IMPLEMENTAÇÃO DO ALGORITMO PARA CONSTRUÇÃO DO SISTEMA ABSTRATO ASSOCIADO A UMA SISTEMA DE TRANSFORMAÇÃO DE GRAFOS TRANSACIONAL COM RELAÇÃO DE DEPENDÊNCIA

TALISSON ROSA GALHO[1]; SIMONE ANDRÉ DA COSTA CAVALHEIRO[2]; LUCIANA FOSS[3]

[1]Universidade Federal de Pelotas – trgalho@inf.ufpel.edu.br
[2]Universidade Federal de Pelotas – simone.costa@inf.ufpel.edu.br
[3]Universidade Federal de Pelotas – lfoss@inf.ufpel.edu.br

1. INTRODUÇÃO

O crescimento tanto em tamanho como em complexidade dos sistemas de software levou a necessidade ferramentas mais precisas para descrever o comportamento desses sistemas. Sistemas de transformação de grafos (GTS) estão presentes entre os métodos existentes para a especificação formal de sistemas. Os GTSs mostram-se bastante adequados como técnica de especificação para descrever distribuição espacial dos estados e reatividade de sistemas (EHRIG, 1999).

Baseado na ideias de redes de Petri “Zero-Safe” foi definida uma extensão de sistemas de transformação de grafos denominada GTS transacional (T-GTS) que permite definir a noção de transação. Uma transação descreve um conjunto de ações executadas de um modo atômico e é definida através de um distinção entre os recursos visíveis e invisíveis de um ponto de vista externo (considerados temporários e “esquecidos” em um nível abstrato). De modo que uma transação é uma sequência de ações (produções) que começa e termina como produções contendo somente itens visíveis (estáveis) e onde todas as produções intermediárias tem algum item invisível (não estável). Esta extensão propõem um formalismo para especificar sistemas reativos, onde as interações entre o sistema e seu ambiente podem ser descritas por relações associadas às produções. Deste modo, permite descrever padrões de interação entre um sistema e seu ambiente.(FOSS, 2008).

Este trabalho visa uma abordagem de implementação na linguagem de programação Java do algoritmo para construção do sistema abstrato associado a um sistema de transformação de grafos transacional com relação de dependência (dT-GTS), proposto na tese de FOSS (2008). Levando em conta contribuições para a área de engenharia de software, ao final da implementação tem-se uma ferramenta capaz obter automaticamente uma visão abstrata de um sistema para verificar a correção de sua implementação. Portanto, afim de obter essa ferramenta ao fim da implementação observa-se e leva em conta o grau de complexidade computacional de algumas rotinas, a tradução de estruturas matemáticas para estruturas computacionais e também o uso de uma ferramenta para definição de um dT-GTS.

2. METODOLOGIA

Uma das primeiras necessidades para a implementação foi a escolha de uma ferramenta apropriada para a definição de um dT-GTS. A ferramenta escolhida foi o software “The Attributed Graph Grammar” (AGG), que possibilita representar grafos e sistemas de transformação de grafos e armazenar as informações utilizando

Extensible Markup Language (XML). Porém algumas informações não são representáveis explicitamente pelo AGG, a exemplo delas temos a representação de elementos não estáveis de um grafo tipo (T) devido à ausência do uso de grafo tipo estável (Ts) pelo AGG. A relação de dependência também não é definida com AGG, uma necessidade ao trabalharmos com dT-GTS. Todavia, o uso de uma nova ferramenta não foi necessário, apenas necessitou-se definir algumas restrições e usar um arquivo adicional contendo as definições de dependência.

A complexidade computacional é algo a ser observado, em vista de que os recursos são finitos é preciso reconhecer qual a complexidade do problema que está sendo abordado e definir quais táticas serão utilizadas. O algoritmo proposto em FOSS (2008) apresenta vários laços de repetição aninhados, que se tem garantias de que eles sempre param. Também há operações da álgebra de conjuntos que internamente, a nível de implementação, acabam se tornando laços de repetição, que também sempre param já que todos os conjuntos são finitos (FOSS, 2008).

Por conseguinte, as restrições e limitações encontradas estão ligadas ao tempo de execução e uso de memória, lembrando também que além dos laços de repetição, é preciso lidar com o problema de encontrar todos os morfismo para um grafo, que se encontra na classe NP-Completo (MEHLHRON, 1984). O que se apresenta como maior desafio para a implementação do algoritmo.

3. RESULTADOS E DISCUSSÃO

A manipulação do arquivo de entrada e tradução do modelo matemático para o modelo computacional foram as bases do trabalho. Fazendo uso dos recursos da interface de programação de aplicações JDOM (JDOM API) obteve-se em estrutura de árvore todas as informações disponíveis a respeito do dT-GTS. Uma análise no padrão utilizado pelo AGG combinado com a forma como a JDOM API manipula XML permitiu obter de forma simplificada todas as informações necessárias para obter-se o dT-GTS.

Todas as estruturas que foram implementadas seguiram as definições utilizadas em FOSS (2008), acrescentando métodos que se mostraram necessários no decorrer da implementação do algoritmo para obter dados sobre os grafos. As escolhas de implementação acabaram visando reduzir a complexidade de algumas rotinas e, quando possível, o uso desnecessário de memória.

A estrutura dos grafos foi implementada da forma que um grafo é composto por: dois HashSets representando os conjuntos de vértices e de arestas e dois HashMaps representando as funções de mapeamento das arestas nos vértices de origem e destino. Utilizando HashMap e HashSet temos algumas restrições asseguradas pela implementação feita pelo “Java Collections Framework” (ORACLE, 2014). Para se beneficiar dessas restrições asseguradas pelas estruturas mencionadas acima, optou-se por representar os vértices e as arestas por números inteiros.

A estrutura de HashMap foi utilizada em todos os momentos que foi preciso representar um mapeamento, e da mesma forma como HashSet para conjuntos. Por utilizarem o conceito de tabela hash em sua implementação, possibilitam consultas em tempo constante (CORMEN, 2009).

Com a implementação de grafo bem definida, foi possível realizar as demais implementações que dependiam da implementação da estrutura de grafo. A classe

que representa um grafo tipado foi composta por três grafos: T, Ts e um grafo genérico G, e um morfismo de G para T. Posteriormente a classe que representa as produções, que são definidas por três grafos tipados (lado esquerdo (L), interface (K) e lado direito (R) de uma produção) e dois morfismos. Várias consulta aos elementos do grafos são realizadas no algoritmo e operações realizada em função destas consultas. Devido as operações sobre as consultas, foi necessário um cuidado para garantir a integridade do grafo. Isto porque uma vez referenciado externamente um elemento de um grafo pode ser modificado externamente o que também implicará em mudanças no grafo. Sendo assim, optou-se pelo uso de consultas por cópia, ao invés de referência, afim de garantir integridade. Porém ocasionando o uso adicional de memória.

Como grande parte das estruturas no algoritmo tem um corresponde direto computacionalmente, não foram necessárias grandes modificações ou adaptações. Alguns recursos foram encontrados em bibliotecas externas, como a do projeto “Guava” mantido pelo Google. A biblioteca desse projeto forneceu um método eficiente para gerar um conjunto das partes. Para gerar o conjunto das partes de um conjunto S, temos de dois elevado à cardinalidade de S subconjuntos distintos (GERSTING, 1993). A biblioteca se mostra eficiente pelo uso de memória O(n) (GUAVA, 2014) em função do elementos do conjunto S. Este método para obter conjunto das partes foi usado como uma sub-rotina de um método para obter todos os subgrafos de um grafo. Esta sub-rotina é executada dois elevado a cardinalidade do conjunto de vértices de um grafo.Portanto, tendo em vista que o método para obter conjunto das partes de um conjunto é utilizado repetidas vezes, justifica-se o uso de uma método eficiente para essa tarefa.

Ainda dentro dos métodos implementados, iniciou-se a implementação de uma abordagem para encontrar todas as ocorrências de um grafo em outro. Abordagem baseia-se no que foi proposto em RUDOLF (2000), aplicando restrições para obter possíveis ocorrências. Umas das vantagens desta abordagem encontra-se em não necessitar traduzir os grafos para um problema de satisfação de restrições (CSP).

4. CONCLUSÕES

O projeto explorou bibliotecas e interfaces de programação de aplicações procurando definir modelos (classes) que pudessem ser reutilizados em novas aplicações e também atender todas as necessidades do algoritmo implementado. Verificou-se que algumas traduções não podiam ser feitas diretamente, devido ao valor semântico das atribuições serem distintos no algoritmo teórico do algoritmo computacional em Java. Também iniciaram-se estudos para explorar formas razoáveis de encontrar todas as ocorrências (*matches*) de um grafo em outro.

5. REFERÊNCIAS BIBLIOGRÁFICAS

CORMEN, T.H.; LEISERSON, C.E.; RIVEST, R.L.; STEIN, C.**Introduction to Algorithms**. Massachusetts: The MIT Press, 2009. 3ed.

GERSTING, J.L.**Fundamentos Matemáticos para a Ciência da Computação**. Rio de Janeiro: LTC - Livros Técnicos e Científicos Editora S.A., 1995. 3ed.

MEHLHORN, K. **Graphalgorithms and NP-completeness**.New York: Springer-Verlag New York, Inc.1984.

EHRIG, H. **Handbook of Graph Grammars and Computing by Graph Transformation.** River Edge: World Scientific, 1999. v.2.

RUDOLF, M. Utilizing Constraint Satisfaction Techniques for Efficient Graph Pattern Matching. In: **INTERNATIONAL WORKSHOP ON THEORY AND APLLICATION OF GRAPH TRANSFORMATIONS**, 6., London, 2000. Proceedings… Springer-Verlag, 2000. p.238-251

FOSS, L. **TranscationalGraphTransformation Systems**. 2008. Tese de Doutorado em Ciência da Computação - Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul.

ORACLE. **Outline of the Collections Framework.** ORACLE, Califórnia.2014.Acessado em 20jul. 2015. Online. Disponível em:http://docs.oracle.com/javase/8/docs/technotes/guides/collections/reference.html

GUAVA. **Guava: Google Core Libraries for Java**. GuavaLibraries,Califórnia, 2014.Acessado em 20jul. 2015. Online. Disponível em: http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/index.html