsável pela aplicação das regras de reescrita. Os micro-componentes `inverter`, `gatekeeper`, `distributor` e `sequencer` são utilizados para conectar componentes. A idéia de células de armazenamento para ASTs é capturada pelo microcomponente `workspace` e o conceito de *iterador,* que permite percorrer ASTs usando modos distintos de navegação é encapsulado pelo microcomponente `locator`.

3.2 Componentes de Ligação

No trabalho com CTs, além de componentes básicos que fazem efetivamente a transformação de *software* são necessários, componentes de ligação que auxiliem a adaptação entre os diversos pacotes que compõem um CT.

Nas Figuras 3 e 4 demonstra-se como os componentes de ligação, inversor (`Inverter`) e chave (`Gateway`), podem ser criados a partir de descrições Scil.

```
(define Inverter
  (package
    (in infoIn)
    (in infoOut)

    (receive data infoIn
      (send (not data) infoOut))
  )package
)define
```

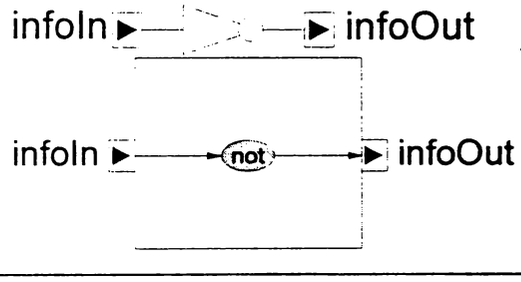
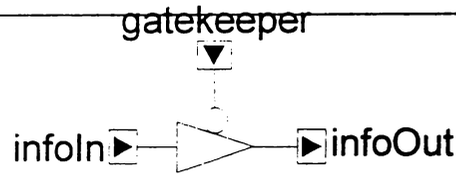


FIGURA 3 – CRIAÇÃO DO COMPONENTE INVERTER

Um inversor (`Inverter`) possui somente uma porta de entrada (`infoIn`) e uma de saída (`infoOut`). Este componente é responsável por inverter o valor de um sinal recebido. Essencialmente um inversor é um componente que encapsula a função pré-definida `not` da linguagem Scil.

```
(define Gateway
  (package
```



```
    (in  infoIn)
    (out infoOut)
    (in  gatekeeper)

    // estado
    (define open false)

    // comportamento
    (receive data infoIn
      (if open
        (send data infoOut)))
    (receive data gatekeeper
      (set! open data))
  )package
)define
```

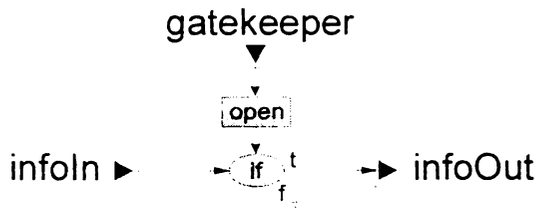


FIGURA 4 – CRIAÇÃO DO COMPONENTE GATEWAY

Uma chave (`Gateway`) possui duas portas de entrada (`infoIn` e `gatekeeper`) e uma porta de saída (`infoOut`). Este componente funciona como uma chave, deixando a informação que chega em `infoIn` ser repassada à `infoOut`, somente se `gatekeeper` tiver recebido um sinal positivo. O componente mantém o estado de chave aberta ou fechada até que um novo estado seja definido por um sinal recebido em `gatekeeper`.

A Figura 5 mostra como componentes simples podem ser interconectados através de suas portas de conexão `input` e `output`.

```
(begin
 (define simpleComponent
   (package
     (in  input)
     (out output)
   )package
 )define

 (define portConnections (package))

 (view portConnections)

 (with portConnections
   (define a (new simpleComponent))
   (define b (new simpleComponent))
   (define c (new simpleComponent))
   (define d (new simpleComponent))
   (connect a.output b.input)
   (connect a.output d.input)
   (connect b.output c.input)
 )with

)begin
```

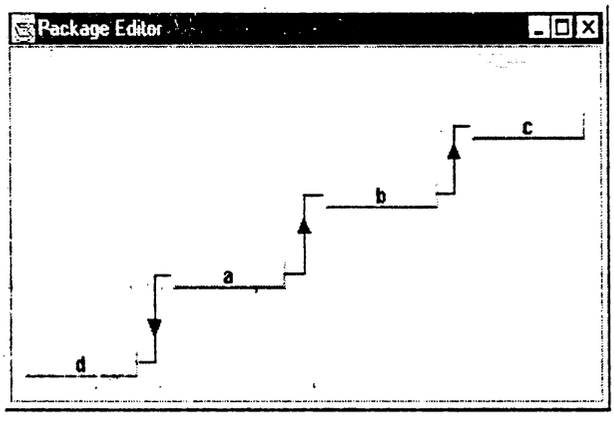


FIGURA 5 – INTERCONEXÃO DE COMPONENTES

Uma vez configurado um CT, tal como um transformador, este pode ser exportado do pacote onde foi definido para o barramento, a fim de ser utilizado em conjunto com outros componentes externos. O transformador pode ser integrado a um sistema baseado em tecnologia Web. Esta integração é possível, através do suporte que o barramento SpinOff provê ao protocolo HTTP. Este suporte permite que qualquer elemento exportado para o barramento possa ser acessado através de uma URL.

4 Modelo de Memória Virtual

O primeiro passo na implementação do SpinOff foi o desenvolvimento da camada para manipulação de ASTs. Esta camada é o subsistema AST do ST Draco-PUC, o qual foi submetido a uma reengenharia e portado para Java. O principal requisito para a reengenharia deste subsistema foi a criação de um serviço de persistência de ASTs em memória secundária, uma vez que, os ST são grandes consumidores de memória principal. Os experimentos com o ST Draco-PUC e as experiências da comunidade de sistemas de transformação têm mostrado que as ASTs são estruturas de dados que podem ser muito grandes consistindo de milhões de nós [6]. Com base nessas experiências está sendo desenvolvido um modelo de memória virtual em Java para estender a memória utilizada pelo SpinOff. O modelo possui um Subsistema Gerenciador de Memória Cache que mantém na memória principal os nós mais prováveis de serem usados durante o processo de transformação e um Subsistema de Persistência que utiliza Bancos de Dados Relacionais para armazenar as ASTs. O Gerenciador de Memória Cache utiliza o padrão de *design Cache Management* [4] [13] e o Subsistema de Persistência está baseado no padrão de integração de objetos com SGBDRs (Sistemas Gerenciadores de Banco de Dados Relacionais) *Crossing Chasms* [12].

A Figura 7 mostra o Diagrama de Classes do Modelo de Memória Virtual em Java. A classe `ASTPersistenceService` e as classes que implementam a interface `ASTBrokerIF` fazem parte do Subsistema de Persistência, as demais compõem o Gerenciador de Memória Cache.

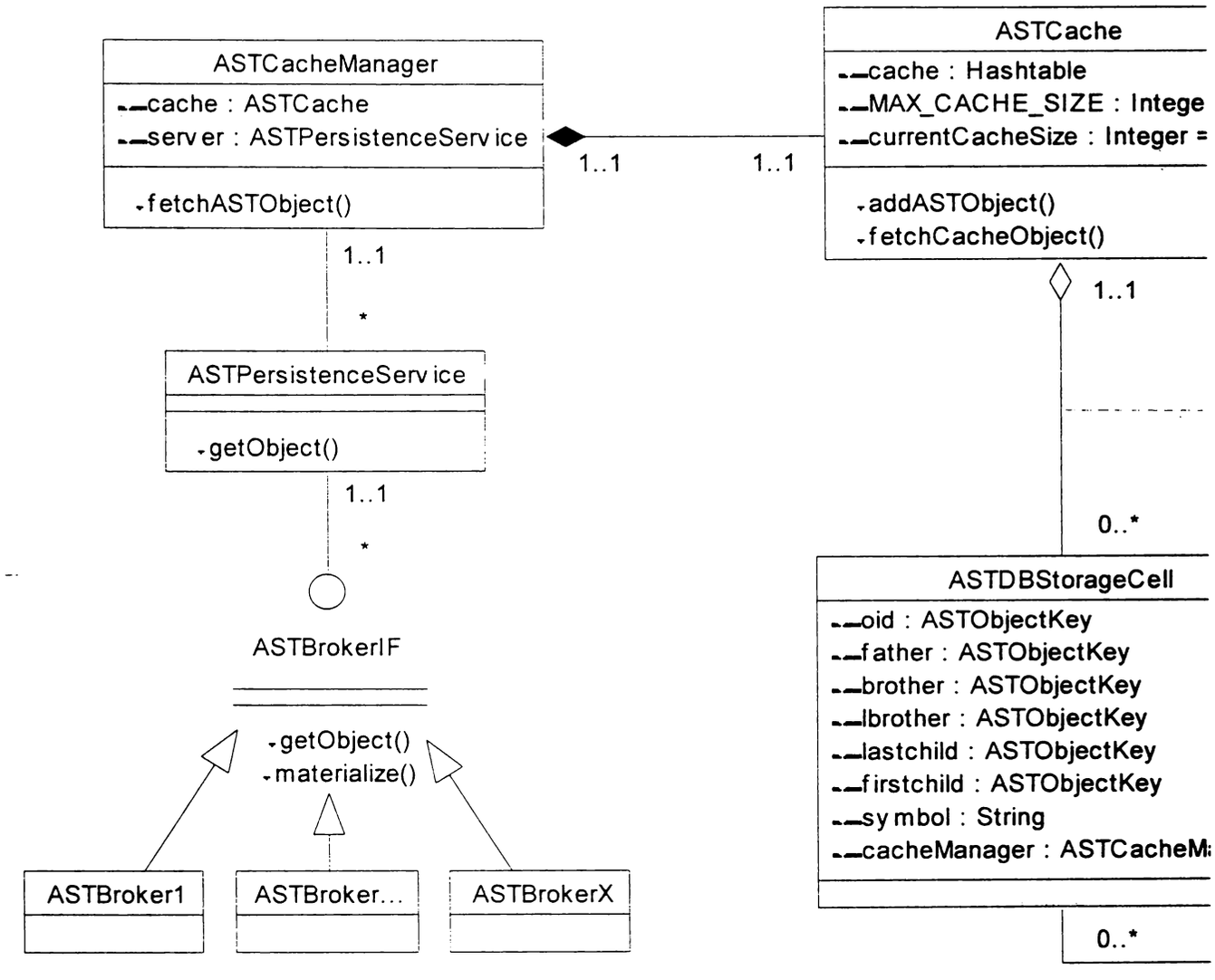


FIGURA 7 – DIAGRAMA DE CLASSES DO MODELO DE MEMÓRIA VIRTUAL EM JAVA

Durante o processo de transformação, nós de uma AST são requisitados a um objeto da classe `ASTCacheManager`. Este objeto é composto de uma memória cache representada por um objeto da classe `ASTCache`. O objeto da classe `ASTCache` é responsável em gerenciar a memória cache para manter os nós mais prováveis de serem usados durante o processo. Para esse gerenciamento é utilizado a estrutura de dados lista linear duplamente enlaçada onde, os nós mais a esquerda e mais a direita da lista são respectivamente os nós mais e menos recentemente utilizados. Quando o limite da memória cache é atingido, o objeto da classe `ASTCache` descarta o nó utilizado menos recentemente para dar lugar ao novo nó. A memória cache é composta pelos nós da AST, representados pelos objetos da classe `ASTDBStorageCell`. A classe `ASTDBStorageCell` possui um auto-relacionamento, denominado *ref*, que faz o encadeamento dos nós da AST. Estes nós são identificados na memória cache pelo atributo *oid* da classe `ASTObjectKey`.

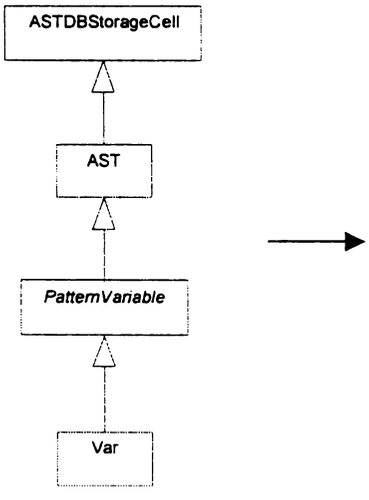
A classe `ASTPersistenceService` é um *Façade Pattern* [4] [13]. O objeto desta classe é responsável em encontrar o *broker* [12] correspondente ao nome da classe que lhe é passado como parâmetro. As classes *brokers*, tais como `ASTBroker1` e `ASTBrokerX`, implementam a interface `ASTBrokerIF` e são responsáveis pela integração com os diversos bancos de dados relacionais.

Quando um objeto da classe `ASTCacheManager` recebe uma requisição de um nó, ele tenta recuperá-lo da memória cache. Se obtiver sucesso, retorna o nó requisitado, senão requisita ao objeto `ASTPersistenceService` a materialização do nó requisitado. Uma vez materializado, este nó é colocado na memória cache como sendo o nó utilizado mais recentemente.

4.1 Protótipo do Modelo de Memória Virtual

O objetivo do protótipo foi validar o modelo de memória virtual. Este modelo de memória foi implementado na camada AST do ST SpinOff. O gerenciador de memória cache foi implementado para manter, na memória principal, os nós das ASTs mais prováveis de serem usados. A memória cache possui um número limitado de nós que foi definido através de testes que medem a porcentagem de sucesso a partir das tentativas de recuperação de nós da memória *cache*. A memória *cache* usa a estrutura de dados lista linear duplamente ligada para determinar qual nó remover da cache quando um novo nó tem que ser adicionado. A política utilizada para descartar nós da memória cache é a LRU (*least recently used*). Toda vez que um nó é removido da memória cache imediatamente é gerada uma chamada SQL, pelo serviço de persistência, para inserir ou atualizar o referido nó. Dessa forma, mantém-se a consistência da memória cache com a fonte de dados.

Para realizar o mapeamento do modelo de objetos para o modelo relacional foi utilizado o padrão estático do *Crossing Chasms*. Aplicando-se o padrão estático, foi criada uma tabela no banco de dados, denominada *ASTDBStorageCell*, como mostra a Figura 8. Cada tupla dessa tabela armazena um nó da AST, que na memória cache corresponde a um objeto da classe `ASTDBStorageCell`. As tuplas têm como chave primária o *oid*. As colunas *father, firstchild, lastchild, rbrother e lbrother* são referências a outras tuplas da tabela *ASTDBStorageCell*, *symbol* é uma *string* que armazena o símbolo encapsulado por um nó da AST. As colunas *varname* e *vartype* são atributos da classe `Var`. Esses atributos são utilizados no reconhecimento e aplicação de padrões durante o processo de transformação. O padrão dinâmico do *Crossing Chasms* foi aplicado para criar uma classe *broker* responsável pela leitura e escrita dos nós das ASTs no banco de dados. Neste protótipo foi utilizado um *driver* JDBC do tipo 1 (*JDBC-ODBC bridge)* [5] para acessar o SGBD *Sybase Anywhere 5.0.*



ASTDBStorageCell	
@oid	integer
father	integer
brother	integer
lbrother	integer
lastchild	integer
firstchild	integer
symbol	varchar
varname	varchar
vartype	varchar

FIGURA 8 - MAPEAMENTO DO MODELO DE OBJETOS PARA O MODELO RELACIONAL.

O protótipo foi testado com uma AST de aproximadamente 17 mil nós de um programa *clipper*. Esta AST gerou 17 mil tuplas no banco de dados. Um computador pessoal *Pentium* 133 Mhz, 16 Mb de memória RAM com 28 processos sendo executados no sistema operacional Windows NT 4.0, foi utilizado nos testes. O protótipo levou 4 minutos para concluir o processo de transformação. Na mesma plataforma, sem persistir os nós em banco de dados, não foi possível realizar o processo de transformação por falta de memória principal.

5 Técnica para Análise Semântica

As informações sobre análise semântica são essenciais durante o processo de transformação de programas. Quando os sistemas de transformação são utilizados em reengenharia e "porte" de programas, a análise semântica permite avaliar a viabilidade do mapeamento de programas entre linguagens que seguem paradigmas distintos.

Na área de compiladores [1] [2], essas informações são geralmente utilizadas no teste e resolução de tipos e, também, na otimização de código. A técnica proposta consiste em fornecer ao SpinOff uma estrutura de representação híbrida, onde o formato básico são ASTs, que são transformadas em grafos específicos por demanda, quando um determinado tipo de tarefa transformacional melhor for aplicado sobre este modelo de representação. É o caso das tarefas de reengenharia e "porte" de programas.

Este trabalho fundamenta-se nas idéias de reescrita em grafos, a partir da descrição abstrata da semântica de uma linguagem. Adotamos uma abordagem para o uso de técnicas modernas, utilizadas pela comunidade de desenvolvedores de compiladores, tais como: o modelo *Static Single Assignment* (SSA) [18], os grafos de fluxo de controle (GFCs), os grafos de fluxo de dados (GFDs), os grafos de dependência de dados (GDDs) ,e similares.

Pode-se adicionar informações sobre o fluxo de controle no conjunto de blocos básicos, que constituem um programa, através da construção de um GFC. Os GDD e GFD são usados para representar informações sobre nomes e tipos de variáveis.

O modelo SSA, durante os últimos anos, tem sido utilizado como uma adequada representação de programa intermediário, que permite otimizações poderosas. A propriedade essencial do modelo SSA é que há somente uma declaração para cada variável em todo programa. Dessa forma, se duas variáveis têm o mesmo nome, estas também possuem o mesmo valor. Os algoritmos conhecidos geram o modelo SSA, a partir de programas com um fluxo de controle arbitrário.

A descrição sintática de uma linguagem é representada por ASTs, da qual pode-se derivar um grafo de informação semântica correspondente. Após a atividade específica de transformação ter sido realizada, a estrutura adaptativa transforma a representação novamente em ASTs, para que seja dada continuidade a outras tarefas. Deste modo, diversas formas híbridas, porém equivalentes, de representação são utilizadas ao longo de um processo transformacional.

Este trabalho iniciou-se com a definição e construção de uma biblioteca em Java para a manipulação de grafos. A biblioteca está sendo construída através de pacotes Java, que implementam as seguintes funcionalidades: teoria de grafos, desenho e *design* de grafos. A classe base contém métodos baseados em teoria dos grafos, tais como vértices, lados, percursos. Esse pacote possui classes especialistas que herdam características da classe base e implementam métodos específicos para os GFC, GFD, os GDD e grafos SSA.

Para validar as idéias desta técnica está sendo utilizado, em uma primeira fase, um estudo de caso baseado no "porte" de programas Clipper para Java [17] [20]. Este "porte" foi realizado no sistema de transformação Draco-PUC, onde o modelo de representação são ASTs associadas a estruturas de dados auxiliares, tais como tabelas de símbolos e bases de conhecimento, que armazenam informações semânticas destas linguagens.

No "porte" de programas Clipper, foi construído um transformador que, a partir da descrição sintática da linguagem Clipper, realiza uma análise estática, e armazena as informações semânticas de verificação de tipos, em uma base de conhecimento. Dessa forma, a partir de ASTs e da base de conhecimento, foi possível derivar GDDs das informações semânticas.

Pretende-se agora, gerar automaticamente os demais tipos de grafos específicos utilizando outros estudos de caso, e a partir dos resultados da técnica antiga, será possível realizar comparações com os resultados alcançados com a técnica proposta.

6 Conclusões e Trabalhos Futuros

Este artigo apresentou as diversas camadas de serviços que compõem a infra-estrutura básica de transformação necessárias para a manipulação de CTs, a linguagem de configuração de software Scil para operacionalizar as idéias de CTs permitindo tornar realizável os conceitos de prototipação dinâmica. Foram apresentados também, alguns exemplos de descrição de componentes utilizando a linguagem Scil, e os dois trabalhos que vêm sendo desenvolvidos na camada de manipulação básica do protótipo SpinOff, o modelo de memória virtual e a técnica para análise semântica.

Os testes realizados com o SpinOff, utilizando o modelo de memória virtual, apresentaram como principal resultado a solução do problema de esgotamento de memória principal durante o processo de transformação. Com a solução desse problema os STs podem ser utilizados em plataformas de hardware comuns, limitados apenas pelo espaço disponível em memória secundária. Atualmente, as plataformas de hardware necessárias para STs que manipulam programas maiores de 400 K linhas de código necessitam de quantidades de memória principal da ordem de *gigabytes*. No modelo de memória virtual os trabalhos futuros estão voltados para a implementação de um subsistema de persistência que suporta *lazy materialization* [3] para materializar os nós das ASTs somente quando necessários. Estudos para persistência de grafos também estão sendo realizados.

Como trabalho futuro da técnica para análise semântica, tem-se a escolha de um modelo abstrato para a descrição da semântica de linguagens e a construção de microcomponentes no SpinOff que manipularão grafos para realizar a aplicação de transformações. A integração dessa técnica ao ambiente SpinOff representa uma importante combinação de tecnologias. Esta combinação flexibiliza o uso de circuitos de transformação principalmente em atividades altamente dependentes de informações semânticas, como é o caso de tarefas de "porte" e reengenharia de sistemas.

Referências Bibliográficas

[1] A. V. Aho, R. Sethi, J. D. Ullman. *Compilers, Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA, 1986.

[2] A. W. Appel. *Modern Compiler Implementation in Java*. Cambridge University Press, 1998.

[3] C. Larman. *Design for Lazy Materialization*. Java Report, pp 45-54, april 1998.

[4] E. Gamma, R. Helm, R.Johnson, J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.

[5] G. Hamilton, R. Cattel, M. Fisher. *JDBC™ Database Access with Java™: A Tutorial and Annotated Reference*. Addison Wesley Longman, 1998

[6] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, M. Bier, *Clone Detection Using Abstract Syntax Trees*. In Proc. International Conference Software Maintenance, 1998.

[7] I. D. Baxter, C. Pidgeon. *Software Change Through Design Maintenance*. Procedings of International Conference Software Maintenance, 1997.

[8] J. C. S. do Prado Leite, M. Sant'Anna, and F. G. de Freitas. Draco-PUC: a Technology Assembly for Domain Oriented Software Development. In W. B. Frakes, editor, *3rd International Conference on Software Reusability*, Rio de Janeiro, Brazil, November 1994. IEEE Press, pp 102-109.

[9] J. C. S. do Prado Leite, M. Sant'Anna, and A. F. do Prado. Porting COBOL Programs Using a Transformational Approach. In Ned Chapin, editor, *Software Maintenance: Research and Practice, vol. 9, 3-31*, 1994. John Wiley & Sons.

[10] J. M. Neighbors. "*The Draco Approach to Constructing Software from Reusable Components*". *IEEE Transactions on Software Engineering*, SE-10(5):564-574, September 1984.

[11] J. M. Neighbors. The Benefits of Generators for Reuse. In M. Sitaraman, editor, *4th International Conference on Software Reusability*, page 218, Orlando, Florida, April 1996. IEEE Press.

[12] K. Brown, B. Whitenack. *"Crossing Chasms: A Pattern Language for Object-Relational Integration"*, in Pattern Languages of Program Design 2, Vlissides, Coplien and Kerth, eds., Addison-Wesley, 1996.

[13] M. Grand. *Patterns in Java, A Catalog of Reusable Design Patterns Illustrated with UML*. Wiley, 1998.

[14] M. Sant'Anna and J. C. S. do Prado Leite. *An Architectural Framework for Software Transformation.* In Proceedings of International Workshop on Software Transformation Systems, Los Angeles, California, USA, 33-38 pp. 1999.

[15] M. Sant'Anna, *Circuitos Transformacionais.* Tese de Doutorado. Pontificia Universidade Católica do Rio de Janeiro. RJ, Brasil, 1999.

[16] M. Sant'Anna, J. C. S. do Prado Leite and A. F. do Prado. *Draco-PUC: a Workbench for Developing Transformation-Based Software Generators.* In 20th International Conference on Software Engineering, Kyoto, Japan, April 1998.

[17] R. A. D. Penteado, P. C. Masiero , A. F. Prado, R. T. V. Braga, *"Reengineering of Legacy Systems Based on Transformation Using the Oriented Object Paradigm"* in Proceedings of 5th IEEE Working Conference on Reverse Engineering, Honolulu, Hawaii - USA, 1998, pp. 144-153.

[18] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman. *Efficiently Computing Static Single Assignment Form and the Control Dependence Graph.* TOPLAS 13(4), 451-490 pp. 1991.

[19] Reasoning Systems. *REFINE Users's Guide.* Reasoning Systems Inc., Palo Alto, CA, 1992.

[20] S. M. Abrahão, A. F. Prado, *"Web-Enabling Legacy Systems Through Software Transformations"* In Proc. IEEE International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems, Santa Clara, USA, April 1999, pp.149-152.