Uma Abordagem Baseada em Ontologias para a Integração de Ferramentas de Apoio à Gerência de Configuração em um Ambiente de Desenvolvimento de Software

2 authors:

Rodrigo Calhau
Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo (I...
13 PUBLICATIONS **35** CITATIONS

SEE PROFILE

Ricardo de Almeida Falbo
Universidade Federal do Espírito Santo
172 PUBLICATIONS **1,661** CITATIONS

SEE PROFILE

Uma Abordagem Baseada em Ontologias para a Integração de Ferramentas de Apoio à Gerência de Configuração em um Ambiente de Desenvolvimento de Software

Rodrigo Fernandes Calhau, Ricardo de Almeida Falbo

Departamento de Informática, Universidade Federal do Espírito Santo, Vitória, Brasil

rodrigocalhau@gmail.com, falbo@inf.ufes.br

Resumo. *Ambientes de Desenvolvimento de Software buscam integrar ferramentas de apoio a diversas atividades do processo de software. Essas ferramentas podem ter sido desenvolvidas no contexto de um mesmo projeto ou por diferentes grupos. Este artigo discute uma abordagem baseada em ontologia para a integração de GCS-ODE, a ferramenta de apoio ao processo de Gerência de Configuração de Software do ambiente ODE, com o sistema de controle de versão Subversion.*

Abstract. *Software Engineering Environments aim to integrate tools supporting several activities of the software process. These tools could be developed inside the same project or by different groups. This paper discusses an ontology-based approach for integrating GCS-ODE, the Software Configuration Management system of ODE (Ontology-based software Development Environment), and Subversion, a version control system.*

1. Introdução

Ferramentas CASE são amplamente utilizadas para apoiar a realização de atividades do processo de software. Entretanto, cada uma delas ferramentas geralmente apóia uma ou um conjunto de atividades relacionadas, resolvendo apenas problemas pontuais, o que implica na utilização de várias ferramentas para apoiar o processo como um todo. A utilização de várias ferramentas sem um certo grau de integração entre elas traz diversos problemas como retrabalho, inconsistência etc. Visando à integração de ferramentas, surgiram os Ambientes de Desenvolvimento de Software (ADSs), que buscam prover ambientes capazes de apoiar todo o processo de software ou pelo menos porções significativas dele [Harrison et al. 2000].

No desenvolvimento de ADSs, duas abordagens podem ser seguidas: desenvolvimento integrado de ferramentas no contexto de um mesmo projeto ou integração de ferramentas desenvolvidas isoladamente. No primeiro caso, um mesmo grupo desenvolve diversas ferramentas de forma integrada, ampliando continuamente as funcionalidades do ambiente. No segundo, busca-se integrar ferramentas produzidas por diferentes grupos, de modo a prover um ambiente de apoio ao processo de software.

Deve-se observar que essas duas abordagens não são opções excludentes, mas sim complementares. Ou seja, um determinado grupo pode desenvolver diversas ferramentas integradas no contexto de um projeto e integrar outras produzidas por

outros grupos. Essa abordagem híbrida tem sido empregada no contexto de diversos projetos de ADSs, tal como o ambiente WebAPSEE [Sales et al. 2007].

Porém, integrar ferramentas, sobretudo aquelas desenvolvidas por diferentes grupos, não é trivial. Um fator crucial para a integração é a existência de um entendimento comum do significado dos termos manipulados pelas ferramentas. Para tal, é preciso ter uma representação compartilhada da conceituação do domínio de interesse. Ontologias podem ser utilizadas para esse fim, servindo como uma interlíngua para se mapear conceitos usados em diferentes ferramentas, (acesso a dados via uma ontologia compartilhada) ou como base para a construção delas (ontologias como especificação) [Jasper e Uschod 1999].

No contexto do Projeto ODE (*Ontology-based software Development Environment*) [Falbo et al. 2003], que visa ao desenvolvimento de um Ambiente de Desenvolvimento de Software Centrado em Processo, ontologias do domínio de Engenharia de Software vêm sendo usadas para apoiar a construção de ferramentas de apoio ao processo de software. A premissa por detrás do uso de ontologias em ODE é a seguinte: se as ferramentas de um ADS são construídas baseadas nas mesmas ontologias, a integração entre elas é facilitada, pois os conceitos envolvidos são compartilhados.

Até a realização deste trabalho, apenas ferramentas produzidas no contexto do próprio projeto, ditas ferramentas internas, compunham o ambiente, em uma abordagem de desenvolvimento integrado de ferramentas no contexto de um mesmo projeto. Contudo, percebeu-se que essa abordagem não é suficiente. Constatou-se que dificilmente seria possível prover as funcionalidades oferecidas por diversas ferramentas, devido às limitações de recursos do Projeto ODE. Mais além, que era um contra-senso reinventar a roda, dado que atualmente há diversas ferramentas bastante poderosas, livres e de código aberto, que poderiam ser incorporadas ao ambiente, tais como ferramentas de apoio à modelagem UML e sistemas de controle de versão. Por fim, mesmo que o ambiente oferecesse uma ferramenta interna para apoiar uma dada atividade, é muito natural que uma organização, mesmo adotando ODE, queria continuar a usar uma outra ferramenta, externa ao ambiente, para apoiar essa mesma atividade, fazendo com que ODE tenha que interoperar com ela. Assim, passou-se a investigar o uso de ontologias para apoiar a integração de ferramentas externas ao ambiente ODE.

Dentre as ferramentas que devem compor um ADS, estão aquelas de apoio à Gerência de Configuração de Software (GCS). Existem diversos sistemas de apoio à GCS livres e de código aberto disponíveis, porém grande parte deles com foco apenas no controle de versão. Por outro lado, em ODE havia uma ferramenta de apoio à GCS [Nunes e Falbo 2006] que visava oferecer serviços básicos de controle de versão e de controle de alteração. Contudo, as funcionalidades providas, sobretudo para o apoio ao controle de versões, eram bastante limitadas quando comparadas a de sistemas como o CVS (*Concurrent Version System*) [Caetano 2004] e o Subversion (SVN) [Nagel 2005]. Assim, decidiu-se integrar o SVN a ODE, usando uma ontologia de gerência de configuração como uma interlíngua e dando origem a uma nova versão da ferramenta de apoio à GCS, denominada agora GCS-ODE [Calhau et al. 2008].

Este artigo discute a integração baseada em ontologia de GCS-ODE com o SVN, mostrando como a utilização de ontologias pode facilitar a integração de ferramentas de apoio a um mesmo processo (no caso o processo de GCS), provendo um apoio mais efetivo para ele. Nessa integração, foi feita a extração do modelo conceitual do SVN, cujos conceitos foram mapeados para os conceitos da ontologia de GCS na qual GCS-ODE é baseada. Tal mapeamento foi utilizado na integração das duas ferramentas.

Este artigo está organizado da seguinte forma: a seção 2 discute brevemente GCS e integração de ferramentas; a seção 3 dá uma visão geral de ODE e de GCS-ODE; a seção 4 discute a integração entre SVN e GCS-ODE; a seção 5 discute trabalhos relacionados; e, por fim, a seção 6 apresenta as considerações finais deste trabalho.

2. Gerência de Configuração de Software e Integração de Ferramentas

A Gerência de Configuração de Software (GCS) é uma disciplina para controlar a evolução dos sistemas de software [Estublier 2000]. Ela busca, dentre outros, identificar os artefatos que podem ser modificados, ditos itens de configuração (ICs), estabelecer relações entre eles e mecanismos para administrar suas diferentes versões, controlar as modificações etc [IEEE 2004].

Dois dos principais processos da GCS são o controle de alterações e o controle de versões. O controle de alterações combina procedimentos humanos e ferramentas automatizadas para gerenciar alterações realizadas nos ICs. Quando uma alteração é solicitada, o impacto em outros ICs e o custo da modificação devem ser avaliados e um responsável deve decidir se a alteração deve ser realizada ou não. Caso a alteração seja aprovada, pessoas são indicadas para a sua execução. Para realizar uma alteração, cópias dos ICs relacionados são feitas a partir do repositório e colocadas em uma área de trabalho, em um procedimento denominado *checkout*. Os desenvolvedores designados fazem as alterações necessárias e, assim que essas forem concluídas, os itens são submetidos a uma revisão. Se as alterações forem aprovadas, os itens são devolvidos ao repositório central, em um procedimento denominado *checkin* [Pressman 2006].

O controle de versão, por sua vez, visa identificar, armazenar e administrar diferentes versões dos ICs, criadas durante o processo de software [Pressman 2006]. A cada modificação em um IC, uma nova versão é criada.

Diferentes tipos de funcionalidades são necessários para apoiar um processo de GCS, tais como [IEEE 2004]: uma biblioteca de itens, procedimentos de solicitação e aprovação de alterações, suporte a diferentes tipos de artefatos, relato do estado de configuração e gerência de construção (*building*) e de liberações.

Existem diversos sistemas de apoio à GCS disponíveis. Os mais populares são focados no controle de versão, tais como o Subversion (SVN) [Nagel 2005] e CVS [Caetano 2004]. Neles, arquivos são armazenados em um repositório central, permitindo que vários usuários tenham acesso. Para se fazer alterações, são providas cópias de trabalho, que são pastas contendo cópias de arquivos do repositório. Nas cópias de trabalho, arquivos podem ser alterados, removidos ou adicionados. As cópias de trabalho são criadas no momento do *checkout* e suas alterações são registradas no momento de um *checkin*. Para se poder efetuar um *checkin*, a cópia de trabalho deve estar atualizada, ou seja, deve conter as alterações registradas por outros

desenvolvedores ocorridas durante o período compreendido entre o *checkout* e o *checkin* da alteração sendo processada. Ao se fazer uma atualização, alterações registradas no repositório são mescladas com as alterações da cópia de trabalho.

Porém o apoio oferecido por esses sistemas a outras atividades da GCS é limitado. No que se refere ao controle de alteração, tais sistemas geralmente dão apoio apenas ao controle de sincronização e ao controle de acesso, deixando de lado a maior parte dos aspectos gerenciais, tal como o controle de solicitações de alteração.

Por outro lado, sistemas de acompanhamento de questões (*issue tracking*), tal como o TRAC [Trac 2008], podem ser utilizados no apoio ao controle de alteração, podendo, inclusive, serem integrados a ferramentas de controle de versão. Por meio desses sistemas, é possível registrar questões (*issues*), que podem ser melhorias a serem feitas, reparo a erros etc, designar uma pessoa responsável por tratar a questão e definir uma prioridade para ela. Contudo, tais ferramentas ainda não oferecem um apoio completo ao controle de alteração, deixando de lado funcionalidades importantes, tal como a análise de impacto de uma alteração.

Pode-se notar que, normalmente, as ferramentas de apoio à GCS enfocam apenas uma parte do processo. Assim, as funcionalidades necessárias para apoiar a GCS são obtidas com uma combinação de ferramentas computadorizadas, procedimentos manuais, funcionalidades de integração entre ferramentas ou com ambientes integrados [IEEE 2004]. Este cenário leva a um importante requisitos para as ferramentas de apoio à GCS: interoperabilidade [Arantes et al. 2007].

Algum tipo de interoperabilidade pode ser conseguido com programas de tradução, que permitem a comunicação de uma ferramenta específica com outra. Entretanto, essa abordagem é limitada, pois à medida que aumenta o número de ferramentas a serem integradas e a informação torna-se mais complexa, é mais difícil prover tradutores para cada par de ferramentas [Schlenoff et al. 2000]. Isso é um problema para o caso da GCS, especialmente quando tratada no contexto de um Ambiente de Desenvolvimento de Software (ADS).

Obstáculos à interoperabilidade surgem, principalmente, do fato das ferramentas serem geralmente criadas de forma independente e não compartilharem a mesma semântica para a terminologia usada para descrever o domínio de interesse. Diferentes termos podem estar sendo usados para representar o mesmo conceito. Sem uma definição explícita do significado dos termos envolvidos é difícil saber a correspondência entre os conceitos em uma ferramenta em relação às outras. Por outro lado, simplesmente compartilhar a terminologia não é suficiente. As ferramentas precisam compartilhar semântica, isto é, os significados de suas respectivas terminologias [Schlenoff et al. 2000].

Seja o exemplo dos sistemas de controle de versão SVN e CVS. Mesmo sendo o SVN uma extensão do CVS, eles possuem algumas diferenças de conceitos. No CVS, por exemplo, existe o conceito de ramificação e marcação (*tag*). Já no SVN, tais conceitos são tratados indiretamente utilizando-se do conceito de pasta. Para um usuário criar uma ramificação ou marcação no SVN, ele deve fazer uma cópia da pasta que deseja marcar ou ramificar. Outra diferença entre os conceitos presentes nessas ferramentas é o conceito de revisão, presente no SVN. Uma revisão de um repositório é

representada por um número e indica um estado dele. A cada alteração, esse número é alterado, indicando a evolução do estado do repositório. Já no CVS, cada arquivo possui seu próprio número que indica a sua evolução isoladamente.

De fato, a integração de ferramentas está diretamente relacionada ao nível de concordância semântica entre as mesmas, o que, no caso de ADSs, envolve diversas dimensões, tais como apresentação, dados e controle [Ruy e Falbo 2006]. A dimensão de apresentação se refere à interação com o usuário e está diretamente relacionada ao grau com que pessoas e sistemas concordam com o significado das interfaces. As dimensões de dados e de controle também são extremamente dependentes de semântica, uma vez que as ferramentas devem possuir o mesmo entendimento a respeito das estruturas lógicas e de dados (integração de dados) e dos serviços providos (integração de controle).

Para lidar com a questão da integração semântica, ontologias podem ser utilizadas para estabelecer um entendimento comum sobre o domínio de Engenharia de Software, servindo como uma interlíngua para a comunicação entre as ferramentas. Neste cenário, é necessário mapear os conceitos específicos das ferramentas para uma ontologia de referência, de modo que uma ferramenta se comunique com outra traduzindo os conceitos específicos daquela para conceitos que todas concordam (ontologia) e depois os traduzindo novamente para os seus conceitos.

3. Gerência de Configuração de Software no Ambiente ODE

ODE (*Ontology-based software Development Environment*) [Falbo et al. 2003] é um ADS desenvolvido tomando por base ontologias do domínio de Engenharia de Software, que são usadas como base para a construção de ferramentas integradas..

Dentre as ferramentas de ODE está GCS-ODE, a ferramenta de apoio à GCS, foco deste trabalho. Sua versão inicial [Nunes e Falbo 2006] foi construída com base em uma ontologia de gerência de configuração que cobria os conceitos envolvidos em algumas das atividades da GCS, tais como a identificação de itens de configuração e o controle de solicitações de alteração. Entretanto, faltavam alguns conceitos importantes, sobretudo os relacionados com o controle de versão e de alteração. Conceitos como repositório, ramificação e cópia, presentes na maioria dos sistemas de GCS, não eram considerados na ontologia. De modo a preencher esta lacuna, Arantes et al. (2007) evoluíram essa ontologia. A Figura 1 mostra parte do modelo conceitual da nova versão da ontologia. A ontologia completa está apresentada em [Arantes et al. 2007].

De acordo com a nova versão da ontologia de GCS, itens de configuração derivam de artefatos e possuem versões, as quais podem ser submetidas a alterações. Uma retirada (*checkout*) é feita para uma alteração, gerando cópias das versões em alteração. Quando concluído o trabalho, um *checkin* é feito, relacionando as cópias que dão origem a novas versões. Toda versão está localizada em uma ramificação dentro de um repositório, podendo pertencer, ainda, a uma linha base (*baseline*). Uma versão pode substituir uma outra (revisão) ou pode coexistir com a outra (variante).

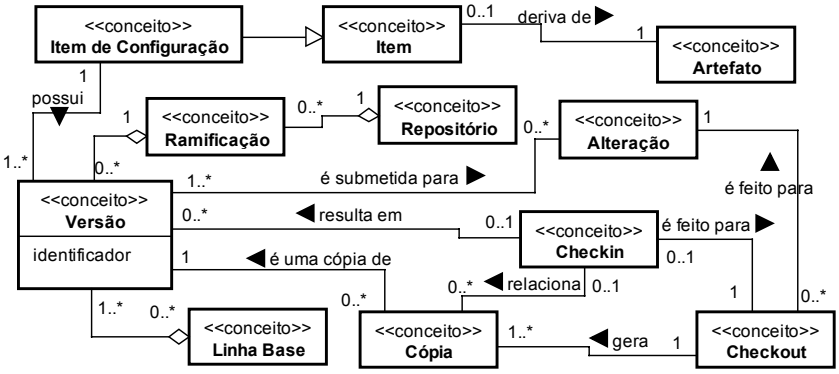

Figura 1 – Excerto da Ontologia de Gerência de Configuração de Software.

Em relação à ferramenta de apoio à GCS de ODE, sua primeira versão procurava oferecer serviços básicos de controle de versão e de alteração, sendo as funcionalidades providas para o apoio ao controle de versões bastante limitadas quando comparadas a de outros sistemas, como o CVS (*Concurrent Version System*) [Caetano 2004] ou o Subversion (SVN) [Nagel 2005].

A evolução da ontologia que a fundamentava levou também à evolução da ferramenta. Neste momento, decidiu-se integrar a ferramenta de apoio à GCS de ODE, denominada agora GCS-ODE, ao SVN, remodelando e reconstruindo diversas funcionalidades. Na atual versão [Calhau et al. 2008], GCS-ODE permite que solicitações de alteração sejam feitas por desenvolvedores. Tais solicitações dão origem a alterações, se aprovadas pelo gerente de configuração. Quando o *checkin* é feito, o gerente de configuração pode visualizar as alterações realizadas e verificar se elas estão de acordo com o que foi estabelecido previamente, autorizando ou não o registro da alteração. A Figura 2 mostra o modelo de classes de GCS-ODE.

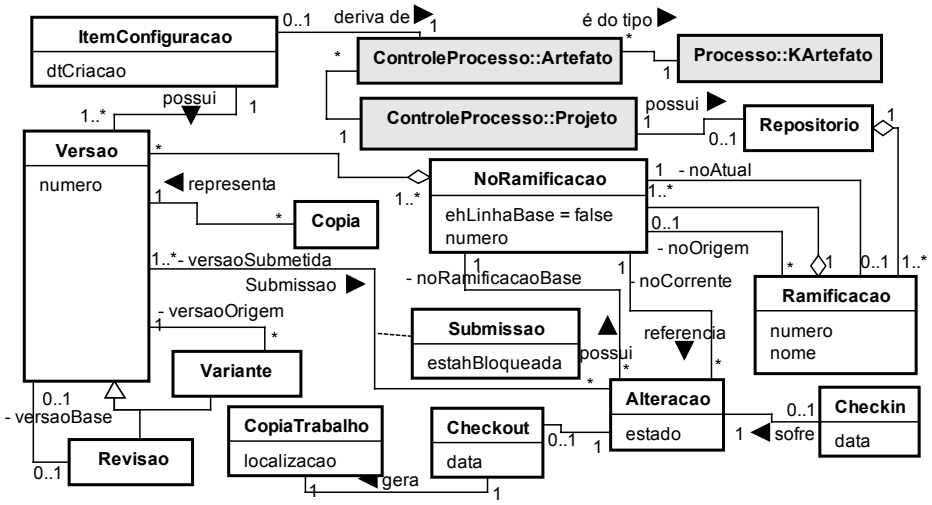

Figura 2 – Modelo de Classes de GCS-ODE.

Como a ferramenta foi baseada na ontologia proposta em [Arantes et al. 2007], seu modelo de classes é bastante similar ao modelo conceitual dessa ontologia, porém

com algumas particularidades. Uma das principais diferenças são os conceitos de nó de ramificação e cópia de trabalho, presentes no modelo de classes da ferramenta. Um repositório é composto de ramificações, que, por sua vez, são compostas de nós de ramificação. Estes são compostos de versões de ICs. Dentre os nós de uma ramificação, um deles é o nó mais atual, i.e., aquele que contém as últimas alterações processadas. À medida que alterações vão sendo feitas, novos conjuntos de versões vão sendo formados, substituindo os antigos. Maiores detalhes sobre GCS-ODE podem ser encontrados em [Calhau et al. 2008] e [Calhau e Falbo 2008].

Para a ferramenta executar as funcionalidades do SVN, bem como acessar informações no seu repositório ou em alguma cópia de trabalho, foi utilizada a biblioteca *svnKit* [SVNKIT 2008], uma biblioteca livre em Java que oferece a maior parte das funcionalidades do SVN. Essa biblioteca foi uma grande facilitadora da integração, fazendo o papel da própria ferramenta SVN. Apesar das facilidades providas por *svnKit*, a integração não se deu de modo direto. Antes houve uma análise dos conceitos tanto de SVN quanto de ODE para que a integração fosse feita. Essa integração é o foco principal deste trabalho e é discutida em maiores detalhes na próxima seção.

4. Integrando o Subversion a GCS-ODE

A integração entre o SVN e GCS-ODE ocorreu em dois níveis: dados e controle. Para tal, foi criado um adaptador, responsável por traduzir a linguagem entendida por GCS-ODE, no caso a ontologia, para a linguagem entendida pelo SVN e vice versa, como ilustra a Figura 3. A ferramenta GCS-ODE é uma ferramenta interna a ODE e, portanto, compartilha os conceitos da ontologia. Já o SVN possui seus próprios conceitos. O adaptador é responsável por traduzir os conceitos pertencentes à ontologia e ao SVN para que as ferramentas possam se comunicar de maneira adequada.

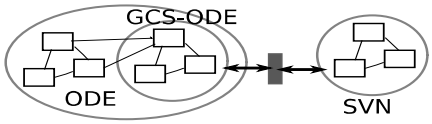

Figura 3 – Conversão de Conceitos entre GCS-ODE e SVN.

Porém, como os conceitos específicos do SVN não eram conhecidos, foi necessário extraí-los para a criação desse adaptador, explicitando seu modelo conceitual. Para a extração do modelo conceitual do SVN, foram analisados os meta-dados das cópias de trabalho e os arquivos com o esquema XML de saída de linha de comando do SVN. Além disso, foi utilizado o método para extração do modelo conceitual de ferramentas denominado *Ontological Excavation* [Hsi 2005], que toma por base a interface gráfica de uma ferramenta para extrair e relacionar seus conceitos. Como o SVN é uma ferramenta originariamente de linha de comando, foi utilizado o cliente SVN TortoiseSVN [TortoiseSVN 2008]. Além disso, foi analisado o código fonte do próprio SVN, já que ele é uma ferramenta de código aberto. A Figura 4 mostra parte do modelo conceitual extraído.

Um repositório (*Repos*) é composto de nós (*ReposNode*), que representam versões de arquivos (*FileNode*) ou de diretórios (*DirNode*). Um nó dá origem a outro quando sofre uma alteração. Assim, se um nó de repositório for um nó folha (*LeafNode*), então ele ainda não sofreu alterações; caso contrário, ele será um

ParenteNode. Dessa forma, nós de repositório relacionam-se formando grafos que representam a evolução de um arquivo ou diretório. Um nó pode ser uma cópia de outro (*CopiedNode*). Um repositório possui estados, chamados de revisão (*Revision*), sendo que cada um deles é identificado por um número. Cada revisão é gerada por um *Commit*, que registra alterações feitas em algum item de cópia de trabalho (*WCItem*), que pode ser um arquivo (*FileItem*) ou um diretório (*DirItem*). Todo item de cópia de trabalho é uma cópia de um nó do repositório, podendo possuir alterações locais ou não.

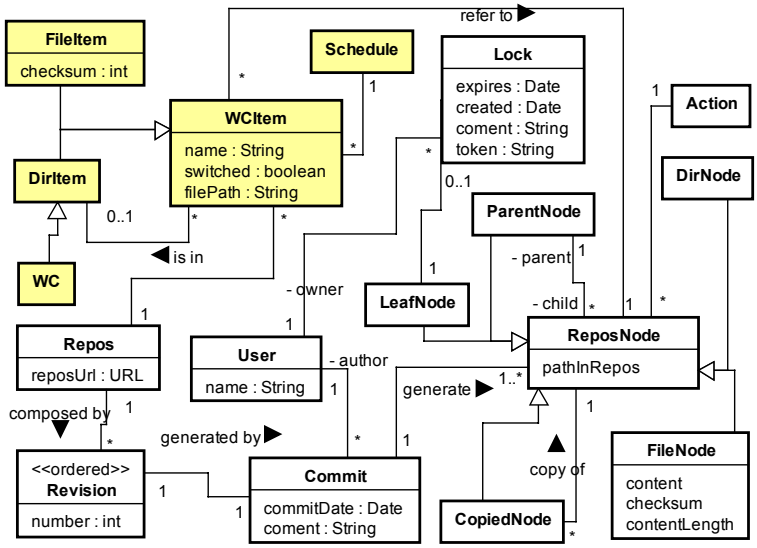

Figura 4 – Parte do Modelo Conceitual Extraído do SVN.

As informações referentes aos conceitos extraídos do SVN são armazenadas em dois lugares distintos: em cópias de trabalho e em repositórios. Na Figura 4, os conceitos localizados em cópias de trabalho estão destacados. Essas informações ficam localizadas em arquivos de meta-dados, que são responsáveis por guardar as informações referentes aos arquivos sob controle de versão em uma cópia de trabalho. Tais meta-dados informam, por exemplo, a qual versão de arquivo no repositório um arquivo da cópia de trabalho referencia, se esse arquivo sofreu modificações locais, se o mesmo foi adicionado ao controle de versão, se está em conflito devido a alguma atualização etc.

Depois de extraído o modelo conceitual do SVN, ele foi comparado com a ontologia de GCS e então com o modelo de classes de GCS-ODE. Dessa forma, foram identificados quais os conceitos da ontologia que correspondem aos conceitos do SVN e de GCS-ODE. A Tabela 1 apresenta o correspondente mapeamento.

Tabela 1 – Mapeamento parcial entre os conceitos da ontologia, SVN e GCS-ODE.

Ontologia	SVN	GCS-ODE
Versão	*FileNode*	Versão
Ramificação	*DirNode*	Ramificação
Linha Base	*Revision*	Nó de Ramificação (do tipo Linha Base)
Cópia	*FileItem*	Cópia
Checkin	*Commit*	*Checkin*
Variante	*CopiedNode*	Variante

Uma *Versão* de um *Item de Configuração* da ontologia, por exemplo, corresponde a um *FileNode*, que representa uma versão de um arquivo no SVN. Uma *Ramificação* corresponde a um *DirNode*, que é uma pasta no repositório SVN. Já o conceito *Cópia* da ontologia corresponde ao conceito *FileItem*, um arquivo em uma cópia de trabalho no SVN.

É importante realçar que, da mesma forma que pode ocorrer o uso de diferentes termos para designar um mesmo conceito, também podem aparecer conceitos diferentes com a mesma denominação. Esse é o caso do conceito *Revisão*, que aparece tanto na ontologia quanto no SVN, mas com significados distintos. Além disso pode ocorrer de as ferramentas possuírem um mesmo conceito, mas a ontologia usada para a integração não possuí-lo. Isso ocorreu com o conceito cópia de trabalho, presente tanto no SVN quanto em GCS-ODE, mas não na ontologia.

O modelo de projeto de GCS-ODE foi elaborado com base no mapeamento descrito acima. A partir dele, decidiu-se quais classes seriam efetivamente implementadas em GCS-ODE e quais não seriam, pois representavam informações disponíveis no SVN. Em outras situações, foram adicionadas informações relativas ao mapeamento com informações do SVN. Por exemplo, a classe `Ramificacao` representa uma ramificação em GCS-ODE e uma pasta no repositório do SVN. Assim foi adicionado a ela um atributo relativo à localização da pasta do repositório que ela representa. Já na classe `Versao` foram adicionados atributos referentes à versão de um arquivo no repositório. Dessa forma, os objetos das classes de GCS-ODE são encontrados de três lugares: no próprio ambiente, nas cópias de trabalho criadas pelo SVN e no repositório SVN. Esse mapeamento entre conceitos apoiou, portanto, a integração no nível de dados.

Com relação à integração de controle, GCS-ODE utiliza as funcionalidades fornecidas pelo *svnkit*. Esse nível de integração é importante para manter a integridade entre as informações relativas a conceitos replicados, presentes tanto em GCS-ODE quanto no SVN. A Figura 5 ilustra um exemplo dessa situação. Uma versão *v1* de um item de configuração no banco de dados do ambiente ODE representa uma versão *f1* de um arquivo localizado no repositório SVN. O arquivo *a*, pertencente à cópia de trabalho *c.p.1*, representa, inicialmente, uma cópia de *f1* que possui alterações locais. No registro das alterações da cópia de trabalho *c.p.1*, GCS-ODE manipula as informações no banco de dados de ODE e usa o *svnkit* para manipular as informações contidas na cópia de trabalho *c.p.1* e no repositório SVN. Neste caso, GCS-ODE cria uma nova versão *v2* e solicita ao *svnkit* que efetue o registro da alteração. O *svnkit,* por sua vez, altera o estado de *a* para indicar que suas alterações foram registradas, cria uma nova versão *f2* do arquivo no repositório e faz *a* referenciar *f2*.

5. Trabalhos Correlatos

No que se refere a trabalhos de integração usando ontologias, há diversos relatos na literatura, grande parte deles no contexto da *web* semântica. No entanto, optamos por discutir nesta seção dois trabalhos relacionados de alguma forma com ferramentas de GCS, a saber a integração de sistemas de controle de versão no ambiente WebAPSEE [Sales et al. 2007] e ModelCVS [Kappel 2006], um sistema que fornece uma infra-

estrutura semântica para apoiar a integração de ferramentas de modelagem, tendo como base um sistema de controle de versão.

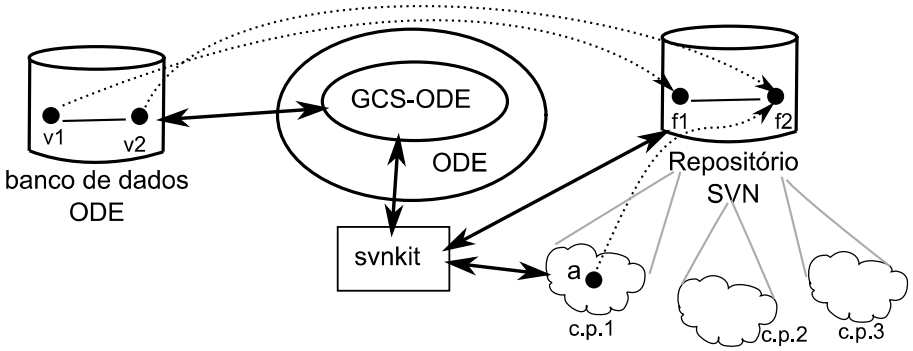

Figura 5 – Localização dos Objetos de GCS-ODE.

WebAPSEE é um ADS centrado em processos, que fornece apoio a atividades da GCS. Como em GCS-ODE, os dados referentes a artefatos (meta-dados) são armazenados em um banco de dados e o conteúdo deles é armazenado em um repositório de artefatos. Para fazer o controle das versões desses artefatos, pode-se utilizar tanto o SVN quanto o CVS. Para tal, foram definidas interfaces de comunicação para acessar os serviços de tais ferramentas, que são acessadas de modo transparente ao usuário, como em GCS-ODE. Para cada ferramenta de controle de versão integrada a WebAPSEE, um componente de gerência de artefatos é provido, em um mapeamento aos pares. A integração, portanto, ocorre tanto no nível de dados quanto de funcionalidades, como em GCS-ODE. Contudo, em GCS-ODE, utilizou-se uma abordagem baseada em ontologia. Com ela, espera-se poder facilitar a integração com outras ferramentas (por exemplo, o TRAC), na medida em que a ontologia atua como uma interlíngua entre as várias ferramentas.

ModelCVS visa combinar diferentes tipos de ferramentas de modelagem, tais como ferramentas de criação, simulação e verificação, e geração de código a partir de modelos, permitindo que modelos de uma ferramenta sejam transformados em modelos de outras ferramentas de modo transparente. Para isso, esse sistema usa mais de um nível de mapeamento. ModelCVS faz o mapeamento entre os meta-modelos dos modelos gerados pelas ferramentas, bem como entre os modelos conceituais dessas ferramentas. A transformação de modelos é feita a partir do mapeamento entre meta-modelos, que, por sua vez, é derivado do mapeamento entre os modelos conceituais das ferramentas. Esse mapeamento pode ser feito de modo direto ou indireto. No modo indireto, é feito usando uma ontologia de mais alto nível e tomando por base uma biblioteca com mapeamentos já elaborados.

Assim, como neste trabalho, os conceitos das ferramentas também devem ser extraídos. Para facilitar a integração, o núcleo do sistema é baseado no sistema de controle de versão CVS, que controla as versões dos modelos considerando sua sintaxe e semântica. Deve-se ressaltar que, diferentemente deste trabalho que visa apoiar a GCS, ModelCVS apóia a modelagem de sistemas. Além disso, a integração realizada neste trabalho foi mais específica, integrando apenas duas ferramentas. Já ModelCVS permite

que mais de uma ferramenta seja integrada. Por outro lado, a integração entre ferramentas em ModelCVS ocorre apenas no nível de dados (modelos), enquanto que neste trabalho também há integração de controle. Por fim, outro ponto que distingue os trabalhos é que neste é utilizada uma ontologia de domínio para a integração.

6. Conclusões

Este artigo apresentou a integração de ferramentas de apoio à GCS em ODE, segundo uma abordagem baseada em ontologias. As ferramentas integradas foram o sistema de controle de versão SVN e a ferramenta GCS-ODE. Para essa integração, foi feita a extração do modelo conceitual do SVN e seu mapeamento com os conceitos da Ontologia de GCS que fundamenta GCS-ODE. O sistema de GCS resultante da integração provê um apoio mais amplo ao processo de GCS quando comparado com ferramentas isoladas, devido à união das funcionalidades de controle de versão providas pelo SVN com as funcionalidades de controle de alteração providas por GCS-ODE.

Como trabalho futuro pretende-se sistematizar o processo de integração de ferramentas externas ao ambiente ODE. Inicialmente esperam-se integrar ferramentas livres, o que facilita a extração dos modelos conceituais das mesmas, visto que o código fonte é acessível. O mapeamento de conceitos deve seguir a abordagem usada neste trabalho, porém a criação de adaptadores para a integração pode variar dependendo da ferramenta a ser integrada. No caso deste trabalho, a ferramenta foi integrada usando uma biblioteca (*svnkit*), o que permitiu a integração nos níveis de dados e de controle. Já para o caso de integração de ferramentas em outras linguagens, por exemplo, a criação de adaptadores se daria de modo mais complexo, podendo prover integração apenas no nível de dados. A partir desses trabalhos, espera-se definir uma abordagem baseada em ontologias para a integração de ferramentas em geral, definindo passos e orientações a serem seguidas.

Agradecimentos

Este trabalho foi realizado com o apoio do CNPq, da Petrobrás, que possui um convênio com o programa de bolsas de iniciação científica da UFES, e das empresas VixTeam Consultoria & Sistemas e Projeta, parceiras que têm apoiado o projeto.

Referências

Arantes, L.O., Falbo, R.A., Guizzardi, G. (2007) "Evolving a Software Configuration Management", Second Workshop on Ontologies and Metamodeling Software and Data Engineering – WOMSDE´2007, João Pessoa, Brasil.

Caetano, C. (2004) CVS: *Controle de Versões e Desenvolvimento Colaborativo de Software*. Editora Novatec.

Calhau, R.F., Arantes, L.O., Falbo, R.A. (2008) "Uso de Gerência de Conhecimento para Apoiar a Rastreabilidade e a Avaliação de Impacto de Alterações", XXII Simpósio Brasileiro de Engenharia de Software – SBES´2008, Campinas, Brasil.

Calhau, R.F., Falbo, R.A. (2008) "GCS-ODE: Uma Ferramenta de Apoio à Gerência de Configuração de Software Integrada ao Ambiente ODE", Sessão de Ferramentas do XXII Simpósio Brasileiro de Engenharia de Software, Campinas, Brasil.

Estublier, J. (2000), "Software Configuration Management: A Roadmap", In: Proc. of the Future of Software Engineering, ICSE'2000, Ireland.

Falbo, R. A., Natali, A. C. C., Mian, P.G., Bertollo, G., Ruy, F.B. (2003) "ODE: Ontology-based software Development Environment", In: Memórias de IX Congreso Argentino de Ciencias de la Computación, p. 1124-1135, La Plata, Argentina.

Harrison, W., Ossher, H., Tarr, P. (2000), "Software Engineering Tools and Environments: A Roadmap". In: Proceedings of the Conference on the Future of Software Engineering – ICSE´2000, 261-277, Limerick, Ireland.

Hsi, I. (2005) Analyzing the Conceptual Integrity of Computing Applications Through Ontological Excavation, PhD Thesis, USA.

IEEE (2004) *SWEBOK - Guide to the Software Engineering Body of Knowledge*, 2004 Version, IEEE Computer Society.

Jasper, R., Uschold, M. (1999) "A framework for understanding and classifying ontology applications". IJCAI-99, *Ontology Workshop,* Stockholm.

Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M. (2006) "On Models and Ontologies - A Layered Approach for Model-based Tool Integration". In Mayr, H. C. and Breu, R., editors, Modellierung, p. 11–27.

Nagel, W. (2005) *Subversion Version Control: Using The Subversion Version Control System in Development Projects*, Prentice Hall PTR.

Nunes, V.B., Falbo, R.A. (2006) "Uma Ferramenta de Gerência de Configuração Integrada a um Ambiente de Desenvolvimento de Software", V Simpósio Brasileiro de Qualidade de Software, Vila Velha - ES.

Pressman, R. S. (2006), *Engenharia de Software*, Mc Graw Hill, 6a edição.

Ruy, F. B.,Falbo, R. A. (2006) "Tratamento Semântico de Conhecimento Organizacional em um Ambiente de Desenvolvimento de Software". 1st Workshop on Ontologies and Metamodeling in Software and Data Engineering, Florianópolis, Brasil.

Sales, E., Reis, C. A. L., Lima, A. M. (2007) "Gestão de Configuração integrada a Gerência de Processos de Software no Ambiente WebAPSEE", XXXIII Conferência Latinoamericana de Informática (CLEI 2007), San José, Costa Rica.

Schlenoff, C., Gruninger, M., Tissot, F., Valois, J., Lubell, J., Lee, J., (2000), The Process Specification Language (PSL) Overview and Version 1.0 Specification, NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD.

Svnkit. "Subversion for Java". Disponível em: < http://www.svnkit.com >. Acesso em: 10 jul. 2008.

TortoiseSVN. Disponível em: < http://tortoisesvn.tigris.org/ >. Acesso em: 10 jul. 2008.

Trac. Disponível em: < http://trac.edgewall.org/ >. Acesso em: 10 jul. 2008.