

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Integração do Sistema de Controle de Versão Subversion (SVN) com
o IDE NetBeans

Lucio Moratelli Prado

Florianópolis – SC

2006/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Integração do Sistema de Controle de Versão Subversion (SVN) com
o IDE NetBeans

Lucio Moratelli Prado

Trabalho de conclusão de curso
apresentado como parte dos requisitos para
obtenção do grau de Bacharel em Sistemas de
Informação.

Florianópolis – SC

2006/2

Resumo

A qualidade de software é um dos assuntos mais atuais em discussão na comunidade de Engenharia de Software. Embora os primeiros esforços mais amplos no sentido de se produzir software com maior qualidade e produtividade datem da década de 70, foi na segunda metade da década de 90 que uma série de novos conceitos e abordagens alcançaram maturidade e visibilidade.

Devido ao grande número de elementos, ou itens de configuração, geralmente envolvidos num desenvolvimento de software, torna-se essencial a utilização de ferramentas que auxiliem na realização das diversas tarefas envolvidas no processo de desenvolvimento de software. Os Ambientes Integrados de Desenvolvimento (IDE – Integrated Development Environment) surgiram para apoiar esse processo através da integração dessas diversas ferramentas.

Dentre as ferramentas destacam-se as de controle de versão, cujos objetivos são: armazenar os itens que são produzidos no desenvolvimento, permitir o acesso de maneira controlada a quaisquer versões destes elementos, armazenar informações de histórico dos itens de forma a estabelecer a base para o controle da evolução do produto de software, e reter informações que permitam automatizar o acesso aos conjuntos de itens de configuração que compõem o produto de software num específico estado de seu desenvolvimento.

Este trabalho apresenta uma proposta de um *plug-in* que implemente a integração entre um dos ambientes integrados de desenvolvimento mais utilizados para desenvolvimento de software JAVA, o NetBeans, com uma ferramenta de

controle de versão que vem ganhando espaço na comunidade *open source*, o Subversion.

Palavras-Chaves: NetBeans, Subversion, ambiente integrado de desenvolvimento de software, gerência de configuração, controle de versão, *plug-in*.

Abstract

The software quality is one of the most recent subjects in the Software Engineering community discussion. Although the first ample efforts in the direction of producing software with bigger quality and productivity date on 70 decade, was in the second half of 90 decade that a series of new concepts and boardings had reached maturity and visibility.

Had the great number of elements, or configuration items, generally involved in a software development, the tools utilization becomes essential that assist in the accomplishment of the diverse involved tasks in the development software process. The Integrated Development Environments (IDE) had appeared to support this process through the tools integration.

Amongst the tools they are distinguished of version control, whose objectives are: to store items that they are produced in the development, to allow the access in controlled way to any versions of these elements, to store information of items description to establish the base for the evolution control of software's product, and to hold back information that allow to automatize the access to sets of configuration items that compose the software's product in a specific development state.

This work presents one proposal plug-in that implements the integration between one of most used IDEs for software development JAVA, the NetBeans, with a tool of version control that comes earning space in the open source community, the Subversion.

Keywords: IDE, SVN, NetBeans, Software Configuration Management, Version Control, plug-in, integrated development environment.

Sumário

Resumo	3
Abstract.....	5
Lista de Figuras.....	10
Lista de Tabelas	13
Lista de Abreviaturas.....	14
1 Introdução.....	15
1.1 Motivação	16
1.2 Objetivos.....	18
1.2.1 Objetivo Geral	18
1.2.2 Objetivo Específico.....	18
1.2.3 Escopo de Estudo	19
2 O Processo de Desenvolvimento de Software	20
2.1 Ambientes Automatizados de Apoio a Gerência do Processo de Software. 23	
2.1.1 Ambientes de Desenvolvimento de Software (ADS)	24
2.1.1.1 Integração de Ferramentas	26
2.1.1.2 Arquitetura	29
2.1.2 Software Configuration Management (SCM).....	32
2.1.2.1 Sistema de Controle de Versão (VCS - Version Control System)...	35
2.1.2.1.1 Termos.....	36
2.1.2.1.2 Atividades Comuns	37
2.1.2.1.3 Algumas Ferramentas de Controle de Versão Existentes.....	38

2.1.2.1.4 Comparação entre o CVS e o SVN.....	42
3 O Ambiente NetBeans.....	46
3.1 A Arquitetura do IDE NetBeans	47
3.2 O Desenvolvimento de Plug-ins no IDE NetBeans	48
3.2.1 As APIs Abertas (Open APIs)	49
3.2.2 Modularidade	50
3.2.3 Hierarquia, Arquivos e Nós	51
3.2.4 A Abstração de Arquivos no NetBeans	52
3.2.5 Mapeando Arquivos Para Objetos.....	54
3.2.6 O Sistema de Arquivos do NetBeans.....	55
3.2.7 Camadas dos Módulos.....	56
3.2.8 Camadas no Sistema de Arquivos	58
4 O <i>Plug-in</i> SVN Proposto	60
4.1 O Processo de Desenvolvimento do Plug-in Proposto	72
4.1.1 A Biblioteca Java SVN	76
4.1.2 Primeiro Ciclo de Desenvolvimento, o Caso de Uso <i>Checkout</i>	76
4.1.2.1 A Fase de Análise.....	77
4.1.2.2 A Fase de Projeto	79
4.1.2.3 A Fase de Implementação	87
4.1.3 Segundo Ciclo de Desenvolvimento, o Caso de Uso <i>Commit</i>	93
4.1.3.1 A Fase de Análise.....	93
4.1.3.2 A Fase de Projeto	95
4.1.3.3 A Fase de Implementação	97
4.1.4 Terceiro Ciclo de Desenvolvimento, o Caso de Uso <i>Update</i>	98

4.1.4.1 A Fase de Análise.....	99
4.1.4.2 A Fase de Projeto.....	99
4.1.4.3 A Fase de Implementação.....	100
5 Conclusões.....	101
5.1 Sugestões para Trabalhos Futuros.....	102
6 Referências Bibliográficas.....	103
Anexo 1 – Artigo.....	107
Anexo 2 – Artefatos UML.....	113
Anexo 3 – Código Fonte.....	124

Lista de Figuras

Figura 1 - Elementos envolvidos em atividades do processo de software [LIM 98].	21
Figura 2 - Níveis de integração de ferramentas em um ADS [BRO 92].	28
Figura 3 - Modelo de Referência ECMA [BRO 92a]	30
Figura 4 – MultiFileSystem [GRE 2002].	53
Figura 5 - Adicionando um .instance para o Sistema de Arquivos [GRE 2002]. ...	55
Figura 6 - Módulo adicionando uma camada XML no sistema de arquivos [GRE 2002].	57
Figura 7 - Estrutura do sistema de arquivos [GRE 2002].	59
Figura 8 - Localização do CVS na barra de menus do NetBeans.	60
Figura 9 - Localização do SVN na barra de menus do NetBeans.	61
Figura 10 - Ícones ilustrando as operações SVN.	61
Figura 11 - Primeira tela do Wizard de Checkout.	62
Figura 12 - O preenchimento dos campos obrigatórios habilita o botão <i>Next</i>	63
Figura 13 - Barra de progresso indicando a tarefa de conexão em curso.	63
Figura 14 - Exemplo de mensagem de erro na tentativa de conexão.	64
Figura 15 - Segunda tela do <i>Wizard de Checkout</i>	64
Figura 16 - Botões para acesso as janelas de auxílio no <i>Wizard de Checkout</i>	65
Figura 17 - Janela de exploração do repositório.	66
Figura 18 - Janela de escolha do diretório a ser feito o <i>checkout</i>	67
Figura 19 - Janela de pesquisa das versões de um artefato.	68

Figura 20 - Barra de progresso indicando o andamento da tarefa de <i>checkout</i>	69
Figura 21 - Exemplo de erro ao executar a operação de <i>checkout</i>	69
Figura 22 - O <i>plug-in</i> oferece a possibilidade de abrir um projeto recebido do repositório.....	70
Figura 23 - O <i>plug-in</i> fornece a possibilidade de criar um projeto a partir dos dados do <i>checkout</i>	70
Figura 24 - Janela com os artefatos para <i>commit</i>	71
Figura 25 - Janela de <i>log</i> das operações.	72
Figura 26 - Diagrama de casos de uso.	73
Figura 27 – Rascunho inicial do modelo conceitual.	74
Figura 28 - Diagrama de seqüência do caso de uso <i>checkout</i>	78
Figura 29 - Modelo conceitual expandido.....	79
Figura 30 - Solicitando a realização do <i>checkout</i>	80
Figura 31 - Janela 1.	81
Figura 32 - Janela 2.	81
Figura 33 - Diagrama de colaboração de registrarRepositório.....	84
Figura 34- Diagrama de colaboração de registrarArtefato.	84
Figura 35 - Diagrama de colaboração de registrarCaminho.....	84
Figura 36 - Diagrama de colaboração de terminarOperação.	84
Figura 37 - Diagrama de Classes de Projeto.....	86
Figura 38 - <i>Wizard</i> de criação de projetos do NetBeans.	88
Figura 39 - Menu de projetos no NetBeans.....	90
Figura 40 - <i>Wizard</i> de criação de novos arquivos.	91
Figura 41 - Um trecho do arquivo XML Layer criado.....	91

Figura 42 - Janela da operação de <i>commit</i>	96
Figura 43 - Diagrama de colaboração de adicionarArtefato.....	97
Figura 44 - Diagrama de colaboração de realizarOperação.....	100

Lista de Tabelas

Tabela 1 - Comparação entre as operações no repositório.	44
Tabela 2 - Comparação entre as características operacionais dos sistemas.....	44
Tabela 3 - Comparação entre as características gerais.	45
Tabela 4 - Comparação entre as interfaces dos sistemas.	45
Tabela 5 - Comparação entre as formas de licenciamento dos sistemas.	45

Lista de Abreviaturas

IDE – Integrated Development Environment

OSSD – Open Source Software Development

CVS – Concurrent Versions System

SVN – Subversion System

ADS – Ambiente de Desenvolvimento de Software

CASE – Computer-Aided Software Engineering

I-CASE – Integrated CASE

ECMA – European Computer Manufacturers Association

SCM – Software Configuration Management

SEI – Software Engineering Institute

SCI – Software Configuration Item

VCS – Version Control System

RCS – Revision Control System

CM – Configuration Management

JDK – Java Developers Kit

XML – Extensible Markup Language

API – Application Programming Interface

SPI – Service Provider Interface

JAR – JAVA Archiver

1 Introdução

O desenvolvimento da qualidade de software é uma tarefa complexa e trabalhosa. Sistemas de software tendem a se tornarem cada vez maiores e a complexidade de desenvolvimento e manutenção se torna mais difícil. Os softwares de controle de tráfego aéreo são algumas das coisas mais complexas já construídas pelo homem. Eles são o resultado da cooperação de muitas pessoas que tem trabalhado juntas como um grande time. Cada pessoa, ou grupo, gerenciando somente uma pequena parte de todo o sistema. Métodos e mecanismos são necessários para que seja possível dividir um grande sistema em partes gerenciáveis e, o mais importante, juntar essas partes novamente para dar forma ao sistema completo novamente.

Podemos citar, como outro exemplo de aplicação complexa, os sistemas de comutação telefônica. O que torna esses sistemas ainda mais complexos é o fato de que eles são desenvolvidos em muitas versões e variantes diferentes. O desenvolvimento dessas versões e variantes tem que ser feito de forma que todas elas sejam mantidas para o caso de haver necessidade de restaurá-las. Além disso, como cada parte do sistema pode existir em diversas versões, nós precisamos ter caminhos para controlar, armazenar e recuperar exatamente qual versão de cada parte que entrou no produto final [LARS 95].

Esta explosão na complexidade dos sistemas tornou necessária a automatização do processo de desenvolvimento para garantir o controle e a gerência efetivos do processo de software [LIM 98]. Em resposta a essa necessidade surgiram ferramentas individuais de apoio à programação, como

compiladores, montadores e depuradores e, em seguida, ferramentas de apoio a outras fases do desenvolvimento de software, como análise e projeto.

Dentre essas ferramentas, destacam-se os sistemas de controle de versão. Muitos sistemas de controle de versão foram desenvolvidos para gerenciar o histórico das versões de software com o intuito de produzir software de maior qualidade e suportar trabalho em equipe.

Porém, o crescimento do número de ferramentas utilizadas no processo de software gerou um segundo problema, a constatação de que a comunicação e a coordenação entre todas as ferramentas usadas no desenvolvimento e manutenção de software são essenciais para a produção eficiente de software de qualidade.

Surgiram, então, os ambientes integrados de desenvolvimento de software (IDE), que se propõem a integrar as diversas ferramentas utilizadas no processo de desenvolvimento de software.

Este trabalho propõe-se a integrar uma ferramenta de controle de versão de código fonte aberto que vem ganhando espaço na comunidade desenvolvimento de código fonte aberto (OSSD) e o IDE NetBeans.

1.1 Motivação

A NetBeans.org é atualmente uma das maiores comunidades de desenvolvimento de software de código fonte aberto do mundo. Ela mantém o desenvolvimento de um estável e robusto ambiente de desenvolvimento de software, que cada vez mais vem ganhando adeptos nas comunidades OSSD,

principalmente para o desenvolvimento de software JAVA para Web, devido ao fato de ser o IDE mais completo para este fim.

Porém, diferente de seu maior concorrente, o Eclipse, o NetBeans recebe pouco esforço no sentido de desenvolvimento de *plug-ins* de terceiros, o que o torna limitado em alguns aspectos. Isto se dá porque a sua interface para implementação de *plug-ins* é bem mais complexa do que a do concorrente.

O desenvolvimento colaborativo de software se tornou a maneira mais utilizada de produzir software de qualidade [WU 2003]. Essa visão de desenvolvimento de software necessita de ferramentas de gerência de configuração de software (SCM) que auxiliem no processo de desenvolvimento. O CVS foi durante muito tempo o padrão *de facto* para este fim, porém, esta ferramenta não foi estruturada para acompanhar as mudanças surgidas na complexidade, tanto do software produzido, como do processo de desenvolvimento em si. Dentro desta perspectiva, o Subversion surge como o substituto do CVS e avança para se tornar a ferramenta de controle de versão padrão nas comunidades de OSSD.

Oficialmente o NetBeans oferece para os usuários apenas a opção de utilizar o CVS como sistema de controle de versão e, nenhum esforço foi encontrado na comunidade para possibilitar o uso do SVN neste IDE. Já os usuários do Eclipse possuem várias opções de *plug-ins* para ambas as ferramentas.

Os usuários do NetBeans deveriam ter a opção de usar uma ferramenta de controle de versão de maior opções e qualidade e, por isso, nós acreditamos que

possibilitar aos usuários do NetBeans o uso da ferramenta de controle de versão Subversion é de suma importância para a comunidade de OSSD.

1.2 Objetivos

1.2.1 Objetivo Geral

Integrar o sistema de controle de versão Subversion no IDE NetBeans.

1.2.2 Objetivo Específico

Pesquisas em integração de ferramentas tem sido um importante tópico de estudo desde o modelo de Stoneman proposto no final dos anos 70 e resumido por Brown [BRO 92c] em duas categorias, o nível conceitual (“o que é integração”) e o nível mecânico (“como prover integração”). Em geral, softwares de prateleira são chamados integrados se funcionarem coerentemente e eficientemente em um ambiente como um todo, como é o caso de um ambiente integrado de desenvolvimento de software (IDE) [KAP 2005]. Neste contexto, integrar uma ferramenta a um IDE significa desenvolver um *plug-in* que disponibilize as funcionalidades dessa ferramenta.

Assim, o objetivo deste trabalho é desenvolver um *plug-in* para o NetBeans que disponibilize as funções do SVN, que seja semelhante, do ponto de vista de utilização, e compatível, com o *plug-in* CVS do NetBeans, que disponibilize acesso aos repositórios através dos protocolos SVN, SVN sobre SSH e HTTP, que seja de fácil utilização e que seja facilmente expansível para o acréscimo de novas funcionalidades.

1.2.3 Escopo de Estudo

Devido ao tempo disponível para o desenvolvimento da tarefa o *plug-in* a ser desenvolvido implementará as funções de *checkout*, *commit* e *uptade*.

2 O Processo de Desenvolvimento de Software

Informalmente, o processo de desenvolvimento de software (processo de software) pode ser compreendido como o conjunto de todas as atividades necessárias para transformar os requisitos do usuário em software [HUM 89b] [OST 87]. Um processo de software é formado por um conjunto de passos de processo parcialmente ordenados, relacionados com conjuntos de artefatos, pessoas, recursos, estruturas organizacionais e restrições e tem como objetivo produzir e manter os produtos de software finais requeridos [LON 93] [DOW 91].

Os passos de processos são atividades ou tarefas. Em [CON 94], uma atividade é um passo de processo que produz mudanças de estado visíveis externamente no produto de software. Atividades incorporam e implementam procedimentos, regras e políticas, e têm como objetivo gerar ou modificar um dado conjunto de artefatos. Elas podem ser organizadas em redes com duas dimensões (horizontal e vertical) e estão associadas com papéis, ferramentas e artefatos. Uma atividade aloca recursos (por exemplo, máquinas e orçamento), é escalonada, monitorada e atribuída a desenvolvedores (agentes), que podem utilizar ferramentas para executá-la. Uma atividade também pode ser executada somente por ferramentas automatizadas, sem intervenção humana. Neste caso, a atividade é automática. Toda atividade possui uma descrição, a qual pode especificar os artefatos necessários, as relações de dependência com outras atividades, as datas de início e fim planejadas, os recursos a serem alocados e os agentes responsáveis pela mesma [DOW 91]. A Figura 1 ilustra os elementos envolvidos com uma atividade de processo de software.

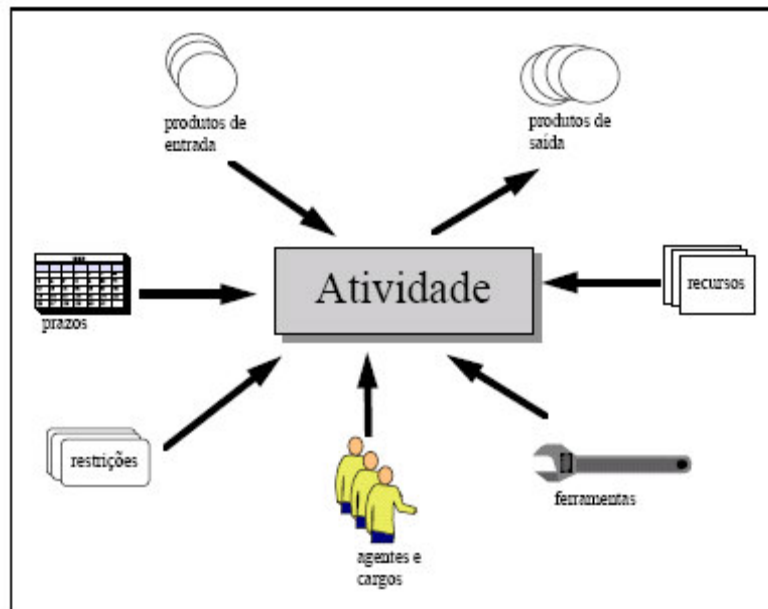


Figura 1 - Elementos envolvidos em atividades do processo de software [LIM 98].

Um agente está relacionado com as atividades de um processo e pode ser uma pessoa ou uma ferramenta automatizada (quando a atividade é automática). Diferentes agentes terão percepções diferentes acerca do que acontece durante o processo de software. Um gerente, por exemplo, perceberá os aspectos de controle e alocação de recursos e cronogramas para atividades, enquanto um desenvolvedor perceberá as suas atividades como atribuições que devem ser feitas para produzir um resultado [DOW 91].

Um artefato é um produto criado ou modificado durante um processo [LON 93]. Tal produto é resultado de uma atividade e pode ser utilizado posteriormente como matéria prima para a mesma ou para outra atividade a fim de gerar novos produtos. Desta forma, uma atividade pode consumir produtos (de entrada) e gerar novos produtos (de saída).

A realização do processo é afetada pelas restrições, que podem atingir atividades, agentes, recursos, artefatos, papéis e seus relacionamentos [LON 93]. Uma restrição é uma condição definida que um passo de processo deve satisfazer antes ou depois de ser executado [FEI 93].

Um cargo ou papel é um conjunto de permissões e obrigações associadas a um objetivo funcional [LON 93]. Um agente sempre ocupa um cargo quando realiza uma tarefa. Projetista, programador e gerente são exemplos de cargos. Os cargos podem ser organizados em estruturas tais como organogramas [DOW 91].

Um modelo de processo de software é uma descrição abstrata do processo de software. Vários tipos de informação devem ser integrados em um modelo de processo de software para indicar quem, quando, onde, como e por que os passos são realizados [LON 93]. Para representar um modelo de processo de software é utilizada uma linguagem de modelagem do processo de software, a qual deve oferecer recursos para descrever e manipular os passos do processo.

Um modelo de processo instanciado ou processo executável é um modelo de processo pronto para execução [FEI 93]. Este modelo pode ser interpretado por uma máquina de processo, a qual é um componente provido por um ambiente de desenvolvimento de software (ADS) com capacidade de suportar a modelagem e a execução de processos de software (ADS orientado a processo).

Um projeto, segundo [CON 94], é a instância de um processo, com objetivos e restrições específicas. Pode-se dizer que um projeto é um esforço para desenvolver um produto de software, ou seja, envolve uma estrutura organizacional, prazos, orçamentos, recursos e um processo de desenvolvimento. Neste sentido, a gerência de projetos tem como responsabilidades o

planejamento, controle e monitoração de um processo em execução [FEI 93] [DOW 91], enquanto a gerência de processos preocupa-se em construir, analisar e verificar modelos de processo, para isso obtendo informações durante a execução desse processo a fim de evoluir tais modelos para que possam ser usados posteriormente [DOW 91].

2.1 Ambientes Automatizados de Apoio a Gerência do Processo de Software

Para garantir o controle e a gerência efetivos do processo de software, tornou-se necessário automatizá-lo onde possível e prover suporte às pessoas que nele trabalham. Os ambientes que auxiliam o desenvolvimento de software permitem esse controle e facilitam a execução das tarefas do desenvolvimento.

Existem diversos tipos de ambientes automatizados de apoio a tarefas dos usuários. Estes ambientes estão classificados segundo [CHR 95] em: software configuration management (SCM), produtos groupware, ADS, ADS orientados a processo, produtos workflow e workgroup. Estas classes se diferenciam em termos de gerência dos dados, suporte ao processo, integração de ferramentas e comunicação humana.

A tecnologia de ambientes de desenvolvimento de software (ADS) evolui à medida que são identificadas as reais necessidades do processo de desenvolvimento de software e sua integração com outras tecnologias. Portanto, o conhecimento dessa evolução leva a uma melhor análise das principais características e requisitos destes ambientes. Dentre as principais características,

destacam-se a integração entre ferramentas e a arquitetura do ambiente, a serem tratadas neste capítulo.

Logo em seguida, será apresentada, também, a evolução das ferramentas de SCM e os principais conceitos envolvidos.

2.1.1 Ambientes de Desenvolvimento de Software (ADS)

As primeiras definições relacionadas aos ADS foram publicadas na década de 80 através do relatório Stoneman [BRO 92a]. Este relatório definiu a arquitetura de um ambiente de suporte à programação em Ada chamado APSE4 e foi o precursor dos trabalhos subseqüentes em ADS.

A evolução da tecnologia de ADS iniciou a partir da construção de ferramentas individuais de apoio a programação, como compiladores, montadores e depuradores. Em seguida houve o surgimento de ferramentas de apoio a outras fases do desenvolvimento de software, como análise e projeto. Tais ferramentas têm como característica principal o suporte a determinados métodos de desenvolvimento e ficaram conhecidas como ferramentas CASE.

Tipicamente, as ferramentas CASE suportam algum método de desenvolvimento para usuários individuais provendo facilidades na interface com o usuário para a construção de sistemas de informação.

As ferramentas CASE evoluíram para ambientes CASE integrados (I-CASE - Integrated CASE), os quais são compostos de várias ferramentas capazes de transferir informações para outras ferramentas [PRE 95]. Entretanto, a característica de suporte a um método específico permaneceu nos I-CASEs. Um

ADS tem como objetivo prover um ambiente no qual produtos de software de grande porte possam ser desenvolvidos através da integração de um conjunto de ferramentas que suportam métodos de desenvolvimento, apoiados por uma estrutura que permite a comunicação e cooperação entre as ferramentas [BRO 92a].

Este conceito veio como resposta ao reconhecimento de que a comunicação e a coordenação entre todas as ferramentas usadas no desenvolvimento e manutenção de software são essenciais para a produção eficiente de software de qualidade. Uma das metas dos estudos nesta área tem sido obter um bom entendimento sobre como as ferramentas são integradas, definidas, implementadas, adaptadas e desenvolvidas.

As diferenças existentes entre ADS e ferramentas CASE são abordadas em [BRO 92b]. Entretanto, o conceito de ADS é considerado bem mais amplo por derivar do trabalho de Stoneman e prover uma estrutura unificadora de serviços onde várias ferramentas de diferentes métodos podem ser integradas.

Para que o ADS proporcione uma garantia da qualidade do produto e do processo, o processo de software utilizado na organização deve estar formalizado e ser obedecido. O ambiente, então, usa o modelo de processo de software descrito em uma linguagem de modelagem do processo para executar de forma independente aquelas tarefas que podem ser completamente automatizadas, coordenar tarefas mais complexas, e garantir informação de gerência atualizada, eliminando cargas administrativas desnecessárias dos usuários. ADS com essas características são conhecidos como ADS orientados ao processo (ou centrados em processo).

Estes ambientes conhecem o modelo de processo, permitem modificações no mesmo e possuem mecanismos para executá-lo. Além disso, o ADS necessita manter um repositório comum com todos os objetos produzidos pelas ferramentas, todos os dados sobre o próprio processo e de todos os produtos já desenvolvidos e em desenvolvimento.

2.1.1.1 Integração de Ferramentas

Ambientes de desenvolvimento de software são compostos de ferramentas. Estas ferramentas deverão compartilhar informações sobre os vários projetos e os usuários necessitarão utilizar em uma ferramenta uma informação gerada por outra ferramenta. Portanto, o ADS deve prover uma estrutura unificadora que integre suas ferramentas. Devido a essa característica, os ambientes de desenvolvimento de software são comercialmente conhecidos como “ambientes integrados de desenvolvimento” (IDE – Integrated Development Environment).

O conceito de integração pode ser interpretado de diferentes formas e possui várias classificações. Segundo [THO 92], a integração é uma propriedade dos relacionamentos entre as ferramentas que formam um ambiente e refere-se ao grau de “acordo” que existe entre essas ferramentas.

A integração é um elemento chave para o suporte ao desenvolvimento de software. Os principais aspectos de integração em ADS são [BRO 92a]:

Integração de Interface: Cada ferramenta possui o mesmo conjunto de construtores na interface com o usuário, ou seja, os usuários interagem com todas as ferramentas do ambiente da mesma forma;

Integração de Processo: O ambiente suporta um processo de software definido;

Integração de ferramentas: Ferramentas compartilham dados através de um mesmo formato de dados e podem ser facilmente combinadas;

Integração de equipes: O ambiente promove o trabalho em grupo assegurando comunicação efetiva e disseminação de informação;

Integração de gerência: O ambiente auxilia os gerentes a controlarem o desenvolvimento. A informação de gerência é derivada das informações técnicas produzidas por engenheiros de software.

Com relação à integração de ferramentas, os ambientes podem se enquadrar em vários níveis, segundo [BRO 92a]. Estes níveis (carregador, léxico, sintático, semântico e de método) não são estritamente hierárquicos, apenas progridem de baixo grau de integração a alto grau de integração.

Os fatores cruciais na avaliação do grau de integração entre ferramentas em um ADS são as formas de armazenamento, compartilhamento e transferência de informações entre as ferramentas. Os níveis são mostrados na Figura 2 e apresentados a seguir.

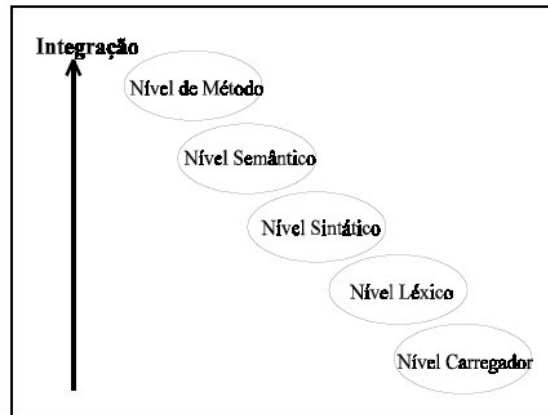


Figura 2 - Níveis de integração de ferramentas em um ADS [BRO 92].

Nível Carregador: Neste nível, cada ferramenta analisa seus dados de entrada e saída pois não há entendimento comum sobre os dados que as ferramentas podem tratar. Um exemplo é o ambiente UNIX onde a integração provida é através da composição de ferramentas em um formato de arquivos simples e consistente. Cada ferramenta é responsável pelos seus dados e o compartilhamento de informações se dá pelo fato de todas as entradas e saídas de ferramentas serem na forma de um fluxo de bytes, não existindo nenhuma forma de cooperação mais aprofundada entre as ferramentas;

Nível Léxico: Neste nível, grupos de ferramentas compartilham formatos de dados e convenções de operações, o que permite sua interação. Porém, para adicionar uma nova ferramenta ao grupo, a comunicação e os formatos de dados dessa ferramenta devem ser compreendidos, e isto se torna bastante trabalhoso;

Nível Sintático: Neste nível um conjunto de regras de formação de estruturas de dados é compreendido por todas as ferramentas do ambiente, não sendo necessário que cada ferramenta analise, valide e converta estrutura de dados de outras ferramentas. Este é o nível encontrado na maioria dos ADS, tipicamente na forma de um esquema de banco de dados que pode ser consultado por todas as ferramentas do ambiente;

Nível Semântico: Para aumentar a integração, um entendimento das estruturas de dados deve ser acompanhado de uma definição da semântica dessas estruturas. Estão disponíveis as definições das estruturas de dados mais os significados das operações sobre essas estruturas;

Nível de Métodos: Neste nível as ferramentas, além de estarem no nível semântico, são usadas no contexto do processo de software implicando que as ferramentas “entendam o processo”. Neste nível as ferramentas devem concordar com as estruturas de dados, operações e com o processo de desenvolvimento específico. Este nível somente pode ser obtido por um ADS orientado a processos.

2.1.1.2 Arquitetura

O modelo de referência ECMA (European Computer Manufacturers Association) [BRO 92a], descreve uma arquitetura para ambientes integrados de desenvolvimento de software. Um modelo de referência é uma estrutura conceitual

e funcional que facilita a descrição e comparação de sistemas, não devendo ser utilizado como uma especificação de implementação [BRO 92a].

O modelo ECMA segue uma visão orientada a serviços abstraindo detalhes de implementação e concentrando-se nas capacidades do ambiente. Em ADS orientados a processo, a escolha do processo específico utilizado em um projeto de desenvolvimento é livre e sua definição e execução é suportada por um conjunto de serviços de gerência do processo: desenvolvimento, execução, visibilidade, monitoração, transação e serviços de recursos. A Figura 3 apresenta o modelo de referência ECMA. O diagrama não deve ser interpretado como um conjunto de camadas. Os serviços de mensagem permitem comunicação nos dois sentidos: serviço a serviço, ferramenta a ferramenta e serviço a ferramenta.

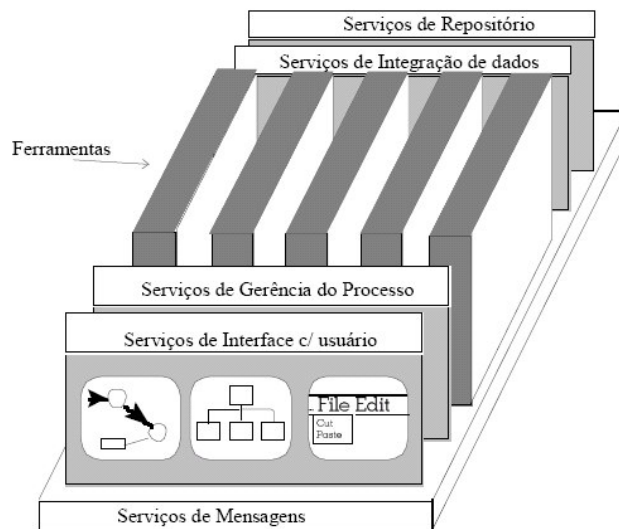


Figura 3 - Modelo de Referência ECMA [BRO 92a]

As ferramentas são componentes de software que não fazem parte da arquitetura do ambiente, mas que utilizam os serviços dessa estrutura e estão integradas em nível de método.

A seguir são descritos os serviços do modelo ECMA:

- Os serviços de repositório de dados mantêm, gerenciam e nomeiam entidades de dados ou objetos e seus relacionamentos. Algum suporte a execução do processo também é tratado por esses serviços além do suporte à distribuição física de dados e processos;
- Serviços de integração de dados incrementam os serviços de repositório provendo uma semântica de alto nível e operações para tratar os dados armazenados no repositório. Estes serviços incluem tratamento de versões e configurações;
- Os serviços de gerência do processo (ou serviços de gerência de tarefas) provêm uma camada de abstração que permite ao usuário lidar com as tarefas, aumentando a "inteligência" do ambiente.
- Os serviços de mensagem fornecem um padrão de comunicação que pode ser usado pelas ferramentas e outros serviços.
- Os serviços de interface com o usuário provêm consistência de interface adotada em todas as ferramentas.

Além desses serviços, a estrutura ainda provê mecanismos de segurança. Mais detalhes sobre os serviços providos pela arquitetura podem ser encontrados em [BRO 92a].

Com esta padronização novas ferramentas poderiam ser integradas ao ambiente permitindo que os usuários percebessem todos os níveis de integração do ambiente [CHE 92].

2.1.2 Software Configuration Management (SCM)

Os conceitos de SCM são normalmente associados com as suas funcionalidades e atividades. Sistemas diferentes de SCM implementam vários dos conceitos descritos no “IEEE Standard for Software Configuration Management Plans”. Segundo [WU 2003], as atividades de SCM são tradicionalmente agrupadas em quatro categorias: identificação de configuração, controle de configuração, contabilidade de estado e auditoria e revisões. Essas quatro funções são descritas no IEEE Standard 828-1990 como segue:

Identificação: identificar, nomear e descrever as características físicas e funcionais do código fonte, especificações, projeto, e elementos de dados a serem controlados pelo projeto;

Controle: requisitar, avaliar, aprovar ou desaprovar e implementar mudanças;

Contabilidade de Estado: gravar e solicitar o estado dos itens de configuração do projeto (por exemplo, versão aprovada inicial, estado das mudanças solicitadas, implementação das mudanças aprovadas);

Auditorias e Revisões: determinar que extensão do item atual de configuração reflete as características físicas e funcionais solicitadas.

Com o crescimento e desenvolvimento de várias ferramentas de SCM, o Software Engineering Institute (SEI) da universidade de Carnegie Mellon reviu as ferramentas de SCM e evoluiu a definição de SCM adicionando três outras funções [DAR 91], são elas:

Manufatura: gerenciar a construção e construir o produto de uma maneira ótima;

Gerência de Processo: assegurar o uso dos procedimentos, políticas e modelo de ciclo de vida da organização;

Trabalho em Equipe: controlar o trabalho e as interações de múltiplos usuários em um produto.

Essas três funções complementam a definição do padrão IEEE para tornar a definição de SCM mais detalhada e completa. Elas também estão de conformidade com o conceito de SCM, que diz que SCM não é somente um processo, mas também uma disciplina de gerenciamento [WU 2003]. Entretanto,

não existem regras fixas para implementar as funcionalidades de SCM, a maioria dos sistemas implementa somente uma parte de todas as funcionalidades e outros uma nova combinação de funcionalidades, que é única para suas tarefas específicas. Como o objetivo do SCM é fazer software de qualidade e não existe um padrão universal que constitui um sistema de SCM, todos os sistemas que fornecem funcionalidades parciais, mas, pretendem fornecer um SCM com todas as funcionalidades para conseguir alta qualidade de software, são considerados sistemas de SCM pela comunidade de engenharia de software [DAR 91].

A despeito da variedade de implementações de SCM, diversos conceitos básicos têm sido adotados no projeto e implementação de todos os sistemas de SCM. Esses conceitos básicos incluem itens de configuração, versão, *release* e *baseline*.

Item de Configuração de Software (SCI – Software Configuration Item): é uma agregação de software que é designada para gerenciamento de configuração e é tratada como uma simples entidade no processo de SCM. Normalmente, SCIs incluem especificações, segmentos de código fonte, bibliotecas de código, documentações, e manuais de usuário e desenvolvedor;

Versão: um termo genérico usado para descrever um *release* inicial de um SCI. Diferentes sistemas de controle de versão têm diferentes interpretações para este termo. Por exemplo, no CVS, “versão” corresponde a um arquivo. Versões de projetos são produzidas através de um *tagging* em um grupo de arquivos num momento particular;

Release: a notificação e distribuição de uma versão aprovada;

Baseline: é um conjunto de itens de software formalmente designados e reparados em um tempo específico durante o ciclo de vida do software. Pode-se dizer que um *baseline* é uma versão específica usada como ponto de referência para desenvolvimento futuro. Porém, uma *baseline* é diferente de uma versão. Tipicamente, um *baseline* representa um marco conceitual no processo de desenvolvimento de software.

2.1.2.1 Sistema de Controle de Versão (VCS - Version Control System)

Controle de versão é uma função importante de SCM. Como descrito acima, um sistema que implementa parcialmente o padrão SCM é considerado um sistema SCM, conseqüentemente, um sistema de controle de versão é considerado um sistema de SCM. Controle de versão envolve o armazenamento de itens de configuração, o armazenamento de mudanças, a gerência dessas mudanças, e a função mais importante, a facilitação e gerenciamento do desenvolvimento colaborativo de software, que permite que vários integrantes da equipe trabalhem no mesmo conjunto de código, enviem suas mudanças, e se comuniquem com os outros integrantes [WU 2003].

2.1.2.1.1 Termos

Existem alguns termos comuns que são compartilhados pelos vários sistemas de controle de versão independentemente dos conceitos básicos de SCM descritos na seção 2.1.2.

Revisão: Uma mudança cometida no histórico de um item de configuração. Existe uma linha tênue entre a definição de revisão e versão, e as pessoas tendem a confundi-las. De fato, diferentes ferramentas de controle de versão usam a terminologia em caminhos ligeiramente diferentes, como mencionado na seção 2. No CVS, uma revisão de um arquivo pode pertencer a mais que uma versão de um projeto, desde que a revisão de um arquivo possa ser “*tagged*” várias vezes.

Repositório: uma biblioteca centralizada de arquivos que estão sob o controle de versão. A noção de repositório foi inicialmente proposta com o Revision Control System (RCS) que foi desenvolvido por Walter F. Tichy. Todas as informações de CM sobre arquivos e o conteúdo desses arquivos são mantidas no repositório;

Workspace: o espaço no qual os programadores executam as tarefas de desenvolvimento;

Working Copy: a cópia dos arquivos do repositório no espaço de um usuário.

2.1.2.1.2 Atividades Comuns

A finalidade de uma ferramenta de controle de versão é suportar as funcionalidades de SCM. Embora sistemas de controle de versão diferentes implementem modelos de controle de versão diferentes, existem algumas características comuns entre eles. Segue um resumo das atividades comuns associadas com os Sistemas de Controle de Versão.

Conexão de Dados: como mencionado acima, a noção de um repositório é extensamente usada em sistemas de controle de versão. Um repositório pode ser um sistema de arquivos local ou um servidor remoto, e providencia árvore de versão para gerenciar versões diferentes de arquivos. Por exemplo, várias ferramentas de controle de versão usam arquitetura de cliente/servidor e os clientes podem pegar quaisquer informações sobre qualquer arquivo acessando o repositório;

Check-in/Check-out: os usuários submetem suas mudanças através de “*check-in*”, os artefatos de software de seus *workspaces* pessoais são enviados para o repositório compartilhado, e recuperam artefatos de software do repositório através do “*check-out*”;

Versioning: depois de tudo submetido, a ferramenta de controle de versão atribui um novo número de revisão para os arquivos modificados, e grava as informações relacionadas, como tempo, usuário, etc;

Operação de *Diff*: quase todas as ferramentas de controle de versão providenciam mecanismos para os usuários compararem as diferenças entre duas revisões de arquivos selecionados. Uma operação de “diff” ideal permitiria a comparação entre uma cópia de trabalho de um usuário e qualquer revisão no repositório, como também, uma comparação entre duas revisões quaisquer. Algumas ferramentas providenciam somente parte desta funcionalidade, enquanto outras providenciam mecanismos mais sofisticados;

Operação de *Get*: este mecanismo é fornecido para os usuários recuperarem qual revisão antiga dos arquivos ou versões de um projeto. Ela é usada quando os usuários procuram versões antigas melhores que a corrente e querem reverter para uma dessas versões;

Operação de *Report*: esta operação é usada para gerar vários relatórios úteis sobre os artefatos de software, como o histórico de um artefato, anotações, etc.

2.1.2.1.3 Algumas Ferramentas de Controle de Versão Existentes

Foram investigados diversos sistemas de controle de versão populares para selecionar o sistema de controle de versão a ser utilizado. Descobrimos que alguns deles são bem projetados e implementados e tem um número considerável de usuários.

Rational ClearCase

Usa um único modelo para a finalidade de configuração e gerenciamento. Os conceitos básicos do UCM são “atividade” e “artefato”. Uma “atividade” é uma parte de um trabalho a ser realizada no projeto. Um “artefato” é um item, normalmente, um arquivo que está sob o controle de versão. Seu modelo de controle de versão atribui versão a todos os artefatos no ciclo de vida do desenvolvimento de software. O artefato pode ser um simples arquivo, um diretório, um subdiretório e todos os tipos de objetos do sistema de arquivos. Esta é sua diferença, a maioria dos outros VCS somente suporta controle de versão de arquivos [RAT 2006].

Perforce

Possui uma interface com o usuário amigável, no Perforce Windows cliente chamado P4Win, uma interface padrão Microsoft é usada para mostrar as informações relacionadas a todas as tarefas de SCM. Os trabalhos em andamento são mostrados e um mecanismo de arraste-e-solte é disponibilizado para o usuário realizar suas tarefas. Ele não apenas disponibiliza uma visão do diretório de trabalho, como outro CVS, mas também uma visão do repositório como um todo. Outra vantagem do P4Win é que ele providencia uma visão da lista de tarefas do usuário, que é excelente para pessoas que trabalham em equipe para verificar os estados do andamento de trabalho de seus colegas [PER 2006].

Revision Control System (RCS)

Permite controle de versão simples e básico dos arquivos que estão no repositório. Os arquivos são armazenados no repositório na forma de árvore. Cada arquivo tem todas as suas revisões em uma árvore ancestral. A raiz da árvore é a revisão mais antiga deste arquivo, e as revisões sucessivas são ligadas aos seus ancestrais. RCS têm seu próprio esquema para atribuir números de revisões para os arquivos. No RCS, somente os arquivos correntes são armazenados no repositório, que conhece somente as diferenças entre as revisões armazenadas, o que economiza muito espaço em disco, mas aumenta o tempo de acesso [TIC 85].

Concurrent Versions System (CVS)

É um sistema de controle de versão de código fonte aberto. O CVS usa o formato para armazenamento de arquivo do RCS, entretanto, ele é mais sofisticado e suporta muitas outras funções, a maior diferença é que o CVS suporta um ambiente de desenvolvimento concorrente, enquanto que o RCS somente suporta um simples usuário. Esta função é importante hoje em dia porque a programação em equipe é o estilo de programação que esta prevalecendo na comunidade de engenharia de software. Porém, as vantagens do CVS também introduzem desvantagens. Devido a suas poderosas funções, ele requer maior administração e o grande número de comandos produz uma curva de aprendizagem íngreme para os novatos. Existem alguns termos definidos especialmente no CVS. Uma *tag* é o nome atribuído a uma coleção de revisões de arquivos em um momento particular de um histórico de versão. Uma versão de um projeto no CVS normalmente é determinada por uma *tag*. Outro conceito é o

conceito de *branch*. Um CVS *branch* corta o desenvolvimento de um projeto em histórias paralelas e as mudanças feitas em um *branch* não afetam o outro [CVS 2006].

Subversion (SVN)

O Subversion foi desenvolvido para ser um CVS melhorado, por isso ele possui a maioria das características do CVS. Geralmente, a interface do Subversion para uma característica em particular é parecida com a do CVS, exceto onde há uma razão específica para isso não acontecer. O controle de versão ocorre sobre diretórios, arquivos e meta-dados. A falta dessas características é uma das queixas mais comuns do CVS. O Subversion controla versões não somente do conteúdo e da existência dos arquivos, mas também de diretórios, cópias e renomeações. Ele também permite o controle de executáveis e os *commits* são verdadeiramente atômicos. O controle do número de revisões é feito por *commits*, e não por arquivos. As mensagens de *log* são anexadas na revisão e não gravadas de forma redundante como no CVS [SVN 2006].

O Subversion tem a opção de trabalhar com o Apache, através do protocolo WebDAV/DeltaV, podendo usar este protocolo para as comunicações de rede e o servidor Web Apache para permitir serviços de repositório no lado do servidor. Isto dá ao Subversion uma vantagem sobre o CVS em interoperabilidade, e providencia várias características importantes, como: autenticação, autorização baseada em arquivo, compressão, e visualização Web do repositório. O Subversion permite a opção de trabalhar como um servidor *standalone*, e pode ser acessado via SSH. *Branching* e *tagging* são operações implementadas sobre a

operação de cópia, que consome pouco espaço e recursos de hardware. Uma cópia é uma *tag* e, se for iniciado um *commit* em uma cópia, isso se caracteriza como um *branch*.

O SVN foi desenvolvido para ser um sistema cliente/servidor, evitando assim alguns dos problemas de manutenção que tem atingido o CVS. O código é estruturado em um conjunto de módulos com interfaces bem definidas, desenvolvidas para serem chamadas por outras aplicações [WOR 2005].

O protocolo cliente/servidor envia *diffs* em ambas as direções, o protocolo de rede usa a banda eficientemente para transmitir *diffs* em ambas as direções sempre que possível. Já o CVS envia *diffs* do servidor para o cliente, mas nunca do cliente para o servidor. Os custos são proporcionais aos tamanhos das mudanças, não ao tamanho dos dados. Em geral, o tempo necessário para uma operação no SVN é proporcional ao tamanho das mudanças resultantes da operação e não ao tamanho absoluto do projeto em que as mudanças estão acontecendo, o que é uma propriedade do modelo de repositório do Subversion.

O Subversion permite implementações do repositório baseadas em texto plano ou em banco de dados e controla a versão de *links* simbólicos.

A seguir faremos uma breve comparação entre o CVS e o Subversion baseando-se nos seguintes aspectos: operações no repositório, características, estado técnico, interface com o usuário e licenciamento.

2.1.2.1.4 Comparação entre o CVS e o SVN

Como citado no item anterior, existem várias ferramentas para controle de versão, algumas *open-source* e outras comerciais. A integração de uma

ferramenta comercial poderia não ser possível se ela não disponibilizasse uma API para integração, ou poderia ser impossibilitada pelos mecanismos legais, como licenciamento, etc. Assim, deveria ser escolhida uma ferramenta *open-source* para ser integrada. As duas principais ferramentas *open-source* existentes são o CVS e o Subversion, fez-se, então, uma comparação entre essas duas ferramentas com o intuito de justificar a escolha do Subversion.

A seguir algumas tabelas comparando várias características das duas ferramentas [WOR 2005].

Operação	CVS	SVN	Descrição
<i>Commit</i> Atômico	Não	Sim	Se uma operação no repositório é interrompida, o repositório não fica em um estado inconsistente.
Mover e renomear arquivos	Não	Sim	Suporta mover um arquivo ou diretório para uma localização diferente e mantém o histórico dos arquivos.
Cópias de arquivos e Diretórios	Não	Sim	Suporta cópia de arquivos ou diretórios para uma localização diferente e mantém seu histórico.
Replicação Remota do Repositório	Não	Sim	Suporta copiar um repositório remoto para ter uma cópia equivalente no sistema local.
Propagação de mudanças para os Repositórios pais	Não	Sim	Suporta a propagação de mudanças de um repositório para outro.

Permissões do Repositório	Limitado	Sim	Suporta permissões de acesso para diferentes partes do repositório remoto
Suporte a <i>Changesets</i>	Não	Parcial	

Tabela 1 - Comparação entre as operações no repositório.

Característica	CVS	SVN	Descrição
Habilidade para trabalhar somente com um diretório do repositório	Sim	Sim	Suporta checkout de somente um diretório do repositório ou é restrito a todo o repositório.
Mensagens de <i>commit</i> por arquivo	Não	Sim	Suporta mensagens de <i>commit</i> por arquivo.

Tabela 2 - Comparação entre as características operacionais dos sistemas.

Característica	CVS	SVN	Descrição
Documentação	Excelente	Boa	Como é a documentação disponível, tutoriais, livros, etc.
Fácil Descoberta de erro (<i>deployment</i>)	Sim	Sim	Possui ferramentas de <i>deployment</i> .
Suporte a rede	Boa	Excelente	Como é a integração de rede nesse sistema. É compatível com os protocolos e infra-estruturas existentes.

Portabilidade	Boa	Excelente	Quão portátil é o sistema para os vários sistemas operacionais e arquiteturas.
---------------	------------	------------------	--

Tabela 3 - Comparação entre as características gerais.

Interface	CVS	SVN	Descrição
Web	Sim	Sim	Possui uma interface baseada em WWW que pode ser usada para explorar a árvore e as várias revisões dos arquivos.
GUI	Sim	Sim	Possui uma interface gráfica com o usuário

Tabela 4 - Comparação entre as interfaces dos sistemas.

Licenciamento	CVS	SVN
Tipo de licenciamento	GNU GPL (open source)	Apache/BSD-style license. (open-source).

Tabela 5 - Comparação entre as formas de licenciamento dos sistemas.

3 O Ambiente NetBeans

A NetBeans.org é uma das maiores comunidades de desenvolvimento de software de código fonte aberto (Open Source Software Development - OSSD), com 1100 usuários ativos, 500 contribuintes, 9500 alterações de código fonte por mês, 1.7 milhões de downloads, e algo em torno de 100.000 participantes de listas de email. Ela é uma comunidade focada em JAVA, patrocinada pela Sun Microsystems e voltada a criar um ambiente integrado de desenvolvimento de software (IDE) para desenvolvimento de aplicações JAVA e, também, uma plataforma para desenvolvimento de outros produtos [JEN 2005].

A comunidade iniciou como um projeto de estudantes, em 1996, que foi, adquirido e, logo em seguida, liberado como uma comunidade de código fonte aberto pela Sun, de quem a equipe NetBeans recebe muitos dos colaboradores [NET 2006].

O IDE NetBeans é desenvolvido em JAVA, assim, ele funciona em qualquer plataforma que suporte JDK. Além disso, a configuração de tempo de execução do NetBeans é descrita em XML. Quando o *core* do NetBeans é carregado, ele configura e lança o resto do sistema. O *core* lê a descrição XML e carrega os *plug-ins* apropriados, que são componentes escritos em Java que implementam a sua API.

É este conceito de *plug-in* que permite a extensão do NetBeans. Muitas das mais importantes funcionalidades do NetBeans são *plug-ins* que podem ser desconectados do sistema e atualizados quando necessário, e, como o IDE

NetBeans é um software de código fonte aberto, pode-se modificar e recompilar qualquer *plug-in* da maneira que se desejar.

3.1 A Arquitetura do IDE NetBeans

A arquitetura do IDE NetBeans divide-se em duas seções principais: O *core* (também chamado de *application runtime*) e as APIs abertas. Elas podem ser encontradas nos pacotes JAVA `org.netbeans.core.*` e `org.openide.*`, respectivamente. As bibliotecas binárias desses arquivos fazem parte da instalação do NetBeans e estão nos arquivos `$NB_HOME/lib/core.jar` e `$NB_HOME/lib/openide.jar`. O *core* implementa muitas interfaces definidas nas APIs abertas e o *runtime engine* do NetBeans. As APIs abertas são o conjunto de ferramentas disponíveis para se escrever *plug-ins* que funcionarão dentro do NetBeans. Um *plug-in* é um arquivo `.jar` (Java ARchive ou JAR) que contém classes JAVA e um arquivo denominado *manifest* que descreve para o NetBeans *runtime* o que é o *plug-in*, como instalá-lo e desinstalá-lo [GRE 2002].

Tanto para IDEs como para qualquer outra aplicação, essa arquitetura traz inúmeras vantagens. Como, por exemplo, a habilidade para suportar múltiplas línguas ou a habilidade para envolver a funcionalidade de outras ferramentas de desenvolvimento e apresentá-la dentro do IDE. Um simples exemplo deste tipo de habilidade são os compiladores externos e máquinas virtuais JAVA. Eles podem ser usados para compilação e execução definindo-se serviços que lançam um processo externo e mostram a saída deste processo (ex. mensagens de erro do compilador). Um exemplo mais complexo seria a integração de uma ferramenta de

XML pré-existente, ela poderia ser integrada diretamente através da interface gráfica do NetBeans.

Alguns dos elementos do NetBeans, como o Sistema de Arquivos, Nós e API Explorer, podem também ser usados como bibliotecas para aplicações externas ao NetBeans. Assim, em certo sentido, o NetBeans é um conjunto de bibliotecas.

Uma desvantagem deste tipo de modelo é que os *plug-ins* não podem chamar funcionalidades que não aparecem nas APIs abertas, por exemplo, um objeto que tem uma interface declarada na API aberta e que modelará uma classe que implementa essa interface mas possui alguns métodos que não são definidos na API. O uso de uma API garante acesso a características que continuarão funcionando mesmo se a implementação for alterada, porém, isto implica que não há nenhuma garantia que qualquer característica não declarada na documentação das APIs abertas estarão presentes nas versões futuras do IDE, assim qualquer código que use as classes que implementam o *core* diretamente e não as interfaces presentes nas APIs podem não funcionar em versões futuras do NetBeans.

3.2 O Desenvolvimento de Plug-ins no IDE NetBeans

Os *plug-ins* interagem com o NetBeans através das APIs Abertas, sendo assim, para o desenvolvimento dos *plug-ins* faz-se necessário entender como as APIs abertas são implementadas e usadas. Nesta seção serão introduzidos os principais conceitos utilizados pelos *plug-ins* que estenderão o IDE.

Internamente o NetBeans chama os *plug-ins* de módulos, assim, quando nos referirmos a módulos, estamos nos referindo a *plug-ins* NetBeans.

3.2.1 As APIs Abertas (Open APIs)

As APIs abertas são o conjunto de interfaces, definidas no pacote `org.openide` e em seus subpacotes, e suas especificações (como arquivos `manifest` e camadas XML) definidas na documentação. Elas podem ser definidas, também, como as interfaces dos módulos com o NetBeans. As APIs abertas podem ser divididas em seções, como segue:

Sistema de Arquivos: Provê comunicação com a camada de persistência dos dados;

Sistemas de Dados: Provê reconhecimento e interpretação de tipos diferentes de dados;

Ações: Provê a abstração da invocação dos usuários;

Nós: Provê relacionamento hierárquico entre dados e objetos e alguns aspectos de como eles são apresentados para o usuário;

Explorer: Provê apresentação das estruturas hierárquicas dos dados.

Serviços/Lookup: Provê a localização de objetos ou serviços providenciados pelos módulos, que podem ser usados por outros módulos ou invocados pelo usuário. Os serviços são usados normalmente para executar operações complexas em grupos de objetos, como compilação, execução e procura.

Sistema de Janela: Provê manipulação e configuração de janelas e componentes visuais da interface com o usuário.

3.2.2 Modularidade

A Modularidade é uma das características principais do NetBeans. Ele implementa a idéia de que funcionalidades podem ser discretamente trabalhadas e a possibilidade de adicioná-las e removê-las tranquilamente [GRE 2002]. A API dos módulos define o que um módulo é e como instalá-lo e desinstalá-lo. Cada módulo pode oferecer, também, sua própria API para que outros módulos utilizem as suas funcionalidades, por exemplo, os módulos XML oferecem suporte básico a documentos XML e podem ser usados por outros módulos que desejem trabalhar com XML. Para casos como esses, existe um caminho para declarar as dependências entre os módulos. O NetBeans *runtime* não permitirá que um módulo seja instalado se este módulo requer um outro que não está presente, como também não permitirá que um módulo seja desabilitado, se outros dependem dele.

3.2.3 Hierarquia, Arquivos e Nós

Aplicações grandes e extensíveis necessitam dispor de caminhos para que os componentes do sistema criem objetos e notifiquem outras partes do sistema que utilizam esses objetos. Elas necessitam, também, de caminhos para apresentar esses objetos para o usuário.

É necessário um sistema genérico que possa fornecer funcionalidades específicas para cada tipo de dado. Para resolver este problema, existe um paradigma muito conhecido utilizado nos Sistemas Operacionais: Arquivos. Aplicações rodando em um sistema operacional criam arquivos no disco e trabalham com ele. O sistema operacional providencia serviços de baixo-nível para criar e acessar arquivos, e não está interessado no conteúdo dos arquivos criados pelos usuários. Os módulos no NetBens são razoavelmente análogos a aplicações em um sistema operacional, e o *core* gerencia os objetos persistidos criados pelos usuários [GRE 2002]. O NetBeans tem um conceito de um sistema de arquivos, que em seu sentido mais básico, significa uma área de armazenamento em que os arquivos podem ser escritos e lidos. Em outro sentido, o sistema de arquivos pode ser entendido como uma fábrica de objetos JAVA.

Para o segundo problema, a apresentação, o NetBeans provê uma abstração sem conhecimento de conteúdo, o “nó”, que é usado para conter a representação hierárquica dos dados e a apresentação desses dados através da interface com o usuário. Um nó não é um recipiente de dados, nem um ponteiro para os dados, ele é, assim como os sistemas de arquivos, um objeto de dados,

que identifica tipos de dados e agrega múltiplos arquivos relacionados em uma única entidade.

3.2.4 A Abstração de Arquivos no NetBeans

O sistema de arquivos no NetBeans é um lugar onde pode-se, hierarquicamente, ler e escrever arquivos. Mas o NetBeans não requer que um sistema de arquivos trabalhe necessariamente com arquivos no disco, somente requer que ele se comporte como tal. Isto pode ser realizado graças a implementação de um conjunto de interfaces definidas na API de sistemas de arquivos.

Um sistema de arquivos precisa somente estar de conformidade com esta interface. Como e onde os dados são fisicamente persistidos é irrelevante e transparente para o código que trabalha com os arquivos. Um sistema de arquivos é, efetivamente, um espaço de nomes hierárquico para entidades nomeadas que contêm dados ou, contêm entidades que contêm dados, ou seja, uma interface de fornecimento de serviço (Service Provider Interface - SPI) para estender o NetBeans com tipos alternativos para armazenamento de arquivos. Como um exemplo concreto, o NetBeans define, e internamente usa, um sistema de arquivos XML que é, para todas as intenções e finalidades, análogo a arquivos físicos em disco [GRE 2002]. Assim, existem extensões para suportar FTP, CVS e outros tipos de armazenamento disponíveis em módulos separados.

Se para existir um sistema de arquivos, a única necessidade é satisfazer as características mostradas acima, é possível existir um sistema de arquivos virtual

que possui uma coleção de subsistemas de arquivos e apresenta todos eles no mesmo espaço virtual. O NetBeans contém uma implementação com essas características na classe `MultiFileSystem`, que faz parte da API de sistemas de arquivos. Esta classe permite que seja construído um simples espaço de nomes que contém um conjunto de subsistemas de arquivos, chamados “camadas”, e funciona como se o conteúdo de todos eles estivessem no mesmo espaço de nomes, como pode ser visto na Figura 4.



Figura 4 – MultiFileSystem [GRE 2002].

Se duas camadas diferentes contém uma pasta chamada `/MinhaPasta/` com conteúdos diferentes, ao listar-se o conteúdo de `MultiFileSystem's / MinhaPasta /` simplesmente o conteúdo das duas pastas serão listados. Para negociar com colisão de nomes (por exemplo, duas camadas contém arquivos diferentes com o mesmo nome e caminho), a classe `MultiFileSystem` possui uma ordem de empilhamento das camadas. No caso de um conflito, qualquer sistema de arquivo no topo da pilha terá preferência. É possível, também, incluir ou remover camadas em tempo de execução.

3.2.5 Mapeando Arquivos Para Objetos

Apenas dispor de maneiras de salvar arquivos não é suficiente em um ambiente integrado, é necessário dispor de maneiras de salvar instâncias de objetos. Assim, o NetBeans define semânticas para mapear arquivos para objetos. Pode-se criar, por exemplo, um arquivo de nome com-mycom-MyClass.instance. A API de sistema de dados contém uma facilidade (implementada em uma classe chamada `InstanceDataObject`) que referencia um arquivo pelo nome e retorna uma instância da classe nomeada, assim, uma instância da classe `MyClass` seria retornada.

Usando-se essa infra-estrutura, arquivos podem ser fábricas para objetos. Esta infra-estrutura serve para duas finalidades principais no NetBeans, primeiro: habilita módulos a registrarem objetos no sistema (por exemplo, adicionar BEANS para a paleta de componentes ou adicionar itens em um menu); segundo: permite que módulos sejam registrados sem que a JVM carregue a classe em questão até que seja realmente necessário, não usando memória desnecessariamente.

No topo desta infra-estrutura está a facilidade chamada *lookup*. *Lookup* adiciona uma camada para ação indireta permitindo que o código do módulo procure objetos registrados através de arquivos em pastas especiais sem trabalhar diretamente com sistemas de arquivos [GRE 2002].

Nós mencionamos na seção 3.1.2 o conceito de um sistema de arquivos XML, onde o conteúdo do sistema de arquivos está atualmente representado em um documento XML. No sistema de arquivos XML do NetBeans, o XML para

adicionar uma classe JAVA para um sistema de arquivos seria algo como o código mostrado na Figura 5.

```
<filesystem>
  <folder name="Menu">
    <folder name="View">
      <file
        name="org-netbeans-examples-quickpanel-ShowQuickPanelAction.instance"/>
      </folder>
    </folder>
  </filesystem>
```

Figura 5 - Adicionando um .instance para o Sistema de Arquivos [GRE 2002].

O resultado do exemplo detalhado na figura acima é que quando o sistema cria o menu principal, ele cria uma instância de `org.netbeans.examples.quickpanel.ShowQuickPanelAction`, que fornece o ícone e mostra o nome da ação do item de menu que será adicionado. Quando o usuário selecionar o ícone do item de menu adicionado, o método `actionPerformed()` da instância criada será chamado.

3.2.6 O Sistema de Arquivos do NetBeans

O Netbeans não somente usa o conceito de sistema de arquivos virtual para manipular arquivos do usuário, como também usa um sistema de arquivos virtual especial que contém informações de configurações do próprio NetBeans, esse sistema é chamado de Sistema de Arquivos. Usando convenções, como arquivos com nome terminado em `.instance` representam instâncias de classes, e outros mecanismos semelhantes, para representar objetos JAVA, o NetBeans armazena uma grande variedade de informações no seu sistema de arquivos. Por

exemplo, o sistema de arquivos contém uma pasta chamada /Menu que contém subpastas com nomes como Arquivo e Editar. Essas subpastas, por sua vez, contém arquivos .instante para classes JAVA que implementam ações que aparecem nos menus Arquivo e Editar. Os módulos são livres para criar suas próprias pastas no sistema de arquivos do NetBeans e armazenar dados que interessem para eles, ou adicionar objetos em pastas do NetBeans.

Uma das razões para o NetBeans usar sistema de arquivos é que ele permite que objetos sejam declarados, mas suas classes não sejam carregadas pela JVM enquanto não forem utilizadas, não ocupando memória desnecessariamente [GRE 2002]. O sistema de arquivos do NetBeans é o registro geral para dados e objetos publicamente acessados. Um dos aspectos importantes do sistema de arquivos virtual do NetBeans é que ele pode disparar eventos para notificar o resto do sistema quando alguma alteração ocorre. Por exemplo, o Netbeans é notificado quando ocorrem alterações no sistema de arquivos e, se foi criado um item de menu, ele mostrará o novo item na barra de menus.

3.2.7 Camadas dos Módulos

A primeira maneira dos módulos adicionarem suas funcionalidades no ambiente é através de arquivos, especificamente através de arquivos virtuais definidos na camada XML do sistema de arquivos. Para isso, existe um pequeno documento XML nos módulos, que segue o padrão DTD do sistema de arquivos do NetBeans, declarado no arquivo *manifest* do módulo.

Como a infra-estrutura do NetBeans permite que arquivos sejam mapeados em instâncias de classes JAVA, o que um módulo geralmente instala são instâncias de classes, usando a convenção `.instance` declarada abaixo. Os arquivos são colocados em pastas no sistema de arquivos que tem significados conhecidos ou para o sistema ou para o módulo. A camada XML define uma hierarquia de arquivos e pastas que pode ou não sobrescrever pastas existentes no sistema de arquivos. Quando um módulo é carregado, este pequeno sistema de arquivos XML é fundido com o sistema de arquivos do NetBeans, como mostra a Figura 6.

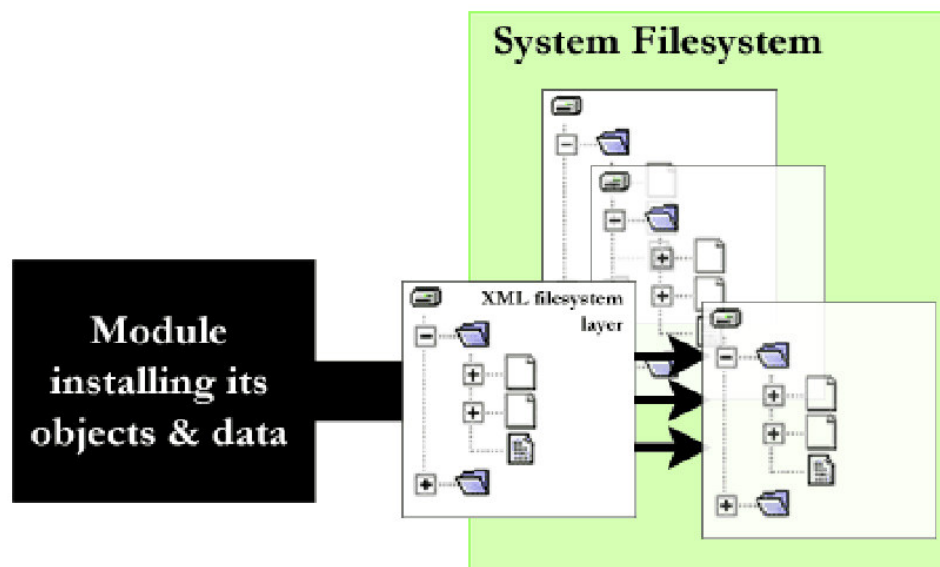


Figura 6 - Módulo adicionando uma camada XML no sistema de arquivos [GRE 2002].

Freqüentemente um módulo precisa instalar objetos que podem ser disponíveis para o *runtime* (como as classes que definem ações para os itens de menu), ou para outros módulos (por exemplo, serviços, como compilação). O

sistema de arquivos provê um caminho para isto ser feito mantendo-se o acoplamento fraco entre os módulos e o sistema, e assim sendo, nenhum deles precisa ser programado para isso, tudo é descrito através do XML. Quando um módulo é removido, sua camada XML é extraída do sistema de arquivos, e os objetos, itens de menus, serviços e qualquer outra funcionalidade que esse módulo ofereça simplesmente desaparecerão do sistema.

3.2.8 Camadas no Sistema de Arquivos

Como mencionado anteriormente, uma classe chamada MultiFileSystem pode representar um conjunto de sistemas de arquivos como se fossem um. O sistema de arquivos é composto de três camadas distintas, que determinam onde as configurações são gravadas, essas camadas são detalhadas abaixo:

Padrão ou Global: Este é o MultiFileSystem que contém o conteúdo da subpasta system/ do diretório onde o NetBeans está instalado. O conteúdo deste sistema de arquivos é fundido com qualquer sistema de arquivos XML dos módulos que são empacotados com o NetBeans. As mudanças nesta camada serão propagadas para todos os usuários.

Sessão ou Usuário: Esta camada contém os diretórios de configuração do usuário, nela são gravadas informações como as posições das janelas, cores dos editores, etc. As configurações são específicas para cada usuário e não podem ser aplicadas globalmente.

Projeto: A camada de projeto contém configurações aplicadas ao projeto corrente (que compreende um conjunto de arquivos, posições de janelas e assim por diante). É projetada para armazenar ajustes específicos de um determinado projeto.

Sob esse ponto de vista, o sistema de arquivos poderia ser definido como um MultiFileSystem que contém outros MultiFileSystems, a Figura 7 detalha esse conceito e as camadas existentes no sistema de arquivos.

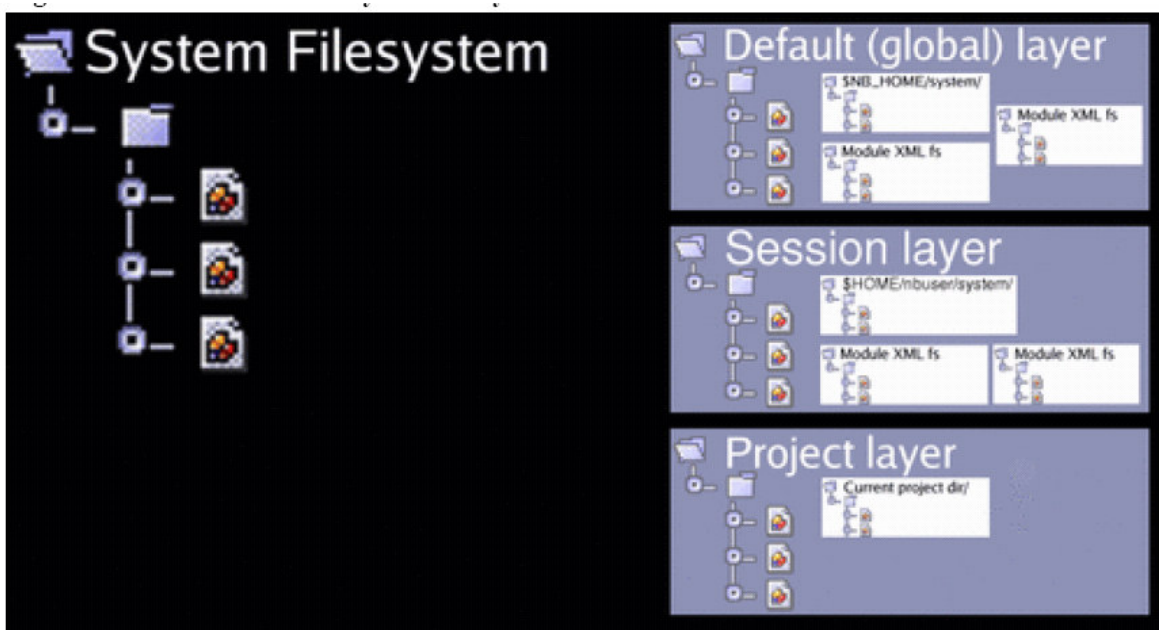


Figura 7 - Estrutura do sistema de arquivos [GRE 2002].

4 O *Plug-in* SVN Proposto

O *plug-in* SVN proposto pretende se parecer ao máximo, do ponto de vista de utilização, com o *plug-in* CVS padrão do NetBeans para tornar mais fácil a migração dos usuários. Em alguns casos manter essa semelhança não será possível de devido às diferenças de arquitetura entre os dois sistemas, e em outros casos, algumas mudanças serão realizadas propositalmente com o intuito de melhorar características do *plug-in* CVS.

O *plug-in* CVS possui um menu na barra de menus que contém os itens para acessar suas funcionalidades, como descrito na Figura 8.

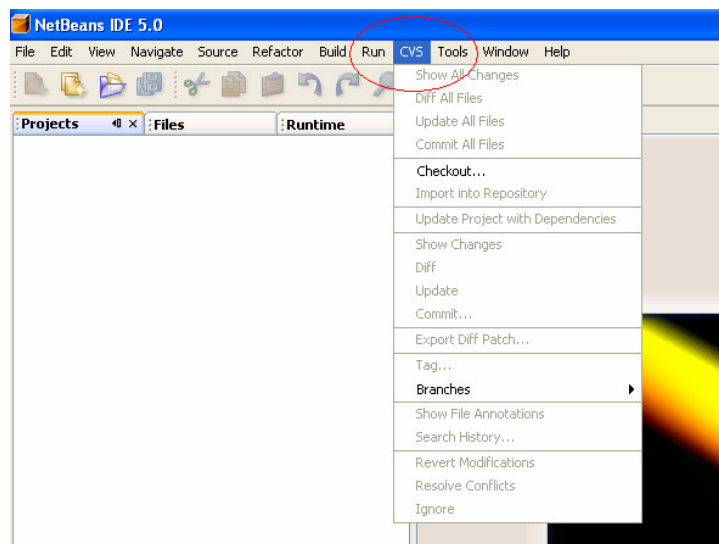


Figura 8 - Localização do CVS na barra de menus do NetBeans.

As ferramentas de controle de versão deveriam estar localizadas em um item de um outro menu, como do menu ferramenta, por exemplo. Porém, para manter o padrão utilizado pelo *plug-in* CVS, nós criamos um menu SVN que possui três itens, um para cada operação implementada, *checkout*, *commit* e *update*, como mostra a Figura 9.

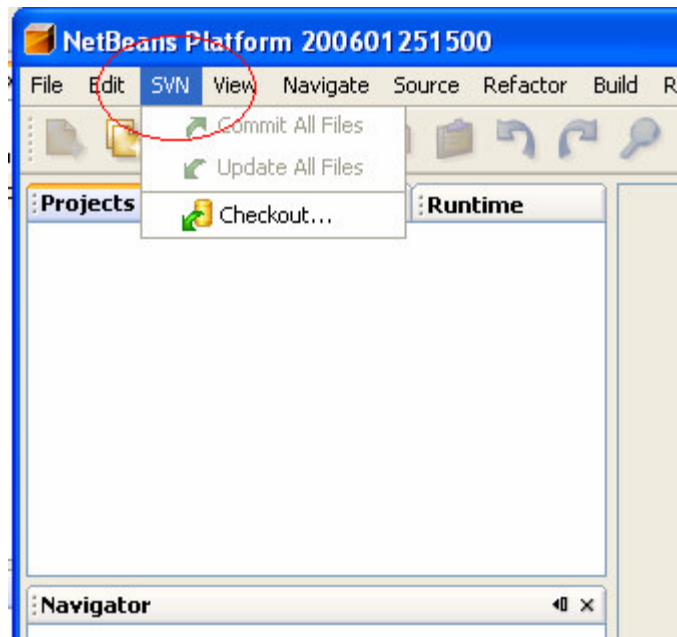


Figura 9 - Localização do SVN na barra de menus do NetBeans.

Optamos por ilustrar as operações com ícones para facilitar o entendimento das ferramentas pelos usuários leigos. Foram utilizadas as mesmas figuras padrão do cliente SVN para Windows e do *plug-in* SVN para Eclipse, para manter a padronização e facilidade de utilização.



Figura 10 - Ícones ilustrando as operações SVN.

Ao clicar em um desses itens de menu, o usuário executa a operação especificada. Para realizar a operação de *checkout* implementamos um *wizard* composto de 2 telas, a primeira onde o usuário insere o servidor ao qual se deseja

conectar, o usuário e a senha para a autenticação neste servidor, como detalha a Figura 11.

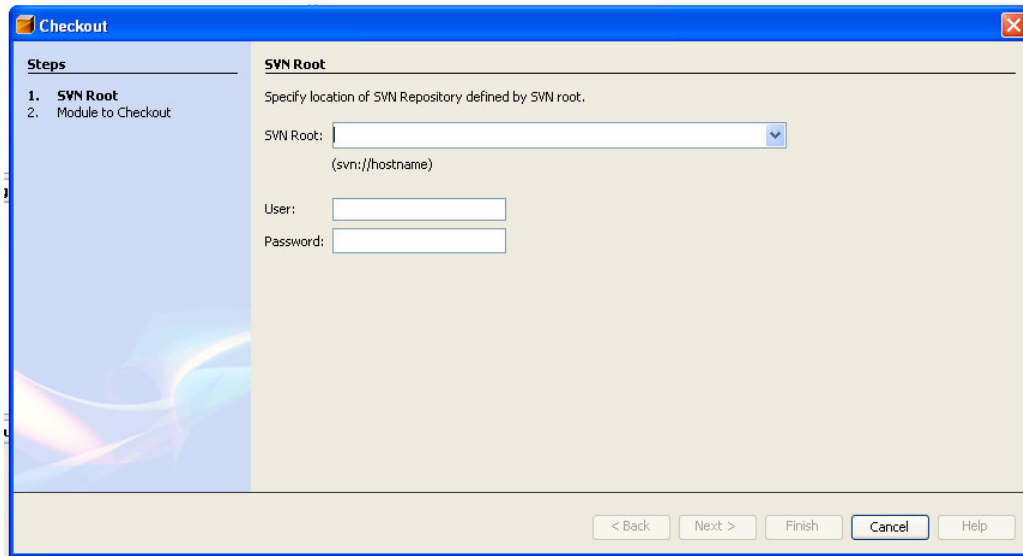


Figura 11 - Primeira tela do Wizard de Checkout.

Ao preencher os três campos obrigatórios o botão “Next” é habilitado, e ao clicar neste botão, o *plug-in* valida os dados e tenta a conexão com o servidor especificado utilizando os dados de autenticação fornecidos. Este pode ser um processo lento pois o servidor fornecido pode ser um servidor localizado na internet tentando ser acessado por uma conexão com pouca banda, ou, no caso do servidor não responder, será necessário esperar um *timeout* de conexão, que pode durar alguns minutos. Assim, o usuário deve ser avisado que o processo de conexão está em curso, pois uma tarefa demorada que não retorna um *feedback* para o usuário pode ser interpretada como um travamento do sistema. Para este fim, inserimos uma barra de progressão que indica para o usuário o estado da tarefa, como vemos na Figura 13.

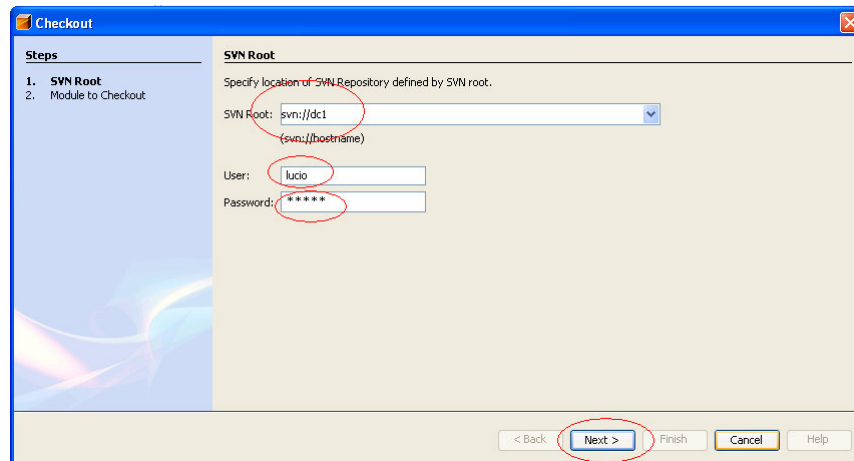


Figura 12 - O preenchimento dos campos obrigatórios habilita o botão *Next*.

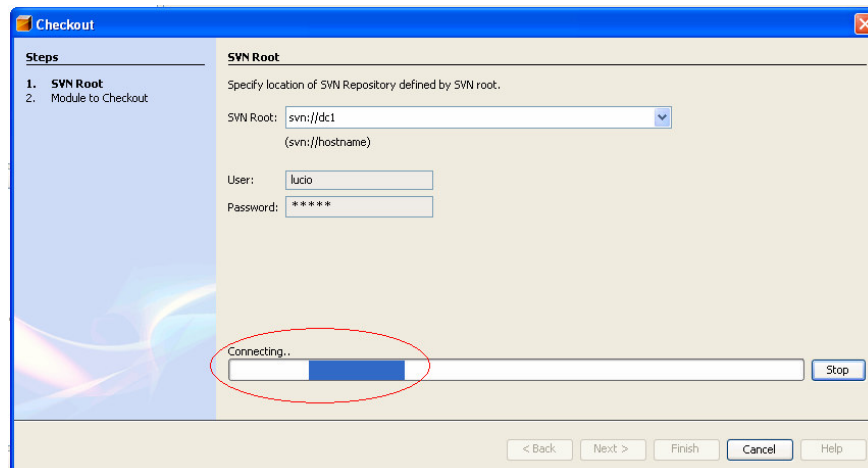


Figura 13 - Barra de progresso indicando a tarefa de conexão em curso.

O usuário pode encerrar a tarefa clicando no botão *Stop* e encerrando, assim, a tentativa de conexão. Se um erro ocorrer na tentativa de conexão, será apresentado ao usuário o motivo do erro da forma padrão dos *wizards* do NetBeans, como podemos ver a seguir, na Figura 14.

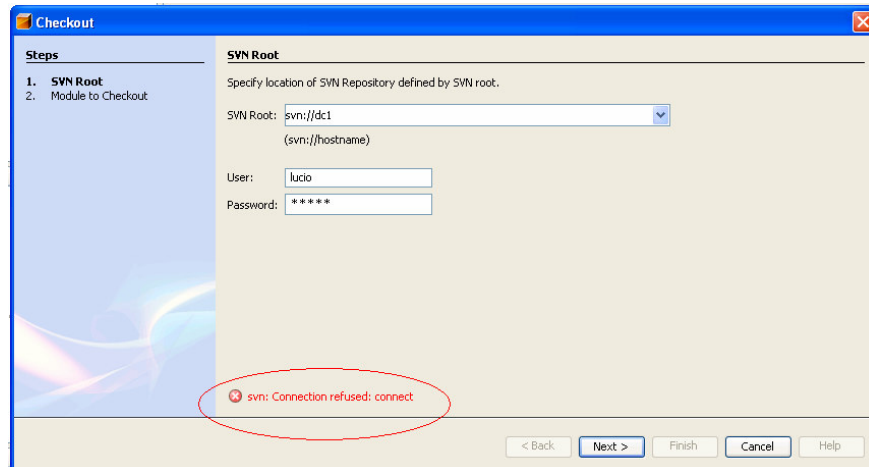


Figura 14 - Exemplo de mensagem de erro na tentativa de conexão.

Se a conexão tiver sucesso, a próxima tela do *wizard* será apresentada. Nela o usuário seleciona o item a ser feito *checkout*, o diretório na máquina local onde o item será salvo e a versão do item que deseja receber, como podemos ver na Figura 15.

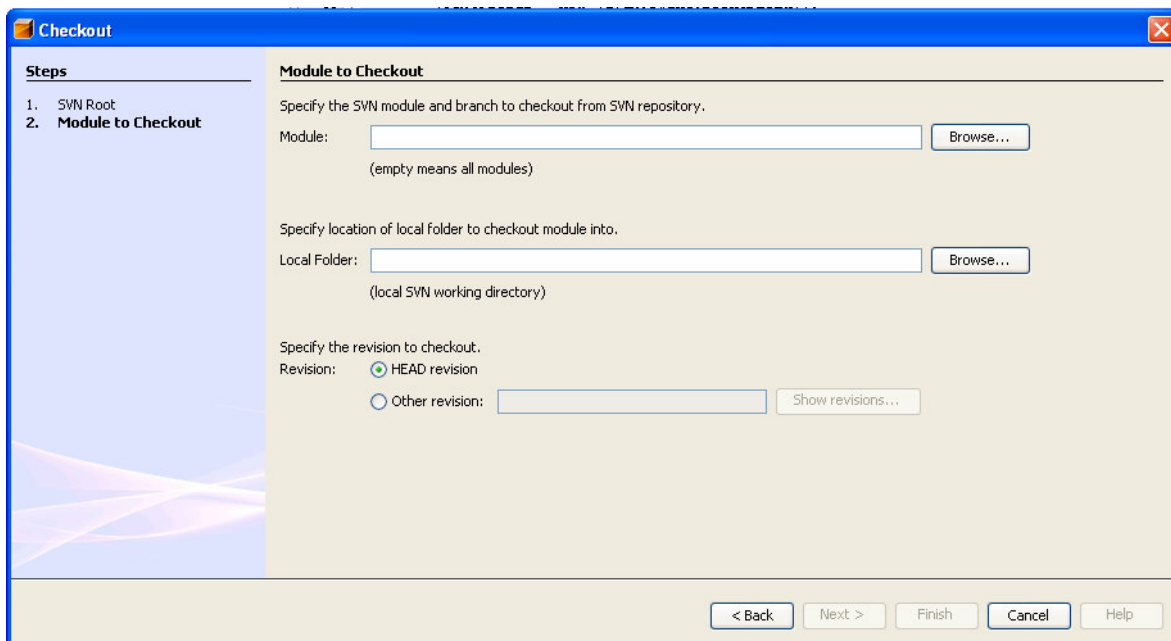


Figura 15 - Segunda tela do *Wizard* de *Checkout*.

Para auxiliar o usuário na escolha do item a ser feito *checkout* e do diretório onde o item será salvo, duas janelas de auxílio foram implementadas. Elas podem ser acessadas através dos respectivos botões “*Browse...*”, como mostra a Figura 16.

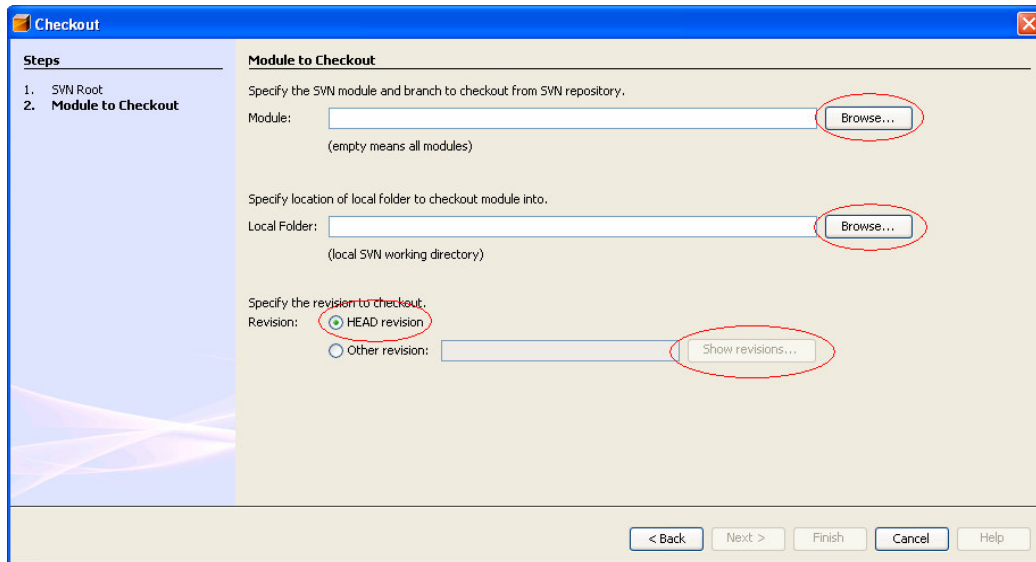


Figura 16 - Botões para acesso as janelas de auxílio no Wizard de Checkout.

Ao clicar neste botão a janela de exploração do repositório, ilustrada na Figura 17, se abrirá. Ela apresentará os itens disponíveis no servidor para *checkout*. Segundo [WU 2003], os usuários de um sistema de controle de versão estão interessados em saber algumas informações específicas ao realizar uma operação em um repositório de dados, principalmente, “quem” criou um item e “quando” este item foi criado. Essas informações não são apresentadas no processo de *checkout* do *plug-in* CVS e achamos relevante disponibilizar estas informações para o usuário neste passo. Assim, o usuário pode escolher o item que deseja fazer *checkout* clicando nele e, em seguida em “OK”.

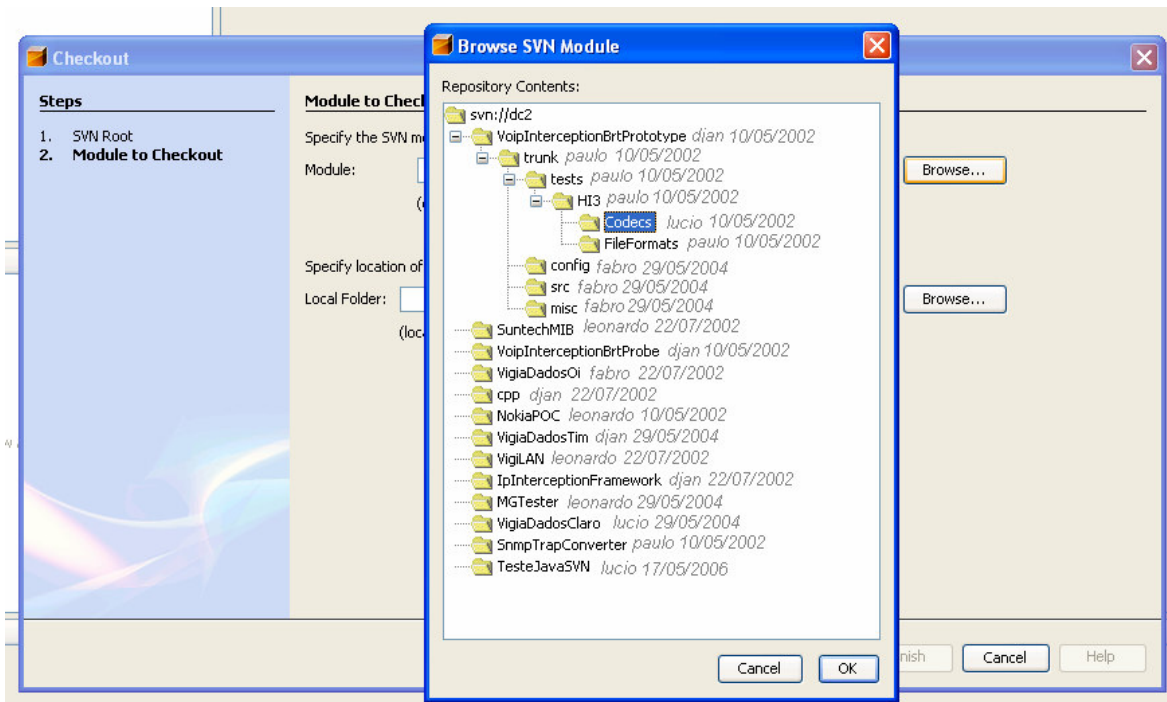


Figura 17 - Janela de exploração do repositório.

Ao clicar no outro botão “*Browse*” uma janela de escolha de diretórios será exibida. O usuário pode navegar pelos seus diretórios locais e escolher o local em que salvará os itens que fez *checkout*. Nesta janela o usuário apenas deve poder escolher um diretório e não um arquivo, assim, ele só tem acesso aos diretórios. Feita a escolha, o usuário clica no botão “*Open*” e a tela é fechada, a janela de escolha de diretórios é mostrada na Figura 18.

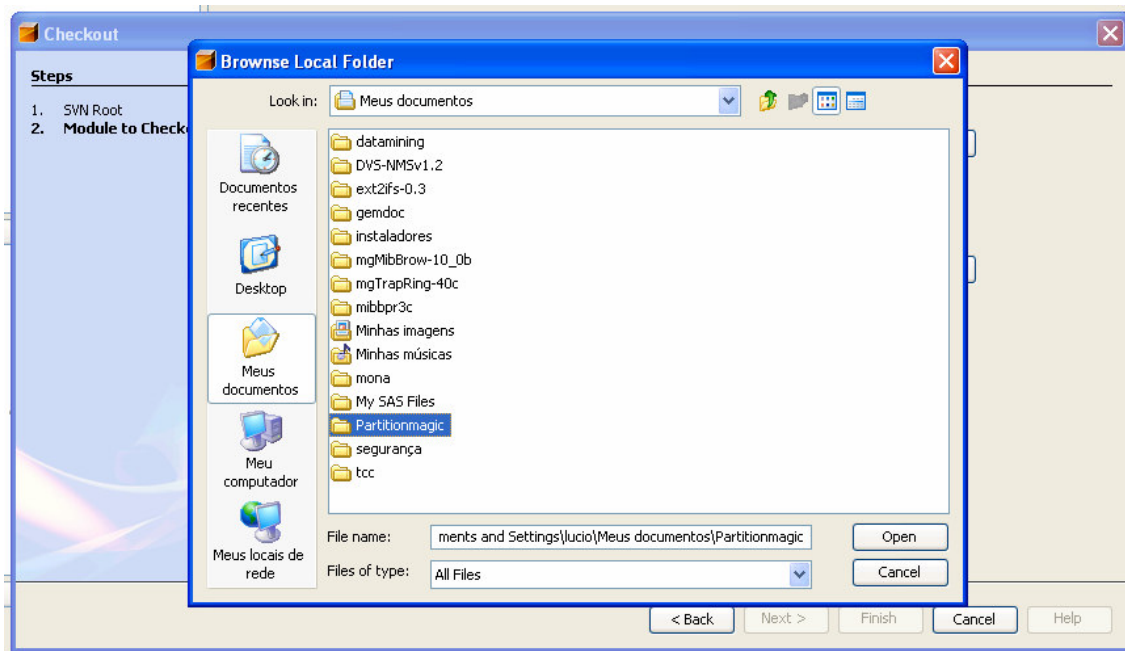


Figura 18 - Janela de escolha do diretório a ser feito o *checkout*.

Por padrão, o usuário realizará o *checkout* da última versão do item escolhido, a chamada versão *head*. Mas, se desejar, o usuário pode clicar *radio box* “*Other revision*” e inserir a versão desejada. O *plug-in* fornece uma opção de busca da versão desejada, clicando no botão “*Show revisions*”, como mostra a Figura 16. Ao clicar nesse botão uma janela para pesquisa da versão desejada se abrirá. Essa janela apresenta as modificações realizadas nessa versão, o autor, a data e o comentário adicionado, como podemos ver na Figura 19.

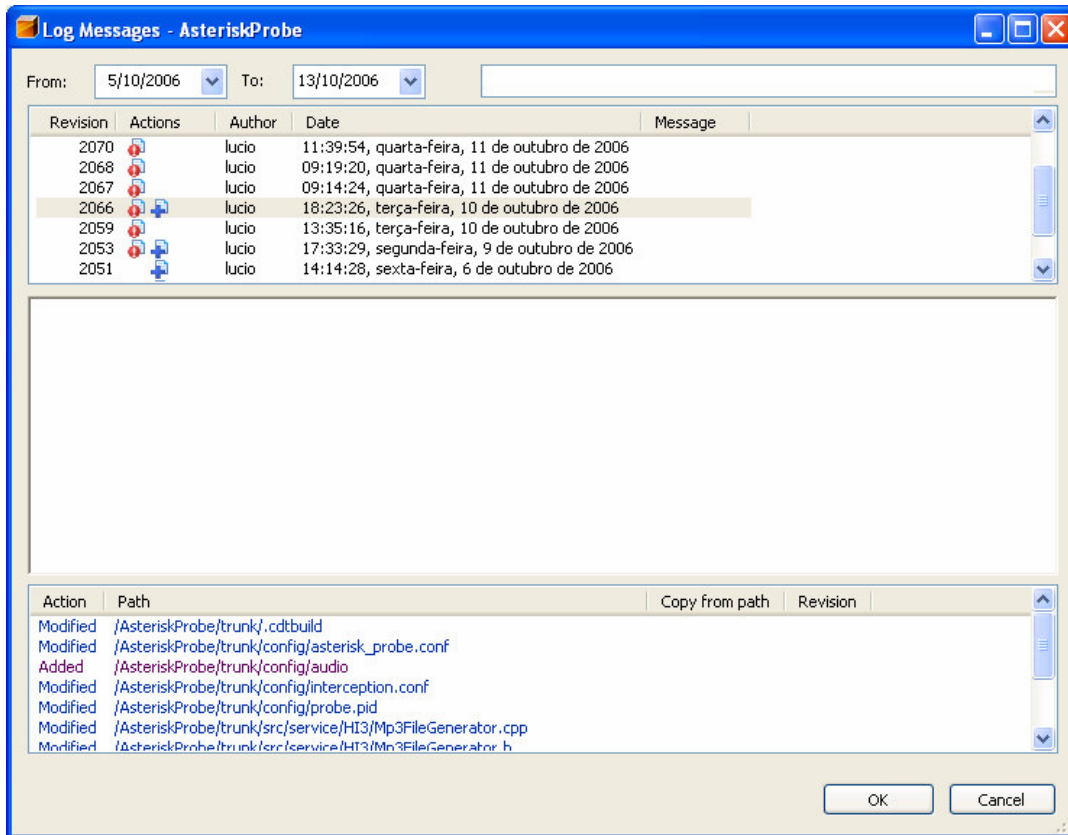


Figura 19 - Janela de pesquisa das versões de um artefato.

Após o preenchimento dos campos “*Module*”, “*Local Folder*” e da escolha da versão o botão “*Finish*” será habilitado. Este é o último passo do *wizard*, ao clicar no botão “*Finish*” a operação de *checkout* será realizada. Esta também pode ser uma tarefa demorada, pois os itens escolhidos do repositório remoto serão gravados localmente. Por isto, durante essa tarefa uma barra de progresso também é mostrada, como ilustra a Figura 20. No *plug-in CVS*, a janela de *wizard* é fechada e uma barra de progresso no canto inferior do IDE é mostrada, porém, se algum erro ocorrer durante o processo, o usuário não tem a possibilidade de alterar os dados no *wizard*. Optou-se por apresentar a barra de progresso na janela de *wizard* e só fechá-la no fim da tarefa. Se um erro ocorrer, ele será

mostrado para o usuário, vide Figura 21, e o usuário terá a opção de alterar alguns dados ou cancelar a tarefa.

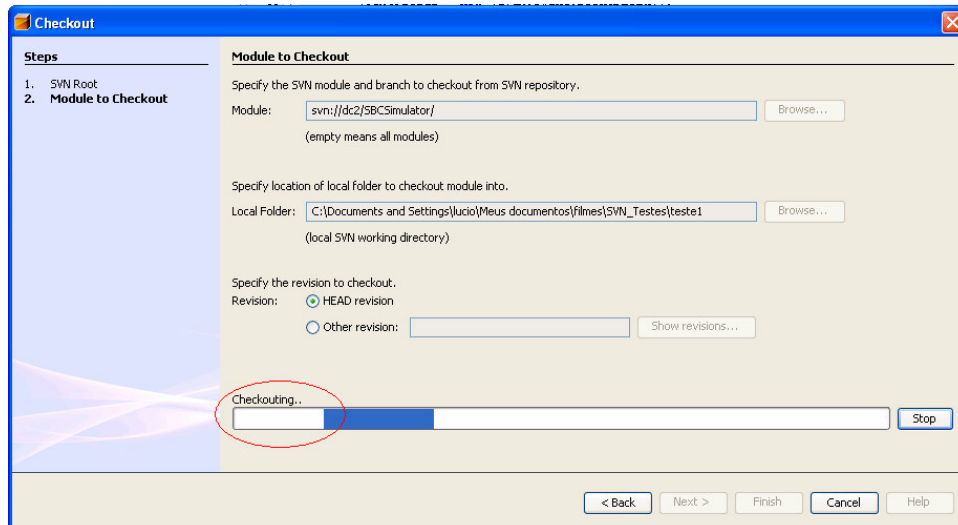


Figura 20 - Barra de progresso indicando o andamento da tarefa de *checkout*.

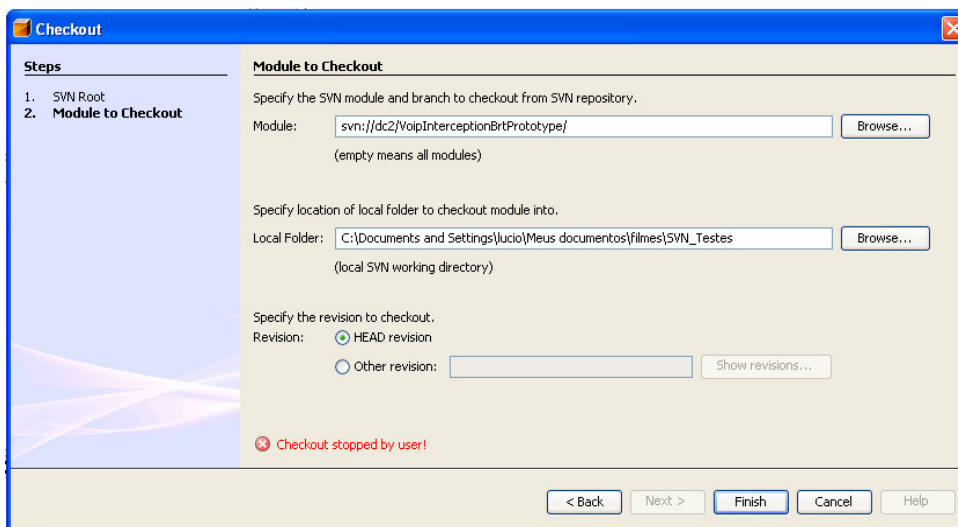


Figura 21 - Exemplo de erro ao executar a operação de *checkout*.

O usuário tem a possibilidade de interromper a tarefa de *checkout* através do botão “*Stop*”.

Ao final do *checkout*, dois caminhos distintos podem ser tomados. Se o item recebido do servidor é um projeto NetBeans, será apresentada ao usuário a opção de abrir esse projeto, como mostra a Figura 22, se não, será apresentada ao usuário a opção de criar um projeto com os dados recebidos, como pode ser visto na Figura 23, se ele aceitar, o *wizard* de criação de projetos do NetBeans será iniciado.

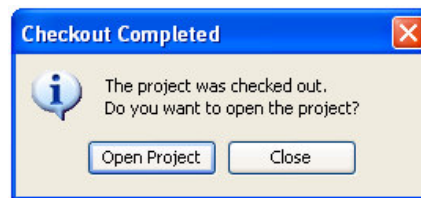


Figura 22 - O *plug-in* oferece a possibilidade de abrir um projeto recebido do repositório.

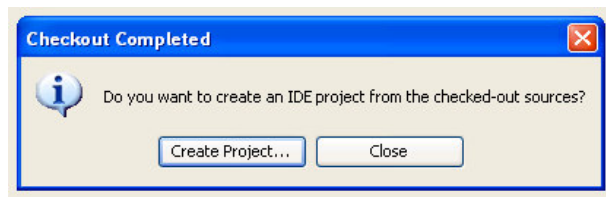


Figura 23 - O *plug-in* fornece a possibilidade de criar um projeto a partir dos dados do *checkout*.

Ao clicar no item de menu *Commit* no menu SVN do NetBeans, o *plug-in* varrerá os projetos do IDE e apresentará a janela ilustrada na Figura 24. Essa janela contém todos os artefatos dos projetos que possuem controle de versão via Subversion. O usuário seleciona os artefatos que deseja atualizar no repositório e clica no botão “*Commit*”. O *plug-in*, então, realizará a operação de *commit* para os artefatos selecionados.

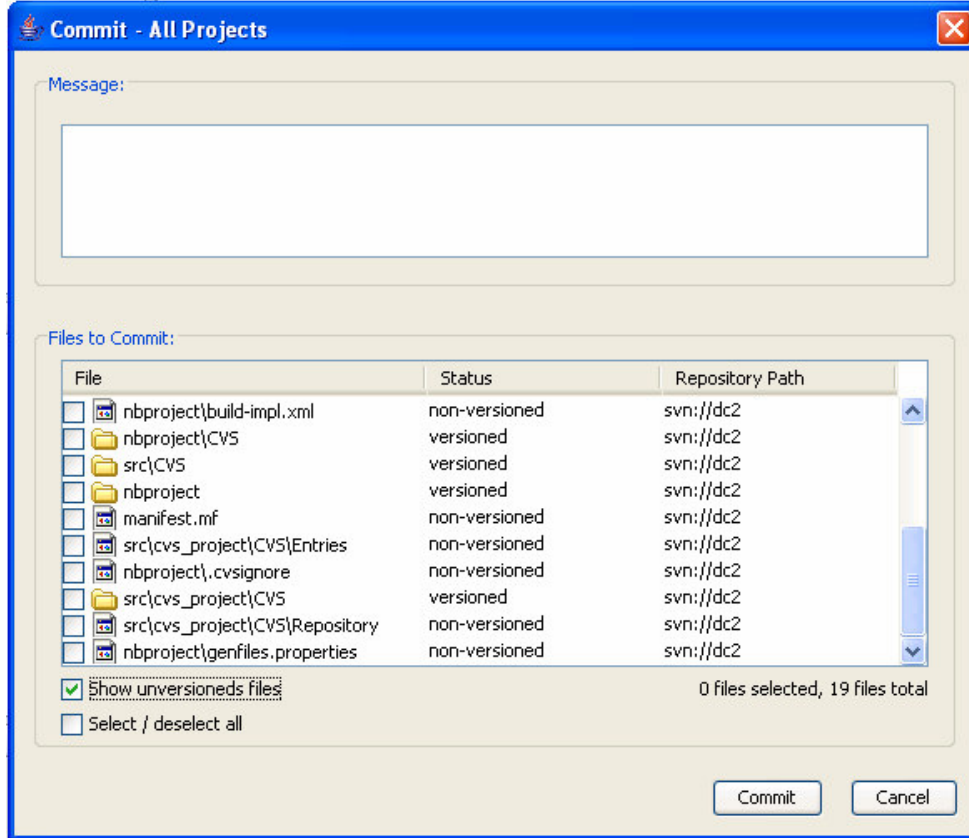


Figura 24 - Janela com os artefatos para *commit*.

Ao clicar no item de menu *Update* do menu SVN do NetBeans, o *plug-in* realizará a operação de *update* para todos os artefatos que possuem controle de versão via Subversion e que tem modificações a serem recebidas do repositório. O *status* dessa operação é enviado para uma janela de *log*, ilustrada na Figura 25.

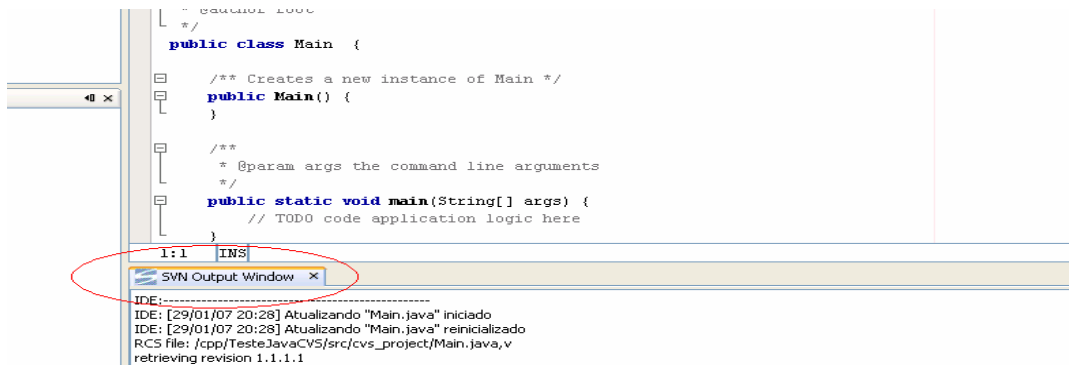


Figura 25 - Janela de log das operações.

4.1 O Processo de Desenvolvimento do Plug-in Proposto

O processo de desenvolvimento de software deve ser iterativo e incremental, guiado por casos de uso e com ênfase na definição da arquitetura [BOO 96]. Assim, nós identificamos três casos de uso primários que foram atacados em três ciclos de desenvolvimento, cada ciclo de desenvolvimento atacou um caso de uso específico. O primeiro ciclo de desenvolvimento resultou em um protótipo que realizava a operação de checkout, ao fim do segundo ciclo de desenvolvimento tínhamos um protótipo que realizava as operações de checkout e commit, e, por fim, após o terceiro ciclo de desenvolvimento, tínhamos o *plug-in* finalizado.

Um caso de uso é um documento narrativo que descreve a seqüência de eventos de um ator (um agente externo) que usa um sistema para completar um processo [JAC 92], ou seja, descrições narrativas de processos do domínio [LAR 2000]. Com base nos objetivos descritos no primeiro capítulo deste relatório, os três casos de uso primários identificados foram os casos de uso *checkout*, *commit* e *update*. Além destes casos de uso primários nós identificamos um caso de uso

secundário, o caso de uso *startup*. Podemos ter uma visão geral dos casos de uso e dos atores envolvidos através do diagrama de casos de uso da Figura 26.

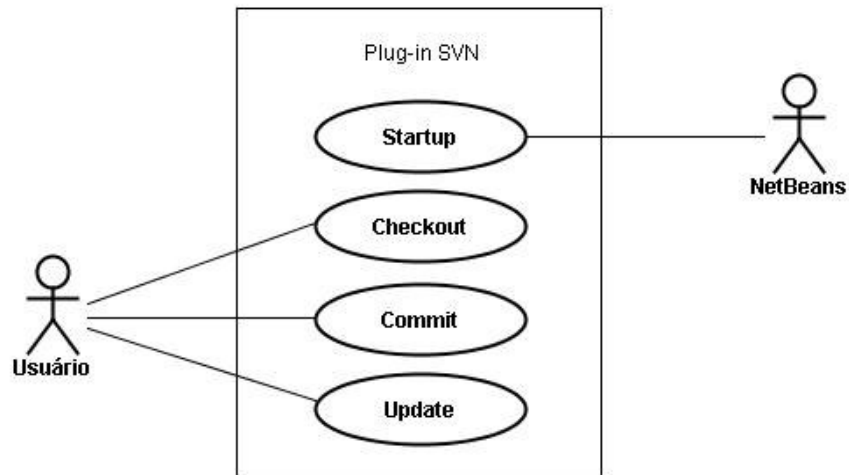


Figura 26 - Diagrama de casos de uso.

Os casos de uso identificados estão detalhados abaixo:

Caso de uso: Checkout

Atores: Usuário

Tipo: Primário

Descrição: O usuário fornece o repositório SVN desejado, os dados de acesso a esse repositório e os artefatos de software a serem recuperados, ao fim, os artefatos solicitados são recuperados.

Caso de uso: Commit

Atores: Usuário

Tipo: Primário

Descrição: O usuário escolhe os artefatos de software que deseja enviar ao repositório SVN, ao fim, a operação de commit é realizada.

Caso de uso: Update

Atores: Usuário

Tipo: Primário

Descrição: O usuário solicita a realização da operação de update, ao final, a operação de update é realizada.

Com base nesses casos de uso podemos ter um rascunho inicial do modelo conceitual, como mostra a Figura 27.

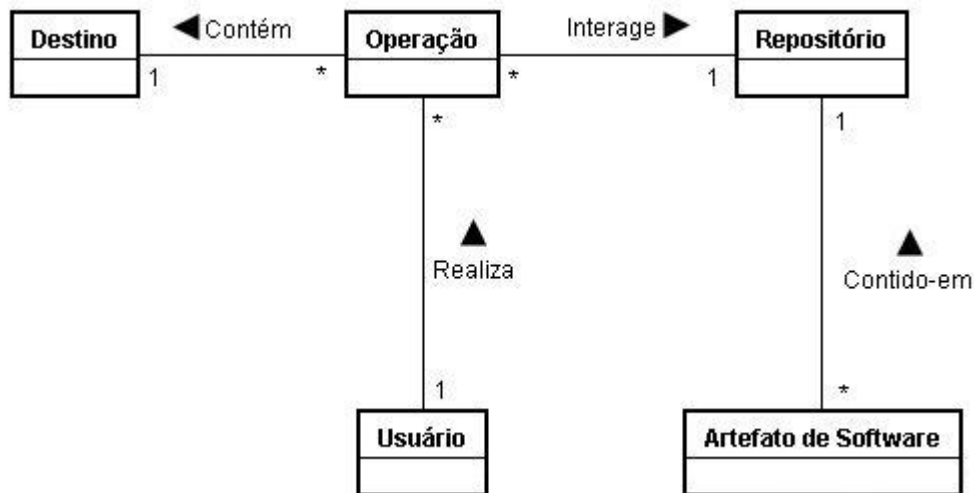


Figura 27 – Rascunho inicial do modelo conceitual.

Com base nos casos de uso identificados e no rascunho do modelo conceitual gerado nós partimos para a execução dos ciclos de desenvolvimento.

Cada ciclo de desenvolvimento englobou fases de análise, projeto, implementação e testes.

Na fase de análise os casos de uso ideais foram refinados gerando os casos de uso essenciais, que são casos de uso expandidos expressos numa forma ideal, a qual permanece livre da tecnologia e de detalhes de implementação [CON 97]. A partir dos casos de uso essenciais foi possível incrementar o modelo conceitual e desenvolver os diagramas de seqüência do sistema. Com esses artefatos UML completos pudemos passar para a fase de projeto.

Na fase de projeto foi desenvolvida uma solução lógica baseada no paradigma orientado a objetos, os casos de uso essenciais da análise deram lugar aos casos de uso reais, que descrevem o projeto em termos da tecnologia concreta de entrada e saída e sua implementação geral, os diagramas de seqüência evoluíram para os diagramas de interação, os quais ilustram como os objetos devem se comunicar de maneira a entender os requisitos, e o modelo conceitual serviu de base para a construção do diagrama de classe de projeto, que sumariza a definição das classes que devem ser implementadas em software.

Na fase de implementação foi gerado código JAVA baseando-se nos diagramas de colaboração e nos diagramas de classe de projeto, foram realizados os testes de unidade e os testes de integração. Nos testes de unidades cada componente do sistema foi testado como se fosse uma caixa preta, testando suas saídas a partir das entradas. Nos testes de integração foi testado o protótipo de cada ciclo de desenvolvimento como um todo, testando a utilização do sistema.

O *plug-in* desenvolvido fez uso da biblioteca Subversion Java (JavaSVN), que é uma biblioteca desenvolvida em JAVA para auxiliar o acesso a repositórios

SVN. A seguir daremos uma breve apresentação desta biblioteca e, em seguida, descreveremos cada fase de cada ciclo de desenvolvimento apresentando os artefatos UML gerados.

4.1.1 A Biblioteca Java SVN

A biblioteca Java SVN é uma biblioteca totalmente desenvolvida em Java que fornece funções para acessar e alterar repositórios Subversion a partir de aplicações Java. Por ser uma biblioteca totalmente desenvolvida em Java ela não necessita de qualquer configuração adicional ou executável para rodar em qualquer sistema operacional que rode Java.

A biblioteca suporta acesso a repositórios via HTTP, HTTPS, SVN e SVN+SSH e possui uma API que permite acesso de baixo nível aos repositórios.

Entre os projetos que utilizam a biblioteca podemos destacar o Subclipse (*Plug-in* Subversion do Eclipse) e o JDeveloper (IDE da Oracle).

4.1.2 Primeiro Ciclo de Desenvolvimento, o Caso de Uso *Checkout*

Como descrito acima, detalharemos cada atividade realizada neste ciclo de desenvolvimento, que gerou, ao final, um protótipo que implementa a operação de checkout.

4.1.2.1 A Fase de Análise

Ao expandir o caso de uso ideal *checkout* chegou-se ao caso de uso essencial descrito abaixo.

Caso de uso: Checkout

Atores: Usuário

Tipo: Primário e essencial

Descrição: O usuário fornece o repositório SVN desejado, os dados de acesso a esse repositório e os artefatos de software a serem recuperados, ao fim, os artefatos solicitados são recuperados.

Seqüência Típica de Eventos

Ação do Autor	Resposta do Sistema
1. Este caso de uso começa quando um usuário solicita a realização de uma operação de checkout.	
2. O usuário registra o repositório SVN desejado e os dados de acesso a esse repositório.	3. Mostra os artefatos de software disponíveis neste repositório e as informações de versão destes artefatos.
4. O usuário registra o artefato e a versão de software que deseja receber do repositório.	5. Acrescenta a informação do artefato à operação corrente e apresenta ao usuário o artefato e a versão escolhida.
6. O usuário registra o caminho (PATH) onde serão colocados os artefatos escolhidos.	
7. O usuário indica que já registrou todas as informações.	8. Realiza a operação solicitada e, ao fim, apresenta uma mensagem de sucesso.

Com base neste caso de uso essencial foi gerado o diagrama de seqüência apresentado na Figura 28.



Figura 28 - Diagrama de seqüência do caso de uso checkout.

Após esses passos o modelo conceitual da análise preliminar foi expandido gerando o modelo conceitual da Figura 29. Este modelo conceitual traz a inserção do conceito “NetBeans” que é responsável por dar início ao *plug-in*. Foram também adicionados os conceitos de “especificação de artefato”, que serve para reduzir informação redundante sobre os artefatos de software e de “artefatos da operação”, que diz respeito aos artefatos envolvidos na operação corrente.

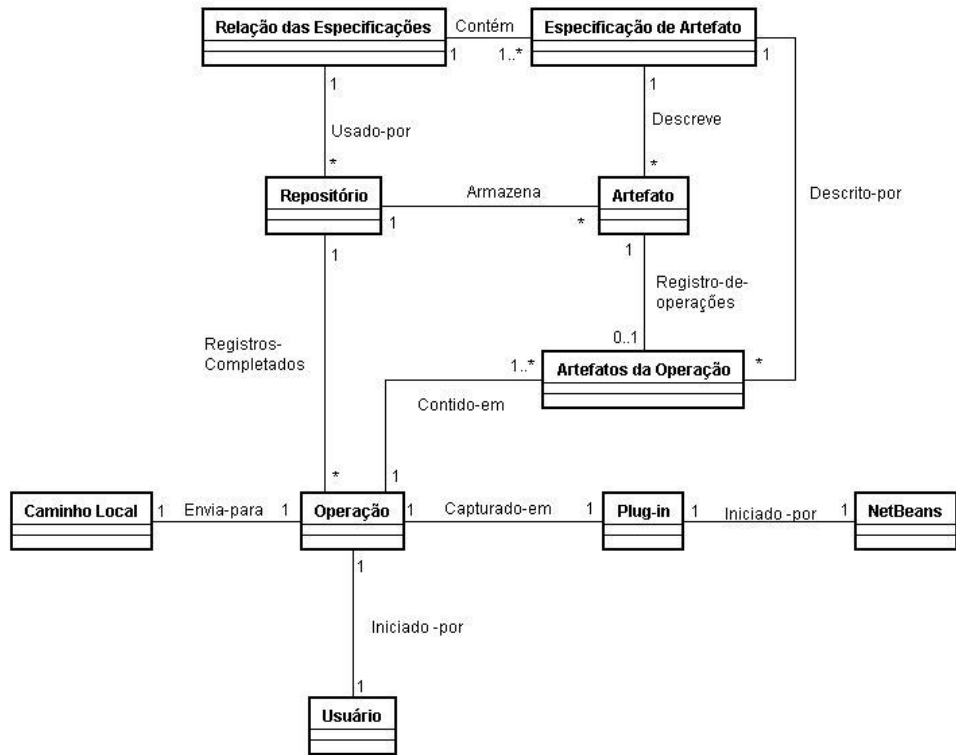


Figura 29 - Modelo conceitual expandido.

A fase de análise enfatiza uma compreensão dos requisitos, dos conceitos e das operações relacionados com um sistema, assim, após a criação destes três artefatos UML, o processo de desenvolvimento encontrava-se maduro o suficiente para partir à fase de projeto, que será descrita a seguir.

4.1.2.2 A Fase de Projeto

Na fase de projeto é desenvolvida uma solução lógica baseada no paradigma orientado a objetos. O coração desta solução é a criação de diagramas de interação, os quais ilustram como os objetos devem se comunicar de maneira a atender os requisitos.

Porém, antes dos diagramas de interação devemos definir os casos de uso reais. Um caso de uso real descreve o projeto real do caso de uso em termos da tecnologia concreta de entrada e saída e sua implementação geral. Eles são criados a partir dos casos de uso essenciais definidos na análise, podemos verificar o caso de uso real *checkout* a seguir. Este caso de uso leva em consideração as figuras: Figura 30, Figura 31 e Figura 32. O *layout* dessa janela foi escolhido tendo em vista a compatibilidade com o *plug-in* CVS do NetBeans, como discutido no início deste capítulo.

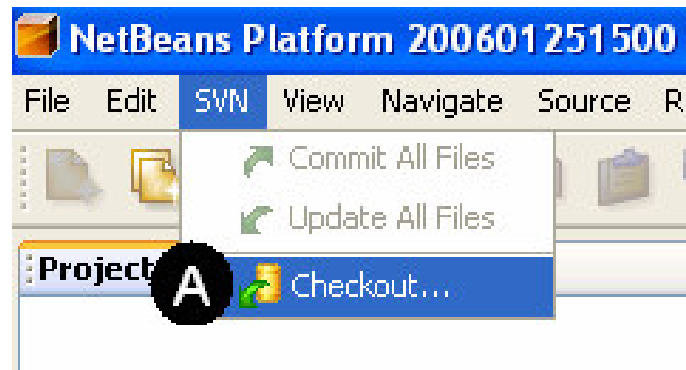


Figura 30 - Solicitando a realização do *checkout*.

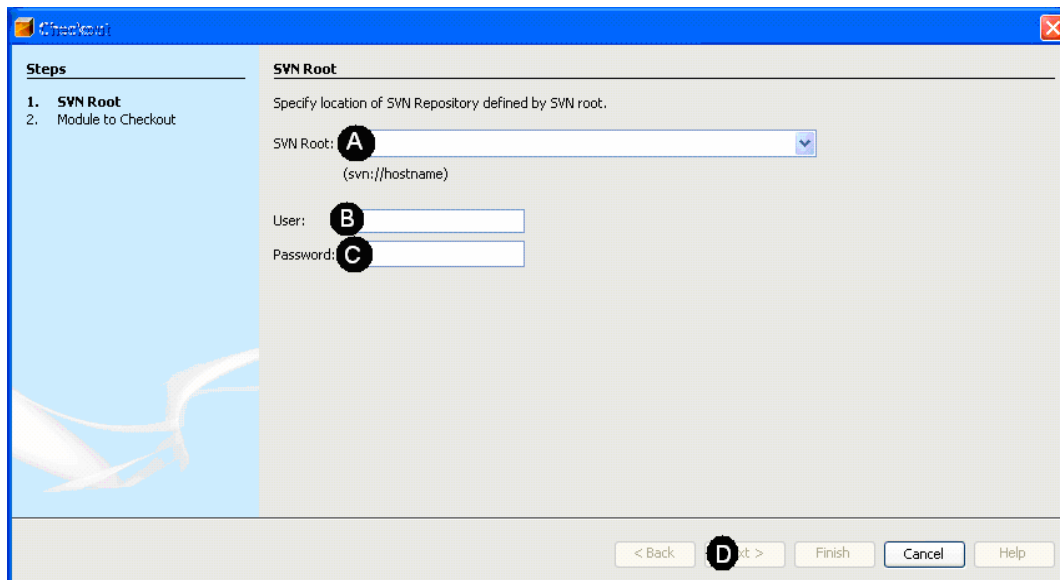


Figura 31 - Janela 1.

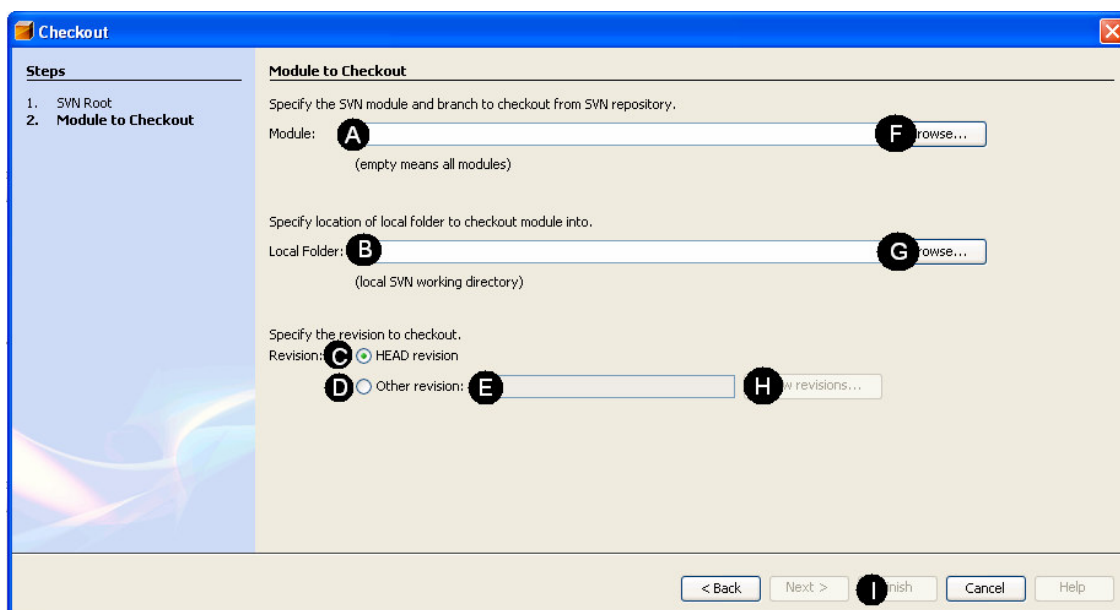


Figura 32 - Janela 2.

Caso de uso: Checkout

Atores: Usuário

Tipo: Primário e real

Descrição: O usuário fornece o repositório SVN desejado, os dados de acesso a esse repositório e os artefatos de software a serem recuperados, ao fim, os artefatos solicitados são recuperados.

Seqüência Típica de Eventos

Ação do Autor	Resposta do Sistema
1. Este caso de uso começa quando um usuário solicita a realização de uma operação de checkout clicando em A da Figura 30.	
2. O usuário registra o repositório SVN desejado em A da Janela 1, o usuário de acesso em B e a senha em C.	
3. O usuário confirma os dados fornecidos clicando em D da Janela 1.	
4. O usuário registra o artefato em A da Janela 2 e a versão de software que deseja receber do repositório em D e E, ou, em C, se ele deseja receber a última versão.	
6. O usuário registra o caminho (PATH) onde serão colocados os artefatos escolhidos em B.	
7. O usuário indica que já registrou todas as informações clicando em I.	8. Realiza a operação solicitada e, ao fim, apresenta uma mensagem de sucesso, demonstrada na Figura 23.

Seqüências alternativas

Linha 4: O usuário clica em F da Janela 2 e escolhe o artefato desejado na janela ilustrada na Figura 17. Ao fim, o *plug-in* apresenta o artefato escolhido em A da Janela 2.

Linha 6: O usuário clica em G da Janela 2 e escolhe, na janela ilustrada na Figura 18, o caminho onde deseja salvar os artefatos solicitados no *checkout*. Após o usuário escolher o caminho desejado o *plug-in* apresenta o caminho escolhido em B da Janela 2.

Linha 8: O usuário pode criar um projeto do Netbeans com os artefatos que foram adquiridos selecionando essa opção na janela ilustrada pela Figura 23. Ao selecionar essa opção o *Wizard* para criação de novos projetos do Netbeans aparecerá.

O próximo passo realizado foi a criação dos diagramas de interação. O UML define dois tipos de diagramas de interação, a saber, os diagramas de colaboração e os diagramas de seqüência. Optou-se por utilizar diagramas de colaboração por sua capacidade de expressão, sua habilidade para expressar mais informações contextuais e da sua relativa economia de espaço. Cada operação do sistema, detalhadas no diagrama de seqüência, resultou em um diagrama de colaboração, como podemos ver nas figuras Figura 33, Figura 34, Figura 35 e Figura 36.

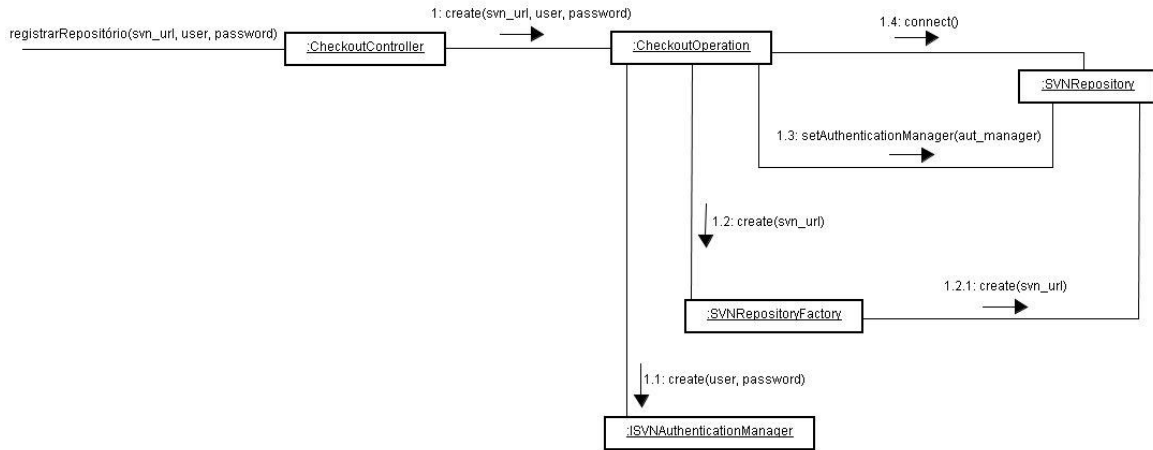


Figura 33 - Diagrama de colaboração de registrarRepositório.

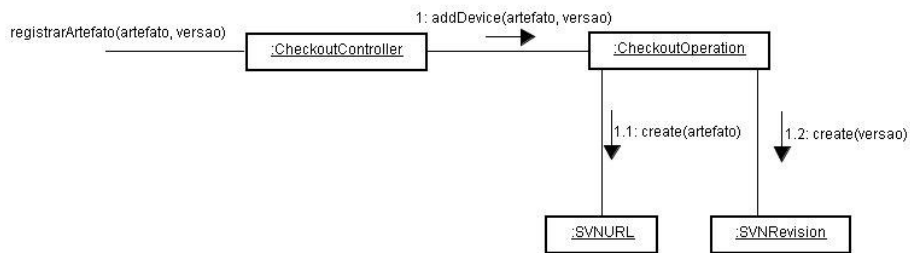


Figura 34- Diagrama de colaboração de registrarArtefato.

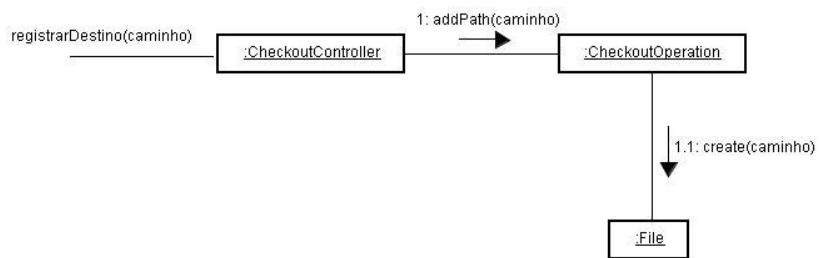


Figura 35 - Diagrama de colaboração de registrarCaminho.

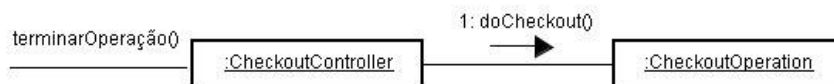


Figura 36 - Diagrama de colaboração de terminarOperação.

Algumas decisões importantes de projeto foram feitas nessa etapa. Nós poderíamos ter optado pelo padrão Fachada (Façade) para construir o controlador das mensagens do sistema tendo em vista que existem poucas operações de sistema. Porém, como a idéia do *plug-in* é que ele se expanda e implemente várias operações SVN essa classe acabaria recebendo muitas responsabilidades e perdendo coesão. Assim, nós optamos por ter controladores de caso de uso, representado, neste caso, pela classe CheckoutController.

Podemos ver na Figura 33 que surgiram as classes SVNRepositoryFactory e ISVNAuthenticationManager. Essas classes pertencem a biblioteca JavaSVN e são necessárias para realizar o acesso aos repositórios Subversion. A classe SVNRepositoryFactory oferece o serviço de criar um SVNRepository a partir de uma SVN URL. Esta URL contém o tipo de acesso que será utilizado pelo usuário, podendo ser, HTTP, HTTPS, SVN ou SVN+SSH, e, assim, a SVNRepositoryFactory pode criar um SVNRepository que atenda ao requisito de acesso requerido pelo usuário. A classe ISVNAuthenticationManager é responsável por realizar a autenticação no repositório.

Assim, com base nos diagramas de colaboração e no modelo conceitual foi criado o diagrama de classes de projeto, apresentado na Figura 37.

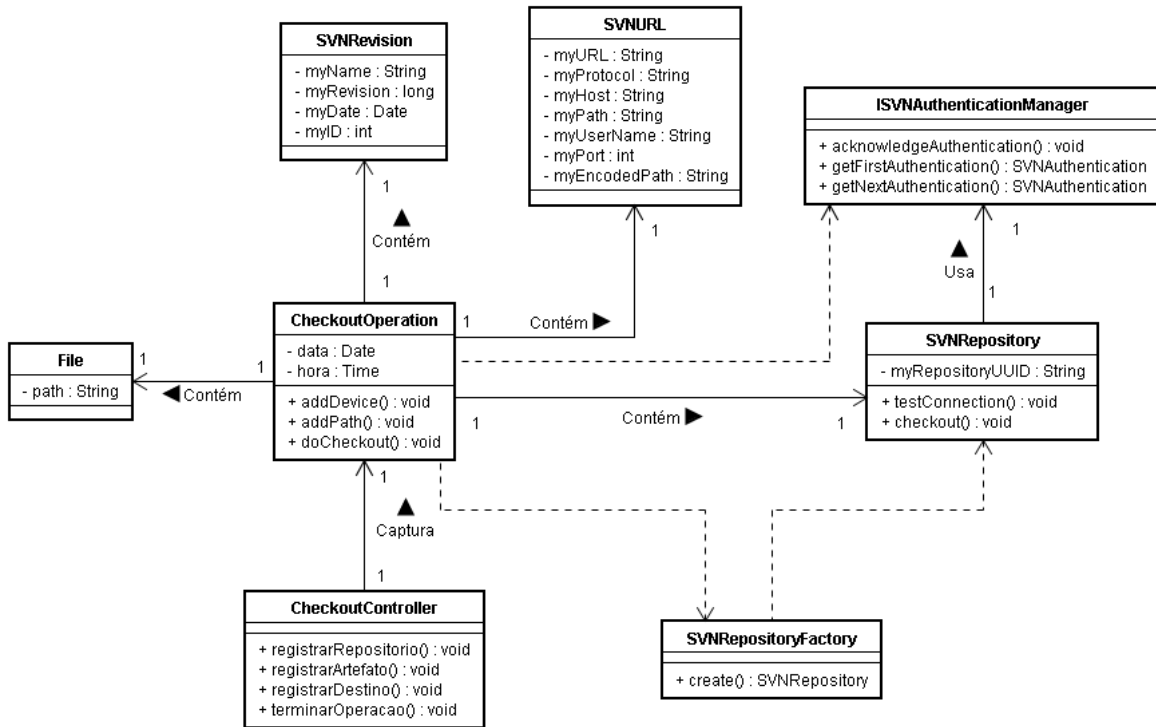


Figura 37 - Diagrama de Classes de Projeto.

A maioria das associações entre os objetos do diagrama de classes de projeto representa visibilidade por atributo, porém três dessas associações diferenciam-se das demais, são elas, o relacionamento entre CheckoutOperation e SVNRepositoryFactory, o relacionamento entre SVNRepositoryFactory e SVNRepository e o relacionamento entre CheckoutOperation e ISVNAuthenticationManager.

Normalmente o criador de um objeto requer uma conexão permanente com o objeto que ele criou, porém, como o objeto responsável por criar um SVNRepository segue o padrão Factory, isso não é necessário. Assim, o relacionamento entre SVNRepositoryFactory e SVNRepository não representa

uma visibilidade por atributo, mas sim declarada localmente, como também o relacionamento entre CheckoutOperation e ISVNAAuthenticationManager.

Com a finalização do caso de uso real, dos diagramas de colaboração e dos diagramas de classes de projeto, havia detalhes suficientes para gerar código para a camada de objetos de domínio. Partimos, então para a terceira fase deste ciclo de desenvolvimento, a fase de implementação.

4.1.2.3 A Fase de Implementação

O objetivo da fase de implementação é mapear artefatos de projeto para código, em uma linguagem orientada a objetos. Para reduzir os riscos e aumentar as chances de desenvolver uma aplicação adequada, o desenvolvimento deveria estar baseado em um volume significativo de trabalho em modelagem de análise e de projeto, antes de começar a codificação. Porém, mesmo assim, a fase de implementação não é apenas um processo de tradução relativamente mecânico, muito pelo contrário, os resultados gerados durante o projeto são o primeiro passo incompleto durante a programação e o teste.

Iniciaremos este tópico descrevendo os passos realizados para criar o projeto do *plug-in* proposto no NetBeans e em seguida passaremos a descrever os passos realizados na implementação do caso de uso *checkout*.

Para auxiliar a criação de *plug-ins* o NetBeans oferece, no *wizard* de criação de novos projetos (que pode ser visto na Figura 38), a categoria *NetBeans Plug-in Modules* que fornece a opção de criação de três tipos de projetos, descritos a abaixo:

Module Project – Cria um projeto para o desenvolvimento de um *plug-in* autônomo.

Library Wrapper Module Project – Cria um *wrapper* para uma biblioteca usada por algum *plug-in*.

Module Suite Project – Cria um projeto contendo um conjunto interdependente de *plug-ins* e *wrappers* de bibliotecas que pode ser usado para *deployment*.

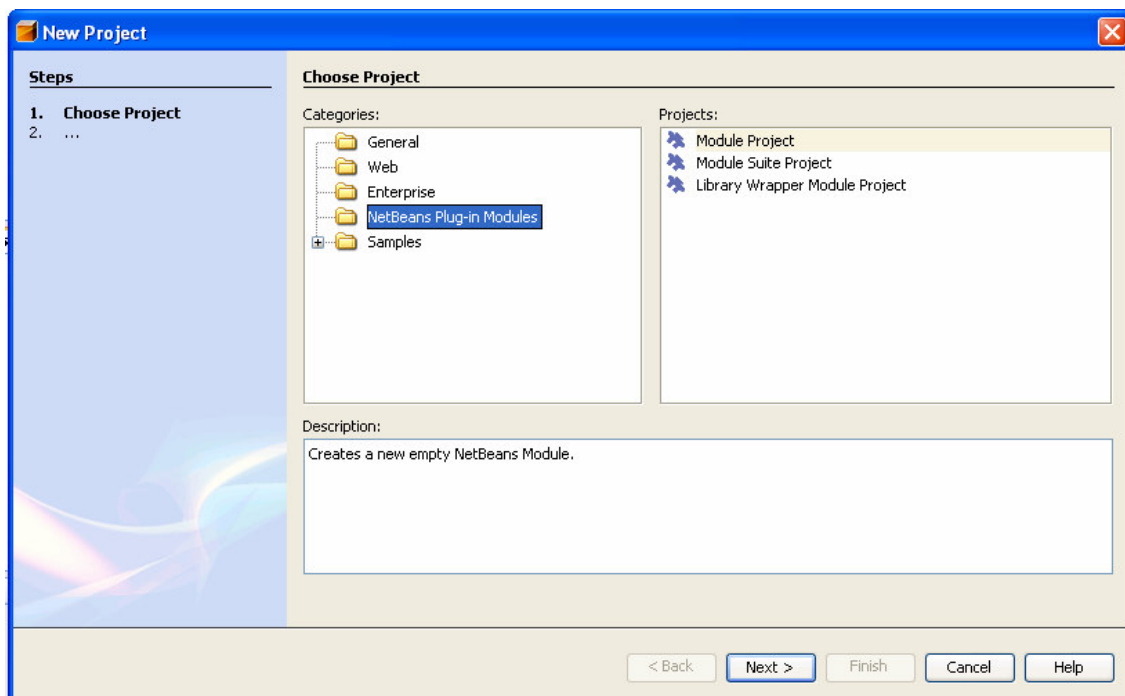


Figura 38 - Wizard de criação de projetos do NetBeans.

Como o *plug-in* proposto faz uso da biblioteca JAVA SVN, fez-se necessário primeiramente a criação de uma suíte que conteria o *wrapper* da biblioteca e o *plug-in*. A criação da suíte é bem simples, bastando indicar, no *wizard* de criação,

o nome da suíte, o diretório onde ela será salva e a plataforma NetBeans em que ela irá funcionar.

Após a criação da suíte foi criado um *wrapper* para a biblioteca JAVA SVN. A criação de *wrappers* também é bem simples, bastando indicar, no *wizard* de criação, o nome do *wrapper*, a localização do arquivo .jar da biblioteca, o diretório onde ele será salvo e a suíte a que ele será adicionado.

Por fim, foi criado o projeto do *plug-in*. No *wizard* de criação de módulos autônomos foi indicado o nome do módulo, o diretório onde ele será salvo, a suíte a que ele pertence e os parâmetros *Localizing Bundle*, que indica um arquivo utilizado para internacionalização, e *XML Layer*, que indica um arquivo XML onde são registrados itens como menus, barras de tarefas e botões no sistema de arquivos do NetBeans.

Podemos conferir o menu de projetos do NetBeans na Figura 39 onde primeiramente é apresentado o *wrapper*, que recebeu o nome de “javasvn”, logo após podemos conferir o projeto do *plug-in*, que recebeu o nome de “SubversionModule”, e, por último, a suíte denominada “SubversionSuite”.

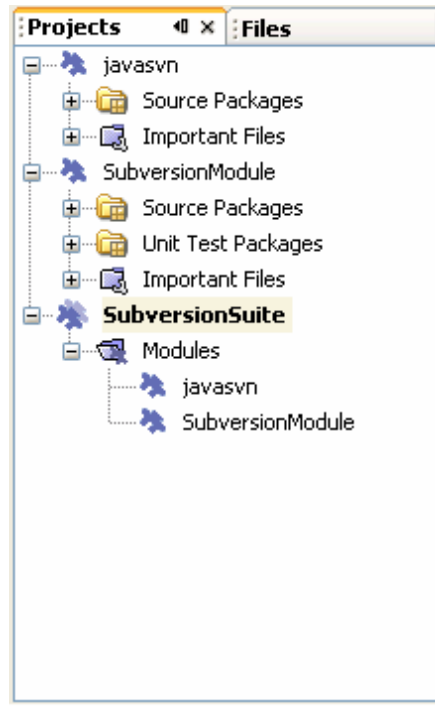


Figura 39 - Menu de projetos no NetBeans.

Após a criação dos projetos partimos para a implementação da interface gráfica. O primeiro passo foi a adição do menu SVN na barra de menus do NetBeans. Para a criação de menus e itens de menus o NetBeans oferece a categoria *NetBeans Module Development* no *wizard* de criação de novos arquivos. Nesta categoria escolhemos o tipo de arquivo *Action* para criar itens de menu, como mostra a Figura 40.

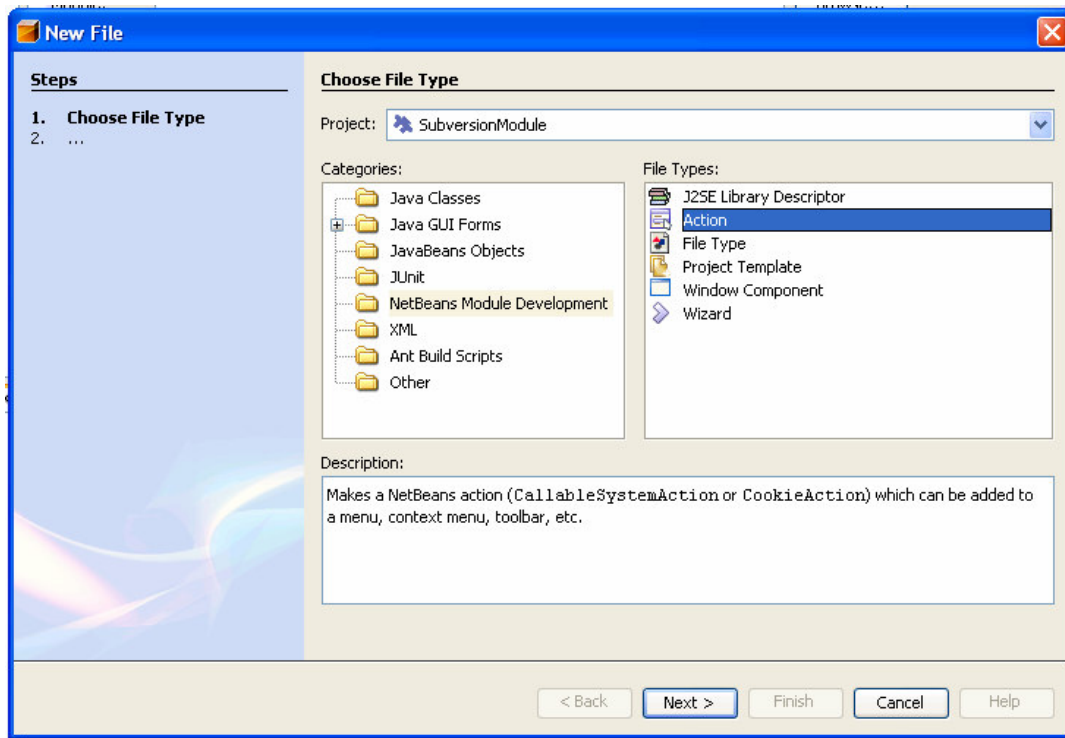


Figura 40 - Wizard de criação de novos arquivos.

Este *wizard* edita o arquivo XML Layer e cria uma classe que estende `CallableSystemAction`, assim, quando o usuário clica no item de menu criado, o método `performAction` desta classe é chamado. Podemos conferir algumas linhas do arquivo XML Layer criado na Figura 41.

```

</folder>
<folder name="Menu">
  <folder name="SVN">
    <file name="org-svn-subversionmodule-actions-CommitAllAction.shadow">
      <attr name="originalFile" stringvalue="Actions/SVN/org-svn-subversionmodule-actions-CommitAllAction.instance"/>
    </file>
    <file name="org-svn-subversionmodule-actions-SampleAction.shadow">
      <attr name="originalFile" stringvalue="Actions/SVN/org-svn-subversionmodule-actions-SampleAction.instance"/>
    </file>
    <attr name="org-svn-subversionmodule-actions-SampleAction.shadow/org-svn-subversionmodule-actions-SampleAction.shadow" boolvalue="true"/>
    <file name="org-svn-subversionmodule-actions-UpdateAllAction.shadow">
      <attr name="originalFile" stringvalue="Actions/SVN/org-svn-subversionmodule-actions-UpdateAllAction.instance"/>
    </file>
    <file name="org-svn-subversionmodule-actions-separatorAfter.instance">
      <attr name="instanceClass" stringvalue="javax.swing.JSeparator"/>
    </file>
    <attr name="org-svn-subversionmodule-actions-CommitAllAction.shadow/org-svn-subversionmodule-actions-separatorAfter.instance" boolvalue="f

```

Figura 41 - Um trecho do arquivo XML Layer criado.

Como pode ser conferido, a *tag folder* chamada Menu indica a barra de menus do NetBeans, abaixo dela, foi criada a *tag* SVN, que indica um menu apresentado na barra de menus e, logo abaixo dela, foram inseridos os itens de menus desejados. Neste arquivo foram inseridos, também, os separadores de menus e as classes responsáveis por responder as ações dos usuários nos menus. Com esses passos realizados, o ambiente de desenvolvimento do *plug-in* estava finalizado e foi possível partir para a implementação do caso de uso *checkout*.

A primeira tarefa da implementação do caso de uso *checkout* foi a criação do *wizard* de *checkout*. Para a criação de *wizards* o NetBeans oferece o tipo de arquivo *wizard* na janela de criação de novos arquivos. A criação de *wizards* torna-se muito simples, bastando o usuário escolher o nome do *wizard* e o número de painéis que ele conterá. Ao final, o NetBeans cria um arquivo *.java* e outro *.form* para cada painel do *wizard* criado e um arquivo *sampleAction.java* que contém um exemplo de como utilizar o *wizard* criado.

O NetBeans oferece dois tipos de validação para os painéis, uma síncrona e outra assíncrona. Por padrão o NetBeans cria os painéis do *wizard* implementado a classe *WizardDescriptor.ValidatingPanel*, que fornecem validação síncrona, assim, quando o usuário clica no botão *Next* ou *Finish* do *wizard* o método *validate()* do painel corrente é chamado.

Porém, os dois painéis utilizados no *wizard checkout* necessitavam de validação assíncrona, o primeiro pois precisava esperar a realização da conexão com o repositório e o segundo pois precisava esperar a realização da operação,

assim tivemos fazer com que os painéis implementassem a classe `WizardDescriptor.AsynchronousValidatingPanel`, utilizada para validações assíncronas. Esta interface contém o método `prepareValidation()` que é chamado de maneira síncrona quando o botão *Next* ou *Finish* do *wizard* é clicado e o método `validate()`, que é chamado em uma *thread* separada quando algum desses botões são clicados. Durante a validação o botão *Cancel* é utilizado para enviar um sinal de `interrupt()` para a *thread* de validação e os botões *Next* e *Finish* são desabilitados, em caso de sucesso o *wizard* passa para o próximo painel ou, se for o caso, finaliza.

Após a criação do *wizard* de *checkout* foram implementadas as classes do diagrama de classes, que podem ser visualizadas no Anexo 2, finalizando assim a fase de implementação do caso de uso *checkout*.

4.1.3 Segundo Ciclo de Desenvolvimento, o Caso de Uso *Commit*

Serão descritas, a seguir, as atividades desenvolvidas no segundo ciclo de desenvolvimento realizado, que atacou o caso de uso *commit*. Descreveremos essas atividades com um grau menor de detalhes dos artefatos UML gerados, que seguem o padrão realizado no caso de uso *checkout*, porém, todos os artefatos UML gerados podem ser encontrados no Anexo 1.

4.1.3.1 A Fase de Análise

O *plug-in* CVS do NetBeans oferece a possibilidade da realização da operação de *commit* por projeto ou um *commit* abrangente que varre todos os

projetos. Nós decidimos implementar, no escopo deste trabalho, a segunda alternativa, ou seja, um *commit* que varra os projetos em busca de artefatos que contêm alterações a serem enviadas ao repositório. Para tal propósito expandimos o caso de uso ideal descrito no tópico 4.1 e geramos o caso de uso expandido apresentado abaixo.

Caso de uso: Commit

Atores: Usuário

Tipo: Primário e essencial

Descrição: O usuário escolhe os artefatos de software que deseja enviar ao repositório SVN, ao fim, a operação de commit é realizada.

Seqüência Típica de Eventos

Ação do Autor	Resposta do Sistema
1. Este caso de uso começa quando um usuário solicita a realização de uma operação de Commit.	
2. O usuário registra uma mensagem descrevendo a operação de <i>commit</i> .	
	3. Mostra os artefatos de software disponíveis para commit.
4. O usuário seleciona os artefatos que deseja atualizar no repositório.	
5. O usuário indica que selecionou todos os artefatos desejados.	6. Realiza a operação.

Para este caso de uso foram identificadas as operações do sistema: *adicionarMensagem*, *adicionarArtefato* e *terminarOperação*. Não foi observada a necessidade de adição de nenhum conceito novo ao diagrama conceitual, que, assim, permaneceu inalterado.

4.1.3.2 A Fase de Projeto

O primeiro passo da fase de projeto foi a criação do frame de *commit*, que é chamado a partir do item de menu *commit*, como descrito no tópico 4.1. Esse frame tem o diferencial, em relação ao frame de *commit* do *plug-in* CVS, de oferecer a possibilidade do usuário selecionar os artefatos que deseja enviar ao repositório, como foi contemplado no caso de uso essencial desenvolvido na análise.

A partir da criação desta interface passamos para a elaboração do caso de uso real, que pode ser contemplado a seguir:

Caso de uso: Commit

Atores: Usuário

Tipo: Primário e real

Descrição: O usuário escolhe os artefatos de software que deseja enviar ao repositório SVN, ao fim, a operação de commit é realizada.

Seqüência Típica de Eventos

Ação do Autor

Resposta do Sistema

1. Este caso de uso começa quando um usuário solicita a realização de uma operação de commit clicando em A da Figura 42.
2. O usuário registra uma mensagem descrevendo a operação de *commit* em B da Figura 42.
3. Mostra os artefatos de software disponíveis para commit em D da Figura 42.
4. O usuário seleciona os artefatos que deseja atualizar no repositório em C da Figura 42.
5. O usuário indica que selecionou todos os artefatos desejados clicando em E da Figura 42.
6. Realiza a operação.

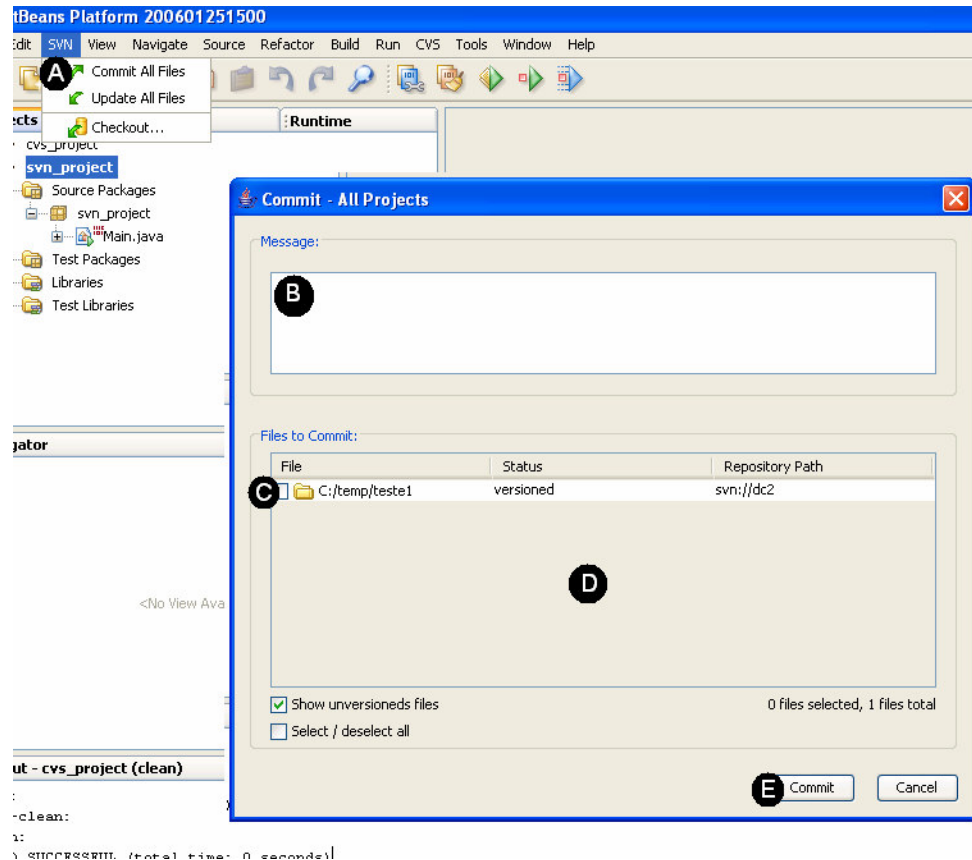


Figura 42 - Janela da operação de *commit*.

Após a elaboração do caso de uso real acima foram criados os diagramas de colaboração. O principal diagrama de colaboração deste ciclo de desenvolvimento foi o diagrama de colaboração de adicionarArtefato, que pode ser visto na Figura 43. Nesse diagrama surgiu a classe ProjectCatalog, que é responsável por solicitar ao NetBeans os projetos abertos e verificar se esses projetos são controlados pelo SVN.

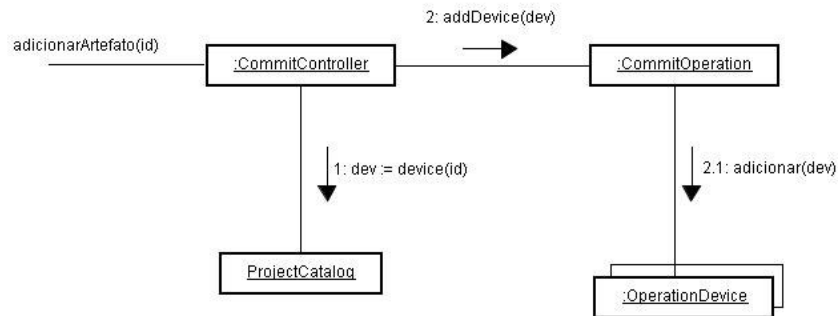


Figura 43 - Diagrama de colaboração de adicionarArtefato.

4.1.3.3 A Fase de Implementação

Na fase de implementação foi criada, primeiramente, a janela da operação de *commit*. Esta janela possui uma tabela que apresenta os artefatos que estão disponíveis para *commit*, oferecendo a possibilidade de o usuário selecionar os artefatos que deseja através de um JCheckBox. Além disso, a tabela disponibiliza ao usuário o ícone, o *status* e o repositório do artefato, o que faz com que ela não possa ser construída com um simples JTable. Para a construção desta tabela tivemos de criar um AbstractTableModel para definir o renderizador e o editor de cada célula, um TableCellRenderer, para renderizar a coluna File de modo que ela

apresente um JCheckBox e um JLabel, e um AbstractCellEditor para possibilitar a edição da coluna File. Além disso, tínhamos de implementar a comunicação entre o AbstractTableModel e a janela, e o fizemos através do padrão Publish/Subscriber, onde a janela faz o papel de Subscriber, que recebe os eventos da tabela.

Depois de desenvolvida a janela da operação de *commit* partimos para a criação da classe ProjectCatalog. Essa classe faz uso da classe OpenProjects, encontrada no pacote “org.netbeans.api.project.ui” da API do NetBeans, através do método getOpenProjects, para solicitar os projetos abertos. Uma vez conhecendo os projetos abertos a classe usa SVNDirectory, encontrada no pacote org.tmatesoft.svn.core.internal.wc da biblioteca JavaSVN, para encontrar os arquivos, de cada projeto, disponíveis para *commit*.

Por fim, implementamos as classes CommitController, que faz a interface entre a camada de apresentação e a camada de domínio, e a classe CommitOperation, que realiza a operação de *commit*.

4.1.4 Terceiro Ciclo de Desenvolvimento, o Caso de Uso *Update*

Este ciclo de desenvolvimento foi o ciclo mais curto do projeto pois o caso de uso *update* realiza pouca interação com o usuário, diminuindo, assim, o número de interfaces a serem desenvolvidas.

Descreveremos esse ciclo de desenvolvimento com o mesmo nível de detalhes utilizados na descrição do segundo ciclo de desenvolvimento.

4.1.4.1 A Fase de Análise

Implementamos a operação de *update* da mesma maneira que o *plug-in* CVS do NetBeans realiza a operação *Update All Files*, ou seja, o usuário solicita a operação de *update* e o *plug-in* realiza a operação para todos os arquivos de todos os projetos abertos na IDE. Com esse objetivo expandimos o caso de uso ideal *update* para o caso de uso essencial detalhado a seguir.

Caso de uso: Update

Atores: Usuário

Tipo: Primário e essencial

Descrição: O usuário solicita a realização da operação de *update*, ao final, a operação de *update* é realizada.

Seqüência Típica de Eventos

Ação do Autor	Resposta do Sistema
1. Este caso de uso começa quando um usuário solicita a realização de uma operação de <i>Update</i> .	2. Realiza a operação de <i>Update</i> .

Foi identificado, pra este caso de uso, apenas a operação realizarOperação.

4.1.4.2 A Fase de Projeto

Para este caso de uso não foi desenvolvida nenhuma janela. Ao clicar no item de menu “Update All Files” do menu SVN, detalhado na Figura 9 no início do

capítulo 4, o *plug-in* realiza a operação de *update*. Podemos conferir o diagrama de colaboração de realizarOperação na Figura 44.

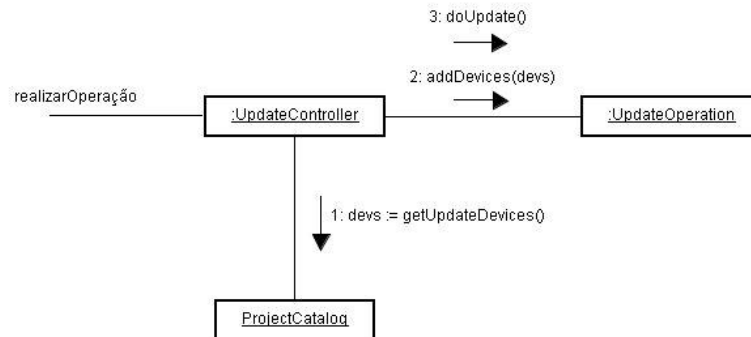


Figura 44 - Diagrama de colaboração de realizarOperação.

Como podemos ver no diagrama de colaboração acima o UpdateController solicita ao ProjectCatalog os arquivos para *update* e os envia para o UpdateOperation, que realiza a operação.

4.1.4.3 A Fase de Implementação

A fase de implementação do terceiro ciclo de desenvolvimento constituiu-se da implementação das classes UpdateController, que faz a interface entre a camada de apresentação e a camada de domínio, a implementação da classe UpdateOperation, que é responsável pela realização da operação de *update* e a implementação do método getUpdateDevices na classe ProjectCatalog, que obtém todos os arquivos, de todos os projetos, disponíveis para *update* utilizando as mesmas classes descritas na fase de implementação do segundo ciclo de desenvolvimento.

5 Conclusões

A implementação de um plug-in para um IDE é uma tarefa complexa e que exige constante interação com as APIs do ambiente e revisão da documentação. Devido a esta complexidade, o ambiente NetBeans fornece alguns *Wizards* com o intuito de acelerar o processo de desenvolvimento.

Desenvolver um artefato de software que forneça as funções de um sistema de controle de versão é uma tarefa ampla e que demanda uma boa quantidade de tempo e trabalho, a utilização da biblioteca JavaSVN diminuiu consideravelmente essas necessidades.

O *plug-in* desenvolvido foi colocado para testes no ambiente de trabalho corporativo da empresa Suntech Telecom Solutions e foi homologado satisfazendo os requisitos de desempenho e usabilidade. Logo, ele será disponibilizado no site sourceforge.net, que é um repositório de projetos *open-source*.

Acreditamos que o objetivo do trabalho foi alcançado pois desenvolvemos uma ferramenta que implementa as operações delimitadas no escopo do trabalho, que é semelhante, do ponto de vista de utilização, e compatível com o *plug-in* CVS do NetBeans, que disponibiliza acesso aos repositórios através dos protocolos SVN, SVN sobre SSH e HTTP, que é de fácil utilização e que traz algumas melhorias, como, por exemplo, a apresentação, na tela de escolha do projeto a ser feito *checkout*, do usuário e versão referente a cada projeto, que é uma informação normalmente necessária ao usuário, como comentada no capítulo 4, e a possibilidade da escolha da versão do artefato nesta mesma tela.

5.1 Sugestões para Trabalhos Futuros

Para trabalhos futuros sugerimos a implementação das demais operações existentes nos sistemas de controle de versão, como a criação de repositórios e as operações de *diff*, *report* e *get*.

6 Referências Bibliográficas

[BOO 96] BOOCH, G. **Object Solutions**: Managing the Object-Oriented Project. Menlo Park: Addison-Wesley, 1996.

[BRO 92a] BROWN, A.; EARL, A.; MCDERMID, J. **Software Engineering Environments**: Automated Support for Software Engineering. London: McGraw-Hill, 1992.

[BRO 92b] BROWN, A.; MCDERMID, J. **Learning from IPSE's mistakes**. v. 9 Los Alamitos: IEEE Software, 1992.

[BRO 92c] BROWN A.; FEILER P.; WALLNAU K. Past and future models of CASE integration, Proc. of the 5th International Workshop on Computer-Aided Software Engineering, IEEE, Julio 1992.

[CHR 95] CHRISTIE, A.. **Software Process Automation**: The Technology and its adoption. Berlin: Springer Verlag, 1995.

[CON 94] CONRADI, R., et al. **EPOS**: Object-Oriented Cooperative Process Modeling. In: FINKELSTEIN, A. et al. (Ed.). Software Process Modeling and Technology. Taunton: Research Studies Press, 1994. p. 33-70.

[CON 97] CONSTANTINE, L. 1997. The Case for Essential Use Cases. Object Magazine May 1997. NY, NY: SIGS Publications.

[CVS 2006] CVS Website, <http://www.cvshome.org/>, 2006.

[DAR 91] DART S. "Concepts in Configuration Management Systems," presented at the 3rd International workshop on software configuration management, Trondheim, Norway, 1991.

[DOW 91] DOWSON, M.; NEJMEH, B.; RIDDLE, W.. Fundamental Software Process Concepts. In: EUROPEAN WORKSHOP ON SOFTWARE PROCESS MODELLING, 1., 1991, Milan, Italy. Proceedings... [S.l.]: AICA Press, Maio 1991.

[FEI 93] FEILER, P.; HUMPHREY, W.. Software Process Development and Enactment: Concepts and Definitions. In: INTERNATIONAL CONFERENCE ON THE SOFTWARE PROCESS, 2., 1993, Berlin. Proceedings... Berlin: IEEE Computer Society Press, Março 1993.

[GRE 2002] GREENE S. **NetBeans: The Definitive Guide**. Sebastopol: O'Reilly & Associates, 2002.

[HUM 89b] HUMPHREY, S. **Managing the Software Process**. New York: Addison-Wesley, 1989.

[JAC 92] JACOBSON, I., et al. **Object-Oriented Software Engineering: A Use Case Driven Approach**. Reading: Addison-Wesley, 1992.

[JEN 2005] JENSEN C.; SCACC W. **Collaboration, Leadership, Control, and Conflict Negotiation and the Netbeans.org Open Source Software Development Community**. Irvine: University of California, 2005.

[KAP 2005] KAPPEL G., et al. ModelCVS A Semantic Infrastructure for Model-based Tool Integration. Business Informatics Group, Vienna University of Technology, 2005.

[LARS 95] BENDIX, L. Configuration Management and Version Control Revisited. Institute for Electronic Systems, Aalborg University Denmark, Dec. 1995.

[LAR 2000] LARMAN, C. **Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos**, trad. Luiz A. Meirellies Salgado. Porto Alegre: Bookman, 2000.

[LIM 98] LIMA, C. **Um Gerenciador de Processos de Software para o Ambiente PROSOFT**. 1998. 197 f.. Dissertação (Mestrado em Ciências da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 1998.

[LON 93] LONCHAMP, J. A Structured Conceptual and Terminological Framework for Software Process Engineering. In: INTERNATIONAL CONFERENCE ON THE SOFTWARE PROCESS, 2., 1993, Berlin. Proceedings... Berlin: IEEE Computer Society Press, Março 1993.

[NET 2006] NetBeans website, <http://www.netbeans.org/>, 2006.

[OST 87] OSTERWEIL, L. Software Processes are Software Too. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 9., 1987, Monterey, California. Proceedings... Monterey: IEEE Computer Society Press, 1987.

[PER 2006] Perforce website, <http://www.perforce.com/>, 2006

[PRE 95] PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Makron Books, 1995.

[RAT 2006] Rational ClearCase website, <http://www.rational.com/products/clearcase/index.jsp>, 2006.

[SVN 2006] SVN Website, <http://subversion.tigris.org/>, 2006.

[TIC 85] TICHY W. F. **RCS - A System for Version Control: Software - Practice and Experience**, vol. 15, no. 7, pp. 637-654, 1985.

[THO 92] THOMAS, L.; NEJMEH, A. **Definitions of Tool Integration for Environments**. Los Alamitos: IEEE Software, Março 1992, v. 9, n. 2., p. 29-35.

[WOR 2005] WORTH D. J.; GREENOUGH C. **Comparison of CVS and Subversion**. Didcot: Oxfordshire, 2005.

[WU 2003] WU, X. Visualization of Version Control Information, Wuhan University of Hydraulic and Electric Engineering, 2003.

Anexo 1 – Artigo

Integração do Sistema de Controle de Versão Subversion (SVN) com o IDE NetBeans

Lucio Moratelli Prado

Sistemas de Informação, Departamento de Informática e estatística – INE

Universidade Federal de Santa Catarina (UFSC), Brasil

lucio@inf.ufsc.br

Resumo

Este artigo apresenta uma breve explicação do desenvolvimento de um *plug-in* que implementa a integração entre um dos ambientes integrados de desenvolvimento de software mais utilizados para desenvolvimento de software JAVA, o NetBeans, com uma ferramenta de controle de versão que vem ganhando espaço na comunidade *Open Source*, o Subversion.

Palavras-Chaves: NetBeans, Subversion, ambiente integrado de desenvolvimento de software, gerencia de configuração, controle de versão, *plug-in*.

Introdução

A crescente busca da qualidade de software tem movido a área de Engenharia de Software nos últimos anos. Como um dos maiores obstáculos desta busca podemos

citar a grande complexidade das aplicações atuais, que tornou necessária a automatização do processo de desenvolvimento para garantir o controle e a gerência efetivos do processo de software [LIM 98].

Em resposta a essa necessidade surgiram ferramentas individuais de apoio à programação, como compiladores, montadores e depuradores e, em seguida, ferramentas de apoio a outras fases do desenvolvimento de software, como as ferramentas de controle de versão.

Porém o uso de várias ferramentas acabou por acrescentar mais complexidade ao processo de desenvolvimento de software. Assim, surgiram os ambientes integrados de desenvolvimento de software (IDE), que se propõe a integrar as diversas

ferramentas utilizadas no processo de desenvolvimento de software.

Assim, propomos a integração da ferramenta de controle de versão Subversion com o IDE Netbeans.

Motivação

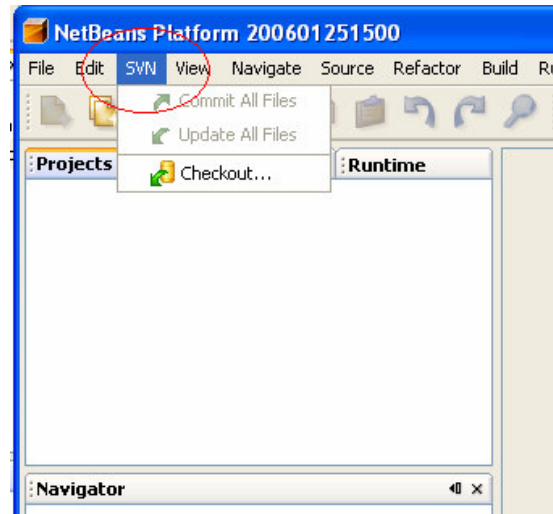
A NetBeans.org é atualmente uma das maiores comunidades de desenvolvimento de software de código fonte aberto do mundo. O uso da ferramenta de controle de versão Subversion vem crescendo e ela tende a tornar-se o padrão *de facto* para este fim na comunidade *Open Source*. O fato dessas duas ferramentas tão utilizadas não possuírem integração nos motivou a promovê-la.

O *Plug-in* Proposto

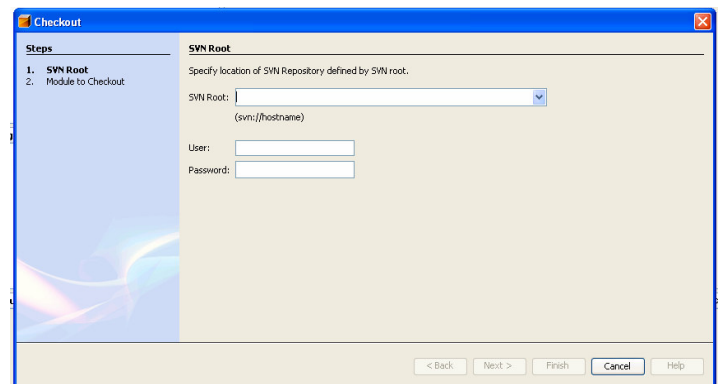
O *plug-in* SVN proposto pretende se parecer ao máximo, do

ponto de vista de utilização, com o *plug-in* CVS padrão do NetBeans para tornar mais fácil a migração dos usuários. Em alguns casos manter essa semelhança não será possível devido as diferenças de arquitetura entre os dois sistemas, e em outros casos, algumas mudanças serão realizadas propositalmente com o intuito de melhorar características do *plug-in* CVS.

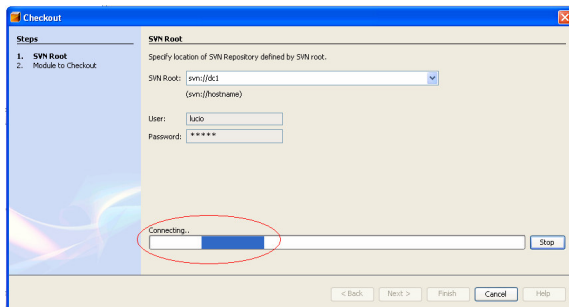
Podemos ter acesso as operações do Subversion através do menu SVN, localizado na barra de menus do NetBeans. Neste menu encontramos três itens de menu, referentes as opções de *Commit*, *Update* e *Checkout*. As operações de *Commit* e *Update* só são habilitadas quando existe um projeto com controle de versão através do Subversion.



Para realizarmos uma operação de *Checkout* basta clicarmos no item de menu *Checkout*. Ao clicarmos neste ícone o *wizard* de checkout se abrirá. Na primeira etapa deste *wizard* inserimos o repositório no qual encontram-se os artefatos que desejamos, o usuário e a senha para *login* neste repositório.



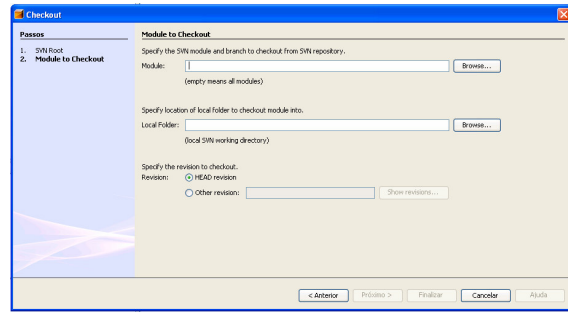
Após inserirmos essas informações podemos prosseguir com a operação clicando no botão *Next*. Neste momento os dados inseridos são validados e é realizada a conexão com o repositório fornecido. Podemos acompanhar o processo de conexão através de uma barra de progresso, e temos, também, a possibilidade de para esse processo a qualquer momento clicando no botão *Stop*.



Caso a conexão ocorra com sucesso a segunda etapa da operação de *checkout* nos é apresentada.

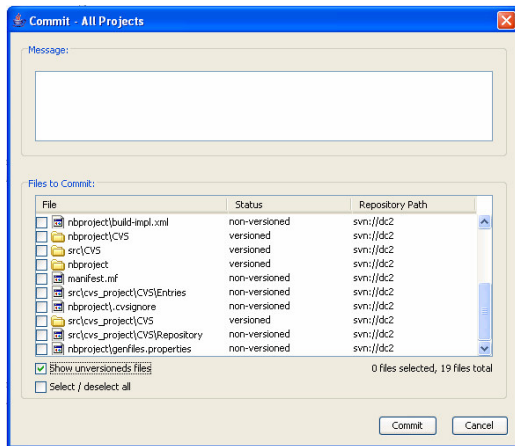
Na segunda etapa da operação de *checkout* indicamos o artefato que desejamos receber do

repositório, o diretório da máquina local onde esse artefato será salvo e a versão do artefato que desejamos adquirir.



Após a inserção dessas informações clicamos em “*Finish*” para terminar a operação.

Para a realização da operação de *commit* clicamos no item de menu *Commit* do menu SVN. O *plug-in* irá apresentar uma janela com os artefatos que possuem controle de versão via Subversion. Escolhemos, então, os artefatos que desejamos em atualizar no repositório e clicamos no botão *Commit*.



Para a realização da operação de *update* clicamos no item de menu *Update* do menu SVN. O *plug-in* varrerá todos os projetos do NetBeans e realizará a operação de *update* para todos os artefatos que possuem controle de versão via Subversion e que possuem alterações a serem recebidas do repositório.

Conclusão

Através do *plug-in* proposto a integração do IDE NetBeans com o sistema de controle de versão Subversion foi realizada. Assim, acreditamos que a integração

proposta atingiu seu objetivo, disponibilizando aos usuários do NetBeans a possibilidade de desfrutar dos benefícios do sistema de controle de verões Subversion com a facilidade e a comodidade que a integração dessas ferramentas proporciona.

Referências

[CHR 95] CHRISTIE, A.. **Software Process Automation: The Technology and its adoption.** Berlin: Springer Verlag, 1995.

[CON 94] CONRADI, R., et al. **EPOS: Object-Oriented Cooperative Process Modeling.** In: FINKELSTEIN, A. et al. (Ed.). *Software Process Modeling and Technology.* Taunton: Research Studies Press, 1994. p. 33-70.

[CON 97] CONSTANTINE, L. 1997. *The Case for Essential Use Cases.* Object Magazine May 1997. NY, NY: SIGS Publications.

[DOW 91] DOWSON, M.; NEJMEH, B.; RIDDLE, W.. *Fundamental Software Process Concepts.* In: EUROPEAN WORKSHOP ON SOFTWARE PROCESS MODELLING, 1., 1991, Milan, Italy.

Proceedings... [S.I.]: AICA Press,
Maio 1991.

[FEI 93] FEILER, P.; HUMPHREY,
W.. Software Process Development
and Enactment: Concepts and
Definitions. In: INTERNATIONAL
CONFERENCE ON THE

SOFTWARE PROCESS, 2., 1993,
Berlin. Proceedings... Berlin: IEEE
Computer Society Press, Março 1993.

[HUM 89b] HUMPHREY, S.
Managing the Software Process.
New York: Addison-Wesley, 1989.

Anexo 2 – Artefatos UML

Casos de uso ideais:

Caso de uso: Checkout

Atores: Usuário

Tipo: Primário

Descrição: O usuário fornece o repositório SVN desejado, os dados de acesso a esse repositório e os artefatos de software a serem recuperados, ao fim, os artefatos solicitados são recuperados.

Caso de uso: Commit

Atores: Usuário

Tipo: Primário

Descrição: O usuário escolhe os artefatos de software que deseja enviar ao repositório SVN, ao fim, a operação de commit é realizada.

Caso de uso: Update

Atores: Usuário

Tipo: Primário

Descrição: O usuário solicita a realização da operação de update, ao final, a operação de update é realizada.

Casos de uso essenciais:

Caso de uso: Checkout

Atores: Usuário

Tipo: Primário e essencial

Descrição: O usuário fornece o repositório SVN desejado, os dados de acesso a esse repositório e os artefatos de software a serem recuperados, ao fim, os artefatos solicitados são recuperados.

Seqüência Típica de Eventos

Ação do Autor	Resposta do Sistema
1. Este caso de uso começa quando um usuário solicita a realização de uma operação de checkout.	
2. O usuário registra o repositório SVN desejado e os dados de acesso a esse repositório.	3. Mostra os artefatos de software disponíveis neste repositório e as informações de versão destes artefatos.
4. O usuário registra o artefato e a versão de software que deseja receber do repositório.	5. Acrescenta a informação do artefato à operação corrente e apresenta ao usuário o artefato e a versão escolhida.
6. O usuário registra o caminho (PATH) onde serão colocados os artefatos escolhidos.	
7. O usuário indica que já registrou todas as informações.	8. Realiza a operação solicitada e, ao fim, apresenta uma mensagem de sucesso.

Caso de uso: Commit

Atores: Usuário

Tipo: Primário e essencial

Descrição: O usuário escolhe os artefatos de software que deseja enviar ao repositório SVN, ao fim, a operação de commit é realizada.

Seqüência Típica de Eventos

Ação do Autor	Resposta do Sistema
1. Este caso de uso começa quando um usuário solicita a realização de uma operação de Commit.	
2. O usuário registra uma mensagem descrevendo a operação de <i>commit</i> .	
	3. Mostra os artefatos de software disponíveis para <i>commit</i> .
4. O usuário seleciona os artefatos que deseja atualizar no repositório.	
5. O usuário indica que selecionou todos os artefatos desejados.	6. Realiza a operação.

Caso de uso: Update

Atores: Usuário

Tipo: Primário e essencial

Descrição: O usuário solicita a realização da operação de *update*, ao final, a operação de *update* é realizada.

Seqüência Típica de Eventos

Ação do Autor	Resposta do Sistema
1. Este caso de uso começa quando um usuário solicita a realização de uma operação de <i>Update</i> .	
	2. Realiza a operação de <i>Update</i> .

Casos de uso reais:

Caso de uso: Checkout

Atores: Usuário

Tipo: Primário e real

Descrição: O usuário fornece o repositório SVN desejado, os dados de acesso a esse repositório e os artefatos de software a serem recuperados, ao fim, os artefatos solicitados são recuperados.

Seqüência Típica de Eventos

Ação do Autor	Resposta do Sistema
1. Este caso de uso começa quando um usuário solicita a realização de uma operação de checkout clicando em A da Figura 30.	
2. O usuário registra o repositório SVN desejado em A da Janela 1, o usuário de acesso em B e a senha em C.	
3. O usuário confirma os dados fornecidos clicando em D da Janela 1.	
4. O usuário registra o artefato em A da Janela 2 e a versão de software que deseja receber do repositório em D e E, ou, em C, se ele deseja receber a última versão.	
6. O usuário registra o caminho (PATH) onde serão colocados os artefatos escolhidos em B.	
7. O usuário indica que já registrou todas as informações clicando em I.	
	8. Realiza a operação solicitada e, ao fim, apresenta uma mensagem de sucesso, demonstrada na Figura 23.

Seqüências alternativas

Linha 4: O usuário clica em F da Janela 2 e escolhe o artefato desejado na janela ilustrada na Figura 17. Ao fim, o *plug-in* apresenta o artefato escolhido em A da Janela 2.

Linha 6: O usuário clica em G da Janela 2 e escolhe, na janela ilustrada na Figura 18, o caminho onde deseja salvar os artefatos solicitados no *checkout*. Após o usuário escolher p caminho desejado o *plug-in* apresenta o caminho escolhido em B da Janela 2.

Linha 8: O usuário pode criar um projeto do Netbeans com os artefatos que foram adquiridos selecionando essa opção na janela ilustrada pela Figura 23. Ao selecionar essa opção o *Wizard* para criação de novos projetos do Netbeans aparecerá.

Caso de uso: Commit

Atores: Usuário

Tipo: Primário e real

Descrição: O usuário escolhe os artefatos de software que deseja enviar ao repositório SVN, ao fim, a operação de commit é realizada.

Seqüência Típica de Eventos

Ação do Autor

Resposta do Sistema

1. Este caso de uso começa quando um usuário solicita a realização de uma operação de commit clicando em A da Figura 42.
2. O usuário registra uma mensagem descrevendo a operação de *commit* em B da Figura 42.
3. Mostra os artefatos de software disponíveis para commit em D da Figura 42.
4. O usuário seleciona os artefatos que deseja atualizar no repositório em C da Figura 42.
5. O usuário indica que selecionou todos os artefatos desejados clicando em E da Figura 42.
6. Realiza a operação.

Caso de uso: Update

Atores: Usuário

Tipo: Primário e real

Descrição: O usuário solicita a realização da operação de *update*, ao final, a operação de *update* é realizada.

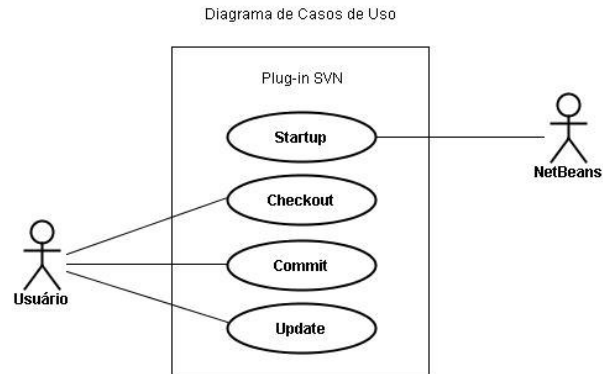
Seqüência Típica de Eventos

Ação do Autor

Resposta do Sistema

1. Este caso de uso começa quando um usuário solicita a realização de uma operação de *Update* clicando no item de menu *Checkout All Files* do menu SVN.
2. Realiza a operação de *Update*.

Diagrama de caso de uso:



Modelos conceituais e diagrama de classes:

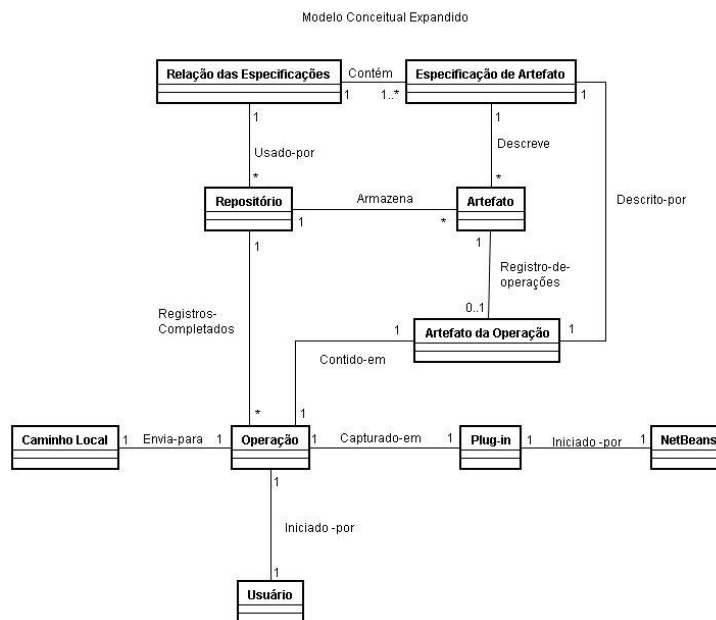
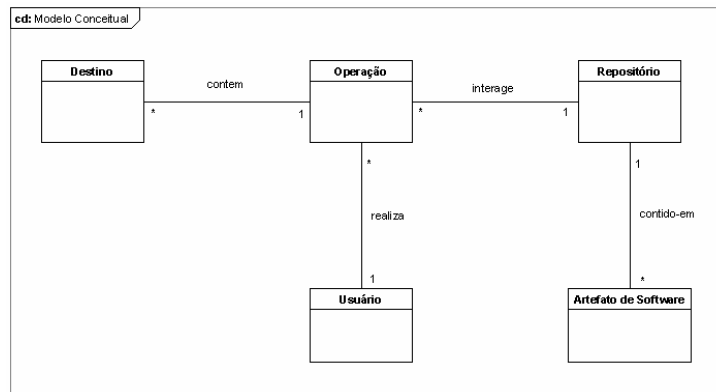
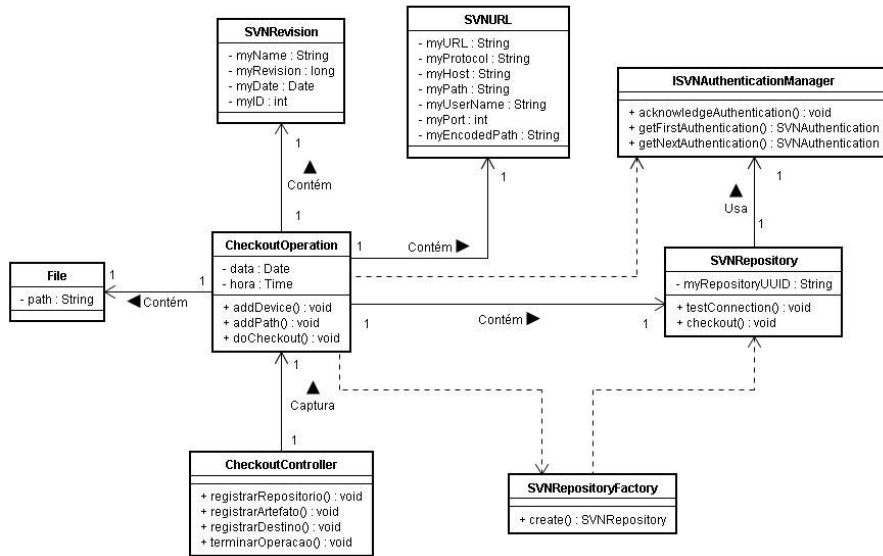


Diagrama de Classes



Diagramas de seqüência:

Diagrama de Seqüência do caso de uso Checkout

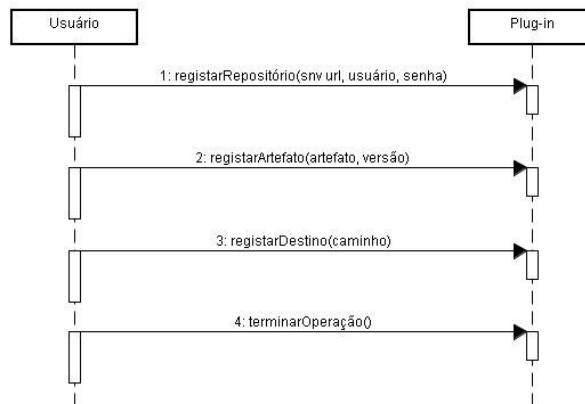


Diagrama de Sequência do caso de uso Commit

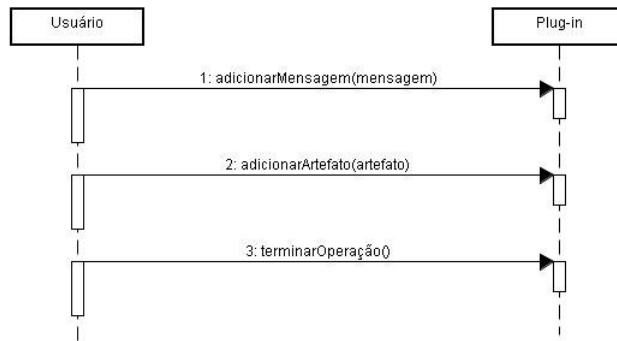
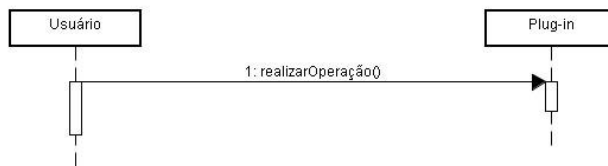


Diagrama de Sequência do caso de uso Update



Diagramas de colaboração do caso de uso *Checkout*:

Diagrama de colaboração da operação registrarRepositório do caso de uso Checkout

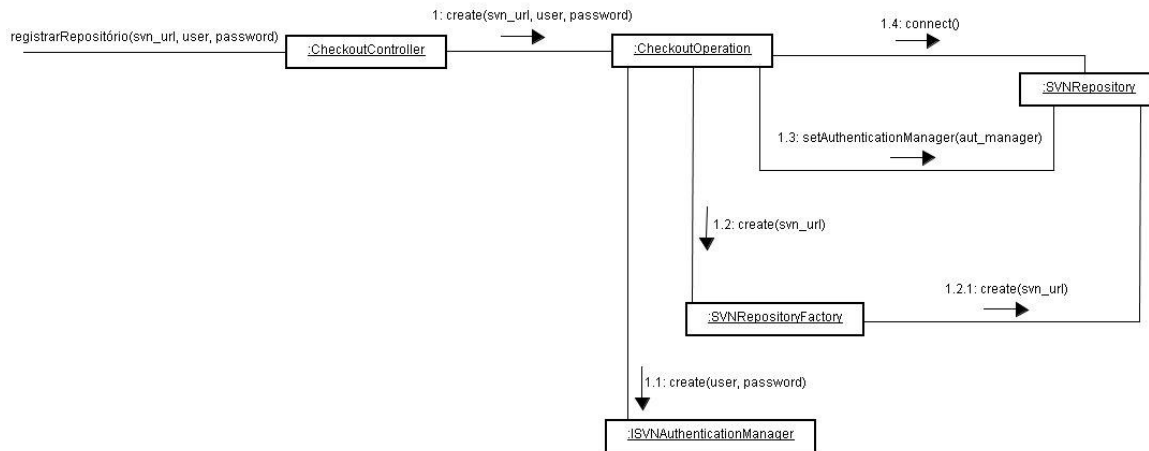


Diagrama de colaboração da operação registrarArtefato do caso de uso Checkout

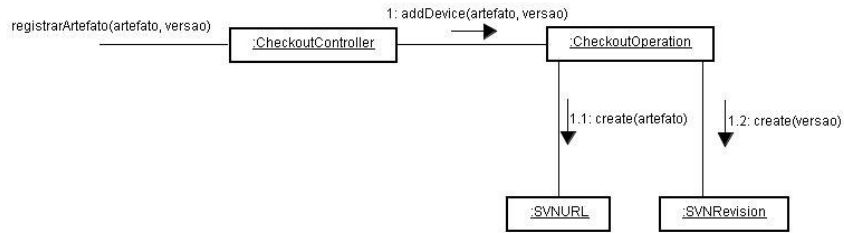


Diagrama de colaboração da operação registrarDestino do caso de uso Checkout

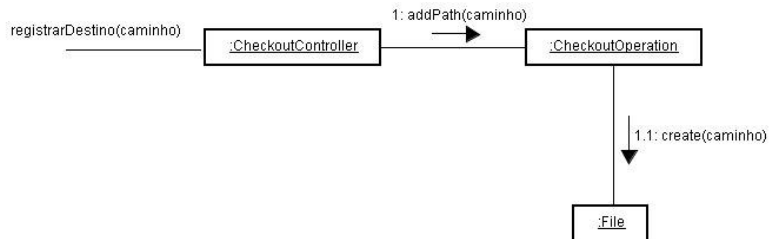


Diagrama de colaboração da operação terminarOperação do caso de uso Checkout



Diagramas de colaboração do caso de uso *Commit*:

Diagrama de colaboração da operação adicionarMensagem do caso de uso Commit

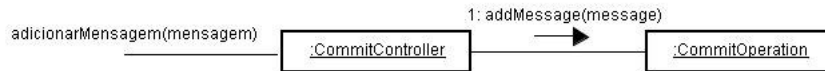


Diagrama de colaboração da operação adicionarArtefato do caso de uso Commit

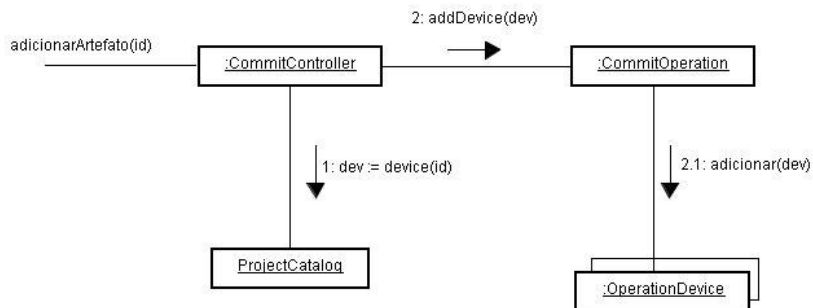


Diagrama de colaboração da operação terminarOperação do caso de uso Commit

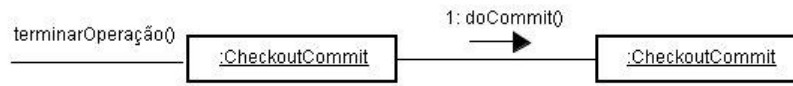
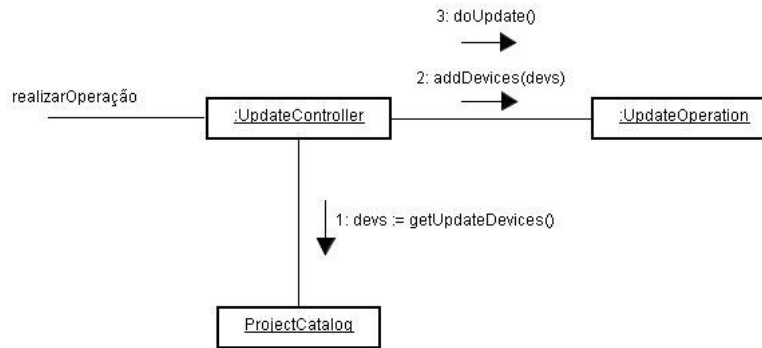


Diagrama de colaboração do caso de uso *Update*:

Diagrama de colaboração da operação realizarOperação do caso de uso Update



Anexo 3 – Código Fonte

Classe CommitAllAction

```
package org.svn.subversionmodule.actions;

import java.awt.Dialog;
import org.netbeans.api.project.Project;
import org.openide.DialogDescriptor;
import org.openide.DialogDisplayer;
import org.openide.NotifyDescriptor;
import org.openide.nodes.Node;
import org.openide.util.HelpCtx;
import org.openide.util.NbBundle;
import org.openide.util.actions.CookieAction;
import org.svn.subversionmodule.common.CommitFile;
import org.svn.subversionmodule.domain.CommitControllerImp;
import org.svn.subversionmodule.gui.CommitFrame;
import org.svn.subversionmodule.interfaces.CommitController;

public final class CommitAllAction extends CookieAction {
    /* objeto controlador das ações */
    private CommitController controller;

    /* janela de commit */
    private CommitFrame commitFrame;

    protected void performAction(Node[] activatedNodes) {
        controller = new CommitControllerImp();
        commitFrame = new
CommitFrame(controller.GetFilesToCommit());

        /* se o usuário apertou ok temos que realizar o commit */
        if(commitFrame.isFinalized()) {
            for(CommitFile file : commitFrame.getSelectedFiles()) {
                controller.adicionarArtefato(file.getId());
            }
            controller.adicionarMensagem(commitFrame.getMessage());
            controller.terminarOperacao();
        }
    }
}
```

```

        /* finalizo td */
        commitFrame.clear();
    }

    protected int mode() {
        return CookieAction.MODE_EXACTLY_ONE;
    }

    public String getName() {
        return NbBundle.getMessage(CommitAllAction.class,
"CTL_CommitAllAction");
    }

    protected Class[] cookieClasses() {
        return new Class[] {
            Project.class
        };
    }

    protected String iconResource() {
        return "org/svn/subversionmodule/resources/commit.gif";
    }

    public HelpCtx getHelpCtx() {
        return HelpCtx.DEFAULT_HELP;
    }

    protected boolean asynchronous() {
        return false;
    }
}

```

Classe SampleAction

```

package org.svn.subversionmodule.actions;

import java.awt.Component;

```

```

import java.awt.Dialog;
import java.awt.event.ActionEvent;
import java.io.File;
import java.text.MessageFormat;
import javax.swing.Action;
import javax.swing.JComponent;
import javax.swing.JOptionPane;
import javax.swing.SwingUtilities;
import org.netbeans.api.project.Project;
import org.netbeans.api.project.ProjectManager;
import org.netbeans.api.project.ProjectUtils;
import org.netbeans.api.project.ui.OpenProjects;
import org.netbeans.spi.project.ProjectFactory;
import org.netbeans.spi.project.ui.support.CommonProjectActions;
import org.netbeans.spi.project.ui.support.ProjectChooser;
import org.openide.DialogDisplayer;
import org.openide.ErrorManager;
import org.openide.WizardDescriptor;
import org.openide.explorer.ExplorerManager;
import org.openide.filesystems.FileObject;
import org.openide.filesystems.FileUtil;
import org.openide.nodes.Node;
import org.openide.util.ContextAwareAction;
import org.openide.util.HelpCtx;
import org.openide.util.Lookup;
import org.openide.util.actions.CallableSystemAction;
import org.openide.util.lookup.Lookups;
import org.openide.windows.TopComponent;
import org.openide.windows.WindowManager;
import org.svn.subversionmodule.domain.CheckoutControllerImpl;
import org.svn.subversionmodule.gui.*;
import org.svn.subversionmodule.interfaces.CheckoutController;

public final class SampleAction extends CallableSystemAction {
    private WizardDescriptor.Panel[] panels;

    /* objeto controlador das ações */
    private CheckoutController controller;

    public void performAction() {

```

```

controller = new CheckoutControllerImpl();

WizardDescriptor wizardDescriptor = new
WizardDescriptor(getPanels());
// {0} will be replaced by
WizardDescriptor.Panel.getComponent().getName()
wizardDescriptor.setTitleFormat(new MessageFormat("{0}"));
wizardDescriptor.setTitle("Checkout");
Dialog dialog =
DialogDisplayer.getDefault().createDialog(wizardDescriptor);
dialog.setVisible(true);
dialogToFront();
File checkoutPath = new File(((CheckoutWizardWizardPanel2)
panels[1]).clear());

//Bom, aki acabou o checkout wizard, entao vemos se foi finish
boolean cancelled = wizardDescriptor.getValue() !=
WizardDescriptor.FINISH_OPTION;
if (!cancelled) {
//verifico se o q foi feito update eh um projeto NetBeansValido
FileObject cP = FileUtil.toFileObject(checkoutPath);
boolean saveSettings = false;

//vou tentar 3x
int tentativas = 0;
while( tentativas < 3) {
if (ProjectManager.getDefault().isProject(cP)) {
saveSettings = this.openProject(cP);
break;
}
tentativas++;
}

if(tentativas >= 3)
saveSettings = this.createProject(cP);
}
}

private boolean openProject(FileObject checkoutPath) {

```

```

    //pergunto pro cara se ele quer criar um project com os dados do
checkout
    Object[] options = {"Open Project", "Close"};
    int resp = JOptionPane.showOptionDialog(null,
        "<html>The project was checked out.<p>Do you want to
open the project?",
        "Checkout Completed",
        JOptionPane.YES_NO_OPTION,
        JOptionPane.INFORMATION_MESSAGE,
        null,
        options,
        options[0]);
    if (resp == JOptionPane.YES_OPTION) {
        try {
            Project p =
ProjectManager.getDefault().findProject(checkoutedPath);
            if (p != null) {
                this.openProject(p);
            }
        } catch (Exception e) {
            ErrorManager err = ErrorManager.getDefault();
            err.annotate(e, "SampleAction::openProject");
            err.notify(e);
        }
        return true;
    }
    return false;
}

```

```

private void openProject(Project p) {
    Project[] projects = new Project[] {p};
    OpenProjects.getDefault().open(projects, false);

    //ja coloco o projeto aberto como main e abro ele
    ContextAwareAction action = (ContextAwareAction)
CommonProjectActions.setAsMainProjectAction();
    Lookup ctx = Lookups.singleton(p);
    Action ctxAction = action.createContextAwareInstance(ctx);
    ctxAction.actionPerformed(null);
    this.selectAndExpandProject(p);
}

```



```

    }

    private void selectAndExpandProject(final Project p) {
        SwingUtilities.invokeLater(new Runnable() {
            TopComponent tc =
WindowManager.getDefault().findTopComponent("projectTabLogical_t
c");
            final ExplorerManager.Provider ptLogial =
(ExplorerManager.Provider) tc;

            public void run() {
                Node root =
ptLogial.getExplorerManager().getRootContext();
                // Node projNode = root.getChildren().findChild(
p.getProjectDirectory().getName());
                Node projNode = root.getChildren().findChild(
ProjectUtils.getInformation(p).getName());
                if (projNode != null) {
                    try {
                        ptLogial.getExplorerManager().setSelectedNodes( new
Node[] { projNode } );
                    } catch (Exception ignore) {}
                }
            }
        });
    }

    private boolean createProject(FileObject checkoutPath) {
        //pergunto pro cara se ele quer criar um project com os dados do
checkout
        Object[] options = {"Create Project...", "Close"};
        int resp = JOptionPane.showOptionDialog(null,
            "Do you want to create an IDE project from the checked-out
sources?",
            "Checkout Completed",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.INFORMATION_MESSAGE,
            null,
            options,

```

```

        options[0]);
    if (resp == JOptionPane.YES_OPTION) {
        this.newProjectWizard(checkoutedPath);
        return true;
    }

    return false;
}

private void newProjectWizard(FileObject checkoutedPath) {
    Action action = CommonProjectActions.newProjectAction();
    if (action != null) {

action.putValue(CommonProjectActions.EXISTING_SOURCES_FOLDER, checkoutedPath);
        performAction(action);
    }
}

private boolean performAction(Action a) {
    if (a == null) {
        return false;
    }
    ActionEvent ae = new ActionEvent(SampleAction.class,
ActionEvent.ACTION_PERFORMED, "command");
    try {
        a.actionPerformed(ae);
        return true;
    } catch (Exception e) {
        ErrorManager.getDefault().notify(ErrorManager.WARNING, e);
        return false;
    }
}

private WizardDescriptor.Panel[] getPanels() {
    if (panels == null) {
        /* Crio os dois paineis de checkout*/
        CheckoutWizardWizardPanel1 panel1 = new
CheckoutWizardWizardPanel1(controller);

```

```

        CheckoutWizardWizardPanel2    panel2    =    new
CheckoutWizardWizardPanel2(controller);

panels = new WizardDescriptor.Panel[] {panel1, panel2};
String[] steps = new String[panels.length];

for (int i = 0; i < panels.length; i++) {
    Component c = panels[i].getComponent();
    // Default step name to component name of panel. Mainly
useful
    // for getting the name of the target chooser to appear in the
    // list of steps.
    steps[i] = c.getName();
    if (c instanceof JComponent) { // assume Swing components
        JComponent jc = (JComponent) c;
        // Sets step number of a component
        jc.putClientProperty("WizardPanel_contentSelectedIndex",
new Integer(i));
        // Sets steps names for a panel
        jc.putClientProperty("WizardPanel_contentData", steps);
        // Turn on subtitle creation on each step
        jc.putClientProperty("WizardPanel_autoWizardStyle",
Boolean.TRUE);
        // Show steps on the left side with the image on the
background
        jc.putClientProperty("WizardPanel_contentDisplayed",
Boolean.TRUE);
        // Turn on numbering of all steps
        jc.putClientProperty("WizardPanel_contentNumbered",
Boolean.TRUE);
    }
}
return panels;
}

public String getName() {
    return "Checkout...";
}

```

```

public String iconResource() {
    return "org/svn/subversionmodule/resources/checkout.gif";
}

public HelpCtx getHelpCtx() {
    return HelpCtx.DEFAULT_HELP;
}

protected boolean asynchronous() {
    return false;
}
}

```

Classe UpdateAllAction

```

package org.svn.subversionmodule.actions;

import org.netbeans.api.project.Project;
import org.openide.nodes.Node;
import org.openide.util.HelpCtx;
import org.openide.util.NbBundle;
import org.openide.util.actions.CookieAction;
import org.svn.subversionmodule.domain.UpdateControllerImpl;
import org.svn.subversionmodule.interfaces.UpdateController;

public final class UpdateAllAction extends CookieAction {
    /* objeto controlador das ações */
    private UpdateController controller;

    protected void performAction(Node[] activatedNodes) {
        controller = new UpdateControllerImpl();
        controller.realizarOperacao();
    }

    protected int mode() {
        return CookieAction.MODE_EXACTLY_ONE;
    }
}

```

```

    public String getName() {
        return NbBundle.getMessage(UpdateAllAction.class,
"CTL_UpdateAllAction");
    }

    protected Class[] cookieClasses() {
        return new Class[] {
            Project.class
        };
    }

    protected String iconResource() {
        return "org/svn/subversionmodule/resources/update.gif";
    }

    public HelpCtx getHelpCtx() {
        return HelpCtx.DEFAULT_HELP;
    }

    protected boolean asynchronous() {
        return false;
    }
}

```

Classe CommitFile

```

/*
 * CommitFile.java
 *
 * Created on 12 de Janeiro de 2007, 19:24
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package org.svn.subversionmodule.common;

/**

```

```

*
* @author lucio
*/
public class CommitFile {
    private int id;
    private String fileName;
    private String repository;
    private String status;

    /** Creates a new instance of CommitFile */
    public CommitFile() {
    }

    public CommitFile(int id, String fileName, String repository, String
status) {
        this.id = id;
        this.fileName = fileName;
        this.repository = repository;
        this.status = status;
    }

    public int getId() {
        return this.id;
    }

    public String getFileName() {
        return this.fileName;
    }

    public String getRepository() {
        return repository;
    }

    public String getStatus() {
        return this.status;
    }
}

```

Classe SVNRepositoryLogin

```
/*
 * SVNRepositoryLogin.java
 *
 * Created on 8 de Janeiro de 2007, 21:04
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package org.svn.subversionmodule.common;

import java.io.Serializable;

/**
 *
 * @author lucio
 */
public class SVNRepositoryLogin implements Serializable {
    private String url;
    private String user;
    private String password;

    /** Creates a new instance of SVNRepositoryLogin */
    public SVNRepositoryLogin(String url, String user, String password)
    {
        this.url = url;
        this.user = user;
        this.password = password;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public void setUser(String user) {
        this.user = user;
    }
}
```

```

    public void setPassword(String Password) {
        this.password = password;
    }

    public String getUrl() {
        return this.url;
    }

    public String getUser() {
        return this.user;
    }

    public String getPassword() {
        return this.password;
    }
}

```

Classe CheckoutControllerImpl

```

/*
 * CheckoutControllerImpl.java
 *
 * Created on 26 de Dezembro de 2006, 16:03
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package org.svn.subversionmodule.domain;

import java.util.Set;
import org.svn.subversionmodule.common.SVNRepositoryLogin;
import org.svn.subversionmodule.interfaces.AuthAccessObject;
import org.svn.subversionmodule.interfaces.CheckoutController;
import
org.svn.subversionmodule.persistence.NetBeansFileAuthAccess;
import org.tmatesoft.svn.core.SVNException;

/**

```



```

*
* @author lucio
*/
public class CheckoutControllerImpl implements CheckoutController {
    private CheckoutOperation operation;
    private AuthAccessObject authManager;

    /** Creates a new instance of CheckoutControllerImpl */
    public CheckoutControllerImpl() {
        authManager = new NetBeansFileAuthAccess();
    }

    /** adiciona o repositório e as informações de login a operação
    corrente */
    public void registrarRepositorio(String svn_url, String user, String
    password) throws Exception {
        operation = new CheckoutOperation(svn_url, user, password);
        authManager.storeAuth(new SVNRepositoryLogin(svn_url, user,
    password));
    }

    /** adiciona um artefato e sua versão a operação corrente */
    public void registrarArtefato(String artefato, String versao) {
        operation.setDevice(artefato, versao);
    }

    /** adiciona o o destino da operação corrente */
    public void registrarDestino(String destino) {
        operation.setPath(destino);
    }

    /** realiza a operação */
    public void terminarOperacao() {
        operation.doCheckout();
    }

    /** retorna os logins que ja foram utilizados para conexão */
    public Set<SVNRepositoryLogin> getUsedLogins() {
        return authManager.getUsedLogins();
    }
}

```

```

    /* retorna a raiz do repositório */
    public String getRepositoryRoot() {
        return operation.getRepositoryRoot();
    }

    /* retorna os nós filhos deste nó */
    public Set<String> getChildrens(String parent) {
        return operation.getChildrens(parent);
    }
}

```

Classe CheckoutOperation

```

/*
 * CheckoutOperation.java
 *
 * Created on 26 de Dezembro de 2006, 16:08
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package org.svn.subversionmodule.domain;

import java.util.Collection;
import java.util.HashSet;
import java.util.Set;
import org.tmatesoft.svn.core.SVNDirEntry;
import org.tmatesoft.svn.core.SVNNodeKind;
import
org.tmatesoft.svn.core.internal.io.svn.SVNRepositoryFactoryImpl;
import org.tmatesoft.svn.core.io.SVNRepositoryFactory;
import org.tmatesoft.svn.core.io.SVNRepository;
import org.tmatesoft.svn.core.wc.ISVNOptions;
import org.tmatesoft.svn.core.wc.SVNRevision;
import org.tmatesoft.svn.core.wc.SVNUpdateClient;
import org.tmatesoft.svn.core.wc.SVNWCUtil;
import org.tmatesoft.svn.core.SVNException;

```

```

import org.tmatesoft.svn.core.SVNURL;
import java.io.File;

/**
 *
 * @author lucio
 */
public class CheckoutOperation {

    private SVNRepository repository;
    private SVNRevision revision;
    private SVNURL url;
    private File destination;

    /** Creates a new instance of CheckoutOperation */
    public CheckoutOperation(String repository, String user, String
password) throws SVNException {
        /* Iniciamos a biblioteca JavaSVN*/
        SVNRepositoryFactoryImpl.setup();

        this.repository =
SVNRepositoryFactory.create(SVNURL.parseURIEncoded(repository))
;

        this.repository.setAuthenticationManager(SVNWCUtil.createDefaultAut
henticationManager(user, password));
        this.repository.testConnection();
    }

    public void setDevice(String artefato, String revision) {
        try {
            this.url = SVNURL.parseURIEncoded(artefato);
            this.revision = SVNRevision.parse(revision);
        } catch(SVNException exp) {
            System.out.println("CheckoutOperation: SVNException setting
device: " + exp.getMessage());
        }
    }

    public void setPath(String destination) {

```

```

    this.destination = new File(destination);
}

public void doCheckout() {
    try {
        ISVNOptions options = SVNWCUtil.createDefaultOptions(true);
        SVNUpdateClient update = new
SVNUpdateClient(repository.getAuthenticationManager(), options);
        update.doCheckout(url, destination, revision, revision, true);
    } catch(SVNException exp) {
        System.out.println("CheckoutOperation: SVNException
checkoutting: " + exp.getMessage());
    }
}

public String getRepositoryRoot() {
    try {
        return repository.getRepositoryRoot(true).toString();
    } catch(SVNException exp) {
        System.out.println("CheckoutOperation: SVNException getting
repository root: " + exp.getMessage());
    }
    return null;
}

public Set<String> getChildrens(String parent) {
    Set<String> childrens = new HashSet<String>();
    try {
        Collection entries = repository.getDir(parent, -1, null,
(Collection) null);

        for(Object oEntry : entries) {
            SVNDirEntry entry = (SVNDirEntry) oEntry;
            if (entry.getKind() == SVNNodeKind.DIR) {
                childrens.add(entry.getName());
            }
        }
    } catch(Exception e) {
        System.out.println("CheckoutOperation: Exception getting
childrens: " + e.getMessage());
    }
}

```

```
    }
    return childrens;
}
}
```

Classe CommitControllerImp

```
/*
 * CommitControllerImp.java
 *
 * Created on 12 de Janeiro de 2007, 19:11
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package org.svn.subversionmodule.domain;

import java.util.Set;
import org.svn.subversionmodule.common.CommitFile;
import org.svn.subversionmodule.domain.ProjectCatalog;
import org.svn.subversionmodule.interfaces.CommitController;

/**
 *
 * @author lucio
 */
public class CommitControllerImp implements CommitController {
    /* esse cara me informa os arquivos q podem ser committed */
    private ProjectCatalog projectCatalog;

    /* o cara que realiza a operação de commit */
    private CommitOperation operation;

    /** Creates a new instance of CommitControllerImp */
    public CommitControllerImp() {
        projectCatalog = new ProjectCatalog();
        operation = new CommitOperation();
    }
}
```

```

    public void adicionarArtefato(Integer id) {
        operation.addDevice(projectCatalog.getDevice(id));
    }

    public void adicionarMensagem(String message) {
        operation.addMessage(message);
    }

    public void terminarOperacao() {
        operation.doCommit();
    }

    public Set<CommitFile> getFilesToCommit() {
        return projectCatalog.getFilesToCommit();
    }
}

```

Classe CommitOperation

```

/*
 * CommitOperation.java
 *
 * Created on 12 de Janeiro de 2007, 19:28
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package org.svn.subversionmodule.domain;

import java.io.File;
import java.util.HashSet;
import java.util.Set;
import org.svn.subversionmodule.common.CommitFile;
import org.tmatesoft.svn.core.SVNException;
import org.tmatesoft.svn.core.wc.ISVNOptions;
import org.tmatesoft.svn.core.wc.SVNCommitClient;
import org.tmatesoft.svn.core.wc.SVNWCUtil;

```

```

/**
 *
 * @author lucio
 */
public class CommitOperation {
    private String message = null;
    private Set<CommitFile> files;

    /** Creates a new instance of CommitOperation */
    public CommitOperation() {
        files = new HashSet<CommitFile>();
    }

    public void addMessage(String message) {
        this.message = message;
    }

    public void addDevice(CommitFile file) {
        files.add(file);
    }

    public void doCommit() {
        try {
            ISVNOptions options = SVNWCUtil.createDefaultOptions(true);

            for(CommitFile file: files) {
                File[] path = new File[1];
                path[0] = new File(file.getFileName());

                SVNCommitClient commitClient = new
                SVNCommitClient(file.getRepository().getAuthenticationManager(),
                options);
                commitClient.doCommit(path, true, message, true, true);
            }
        } catch(SVNException ex) {
            System.out.println("CommitOperation: SVNException comitting:
            " + ex.getMessage());
        }
    }
}

```

```
}
```

Classe ProjectCatalog

```
/*
 * ProjectCatalog.java
 *
 * Created on 12 de Janeiro de 2007, 19:33
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package org.svn.subversionmodule.domain;

import java.io.File;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
import org.netbeans.api.project.Project;
import org.netbeans.api.project.ui.OpenProjects;
import org.svn.subversionmodule.common.CommitFile;
import org.tmatesoft.svn.core.SVNException;
import
org.tmatesoft.svn.core.internal.io.svn.SVNRepositoryFactoryImpl;
import org.tmatesoft.svn.core.internal.wc.SVNDirectory;
import org.tmatesoft.svn.core.internal.wc.SVNEntry;

/**
 *
 * @author lucio
 */
public class ProjectCatalog {

    /** Creates a new instance of ProjectCatalog */
    public ProjectCatalog() {
        /* Iniciamos a biblioteca JavaSVN*/
        SVNRepositoryFactoryImpl.setup();
    }
}
```



```

public CommitFile getDevice(Integer id) {
    return null;
}

private Set<CommitFile> getVersionedProjectPaths() {
    Set<CommitFile> files = new HashSet<CommitFile>();

    /* varro todos os projetos */
    Project[] projects = OpenProjects.getDefault().getOpenProjects();
    for(int i = 0; i < projects.length; i ++) {
        int id = 0;
        String fileName = null;
        String repository = null;
        String status = "versioned";

        /* verifico se esse projeto esta em um diretório versionado */
        fileName = projects[i].getProjectDirectory().getPath();
        File projectPath = new
File(projects[i].getProjectDirectory().getPath());
        SVNDirectory file = new SVNDirectory(null, "", projectPath);
        if(!file.isVersioned()) continue;

        try {
            Iterator it = file.getEntries().entries(true);
            if(it.hasNext()) {
                SVNEntry entry = (SVNEntry) it.next();
                repository = entry.getRepositoryRoot();
            }
        } catch(SVNException exp) {
            System.out.println("ProjectCatalog: SVNException getting
files to commit: " + exp.getMessage());
        }

        files.add(new CommitFile(id, fileName, repository, status));
    }
    return files;
}

public Set<CommitFile> getFilesToCommit() {

```

```

        return this.getVersionedProjectPaths();
    }

    public Set<CommitFile> getFilesToUpdate() {
        return this.getVersionedProjectPaths();
    }
}

```

Classe UpdateControllerImpl

```

/*
 * UpdateControllerImpl.java
 *
 * Created on 18 de Janeiro de 2007, 18:08
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package org.svn.subversionmodule.domain;

import org.svn.subversionmodule.domain.UpdateOperation;
import org.svn.subversionmodule.interfaces.UpdateController;

/**
 *
 * @author lucio
 */
public class UpdateControllerImpl implements UpdateController {
    /* esse cara me informa os arquivos q podem ser committed */
    private ProjectCatalog projectCatalog;

    /* o cara que realiza a operação de update */
    private UpdateOperation operation;

    /** Creates a new instance of UpdateControllerImpl */
    public UpdateControllerImpl() {
        projectCatalog = new ProjectCatalog();
        operation = new UpdateOperation();
    }
}

```

```

    }

    public void realizarOperacao() {
        operation.addDevices(projectCatalog.GetFilesToUpdate());
        operation.doUpdate();
    }
}

```

Classe UpdateOperation

```

/*
 * UpdateOperation.java
 *
 * Created on 18 de Janeiro de 2007, 19:39
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package org.svn.subversionmodule.domain;

import java.io.File;
import java.util.Set;
import org.svn.subversionmodule.common.CommitFile;
import org.tmatesoft.svn.core.SVNException;
import org.tmatesoft.svn.core.wc.ISVNOptions;
import org.tmatesoft.svn.core.wc.SVNRevision;
import org.tmatesoft.svn.core.wc.SVNUpdateClient;
import org.tmatesoft.svn.core.wc.SVNWCUtil;

/**
 *
 * @author lucio
 */
public class UpdateOperation {
    Set<CommitFile> files = null;

    /** Creates a new instance of UpdateOperation */
    public UpdateOperation() {}
}

```

```

public void addDevices(Set<CommitFile> files) {
    this.files = files;
}

public void doUpdate() {
    try {
        ISVNOptions options = SVNWCUtil.createDefaultOptions(true);

        for(CommitFile file: files) {
            SVNUpdateClient updateClient = new
SVNUpdateClient(file.getRepository().getAuthenticationManager(),
options);
            updateClient.doUpdate(new File(file.getFileName()),
SVNRevision.HEAD, true);
        }
    } catch(SVNException ex) {
        System.out.println("CommitOperation: SVNException comitting:
" + ex.getMessage());
    }
}
}

```

Classe CheckoutWizardVisualPanel1

```

package org.svn.subversionmodule.gui;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Collection;
import java.util.HashMap;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JPanel;
import org.svn.subversionmodule.common.SVNRepositoryLogin;

```

```

public final class CheckOutWizardVisualPanel1 extends JPanel {
    protected CheckoutWizardWizardPanel1 controller;
    HashMap<String, SVNRepositoryLogin> logins = new HashMap();

    //the components used in connect thread
    private JComponent progressComponent;
    private JLabel progressLabel;

    public CheckOutWizardVisualPanel1(CheckoutWizardWizardPanel1
controller) {
        this.controller = controller;
        initComponents();

        // adiciono ao combo box os logins ja efetuados
        for(SVNRepositoryLogin login : controller.getConnectionLogins())
        {
            logins.put(login.getUrl(), login);
            jComboBoxRoot.addItem(login.getUrl());
        }
        // Seto o usuário e o password do login atual
        this.setSelectedLogin();
    }

    private void setSelectedLogin() {
        SVNRepositoryLogin          actualLogin          =
logins.get(jComboBoxRoot.getSelectedItem());
        if(actualLogin == null) return;
        jTextFieldUser.setText(actualLogin.getUser());
        jTextFieldPassword.setText(actualLogin.getPassword());
        controller.fireChangeEvent();
    }

    public String getName() {
        return "SVN Root";
    }

    public String getUrl() {
        Object o = jComboBoxRoot.getSelectedItem();
        if ((o instanceof String) && o != null)
            return (String) o;
    }
}

```

```

    return new String("");
}

public String getUser() {
    return jTextFieldUser.getText();
}

public String getPassword() {
    return new String(jTextFieldPassword.getPassword());
}

// <editor-fold defaultstate="collapsed" desc=" Generated Code ">
private void initComponents() {
    jLabel2 = new javax.swing.JLabel();
    jLabel1 = new javax.swing.JLabel();
    jTextFieldUser = new javax.swing.JTextField();
    jLabel4 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jLabel5 = new javax.swing.JLabel();
    jTextFieldPassword = new javax.swing.JPasswordField();
    jComboBoxRoot = new javax.swing.JComboBox();

    org.openide.awt.Mnemonics.setLocalizedText(jLabel2, "User:");

    org.openide.awt.Mnemonics.setLocalizedText(jLabel1, "SVN
Root:");

    jTextFieldUser.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jTextFieldUserActionPerformed(evt);
        }
    });
    jTextFieldUser.addInputMethodListener(new
java.awt.event.InputMethodListener() {
        public void
caretPositionChanged(java.awt.event.InputMethodEvent evt) {
}
}

```

```

        public void
inputMethodTextChanged(java.awt.event.InputMethodEvent evt) {
    jTextFieldUserInputMethodTextChanged(evt);
}
});
jTextFieldUser.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyReleased(java.awt.event.KeyEvent evt) {
        jTextFieldUserKeyReleased(evt);
    }
    public void keyTyped(java.awt.event.KeyEvent evt) {
        jTextFieldUserKeyTyped(evt);
    }
});

    org.openide.awt.Mnemonics.setLocalizedText(jLabel4, "Specify
location of SVN Repository defined by SVN root.");

    org.openide.awt.Mnemonics.setLocalizedText(jLabel3,
"(svn://hostname)");

    org.openide.awt.Mnemonics.setLocalizedText(jLabel5,
"Password:");

    jTextFieldPassword.addKeyListener(new
java.awt.event.KeyAdapter() {
        public void keyReleased(java.awt.event.KeyEvent evt) {
            jTextFieldPasswordKeyReleased(evt);
        }
    });

    jComboBoxRoot.setEditable(true);
    jComboBoxRoot.addItemListener(new
java.awt.event.ItemListener() {
        public void itemStateChanged(java.awt.event.ItemEvent evt) {
            jComboBoxRootItemStateChanged(evt);
        }
    });
    jComboBoxRoot.addInputMethodListener(new
java.awt.event.InputMethodListener() {

```

```

        public void
caretPositionChanged(java.awt.event.InputMethodEvent evt) {
    }
    public void
inputMethodTextChanged(java.awt.event.InputMethodEvent evt) {
        JComboBoxRootInputMethodTextChanged(evt);
    }
});
jComboBoxRoot.addKeyListener(new
java.awt.event.KeyAdapter() {
    public void keyPressed(java.awt.event.KeyEvent evt) {
        JComboBoxRootKeyPressed(evt);
    }
    public void keyReleased(java.awt.event.KeyEvent evt) {
        JComboBoxRootKeyReleased(evt);
    }
    public void keyTyped(java.awt.event.KeyEvent evt) {
        JComboBoxRootKeyTyped(evt);
    }
});
jComboBoxRoot.addMouseListener(new
java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        JComboBoxRootMouseClicked(evt);
    }
});

    org.jdesktop.layout.GroupLayout layout = new
org.jdesktop.layout.GroupLayout(this);
    this.setLayout(layout);
    layout.setHorizontalGroup(

layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING
)
        .add(org.jdesktop.layout.GroupLayout.TRAILING,
layout.createSequentialGroup()

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAI
LING)

```



```

        .add(org.jdesktop.layout.GroupLayout.LEADING,
layout.createSequentialGroup())

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEA
DING)
        .add(jLabel1)
        .add(jLabel2)
        .add(jLabel5))

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEA
DING)
        .add(jLabel3))

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAI
LING, false)
        .add(org.jdesktop.layout.GroupLayout.LEADING,
jTextFieldPassword)
        .add(org.jdesktop.layout.GroupLayout.LEADING,
jTextFieldUser, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
145, Short.MAX_VALUE))
        .add(jComboBoxRoot,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,          379,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED,      22,
Short.MAX_VALUE))
        .add(jLabel4,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,          476,
Short.MAX_VALUE))
        .add(150, 150, 150))
);
layout.setVerticalGroup(

layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING
)
        .add(layout.createSequentialGroup()
        .add(jLabel4)
        .add(14, 14, 14)

```

```

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BAS
ELINE)
        .add(jLabel1)
        .add(jComboBoxRoot,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
        .add(jLabel3)
        .add(21, 21, 21)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BAS
ELINE)
        .add(jLabel2)
        .add(jTextFieldUser,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BAS
ELINE)
        .add(jLabel5)
        .add(jTextFieldPassword,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
        .addContainerGap(161, Short.MAX_VALUE)
    );
} // </editor-fold>
private void jComboBoxRootKeyTyped(java.awt.event.KeyEvent evt)
{
    controller.fireChangeEvent();
}
private void jComboBoxRootInputMethodTextChanged(java.awt.event.InputMethod
Event evt) {

```

```

        controller.fireChangeEvent();
    }
    private void jComboBoxRootKeyPressed(java.awt.event.KeyEvent
evt) {
        controller.fireChangeEvent();
    }
    private void
jComboBoxRootMouseClicked(java.awt.event.MouseEvent evt) {
        controller.fireChangeEvent();
    }
    private void
jComboBoxRootItemStateChanged(java.awt.event.ItemEvent evt) {
        this.setSelectedLogin();
    }
    private void jComboBoxRootKeyReleased(java.awt.event.KeyEvent
evt) {
        controller.fireChangeEvent();
    }
    private void
jTextFieldPasswordKeyReleased(java.awt.event.KeyEvent evt) {
        controller.fireChangeEvent();
    }
    private void jTextFieldUserKeyReleased(java.awt.event.KeyEvent
evt) {
        controller.fireChangeEvent();
    }
    private void jTextFieldUserKeyTyped(java.awt.event.KeyEvent evt) {
        controller.fireChangeEvent();
    }
    private void
jTextFieldUserActionPerformed(java.awt.event.ActionEvent evt) {

    }
    private void
jTextFieldUserInputMethodTextChanged(java.awt.event.InputMethodE
vent evt) {

    }

    void startThreadComponents(JComponent bar) {

```

```

//desabilito as entradas do usuario
jTextFieldUser.setEditable(false);
jTextFieldPassword.setEditable(false);
jComboBoxRoot.setEditable(false);

//Crio o q sera usado para indicar o progresso da thread
JButton stopButton = new JButton("Stop");
stopButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        controller.stopThread();
    }
});

progressComponent = new JPanel();
progressComponent.setLayout(new BorderLayout(6, 0));
progressLabel = new JLabel("Connecting..");
progressComponent.add(progressLabel, BorderLayout.NORTH);
progressComponent.add(bar, BorderLayout.CENTER);
progressComponent.add(stopButton, BorderLayout.LINE_END);

this.setLayout(new BorderLayout());
this.add(progressComponent, BorderLayout.SOUTH);
this.revalidate();
this.repaint();
}

void stopThreadComponents() {
    //i remove the visual thread components
    this.remove(progressComponent);

    //and set the input components
    jTextFieldUser.setEditable(true);
    jTextFieldPassword.setEditable(true);
    jComboBoxRoot.setEditable(true);

    this.revalidate();
    this.repaint();
}

// Variables declaration - do not modify

```

```

private javax.swing.JComboBox jComboBoxRoot;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JPasswordField jTextFieldPassword;
public javax.swing.JTextField jTextFieldUser;
// End of variables declaration

}

```

Classe CheckoutWizardVisualPanel2

```

package org.svn.subversionmodule.gui;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.filechooser.FileSystemView;
import javax.swing.tree.DefaultMutableTreeNode;

public final class CheckoutWizardVisualPanel2 extends JPanel {
    protected CheckoutWizardWizardPanel2 controller;

    /* componentes usados na pela thread de execução de checkout */
    private JComponent progressComponent;
    private JLabel progressLabel;

```

```

    public CheckoutWizardVisualPanel2(CheckoutWizardWizardPanel2
controller) {
        initComponents();
        this.controller = controller;
    }

    public String getName() {
        return "Module to Checkout";
    }

    public String getModule() {
        return jTextFieldModule.getText();
    }

    public String getLocalFolder() {
        return jTextFieldLocalFolder.getText();
    }

    public String getVersion() {
        if(jRadioButtonHead.isSelected()) return "HEAD";
        return jTextFieldRevision.getText();
    }

    void startThreadComponents(JComponent bar) {
        //desabilito as entradas do usuario
        jTextFieldModule.setEditable(false);
        jTextFieldLocalFolder.setEditable(false);
        jButton1.setEnabled(false);
        jButton3.setEnabled(false);

        //Crio o q sera usado para indicar o progresso da thread
        JButton stopButton = new JButton("Stop");
        stopButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                controller.stopThread();
            }
        });

        progressComponent = new JPanel();
    }

```

```

progressComponent.setLayout(new BorderLayout(6, 0));
progressLabel = new JLabel("Checkoutting..");
progressComponent.add(progressLabel, BorderLayout.NORTH);
progressComponent.add(bar, BorderLayout.CENTER);
progressComponent.add(stopButton, BorderLayout.LINE_END);

this.setLayout(new BorderLayout());
this.add(progressComponent, BorderLayout.SOUTH);
this.revalidate();
this.repaint();
}

```

```

void stopThreadComponents() {
    //i remove the visual thread components
    this.remove(progressComponent);

    //and set the input components
    jTextFieldModule.setEditable(true);
    jTextFieldLocalFolder.setEditable(true);
    jButton1.setEnabled(true);
    jButton3.setEnabled(true);

    this.revalidate();
    this.repaint();
}
// <editor-fold defaultstate="collapsed" desc=" Generated Code ">
private void initComponents() {
    jLabel1 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jTextFieldModule = new javax.swing.JTextField();
    jTextFieldLocalFolder = new javax.swing.JTextField();
    jLabel4 = new javax.swing.JLabel();
    jButton1 = new javax.swing.JButton();
    jLabel5 = new javax.swing.JLabel();
    jButton3 = new javax.swing.JButton();
    jLabel6 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    jButtonHead = new javax.swing.JRadioButton();
    jLabel7 = new javax.swing.JLabel();
    jLabel8 = new javax.swing.JLabel();
}

```

```

jRadioButtonOtherRevision = new javax.swing.JRadioButton();
jTextFieldRevision = new javax.swing.JTextField();
jButton2 = new javax.swing.JButton();

org.openide.awt.Mnemonics.setLocalizedText(jLabel1, "Module:");

org.openide.awt.Mnemonics.setLocalizedText(jLabel3, "Local
Folder: ");

jTextFieldModule.addKeyListener(new
java.awt.event.KeyAdapter() {
    public void keyReleased(java.awt.event.KeyEvent evt) {
        jTextFieldModuleKeyReleased(evt);
    }
    public void keyTyped(java.awt.event.KeyEvent evt) {
        jTextFieldModuleKeyTyped(evt);
    }
});

jTextFieldLocalFolder.addKeyListener(new
java.awt.event.KeyAdapter() {
    public void keyReleased(java.awt.event.KeyEvent evt) {
        jTextFieldLocalFolderKeyReleased(evt);
    }
});

org.openide.awt.Mnemonics.setLocalizedText(jLabel4, "Specify
the SVN module and branch to checkout from SVN repository.");

org.openide.awt.Mnemonics.setLocalizedText(jButton1,
"Browse...");
jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jButton1MouseClicked(evt);
    }
});

org.openide.awt.Mnemonics.setLocalizedText(jLabel5, "Specify
location of local folder to checkout module into.");

```



```

    org.openide.awt.Mnemonics.setLocalizedText(jButton3,
    "Browse...");
    jButton3.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseClicked(java.awt.event.MouseEvent evt) {
            jButton3MouseClicked(evt);
        }
    });

    org.openide.awt.Mnemonics.setLocalizedText(jLabel6, "(local
    SVN working directory)");

    org.openide.awt.Mnemonics.setLocalizedText(jLabel2, "(empty
    means all modules)");

    jButtonHead.setSelected(true);
    org.openide.awt.Mnemonics.setLocalizedText(jButtonHead,
    "HEAD revision");

jRadioButtonHead.setBorder(javax.swing.BorderFactory.createEmptyB
order(0, 0, 0, 0));
    jButtonHead.setMargin(new java.awt.Insets(0, 0, 0, 0));

    org.openide.awt.Mnemonics.setLocalizedText(jLabel7, "Specify
    the revision to checkout.");

    org.openide.awt.Mnemonics.setLocalizedText(jLabel8, "Revision:
    ");

org.openide.awt.Mnemonics.setLocalizedText(jButtonOtherRevis
ion, "Other revision: ");

jRadioButtonOtherRevision.setBorder(javax.swing.BorderFactory.creat
eEmptyBorder(0, 0, 0, 0));
    jButtonOtherRevision.setMargin(new java.awt.Insets(0, 0, 0,
    0));

    jTextFieldRevision.setEditable(false);

```

```

        org.openide.awt.Mnemonics.setLocalizedText(jButton2, "Show
revisions...");
        jButton2.setEnabled(false);

        org.jdesktop.layout.GroupLayout layout = new
org.jdesktop.layout.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING
)
        .add(jLabel4)
        .add(layout.createSequentialGroup()
            .add(jLabel5)
            .add(ContainerGap())
            .add(layout.createSequentialGroup()

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAI
LING, false)
            .add(org.jdesktop.layout.GroupLayout.LEADING,
layout.createSequentialGroup()

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEA
DING)
            .add(jLabel3)
            .add(jLabel8))

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEA
DING)
            .add(jLabel6)
            .add(jRadioButtonHead)
            .add(layout.createSequentialGroup()
                .add(jRadioButtonOtherRevision)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
            .add(jTextFieldRevision,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 210,
Short.MAX_VALUE)

```

```

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
    .add(jButton2))
    .add(jTextFieldLocalFolder,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,           390,
Short.MAX_VALUE)))
    .add(org.jdesktop.layout.GroupLayout.LEADING,
layout.createSequentialGroup())
    .add(jLabel1)
    .add(32, 32, 32)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEA
DING)
    .add(jLabel2)
    .add(jTextFieldModule,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,           430,
Short.MAX_VALUE))))

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED,   6,
Short.MAX_VALUE)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEA
DING)
    .add(jButton1)
    .add(jButton3))
    .add(103, 103, 103))
    .add(layout.createSequentialGroup())
    .add(jLabel7)
    .add(ContainerGap())
);
layout.setVerticalGroup(

layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING
)
    .add(layout.createSequentialGroup())
    .add(jLabel4)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

```

```
.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BAS  
ELINE)
```

```
    .add(jLabel1)  
    .add(jButton1)  
    .add(jTextFieldModule,  
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,  
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,  
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
```

```
.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)  
    .add(jLabel2)  
    .add(33, 33, 33)  
    .add(jLabel5)
```

```
.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
```

```
.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BAS  
ELINE)
```

```
    .add(jLabel3)  
    .add(jButton3)  
    .add(jTextFieldLocalFolder,  
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,  
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,  
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
```

```
.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEA  
DING)
```

```
    .add(layout.createSequentialGroup())
```

```
.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)  
    .add(jLabel6)  
    .add(46, 46, 46)
```

```
.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BAS  
ELINE)
```

```
    .add(jRadioButtonHead)  
    .add(jLabel8)))  
    .add(layout.createSequentialGroup())  
    .add(49, 49, 49)
```

```

        .add(jLabel7)))

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BAS
ELINE)
        .add(jRadioButtonOtherRevision)
        .add(jButton2)
        .add(jTextFieldRevision,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
        .add(87, 87, 87))
    );
} // </editor-fold>
private void
jTextFieldModuleKeyReleased(java.awt.event.KeyEvent evt) {

}
private void
jTextFieldModuleKeyTyped(java.awt.event.KeyEvent evt) {

}
private void
jTextFieldLocalFolderKeyReleased(java.awt.event.KeyEvent evt) {
    controller.fireChangeEvent();
}
private void
jButton3MouseClicked(java.awt.event.MouseEvent evt) {
    /* Abre uma janela para escolha de diretório
(JFileChooser) */
    JFileChooser chooser = new JFileChooser();
    chooser.setDialogTitle("Brownse Local Folder");

chooser.setSelectionMode(JFileChooser.DIRECTORIES_ONLY);
    int returnVal = chooser.showOpenDialog(this);
    if(returnVal == JFileChooser.APPROVE_OPTION) {

jTextFieldLocalFolder.setText(chooser.getSelectedFile().getPath());
    }
}

```

```

        controller.fireChangeEvent();
    }

    private void jButton1MouseClicked(java.awt.event.MouseEvent evt)
    {
        /* mostra uma tela com os diretórios do repositório */
        jTextFieldModule.setText(ListDialog.showDialog(this, jLabel1,
"Repository Contents:", "Browse SVN Module",
controller.getController()));
    }

    public String clear() {
        String temp = jTextFieldLocalFolder.getText();
        jTextFieldLocalFolder.setText("");
        jTextFieldModule.setText("");
        return temp;
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButton3;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JRadioButton jButtonHead;
    private javax.swing.JRadioButton jButtonOtherRevision;
    private javax.swing.JTextField jTextFieldLocalFolder;
    private javax.swing.JTextField jTextFieldModule;
    private javax.swing.JTextField jTextFieldRevision;
    // End of variables declaration

}

```

Classe CheckoutWizardWizardPanel1

```
package org.svn.subversionmodule.gui;

import java.awt.Component;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import javax.swing.SwingUtilities;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import org.netbeans.api.progress.ProgressHandle;
import org.netbeans.api.progress.ProgressHandleFactory;
import org.openide.ErrorManager;
import org.openide.WizardDescriptor;
import org.openide.WizardValidationException;
import org.openide.util.HelpCtx;
import org.openide.util.NbBundle;
import org.svn.subversionmodule.interfaces.CheckoutController;
import org.svn.subversionmodule.common.SVNRepositoryLogin;

public class CheckoutWizardWizardPanel1 implements
WizardDescriptor.AsynchronousValidatingPanel {
    /* atributos referentes ao wizard */
    private final Set<ChangeListener> listeners = new
HashSet<ChangeListener>(1);
    private Component component;

    /* objeto que gerencia as ações */
    private CheckoutController controller;

    /* objetos para controlar da thread de validação da conexão */
    private ProgressHandle progress;
    private boolean force_validate = false;
    private Thread validationThread = null;
    private String error = null;
```

```

public CheckoutWizardWizardPanel1(CheckoutController controller)
{
    this.controller = controller;
}

public void setError(String error){
    this.error = error;
}

public String getError() {
    return this.error;
}

public Set<SVNRepositoryLogin> getConnectionLogins() {
    return controller.getUsedLogins();
}

public Component getComponent() {
    if (component == null) {
        component = new CheckOutWizardVisualPanel1(this);
    }
    return component;
}

public HelpCtx getHelp() {
    return HelpCtx.DEFAULT_HELP;
}

public boolean isValid() {
    if (force_validate) return false;

    CheckoutWizardVisualPanel1 panel1 =
    (CheckoutWizardVisualPanel1) this.getComponent();
    return (panel1.getUrl().length() != 0) &&
    (panel1.getUser().length() != 0) &&
    (panel1.getPassword().length() != 0) );
}

public final void addChangeListener(ChangeListener l) {
    synchronized (listeners) {

```



```

        listeners.add(l);
    }
}

public final void removeChangeListener(ChangeListener l) {
    synchronized (listeners) {
        listeners.remove(l);
    }
}

protected final void fireChangeEvent() {
    Iterator<ChangeListener> it;
    synchronized (listeners) {
        it = new HashSet<ChangeListener>(listeners).iterator();
    }
    ChangeEvent ev = new ChangeEvent(this);
    while (it.hasNext()) {
        it.next().stateChanged(ev);
    }
}

public void stopThread() {
    try {
        if (validationThread == null) return;

        validationThread.interrupt();
    } catch (SecurityException exp) {
        System.out.println("==== [ERROR] The current thread cannot
modify this thread");
    }
}

public void validate() throws WizardValidationException {
    final CheckoutWizardVisualPanel1 panel1 =
(CheckoutWizardVisualPanel1) this.getComponent();
    validationThread = Thread.currentThread();
    this.setError(null);

    Thread tempThread = new Thread(){
        public void run() {

```

```

        try {
            controller.registrarRepositorio(panel1.getUrl(),
panel1.getUser(), panel1.getPassword());
        } catch (Exception e) {
            setError(e.getMessage());
        }
    }
};

try {
    tempThread.start();
    tempThread.join();
} catch (InterruptedException e) {
    this.setError("Verification stopped by user!");
}

// ao fim, liberamos os botões
force_validate = false;
fireChangeEvent();
panel1.stopThreadComponents();
validationThread = null;
tempThread = null;

if(this.getError() != null) {
    String temp = this.getError();
    this.setError(null);
    throw new WizardValidationException(panel1, temp, temp);
}
}

public void prepareValidation() {
    CheckoutWizardVisualPanel1 panel1 =
(CheckOutWizardVisualPanel1) this.getComponent();

    progress = ProgressHandleFactory.createHandle("The Thread
handle");

panel1.startThreadComponents(ProgressHandleFactory.createProgressComponent(progress));
    progress.start();
}

```

```

        force_validate = true;
        this.fireChangeEvent();
    }

    public void readSettings(Object object) {}
    public void storeSettings(Object object) {}
}

```

Classe CheckoutWizardWizardPanel2

```

package org.svn.subversionmodule.gui;

import java.awt.Component;
import java.io.File;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import org.netbeans.api.progress.ProgressHandle;
import org.netbeans.api.progress.ProgressHandleFactory;
import org.openide.WizardDescriptor;
import org.openide.WizardDescriptor;
import org.openide.WizardDescriptor;
import org.openide.WizardValidationException;
import org.openide.util.HelpCtx;
import org.openide.util.RequestProcessor;
import org.svn.subversionmodule.interfaces.CheckoutController;

public class CheckoutWizardWizardPanel2 implements
    WizardDescriptor.AsynchronousValidatingPanel {
    /* atributos referentes ao wizard */
    private final Set<ChangeListener> listeners = new
    HashSet<ChangeListener>(1);
    private Component component;

    /* objeto que gerencia as ações */
    private CheckoutController controller;

```

```

/* objetos para controlar da thread de validação da conexão */
private Thread validationThread = null;
private ProgressHandle progress;
private boolean force_validate = false;
private String error = null;

public CheckoutWizardWizardPanel2(CheckoutController controller)
{
    this.controller = controller;
}

public CheckoutController getController() {
    return controller;
}

public Component getComponent() {
    if (component == null) {
        component = new CheckOutWizardVisualPanel2(this);
    }
    return component;
}

public HelpCtx getHelp() {
    return HelpCtx.DEFAULT_HELP;
}

public boolean isValid() {
    // usado na thread de checkout
    if (force_validate) return false;

    CheckoutWizardVisualPanel2 panel2 =
    (CheckoutWizardVisualPanel2) this.getComponent();
    return (panel2.getLocalFolder().length() != 0);
}

private String getError() {
    return error;
}

public void setError(String error) {

```

```

    this.error = error;
}

public final void addChangeListener(ChangeListener l) {
    synchronized (listeners) {
        listeners.add(l);
    }
}

public final void removeChangeListener(ChangeListener l) {
    synchronized (listeners) {
        listeners.remove(l);
    }
}

protected final void fireChangeEvent() {
    Iterator<ChangeListener> it;
    synchronized (listeners) {
        it = new HashSet<ChangeListener>(listeners).iterator();
    }
    ChangeEvent ev = new ChangeEvent(this);
    while (it.hasNext()) {
        it.next().stateChanged(ev);
    }
}

public void stopThread() {
    try {
        if (validationThread == null) return;

        validationThread.interrupt();
    } catch (SecurityException exp) {
        System.out.println("CheckoutWizardWizardPanel2:
SecurityException stopping thread: " + exp.getMessage());
    }
}

public void validate() throws WizardValidationException {
    final CheckoutWizardVisualPanel2 panel2 =
(CheckOutWizardVisualPanel2) getComponent();
    validationThread = Thread.currentThread();
    this.setError(null);
}

```

```

Thread connection_thread = new Thread(){
    public void run() {
        try {
            controller.registrarArtefato(panel2.getModule(),
panel2.getVersion());
            controller.registrarDestino(panel2.getLocalFolder());
            controller.terminarOperacao();
        } catch (Exception e) {
            setError(e.getMessage());
        }
    }
};

try {
    connection_thread.start();
    connection_thread.join();
} catch (Exception e) {
    setError("Checkout stopped by user!");
}

force_validate = false;
fireChangeEvent();
panel2.stopThreadComponents();
validationThread = null;
connection_thread = null;

if (this.getError() != null) {
    String temp = this.getError();
    this.setError(null);
    throw new WizardValidationException(panel2, temp, temp);
}

}

public void prepareValidation() {
    CheckoutWizardVisualPanel2 panel2 =
(CheckOutWizardVisualPanel2) this.getComponent();

    progress = ProgressHandleFactory.createHandle("The Thread
handle");

```

```

panel2.startThreadComponents(ProgressHandleFactory.createProgressComponent(progress));
    progress.start();
    force_validate = true;
    this.fireChangeEvent();
}

public String clear() {
    return ((CheckoutWizardVisualPanel2) getComponent()).clear();
}

public void readSettings(Object settings) {}
public void storeSettings(Object settings) {}
}

```

Classe CommitFrame

```

/*
 * CommitFrame.java
 *
 * Created on 11 de Janeiro de 2007, 14:47
 */

package org.svn.subversionmodule.gui;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;
import java.util.HashSet;
import java.util.Set;
import javax.swing.Icon;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JTable;
import javax.swing.filechooser.FileSystemView;
import javax.swing.table.DefaultTableModel;
import org.svn.subversionmodule.common.CommitFile;
import org.svn.subversionmodule.gui.CommitTableRender;

```

```

/**
 *
 * @author lucio
 */
public class CommitFrame extends javax.swing.JDialog implements
ItemChangeListener {
    /* os arquivos habilitados para commit */
    Set<CommitFile> filesToCommit;

    /* variavel que indica como terminou o dialog */
    private boolean finalized = false;

    /** Creates new form CommitFrame */
    public CommitFrame(Set<CommitFile> filesToCommit) {
        this.filesToCommit = filesToCommit;
        initComponents();

        /* p/ fechar a janela */
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                ((CommitFrame) we.getSource()).setFinalized(false);
                ((CommitFrame) we.getSource()).setVisible(false);
            }
        });

        /* configurando a tabela */
        jTable1.setModel(new CommitTableModel());
        jTable1.setShowGrid(false);
        jTable1.setShowVerticalLines(false);
        jTable1.setShowHorizontalLines(false);
        jTable1.setDefaultRenderer(CommitTableColumn.class,      new
CommitTableRender());
        jTable1.setDefaultEditor(CommitTableColumn.class,      new
CommitTableEditor());

        /* populando a tabela com os arquivos */
        this.populateTable(filesToCommit);

        /* inicio */

```



```

        this.setLocationRelativeTo(null);
        this.setModal(true);
        this.setVisible(true);
        thisToFront();
    }

    public void setFinalized(boolean finalized) {
        this.finalized = finalized;
    }

    public boolean isFinalized() {
        return this.finalized;
    }

    public Set<CommitFile> getSelectedFiles() {
        CommitTableModel model = (CommitTableModel)
jTable1.getModel();
        return model.getSelectedFiles();
    }

    /** This method is called from within the constructor to
    * initialize the form.
    * WARNING: Do NOT modify this code. The content of this method
is
    * always regenerated by the Form Editor.
    */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents() {
        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        jPanel1 = new javax.swing.JPanel();
        jScrollPane2 = new javax.swing.JScrollPane();
        jTable1 = new javax.swing.JTable();
        jCheckBox1 = new javax.swing.JCheckBox();
        jCheckBox2 = new javax.swing.JCheckBox();
        jLabel1 = new javax.swing.JLabel();
        jPanel2 = new javax.swing.JPanel();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTextArea1 = new javax.swing.JTextArea();

```

```

setDefaultCloseOperation(javax.swing.WindowConstants.DO_NOTHING_ON_CLOSE);
    setTitle("Commit - All Projects");
    setModal(true);
    jButton1.setText("Commit");
    jButton1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton1ActionPerformed(evt);
        }
    });

    jButton2.setText("Cancel");
    jButton2.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton2ActionPerformed(evt);
        }
    });

jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder("Files to Commit:"));
jScrollPane2.setBackground(new java.awt.Color(255, 255, 255));
jTable1.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {

    },
    new String [] {
        "File", "Status", "Repository Path"
    }
) {
    boolean[] canEdit = new boolean [] {
        false, false, false
    };

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }
});
jScrollPane2.setViewportViewView(jTable1);

```

```

jCheckBox1.setSelected(true);
jCheckBox1.setText("Show unversioned files");

jCheckBox1.setBorder(javax.swing.BorderFactory.createEmptyBorder(
0, 0, 0, 0));
jCheckBox1.setMargin(new java.awt.Insets(0, 0, 0, 0));
jCheckBox1.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jCheckBox1ActionPerformed(evt);
    }
});

jCheckBox2.setText("Select / deselect all");

jCheckBox2.setBorder(javax.swing.BorderFactory.createEmptyBorder(
0, 0, 0, 0));
jCheckBox2.setMargin(new java.awt.Insets(0, 0, 0, 0));
jCheckBox2.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jCheckBox2ActionPerformed(evt);
    }
});

jLabel1.setText("0 files selected, 0 files total");

org.jdesktop.layout.GroupLayout jPanel1Layout = new
org.jdesktop.layout.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.L
EADING)
    .add(jPanel1Layout.createSequentialGroup()
        .addContainerGap()

.add(jPanel1Layout.createParallelGroup(org.jdesktop.layout.GroupLay
out.LEADING)

```

```

        .add(jScrollPane2,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,           520,
Short.MAX_VALUE)
        .add(jCheckBox2)
        .add(jPanel1Layout.createSequentialGroup())
        .add(jCheckBox1)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED,  256,
Short.MAX_VALUE)
        .add(jLabel1)))
        .addContainerGap()
    );
    jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.L
EADING)
        .add(org.jdesktop.layout.GroupLayout.TRAILING,
jPanel1Layout.createSequentialGroup())
        .add(jScrollPane2,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,           184,
Short.MAX_VALUE)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(jPanel1Layout.createParallelGroup(org.jdesktop.layout.GroupLay
out.BASELINE)
        .add(jCheckBox1)
        .add(jLabel1))

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
        .add(jCheckBox2))
    );

jPanel2.setBorder(javax.swing.BorderFactory.createTitledBorder("Mes
sage:"));
    jTextArea1.setColumns(20);
    jTextArea1.setRows(4);
    jScrollPane1.setViewportView(jTextArea1);

```

```

    org.jdesktop.layout.GroupLayout jPanel2Layout = new
org.jdesktop.layout.GroupLayout(jPanel2);
    jPanel2.setLayout(jPanel2Layout);
    jPanel2Layout.setHorizontalGroup(

```

```

jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.L
EADING)
    .add(jPanel2Layout.createSequentialGroup()
        .addContainerGap()
        .add(jScrollPane1,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,          520,
Short.MAX_VALUE)
        .addContainerGap())
    );
    jPanel2Layout.setVerticalGroup(

```

```

jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.L
EADING)
    .add(jPanel2Layout.createSequentialGroup()
        .addContainerGap()
        .add(jScrollPane1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,          80,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
        .addContainerGap(org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
    );

```

```

    org.jdesktop.layout.GroupLayout layout = new
org.jdesktop.layout.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(

```

```

layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING
)
    .add(org.jdesktop.layout.GroupLayout.TRAILING,
layout.createSequentialGroup()
        .addContainerGap()

```

```

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING)
    .add(org.jdesktop.layout.GroupLayout.LEADING, jPanel1,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
    .add(org.jdesktop.layout.GroupLayout.LEADING, jPanel2,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
    .add(layout.createSequentialGroup()
        .add(jButton1)
        .add(16, 16, 16)
        .add(jButton2)))
    .addContainerGap())
);
layout.setVerticalGroup(

```

```

layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING
)
    .add(layout.createSequentialGroup()
        .addContainerGap()
        .add(jPanel2,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
        .add(20, 20, 20)
        .add(jPanel1,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
        .add(17, 17, 17)
    )

```

```

.add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
    .add(jButton2)
    .add(jButton1))

```

```

.addContainerGap(org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
    );
    pack();
} // </editor-fold>

private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
    this.finalized = true;
    this.setVisible(false);
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent
evt) {
    this.finalized = false;
    this.setVisible(false);
}

private void populateTable(Set<CommitFile> filesToCommit) {
    /* uso um JChosser para pegar os ícones dos arquivos, tem jeito
mais facil??? */
    JFileChooser tempChooser = new JFileChooser();
    FileSystemView systemView =
tempChooser.getFileSystemView();
    if(systemView == null) {
        System.out.println("CommitFrame: error getting
FileSystemView");
        return;
    }
}

```

```

        CommitTableModel      model      =      (CommitTableModel)
jTable1.getModel();

        for(CommitFile file : filesToCommit) {
            File tempFile = new File(file.getFileName());
            Icon fileIcon = systemView.getSystemIcon(tempFile);

            model.addRow(new      Object[]      {new
CommitTableColumn(file.getFileName(), fileIcon, this), file.getStatus(),
file.getRepository()});
        }
        this.updateSelectedFilesStatus(model.getSelectedFilesNumber(),
model.getRowCount());
    }

    private      void
jCheckBox1ActionPerformed(java.awt.event.ActionEvent evt) {
        CommitTableModel      model      =      (CommitTableModel)
jTable1.getModel();
        model.clear();

        if(jCheckBox1.isSelected()) {
            this.populateTable(filesToCommit);
            return;
        }
        Set<CommitFile> versionedFiles = new HashSet<CommitFile>();
        for(CommitFile file : filesToCommit) {
            if(file.getStatus().equalsIgnoreCase(new      String("non-
versioned"))) continue;
            versionedFiles.add(file);
        }
        this.populateTable(versionedFiles);
    }

    public String getMessage() {
        return jTextArea1.getText();
    }

    public void clear() {

```



```

        CommitTableModel    model    =    (CommitTableModel)
jTable1.getModel();
        model.clear();
        jTextArea1.setText(null);
        jCheckBox1.setSelected(true);
        jCheckBox2.setSelected(false);
        jLabel1.setText("0 files selected, 0 files total");
    }

    private void updateSelectedFilesStatus(int selectedFiles, int
totalFiles) {
        jLabel1.setText(String.valueOf(selectedFiles) +
            " files selected, " +
            String.valueOf(totalFiles) +
            " files total");
    }

    public void itemStateChanged(boolean isSelected) {
        CommitTableModel    model    =    (CommitTableModel)
jTable1.getModel();
        this.updateSelectedFilesStatus(model.getSelectedFilesNumber(),
model.getRowCount());
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JCheckBox jCheckBox1;
    private javax.swing.JCheckBox jCheckBox2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTable jTable1;
    private javax.swing.JTextArea jTextArea1;
    // End of variables declaration
}

```

**Classes CommitTableRender,
CommitTableModel e CommitTableColumn**

CommitTableEditor,

```
/*  
 * CommitTableRender.java  
 *  
 * Created on 15 de Janeiro de 2007, 22:13  
 *  
 * To change this template, choose Tools | Template Manager  
 * and open the template in the editor.  
 */
```

```
package org.svn.subversionmodule.gui;
```

```
import java.awt.Color;  
import java.awt.Component;  
import java.awt.Dimension;  
import java.awt.GridLayout;  
import java.awt.SystemColor;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.ItemEvent;  
import java.awt.event.ItemListener;  
import java.util.EventObject;  
import java.util.HashSet;  
import java.util.Set;  
import javax.swing.AbstractCellEditor;  
import javax.swing.DefaultCellEditor;  
import javax.swing.JCheckBox;  
import javax.swing.JLabel;  
import javax.swing.JPanel;  
import javax.swing.JTable;  
import javax.swing.event.CellEditorListener;  
import javax.swing.table.AbstractTableModel;  
import javax.swing.table.TableCellEditor;  
import javax.swing.table.TableCellRenderer;  
import javax.swing.DefaultCellEditor;  
import javax.swing.Icon;  
import org.jdesktop.layout.GroupLayout;  
import org.svn.subversionmodule.common.CommitFile;
```

```

/**
 *
 * @author lucio
 */
public class CommitTableRender implements TableCellRenderer {
    /** Creates a new instance of CommitTableRender */
    public CommitTableRender() {}

    public Component getTableCellRendererComponent(JTable table,
Object value, boolean isSelected, boolean hasFocus, int row, int
column) {
        if(column != 0) return null;
        JPanel panel = ((CommitTableColumn) value).getPanel();

        if (isSelected)
            panel.setBackground(SystemColor.textHighlight);
        else
            panel.setBackground(SystemColor.window);

        panel.repaint();
        return panel;
    }
}

```

```

class CommitTableEditor extends AbstractCellEditor implements
TableCellEditor {
    public CommitTableEditor() {}

    public Object getCellEditorValue() {
        return null;
    }

    public Component getTableCellEditorComponent(JTable table,
Object value, boolean isSelected, int row, int column) {
        if(column != 0) return null;

        JPanel panel = ((CommitTableColumn) value).getPanel();
        return panel;
    }
}

```

```

public boolean isCellEditable(EventObject eventObject) {
    return true;
}

public boolean shouldSelectCell(EventObject eventObject) {
    return true;
}

public boolean stopCellEditing() {
    return true;
}

public void cancelCellEditing() {
}

public void addCellEditorListener(CellEditorListener
cellEditorListener) {
}

public void removeCellEditorListener(CellEditorListener
cellEditorListener) {
}
}

class CommitTableModel extends AbstractTableModel {
    private String[] columnNames = {"File", "Status", "Repository Path"};
    private Object[][] data = new Object[100][3];
    //private Map<Integer, Object[]> data = new HashMap<Integer,
Object[]>();
    int rows = 0;

    public void clear() {
        this.data = new Object[100][3];
        rows = 0;
        fireTableDataChanged();
    }

    public void setSelectedToAll(boolean selected) {
        for(Integer i = 0; i < rows; i++) {

```

```

        ((CommitTableColumn)
data[i][0]).getCheckBox().setSelected(selected);
    }
    fireTableDataChanged();
}

public int getColumnCount() {
    return columnNames.length;
}

public int getRowCount() {
    return rows;
}

public String getColumnName(int col) {
    return columnNames[col];
}

public Object getValueAt(int row, int col) {
    return data[row][col];
}

/*
 * JTable uses this method to determine the default renderer/
 * editor for each cell. If we didn't implement this method,
 * then the last column would contain text ("true"/"false"),
 * rather than a check box.
 */
public Class getColumnClass(int c) {
    return getValueAt(0, c).getClass();
}

public boolean isCellEditable(int row, int col) {
    //Note that the data/cell address is constant,
    //no matter where the cell appears onscreen.
    if (col == 0) {
        return true;
    } else {
        return false;
    }
}

```

```

}

public void setValueAt(Object value, int row, int col) {
    if(row >= rows) return;
    data[row][col] = value;
    fireTableCellUpdated(row, col);
}

public void addRow(Object[] row) {
    data[rows] = row;
    fireTableCellUpdated(rows, 0);
    fireTableCellUpdated(rows, 1);
    fireTableCellUpdated(rows, 2);
    rows++;
}

public Set<CommitFile> getSelectedFiles() {
    Set<CommitFile> selectedFiles = new HashSet<CommitFile>();
    for(Integer i = 0; i < rows; i++) {
        CommitTableColumn firstColumn = (CommitTableColumn)
data[i][0];
        if(!firstColumn.getCheckBox().isSelected()) continue;
        selectedFiles.add(new CommitFile(i,
firstColumn.getLabel().getText(),
(String) data[i][2], (String) data[i][1]));
    }
    return selectedFiles;
}

public int getSelectedFilesNumber() {
    int selectedFiles = 0;
    for(Integer i = 0; i < rows; i++) {
        if(!((CommitTableColumn)
data[i][0]).getCheckBox().isSelected()) continue;
        selectedFiles++;
    }
    return selectedFiles;
}
}

```

```

class CommitTableColumn implements ItemListener {
    private JPanel panel = null;
    private JCheckBox checkBox = null;
    private JLabel label = null;
    private ItemChangeListener observer;

    public CommitTableColumn(String fileName, Icon fileIcon,
        ItemChangeListener observer) {
        this.observer = observer;

        /* criamos o JCheckBox */
        checkBox = new JCheckBox();
        checkBox.addItemListener(this);
        checkBox.setPreferredSize(new Dimension(5,5));
        checkBox.setBackground(Color.white);

        checkBox.setBorder(javax.swing.BorderFactory.createEmptyBorder(0,
            0, 0, 0));
        checkBox.setMargin(new java.awt.Insets(0, 0, 0, 0));

        /* criamos o label */
        JLabel label = new JLabel(fileName, fileIcon, JLabel.CENTER);

        /* criamos o panel */
        panel = new JPanel();
        panel.setBackground(Color.white);
        GroupLayout jPanel3Layout = new GroupLayout(panel);
        panel.setLayout(jPanel3Layout);
        jPanel3Layout.setHorizontalGroup(

jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.L
EADING)
            .add(jPanel3Layout.createSequentialGroup()
                .add(checkBox)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(label)

.addContainerGap(org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))

```

```

        );
        jPanel3Layout.setVerticalGroup(

jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.L
EADING)

.add(jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLay
out.BASELINE)
        .add(checkBox)
        .add(label))
    );
}

public JPanel getPanel() {
    return this.panel;
}

public JCheckBox getCheckBox() {
    return this.checkBox;
}

public JLabel getLabel() {
    return this.label;
}

public void itemStateChanged(ItemEvent itemEvent) {
    observer.itemStateChanged(itemEvent.getStateChange()
itemEvent.SELECTED);
}
}

interface ItemChangeListener {
    public void itemStateChanged(boolean isSelected);
}

```

Classe ListDialog

```
package org.svn.subversionmodule.gui;
```



```

import java.util.Iterator;
import java.util.Set;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeCellRenderer;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreeNode;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;
import org.svn.subversionmodule.interfaces.CheckoutController;

public class ListDialog extends JDialog implements ActionListener {
    private static ListDialog dialog;
    private static String value = "";
    private JTree repository_tree;
    private DefaultTreeModel treeModel;
    private CheckoutController controller;

    /* Um cara estático que apresenta um ListDialog */
    public static String showDialog(Component frameComp, Component
locationComp, String labelText, String title, CheckoutController
controller) {
        Frame frame =
JOptionPane.getFrameForComponent(frameComp);
        dialog = new ListDialog(frame,
locationComp,
labelText,
title, controller);
        dialog.setVisible(true);
        return value;
    }

    private void setValue(String newValue) {
        value = newValue;
        repository_tree.setSelectionPath(new TreePath(value));
    }
}

```

```

private ListDialog(Frame frame, Component locationComp, String
labelText, String title, CheckoutController controller) {
    super(frame, title, true);
    this.controller = controller;

    JButton cancelButton = new JButton("Cancel");
    cancelButton.addActionListener(this);

    final JButton setButton = new JButton("OK");
    setButton.setActionCommand("Set");
    setButton.addActionListener(this);
    setButton.setEnabled(false);
    getRootPane().setDefaultButton(setButton);

    try {
        //Iniciando a JTree
        DefaultMutableTreeNode top = new
DefaultMutableTreeNode(controller.getRepositoryRoot());
        treeModel = new DefaultTreeModel(top);
        repository_tree = new JTree(treeModel);

        repository_tree.getSelectionModel().setSelectionMode(TreeSelectionM
odel.SINGLE_TREE_SELECTION);

        //Set the icon for leaf nodes.
        DefaultTreeCellRenderer renderer = new
DefaultTreeCellRenderer();
        renderer.setLeafIcon(renderer.getOpenIcon());
        repository_tree.setCellRenderer(renderer);

        addDirectories(top, "");

        repository_tree.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                if (e.getClickCount() == 2) {
                    try {
                        setButton.setEnabled(true);
                        TreePath path =
repository_tree.getSelectionPath();
                    }
                }
            }
        });
    }
}

```

```

        DefaultMutableTreeNode    parent_node    =
(DefaultMutableTreeNode) path.getLastPathComponent();
        path = new TreePath(addDirectories(parent_node,
pathToString(path)).getPath());
        repository_tree.collapsePath(path);
        repository_tree.repaint();
    } catch(Exception exc) {}
    } else {
        if (repository_tree.getSelectionPath() != null)
            setButton.setEnabled(true);
    }
}
});
JScrollPane treeScroller = new JScrollPane(repository_tree);
treeScroller.setPreferredSize(new Dimension(250, 400));
treeScroller.setAlignmentX(LEFT_ALIGNMENT);

JPanel listPane = new JPanel();
listPane.setLayout(new BoxLayout(listPane,
BoxLayout.PAGE_AXIS));
JLabel label = new JLabel(labelText);
label.setLabelFor(repository_tree);
listPane.add(label);
listPane.add(Box.createRigidArea(new Dimension(0,5)));
listPane.add(treeScroller);

listPane.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

//Lay out the buttons from left to right.
JPanel buttonPane = new JPanel();
buttonPane.setLayout(new BoxLayout(buttonPane,
BoxLayout.LINE_AXIS));
buttonPane.setBorder(BorderFactory.createEmptyBorder(0, 10,
10, 10));
buttonPane.add(Box.createHorizontalGlue());
buttonPane.add(cancelButton);
buttonPane.add(Box.createRigidArea(new Dimension(10, 0)));
buttonPane.add(setButton);

```

```

        //Put everything together, using the content pane's
        BorderLayout.
        Container contentPane = getContentPane();
        contentPane.add(listPane, BorderLayout.CENTER);
        contentPane.add(buttonPane, BorderLayout.PAGE_END);

        //Initialize values.
        pack();
        setLocationRelativeTo(locationComp);

    } catch (Exception svne) {
        System.err.println("error while listing entries: " +
svne.getMessage());
        //ListDialog.dialog.setVisible(false);
    }
}

/*private Map<String, Boolean> getDirectories(String path) {
    try {
        //return svnManager.getRepository().getDir(path, -1, null,
(Collection) null);
        return
    } catch (Exception e) {
        System.out.println("ListDialog::getDirectories: " +
e.getMessage());
    }

    return null;
}*/

private DefaultMutableTreeNode
addDirectories(DefaultMutableTreeNode top, String path) {
    Set<String> childrens = controller.getChildren(path);
    DefaultMutableTreeNode node = null;

    for(String children : childrens) {
        top.add(new DefaultMutableTreeNode(children));
    }
    return node;
}

```

```

private String pathToString(TreePath path) {
    String temp = new String("");
    Object[] o = path.getPath();
    for (int i = 0; i < o.length; i++) {
        if (o[i] instanceof DefaultMutableTreeNode) {
            Object p = ( (DefaultMutableTreeNode) o[i]).getUserObject();
            if (p instanceof String)
                temp += (String) p + "/";
        }
    }
    return temp;
}

//Handle clicks on the Ok and Cancel buttons.
public void actionPerformed(ActionEvent e) {
    if ("Set".equals(e.getActionCommand())) {
        TreePath path = (TreePath) repository_tree.getSelectionPath();
        String temp = pathToString(path);
        ListDialog.value = temp;
    }
    ListDialog.dialog.setVisible(false);
}
}

```

Interface AuthAccessObject

```

/*
 * AuthAccessObject.java
 *
 * Created on 11 de Janeiro de 2007, 09:01
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

```

```

package org.svn.subversionmodule.interfaces;

```

```

import java.util.Set;

```

```

import org.svn.subversionmodule.common.SVNRepositoryLogin;

/**
 *
 * @author lucio
 */
public interface AuthAccessObject {
    /* persiste essa autenticação */
    public void storeAuth(SVNRepositoryLogin login);

    /* retorna todas as autenticações */
    public Set<SVNRepositoryLogin> getUsedLogins();
}

```

Interface CheckoutController

```

/*
 * CheckoutController.java
 *
 * Created on 26 de Dezembro de 2006, 15:57
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package org.svn.subversionmodule.interfaces;

import java.util.Set;
import org.svn.subversionmodule.common.SVNRepositoryLogin;

/**
 *
 * @author lucio
 */
public interface CheckoutController {
    /* adiciona o repositório e as informações de login a operação corrente */
    public void registrarRepositorio(String svn_url, String user, String password) throws Exception;
}

```

```

/* adiciona um artefato e sua versão a operação corrente */
public void registrarArtefato(String artefato, String versao);

/* adiciona o o destino da operação corrente */
public void registrarDestino(String destino);

/* realiza a operação */
public void terminarOperacao();

/* retorna os logins que ja foram utilizados para conexão */
public Set<SVNRepositoryLogin> getUsedLogins();

/* retorna a raiz do repositório */
public String getRepositoryRoot();

/* retorna os nós filhos deste nó */
public Set<String> getChildrens(String parent);
}

```

Interface CommitController

```

/*
 * CommitController.java
 *
 * Created on 11 de Janeiro de 2007, 19:47
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package org.svn.subversionmodule.interfaces;

import java.util.Set;
import org.svn.subversionmodule.common.CommitFile;

/**
 *
 * @author lucio

```

```

*/
public interface CommitController {
    /* retorna os arquivos que estão disponpiveis para commit */
    public Set<CommitFile> getFilesToCommit();

    /* Adiciona um artefato para ser committed */
    public void adicionarArtefato(Integer id);

    /* Adiciona uma mensagem que descreve esse commit */
    public void adicionarMensagem(String message);

    /* realiza o commit dos artefatos adicionados */
    public void terminarOperacao();
}

```

Interface UpdateController

```

/*
 * UpdateController.java
 *
 * Created on 18 de Janeiro de 2007, 18:07
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package org.svn.subversionmodule.interfaces;

/**
 *
 * @author lucio
 */
public interface UpdateController {
    /* realiza a operação de Update*/
    public void realizarOperacao();
}

```


Classe NetBeansFileAuthAccess

```
/*
 * NetBeansFileAuthAccess.java
 *
 * Created on 11 de Janeiro de 2007, 09:04
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package org.svn.subversionmodule.persistence;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.HashSet;
import java.util.Set;
import org.openide.filesystems.FileLock;
import org.openide.filesystems.FileObject;
import org.svn.subversionmodule.common.SVNRepositoryLogin;
import org.openide.filesystems.Repository;
import org.svn.subversionmodule.interfaces.AuthAccessObject;

/**
 *
 * @author lucio
 */
public class NetBeansFileAuthAccess implements AuthAccessObject {
    private Set<SVNRepositoryLogin> logins = new
HashSet<SVNRepositoryLogin>();
    private FileObject folder = null;
    private FileObject file = null;
    private FileLock lock = null;

    /** Creates a new instance of NetBeansFileAuthAccess */
    public NetBeansFileAuthAccess() {
        try {
            //se n tem folderObject, crio ele

```

```

        if (Repository.getDefault().getDefaultFileSystem().getRoot().getFileObject(
            "Settings") == null)
            folder = Repository.getDefault().getDefaultFileSystem().getRoot().createFolder(
                "Settings");

        //se n tem settingFile, crio ele
        if ((file = folder.getFileObject("AuthSettings","Cfg")) == null)
            file = folder.createData("AuthSettings","Cfg");

        //pego as informaÃ§Ãµes do arquivo
        this.retrieveLogins();

    } catch (IOException ex) {
        System.out.println("NetBeansFileAuthAccess: IO Exception
        creating manager: " + ex.getMessage());
    }
}

private void dumpLogins() {
    try {
        //salvo o map no arquivo
        lock = file.lock();
        ObjectOutputStream objectOutStr = new
        ObjectOutputStream(file.getOutputStream(lock));
        objectOutStr.writeObject(logins);
        objectOutStr.close();
        lock.releaseLock();
    } catch (IOException ex) {
        System.out.println("NetBeansFileAuthAccess: IO Exception
        dumping logins: " + ex.getMessage());
    }
}

private void retrieveLogins() {
    try {
        ObjectInputStream objectInStr = new
        ObjectInputStream(file.getInputStream());
        logins = (Set<SVNRepositoryLogin>) objectInStr.readObject();
    }
}

```

```

        objectInStr.close();
    } catch (IOException ex) {
        System.out.println("NetBeansFileAuthAccess: IO Exception
retrieving logins: " + ex.getMessage());
    } catch (Exception ex) {
        System.out.println("NetBeansFileAuthAccess: Exception
retrieving logins: " + ex.getMessage());
    }
}

public void storeAuth(SVNRepositoryLogin login) {
    logins.add(login);
    this.dumpLogins();
}

public Set<SVNRepositoryLogin> getUsedLogins() {
    return logins;
}
}

```

Arquivo layer.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE filesystem PUBLIC "-//NetBeans//DTD Filesystem
1.1//EN" "http://www.netbeans.org/dtds/filesystem-1_1.dtd">
<filesystem>
    <folder name="Actions">
        <folder name="SVN">
            <file name="org-svn-subversionmodule-actions-
CommitAllAction.instance">
                <attr name="instanceClass"
stringValue="org.svn.subversionmodule.actions.CommitAllAction"/>
            </file>
            <file name="org-svn-subversionmodule-actions-
SampleAction.instance">
                <attr name="instanceClass"
stringValue="org.svn.subversionmodule.actions.SampleAction"/>
            </file>
        </folder>
    </folder>
</filesystem>

```

```

        <file                name="org-svn-subversionmodule-actions-
UpdateAllAction.instance">
            <attr                name="instanceClass"
stringvalue="org.svn.subversionmodule.actions.UpdateAllAction"/>
        </file>
    </folder>
</folder>
<folder name="Menu">
    <folder name="SVN">
        <file                name="org-svn-subversionmodule-actions-
CommitAllAction.shadow">
            <attr name="originalFile" stringvalue="Actions/SVN/org-svn-
subversionmodule-actions-CommitAllAction.instance"/>
        </file>
        <file                name="org-svn-subversionmodule-actions-
SampleAction.shadow">
            <attr name="originalFile" stringvalue="Actions/SVN/org-svn-
subversionmodule-actions-SampleAction.instance"/>
        </file>
        <attr                name="org-svn-subversionmodule-actions-
SampleAction.shadow/org-svn-subversionmodule-actions-
SampleAction.shadow" boolvalue="true"/>
        <file                name="org-svn-subversionmodule-actions-
UpdateAllAction.shadow">
            <attr name="originalFile" stringvalue="Actions/SVN/org-svn-
subversionmodule-actions-UpdateAllAction.instance"/>
        </file>
        <file                name="org-svn-subversionmodule-actions-
separatorAfter.instance">
            <attr                name="instanceClass"
stringvalue="javax.swing.JSeparator"/>
        </file>
        <attr                name="org-svn-subversionmodule-actions-
CommitAllAction.shadow/org-svn-subversionmodule-actions-
separatorAfter.instance" boolvalue="true"/>
        <attr                name="org-svn-subversionmodule-actions-
separatorAfter.instance/org-svn-subversionmodule-actions-
SampleAction.shadow" boolvalue="true"/>

```

```
        <attr                name="org-svn-subversionmodule-actions-
UpdateAllAction.shadow/org-svn-subversionmodule-actions-
SampleAction.shadow" boolvalue="true"/>
    </folder>
</folder>
</filesystem>
```