Uma Abordagem para o Controle da Evolução de Software no Desenvolvimento Orientado a Modelos

Chessman Kennedy Faria Corrêa[1]
Leonardo G. P. Murta[1]
Claudia M. L. Werner [1]

[1]Programa de Engenharia de Sistemas e Computação - COPPE
Universidade Federal do Rio de Janeiro (UFRJ) – Rio de Janeiro, RJ – Brasil

{chessman,murta,werner}@cos.ufrj.br

Nível: Mestrado
Ano de ingresso: 2006
Previsão de conclusão: Julho de 2007
Aprovação da proposta: 12/02/2007

Resumo. *Este trabalho apresenta uma abordagem para o controle da evolução de software no desenvolvimento orientado a modelos usando o framework MDA. A evolução de software será controlada a partir da manutenção da sincronização e versionamento dos modelos PIM, PSM e código fonte. Também será considerado o versionamento dos rastros e das transformações a serem usadas na geração de modelos.*

Palavras-chave. *MDA, MDD, Evolução de Software, Sincronização de Modelos, Versionamento*.

1. Introdução

O **Desenvolvimento Orientado a Modelos** (*MDD – Model Driven Development*) [Soley 2000] é caracterizado pelo uso da modelagem como principal recurso para a criação de *software*, e os modelos criados podem ser usados no final do processo para a geração automática de código fonte.

O aumento constante da complexidade do software faz com que a modelagem seja um recurso essencial para uma melhor compreensão do problema e elaboração de uma solução que permita resolvê-lo da melhor maneira possível [Booch *et al.* 2005]. Assim, é possível imaginar que o MDD poderá se tornar a principal abordagem de desenvolvimento de software no futuro. Porém, problemas que tornaram necessário o versionamento do código fonte, como, por exemplo, o desenvolvimento concorrente e o aparecimento de defeitos, poderão ter um impacto ainda maior no MDD. Assim, este trabalho tem como objetivo apresentar uma abordagem para o controle da evolução de software no Desenvolvimento Orientado a Modelos.

2. O Contexto

O **MDA** (*Model Driven Architecture* – Arquitetura Orientada a Modelos) [OMG 2003] é o *framework* da OMG (*Object Management Group*) para o MDD. No MDA, um modelo em nível mais alto de abstração, independente de plataforma, é usado para gerar modelos com detalhes suficientes para a geração de código para plataformas tecnológicas diferentes, como, por exemplo, o *J2EE* [Sun 2006] e o *.NET* [Microsoft 2007].

O MDA possui os seguintes modelos [OMG 2003]: **CIM** (*Computation Independent Model* – Modelo Independente de Computação), **PIM** (*Platform Independent Model* – Modelo Independente de Plataforma), **PSM** (*Platform Specific Model* – Modelo Específico de Plataforma) e **Modelo de Plataforma (MP)**. O **CIM** representa os requisitos do sistema, sem qualquer relação com a solução do problema. O **PIM** representa a solução independente de plataforma tecnológica. O **PSM** é gerado a partir PIM e possui detalhes de uma plataforma tecnológica específica. O **MP** fornece os conceitos de uma plataforma tecnológica que são usados para a criação do PSM. Um exemplo de modelo de plataforma é o *CORBA Component Model* [OMG 2002], que fornece os conceitos *EntityComponent*, *SessionComponent*, etc.

3. O Problema

Na abordagem elaboracionista, os modelos MDA em diferentes níveis de abstração podem ser modificados independentemente, principalmente quando existem diferentes pessoas trabalhando no mesmo projeto ou quando são usadas ferramentas diferentes para a criação do PIM, do PSM e do código. Por exemplo, modificações realizadas em um PSM para uma plataforma específica podem tornar o respectivo modelo PIM desatualizado. Nesse caso, além dos modelos tornarem-se inconsistentes, a desatualização do PIM pode dificultar ou inviabilizar a geração do mesmo sistema para uma outra plataforma tecnológica. Assim, as diferentes representações do mesmo software podem evoluir ao longo do tempo em direções opostas, ou então, algumas partes podem evoluir, enquanto outras ficam estagnadas.

Acrescentando-se à possibilidade das representações do software se tornarem inconsistentes, o trabalho realizado por um desenvolvedor pode apresentar conflitos em relação aos de outros desenvolvedores, e outros problemas inerentes ao desenvolvimento, como, por exemplo, a manifestação de defeitos, podem fazer com que versões anteriores do software tenham de ser recuperadas. Porém, no MDD com MDA, considerando a existência de rastros entre os diferentes modelos decorrentes das transformações, os modelos também precisam ser mantidos juntamente com o código fonte.

4. A Abordagem Proposta

A abordagem proposta tem como objetivo manter o controle da evolução do software desenvolvido a partir do MDD usando o *framework* MDA, de modo que modelos e código permaneçam sincronizados e versionados após a conclusão de modificações realizadas por pessoas diferentes. Visando a independência das tecnologias usadas para a criação de modelos e código, a abordagem será desenvolvida no lado servidor e o repositório usará formatos padronizados como o XMI (*XML Metadata Interchange*), para a representação dos modelos, XML (*Extensible Markup Language*) para a representação de rastros e transformações, e texto para a representação do código. O esquema da abordagem pode ser observado na Figura 1.

O repositório (1) irá armazenar os modelos PIM, PSM, código, rastros (registros de transformação) e transformações. O motivo para se armazenar os rastros no repositório é facilitar a localização dos elementos de diferentes modelos que estão relacionados. Isso permitirá que a atualização dos modelos seja simplificada. Todos esses elementos serão versionados pelo Controlador de Versão (CV) sempre que necessário (2). Diferentes sistemas poderão ser usados, como, por exemplo, o *Odyssey-VCS* [Oliveira *et al.* 2005].

Sempre que o desenvolvedor tentar atualizar o repositório, um mecanismo para a execução das tarefas da abordagem será acionado (3) (*hook pré-checkin*). A sua finalidade é verificar a necessidade de sincronização ou criação de modelos e acionar a máquina de *roundtrip* (4) ou a máquina de transformação de modelos (5). A máquina de *roundtrip* será executada sempre que os modelos tiverem de ser sincronizados, sem que ocorram perdas em relação a modelos pré-existentes. A máquina de transformação será executada quando um novo modelo tiver de ser gerado. O *Odyssey-MDA* [Maia 2006] é um exemplo de máquina de transformação que poderá ser usada. Sempre que a transformação for realizada, o gerador de rastros (6) irá gerar os rastros entre os elementos, registrando a versão da transformação que foi usada. A base de transformações será usada como referência para a realização das transformações e para a sincronização dos modelos. No segundo caso, a base será essencial para que a máquina de *roundtrip* tenha condições de inserir automaticamente marcações nos modelos, quando necessário, de acordo com as especificações da transformação. O objetivo é evitar que o desenvolvedor tenha que inserir as marcações manualmente, quando novos elementos são incluídos em um modelo, já que esta é uma tarefa suscetível a erros.

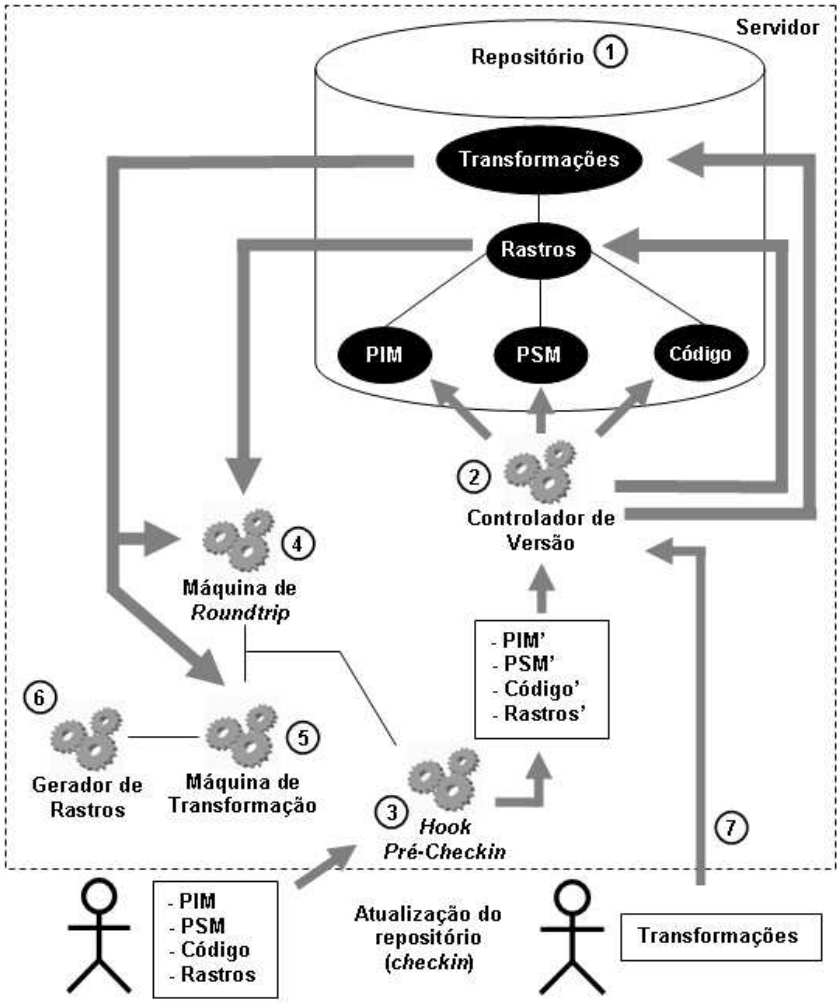

Figura 1. Abordagem proposta.

As transformações também serão versionadas na abordagem (7). O objetivo é permitir a identificação da versão correta das transformações que foram usadas para a criação de modelos e código ao longo da evolução do software.

A lógica operacional básica da abordagem proposta pode ser descrita da seguinte maneira: sempre que o desenvolvedor tentar fazer o *checkin* (atualização de um repositório versionado) a partir de um software cliente, o *hook pré-checkin* verificará quais são as atualizações e as gerações de modelos necessárias. Havendo a necessidade de geração de um novo modelo, a máquina de transformação será executada, juntamente com o gerador de rastros. No caso de sincronização, a máquina de *roundtrip* será acionada. No último caso, a base de transformações será usada de modo que a máquina de *roundtrip* tenha como saber quais são as marcações que devem ser inseridas em novos elementos que eventualmente tenham de ser criados, como, por exemplo, uma nova classe. Depois que todos os modelos estiverem sincronizados, os itens serão enviados para o CV, que se encarregará da verificação de conflitos e atualização do repositório. Dois tipos de conflitos que podem ocorrer são intra-modelos e inter-modelos. Por exemplo, dois desenvolvedores podem ter modificado o mesmo método

existente em uma classe PSM (conflito intra-modelo). A modificação de um método de uma classe PSM pode entrar em conflito com uma modificação realizada no método da classe PIM equivalente (conflito inter-modelo).

5. Trabalhos Relacionados

Um dos trabalhos relacionados é o *Hismo* [Girba *et al.* 2005]. Essa abordagem é direcionada para o acompanhamento da evolução do software a partir de um histórico constituído de diferentes versões dos elementos de modelo, usando meta-modelos para armazenar as informações de versionamento. Contudo, não considera os rastros existentes entre modelos em diferentes níveis de abstração, nem a sincronização de modelos.

Outro trabalho relacionado é o *ModelBus* [Afonso *et al.* 2006], uma abordagem MDD que integra e coordena ferramentas de modelagem, transformação de modelos, geração de código, etc. Apesar de realizar a criação do CIM, PIM, PSM, código e rastros e a sincronização desses modelos, não realiza controle de versão, impossibilitando o controle da evolução do software.

Matheson *et al.* [2004] propuseram uma abordagem muito semelhante à deste trabalho. Essa abordagem considera a existência de modelos em diferentes níveis de abstração e código, as ligações entre esses modelos, a necessidade de se manter o versionamento de modelos e suas ligações (rastros), e a independência de ferramentas de desenvolvimento. É uma abordagem genérica, focada não somente do MDD, mas também na modelagem orientada a aspectos. Portanto, não está focada nos problemas específicos do MDD com MDA, não apresentando quais os procedimentos a serem realizados para manter os modelos MDA e código sincronizados e versionados.

Finalmente, o *Odyssey-VCS* [Oliveira *et al.* 2005], uma abordagem desenvolvida pelo próprio grupo, é um controlador de versão para modelos UML. Assim como o *Hismo*, essa ferramenta não considera a existência dos rastros entre os elementos de modelos diferentes, nem transformações. Além disso, o código fonte não é considerado nesta abordagem.

6. Estado Atual do Trabalho

Este trabalho foi iniciado a partir da necessidade de se ter o controle da evolução do software desenvolvido a partir do MDD e do levantamento da literatura relacionada. Atualmente, encontra-se na fase de especificação dos requisitos.

7. Avaliação da Abordagem

Após a implementação das ferramentas, serão realizados estudos de caso para a avaliação da abordagem. Serão usadas métricas de **precisão** (*precision*) e **revocação** (*recall*) para a análise estatística do estado dos diferentes modelos posteriormente às operações de sincronização e versionamento.

8. Considerações Finais

A possibilidade do MDD tornar-se a principal metodologia de desenvolvimento de software no futuro torna necessário o uso de uma infra-estrutura que possibilite o

controle da evolução do software criado a partir dessa abordagem, independente do número de pessoas trabalhando em um projeto de software ou das tecnologias de desenvolvimento usadas. Além disso, quando o MDA é usado no MDD, a sincronização dos modelos e código versionados é essencial. Portanto, a contribuição da abordagem proposta será possibilitar que esses objetivos sejam alcançados, e, como a solução será implementada no lado do servidor, qualquer ferramenta cliente que tenha suporte para XMI poderá utilizá-la.

A realização desse trabalho abrirá novas oportunidades de pesquisa, como: (1) controle da evolução do CIM (sincronização e versionamento), juntamente com os demais modelos e código; (2) visualização da evolução do software a partir de um cliente Web usando os dados contidos no repositório; e (3) notificação aos desenvolvedores sobre problemas que não puderam ser resolvidos pelo sistema.

9. Referências

Afonso, M., Vogel, R., Teixeira, J. (2006) "From Code Centric to Model Centric Software Engineering: Practical case study of MDD infusion in a Systems Integration Company". In: *Fourth Workshop on Model-Based Development of Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software - Cover*, pp. 125-134, Postdam, Germany, March.

Booch, G., Rumbaugh, J., Jacobson, I. (2005) *UML - Guia do Usuário*, 2 ed., Editora Campus.

Girba, T., Favre, J.-M., Ducasse, S.e. (2005) "Using Meta-Model Transformation to Model Software Evolution", *Electronic Notes in Theoretical Computer Science*, v. 137, pp. 57-64.

Maia, N.E.N. (2006) *Odyssey-MDA: Uma Abordagem para Transformação de Modelos*, Tese de Mestrado, COPPE, UFRJ, Rio de Janeiro - RJ.

Matheson, D., France, R., Bieman, J.*, et al.* (2004) "Managed Evolution of a Model Driven Development Approach to Software-based Solutions". In: *Workshop on Best Practices for Model Driven Development*, Vancouver, Canada, October.

Microsoft (2007), "Microsoft .NET Home Page". In: http://microsoft.com/net, accessed in 28/01/2007.

Oliveira, H., Murta, L., Werner, C.M.L. (2005) "Odyssey-VCS: a Flexible Version Control System for UML Model Elements". In: *International Workshop on Software Configuration Management (SCM-12)* Lisbon, Portugal, September.

OMG (2002) *CORBA Component Model, Version 3.8*, Object Management Group.

OMG (2003) "MDA Guide Version 1.0.1", *Object Management Group*.

Soley, R. (2000) *Model Driven Architecture*, OMG document omg/00-05-05, Object Management Group.

Sun (2006), "The J2EE 1.4 Tutorial". In: http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html, accessed in 03/09/2006.