UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

AUTOMAÇÃO DO PROCESSO DE TRANSFORMAÇÃO DE MODELOS UML-RT

PROPOSTA DE TRABALHO DE GRADUAÇÃO

O objetivo deste trabalho é desenvolver uma ferramenta que vise à automação da aplicação de transformações sobre modelos[17] definidos na notação UML-RT[16]. Baseado em um conjunto de regras de transformação[5], podemos realizar mudanças em um modelo independente de plataforma (PIM, Plataform Independent Model) agregando informação durante o ciclo de vida de seu projeto. Facilitando assim o processo de definição de sua plataforma (PSM, Plataform Specific Model). A automação da transformação entre estes modelos será possivelmente realizada implementada como um plug-in para a ferramenta Rational Rose RT.

Aluno: Gustavo da Fonseca Limaverde Cabral (gflc@cin.ufpe.br)
Orientador: Augusto César Alves Sampaio (acas@cin.ufpe.br)

8 de Junho de 2004

1. MOTIVAÇÃO

As metodologias tradicionais de desenvolvimento de sistema baseadas em um ciclo de vida do sistema dividido em concepção, projeto, implementação, teste e implantação possuem diversos problemas como:

- Artefatos produzidos durante as fases de especificação e projeto do sistema são deixados de lado assim que a codificação do sistema é iniciada. Isso ocorre devido ao alto custo de manutenção dos artefatos de documentação durante as fases finais do processo de desenvolvimento;
- Após o início da codificação do sistema já deve ter sido especificada toda a arquitetura da aplicação, ou seja, modificações na arquitetura durante a fase de codificação são impraticáveis. Ou seja, todo os artefatos que descrevem o sistema devem ser especificados de acordo com o ambiente, a plataforma escolhida;
- A necessidade de interoperabilidade de sistemas pode surgir no decorrer do tempo. Com isso, novos sistemas que venham a depender do software legado projetado para uma plataforma específica serão penalizados com a dependência de tecnologias não mais empregadas, ultrapassadas.

É muito simples desenvolver um sistema quando as especificações são estáticas. Os problemas descritos acima ocorrem justamente porque novos requisitos e modificações são quase sempre necessários.

A complexidade de manter todo um conjunto de artefatos devidamente atualizado de acordo com as alterações é altíssima [6]. Paralelamente temos o surgimento de novos sistemas, que utilizam novas tecnologias, e que tornam o processo de integração necessário. A seguir iremos apresentar alguns aspectos que estão relacionados à evolução de sistemas:

- **Modificações no sistema**

Para manter a satisfação dos clientes, as funcionalidades do sistema são continuamente incrementadas. Essas mudanças fazem com que o sistema tenha que ser continuamente adaptado.

- **Qualidade**

Como as alterações realizadas tendem a aumentar a complexidade, a qualidade do código, produzido por vários desenvolvedores, tende a cair. Isso porque modificações são realizadas por muitos e a diferença entre real, especificado, e o implementado tende a aumentar se o processo de alteração não for rigorosamente seguido.

- **Impactos**

Para manter a qualidade do sistema, é necessário aplicar diversas regras de refatoramente que venham a distribuir a complexidade, reorganizar e repadronizar o sistema. O processo é custoso e complexo.

2. CONTEXTO

Recentemente a OMG, Object Managment Group [1], lançou um novo padrão de arquitetura de modelos, descrevendo possíveis papeis e relações entre modelos num processo baseado na utilização destes artefatos, chamada MDA [2]. Em um processo como este, a utilização de modelos continuaria a ser feita nas fases de projeto e análise, porém diferencia-se pelo fato de incorporarem muito mais informações sobre as regras de negócio do que processos tradicionais.

A incorporação de novas informações aos artefatos neste processo é feita através da aplicação de transformações sobre modelos, que são vistas como artefatos tão importantes quanto os próprios modelos. De fato a seqüência de transformações aplicadas a um modelo inicial do sistema irá sempre indicar o estado atual de seu desenvolvimento. Assim, transformações seriam feitas até que o sistema estivesse o mais próximo possível do código, servindo como entrada da fase de implementação.

Para que a utilização do processo acima descrito seja aceito como forma padrão de desenvolvimento de software, devem existir ferramentas que automatizem o processo de aplicação das regras de transformação.

Atualmente, só existem ferramentas dedicadas à aplicação de transformação sobre código, modelo gerado na fase de implementação. Este processo é chamado de *Refactoring* [8]. Estas ferramentas são de extrema importância quando temos que manter a qualidade do código de um sistema.

Complementarmente, seria igualmente importante poder aplicar transformações sobre os modelos mais abstratos do que código fonte. O poder de tais transformações seria bem maior. Transformações de maior complexidade poderiam ser aplicadas a estes modelos, provocando impactos positivos com menor custo de realização.

2.1 Processo de desenvolvimento baseado em modelos

Modelos aumentam o grau de abstração na especificação do sistema. Assim podemos idealizar todo um sistema sem se preocupar com questões como portabilidade e interoperabilidade. Além disso, modelos são bem mais fáceis de serem entendidos, modificados e documentados.

Um problema que surge, porém, é como a partir de um modelo criar a aplicação. Foi para resolver esse problema que foi introduzida a idéia de ciclo de vida do desenvolvimento MDA. Nele encontramos os seguintes artefatos, utilizados nas fases de análise, projeto e implementação do sistema respectivamente:

- CIM (Computational Independent Model), seria um documento de requisitos do sistema, produzido na fase de análise, descrevendo textualmente as funcionalidades do sistema. Poderia utilizar também diagramas de Caso de Uso de UML, por exemplo, para facilitar o entendimento;

- PIM (Plataform Independent Model), artefato central do processo de desenvolvimento do sistema, também produzido na fase de análise do ciclo de vida do sistema. É formado por modelos que podem ser descritos em UML e outras linguagens de modelagem bem definidas, que descrevam comportamento e dados do sistema, ou seja, informação suficiente para especificar o sistema;

- PSM (Plataform Specific Model), artefato descrito através de uma linguagem de especificação de modelos, pode ser alguma extensão de UML, ou seja, um *profile*[15], instanciação, de UML para uma determinada tecnologia, plataforma. Sua geração ocorre na fase de projeto do sistema;

- Código, gerado a partir de um PSM onde foi definida a plataforma onde o sistema irá funcionar.

Depois de especificado o sistema, produzimos o primeiro modelo, um PIM, em UML, por exemplo. A partir dele podemos gerar PIM's mais detalhados ou dar o próximo passo e tomar decisões com relação à plataforma do sistema e gerar o PSM do sistema e ai então código.

Para modificar PIM's, PSM's e código (refactoring) utilizamos regras de transformação. Quando definidas, podemos realizar as seguintes transformações:

- PIM para PIM: aumentando assim o detalhe da especificação do sistema. Com isso o PSM gerado poderá ser mais específico, ou seja, conterá mais informação importante no processo de transformação;

- PIM para PSM: após a definição da arquitetura do sistema, aplicamos transformações ao PIM para gerarmos um novo modelo que leva em conta características da plataforma. Podemos gerar mais de um PSM a partir de um PIM. Assim, teremos mais modelos, cada um compondo uma parte do sistema, como por exemplo, em uma arquitetura de camadas;

- PSM para código: a partir do PSM já temos condição de produzir o código. Possuímos detalhes da arquitetura, regras de negócio e especificações sobre o comportamento do sistema.

No processo de transformação entre modelos, o ideal é que se tenha a definição da transformação e sua inversa. Assim, modificações no PSM poderiam ser refletidas no PIM e modificações no código no PSM.

Considerações relevantes

Abaixo iremos introduzir alguns aspectos que devem ser levados em contas quando estamos tratando de transformação de modelos.

- **Interoperabilidade entre modelos**

Também é necessário definir pontes de comunicação, interoperabilidade, entre os PSM e os códigos, respectivamente. Só assim PSM´s, dois a dois, poderão se comunicar. O mesmo ocorre entre dois modelos de código gerados a partir de PSM´s.

- **Consistência incremental**

Um outro fato importante que deve ser levado em conta é a consistência incremental entre os modelos. Após a aplicação das transformações sobre um modelo, temos que garantir que o novo modelo refine o modelo fonte. Desta forma teremos um incremento no modelo sem modificar a semântica inicial.

- **Utilização de ferramentas**

Uma vez implementada as *definições de transformações* estas podem ser aplicadas ao modelo. Para automatizar esse processo podemos utilizar ferramentas que interpretem a *definição da transformação* e aplique-a ao modelo. Existem diversas categorias de ferramentas para manutenção e manipulação de modelos, aqui iremos apenas nos preocupar com as que automatizam o processo de aplicação das transformações.

- **Linguagem**

UML[13] é considerada a linguagem padrão para definição de PIM´s. Mas somente UML não é necessário para descrever um sistema. Existem abordagens que fazem uso de UML com OCL (Object Constraint Language) [13] e outras que usam UML com AS (Action Semantics),conhecido com *Executable UML*. Mas definitivamente ainda não existe uma linguagem completa, que possa descrever um sistema levando em conta aspectos estruturais e comportamentais.

- **Processo**

No desenvolvimento de sistemas baseado em modelos não é necessário utilizar nenhum processo de desenvolvimento específico. Assim a equipe de desenvolvimento pode agregar os benefícios de diversas metodologias como RUP[9], o XP[10] ou outras metodologias ágeis[11]. A única diferença seria a adição de novas atividades e artefatos dentro do processo de desenvolvimento.

2.2 Classes ativas e regras de transformação

Comumente a modelagem de sistemas se restringe a produção de diagramas estáticos contendo classes passivas em relação ao ambiente externo. Entretanto existem casos em que é necessário descrever o comportamento de uma classe. Isso ocorre principalmente quando esta se encontra em um contexto que envolva execução concorrente e independente do ambiente.

- **UML-RT**

Classes ativas é um conceito utilizado para descrever melhor o comportamento de objetos em tempo de execução. Uma extensão de UML que tem sido bastante utilizada para descrever este conceito é UML-RT [7]. Nele são definidos novos conceitos através de estereótipos: cápsulas, protocolos e conexões.

Cápsulas representam classes ativas, independentes do resto do sistema, com comportamento bem definido e utilizam portas como única interface de comunicação com o resto do sistema. Estas portas são canais que coordenam a comunicação entre as cápsulas, e realizam os serviços definidos em um protocolo. Portas são instâncias de protocolos. A cada cápsula é associado um único comportamento, representado por um *statecharts*, diagrama de estados de UML. Em adição, a cápsula pode ser definida hierarquicamente, em termos de outras cápsulas internas, cada uma com seu respectivo comportamento. Desta forma fica mais fácil definir sistema que fazem uso de concorrência e distribuição.

2.3 Estado da Arte

Diversos trabalhos na área de transformação de modelos vêm sendo realizados no intuito de refinar modelos, porém em sua maioria se dedicam somente a explicar *refactorings* e refinamentos em diagramas de classe. Quando diagrama comportamentais (como diagramas de atividades e estados) são tratados, estes trabalhos nunca relacionam as possíveis implicações que uma transformação em um diagrama pode acarretar em outro.

Desta forma podemos concluir como uma grande contribuição de nosso trabalho a implementação de transformações que levassem em conta tanto aspectos estruturais quanto comportamentais. Além disso, até o momento não foram encontrados trabalhos relacionados à implementação de transformações de modelos para UML-RT.

Esse trabalho se baseia na implementação de um conjunto básico de transformações para UML-RT definido em [5] sobre classes ativas. Estas transformações são focalizadas principalmente em modelos PIM, porém nada impede que seus conceitos sejam aplicados a modelos PSM que utilizam UML-RT, já que na maioria das vezes a descrição da plataforma pode ser feita somente através da adição de novos estereótipos. Através deste conjunto básico de transformações será possível compor transformações mais complexas utilizadas durante a fase de projeto de forma que propriedades do modelo não sejam perdidas durante o seu ciclo de vida. Um exemplo de transformação mais complexa pode ser a aplicação de padrões de projeto ao modelo.

3. OBJETIVOS

A utilização de ferramentas para a aplicação das transformações é fundamental para a automatização do processo. Estamos introduzindo esse novo paradigma de desenvolvimento de sistema justamente para aumentar a produtividade e qualidade dos sistemas produzidos; não poderíamos assim tornar o processo mais custoso. Devemos fazer uso de ferramentas que implementem a aplicação de transformação como um dos focos da prática.

Será desenvolvido uma extensão para o Rational Rose RT[12], um *plug-in*, que implemente a aplicação de transformações sobre modelos UML-RT. Inicialmente, serão implementadas as transformações descritas em [5], ou seja, um conjunto pré-definido de regras. Regras adicionais deverão ser propostas e implementas a partir das necessidades que devem surgir com o desenvolvimento de um estudo de caso.

Na realidade o principal objetivo desse trabalho é avaliar as regras de transformações definidas em [5] e estudar a viabilidade da utilização desse paradigma de desenvolvimento.

A aplicação automática das regras ajudará no processo de validação do resultado esperado após a aplicação das transformações. Ou seja, iremos verificar se os resultados obtidos são equivalentes aos resultados esperados.

Esse processo de validação pode facilitar a correção e melhoria das regras de transformação já existentes.

Para validar a aplicação das transformações iremos desenvolver um estudo de caso onde possamos aplicar todas as regras de transformação e verificar a usabilidade da utilização de um *plug-in* no processo.

Trabalhos Futuros

Como foi citado anteriormente, a introdução de um processo de desenvolvimento focado na utilização de modelos como principal artefato geraria algum impacto nas metodologias tradicionais.

De fato, teríamos que introduzir novas atividades e responsabilidades que considerem a existência de transformações de modelos como ponto crucial no novo processo.

Um trabalho futuros seria definir uma disciplina, *workflow*, que inclua estas novas atividades, com base na disciplina de Análise e Projeto do RUP. Desta forma, a utilização deste novo paradigma de desenvolvimento não fica desvinculada do resto das atividades de uma metodologia de desenvolvimento de sistemas. Seria uma extensão do RUP para a abordagem MDA.

O estudo do processo de desenvolvimento baseado em modelos é de extrema importância para desenvolvimento de trabalhos na área. O conhecimento dos fundamentos da teoria envolvida irá produzir trabalhos futuros bem mais embasados.

O desenvolvimento de um conjunto de regras de transformação gera artefatos, regras, que podem ser reutilizados sempre que as condições da regra forem supridas. Ou seja, é um trabalho que aumenta o grau de produtividade e manutenibilidade em larga escala.

4. CRONOGRAMA

	4.1.1 Mês											
4.2 Atividades	*Junho*				*Julho*				*Agosto*			
Estudar linguagens de definição de transformações												
Estudar as transformações propostas em [5]												
Estudar a plataforma de desenvolvimento de *plug-in* no Rational Rose RT												
Desenvolver o *plug-in*												
Definir estudo de caso, exemplo de aplicação da ferramenta												
Redação do Relatório Final												

5. REFERÊNCIA

[1] OMG, Object Management Group (2004).URL: www.omg.org
[2] MDA. www.omg.org/mda/
[3] MDA Explained, the Model Driven Architecture: Practice and Promise, Kleppe, Anneke; Warmer, Jos; Bast, Win; Addison-Wesley, 2003
[4] Mundo Java, 1a. Edição, Matéria: O que é MDA?
[5] Ramos, Rodrigo e Sampaio, Augusto - Algebraic Transformation Laws for UML-RT www.cin.ufpe.br/~rtr/publications/RamosLawsForUmlRTReport04.pdf
[6] Lehman, M.M. Theory and Practice of Software Evolution
[7] PDPTA'2001 Shang-Wen Cheng, and David Garlan - Mapping Architectural Concepts to UML-RT
[8] Fowler, M.: Refactoring-Improving the design of existing code. Addison Wesley (1999)
[9] Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley (2000)
[10] XP, http://www.xispe.com.br/index.html
[11] Metodologias Ágeis, http://www.agilealliance.org/
[12] Rational Rose RT, Software(2003), www.rational.com/products/rosert.
[13] Stephan Flake, Real-time constraint with the OCL, C-LAB, Paderborn University
[14] Engels, G., Heckel, R., K¨uster, J.M., Groenewegen, L.: Consistency-Preserving Model Evolution through Transformations. In J´ez´equel, J.M., Hussmann, H., Cook, S., eds.: UML 2002 - The Unified Modeling Language. Model Engineering, Languages, Concepts, and Tools. 5th International Conference. Volume 2460 of LNCS., Dresden, Germany, Springer (2002) 212–226
[15] Object Management Group: UML Profile for Scheduling, Performace and Time Specification (2003) Version 1.0, OMG Document formal/2003-09-01.
[16] Selic, B., Rumbaugh, J.: Using UML for Modeling Complex RealTime Systems. Rational Software Corporation (1998) URL: http://www. rational.com.
[17] Gogolla, M., Richters, M.: Transformation Rules for UML Class Diagrams. In: First International Workshop on The Unified Modeling Language (UML)'98. LNCS, Springer-Verlag (1999) 92–106

6. DATA E ASSINATURAS

01 de Junho de 2004

Gustavo da Fonseca Limaverde Cabral (Aluno)	Augusto César Alves Sampaio (Orientador)