DIAGRAMAS UML NA VERIFICAÇÃO FORMAL DE SOFTWARE

Relatório final PIBIC

Bolsista: Eduardo Rohde Eras
e-mail: eduardorohdeeras@gmail.com
Responsável: Prof. Dr. Nandamudi L. Vijaykumar
e-mail: vijay@lac.inpe.br
Colaborador: MSc. Luciana Brasil R. Santos

Julho/2014
São José dos Campos

1 – Introdução

Métodos de Verificação Formal oferecem grande potencial para prover técnicas de verificação mais efetivas, pois utilizam rigor matemático para estabelecer a correção do sistema (BAIER, 2008). Além disso, Métodos de Verificação Formal, como por exemplo *Model Checking*, são aplicados de forma mais eficiente nos estágios iniciais do projeto de software, quando os custos são ainda baixos e os benefícios podem ser altos, aumentando a qualidade dos sistemas de software. Todavia, exatamente por utilizarem um rigor matemático que muitas vezes é exigido dos usuários, Métodos de Verificação Formal não são muito adotados como técnica de verificação de sistemas, pois a preferência dos usuários é dada a processos de verificação mais simples.

A Linguagem de Modelagem Unificada (UML) é atualmente aceita como padrão para modelagem de projeto de software, e seu uso tem crescido na indústria aeroespacial. UML apresenta diversos diagramas focando em aspectos distintos do software mas, ao mesmo tempo, provê descrições redundantes dos mesmos aspectos do sistema. Isso dá a oportunidade para técnicas de verificação e validação de assegurar a consistência das descrições. Contudo, verificação e validação de sistemas complexos desenvolvidos de acordo com UML não são tarefas triviais, devido a complexidade do software em si, e a diversos diagramas/modelos UML diferentes que podem ser usados para modelar o comportamento e a estrutura do sistema.

Nesse contexto, o objetivo do presente trabalho é o estudo, análise e manipulação dos diagramas UML para viabilizar o uso de Verificação Formal (mais precisamente, *Model Checking*) na indústria de software. Para isso, é necessário projetar e implementar um aplicativo para converter a representação UML para algum formato específico utilizado por uma ferramenta de *Model Checking*. Diagramas UML, ao serem criados, são representados por XMI. Então, o arquivo com formato XMI é a entrada para o aplicativo para que possa ser processado e convertido em um modelo chamado de Sistema de Transições (TS), base para entrada da ferramenta de *Model Checking*.

Para alcançar tal objetivo, é preciso se familiarizar com os fundamentos da linguagem UML, sua base em XML onde é construída a linguagem XMI (padrão de compartilhamento de metadados UML). A segunda parte é a elaboração de um leitor de XMI em Java para posterior e a elaboração de um conversor do XMI para TS. Essa etapa é auxiliada pelo estudo de máquinas de estado finito (autômatos), que provê uma visão formal de como elaborar a saída do programa: é preciso ter uma saída consistente que represente o diagrama TS e formular o conversor visando obter essa saída. Uma vês implementado e testado o conversor, suas saídas podem ser utilizadas diretamente para criação da linguagem de entrada do *Model Checking*. É previsto a criação de mais dois novos módulos para auxiliar esse processo

Este projeto, iniciado em março de 2013, foi desenvolvido em conjunto com um trabalho de pesquisa em doutorado no curso de Computação Aplicada (CAP) no Instituto Nacional de Pesquisas Espaciais (INPE).

2 – UML

Em uma definição formal, (BOOCH, 1998) "*A Linguagem de Modelagem Universal (UML) é uma linguagem padrão para escrever diagramas de software*".

2.1 Conceitos e Características

A UML é uma linguagem gráfica para visualizar, especificar, construir e documentar os artefatos de um sistema de software. A UML proporciona uma maneira padrão para escrever diagramas de sistema, abrangendo conceitos, como processos de sistema, assim como coisas concretas, como classes escritas em uma determinada linguagem de programação, esquemas de banco de dados e componentes reutilizáveis de software (BOOCH, 1998).

Os diagramas UML são divididos em estruturais e comportamentais, sendo o foco deste trabalho os diagramas classificados como comportamentais, em especial os de Sequência e Atividades. O diagrama de Máquinas de Estado está previsto em uma expansão futura para o módulo conversor. Segundo Booch (BOOCH, 1998), tem-se as seguintes definições:

- Diagramas de sequência são diagramas de interação com ênfase na ordem cronológica das mensagens. Um Diagrama de Sequência mostra uma coleção de objetos e as mensagens enviadas e recebidas por ambos objetos.

- Diagramas de Atividade são essencialmente um fluxograma, mostrando o fluxo de controle em atividades. Focam em representar atividades ou partes de processamento que podem ou não corresponder a métodos de classes. Uma atividade é um estado com uma ação interna e uma ou mais transições de saída que automaticamente ocorrem com o término da ação interna.

- Uma Máquina de Estado define o comportamento que especifica os estados de um objeto durante sua vida em resposta a eventos, juntamente a suas respostas a esses eventos.

Os diagramas de Sequência, Atividade e Máquina de estado mostram, por três diferentes ângulos, o mesmo processo dentro de um sistema de software. São três diferentes visões do comportamento de uma aplicação quanto à sua sequência de interações, às atividades exercidas e aos estados de cada processo, que juntos serão utilizados para formar um único modelo de estado finito (ou, como já definimos, Sistema de Transições) que servirá de entrada para realizar a Verificação Formal de software.

3 – XML e XMI

"Extensible Markup Language (XML) é uma tecnologia para criar linguagens de marcação que descrevem dados de praticamente qualquer tipo de uma forma estruturada" (DEITEL, 2003).

A linguagem XML por sua vez é definida por arquivos DTDs e *schemas*, que descrevem as regras que serão obedecidas no documento XML a eles referenciados. O entendimento de DTDs e *schemas* é essencial para a real compreensão da linguagem XML.

O propósito de um DTD (Documento de Definição de Tipo) é definir os blocos de construção válidos de um documento XML (W3SCHOOLS, 2013). Esses documentos descrevem de forma estruturada, cada elemento que será utilizado no XML que a ele se referir, formalizando sua escrita. Desta forma é possível validar o conteúdo de um documento XML.

Um *Schema XML* é uma alternativa ao DTD baseada em XML que descreve a estrutura de um documento XML (W3SCHOOLS, 2013). Diferente dos DTDs, um *Schema XML* usa a própria linguagem XML como base, garantindo maior flexibilidade e expansibilidade além de manter os documentos XML e seus *schemas* no mesmo padrão.

"XMI, que significa XML para troca de metadados, é o padrão para representação de informações orientadas à objeto usando XML" (GROSE, 2002).

Como as informações dos diagramas UML de Caso de Uso, Sequência, Atividade e Máquina de Estados são exportados em arquivos XMI pelos softwares de edição UML, sabemos que o produto de entrada da aplicação será por fim, um arquivo XML.

Sendo XML é um dos formatos mais usados para troca de informações estruturadas nos dias de hoje (W3C, 2013), dispomos de amplo suporte, documentação atualizada e uma comunidade ativa, facilitando o desenvolvimento de aplicações de leitura e manipulação dessa linguagem.

4 – *Parsers* DOM e SAX

Parsers são analisadores sintáticos de uma linguagem de programação. No caso da linguagem XML, um *parser* deve separar cada *tag*, (pequenas estruturas de uma linguagem de marcação que indicam o início ou o fim de um bloco) seus atributos e conteúdos, preservando sua estrutura. Os dois *parsers* estudados para esse projeto são o DOM (*Document Object Model*) e SAX (*Simple API for XML*), que segundo Deitel (DEITEL, 2003), têm as seguintes definições:

4.1 – *Parser* DOM

O DOM é um modelo baseado em árvore que armazena os dados do documento XML de entrada em uma estrutura hierárquica. Todo o documento XML lido é armazenado

em uma estrutura de árvore na memória, onde os dados podem ser rapidamente acessados. Também existem recursos para modificar o documento (adicionar ou remover informações do arquivo original).

4.2 – *Parser* SAX

Os analisadores sintáticos baseados em SAX fazem uma leitura dos documentos XML de entrada invocando métodos quando encontram uma marcação (início ou fim de um documento por exemplo), esses métodos retornam o conteúdo do documento a medida em que o arquivo é lido. Com esse modelo baseado em eventos, não é criado uma estrutura de árvore pelo analisador baseado em SAX para armazenar os dados do documento.

4.3 – Escolha do *parser*

Para o projeto foi escolhida a tecnologia SAX como analisador sintático do documento XMI. É mais indicado para criar a própria estrutura de dados para armazenar as informações do arquivo XMI de entrada. Isso facilita na implementação filtros de conteúdo armazenando somente as informações necessárias do documento analisado.

5 – Estrutura de Dados

As informações estruturadas no arquivo XMI (XML) de entrada serão armazenadas em uma lista encadeada para serem acessados posteriormente pela aplicação que fará a leitura desses dados. Essa lista será uma seleção do arquivo original contendo apenas os valores interessantes para serem trabalhado pela aplicação, porém mantendo sua sequência original.

6 – Compiladores (Máquina de estados, autômatos)

A segunda parte do projeto é a conversão das informações do documento XMI para um Sistema de Transições. Para esse fim foi estudado a teoria dos autômatos finitos, que auxiliaram na elaboração da saída do sistema.

6.1 – Autômatos Finitos

Os autômatos finitos envolvem estados e transições entre estados em resposta às entradas. Eles são úteis na elaboração de vários tipos diferentes de software, inclusive o componente de análise léxica de um compilador.

7 – Sistema de Transições e *Model Checking*

Um pré-requisito para realizar *Model Checking* é a propriedade a ser checada e um

modelo do sistema que está sendo considerado. Modelos de sistema descrevem o comportamento do sistema de maneira clara e precisa. Esses modelos normalmente são expressados usando autômatos de estados finitos, conhecidos por Sistema de Transições.

7.1 – Sistema de Transições

Um Sistema de Transições é um conjunto finito de estados e um conjunto de transições. São, na ciência da Computação, normalmente utilizados para descrever o comportamento de sistemas. São basicamente grafos direcionados, onde nós representam estados e arestas representam as transições, ou seja, mudanças de estado. Um estado descreve uma informação sobre o sistema em um determinado instante de seu comportamento. As transições especificam como um sistema pode evoluir de um estado para outro (BAIER, 2008). A Figura 1 mostra um exemplo de Sistema de Transição bem simples, composto por dois estados. A transição especifica que o próximo estado é o oposto do estado atual, ou seja, se o estado atual é A, o próximo estado é !A. Se o estado atual é !A, o próximo estado é A. No exemplo, isso é ilustrado com o valor 0 e 1.

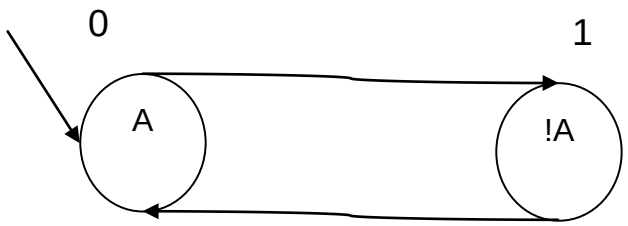

Figura 1: Exemplo de um Sistema de Transição

7.2 – *Model Checking*

"Model Checking é um método executado automaticamente para verificar se um modelo de um sistema cumpre determinadas especificações" (SANTOS, 2012).

De acordo com Baier (BAIER, 2008), *Model Checking* é uma técnica automatizada que, dado um modelo de estado-finito de um sistema e uma propriedade especificada em lógica formal, sistematicamente checa onde essas propriedades suprem um dado estado no modelo. A Figura 2 ilustra o funcionamento do processo de *Model Checking*.

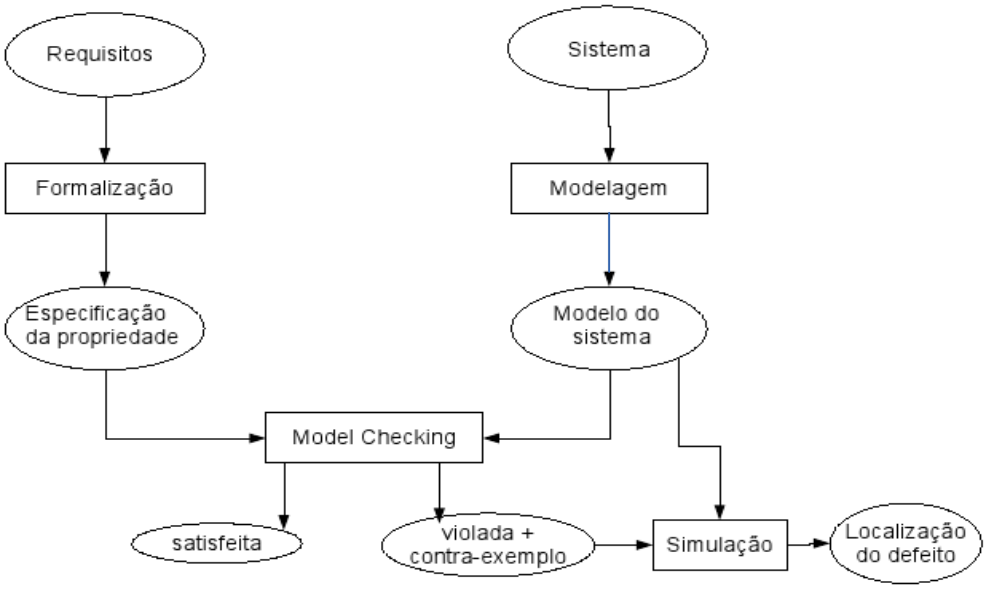

Figura 2: Visão do *Model Checking*. Fonte: (Baier, 2008)

8 – Atividades realizadas e atividades futuras

Segue um resumo das atividades realizadas e uma lista das atividades futuras, previstas para conclusão do projeto.

8.1 – Atividades Realizadas

A primeira etapa realizada foi o estudo da linguagem UML com foco principal nos diagramas de Caso de Uso, Sequência, Atividades e Máquinas de estado (diagramas que serão utilizados no projeto). Foi dado uma atenção especial à sintaxe da linguagem.

A segunda etapa foi o estudo completo da linguagem XML, base para compreensão e manipulação dos arquivos XMI gerados pelos editores UML. O estudo foi guiado pelos tutoriais da W3Schools (W3SCHOOLS, 2013). O estudo da linguagem XML incluiu a abordagem detalhada dos documentos DTDs e *Schemas* XML. Todo o material disponível de XML, DTD e *Schema* da W3Schools foi abordado.

Após um breve estudo sobre a linguagem XMI, o estudo foi direcionado aos *parsers* XML e a implementação de um leitor em Java. Nessa etapa foram estudados os *parsers* DOM e SAX, sua teoria e a implementação em Java de cada um deles. Também foi dedicado um estudo à estrutura de dados com foco especial nas listas encadeadas.

Após esses estudos foi implementado o leitor XML em Java que tem como entrada um arquivo XML ou XMI, armazenando todo seu conteúdo de forma estruturada em listas encadeadas. O correto funcionamento do leitor foi documentado em classes de teste unitário.

Antes da primeira implementação do conversor foi realizada uma pesquisa sobre Autômatos Finitos e Máquinas de Estado. O foco dessa pesquisa está na elaboração da saída do sistema, como o TS é uma máquina de estados, sua representação da forma mais formal possível, é desejada como saída do sistema.

O conversor passou por quatro versões: A primeira implementação ajudou a

elaborar os primeiros mecanismos de tratamento dos arquivos XMI lidos, já sendo capaz de identificar as primeiras mensagens contidas nos diagramas de sequência. A segunda verão reformulou toda arquitetura do sistema e criou o padrão de construção que seria seguido durante o projeto. A segunda versão já fazia toda a conversão dos diagramas de sequência. A terceira versão incrementou a lógica principal e incorporou a conversão dos diagramas de atividades. Nesse ponto a lógica do módulo conversor estava apresentando problemas para acomodar as funções que tratavam os diagramas de atividades. A quarta versão do conversor corrigiu algumas falhas de arquitetura e conseguiu completar a implementação da conversão dos diagramas de atividades, incluindo o tratamento de fragmentos paralelos. Essa quarta versão rendeu uma publicação aceita na ICCSA 2014 em Portugal.

Atualmente o módulo conversor está em sua quinta versão, onde uma profunda revisão foi feita em todo o código, alterando radicalmente sua arquitetura e melhorando significativamente sua organização e flexibilidade, finalizando a plataforma para receber implementações futuras. A quinta versão do sistema, batizada de XMITS 5, já conta com boa parte TUTS (o quarto módulo) e os módulos leitor e conversor em sua versão mais atual, prontos para qualquer nova expansão.

8.2 – Atividades Futuras

A base criada pelo projeto foi pensada para acomodar novos módulos e novas funcionalidades de forma bastante flexível. Em cima da plataforma XMITS 5 são previstas as seguintes atividades futuras:

- Término do quarto módulo, o TUTS, que será responsável por unir os diagramas convertidos em um único diagrama unificado de transição de estados.
- Implementação da conversão de diagramas UML de máquina de estados pelo módulo conversor.
- Implementação do tratamento de fragmentos paralelos à conversão de diagramas de sequência.
- Implementação de um quinto módulo que servirá de ponte entre os XMITS e *Model Checking*, automatizando o processo de verificação dos diagramas convertidos.

9 – Referências e Material de Pesquisa

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J.; **The Unified Modeling Language User Guide**. Addison Wesley, 1998.

GROSE, T. J.; DONEY, G. C.; BRODSKY, S. A.; **Mastering XMI Java Programing with XMI, XML and UML**. John Wiley & Sons, Inc., 2002.

DEITEL, H. M.; DEITEL, P. J.; NIETO, T. R.; LIN, T. M.; SADHU, P.; **XML, como programar**. Bookman, 2003.

HOPCROFT, J. E.; ULLMAN, J. D.; MOTWANI, R.; **Introdução à teoria dos Autômatos, Linguagens e Computação**. Elsevier, 2002.

BAIER, C.; KATOEN, J. P.; **Principles of Model Checking**. MIT Press, 2008.

MOGENSEN, T. Æ.; **Basics of Compiler Design**. DIKU University of Copenhagen, 2007.

FEOFILOFF, P.; **Algoritmos em Linguagem C**. Elsevier, 2009.

SANTOS, L. R. B.; **Formal Verification of UML-Based Software**. INPE, 2012.

W3C; **XML Essentials**. Disponível em <http://www.w3.org/standards/xml/core> Acessado em 30 de Junho de 2013.

W3SCHOOLS; **Introduction to DTD**. Disponível em <http://www.w3schools.com/dtd/dtd_intro.asp> .Acessado em 29 de Junho de 2013.

W3SCHOOLS, **Introduction to XML Schema**. Disponível em <http://www.w3schools.com/schema/schema_intro.asp> Acessado em 29 de Junho de 2013.