UNIVERSIDADE FEDERAL DE PERNAMBUCO – UFPE
CENTRO DE INFORMÁTICA – CIN

PROPOSTA DO TRABALHO DE GRADUAÇÃO

EVOLUÇÃO AUTOMATIAZDA DE MODELOS ARQUITETURAIS CONCORRENTES

Professor Orientador: Augusto Cezar Sampaio
Aluna: Camila Sá da Fonseca

Recife, 31 de agosto de 2010

1. Introdução

O trabalho de graduação, aqui proposto, está essencialmente baseado na necessidade de automatização de transformações de modelos de sistemas distribuídos, especialmente no contexto de *MDA* (*Model Driven Architecture*) [BRN04].

O primeiro passo é uma análise comparativa de modelos de representação e transformação de sistemas concorrentes, porque - para a metodologia de desenvolvimento de software *MDA* - a busca por padrões arquiteturais é fundamental para verificar se um modelo pode ser transformado em outro, mantendo o comportamento, mas com uma estrutura mais adequada. Esse cenário promove a evolução do sistema de maneira mais simplificada.

Para mecanizar este tipo de refatoração, há algumas ferramentas disponíveis, em especial as linguagens de transformação. Para concretizar este trabalho será realizada a implementação de algumas dessas transformações com a linguagem/ferramenta de transformação de programas Stratego/XT. Assim, serão implementadas Leis de Transformações para a linguagem UML-RT, um profile de UML com construções que suportam concorrência e tempo real.

2. Motivação

Os sistemas de software estão - para atender as necessidades das organizações e da própria sociedade - se tornando mais complexos a uma velocidade exponencial. Para tentar atender a esta demanda é necessário buscar - e encontrar! - alternativas que consigam reduzir os recursos para o desenvolvimento desses sistemas. Uma das técnicas para otimizar o processo de desenvolvimento está relacionada à criação e à evolução de modelos do sistema. Estudar este contexto é fundamental para compreender uma das metodologias mais difundidas atualmente.

Por outro lado, os sistemas concorrentes e distribuídos enquadram-se nesse espectro de complexidade e são necessários para boa parte dos sistemas a serem criados e aos que ainda são mantidos.

Diante desse contexto, buscar ferramentas que deem suporte à criação e à manutenção (evolução) de modelos de sistemas concorrentes e distribuídos é duplamente relevante. Além disso, a tentativa de utilizar uma ferramenta nova, como Stratego/XT, para implementar algumas das transformações de maneira automatizada é desafiador e recompensador.

3. Sistemas concorrentes e distribuídos

Atualmente, sistemas de software estão presentes em qualquer negócio e no próprio dia-a-dia das pessoas. Além disso, a disseminação do uso da rede mundial de computadores ao mesmo tempo em que intensifica esse processo, torna-se indispensável por promover a troca de informações entre esses computadores. Mas, por trás dessa sistemática de comunicação, há sempre um ou mais programas - sistemas - que coordenam essa comunicação.

Segundo [CDK94], “um sistema distribuído é uma coleção de computadores autônomos conectados por uma rede e equipados com um sistema de software distribuído". Por essa afirmação percebemos que para a existência de um sistema distribuído (também chamados de SD) são necessários tanto computadores - ou processadores -, como uma rede e o próprio software que possua componentes em execução em cada um dos computadores pertencentes ao sistema.

Com o avanço tecnológico tanto na capacidade de processamento como nas redes de computadores, fica mais fácil e, muitas vezes, mais eficiente agrupar um grande número de CPUs conectadas por meio de uma rede trabalhando em um sistema distribuído.

Acrescentando à definição acima, [TAN95] afirma que “um sistema distribuído é uma coleção de computadores independentes que aparentam ao usuário ser um computador único", nota-se uma importante característica dos sistemas distribuídos, a transparência. Muitas vezes as pessoas estão usando esse tipo de sistemas mesmo sem ter conhecimento deste fato.

Alguns exemplos de sistemas distribuídos são: computadores e PDAs em um estoque de uma companhia que estejam rodando o mesmo sistema de controle; computadores de bancos que trocam diversas informações constantemente e, atualmente, pode-se destacar a computação em nuvem (*cloud computing)*, entre muitas outras aplicações distribuídas.

O crescimento do uso de sistemas distribuídos ocorre especialmente porque, para certas aplicações, essa modalidade consegue obter uma capacidade de processamento muito superior à oferecida por um sistema centralizado. Além disso, um sistema distribuído é escalável e pode reduzir os riscos caso haja falha em alguns dos processadores envolvidos, por não dependerem exclusivamente de um computador. Consequentemente, há maior confiabilidade nos dados e disponibilidade de informações. Os diversos usuários também podem, por meio dos SD, compartilhar recursos não só de software, mas também de hardware.

A concorrência em sistemas de software significa que há mais de um processo em execução a cada instante, assim está inerentemente presente nos sistemas distribuídos; afinal cada computador operando no sistema pode realizar algumas atividades independentemente. Há situações em que há o acesso concorrente aos recursos compartilhados, diante disso deve haver sincronização para controlar esses acessos. Assim, temos que um sistema de software distribuído é composto por instruções modularizadas em processos que são executadas concorrente ou paralelamente por um ou mais processadores.

Apesar dos benefícios expostos, segundo [GUI08] o desenvolvimento de sistemas em ambientes distribuídos é muito complexo, afinal é preciso tratar das heterogeneidades das plataformas, sistemas operacionais e linguagens de programação; concorrência; comunicação entre os elementos distribuídos; segurança e qualidade de serviço. Além da necessidade de promover a transparência: esconder os detalhes inerentes à distribuição do processamento, tais como a localização geográfica dos elementos funcionais, replicação de dados e funcionalidades, além de ocorrência de falhas. Exposto esse cenário fica clara a importância da comunicação entre os processos - entre as máquinas - para que o sistema funcione corretamente.

Atualmente, formalismos e técnicas de análise (verificação) de modelos conseguem oferecer suporte mecânico à verificação. Assim sendo, garantir que programas concorrentes

satisfaçam certas propriedades desejáveis não tem sido um desafio na área de engenharia de sofware. Nossa abordagem é baseada na definição e evolução de modelos de forma construtiva, garantindo que os modelos evoluídos preservem o comportamento dos modelos iniciais e mais abstratos.

4. Modelagem no desenvolvimento de sistemas

4.1. Modelagem

Modelos são utilizados nas mais diversas áreas de engenharia. A Engenharia Civil, por exemplo, não inicia um projeto sem antes ter um modelo daquilo que será construído, muitas vezes além da planta, tem a maquete, arquivos digitais 3D, entre outros. Para a Engenharia de Software não deve ser diferente e os modelos têm se mostrado fundamentais para permitir a pré-visualização do que será desenvolvido e representar a especificação e documentação do software.

Segundo a OMG (Object Management Group) - organização que coordena padrões para aplicações orientadas a objetos - estruturar é uma forma de lidar com a complexidade dos sistemas e de promover o reuso de código. Afinal, a fase de análise e projeto do sistema é o momento mais fácil de estruturar uma aplicação como uma coleção de módulos e componentes.

A modelagem de software, em geral, utiliza algum tipo de notação gráfica - que representa os artefatos dos componentes utilizados e seus relacionamentos - e é suportada por alguma ferramenta CASE.

No final de 1994, a OMG criou a UML - Unified Modeling Language - segundo [SIL01]: “A UML (Unified Modeling Language) é uma linguagem para especificação, documentação, visualização e desenvolvimento de sistemas orientados a objetos. Sintetiza os principais métodos existentes, sendo considerada uma das linguagens mais expressivas para modelagem de sistemas orientados a objetos. Por meio de seus diagramas é possível representar sistemas de softwares sob diversas perspectivas de visualização”.

O surgimento dessa linguagem conseguiu padronizar a modelagem visual de software e aumentou dramaticamente o uso da modelagem, em 1995 as ferramentas de modelagem eram utilizadas por uma pequena fração de projetos de software [WAT]; por volta de 2006 estimou que mais de 10 milhões de profissionais de TI usavam UML e por volta de 2008 mais do que 70% das organizações de desenvolvimento de software em todo o mundo estava utilizando a linguagem [NOR06]. Atualmente, a OMG afirma que a UML é a forma na qual o mundo modela não apenas a estrutura, o comportamento, e a arquitetura de um sistema, mas também os processos de negócio e a estrutura dos dados.

4.2. MDA

Contemporaneamente, além das funções tradicionais, os modelos de software podem ser usados como componentes principais ao longo do desenvolvimento, é o que se chama de *MDA - Model Driven Architecture* (Arquitetura Dirigida a Modelos). Para isso, ferramentas de modelagem capturam o significado dos elementos dos diagramas e dos seus relacionamentos e utilizam essa

compreensão para compor elementos de projeto, testes de performance e até geração automática de partes do código da aplicação.

A MDA auxilia os usuários de software a lidar com duas realidades primordiais no desenvolvimento de software: múltiplas tecnologias de implementação e a necessidade de manutenção ao longo de toda vida do sistema.

O desenvolvimento MDA utiliza duas formas de modelos [OMG09]:

- PIM - Platform-Independent Model (Modelo Independente de Plataforma): inclui a representação das funcionalidades de negócios e comportamento, mas não expressa aspectos técnicos;
- PSMs - Platform-Specific Models (Modelos Específicos de Plataforma): após mapeamentos realizados a partir do PIM pode-se chegar a um PSM que contém as mesmas informações que a própria implementação, mas na forma de UML ao invés de código fonte.

Algumas ferramentas dão suporte às transformações em um modelo em diversos passos: desde a análise inicial até o código executável [BRN04], ou seja, desde uma visão mais abstrata até a mais concreta do sistema. Esse processo de transformações no modelo é o que chamamos de refinamentos e segue a tendência que as próprias linguagens de desenvolvimentos sofrem: tornar-se cada vez mais abstrato e tentar automatizar o processo de refinamento [LUB09].

Com MDA também é possível importar modelos utilizados em outros projetos, mesmo que esse projeto tenha utilizado uma metodologia diferente. Isso é possível por meio da representação de UML em XMI - XML Metadata Interchange (Intercâmbio de Metadados XML).

5. Transformações de Modelos

Segundo [BRJ98] todo modelo pode ser expresso em diferentes níveis de precisão. Utilizando-se dessa premissa no desenvolvimento em MDA, um sistema é definido em um modelo de alto nível e passa por transformações - que adicionam detalhes sobre o sistema ou que apenas convertem representações - até chegar a um modelo de uma tecnologia específica. Percebe-se que as transformações entre modelos é o conceito chave para obter benefício do MDA pois possibilita uma maior automação no desenvolvimento e implementação de software, reduzindo, consequentemente, esforço e custo [LUB09].

Ainda segundo [BRN04] as diferenças entre os diferentes tipos de modelos nos permitem pensar em software e desenvolvimento de sistemas como uma série de refinamentos entre representações diferentes do modelo.

Diante do exposto, percebe-se a importância que as transformações exercem quando se trata de modelagem de sistemas, e, consequentemente, do desenvolvimento de sistemas.

Essa ideia de transformações de modelos tem raízes na representação formal de sistemas - os métodos formais. Atualmente há diversas abordagens para execução de transformações, entretanto a maioria delas apresenta limitações à interoperabilidade com ambientes de desenvolvimento, apoio à definição e execução de diversos tipos de transformações e à

possibilidade de expansão de suas capacidades [LUB09], além de geralmente seguir a tradição de formalismos.

Existem três categorias de transformações de modelos [OLI08]:

I. *Refactorings*: a transformação produz mudanças de reorganização no modelo de entrada (PIM ou PSM), preservando a semântica, mas alterando a estrutura. Em uma refatoração qualquer mudança não será percebida por entidades externas.
II. Modelos para Modelos: comumente aplicada para diminuir o nível de abstração do modelo; são refinamentos.
III. Modelos para código: gera código ou trechos de código executável a partir de um dado modelo.

Para este trabalho, as transformações a serem exploradas serão as do tipo I – *Refactorings* e do tipo II – Modelos para Modelos, colaborando com a automatização dessas reestruturações de um sistema.

6. Modelos com suporte à distribuição e à concorrência

Diante do exposto, percebemos que à medida que os sistemas distribuídos são cada vez mais necessários, eles trazem complexidades ao desenvolvimento; isso porque a especificação e projeto de sistemas distribuídos é uma tarefa complexa que envolve a especificação de dados, comportamento, comunicação e de aspectos arquiteturais do sistema [RSM06].

Por outro lado, há a busca por metodologias que invertam as complexidades desses sistemas em um desenvolvimento mais fácil, menos dispendioso e com menor necessidade de retrabalho. Ou seja, apesar do aumento da complexidade dos sistemas é desejável que o aumento da complexidade no seu desenvolvimento não siga na mesma proporção.

Como exibido, uma das metodologias que visa facilitar o desenvolvimento é a MDA. Para encaixar essas duas tendências, precisa-se de modelos que ofereçam suporte à modelagem da concorrência e da distribuição, permitindo, assim, o uso da MDA em sistemas distribuídos. Para suprir essa necessidade no campo da modelagem desses sistemas a linguagem UML for Real-Time (UML-RT) foi criada [SER98] e funciona como um profile de UML e ROOM para aplicações concorrentes e de tempo real. Há outras linguagens que têm o mesmo objetivo, com destaque para Wright [RJA97] que é uma ADL (*Architectural Description Language).* Contudo, neste trabalho usamos UML-RT como linguagem de modelagem.

Em UML-RT quatro novos construtores são introduzidos de maneira conservativa à UML e ROOM: cápsulas, protocolos, portas e conectores. Além disso, alguns diagramas de UML são estendidos: diagramas de classe, estado e estrutura (extensão de diagramas de colaboração).

Percebe-se que essas duas tendências realmente devem ser bem exploradas porque em sistema distribuídos há extrema necessidade de modularização e, com modelos para representar essa disposição, a implementação e o entendimento do sistema são bastante aperfeiçoados.

Antes do desenvolvimento de qualquer sistema que promova automatização de transformações em modelos, é necessário que essas transformações já tenham sido validadas. Nas

transformações a serem exploradas - *refactorings* - é preciso que uma lei de transformação possa ser aplicada em ambos os sentidos (obedecendo às condições) e mantenha o comportamento do sistema.

As transformações definidas por [RAM05] serão as bases para o desenvolvimento dos experimentos desse trabalho: essas leis são divididas em leis básicas e leis de refatoramento. Contudo, ao longo do desenvolvimento deste projeto outras leis poderão ser sugeridas como resultado do próprio processo de implementação das transformações.

7. Experimento com Stratego/XT

Este trabalho visa dar suporte à implementação de transformações expressas em formalismos [RSM06] dentro da evolução dos sistemas e explora a possibilidade de permitir as transformações confiáveis de forma mecanizada para os desenvolvedores.

Será investigada a possibilidade de – com uma nova ferramenta e linguagem: Stratego/XT – efetivar a automação de refatorações arquiteturais de modelos de sistemas baseados em componentes. Mostrando-se possível para parte das transformações, poder-se-á estender para as outras e atividades comuns de modelagem poderão ser simplificadas, facilitando e agilizando a evolução dos sistemas distribuídos no contexto de MDA.

Stratego/XT é um conjunto de linguagem e ferramenta que fornece regras para expressar transformações básicas, estratégias programáveis para controlar a aplicação das regras, sintaxe concreta para expressar padrões de regras na sintaxe da linguagem destino e reescrita dinâmica de regras para expressar transformações sensíveis ao contexto, suportando, assim, o desenvolvimento de componentes de transformação num alto nível de abstração [SXT10].

Verifica-se que o projeto tem muita relevância porque permite que se conheça a possibilidade de promover transformações confiáveis em um sistema de software representado em UML-RT com uma ferramenta inovadora que é Stratego. Constatando-se essa possibilidade poderemos estender a abrangência das regras automatizadas e facilitar a evolução de sistemas tão necessária no contexto de MDA.

8. Cronograma das Atividades

Atividades / Meses	Agosto	Setembro	Outubro	Novembro	Dezembro
Levantamento bibliográfico					
Estudo MDA					
Estudo das transformações e escolha das que serão implementadas					
Análise do suporte à concorrência em modelos (UML-RT)					
Implementação em Stratego/XT					
Avaliação dos resultados					
Consolidação das Informações					
Apresentação					

BILBIOGRAFIA

[CDK94] G. Coulouris; J. Dollimore e T. Kindberg. Distributed Systems: Concepts and Design. Addison-Wesley, 1994. ISBN 0-201-62433-8.

[TAN95] A. S. Tanenbaum. Distributed Operating Systems. Prentice-Hall, 1995. ISBN 0-13-219908-4.

[GUI08] E. G. Guimarães. Introdução aos Sistemas Distribuídos e Componentes de Software. CTI Renato Archer, 2008.

[NOR06] D. Norton. "View DSLs and UML as 'Fraternal Twins', Not Competitors". Gartner Research, 2006

[WAT] A. Watson. Visual Modelling: past, present and future. Object Management Group.

[SIL01] D. M. Silva. UML - Um guia de consulta rápida. Editora Novatec, 2001. ISBN: 85-7522-004-7

[OMG09] Introduction to OMG's Unified Modeling Language (UML). Object Management Group, Inc, 2009. Disponível em: http://www.omg.org/gettingstarted/what_is_uml.htm. Acessado em: 20/08/2010.

[BRN04] A. Brown. An introduction to Model Driven Architecture. IBM Technical Library, 2004. Disponível em: http://www.ibm.com/developerworks/rational/library/3100.html. Acessado em: 21/08/2010

[BRJ00] G. Booch; J. Rumbaugh e I. Jacobson. UML, Guia do Usuário; tradução: Fábio Freitas da Silva, Rio de Janeiro, Campus ,2000.

[UML10] http://www.uml.org/ Acessado em: 20/08/2010.

[LUB09] V. C. Lunelli e A. T. Bacelo. Transformação de modelos em processos de desenvolvimento de software. X Salão de Iniciação Científica PUCRS, 2009.

[RAM05] R. Ramos. Desenvolvimento Rigoroso com UML-RT. Dissertação de Mestrado CIn-UFPE, 2005.

[RSM06] R. Ramos; A. Sampaio e A. Mota. Transformation Laws for UML-RT. In: IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 06), 2006, Bologna - Itália. Springer - Lecture Notes in Computer Science. v. 4037. 2006. p. 123-138.

[RSM062] R. Ramos; A. Sampaio e A. Mota. Rigorous Development with UML-RT. 19th Brazilian Contest on Dissertations and Thesis (CTD'06), SBC, 2006.

[OLI08] J. D. S. Oliveira. Automação de Leis de Refatoração Arquitetural. Trabalho de Graduação - Centro de Informática, UFPE, 2008.

[SER98] B. Selic e J. Rumbaugh. Using UML for Modeling Complex RealTime Systems. Rational Software Corporation, 1998.

[SXT10] http://strategoxt.org/Stratego/WebHome, acessado em: 20/08/2010

[RJA97] R. J. Allen. A Formal Approach to Software Architecture. Ph.D. Thesis, Carnegie Mellon University,CMU Technical Report CMU-CS-97-144,1997. Disponível em: http://www.cs.cmu.edu/~able/paper_abstracts/rallen_thesis.htm. Acessado em: 30/08/2010