Odyssey-VCS: Um Sistema de Controle de Versões Para Modelos Baseados no MOF

Hamilton Oliveira, Leonardo Murta, Claudia Werner

COPPE/UFRJ – Programa de Engenharia de Sistemas e Computação
Caixa Postal 68511 – CEP. 21945 - 970
e-mail: [hamilton, murta, werner]@cos.ufrj.br

Resumo

Este artigo descreve o Odyssey-VCS, um sistema de controle de versões para modelos baseados no Meta002D Object Facility (MOF). O Odyssey-VCS fornece políticas configuráveis para definir grãos de versionamento e comparação para meta-modelos (M2), e uma política genérica para o meta-meta-modelo(M3).

Abstract

This paper presents Odyssey-VCS, a version control system for models based on Meta-Object Facility (MOF). Odyssey-VCS provides a configurable policy to define grains of versioning and comparison for meta-models (M2), and a generic policy for the meta-meta-model layer (M3).

1. Introdução

Os sistemas modernos estão se tornando mais complexos em tamanho, sofisticação e tecnologias utilizadas [5]. O aumento da complexidade é um obstáculo na tentativa de melhorar o grau de previsibilidade dos produtos a serem construídos, porque aumenta o número de incertezas nos projetos de desenvolvimento. Por outro lado, na grande maioria dos projetos, persiste a convicção de que o sistema irá mudar porque o ambiente no qual este se encontra inserido está sujeito a mudanças. Segundo [9], ambientes de desenvolvimento que não controlam a mudança são conduzidos rapidamente ao caos.

A Gerência de Configuração de Software (GCS) consiste numa abordagem disciplinada para controlar o processo de desenvolvimento e manutenção do software [1]. A GCS é uma das disciplinas mais maduras e utilizadas da engenharia de software. No entanto, suas potencialidades são ainda pouco exploradas, pois somente o controle de versões sobre código-fonte tem sido efetivamente adotado nos projetos.

Muitas abordagens têm sido propostas para controlar o código-fonte. Neste contexto, a utilização de um modelo de dados baseado em sistemas de arquivos é suficiente. Entretanto, sistemas complexos demandam um esforço adicional para sua compreensão, tornando necessário o uso de artefatos que descrevem aspectos não representados pelo código-fonte, agregando informação em um nível de abstração mais elevado [11].

Quando se versiona artefatos de alto nível de abstração (modelos UML, por exemplo), utilizando modelo de dados baseado em sistemas de arquivos, os resultados obtidos são pouco satisfatórios. O problema está relacionado à granularidade, porque nesse modelo de dados um arquivo é um item de configuração indivisível. Portanto, um modelo persistido em um arquivo é versionado como um elemento único.

Diante do exposto, este artigo descreve o Odyssey-VCS, um sistema de controle de versões para modelos baseados no *Meta-Object Facility* (MOF)[8]. O Odyssey-VCS fornece políticas configuráveis para definir grãos de versionamento e comparação para meta-modelos (M2), e uma política genérica para o meta-meta-modelo (M3).
O restante do artigo está organizado em cinco seções: A seção 2 descreve os padrões utilizados na implementação. A seção 3 apresenta os trabalhos relacionados. A seção 4 descreve a abordagem proposta. A seção 5 apresenta um exemplo de utilização da ferramenta e a Seção 6 conclui o artigo e descreve algumas vantagens e limitações.

2.Padrões Relacionados

A implementação do Odyssey-VCS está baseada nos padrões MOF e *Java Metadata Interface* (JMI) [4]. O MOF especifica uma linguagem abstrata para descrever outras linguagens. MOF é também conhecido como meta-meta-modelo (M3) e as linguagens abstratas descritas a partir dele são chamadas de meta-modelo (M2) [6].

JMI gera interfaces programáticas em linguagem Java para um determinado meta-modelo MOF. Assim, é possível manipular instâncias desse meta-modelo usando os próprios elementos do domínio em que ele se encontra. O metamodelo da UML, por exemplo, é descrito a partir do MOF. Utilizando JMI é possível manipular os elementos da UML (Classe, Caso de Uso, Pacote,...) como objetos Java, através das interfaces geradas. Esse tratamento pode ser adotado, não somente para a UML, mas para qualquer meta-modelo baseado no MOF.

O *Metadata Repository* (MDR) é um repositório que implementa os padrões MOF e JMI, o que lhe permite carregar metamodelos baseados no MOF, e armazenar instâncias (M1) desses metamodelos. As instâncias armazenadas no repositório podem ser manipuladas programaticamente, utilizando as interfaces geradas a partir do padrão JMI [6].

3.Trabalhos Relacionados

[12] é um sistema de controle de versões que utiliza sistema de arquivos como modelos de dados. Em virtude de sua simplicidade, esse modelo de dados não é adequado para manipular artefatos com estruturas de dados complexas gerados nas fases de análise e projeto.

Outras abordagens versionam documentos XML. No entanto, um pré-processamento se faz necessário para transformar a metáfora de documentos XML em metáfora de artefatos de alto nível para que operações comuns de controle de versões (*checkin*, *checkout*, junção) possam ser aplicadas. Essa abordagem introduz *overhead* e cria possibilidades de falhas na implementação.

O MIMIX [2] é uma abordagem que utiliza XMI como formato de persistência. A ferramenta não permite a variação do grão de versionamento, e conseqüentemente trata o modelo como uma entidade única, da mesma forma que as ferramentas que se baseiam em sistemas de arquivos.

[10] apresenta uma abordagem que faz uso de um modelo temporal versionado (TVM) aplicado a uma perspectiva de controle de versões. Essa abordagem faz com que um item de configuração passe a ser um objeto, ao invés de um arquivo do sistema operacional. No

entanto, é empregado um meta-modelo proprietário, o que torna difícil utilizá-lo em ambientes onde a interoperabilidade é um requisito importante. Além disso, toda a informação de versão é armazenada no próprio artefato versionado, constituindo-se numa abordagem intrusiva.

4.Abordagem Proposta

A figura 1 apresenta um cenário de utilização do Odyssey-VCS. O padrão XMI é utilizado como formato de representação dos dados que trafegam entre o cliente, (Ambientes de Reutilização, Ferramentas CASE e Ambientes de Modelagem) e o Odyssey-VCS, o que o torna apto a ser utilizado por qualquer ferramenta que seja capaz de importar/exportar modelos utilizando este padrão. Como meio de transporte, é utilizado *Web Service*, habilitando o Odyssey-VCS a ser utilizado por qualquer equipe, independentemente de sua localização geográfica, através da Internet.

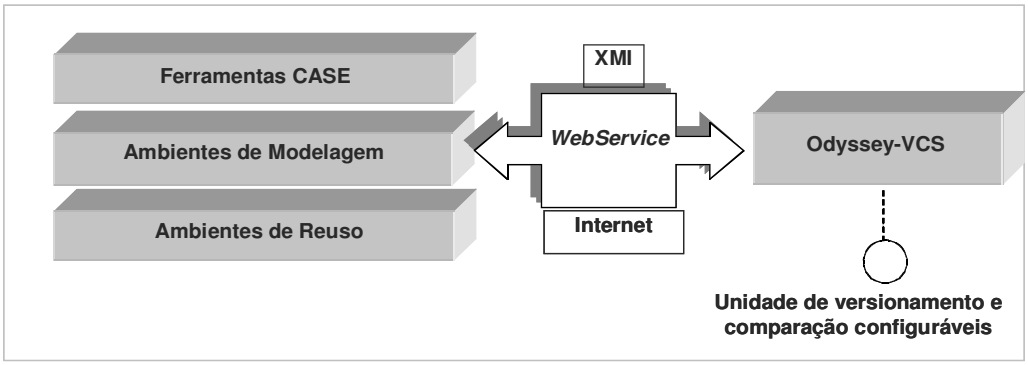

Figura 1. Um cenário de utilização do Odyssey-VCS

Ainda na figura 1, observa-se que a Unidade de Versionamento (UV) e a Unidade de Comparação (UC) são configuráveis, diferentemente das abordagens convencionais, que possuem grãos de comparação e versionamento fixos [7]. A UV representa os elementos que serão versionados e, portanto, irão se tornar itens de configuração quando enviados para o repositório. A UC é utilizada como parâmetro para a identificação de conflitos quando a operação de junção (*merge*) é realizada. Se dois desenvolvedores modificam um elemento que é UC, um conflito ocorre. Este conflito é solucionado com base na política para resolução de conflitos fornecida pelo Odyssey-VCS.

A figura 2 exibe um trecho de código que descreve grãos de comparação e de versionamento para elementos do meta-modelo da UML.

```
    .
    .
<elemento tipo="org.omg.uml.foundation.core.UmlClass" graoVersionamento="true">
        <elemento tipo="org.omg.uml.foundation.core.Attribute" graoComparacao="true" />
        <elemento tipo="org.omg.uml.foundation.core.Operation" graoComparacao="true" />
</elemento>
    .
    .
```

Figura 2. Exemplo de configuração dos grãos de comparação e versionamento (Murta, 2004)

Nesse descritor, elementos do tipo classe são considerados grãos de versionamento e elementos dos tipos atributos e métodos são considerados grãos de comparação. Por ser definida como UV, toda classe receberá informação de versionamento, como, por exemplo, o

número da versão atual e as razões dessa versão ter sido criada. Mais ainda, toda vez que mais de uma pessoa editar concomitantemente um método ou atributo, será detectado um conflito. Além disso, é definida a hierarquia de composição onde atributo e método são partes de classe [7]. Essa hierarquia é útil para, ao se editar, inserir ou remover um método ou atributo, saber que uma nova versão da classe deve ser gerada.

O tratamento diferenciado, para cada elemento de modelo versionado, introduz alguns problemas inexistentes nas demais abordagens de controle de versões: um pacote possui características e sub-elementos diferentes de uma classe, e ambos apresentam diferenças em relação a um caso de uso. Para solucionar o problema, foi definida a interface Tratador. A figura 3 descreve o código Java para a interface Tratador.

```
public interface Tratador {
        public void insere(byte[] elementoModelo, String descricao, double IDUsuario);
        public byte[] checkout(String IDVersao, double IDUsuario);
        public String checkin(byte[] elementoModelo, String descricao, double IDUsuario);
        public boolean isUnidadeVersao();
        public boolean isUnidadeComparacao();
}
```

Figura 3. Interface Tratador, descrita em Java

O parâmetro 'elementoModelo', do método 'insere', representa o elemento enviado pelo cliente e que será inserido no repositório. O parâmetro foi definido como um *array* de bytes, que é o formato no qual o cliente envia o elemento para o Odyssey-VCS, conforme descrito na figura 1. O elemento é convertido para objetos Java, utilizando as funcionalidades providas pelo MDR, e inserido no repositório. O método '*ckeckout'* permite transferir o artefato do repositório para a área de trabalho do desenvolvedor. Depois de efetuadas as modificações, o '*checkin'* é utilizado para devolver o artefato para o repositório novamente.

Os métodos 'isUnidadeVersao' e 'isUnidadeComparacao' retornam *true* ou *false*, informando se o elemento em questão é UV e/ou UC. Esse processamento é realizado com base nos valores definidos no trecho de código descrito na figura 2. Para cada elemento de modelo, deve existir uma classe Tratadora que implementa a interface Tratador, da seguinte forma: para um pacote, temos a classe TratadorPacote, para uma classe, TratadorClasse, e assim por diante. A implementação das classes tratadores viabiliza a definição de políticas configuráveis no nível do meta-modelo (M2).

No entanto, é possível que um elemento de modelo seja configurado para ser versionado, mas não tenha definida uma classe tratadora, acarretando a ausência de tratamento específico no nível do meta-modelo(M2). Neste caso, o elemento é versionado utilizando tratadores definidos para os elementos descritos no meta-meta-modelo (M3). Por exemplo, o elemento Caso de Uso, da UML, é uma instância do elemento Classificador, do MOF. Supondo que a classe TratadoraCasoDeUso não tenha sido definida, o versionamento pode ser realizado através da classe TratadoraClassificador. Esse procedimento constitui-se na adoção de políticas genéricas definidas ao nível do MOF (M3).

5.Exemplo de Utilização

Para ilustrar um exemplo de utilização do Odyssey-VCS, temos, na figura 4, a classe 'Professor' inserida no pacote 'academia', ambos pertencentes ao modelo 'Controle Acadêmico'. Se a operação de inserção for invocada para esses elementos, teremos três

instâncias de itens de configuração criadas no repositório: uma para o modelo, e recursivamente, outras duas instâncias de itens de configuração para o pacote 'Academia' e classe 'Professor'.

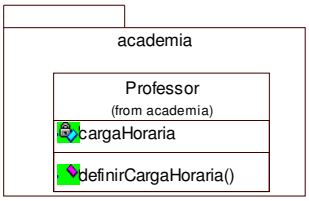

Figura 4. Exemplo de elementos de modelo

A figura 5 mostra um Cliente-Odyssey-VCS que consiste num protótipo de um cliente interagindo com o Odyssey-VCS. O protótipo realiza o papel dos ambientes/ferramentas *CASE* que estão descritos no lado esquerdo da figura 1.

Após os elementos terem sido carregados pelo cliente, foi invocado o método de inserção. Três itens de configuração foram criados, um para modelo 'ControleAcadêmico', e recursivamente, outros dois itens de configuração para o pacote 'Academia' e para a classe 'Professor', que pode ser observado no lado esquerdo do protótipo, da figura 5. As funcionalidades *Listar ICs* e *Historia* possibilitam acessar o repositório e obter todos os itens de configuração criados, bem como suas versões, respectivamente.

Para reduzir espaço de armazenamento, os sistemas de controle de versões geralmente utilizam o conceito de *delta(ou diff).* Desta forma, armazena-se uma versão na íntegra, além das diferenças entre esta e as demais versões. O Odyssey-VCS armazena todas as versões integralmente no repositório, sem a utilização do conceito de delta, o que acarreta um grande uso de disco. Por outro lado, a escalabilidade de processamento ocorre, pois para recuperar uma versão basta navegar pelos modelos armazenados, sem ter que computar *diffs*. [3] argumenta que essa abordagem é viabilizada atualmente pelo barateamento dos recursos de memória e pela evolução das técnicas de compactação de arquivos.

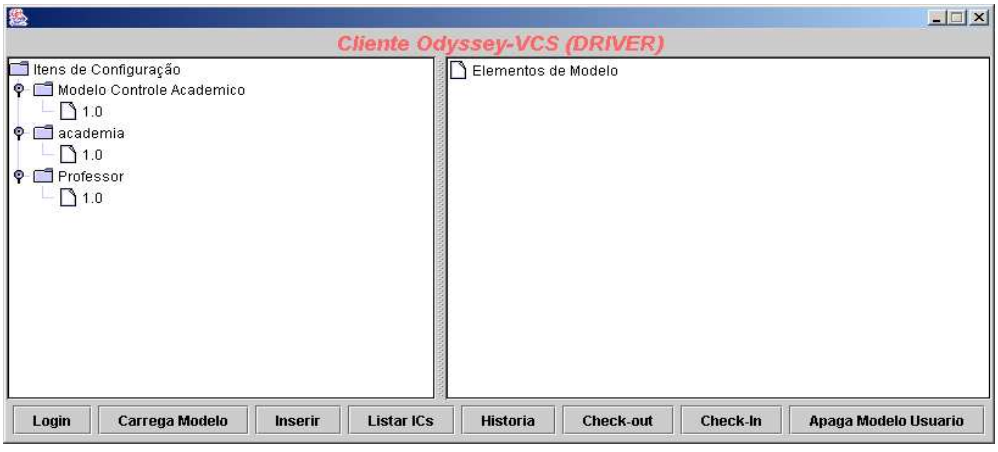

Figura 5. Cliente Odyssey-VCS

6.Conclusões

Apresentamos neste artigo a ferramenta Odyssey-VCS, desenvolvida pelo grupo de reutilização da COPPE/UFRJ, para controlar versões de modelos baseados no MOF. A

ferramenta foi desenvolvida em linguagem Java, e sua utilização contribui com as seguintes vantagens: (1) independência de ambiente de desenvolvimento, porque utiliza XMI como formato de representação para transporte; (2) independência de localização geográfica da equipe de desenvolvimento, pois utiliza *Web Service* para a camada de transporte; e (3) facilidade de extensão, para qualquer modelo baseado no MOF.

Embora a proposta do Odyssey-VCS seja versionar qualquer modelo baseado no MOF, a versão atual trata somente modelos UML, em virtude da estratégia *bottow-up* adotada para implementação.

Atualmente, está sendo implementada a funcionalidade que permitirá a dois ou mais desenvolvedores modificarem o mesmo artefato concorrentemente. Como próximos passos, serão implementadas as funcionalidades para tornar a ferramenta apta a versionar efetivamente qualquer modelo baseado no MOF.

Referências

1. Burrows, C., George, G., Dart, S., Configuration Management, Ovum Limited (Ed.), London, UK, ISBN: 1898972761, http://www.ovum.com, May.
2. El-Jaick, D., 2004, MIMIX: Sistema de Apoio à Modelagem Cooperativa de Software Utilizando Ferramentas CASE Heterogêneas, Dissertação de mestrado (em fase final de elaboração), COPPE, UFRJ, Rio de Janeiro, RJ.
3. Estublier, J., Leblang, D. B., Clemm, G., Conradi, R., Hoek, A., Ticky, W. F, Wiborg-Weber, D., "Impact of the research community for the field of software configuration management", In: International Conference Software Engineering, Orlando, USA, 2002
4. Java Community Process, (2004), Java Metadata Interface (JMI) API 1.0 Specification, In: http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html, Accessed in 17/05/2004.
5. Leon, A., A Guide to Software Configuration Management, Norwood, MA, Artech House Publishers, 2000.
6. Matula, M., 2003, "NetBeans Metadata Repository", In: http://mdr.netbeans.org/docs.html, accessed in 17/05/2004.
7. Murta, L. G. P., 2004, "Odyssey-SCM: Uma Abordagem de Gerência de Configuração de Software para o Desenvolvimento Baseado em Componentes", Exame de Qualificação, COPPE/UFRJ, Rio de Janeiro, RJ, Maio.
8. OMG, 2003a, Meta Object Facility (MOF) specification, version 1.4, Object Management Group.
9. Pressman, R. S., Software Engineering, A Practitioner's Approach, McGraw-Hill, 2001.
10. Silva, F. A., Costa, R. V. C., Edelweiss, N., et. al., 2003, "Using the Temporal Versions Model in a Software Configuration Management Environment", In: Simpósio Brasileiro de Engenharia de Software, Manaus, Amazonas, October.
11. Teixeira, H. V., Murta, L. G., Werner, C. M., 2001 "LockED: Uma Abordagem para o Controle de Alterações de Artefatos de Software", In: IV Workshop Ibero-americano de Engenharia de Requisitos de Engenharia de Software, San José, Costa Rica, Abril.
12. Tichy, W., 1985, "RCS: a system for version control", Software – Practice and Experience, v. 15, n. 7, pp. 637-654