2
Sistema Transformacional Draco-PUC

O paradigma Draco [46] pode ser caracterizado por diversos pontos de vista [47]: originalmente, foi desenvolvido para prover um método para construir software a partir de componentes preexistentes; pode ser utilizado na construção de geradores de software capazes de gerar e manter um conjunto de sistemas similares; e finalmente como um conjunto de linguagens de domínios utilizadas no processo de construção de sistemas, através de um mecanismo transformacional capaz de gerar um programa executável a partir de uma especificação descrita em uma linguagem específica a um domínio (DSL – Domain Specific Language).

Diferentemente da maneira tradicional de obtenção de reuso, baseada em biblioteca de componentes, no paradigma Draco os elementos reutilizáveis, chamados de domínio, são definidos por linguagens formais. Estas linguagens são construídas com o objetivo de encapsular objetos e operações de um determinado domínio. Os programas escritos nas linguagens de domínio necessitam de um processo de refinamento para que possa ser obtido o programa executável que atende as especificações constantes do programa original.

A Máquina Draco-PUC é um sistema de software que implementa parcialmente o paradigma Draco. Sua utilização é feita através da construção de domínios descritos pela sua linguagem específica. Um domínio Draco-PUC precisa especificar sua sintaxe, descrita através da sua gramática, e sua semântica, expressa por transformações entre domínios. A figura 1 mostra um exemplo da construção do domínio CRC [48] (Responsabilidades e Colaborações de Classes). A especificação léxica e sintática é feita no arquivo *crc.grm*, utilizando os domínios GRM, que especifica a meta-linguagem utilizada para a especificação de domínios Draco-PUC, e PPD, que especifica a meta-linguagem de visualização ou pretty-printing de programas. A semântica do domínio CRC é dada através de transformações do domínio CRC para o domínio Java. Estas transformações são descritas usando o domínio TFM, que especifica a meta-linguagem de

transformações entre domínios Draco-PUC. Uma transformação é composta basicamente pelo padrão de reconhecimento (LHS- left hand side) e pelo padrão de substituição (RHS – right hand side): quando é identificada uma ocorrência do padrão de reconhecimento no programa original, esta ocorrência é substituída pelo padrão de substituição. O arquivo *crc2java.tfm* da figura 1 mostra um exemplo da especificação da semântica do domínio CRC: a transformação *CreateClass* identifica cada classe no cartão de CRC (o LHS) e cria o código Java (o RHS) correspondente; a transformação *createMethod* cria a definição de um método em Java para cada responsabilidade da classe no cartão CRC.

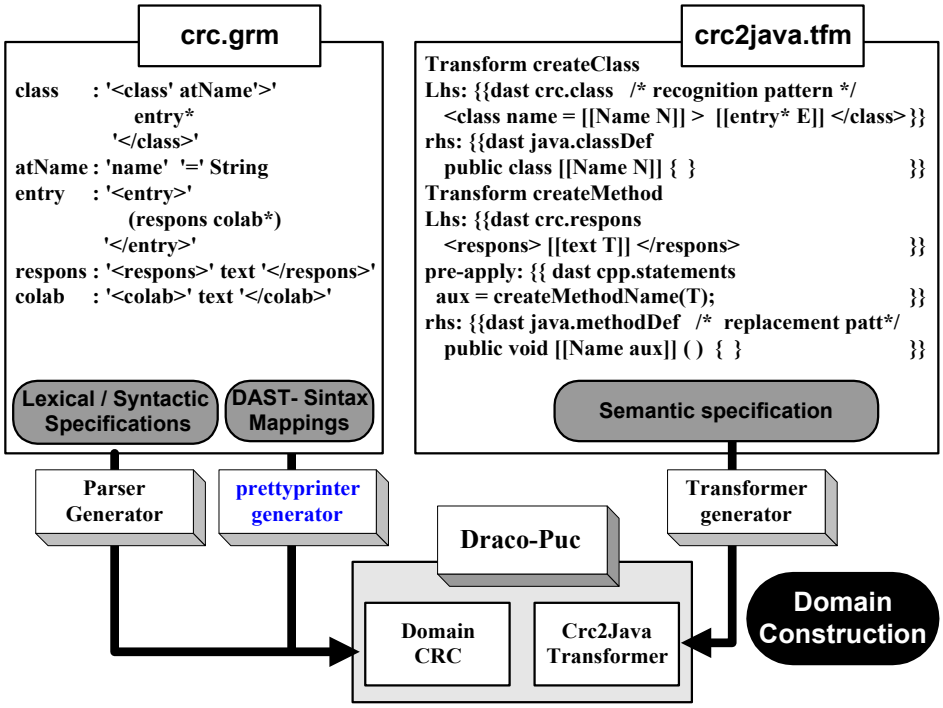

Figura 1 - Exemplo de Construção de Domínio na Máquina Draco-PUC

Além dos padrões de reconhecimento e substituição de uma transformação, existem 5 possíveis pontos de controle onde podem ser adicionados códigos. O ponto *Pre-Match* é ativado cada vez que o LHS é testado sobre o programa original. O ponto *Match-Constraint* é ativado quando ocorre um casamento entre uma variável do LHS com alguma parte do programa original. *Post-Match* é ativado após um casamento entre o programa e o LHS. *Pre-Apply* é ativado imediatamente antes da ocorrência do LHS no programa ser substituída pelo RHS. Finalmente, o ponto *Post-Apply* é ativado após esta ocorrência ter sido substituída pelo RHS.

As transformações podem ser agrupadas com o objetivo de facilitar o desenvolvimento e entendimento de um sistema de transformações, permitindo

que se varie a granularidade do sistema de transformações em desenvolvimento. A estrutura disponível na Máquina Draco-PUC permite o agrupamento de várias transformações em um conjunto de transformações (*Set of Transforms*). De maneira similar, conjuntos são agrupados formando um transformador (*Transformer*), que é o elemento básico a ser utilizado quando do acionamento da Máquina Draco-PUC para transformar um programa. De maneira similar às transformações, os conjuntos de transformações e os transformadores também possuem pontos de controle que são acionados quando do inicio (*Initialization*) e término da execução (*end*) de cada um.

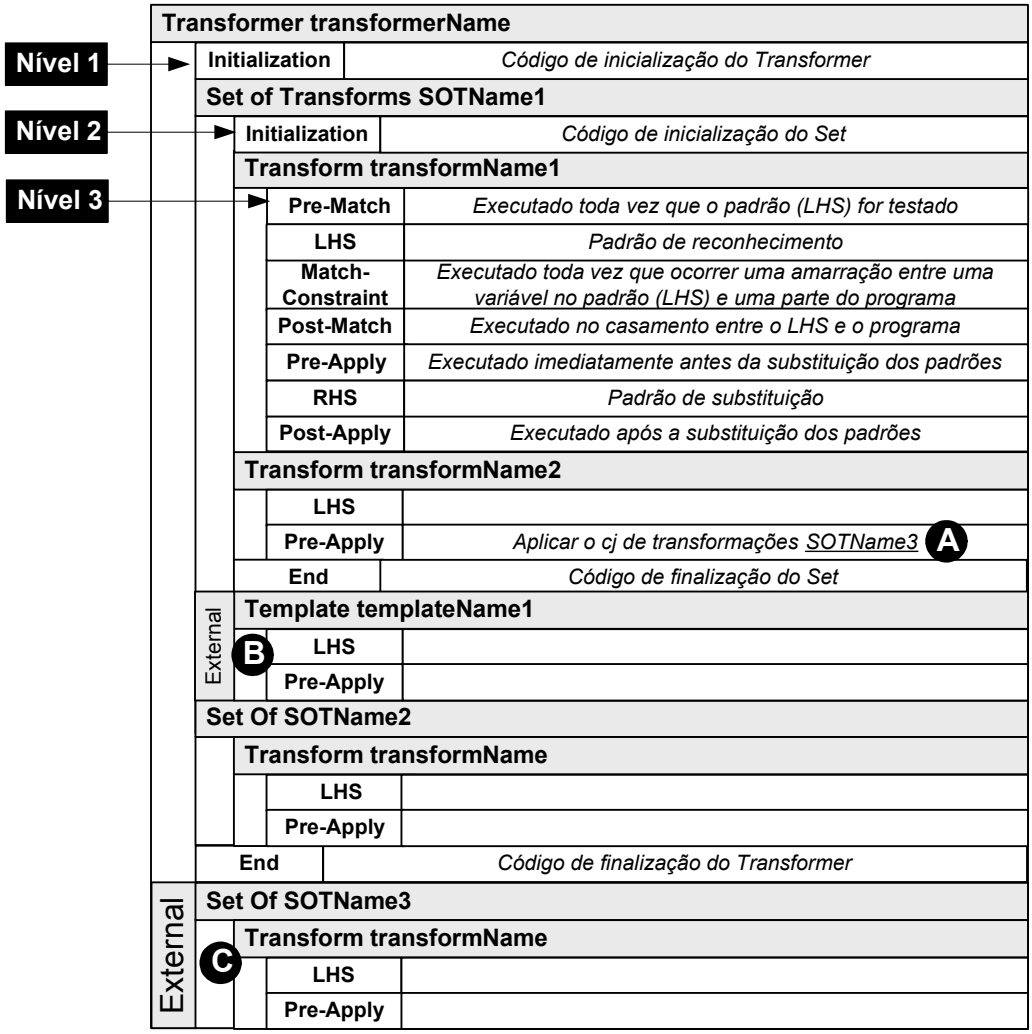

Figura 2 – Notação utilizada na descrição das transformações

A notação utilizada nesta tese para descrever as transformações é apresentada na figura 2. Esta notação é baseada em blocos funcionais organizados em níveis. O primeiro nível contém os elementos do transformador, ou seja, os pontos de controle de início e término, bem como seus conjuntos de transformações. Na ativação do transformador, os pontos de controle e os

conjuntos são ativados na ordem em que aparecem no modelo. Os conjuntos definidos como *External* (como os marcados com as letras B e C na figura 2) somente são ativados por outros conjuntos, ou seja, não são automaticamente aplicados ao programa. Um exemplo de ativação de um conjunto *External* é mostrado na marca A da figura, onde o conjunto *SOTName3* é ativado no *Pré-Apply* da transformação *transformName2*.

O segundo nível contém os elementos dos conjuntos: suas transformações, pontos de controle e ainda *Templates*, que são transformações que só possuem o lado esquerdo (LHS) ou direito da regra (RHS) e só podem ser acionados através de chamadas explícitas no código, sendo por este motivo igualmente marcadas como *External*. Por fim, o nível três apresenta os elementos das transformações que são ativados de acordo com suas regras próprias apresentadas anteriormente.

Uma visão geral do uso da Máquina Draco-PUC é mostrada na figura 3. O parser do domínio do programa fonte é utilizado pela Máquina Draco-PUC para converter o programa na forma interna utilizada pela máquina (DAST – *Draco Abstract Syntax Tree*). Transformações podem ser aplicadas para modificar a DAST. Estas transformações podem ser intra-domínio, caso em que a DAST modificada pertence ao mesmo domínio original, ou inter-domínio*, quando a DAST modificada pertence a um novo domínio. A qualquer momento pode ser utilizado o *pretty-printer* correspondente para realizar o *unparse* de uma DAST e obter sua descrição na linguagem de domínio correspondente.

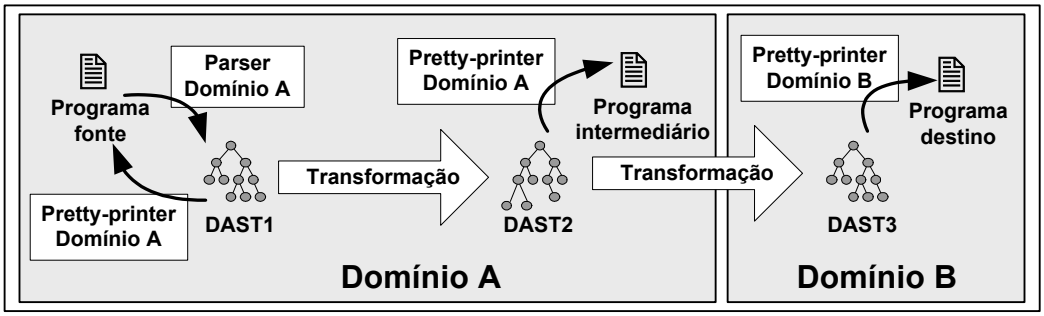

Figura 3 - Visão Geral do Uso da Máquina Draco-PUC

De uma maneira geral, podemos dizer que a Máquina Draco-PUC aplica um conjunto de transformações em um programa fonte de maneira a obter um

* Na definição original do paradigma Draco [46], as transformações inter-domínios, realizadas de um domínio mais abstrato para um domínio mais concreto, eram chamadas de refinamentos e correspondiam a implementação dos componentes reutilizáveis do paradigma.

programa de destino. Os programas fonte e destino podem pertencer a qualquer domínio, inclusive ao próprio domínio de definição de transformações, ou seja, podemos aplicar transformações que irão gerar novas transformações a serem aplicadas em outros programas. Esta flexibilidade inerente ao paradigma transformacional cria uma série de oportunidades de uso da Máquina Draco-PUC em função do significado dos programas fonte e destino utilizados ou gerados, que podem ser agrupados para formar alguns padrões de utilização. Os principais padrões de utilização identificados nesta tese são mostrados na figura 4. Estes padrões podem ser utilizados de maneira isolada ou combinados entre si.

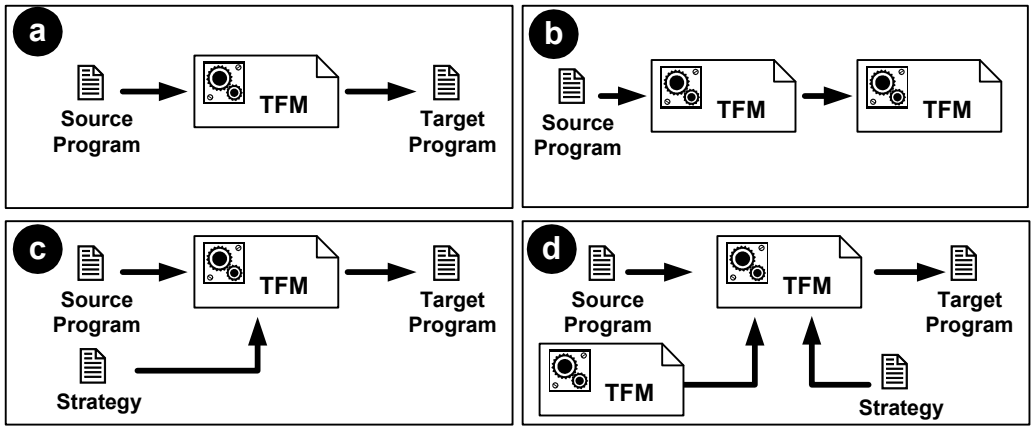

Figura 4 – Padrões de utilização da Máquina Draco-PUC

O padrão tradicional (figura 4a) permite a aplicação de transformações sobre um programa fonte com o intuito de gerar um novo programa. O programa destino pode pertencer ao mesmo domínio do programa original, resultado da aplicação de transformações intra-domínio, ou a um novo domínio, como conseqüência da aplicação de transformações inter-domínios. Como exemplo clássico da utilização deste padrão temos o porte do código fonte de programas para uma nova linguagem, tendo sido utilizado nas transformações de COBOL para C++[49], de Estelle para C++[50] e de Clipper para Java[51].

Um caso particular do padrão tradicional é quando o programa de destino é uma transformação, ou seja, transformações que aplicadas sobre um programa fonte irão gerar outras transformações (figura 4b). Esta forma de utilização possibilita que se alcance uma maior produtividade através da utilização da própria máquina para gerar seus componentes. Este padrão foi utilizado na construção de ferramentas para engenharia reversa [52] onde, a partir da definição das extrações que se desejam aplicar, descritas em uma linguagem específica ao domínio de extração, são gerados conjuntos de transformações, chamados de

transformações de extração, que efetivamente extraem as informações especificadas no programa e as armazenam em uma base de dados. Estas transformações de extração se constituem no núcleo de uma ferramenta de engenharia reversa gerada automaticamente pela máquina Draco-PUC para a situação definida no programa original.

Este padrão foi estendido no presente trabalho ao se realizar a geração de novas transformações que podem ser utilizadas de maneira dinâmica, ou seja, sem a necessidade de que se realize um processo de geração de código intermediário e subsequente compilação (ou transformação) para cada nova transformação gerada. Esta extensão foi necessária para permitir a utilização de transformações que representem as diferenças identificadas entre duas versões de um artefato e é apresentado em mais detalhes na seção 4.2.2.

Transformer AplicarEstrategia				
	Set of Transforms ConjuntoPrincipal			
		Transform TrataDeclaraçãoA		
			LHS	declaracao A ;
			Pre-Apply	*1) ler o módulo programaFonte* *2) Aplicar o cj de transformações CJA em programaFonte*
		Transform TrataDeclaraçãoB		
			LHS	declaracao B ;
			Pre-Apply	*1) ler o módulo programaFonte* *2) Aplicar o cj de transformações CJA em programaFonte* *3) Aplicar o cj de transformações CJB em programaFonte*
External	**Set of Transforms CJA**			
		Transform		
			LHS	
			RHS	
	Set of Transforms CJB			
		Transform		
			LHS	
			RHS	

Figura 5 – Exemplo de transformação dirigida por estratégia

O padrão apresentado na figura 4c representa a utilização de um programa que define a estratégia a ser aplicada pelas transformações sobre o programa original. A estratégia pode ser descrita utilizando a linguagem de qualquer domínio Draco-PUC e define um conjunto de regras que orientam a aplicação das transformações. A figura 5 exemplifica esta forma de utilização. As transformações do conjunto *ConjuntoPrincipal* reconhecem as declarações do programa de estratégia e aplicam conjuntos de transformações auxiliares sobre o programa fonte. Por exemplo, se existir uma *declaração B* no arquivo de

estratégia, a transformação *TrataDeclaração B* é ativada e aplica os conjuntos de transformações *CJA* e *CJB* no programa fonte. Este padrão foi utilizado na implementação da técnica de reconhecimento de planos (seção 4.2.3), onde os fatos observados constituem a estratégia e o programa fonte é formado pela biblioteca dos possíveis planos a serem inferidos. A aplicação das transformações lê cada fato observado e utiliza as definições existentes na biblioteca de planos para construir os planos que os explicam.

No último padrão (figura 4d), um programa contendo a definição de uma estratégia orienta a transformação principal na escolha de quais transformações aplicar sobre o programa fonte, definindo ou restringindo as combinações possíveis. Neste trabalho este padrão foi utilizado, em complemento ao padrão da figura 4b, na construção do mecanismo de rastreamento baseado em transformações, permitindo que se realize a evolução de uma versão de um artefato para sua versão consecutiva através da aplicação de um conjunto de transformações anteriormente gerado a partir das diferenças existentes entre as duas versões. A seção 4.3 apresenta em detalhes a utilização deste padrão.

2.1. Rede de Domínios

Uma Rede de Domínios (DN – Domain Network) [53][54][55] é um conjunto de domínios interconectados através de transformações. Cada domínio encapsula o conhecimento específico a uma área e é descrito por uma linguagem especifica do domínio.

Neighbors [55] propõem uma classificação baseada no grau de abstração dos domínios, mostrada na figura 6, classificado-os em: Domínios de Aplicação, que expressam a semântica dos domínios reais, ou seja, descrevem o mundo real no qual os sistemas de software serão inseridos; Domínios de Modelagem, que são os domínios que encapsulam conceitos de forma a fornecer um alto nível de reuso e facilitar a implementação dos domínios de aplicação; e os Domínios Executáveis, que são aqueles domínios que podem ser diretamente implementados em uma plataforma computacional.

Os programas escritos utilizando a linguagem de um domínio de aplicação necessitam de um processo de refinamento para que, passando por uma série de domínios intermediários, possam alcançar um domínio executável, como mostra a

figura 7. Este processo de refinamento é realizado através de transformações que, quando aplicadas em um programa descrito na linguagem de um domínio fonte, criam a descrição correspondente na linguagem do domínio de destino.

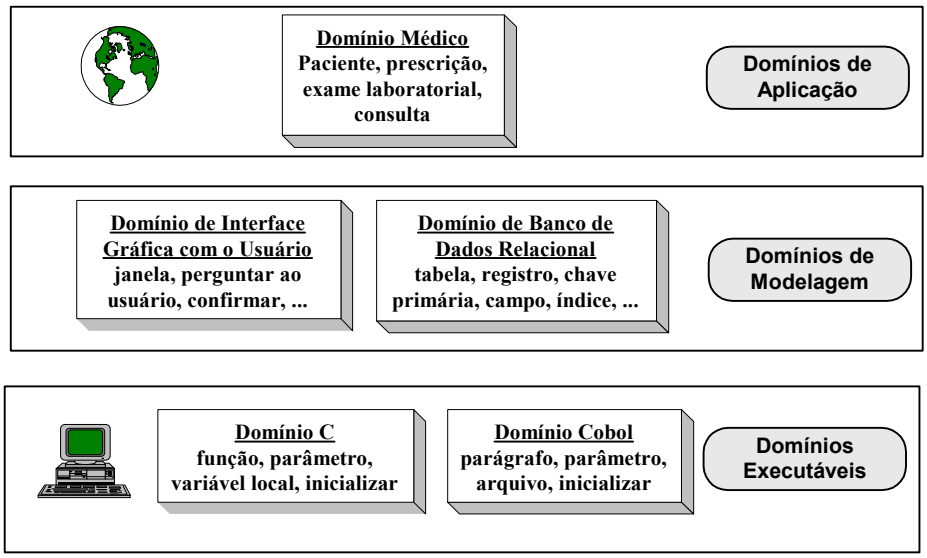

Figura 6- Classificação dos Domínios de Acordo com o Nível de Abstração

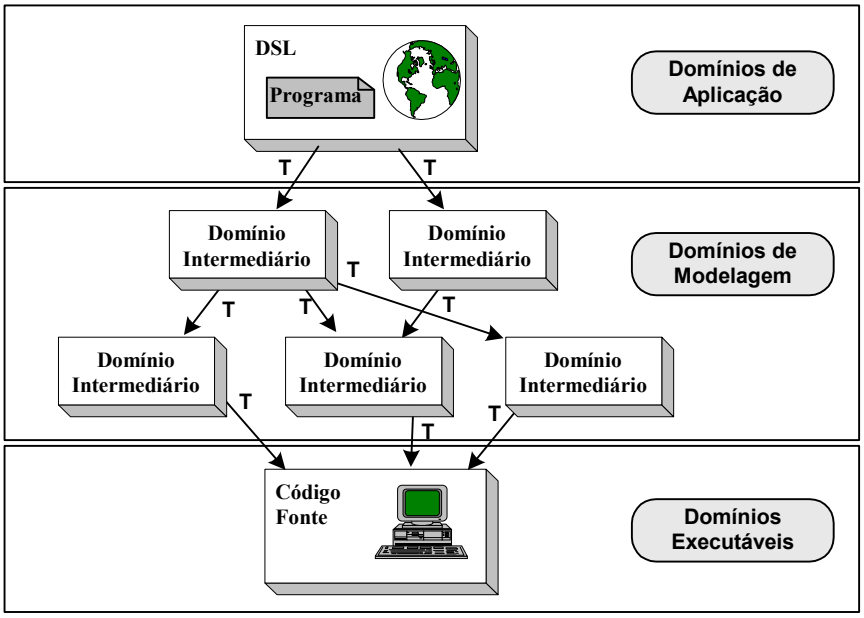

Figura 7 – Rede de Domínios

O uso de Redes de Domínios na geração de software apresenta duas vantagens principais. Primeiro, enquanto os geradores de software tradicionais utilizam apenas um passo de refinamento, um gerador baseado em Rede de Domínios permite que se alcance um maior nível de reuso através da utilização de diversos passos de refinamento. Segundo, a implementação de um novo domínio é

realizada de maneira mais fácil pois basta criar as transformações para outros domínios já existentes.

No restante deste capítulo, serão apresentadas três técnicas para implementar Redes de Domínios: baseada na Máquina Draco-PUC, baseada na tecnologia XSL e uma abordagem combinada Draco-PUC e XSL.

2.1.1. Implementação de Rede de Domínios usando a Máquina Draco-PUC

Uma Rede de Domínios tradicional tem sua origem em um domínio de aplicação e pode envolver diversos outros domínios de aplicação e de modelagem até alcançar os domínios executáveis correspondentes. Um exemplo parcial de uma Rede de Domínios implementada na Máquina Draco-PUC é apresentado na figura 8. Esta rede tem sua origem no programa *prog.rdoo* descrito no domínio RDOO* e, após a aplicação de sucessivas transformações, é obtido o programa correspondente no domínio executável da linguagem C. O programa *prog.rdoo* define que se deseja extrair as informações de dependências entre módulos (linha 5) a partir de programas escritos na linguagem C++ (linha 2). O primeiro transformador (*rdoo2exl*) cria o programa *prog.exl*, correspondente a representação do programa original no domínio de modelagem EXL†. Este programa é composto pela definição das tabelas da base de dados que irão armazenar as informações extraídas (linhas 01 a 04) e pela transformação de extração propriamente dita (linhas 100 a 113). Em seguida é aplicado o transformador *exl2c* que reconhece as definições das tabelas no programa *prog.exl* e cria o programa *prog.c*, que define as estruturas de dados e o esquema da base de dados a ser utilizada. A transformação de extração existente no programa *prog.exl* é reconhecida pelo transformador *exl2tfm*, que gera sua representação na linguagem utilizada na Máquina Draco-Puc para definição de transformações. Finalizando o processo de refinamento, é aplicado o transformador *tfm2cpp* que gera o programa em C++ para estas transformações.

---

* Domínio para especificação de ferramentas para extração de informações de programas orientados a objetos.

† Domínio para especificação de programas para extração e visualização de informações de programas

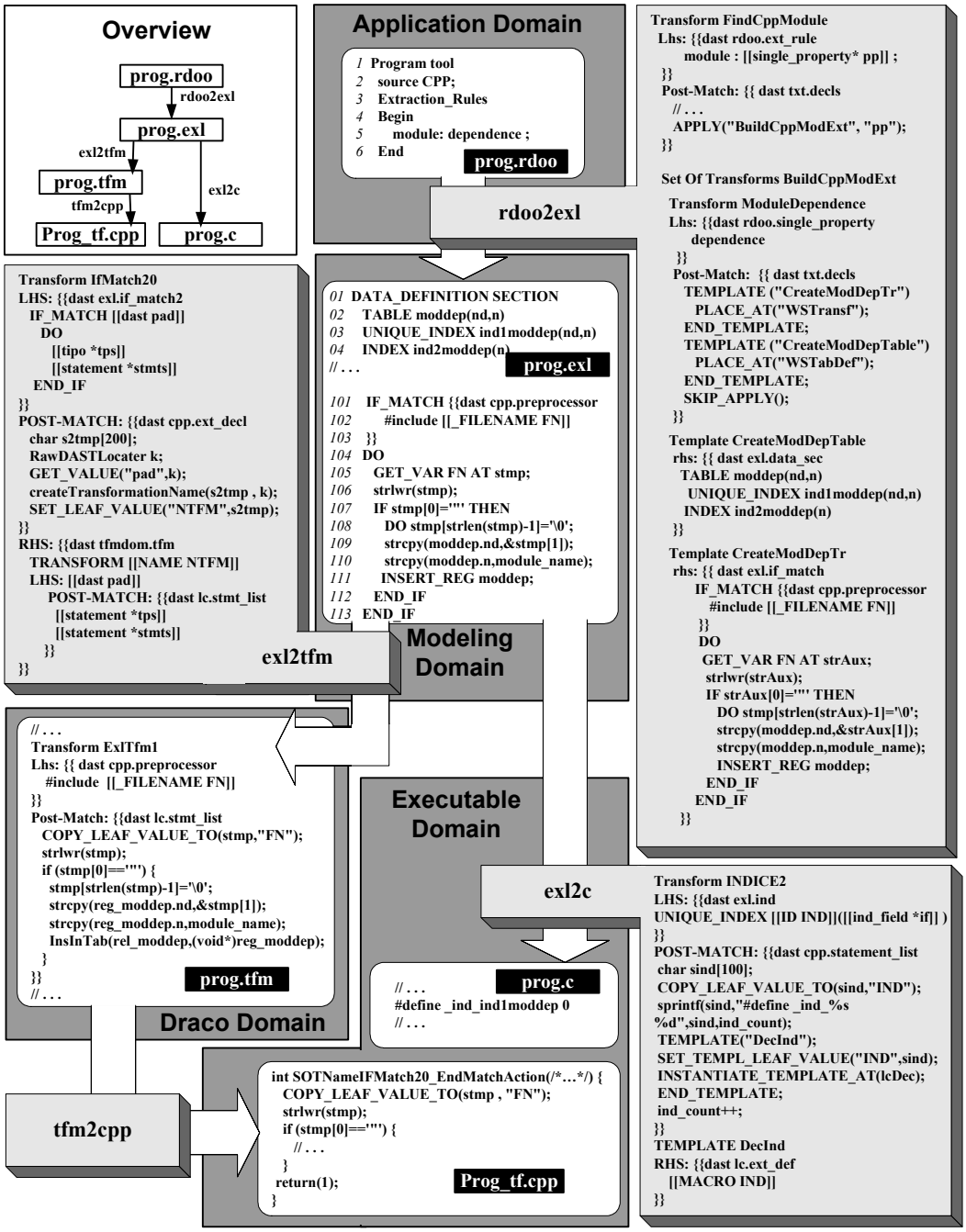

Figura 8 - Visão Parcial de uma Rede de Domínios e suas Conexões

Atualmente a Rede de Domínios Draco-PUC possui diversos domínios implementados. As figuras 9 e 10 apresentam esta rede com uma breve descrição de cada domínio.

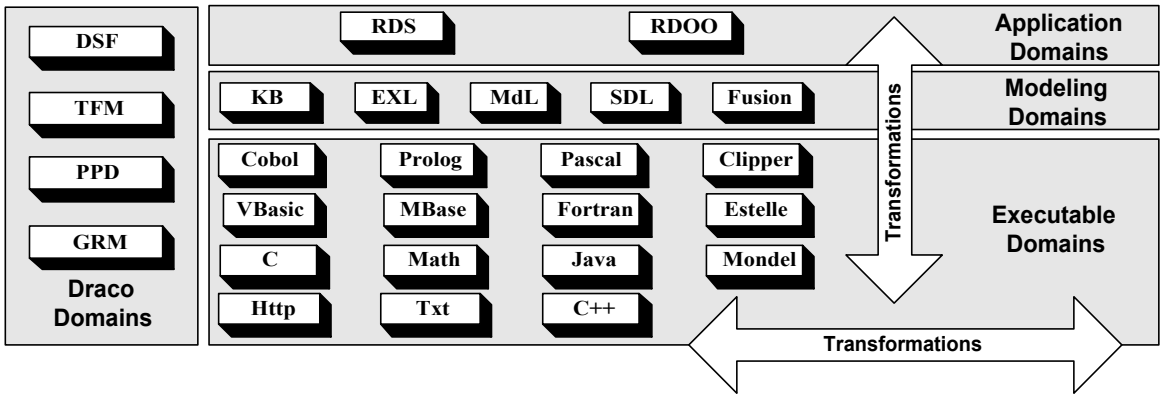

Figura 9 - Rede de Domínios Draco-PUC

Tipo	Domínio	Transformadores	Descrição
Aplicação	RDOO[53]	Rdoo2Exl	Recuperação de Design de Sistemas OO
	RDS[53]	Rds2Exl	Recuperação de Design de Sistemas Estruturados
Modelagem	KB[49]		Manipulação de Bases de Conhecimento
	EXL[56]	Exl2Cpp	Especificação de extração de informações de código fonte
	MdL[56]	Mdl2Cpp	Especificação de visualizações das informações extraídas do código fonte
	SDL		Especificação de interface gráfica com o usuário
	Fusion[57]		Descrição de modelos do Método Fusion
Executável	C++	Cpp2Java	Linguagem de Programação
	C	CStruct	
	Cobol	CobC	
	Clipper	Clipper2KB Clipper2Mdl Clipper2UmlJava	
	Txt		Arquivos textuais
	HTTP		Protocolo HTTP
	Estelle[58]	Est2Cpp	
	Mbase		Especificação de Base de Dados MBase
	Math	Derive	Definição de expressões matemáticas
	Java, Prolog, Pascal, Vbasic, Fortran, Mondel[59]		
Draco	GRM	Grm2Y	Especificação de linguagens de domínio (léxico e sintaxe)
	TFM[49]	Tfm2Cpp	Especificação de transformações
	PPD	Ppd2Txt	Especificação de formatação para o pretty-printer

Figura 10 - Descrição dos domínios Draco-PUC

2.1.2. Implementação de Rede de Domínios usando XSL

Durante a execução dos trabalhos desta tese identificamos uma outra técnica que poderia ser utilizada para implementar uma Rede de Domínios. Esta técnica

baseia-se na utilização de diversas recomendações do W3C* e das ferramentas disponíveis no mercado que as implementam. Cada domínio é definido como uma aplicação XML, tendo sua sintaxe definida através de uma DTD (Document Type Definition). Esta sintaxe não define uma DSL real mas sim a estrutura de um arquivo de marcação (arquivo XML) cujos elementos representam objetos e operações do domínio real. A semântica do domínio é expressa por um conjunto de transformações estruturais definidas usando folhas de estilo XSL (Extensible Stylesheet Language). Uma folha de estilo consiste de uma série de templates. Cada template utiliza instruções XPath na definição do padrão de reconhecimento a ser procurado por um processador XSL no programa original representado em um arquivo XML. Quando o padrão de reconhecimento for identificado em alguma parte do programa original as instruções no template são executadas gerando um novo arquivo XML. A figura 11 mostra a construção do domínio CRC[48] utilizando a tecnologia XSL. O arquivo *crc.dtd* define a estrutura da aplicação XML (a sintaxe do domínio CRC) utilizada para especificar cartões CRC e o arquivo *crc2java.xsl* contém o conjunto dos templates que geram o código Java para cada classe dos cartões CRC (a semântica do domínio CRC, expressa através de transformações para o domínio Java).

A figura 12 mostra a operacionalização de uma Rede de Domínios XSL. Um parser DOM† é utilizado na leitura do programa descrito no domínio original e na construção da árvore DOM correspondente. Um processador XSL é utilizado tanto na transformação da árvore do domínio original para a árvore do domínio de destino, quanto para o unparse de árvores DOM, gerando a descrição do programa na linguagem de domínio correspondente. A especificação do domínio de origem da rede é feita através dos arquivos *sourceDef.dtd*, que define sua sintaxe, e do arquivo *transf.xsl*, que contém sua semântica, expressa através de transformações para o domínio de destino. De maneira similar, o domínio de destino possui sua sintaxe definida através do arquivo *target.xml* e os comandos para o unparse da

* As recomendações técnicas estão disponíveis em http://www.w3c.org/TR

† Um parser DOM cria uma visão utilizando o modelo DOM (Document Object Model) de um documento XML. A especificação do modelo DOM pode ser encontrada em http://www.w3c.org/TR/REC-DOM-Level-1. O parser utilizado nesta implementação foi o Xerces do Projeto Apache XML, disponível em http://xml.apache.org

árvore DOM estão contidos no arquivo *view.xsl* .

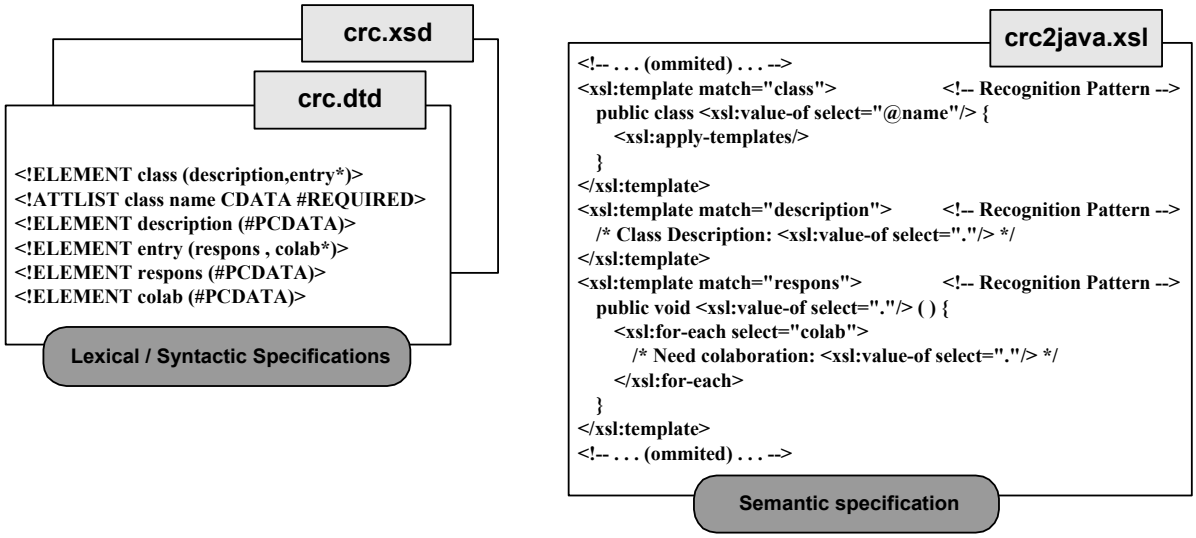

Figura 11 – Exemplo de construção de um domínio XSL

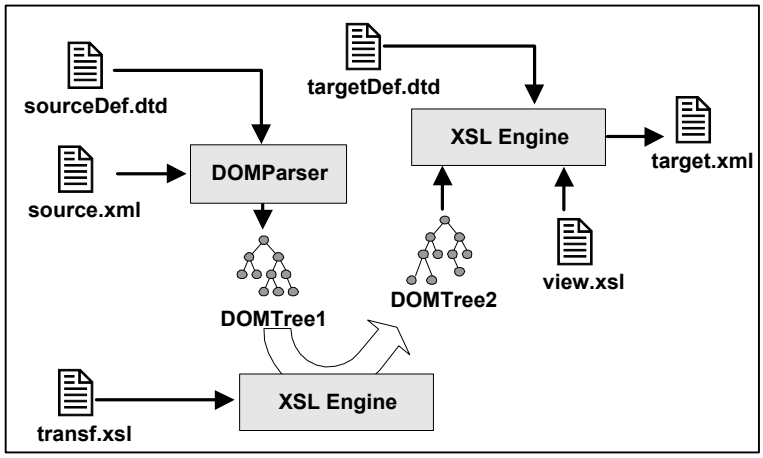

Figura 12 – Uso da Rede de Domínios XSL

2.1.3.
Rede de Domínios Integrada Draco-PUC – XSL

As técnicas anteriores de implementação de Redes de Domínio apresentam alguns aspectos que limitam sua maior utilização no processo de desenvolvimento de software. A implementação de Redes de Domínio utilizando a Máquina Draco-PUC disponibiliza um poderoso mecanismo transformacional mas não é amplamente difundido, tendo seu uso restrito a comunidade acadêmica. Em contrapartida, a implementação baseada em XSL utiliza uma série de tecnologias amplamente utilizadas pela indústria de software, mas não possui muita flexibilidade na definição de transformações. A figura 13 mostra uma comparação mais detalhada das duas formas de implementação.

	Draco-PUC	XSL
Linguagem do Domínio	- Permite a utilização do vocabulário do mundo real	- Utiliza elementos XML, ou seja, a definição é feita através da marcação do texto e não pela utilização direta do vocabulário do usuário.
Especificação da Sintaxe	- Utiliza BNF que é uma maneira mais flexível de definir a sintaxe.	- Pode utilizar DTD ou XML Schema. Todo documento é uma árvore.
Especificação da Semântica	- Pode utilizar a linguagem de qualquer domínio na especificação das transformações. - Possui diversos pontos de controle onde podem ser inseridos códigos executáveis: Pre-Match, Match-Constraint, Post-Match, Pre-Apply, lhs, rhs e Post-Apply	- Pode utilizar extensões para acessar código Java - Somente um local para inserir código executável no template, correspondente ao rhs
Utilização	- Uso restrito a um pequeno número de desenvolvedores. - Não é utilizado pela indústria	- Grande número de pesquisadores - Utilizado por várias aplicações reais

Figura 13 – Comparação das formas de implementação de Redes de Domínios

De maneira a eliminar os problemas mencionados, projetamos e implementamos uma Rede de Domínios que integra as duas alternativas apresentadas anteriormente através do estabelecimento de um mecanismo de importação de domínios entre eles. A figura 14 mostra a implementação deste mecanismo (*Gateway*), operacionalizado através de um conjunto de transformadores Draco-PUC, que permite que todo domínio A definido para uma Rede de Domínios XSL, tenha um domínio A' equivalente na Rede de Domínios Draco-PUC. A criação deste domínio A' é realizada de maneira automática pela aplicação do transformador *dtd2grm.tfm*, se a sintaxe do domínio A foi descrita utilizando DTD, ou do transformador *xsd2grm.tfm*, caso a sintaxe tenha sido descrita através de XML Schema. De maneira similar, a semântica do domínio A' é gerada pela aplicação do transformador *xsl2tfm.tfm* sobre a semântica especificada em XSL para o domínio A. Como resultado da utilização deste mecanismo de importação, podemos utilizar tanto a rede de domínios Draco-Puc, quanto a rede de domínios XSL para refinar o programa *source.xml* .

O transformador Dtd2Grm é apresentado na figura 15. O primeiro conjunto de transformações aplicado (*BuildKBForAttributeDefinition*) encontra todas as entradas do arquivo original que especificam os atributos dos elementos, identifica suas características e as insere em uma base de conhecimento (*KB*) que será posteriormente utilizada na geração das regras correspondentes aos atributos no domínio GRM da Máquina Draco-PUC. O segundo conjunto (*CreateGrm*) é o responsável pela efetiva geração das regras no domínio GRM. Toda vez que for localizada uma definição de elemento no arquivo de origem, é ativado o conjunto

CreateElementRule que insere a regra correspondente ao tipo do elemento definido na DTD (*EMPTY*, *ANY*, *#PCDATA*, *Mixed* ou *Elements*). As regras de definição dos atributos na DTD são identificadas pela transformação *FindAttribute* que consulta a KB para obter as informações dos atributos e gera a regra correspondente.

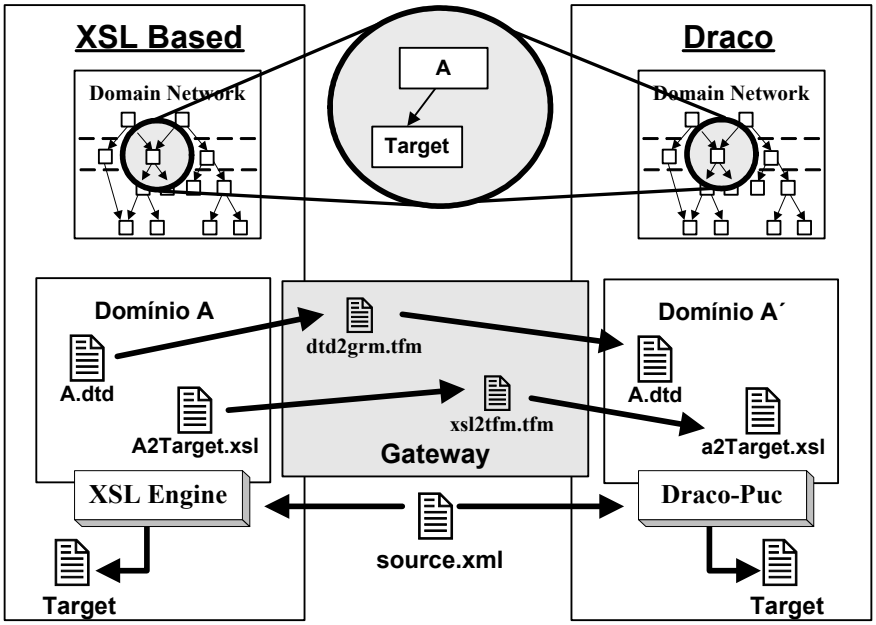

Figura 14 - Integração Draco-PUC – XSL

A figura 16 exemplifica a utilização do transformador Dtd2Grm: a sintaxe do domínio CRC (Class Responsibilities and Collaborations), originalmente definida para uma Rede de Domínios XSL utilizando DTD (arquivo *crc.dtd*), é transformada na sua definição equivalente para uma Rede de Domínios Draco-PUC (arquivo *crc.grm*) pela aplicação do transformador *dtd2grm.tfm*. De maneira similar, a semântica no domínio Draco-PUC pode ser gerada pela aplicação de um transformador de XSL para TFM sobre as folhas de estilo XSL.

A utilização deste mecanismo de importação de domínios permite que qualquer aplicação XML possa ser utilizada no contexto da Rede de Domínios Draco-PUC, ou seja, permite a utilização da Máquina Draco-PUC na aplicação de transformações entre programas descritos em XML e os diversos domínios existentes no Draco-PUC.

Transformer DTD2GRM			
	Set of Transforms BuildKBForAttributesInformation		
	Transform FindAttributeDefinition		
		LHS	<!ATTLIST [[Name en]] [[Name an]] [[attDef* AD]] >
		Pre-Apply	*Preenche a KB com as informações sobre os atributos dos elementos*
	Set Of Transforms CreateGrm		
	Transform FindElement		
		LHS	<!ELEMENT [[Name N]] [[elementDefinition ED]] >
		Pre-Apply	*Aplica o cj de transformações CreateElementRule*
	Transform FindAttribute		
		LHS	<!ATTLIST [[Name en]] [[Name an]] [[attDef* AD]] >
		Pre-Apply	*Consulta a KB para obter informações sobre os atributos* *Insere a regra correspondente em WSRules*
		End	*Cria o arquivo .GRM* *- Copia as regras default de início* *- Copia as regras de WSRules* *- Copia as regras default de fim* *Cria os arquivos de gerência (.bat, makefile, …)*
External	**Set of Transforms CreateElementRule**		
	Transform emptyElement		
		LHS	EMPTY
		Pre-Apply	*Cria a regra e insere em WSRules*
	Transform anyElement		
		LHS	ANY
		Pre-Apply	*Cria a regra e insere em WSRules*
	Transform pcdataElement		
		LHS	#PCDATA
		Pre-Apply	*Cria a regra e insere em WSRules*
	Transform mixedElement		
		LHS	[[mixed ME]]
		Pre-Apply	*Cria a regra e insere em WSRules*
	Transform elementsElement		
		LHS	[[element E]]
		Pre-Apply	*Cria a regra e insere em WSRules*

Figura 15 – Transformador Dtd2Grm

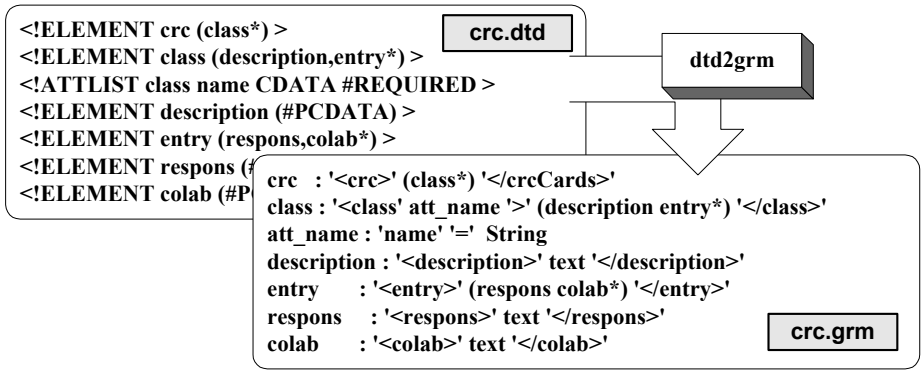

Figura 16 – Exemplo de geração da definição da sintaxe Draco-PUC

2.2. Ferramentas Desenvolvidas

Um problema encontrado durante o desenvolvimento dos diversos transformadores necessários a esta tese refere-se à dificuldade de análise dos resultados intermediários de sua aplicação, necessária para verificar a correção de cada elemento dos transformadores. Para solucionar este problema optamos nesta tese pela especificação e construção de um mecanismo, chamado de *DracoDebugger*, que facilitasse a instrumentação das transformações e a visualização dos seus resultados. O dracoDebugger foi implementado através da extensão do motor transformacional da Máquina Draco-PUC e da construção de uma ferramenta de visualização de transformações.

A utilização deste mecanismo é realizada em duas fases. Na primeira fase são gerados os passos de execução do transformador. Cada passo representa um conjunto de dados referentes ao contexto de uma transformação ou de um comando de instrumentação. O passo referente a uma transformação é gerado automaticamente pela Máquina Draco-PUC através do armazenamento das DASTs imediatamente antes e após a aplicação de cada transformação. Esta geração é definida através do uso dos seguintes comandos no arquivo de script (extensão .dsf) que aplica a transformação:

```
DracoDebbuger-on    // inicia a geração automática de dados das transformações
DracoDebbuger-all   // inicia a geração automática de dados das transformações.
                    // Inclui também a geração dos dados sobre as tentativas de
                    // aplicação da transformação, ou seja, cada tentativa de
                    // amarração do LHS
DracoDebbuger-off   // finaliza a geração automática para as transformações
```

Os comandos de instrumentação são apresentados a seguir e podem ser colocados nos pontos de controle de qualquer elemento (transformador, conjunto ou transformação).

```
DRACO_DEBUGGER( char *msg);        // Cria um novo passo com a mensagem
DRACO_DEBUGGER1( char *msg , RawDASTLocater aLoc );
      // Cria um novo passo com a mensagem e a DAST apontada por aLoc
DRACO_DEBUGGER2( char *msg , RawDASTLocater aLoc , RawDASTLocater aLoc1 );
      // Cria um novo passo com a mensagem e a DAST apontada por aLoc e aLoc1
```

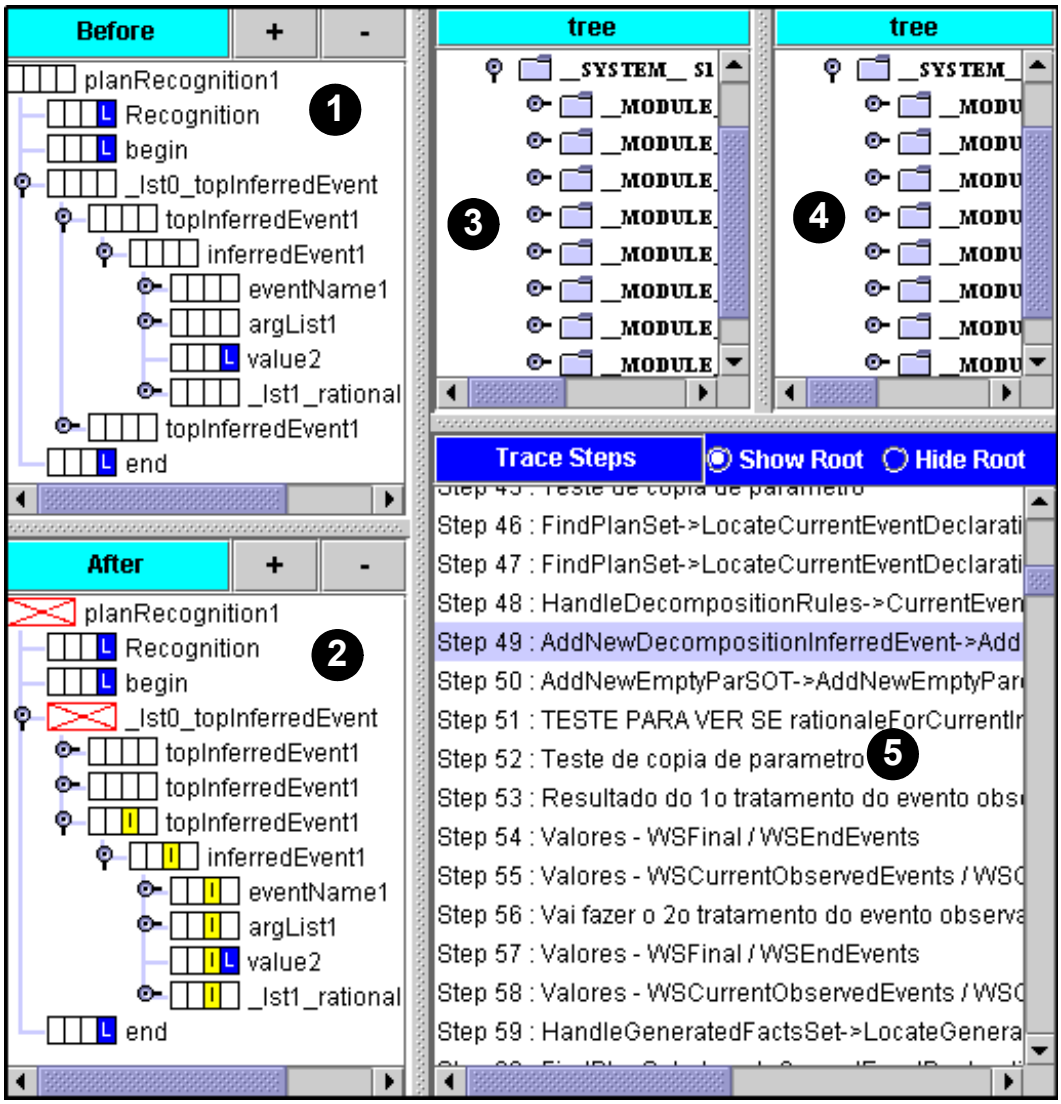

Figura 17 – Ferramenta DracoDebugger

A segunda fase envolve a utilização da ferramenta DracoDebugger na visualização do conjunto de dados gerados. A figura 17 mostra a tela da ferramenta. O painel 5 mostra os passos que foram gerados. A numeração dos passos é feita na ordem de execução e o nome do passo é formado pela mensagem definida pela instrumentação (marca 5 na figura) ou pela concatenação do nome do conjunto e o nome da transformação aplicada (passo selecionado na figura). Os painéis 1 e 2 mostram, para o passo selecionado, a DAST antes e depois da aplicação da transformação, se gerados automaticamente, ou o primeiro e segundo parâmetros do comando de instrumentação. As diferenças existentes entre estas duas DASTs foram obtidas a partir da aplicação do algoritmo *TreeDiff*, descrito na seção 4.2.1.1, e são apresentadas através de marcas nos nós. Os painéis 3 e 4 mostram todo o conjunto de DASTs existentes correspondentes aos painéis 1 e 2 respectivamente.

2.3. Conclusão

Sant´Anna [60] apresenta em sua tese uma análise do uso da Máquina Draco-PUC, apresentando diversas experiências de uso da máquina conduzidas por alunos e pesquisadores da PUC-Rio, IME, UFScar e USP, bem como os problemas existentes com o protótipo atual e possíveis alternativas de solução destes problemas. Desta análise podemos concluir sobre a maturidade do protótipo da Máquina Draco-PUC que vem se firmando como um importante instrumento de pesquisa em reengenharia.

Os trabalhos realizados nesta tese e relacionados à Máquina Draco-PUC, enquanto indispensáveis para a implementação do mecanismo de rastreamento proposto, também abrem novas perspectivas de uso da Máquina Draco-PUC. A construção de uma ferramenta para visualização da aplicação das transformações facilita o entendimento dos conceitos de sistemas transformacionais, a depuração do código das transformações e o próprio projeto das transformações, contribuindo para uma maior facilidade de uso da Máquina Draco-PUC. A possibilidade de uso da Máquina Draco-PUC na manipulação de aplicações XML favorece a disseminação do uso da Máquina Draco-PUC e possibilita a utilização de uma ferramenta mais adequada ao paradigma transformacional, a Máquina Draco-PUC, em conjunto com a tecnologia XML.