CÉSAR ALBERTO DA SILVA

DiSEN-SCV: GERENCIADOR DE VERSÕES DE ARTEFATOS PARA UM AMBIENTE DE DESENVOLVIMENTO DISTRIBUÍDO DE SOFTWARE

MARINGÁ
2008

CÉSAR ALBERTO DA SILVA

DiSEN-SCV: GERENCIADOR DE VERSÕES DE ARTEFATOS PARA UM AMBIENTE DE DESENVOLVIMENTO DISTRIBUÍDO DE SOFTWARE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientadora: Profa. Dra. Elisa Hatsue Moriya Huzita

MARINGÁ
2008

Dados Internacionais de Catalogação-na-Publicação (CIP)
(Biblioteca Central - UEM, Maringá – PR., Brasil)

S586d

Silva, César Alberto da
DiSEN-SCV: gerenciador de versões de artefatos para um ambiente de desenvolvimento distribuído de software / César Alberto da Silva. -- Maringá : [s.n.], 2008.
94 f. : il., figs.

Orientador : Profª. Drª. Elisa Hatsue Moriya Huzita.
Dissertação (mestrado) - Universidade Estadual de Maringá, Programa de Pós-Graduação em Ciência da Computação, 2008.

1. Gerenciamento de versões de artefatos. 2. Desenvolvimento distribuído de software. 3. Percepção. I. Universidade Estadual de Maringá. Programa de Pós-Graduação em Ciência da Computação. II. Título.

CDD 22.ed. 005.1068

CÉSAR ALBERTO DA SILVA

DiSEN-SCV: GERENCIADOR DE VERSÕES DE ARTEFATOS PARA UM AMBIENTE DE DESENVOLVIMENTO DISTRIBUÍDO DE SOFTWARE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Aprovado em 24/07/2008.

BANCA EXAMINADORA

Profa. Dra. Elisa Hatsue Moriya Huzita
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Dante Alves Medeiros Filho
Universidade Estadual de Maringá – DIN/UEM

Profa. Dra. Rosângela Aparecida Dellosso Penteado
Universidade Federal de São Carlos – DC/UFSCar

DEDICATÓRIA

Aos meus pais,

José Alberto da Silva e **Ivete Luiza Pachega da Silva**,

pelo incentivo, carinho, amor e compreensão.

AGRADECIMENTOS

Agradeço primeiramente a DEUS, pois está sempre presente em minha vida.

Aos meus pais José Alberto e Ivete, e à minha namorada Soellyn, que estiveram sempre presente, me dando força, amor e segurança.

Aos meus avôs paterno, Benedito e Olívia, e materno, Mário e Santina.

À minha orientadora Elisa Hatsue Moriya Huzita, pela sua dedicação e confiança em mim para a realização deste trabalho. Você é um exemplo para todos!!!

Aos professores do departamento de informática que contribuíram para minha formação, em especial a professora Tania Tait, que acompanhou a realização deste trabalho.

Agradeço aos amigos do grupo de pesquisa GESSD. Éderson Amorim, Igor Steinmacher, Igor Wiese, Flávio Schiavoni, Gustavo Sato, William Donegá, Lúcia Enami, Rogério Pozza, Ana Chaves, Camila Leal, Rodrigo Pagno, pela amizade, ensinamentos e dedicação.

Agradeço pelo companheirismo de todos os amigos do mestrado da turma de 2006 e aos amigos: Francisco Thesko, Fábio Zaupa, Carlos Henrique, Welligton Carvalho (Ton), Elison Lusvardi, André Schwerz, Rafael Liberato, André Pozza, Edson Oliveira.

Aos funcionários do departamento de informática que, de forma direta ou indireta, contribuíram com este trabalho e, em especial à Maria Inês Davanço, pelo apoio e amizade.

Ao Sr. Fernando Linschoten, Diretor Presidente da empresa ABRE – Agência Brasileira de Estágios, pela amizade e compreensão do meu desligamento da empresa, em razão do mestrado.

Ao CNPq – Conselho Nacional de Desenvolvimento Científico e Tecnológico pelo apoio financeiro.

RESUMO

A crescente complexidade das atividades a serem realizadas em um projeto de desenvolvimento de software exige maior interação entre os participantes de um projeto. Atualmente este tipo de projeto tem demandado que o desenvolvimento seja realizado por equipes distribuídas o que pode levar ao aumento de conflitos quando mais do que uma pessoa estiver envolvida em uma mesma atividade. Os artefatos resultantes de atividades realizadas durante o desenvolvimento de um projeto, constituem-se de programas, modelos e documentos textuais que precisam ser armazenados. Assim, é importante que esteja disponível, principalmente em um ambiente de desenvolvimento distribuído de software, um gerenciador de artefatos. Esta dissertação apresenta um gerenciador de artefatos que, além de permitir o gerenciamento de versões, suporta também, o tratamento de conflitos que ocorrerem em artefatos no formato XMI produzidos por ferramentas CASE.

Palavras-chave: gerenciamento de versões de artefatos, desenvolvimento distribuído de software, percepção.

ABSTRACT

Growing complexity of activities to be performed in a software development project demands further interaction among its participants. Such projects currently demand development by distributed teams, which may lead to increased conflicts when more than one individual is involved in the same task. Artefacts resulting of activities performed during project development include softwares, models and text documents that must be stored. Therefore, availability of an artefact manager is important, specially in a distributed software development environment. This work showcases an artefact manager that not only allows version management, but also supports treatment of conflicts which may occur in XMI format artefacts produced by CASE tools.

Key-words: *artefact management, software distributed development, awareness.*

Índice de Ilustrações

Figura 1: Ramificações de versão.....22
Figura 2: Sistema cliente/servidor.....26
Figura 3: O problema que deve ser evitado.....27
Figura 4: Solução travar-modificar-destravar.....28
Figura 5: Solução copiar-modificar-fundir (1).....30
Figura 6: Solução copiar-modificar-fundir (2).....31
Figura 7: Fragmento de um arquivo XMI.....33
Figura 8: Modelo 3C.....34
Figura 9: Arquitetura DiSEN.....39
Figura 10: FRADE - Camadas lógicas.....40
Figura 11: Camada de transporte – Odyssey-VCS.....44
Figura 12: Ferramenta Gconf.....45
Figura 13: Componentes do modelo de interoperabilidade.....51
Figura 14: Diagrama de caso de uso do Gerenciador de Artefatos.....55
Figura 15: Diagrama de Componentes.....57
Figura 16: Diagrama de classes do componente Alocação.....58
Figura 17: Diagrama de classes do componente Tratador Conflito.....60
Figura 18: Descritor artefatos locais.....62
Figura 19: Diagrama de classes do componente artefato.....62
Figura 20: Gerenciador de Artefato - DiSEN-SCV.....64
Figura 21: Modificações realizadas pelo primeiro desenvolvedor.....65
Figura 22: Modificações realizadas pelo segundo desenvolvedor.....66
Figura 23: Resultado da junção das versões.....66
Figura 24: Arquitetura do Subversion.....68
Figura 25: Um exemplo de localização de repositórios em ADDS.....69
Figura 26: Sequoia.....71

Figura 27: Ambiente para realização dos testes.....75
Figura 28: Estrutura para armazenamento de artefatos binários e XMI.....76
Figura 29: Estrutura para armazenamento de artefatos texto.....77
Figura 30: Artefato inicial para realização dos testes.....79
Figura 31: Artefato modificado pelo desenvolvedor (A).....80
Figura 32: Artefato modificado pelo desenvolvedor (B).....81
Figura 33: Merge entre as modificações dos desenvolvedores (A e B).....82
Figura 34: Visualização do conflito utilizando o NetBeans.....83

Índice de Tabelas

Tabela 1: Comparação do uso do Subversion e SGBD como repositório de artefatos...71

Tabela 2: Comparativo entre os trabalhos relacionados.....72

LISTA DE SIGLAS

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
ADDS	Ambiente de Desenvolvimento Distribuído de Software
ADS	Ambiente de Desenvolvimento de Software
ADSOrg	Ambiente de Desenvolvimento de Software Orientado a Organização
CASE	*Computer-Aided Software Engineering*
CSCW	*Computer Supported Cooperative Work*
CVS	*Concurrent Versions System*
DDS	Desenvolvimento Distribuído de Software
DiSEN	*Distributed Software Engineering Enviroment*
DS	Desenvolvimento de Software
FIPA	*Foundation for Intelligent Physical Agents*
FRADE	*Framework to Infrastructure of Distributed Software Development Environment*
GC	Gerência de Configuração
GCS	Gerenciamento de Configuração de Software
IC	Item de Configuração
MDSODI	Metodologia para Desenvolvimento de Software Distribuído
MOF	*Meta-Object Facility*
OLAP	*OnLine Analytical Processing*
OMG	*Object Management Group*
SCV	Sistema de Controle de Versão
SGBD	Sistema de Gerenciamento de Banco de Dados
UML	*Unified Modeling Language*
VID	*Version Identifier*
XMI	*XML Metadata Interchange*
XML	*Extensible Markup Language*

Sumário

Capítulo 1 - Introdução....13
1.1 Preâmbulo....13
1.2 Motivação....14
1.3 Problema....15
1.4 Contexto....17
1.5 Objetivo....18
1.6 Organização....18
Capítulo 2 - Fundamentação Teórica....20
2.1 Conceitos de Controle de Versões....20
2.2 Gerência de Configuração de Software....22
2.3 Compartilhamento de Artefatos - Problema....24
2.4 Sistema de Controle de Versão....25
2.4.1 Compartilhando Artefatos....27
2.4.2 A Solução Travar-Modificar-Destravar....28
2.4.3 A Solução Copiar-Modificar-Fundir....29
2.5 XML Metadata Interchange (XMI)....32
2.6 CSCW e Percepção....33
2.7 Desenvolvimento Distribuído de Software (DDS)....37
2.8 Arquitetura do DiSEN....38
Capítulo 3 - Trabalhos Relacionados....42
3.1 Ferramentas para Apoio à Percepção....42
3.2 Sistema de Controle de Versões (SCV)....43
3.3 MAIS (Multi-synchronous Awareness InfraStructure)....46
3.4 Augur: Combina Informações de Artefatos e Atividades....46
3.5 MIMIX....47
3.6 ARIANE....47

3.7 Odyssey-CCS.....49
3.8 IMART.....50
3.9 Considerações Finais.....52
Capítulo 4 - DiSEN-SCV.....53
4.1 Funcionalidades.....54
4.2 Arquitetura do DiSEN-SCV.....57
4.3 Implementação.....63
4.4 Comparação com os trabalhos relacionados.....72
4.5 Avaliação.....74
Capítulo 5 - Conclusões.....84
Referências Bibliográficas.....89

Capítulo 1 - Introdução

1.1 Preâmbulo

Os projetos de desenvolvimento de software têm, progressivamente, aumentado de tamanho e complexidade, sendo cada vez mais comum sua realização por equipes de médio porte (entre dez e vinte desenvolvedores) e grande porte (acima de vinte desenvolvedores). Com as facilidades de comunicação proporcionadas pela Internet, a necessidade de experiência em diversas áreas de conhecimento e a pressão por cronogramas mais restritos, fazem com que alguns projetos sejam desenvolvidos por diversas equipes trabalhando cooperativamente. Além disso, as dificuldades em reunir os especialistas necessários em um mesmo local físico e a delegação do desenvolvimento de determinados componentes para outras empresas, são exemplos de fatores que podem exigir que as equipes participantes de um projeto estejam geograficamente distribuídas (MAIR, 1997). As organizações têm cada vez mais utilizado o desenvolvimento de software remoto como uma facilidade adicional, levando ao que é conhecido como Desenvolvimento Distribuído de Software (DDS) (KIEL, 2003); (HERBSLEB e MOITRA, 2001).

A existência de mecanismos eficientes de comunicação, como a Internet, não é suficiente para solucionar os problemas de desenvolvimento cooperativo de projetos de software. Ao contrário, o novo cenário traz novos problemas para o processo de desenvolvimento. Por exemplo, o trabalho de equipes geograficamente distribuídas dificulta o controle de alterações nos componentes de um projeto em desenvolvimento (PRESSMAN, 2001). Mesmo quando precedida de uma definição precisa das interfaces entre os componentes, a realização de um projeto pode exigir que diversos

desenvolvedores alterem, simultaneamente, os mesmos componentes. Essa situação exige a adoção de políticas adequadas para manter a consistência entre as versões do componente do projeto ou permitir que essa consistência seja posteriormente restituída.

O desenvolvimento de software deve tratar de duas fontes de complexidade: a complexidade do artefato que está sendo produzido e a complexidade das atividades em torno desse artefato. Um artefato é resultado de uma atividade e pode ser utilizado posteriormente como matéria-prima para aquela ou para outra atividade a fim de gerar novos artefatos. Dessa forma, uma atividade pode consumir artefatos (de entrada) e gerar novos artefatos (de saída). Artefatos são freqüentemente persistentes, armazenados em repositórios e possuem versões (PASCUTTI, 2002). Como exemplo de artefato pode-se citar documentos, manuais, código fonte, etc..

Modelos de processo de software tentam ajudar as equipes com a complexidade das atividades, enquanto técnicas, análise e testes focam no artefato (FROEHLICH e DOURICH, 2004). O trabalho cooperativo e a utilização de técnicas de *awareness* (percepção) ajudam a diminuir as dificuldades para a realização de atividades que são executadas por mais de uma pessoa. Desenvolvedores pertencentes à equipe de projeto podem estar dispersos geograficamente, apoiados por decisões estratégicas, motivados por redução de custos e aumento de produtividade. Assim, é desejável que as atividades sejam apoiadas por algum software *(groupware)*, que permita minimizar o retrabalho, a fim de manter a consistência e uniformidade dos artefatos produzidos.

1.2 Motivação

Com base no cenário de Desenvolvimento Distribuído de Software (DDS),

pode existir uma ou mais equipes pertencentes a mesma organização e duas ou mais organizações podem se reunir para a realização de um projeto. Além disso, as equipes podem estar geograficamente dispersas e trabalharem em horários diferentes.

Durante o processo de Desenvolvimento de Software (DS), os membros das equipes produzem artefatos que podem ser compartilhados com membros de outras equipes. Dependendo da seqüência da realização das atividades, os artefatos produzidos por uma equipe podem servir como artefatos de entrada para a realização de uma atividade posterior.

Devido à possibilidade de um artefato ser produzido por mais de uma pessoa, independentemente de qual equipe ela pertença, há a necessidade de manter o controle das versões do artefato. Esse controle pode ser obtido pelo uso de técnicas de Gerência de Configuração de Software[1] (GCS).

A utilização de GCS é fundamental para manter o controle sobre os artefatos produzidos e/ou modificados durante o processo de desenvolvimento do software. A sua utilização não se restringe somente à fase de implementação. Os sistemas de GCS podem, por exemplo, auxiliar no acompanhamento da realização das atividades (IEEE, 1987; LEON, 2000; CHRISSIS et al., 2003; SOFTEX, 2006) apud (MURTA et al., 2007).

1.3 Problema

Durante o desenvolvimento de software é inevitável que alguns artefatos sofram modificações e conseqüentemente tenham várias versões. O gerenciamento das modificações efetuadas em artefatos é um fator importante quando estes são compartilhados por várias pessoas em um Ambiente de Desenvolvimento Distribuído

1 Gerência de Configuração de Software é uma disciplina para o controle da evolução de sistemas de software (DART, 1991)

de Software (ADDS). Em Pascutti (2002) é proposto um gerenciador de artefatos, que é responsável pelo controle de versões dos artefatos que são gerados durante a realização das atividades. No entanto, mesmo as atividades sendo executadas cooperativamente, existem momentos em que os desenvolvedores necessitam trabalhar individualmente. Portanto, é interessante que cada desenvolvedor tenha uma cópia do artefato compartilhado e, de tempos em tempos, sincronizem estas cópias. Dependendo do tempo em que um desenvolvedor fica sem sincronizar sua cópia, esta pode estar divergente das demais, acarretando em esforço para convergência.

Um mecanismo para reduzir esse isolamento entre os desenvolvedores é a percepção de mudanças entre as cópias do artefato. A informação de mudança deve ser disponibilizada para os desenvolvedores de forma a agilizar a detecção de possíveis conflitos, permitindo acompanhar a evolução das versões do artefato, tornando possível a tomada de decisões baseadas nas mudanças realizadas.

Um outro problema é em relação a alocação de artefatos. Um desenvolvedor pode alocar um artefato e, a partir desse momento, nenhum outro desenvolvedor poderá submeter uma nova versão do artefato alocado. A desalocação do artefato é realizada quando o desenvolvedor, que está com o artefato alocado, submete uma nova versão ou desaloca-o por vontade própria. Dessa maneira, os desenvolvedores que trabalham na mesma atividade e não possuem a alocação do artefato, deverão aguardar, até que o artefato seja desalocado, para submeter uma nova versão. Assim, diminui o paralelismo na execução da atividade.

Também podem ocorrer conflitos entre as versões dos artefatos. Os conflitos em artefatos no formato XMI são produzidos por ferramentas CASE, por isso são mais difíceis de serem resolvidos, se for comparado com os artefatos no formato texto (por exemplo, artefatos de código-fonte).

Esses problemas serão discutidos com mais detalhes no Capítulo 4 e também serão apresentadas soluções para esses problemas.

1.4 Contexto

Este trabalho de pesquisa está inserido no contexto do Projeto DiSEN (*Distributed Software Engineering ENvironment*), que visa prover um ambiente distribuído de desenvolvimento de software que considera as características de sistemas distribuídos e fornece suporte à cooperação entre espaços de trabalho distintos, além de apoiar um conjunto de ferramentas que auxiliam na tarefa do desenvolvimento de software.

O DiSEN se encontra em desenvolvimento pelo Grupo de Estudos de Engenharia de Software Distribuído, da Universidade Estadual de Maringá. Ele também incorpora a tecnologia de agentes segundo o padrão da FIPA (*Foundation for Intelligent Physical Agents)*. Segundo Pascutti (2002), a arquitetura do DiSEN foi projetada para utilizar, dentre outras, a MDSODI (HUZITA, 1999), uma metodologia para desenvolvimento de software que leva em consideração algumas características identificadas em sistemas distribuídos, tais como concorrência, paralelismo, comunicação, sincronização e distribuição. A MDSODI também mantém as principais características do *Unified Process* (JACOBSON, BOOCH, RUMBAUGH, 1999), dirigida a *use-case*, centrada na arquitetura e desenvolvimento iterativo e incremental.

O objetivo do DiSEN é fornecer o suporte necessário para o desenvolvimento distribuído de software; a equipe poderá estar distribuída em locais geográficos distintos e trabalhar de forma cooperativa com uma metodologia para desenvolvimento distribuído de software.

1.5 Objetivo

O objetivo geral desta dissertação é desenvolver um gerenciador de artefatos para o ambiente DiSEN. Assim, os objetivos específicos são:

- Implementar as funcionalidades para o gerenciador de artefatos.
- Desenvolver um mecanismo para gerenciar conflitos entre duas versões de um artefato.
- Implementar técnicas de percepção para auxiliar o desenvolvedor durante a realização de suas atividades.
- Implementar a alocação de artefatos, entre os membros da equipe, por um período de tempo.

Nesta dissertação serão tratados os problemas referentes à execução das atividades e dos artefatos gerados por elas. Embora interessante, os problemas de relacionamento entre os membros do grupo, estão além do contexto desta dissertação e, serão, tratados em trabalhos futuros.

1.6 Organização

Esta dissertação está organizada da seguinte maneira:

O Capítulo 1 contextualiza este trabalho, destacando a motivação, os problemas e os objetivos desta pesquisa.

O Capítulo 2 descreve os conceitos utilizados para fundamentar esta pesquisa, envolvendo controle de versões, gerência de configuração de software, desenvolvimento distribuído de software e outros conceitos relacionados.

No Capítulo 3, são apresentados os trabalhos encontrados na literatura que se referem aos conceitos aplicados neste trabalho e que, por esse motivo, influenciaram

de alguma forma em seu desenvolvimento.

O Capítulo 4 detalha a pesquisa realizada, apresentando as características do Gerenciador de Artefatos, sua arquitetura, funcionalidades e avaliação.

No Capítulo 5 são apresentadas as considerações finais, conclusões e trabalhos futuros.

Capítulo 2 - Fundamentação Teórica

Para fundamentar este trabalho, alguns conceitos foram estudados e aplicados no desenvolvimento da pesquisa. Este capítulo apresenta esses conceitos, envolvendo controle de versões, gerência de configuração de software, CSCW – *Computer Supported Coopertive Work,* percepção e desenvolvimento distribuído de software.

2.1 Conceitos de Controle de Versões

Durante o desenvolvimento de um projeto de software é comum que ocorram mudanças para correção ou melhoria de suas funcionalidades. Conseqüentemente, há necessidade de manter uma cópia de tudo que é modificado e essa cópia deve ser armazenada e gerenciada de tal maneira que possa ser recuperada, se necessária (SOARES, 2000).

Portanto, esta seção apresenta alguns termos importantes relacionados ao controle de versões. Uma descrição detalhada pode ser encontrada em (CONRADI e WESTFECHTEL, 1998) e (ESTUBLIER et al., 2002).

Uma **versão** v_i representa um estado de um **Item de Configuração** (IC) que está sendo modificado. IC pode significar, por exemplo, arquivos, diretórios, objetos armazenados em base de dados orientada a objetos, entidades, relacionamentos, etc. (SOARES, 2000) (OLIVEIRA, 2005).

Cada versão de um IC deve possuir um identificador de versão (VID – *version identifier*) único. O controle de versão sobre um IC pode ser aplicado em qualquer nível de granularidade, abrangendo desde o produto de software até as linhas de texto de um determinado arquivo (SOARES, 2000). Em modelos UML a granularidade

também pode ser definida em: classes, atributos, métodos, *use-case*, etc. Nesta dissertação, o IC é chamado de artefato.

Uma **configuração** é composta de versões de documento de requisitos, da arquitetura do software, dos modelos de análise e projeto, do código fonte, etc. (OLIVEIRA, 2005).

O conceito de *delta* foi proposto para solucionar o problema de escassez de memória nas décadas de 70 e 80. Duas versões sucessivas são similares (em média 98%), portanto, somente os 2% diferentes são armazenados (CONRADI e WESTFECHTEL, 1998) (ESTUBLIER, 2000).

Existem duas técnicas para o armazenamento de *delta:* diferença para frente (*forward delta*) e diferença para trás (*reverse delta*) (TICHY, 1985) (LEON, 2000) (MURTA, 2007).

Na técnica de *delta* para frente é armazenada a versão mais antiga do IC e as diferenças para as versões posteriores. A técnica de *delta* para trás armazena a versão mais recente do IC e as diferenças para as versões anteriores.

As versões são classificadas de acordo com sua evolução como: **revisões** ou **variantes**. Revisões são versões que resultam de correção de defeitos ou implementação de uma nova funcionalidade e evoluem seqüencialmente. Variantes são versões paralelas ou alternativas, não substituem as anteriores e são usadas concorrentemente em configurações alternativas (OLIVEIRA, 2005).

A Figura 1 simula a evolução das versões de um artefato. Também é possível destacar versões de **revisões** e **variantes**. A versão 5 substitui a versão 4, por isso pode-se dizer que a versão 5 é uma **revisão** da versão 4. Já a versão 3.1 é uma versão alternativa à versão 3, por isso pode-se dizer que a versão 3.1 é uma **variante** da versão 3 (ESTUBLIER et al., 2002).

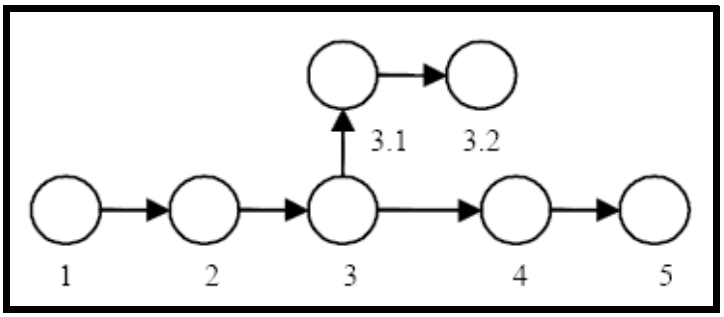

Figura 1: Ramificações de versão
Fonte: (ESTUBLIER et al., 2002)

Espaço de trabalho é a área onde o desenvolvedor realiza modificações sobre os artefatos isoladamente (OLIVEIRA, 2005).

Check-out é o processo de solicitação, aprovação e cópia dos artefatos do repositório para o espaço do trabalho de desenvolvedor (LEON, 2000).

Check-in é o processo de revisão, aprovação e cópia dos artefatos do espaço de trabalho do desenvolvedor para o repositório (LEON, 2000).

2.2 Gerência de Configuração de Software

Devido à necessidade dos norte-americanos de controlar as modificações na documentação de produção de aviões de guerra na década de 50, surgiu a Gerência de Configuração (GC) (LEON, 2000; HASS, 2003; ESTUBLIER et al., 2005). A Gerência de Configuração de Software (GCS) surgiu nas décadas de 60 e 70 quando a GC passou a abranger os artefatos de software (CHRISTENSEN e THAYER, 2002; MURTA, 2007).

Segundo Estublier et al. (2002) a GCS é um importante campo da engenharia de software e vem sendo adotada de forma mais ampla a cada dia. A GCS apóia o desenvolvimento do projeto de software durante toda sua execução, pois mudanças

podem ocorrer a qualquer momento. Suas atividades identificam e controlam as mudanças, fazem auditoria e relatam as modificações para garantir que as mesmas sejam implementadas corretamente e sejam reportadas às pessoas de interesse (PRESSMAN, 2001) (FIGUEIREDO, SANTOS e ROCHA, 2004).

Durante o desenvolvimento do projeto de software, grande quantidade de artefatos são produzidos. Vários desses artefatos sofrerão alterações ao longo do projeto, em razão de mudanças nos requisitos, alteração na legislação ou correção de defeitos. Portanto, para evitar perda do controle do projeto, é necessário que essas alterações sejam controladas e gerenciadas (FIGUEIREDO, SANTOS e ROCHA, 2004).

Esta dissertação utiliza a definição de GCS presente em (IEEE, 1990 apud MURTA, 2007):

> "uma disciplina que aplica procedimentos técnicos e administrativos para identificar e documentar as características físicas e funcionais de um Item de Configuração (IC), controlar as alterações nessas características, armazenar e relatar o processamento das modificações e o estágio da implementação e verificar a compatibilidade com os requisitos especificados".

Portanto, a GCS auxilia no controle e acompanhamento do desenvolvimento e não se propõe a definir quando e como as atividades do processo de desenvolvimento deverão ser realizadas.

Na fase de desenvolvimento a GCS é dividida em três sistemas principais, que são: controle de modificações, controle de versões e gerenciamento de construção (IEEE, 1990 apud MURTA, 2007).

- **Controle de modificações** - responsável por: executar a função que

controla a evolução dos artefatos, armazenar as informações geradas durante a realização das atividades e notificar os participantes interessados e autorizados.

- **Controle de versões:** controla a evolução disciplinada dos artefatos de forma distribuída e concorrente.
- **Gerenciamento de construção:** transforma os diversos artefatos de software de um projeto em um sistema executável de maneira automatizada.

2.3 Compartilhamento de Artefatos - Problema

Para o armazenamento e controle das versões de um artefato que são produzidas na execução das atividades do projeto é utilizado um repositório. Qualquer membro da equipe, que tenha permissão de acesso ao repositório pode requisitar, modificar e submeter um artefato.

O problema pode surgir quando os artefatos são modificados e enviados novamente para o repositório, pois neste momento pode ocorrer um conflito. Por exemplo, dois membros da equipe que estão trabalhando em uma mesma atividade, fazem a solicitação do mesmo artefato no repositório, que está na primeira versão. Eles estão trabalhando individualmente, cada um fazendo suas modificações na cópia do artefato. Após as modificações, eles deverão submeter o artefato para o repositório. O primeiro que submeter o artefato para o repositório não terá problema e uma segunda versão será gerada. Quando o segundo membro for submeter o artefato, ele não conseguirá, gerando assim um conflito, pois a versão que ele tinha do artefato está diferente daquela que está no repositório atualmente.

Segundo Borghoff e Schlichter (2000), existem no mínimo duas formas de resolver esse problema. A coordenação das ações de cada membro do grupo pode ser realizada de forma pessimista ou otimista. Na forma pessimista, assume-se que as

ações serão incoerentes e atingirão um estado inválido e indesejado. Neste caso, a ação de um membro deve bloquear as ações dos demais sobre o mesmo artefato.

Na forma otimista, assume-se que as ações serão coerentes, mesmo ocorrendo de forma isolada, e o resultado final será um estado válido. Neste caso, mais de um membro realiza ações sobre um mesmo artefato. O membro da equipe, que desejar trabalhar isoladamente, deverá solicitar o bloqueio sobre um determinado artefato. No entanto, o uso de bloqueios impossibilita o paralelismo de atividades que concorrem pelo artefato bloqueado. Em geral, os membros adotam a estratégia de geração de cópias do artefato, que são trabalhadas em paralelo e, posteriormente, consolidadas em uma única versão. Assim, por um lado o bloqueio evita a ocorrência de conflitos, mas reduz o paralelismo no trabalho, enquanto que o uso de cópias apenas posterga a detecção dos conflitos, mas possibilita um maior paralelismo de atividades.

A seção seguinte apresenta com mais detalhes esse problema e a solução apresentada pelo repositório de artefatos Subversion[2].

2.4 Sistema de Controle de Versão

De maneira geral, os sistemas de controle de versão compartilham as informações de maneira centralizada. Todo cliente autorizado, pode ler e gravar informações no repositório. A partir do momento que as informações são armazenadas, qualquer cliente autorizado pode recebê-las (Figura 3) (COLLINS-SUSSMAN, FITZPATRICK e PILATO, 2004).

As informações armazenadas no repositório, nesta dissertação, são chamadas de artefatos.

2 Version Control System. Disponível em: <http://subversion.tigris.org>

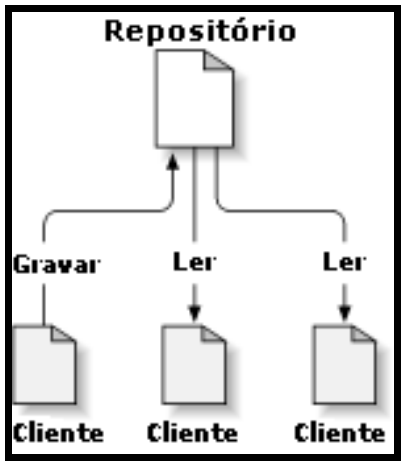

Figura 2: Sistema cliente/servidor
Fonte: Adaptado de COLLINS-SUSSMAN, FITZPATRICK e PILATO (2004)

No repositório, também são armazenadas as informações referentes a qualquer modificação, exclusão ou inclusão de um artefato que estiver no repositório. Essas informações são sobre quem efetuou, quando, o que foi modificado e porque foi modificado.

Quando um cliente (Figura 2) solicita os artefatos do repositório, ele recebe apenas a última versão. Entretanto, é possível recuperar versões anteriores e verificar quais foram as modificações e quem as efetuou.

O principal objetivo de um Sistema de Controle de Versão (SCV) é permitir o compartilhamento e a edição por várias pessoas. Para que o objetivo seja atingido, cada SCV possui suas soluções (COLLINS-SUSSMAN, FITZPATRICK e PILATO, 2004). O Subversion, por exemplo, apresenta duas soluções: **Travar-Modificar-Destravar** e **Cópiar-Modificar-Fundir**, que serão apresentadas, respectivamente, nas subseções 2.4.2 e 2.4.3.

A seguir é apresentado o problema que pode ocorrer quando se deseja compartilhar artefatos.

2.4.1 Compartilhando Artefatos

Como permitir que os usuários modifiquem os artefatos, garantindo-lhes que não se atrapalhem mutuamente? Não é raro um usuário sobrescrever, acidentalmente, uma alteração realizada por outro. A Figura 3 representa esse problema.

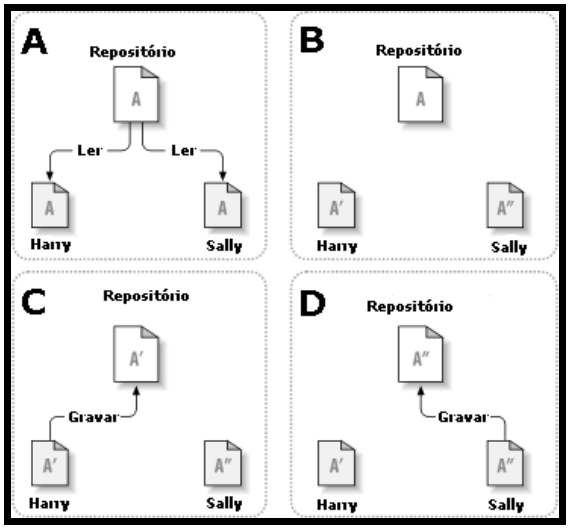

Figura 3: O problema que deve ser evitado
Fonte: Adaptado de COLLINS-SUSSMAN, FITZPATRICK e PILATO (2004)

A Figura 3-A mostra os usuários Harry e Sally copiando do repositório o mesmo artefato. Após o artefato copiado, eles podem realizar suas modificações (Figura 3-B). Harry envia suas modificações ao repositório primeiro do que Sally (Figura 3-C). Ao terminar de realizar suas modificações, Sally também as envia ao repositório. Nesse momento, as modificações realizadas por Harry são sobrepostas pelas modificações de Sally, porque elas não existiam quando ele obteve a cópia do artefato.

As modificações realizadas por Harry não são perdidas para sempre, apenas não constam na última versão, porque a cada momento que um usuário envia um artefato modificado, o SCV gera uma nova versão para o mesmo.

As seções 2.4.2 e 2.4.3 apresentam, respectivamente, duas soluções para evitar esse problema: Travar-Modificar-Destravar e Copiar-Modificar-Fundir.

2.4.2 A Solução Travar-Modificar-Destravar

Essa solução permite que o artefato seja modificado apenas por um usuário de cada vez. Para que essa ação seja garantida são utilizados bloqueios. Antes que um usuário comece a modificar um artefato, ele deve bloqueá-lo (Figura 4-A). A partir do momento em que o artefato está bloqueado para um usuário, nenhum outro usuário conseguirá bloquear e nem realizar modificações no artefato (Figura 4-B). Os usuários que não obtiverem o bloqueio poderão apenas receber o artefato como leitura e esperar que o usuário que obteve o bloqueio, libere-o. No momento em que o usuário (que possui o bloqueio) enviar suas modificações ao repositório, automaticamente o artefato é desbloqueado (Figura 4-C). Após a liberação do bloqueio, qualquer usuário pode bloquear o artefato para modificá-lo (Figura 4-D) (COLLINS-SUSSMAN, FITZPATRICK e PILATO, 2004).

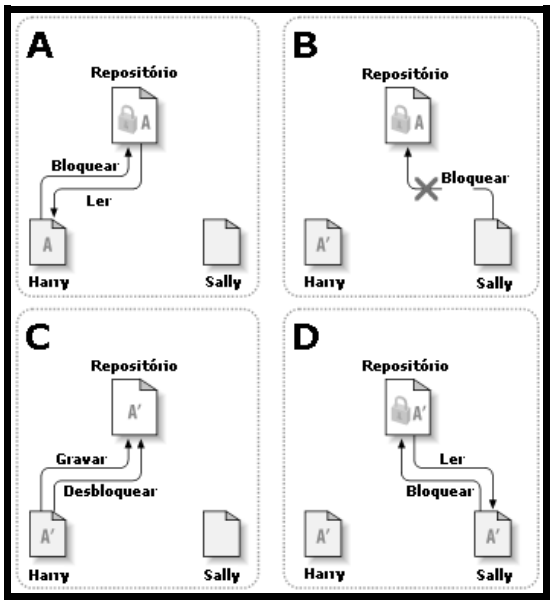

Figura 4: Solução travar-modificar-destravar
Fonte: Adaptado de COLLINS-SUSSMAN, FITZPATRICK e PILATO (2004)

Esse tipo de solução restringe que dois ou mais usuários realizem, ao mesmo tempo, modificações no mesmo artefato. Algumas situações que podem ocorrer com o uso dessa solução (COLLINS-SUSSMAN, FITZPATRICK e PILATO, 2004):

Problemas administrativos - Se um usuário obtiver o bloqueio sobre um artefato e esquecer de liberá-lo, os demais usuários terão que aguardar até que o artefato seja liberado ou terá que contactar o administrador do sistema para liberá-lo. Essa situação causa atraso desnecessário e perda de tempo.

Bloqueio desnecessário - Com o uso do bloqueio, dois usuários não podem modificar o mesmo arquivo simultaneamente, mesmo que façam modificações em lugares distintos. Assumindo que as mudanças sejam corretamente fundidas, os usuários poderiam realizá-las simultaneamente.

Bloqueio de artefato com dependência - O usuário Harry bloqueia e modifica o artefato "A", enquanto o usuário Sally bloqueia e modifica o artefato "B". Os artefatos "A" e "B" são dependentes. Pode ocorrer que as mudanças realizadas em cada um dos artefatos, no final, sejam incompatíveis. Assim, o bloqueio pode passar uma falsa impressão de segurança. Portanto, além dos usuários se preocuparem em bloquear os artefatos, eles devem se comunicar e discutir sobre as modificações.

2.4.3 A Solução Copiar-Modificar-Fundir

O modelo Copiar-Modificar-Fundir é utilizado por sistemas de controle de versão como o CVS[3] (*Concurrent Versions System*) e Subversion. Esse modelo é uma alternativa ao sistema Travar-Modificar-Destravar.

Nesse modelo, cada usuário cria uma cópia do repositório de artefatos em seu computador. Dessa maneira, os usuários podem modificá-las simultaneamente e, após

3 Concurrent Version System. Disponível em: <http://www.cvshome.org>

as modificações, as cópias serão fundidas e uma nova versão do artefato no repositório é gerada. O responsável pela correta fusão é o usuário, o SCV apenas auxilia (COLLINS-SUSSMAN, FITZPATRICK e PILATO, 2004).

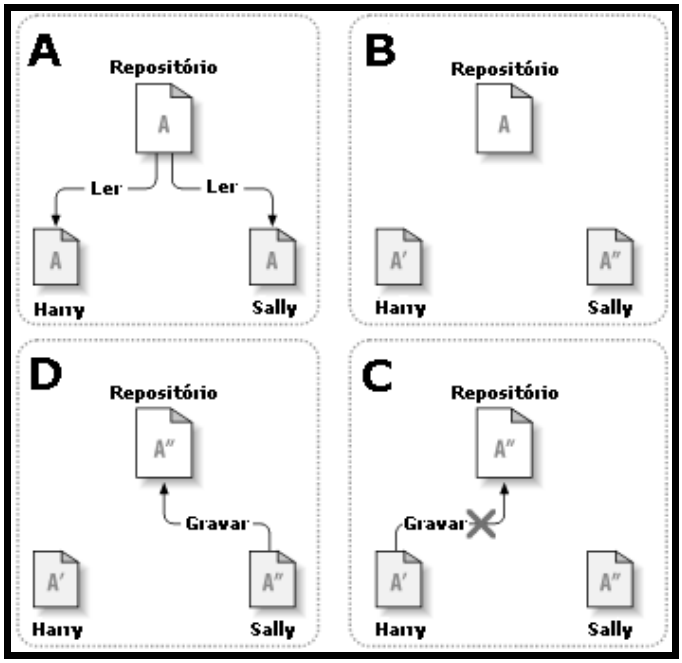

Figura 5: Solução copiar-modificar-fundir (1)
Fonte: Adaptado de COLLINS-SUSSMAN, FITZPATRICK e PILATO (2004)

A Figura 5 mostra um exemplo onde Harry e Sally fazem a cópia dos artefatos do mesmo projeto, que estão no repositório (Figura 5-A). Os dois modificam em suas cópias o mesmo artefato (Figura 5-B). Sally envia primeiro suas modificações ao repositório (Figura 5-C). Posteriormente, quando Harry tentar enviar suas modificações, será informado que o artefato está desatualizado, ou seja, depois de ter feito a cópia do artefato, o mesmo foi modificado por outra pessoa, no repositório (Figura 5-D).

Assim, Harry deverá comparar a nova versão do artefato, que está no repositório, com a sua cópia de trabalho (Figura 6-A). Nesse momento, em seu espaço de trabalho, as modificações realizadas por Harry foram fundidas com a nova versão

do artefato (Figura 6-B). Após a fusão, Harry envia o artefato com suas modificações para o repositório (Figura 6-C). Agora, ambos usuários possuem o artefato com todas as modificações (Figura 6-D).

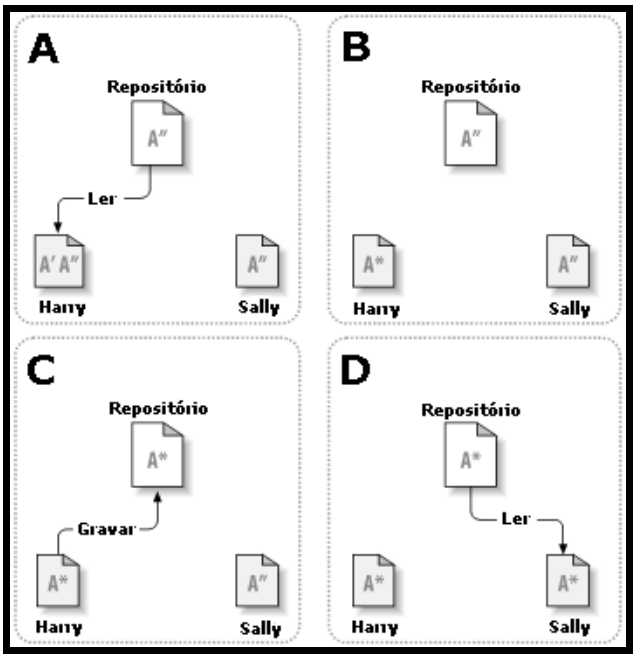

Figura 6: Solução copiar-modificar-fundir (2)
Fonte: Adaptado de COLLINS-SUSSMAN, FITZPATRICK e PILATO (2004)

Uma situação de conflito pode ocorrer se as mudanças realizadas por Harry sobrescreverem as modificações de Sally. Quando um usuário solicita que suas modificações sejam fundidas com as modificações de outro usuário, a sua cópia é marcada em estado de conflito. Então, Harry poderá comparar suas modificações com as modificações de Sally e escolher manualmente entre elas. Vale ressaltar que em uma situação de conflito é importante que os respectivos usuários conversem e cheguem a uma solução. Após resolver o conflito, o artefato pode ser salvo com segurança no repositório (COLLINS-SUSSMAN, FITZPATRICK e PILATO, 2004).

A vantagem do modelo copiar-modificar-fundir é que os usuários podem trabalhar simultaneamente no mesmo artefato. Uma desvantagem é o tempo gasto para

resolver os conflitos entre artefatos que possuam várias dependências e/ou que possuam código muito complexo.

O modelo copiar-modificar-fundir é baseado em artefatos do tipo texto (por exemplo, arquivo de código fonte), que são possíveis de realizar a fusão. Entretanto, em arquivos no formato binário (por exemplo, arquivo de som e imagem), não é possível realizar a fusão. Portanto, nesse caso, o modelo travar-modificar-destravar é o mais indicado.

2.5 *XML Metadata Interchange* (XMI)

A linguagem eX*tensible Markup Language* (XML) tem sido utilizada por várias aplicações, entre as quais em ferramentas que dão apoio ao desenvolvimento de software. Neste caso, o formato XMI está sendo utilizado para expressar o conteúdo de modelos orientados a objetos (OO) e outros documentos gerados por essa linguagem, em um formato que permita a troca de informações.

Segundo BRODSKY (1999), não existe uma ferramenta que apóie sozinha todo o processo de desenvolvimento do software, o que torna necessário utilizar-se de ferramentas diferentes para cada uma das fases do desenvolvimento do software. Fazer com que estas ferramentas troquem informações é um grande desafio. O padrão XMI da OMG permite representar modelos UML em XML. Dessa forma, é possível obter uma representação XML de um artefato criado por uma ferramenta que ofereça esse apoio (importação/exportação de XMI).

A Figura 7 mostra um fragmento de um arquivo XMI, que representa um diagrama de caso de uso gerado por uma ferramenta CASE.

```
<UML:Model name="Use Case Model" xmi.id="MX_EAID_E77A5D80_0EE2_4dfc_99C4_C8E98BA4992F" isSp
    <UML:Namespace.ownedElement>
        <UML:Actor name="Actor2" xmi.id="EAID_35E3E7B1_491B_41d2_8A7D_FD15F0D5A9BD" isSpe
        <UML:Actor name="Actor1" xmi.id="EAID_821FB01E_4C84_4f3c_8892_87D1A4956477" isSpe
        <UML:UseCase name="Use Case5" xmi.id="EAID_0B541C34_A031_48d9_B9B4_7A19141B4036"
            <UML:UseCase.extend>
                <UML:Extend xmi.idref="EAID_EB47E770_AEB1_48a6_908D_E4275D1E0001"/>
            </UML:UseCase.extend>
        </UML:UseCase>
        <UML:UseCase name="Use Case1" xmi.id="EAID_101E7E17_5FDC_4c76_826D_B4F37199A547"
            <UML:UseCase.extend>
                <UML:Extend xmi.idref="EAID_EB47E770_AEB1_48a6_908D_E4275D1E0003"/>
            </UML:UseCase.extend>
            <UML:UseCase.include>
                <UML:Include xmi.idref="EAID_669A181D_4BE0_4bf0_A711_805B08D50005"/>
            </UML:UseCase.include>
        </UML:UseCase>
```

Figura 7: Fragmento de um arquivo XMI
Fonte: WIESE (2006)

Dentre os elementos do XMI, destacam-se (OMG XMI, 2005):

- XMI: elemento raiz de um documento XMI;
- XMI.header: contém elementos que identificam o modelo, metamodelo e metametamodelo que formam o cabeçalho de um documento XMI;
- XMI.model: identifica o modelo ao qual pertencem os dados contidos no XMI;
- XMI.content: contém os metadados do modelo que foi construído na ferramenta CASE. Constitui o conteúdo do documento XMI.

Os elementos apresentados, representam as partes principais de um documento XMI. A utilização do padrão XMI é importante, pois atualmente tem sido utilizado em larga escala pela indústria de software; portanto, passa a ser imprescindível, uma vez que deve apoiar à importação/exportação de artefatos nesta estrutura sintático-semântica (WIESE, 2006).

2.6 CSCW e Percepção

Ao trabalhar em grupo, os indivíduos podem, potencialmente, produzir melhores resultados do que se atuassem individualmente. Em um grupo pode ocorrer a

complementação de capacidades, de conhecimentos e de esforços individuais, e a interação entre pessoas com entendimentos, pontos de vista e habilidades complementares. Ao argumentar suas idéias, os membros de um grupo obtém informação para identificar inconsistências e falhas em seu raciocínio e, juntos, podem buscar idéias, informações e referências para auxiliar na resolução dos problemas. Um grupo também tem mais capacidade de gerar alternativas, levantar as vantagens e desvantagens de cada uma, selecionar as viáveis e tomar decisões (FUKS et al., 2003).

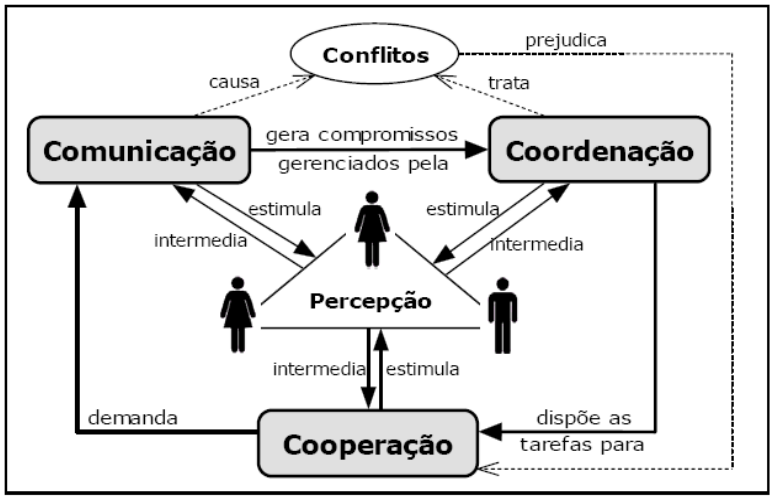

Figura 8: Modelo 3C
Fonte: FUKS et al. (2003)

O diagrama da Figura 8 é um refinamento do modelo 3C apresentado originalmente em (ELLIS et al., 1991) e difundido na literatura, como por exemplo em (BORGHOFF e SCHLICHTER, 2000).

Para colaborar, os indivíduos trocam informações (comunicação), organizam-se (coordenação) e operam em conjunto em um espaço compartilhado (cooperação). As trocas ocorridas durante a comunicação geram os compromissos gerenciados pela coordenação, que organiza e dispõe as atividades executadas na cooperação. Ao cooperar os indivíduos necessitam se comunicar para renegociar e decidir sobre situações não previstas inicialmente, mostrando o aspecto cíclico da colaboração.

Trabalho cooperativo é o trabalho que envolve duas ou mais pessoas, compartilhando informações, sem barreiras e com sinergia. Para haver um trabalho cooperativo é necessário que os indivíduos possuam habilidades para trabalho em grupo, base de dados organizada e padronizada, eficiência de comunicação, espírito de coletividade e bom relacionamento. (MORANDINI, 1998).

A área de *Computer Supported Cooperative Work* (CSCW) estuda a maneira como as pessoas trabalham em equipe e como o computador pode auxilia-los na realização das atividades (ELLIS et al., 1991; BORGES, 1995). Entretanto, o apoio fornecido pelo computador pode apresentar vários problemas. Um desses problemas, destacado por Pinheiro et al. (2001), é a falta de conhecimento do que cada membro da equipe esta fazendo ou onde o seu trabalho se encaixa com os demais.

Borges (1995) define esse contexto como um ponto importante em sistemas cooperativos, aplicando não somente ao conteúdo das contribuições individuais, mas também o seu significado para o grupo como um todo e seu objetivo.

A disseminação dessas informações entre os membros da equipe é chamado de percepção. Com a percepção, o indivíduo se informa sobre o que está acontecendo, o que as outras pessoas estão fazendo e adquire informações necessárias para seu trabalho (FUKS et al., 2003).

Percepção significa a compreensão do estado total do sistema, incluindo atividades passadas, status atual e opções futuras (ARAÚJO, 1997). Portanto, percepção refere-se a ter conhecimento das atividades do grupo, saber o que aconteceu, o que está acontecendo e/ou o que poderá acontecer.

Manter os usuários informados dentro da equipe é fundamental para a execução das atividades individuais e para a coordenação do projeto como um todo. A percepção é uma forma de auxiliar nessa tarefa (GUTWIN e GREENBERG, 2002).

Ao compartilhar artefatos na realização de uma atividade, os membros da equipe precisam conhecer as mudanças realizadas pelos outros, para compreender o

que pode afetar sobre o seu trabalho.

É importante que cada membro da equipe conheça o andamento das atividades dos outros participantes, para que consiga medir a qualidade do seu próprio trabalho em relação aos objetivos e progressos da equipe (DOURISH e BELLOTTI, 1992). O conhecimento sobre o que cada membro da equipe está realizando, evita a duplicidade na realização de uma atividade e reduz os conflitos. Segundo Pinheiro et al. (2001), a falta desse conhecimento gera diversos problemas como, redundância na realização das atividades, inconsistências e contradições, compromete a qualidade e produtividade do projeto e pode até não atingir os objetivos.

Segundo Dourish e Bellotti (1992), basicamente existem dois tipos de percepção: ativa e passiva. A **percepção ativa** é quando as informações são trocadas entre os membros da equipe por meio de reuniões presenciais, conversa nos corredores, por e-mail ou por troca de relatórios.

A maneira que é realizada a troca de informações entre os membros da equipe está sujeita a falhas e esquecimento. Ao final de uma sessão de trabalho, que pode durar até um dia inteiro, é difícil lembrar tudo o que foi realizado. Decidir qual informação é importante ou não, varia muito de pessoa para pessoa. Conseqüentemente, podem existir desentendimentos entre os membros da equipe, trabalhos duplicados ou perda de rendimento (SANTOS, 2003).

Na **percepção passiva** não existe esforço entre os membros da equipe para informar sobre o andamento de suas atividades. Esse tipo de percepção pode ocorrer quando uma pessoa escuta uma conversa por acaso e fica sabendo do que está acontecendo, ou quando a pessoa é informada por notificações, que são enviadas por algum tipo de monitoramento.

Santos (2003) explica que essa forma de percepção é mais difícil de ser alcançada e um meio para provê-la é com a manipulação dos artefatos compartilhados, permitindo que os membros da equipe consultem o histórico para compreender as

mudanças realizadas.

2.7 Desenvolvimento Distribuído de Software (DDS)

Ao longo dos anos, o software se tornou um componente vital da maioria dos negócios. O sucesso das organizações depende da utilização do software como uma arma competitiva. Na década de 80, muitas organizações começaram a experimentar o desenvolvimento de software remoto como uma facilidade a mais (HERBSLEB e MOITRA, 2001).

A crescente globalização do ambiente de negócios e da economia tem afetado diretamente o mercado de desenvolvimento de software. Os engenheiros de software vêm reconhecendo há algum tempo a profunda influência do desenvolvimento global de software na globalização dos negócios e, através de reações alarmistas, estão se movimentando para encontrar um modelo de negócio que possa atender a este mercado. Recentemente, a atenção está se voltando para o entendimento dos fatores que permitem às multinacionais e às corporações virtuais operar com sucesso ultrapassando as fronteiras geográficas e culturais, em busca de vantagens competitivas como baixos custos, maior produtividade ou qualidade na área de desenvolvimento de sistemas (HERBSLEB e MOITRA, 2001) (FREITAS et al., 2004).

Este fenômeno é alimentado por fatores, tais como, o acesso a grande quantidade de mão de obra especializada, redução nos custos do desenvolvimento, presença global e proximidade ao consumidor. Apesar do sucesso de várias equipes globais, as pesquisas revelam que a distância contribui para aumentar a complexidade nos processos organizacionais. Primeiramente, os processos de comunicação e coordenação são afetados pela distância, com consequências diretas na definição, construção, testes e entrega do software ao cliente final, assim como no gerenciamento

do desenvolvimento (LANUBILE et al., 2003).

Devido à necessidade de se manter equipes geograficamente dispersas, várias ferramentas e ambientes têm sido construídos para ajudar no controle e coordenação dessas equipes. O desenvolvimento distribuído de software (DDS) tem sido caracterizado, principalmente, pela colaboração e cooperação entre equipes que realizam atividades em conjunto, mas estão localizados temporal e fisicamente distantes, acrescentando assim, novos desafios ao processo de desenvolvimento de software (CARMEL, 1999), (PRIKLADNICKI et al., 2004). Em particular, o gerenciamento das versões dos artefatos produzidos na execução das atividades do projeto, demanda uma maior atenção por parte da equipe, devido ao fato dos membros da equipe estarem geograficamente distribuídos e compartilharem artefatos.

2.8 Arquitetura do DiSEN

A arquitetura do ambiente DiSEN foi proposta, inicialmente, por Pascutti (2002) e, posteriormente, Schiavoni (2007) definiu um *framework,* denominado FRADE (*Framework to Infrastructure of Distributed Software Development Environment*), para especificar a infra-estrutura que capacita o ambiente a gerenciar a comunicação entre os diversos participantes.

A arquitetura definida por Pascutti é apresentada na Figura 9.

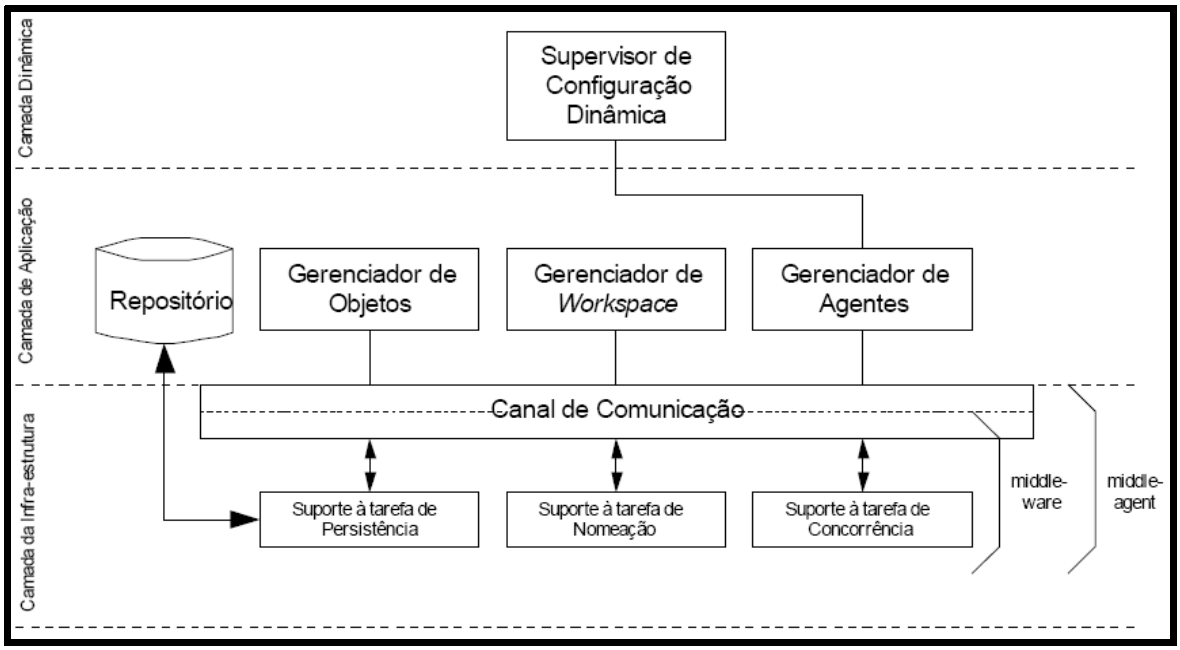

Figura 9: Arquitetura DiSEN
Fonte: PASCUTTI (2002)

A arquitetura (Figura 9) é constituída por gerenciadores (objetos, *workspace* e agentes), suportes (persistência, nomeação e concorrência) e um supervisor de configuração dinâmica para prover infra-estrutura necessária na realização do desenvolvimento distribuído. A seguir uma breve descrição desses elementos:

- **Supervisor de configuração dinâmica:** é responsável pelo controle e gerenciamento da configuração do ambiente, bem como dos serviços que podem ser acrescentados ao ambiente em tempo de execução;
- **Gerenciador de objetos:** é composto pelos gerenciadores de acesso, de atividades, de recursos, de artefatos, de projetos, de processos e de versão e configuração. Esses gerenciadores, com exceção do gerenciador de artefatos, possuem maior detalhamento em Pascutti (2002), Pozza (2005) e Schiavoni (2007);
- **Gerenciador de *workspace*:** responsável pelo controle e gerenciamento da edição cooperativa de documentos. Portanto, deverá prover suporte para um ou mais *workspaces* na utilização dos dados mantidos no repositório;
- **Gerenciador de agentes:** responsável pela criação, registro,

localização, migração e destruição dos agentes;

- **Repositório:** responsável pelo armazenamento dos artefatos, dos dados das aplicações, bem como o conhecimento necessário para a comunicação entre os agentes. Um detalhamento maior pode ser encontrado em Schiavoni (2007).
- **Canal de comunicação:** responsável pela comunicação entre as camadas da infra-estrutura (*middleware* e *middle-agent*) e a camada de aplicação. A comunicação entre os elementos da arquitetura é realizada por intermédio do canal de comunicação.

Uma visão do *framework* FRADE é apresentada na Figura 10.

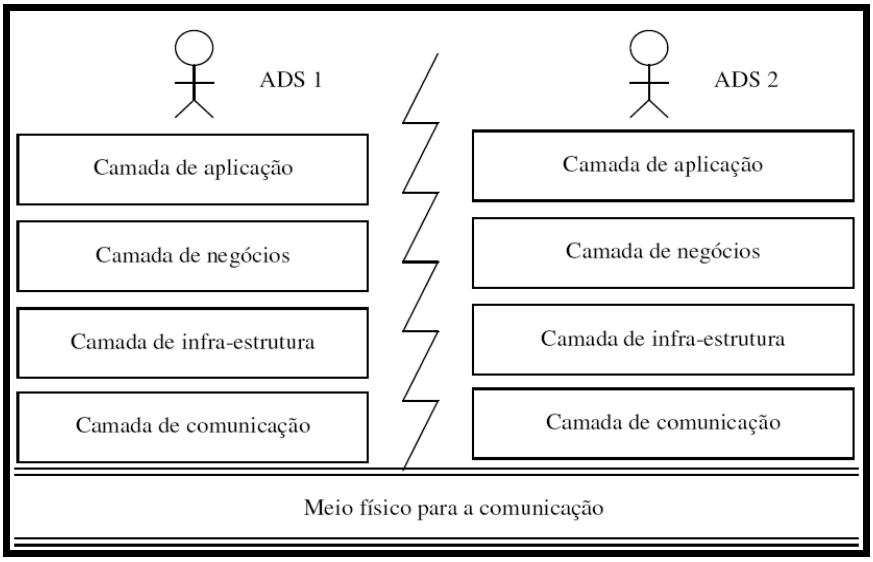

Figura 10: FRADE - Camadas lógicas
Fonte: Schiavoni (2007)

As camadas da Figura 10 são descritas a seguir, mais detalhes podem ser encontradas em Schiavoni (2007).

A camada de aplicação disponibiliza facilidades para implementação de aplicativos e ferramentas para o ambiente DiSEN, como por exemplo, o conjunto de componentes gráficos para auxiliar o desenvolvimento. É por meio desta camada que o usuário interage com o ambiente.

A camada de negócios realiza o mapeamento de classes – entidades que representam o metamodelo de processo para o FRADE. Essas classes são agrupadas em gerenciadores, facilitando a construção de componentes independentes. Os gerenciadores são abstrações dos requisitos funcionais de ADDSs e possuem a definição de seus atributos e as suas funcionalidades.

A camada de infra-estrutura possui um conjunto de Serviços e Suportes para atender às necessidades dos gerenciadores. A comunicação da camada de negócios com a camada de infra-estrutura é realizada por meio do Suporte. O Suporte é responsável por realizar a comunicação com o Serviço.

A camada de comunicação realiza a comunicação de cada estação do DiSEN com o meio físico e assim, com as outras estações. Essa camada foi implementada utilizando *socket* por permitir comunicação *full-duplex*. Dessa maneira, os servidores podem ativar a comunicação com os clientes para, por exemplo, enviar avisos de notificações e instanciar objetos remotos nos clientes.

Capítulo 3 - Trabalhos Relacionados

Este capítulo apresenta os trabalhos encontrados na literatura que se referem aos conceitos envolvidos neste trabalho e que, por esse motivo, influenciaram, direta ou indiretamente, no desenvolvimento desta dissertação.

3.1 Ferramentas para Apoio à Percepção

Existem diversas propostas para apoiar a percepção em edição colaborativa. A ferramenta CO2DE (MEIRE et al., 2003) se apresenta como uma implementação de editor gráfico e síncrono de diagramas UML, baseado em uma metáfora de "máscaras", representando versões. Outro editor colaborativo de artefatos UML é D-UML (BOULILA et al., 2003). Tukan (SCHUMMER e SCHUMMER, 2000) é um ambiente síncrono e distribuído de programação Smalltalk. O ambiente SAMS (MOLLI et al, 2002) permite a edição colaborativa através de interações síncrona, assíncrona e multi-síncrona. NetEdit (ZAFFER et al, 2001) é um editor colaborativo multi-síncrono de documentos texto acessível pela Web.

A ferramenta Palantír (SARMA et al., 2003) fornece aos desenvolvedores, pertencentes à sessão de colaboração, informações sobre os espaços de trabalho dos demais desenvolvedores. Koblylinski et. al (2002) apresentam uma proposta de sistema de percepção, que permite que colaboradores monitorem atividades de outros sobre artefatos de software.

Em termos de mecanismos de percepção, CO2DE, D-UML e Tukan possuem suporte à interação síncrona, sendo que CO2DE provê também suporte assíncrono para percepção. Palantír, SAMS, NetEdit e a abordagem descrita em KOBLYLINSKI et al.

(2002) contemplam mecanismos de percepção em interações multi-síncrona, além do suporte síncrono e assíncrono à percepção. Apoiando a percepção em modelos UML, tem-se CO2DE e D-UML.

Tukan utiliza como metáfora ícones usados em boletins meteorológicos; mudanças com alto impacto são associadas à ícones de tempo instável. Palantír fornece informações sobre severidade de mudanças sobre o artefato compartilhado por meio de diversas formas gráficas de representação (por exemplo, barra de progresso).

3.2 Sistema de Controle de Versões (SCV)

OSCAR (NUTTER et al., 2002) é um sistema de gerenciamento de artefatos do projeto GENESIS (*Generalized Environment for Process Management In Cooperative Software Engineering*). Ele utiliza o CVS para o controle de versões dos artefatos e possui um mecanismo de notificação de eventos para reportar as mudanças que ocorrem sobre os artefatos.

O sistema ADAMS (BRUEGGE et al., 2006) é desenvolvido como uma aplicação Web e permite a visualização gráfica da rastreabilidade dos artefatos e suas versões apoiando o desenvolvimento colaborativo. Esse sistema também possui uma funcionalidade para notificação de novas versões dos artefatos.

O ambiente WebAPSSE (SALES et al., 2007) possui um sistema de controle de versões baseado no paradigma de *check-in/check-out*. Atualmente, esse sistema utiliza o CVS para controle de versões de artefatos. Esse ambiente possui um serviço de notificação que reporta aos interessados todas as mudanças realizadas sobre os artefatos. As notificações são percebidas na agenda dos desenvolvedores.

O Odyssey-VCS (OLIVEIRA, 2005) funciona como um módulo para o ambiente Odyssey (WERNER et al., 2003), que é um ambiente de reutilização baseado

em modelos de domínio, mantido pela COPPE/UFRJ desde 1997.

O Odyssey-VCS controla as versões de modelos UML (OMG, 2001), utilizando políticas para unidades de comparação e versionamento, de acordo com cada projeto, permitindo que os modelos sejam tratados em granularidade fina.

Para permitir o acesso concorrente o Odyssey-VCS utiliza a estratégia otimista. Essa estratégia é a mesma utilizada em sistemas de controle de versões baseados em arquivos e necessitam de algoritmos de junção.

O Odyssey possui uma camada de transporte para que as ferramentas *CASE* possam se comunicar com o Odyssey-VCS. A camada de transporte utiliza *Web Service* como protocolo de comunicação, como é mostrado na Figura 11.

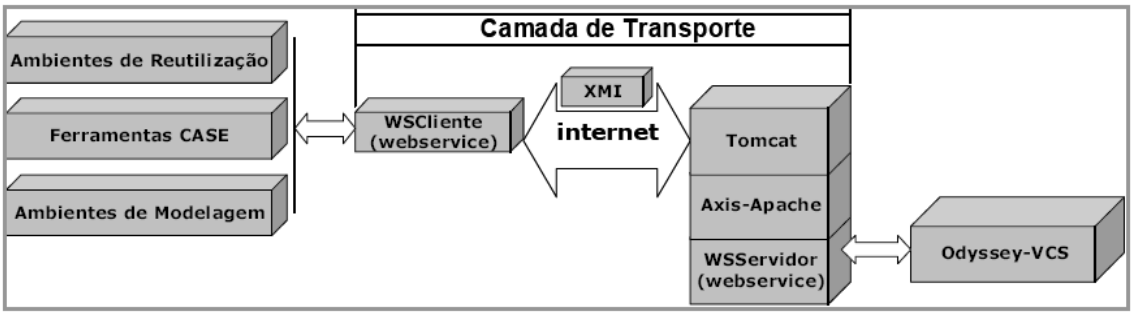

Figura 11: Camada de transporte – Odyssey-VCS
Fonte: OLIVEIRA (2005)

O repositório de armazenamento dos artefatos do Odyssey-VCS é centralizado e ele controla somente modelos UML, não levando em consideração os artefatos binários e de código fonte. Por utilizar a estratégia otimista, existe uma maior possibilidade de ocorrer conflitos e assim utiliza uma parte do tempo de desenvolvimento para corrigí-los.

Figueiredo (2004) definiu um processo baseado em técnicas de engenharia de software para GCS e também desenvolveu a ferramenta GConf para apoiar tal processo. A ferramenta GConf apóia as atividades de: identificação da configuração,

controle da configuração, relato da situação, auditoria da configuração e gerência de liberação e entrega. Essa ferramenta foi desenvolvida para o Ambiente de Desenvolvimento de Software Orientado a Organização (ADSOrg).

O ADSorg enfatiza o aprendizado organizacional relacionado com o desenvolvimento de software. O conhecimento adquirido em projetos anteriores é utilizado na execução de atividades de engenharia de software (VILLELA et al., 2003).

Segundo o autor, a ferramenta GConf apresenta algumas vantagens em relação às ferramentas: CVS (*Concurrent Versions System*), *ClearCase* e o *Visual Source Safe.* Entre elas, destaca-se o controle intenso sobre as modificações a serem realizadas. Antes de realizar uma modificação, o desenvolvedor deve fazer um pedido formal e este deve ser aprovado pelo gerente de projeto. O mesmo ocorre após a realização da modificação, para se tornar uma nova versão, a modificação deve ser analisada e aprovada pelo gerente de projeto.

A Figura 12 apresenta a tela de consulta de conhecimento, da ferramenta GConf, com a descrição da atividade.

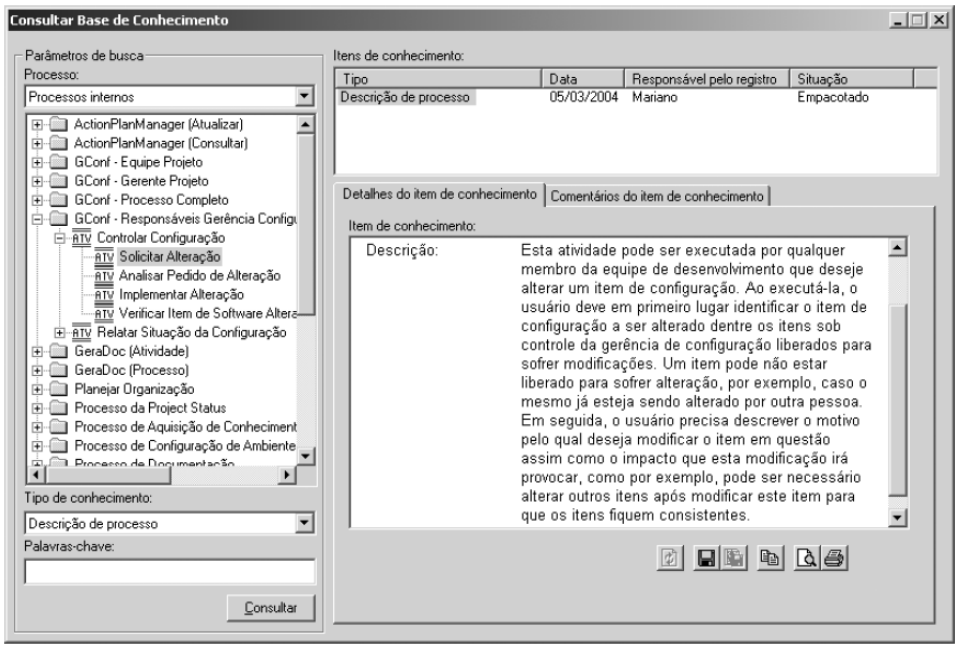

Figura 12: Ferramenta Gconf
Fonte: FIGUEIREDO (2004)

O objetivo da ferramenta GConf é o controle das modificações dos artefatos em um ADS e a disseminação do conhecimento, de projetos anteriores, dentro da organização. O controle das versões dos artefatos não é tratado por essa ferramenta.

3.3 MAIS (*Multi-synchronous Awareness InfraStructure*)

A ferramenta MAIS está inserida no contexto do Projeto OdysseyShare (ODYSSEY, 2004), baseada na arquitetura cliente-servidor. Seu objetivo é coletar e apresentar informações de mudanças para o desenvolvedor. Os eventos coletados são armazenados em um espaço de tuplas (CARRIERO e GELERNTER, 1989), visível a todos os desenvolvedores. Quando novos eventos são gerados, os desenvolvedores são notificados e os obtém do espaço de tuplas.

Os eventos são coletados do Ambiente onde MAIS está inserido e são apresentados aos desenvolvedores na forma de mensagens, descritas textualmente. Os desenvolvedores visualizam os eventos gerados e quem os gerou, além dos elementos do modelo envolvidos nos eventos. Eventos gerados pelo próprio desenvolvedor são apresentados em uma lista diferente daqueles gerados pelos demais.

3.4 Augur: Combina Informações de Artefatos e Atividades

Augur é uma ferramenta desenvolvida por Froehlich e Dourich (2004) que permite a visualização de processos distribuídos do desenvolvimento do software. Augur gera representações visuais dos artefatos e das atividades do desenvolvimento do software e permite que os colaboradores explorem o relacionamento entre eles. Essa ferramenta é projetada para os colaboradores que participam no processo do desenvolvimento do software.

Froehlich e Dourich (2004) mostram que os artefatos podem carregar informações sobre as atividades em andamento ou concluídas, assim, tornando possível que o próprio artefato extraia as informações sobre as atividades que possuam dependências.

3.5 MIMIX

A possibilidade de compartilhar modelos UML produzidos por diferentes ferramentas CASE se tornou possível a partir da criação do MOF como metamodelo da UML e da especificação XMI (*XML Metadata Interchange*) (OMG, 2002) (MURTA, 2007).

O MIMIX (EL-JAICK, 2004) tem como objetivo apoiar a integração de ferramentas CASE heterogêneas. A integração é realizada utilizando o XMI como formato de dados entre as ferramentas. *Web Service* é utilizado na comunicação e atua entre as ferramentas CASE e o MIMIX, permitindo que as equipes se encontrem geograficamente distribuídas.

Segundo o autor, o MIMIX possui mecanismos para a resolução de conflitos quando for necessário reintegrar elementos em um único modelo.

3.6 ARIANE

O mecanismo ARIANE (SANTOS, 2003) monitora os eventos de Sistemas de Gerênciamento de Banco de Dados (SGBD). Esse mecanismo é independente de um SGBD específico e trata somente interações assíncronas. ARIANE utiliza os eventos capturados com o intuito de apoiar a percepção. Os eventos são armazenados em um repositório de eventos, para posteriormente serem analisados.

Os eventos, que estão armazenados no repositório de eventos, podem ser consultados: por um determinado período, por eventos referentes a um determinado artefato, por eventos produzidos por um determinado usuário e por eventos que indiquem uma determinada operação.

Santos (2003) propõem o uso de uma estrutura multidimensional para o armazenamento das informações de percepção produzidas, possibilitando que consultas analíticas possam ser realizadas por ferramentas de processamento analítico (OLAP - *On-Line Analytical Processing*).

O processo de percepção está dividido em quatro etapas principais: produção, distribuição, consumo e análise de eventos (SANTOS, 2003).

A etapa de **produção** capta as ações dos usuários sobre os artefatos armazenados no SGBD. São extraídas as informações relevantes, gerando um evento estruturado para que sejam compreendidos. Os eventos são armazenados para posterior consulta e recuperação dessa informação.

A etapa de **distribuição** é responsável por transmitir e distribuir os eventos aos componentes visuais que tenham registrado interesse.

A etapa de **consumo** compreende na apresentação das informações de percepção para os usuários consumidores.

Por último, a etapa de **análise** prepara os eventos e os carrega em uma estrutura apropriada para o processamento analítico e a mineração de dados.

Para evitar a sobrecarga de informações, ARIANE fornece filtros para que sejam aplicados em cada uma dessas etapas, porque nem toda ação deve ser monitorada e nem todas as informações de percepção devem ser distribuídas e apresentadas.

Um exemplo de utilização de filtro é restringir somente as informações que interessam ao usuário. Nesse contexto, o filtro pode ser aplicado aos artefatos, ao tipo de operação ou por um determinado período de interesse do usuário.

Um problema pode ocorrer quando se está utilizando o filtro sobre os

artefatos. Dessa maneira, quando um usuário cria um novo artefato, os demais usuários não receberão informações sobre esse novo artefato, porque no momento em que o filtro foi aplicado o novo artefato não existia.

3.7 Odyssey-CCS

Odyssey-CCS (LOPES, MURTA e WERNER, 2006) é uma ferramenta de apoio ao controle de modificações de software. Ela auxilia na comunicação e coordenação na execução das atividades do projeto, mantendo o controle sobre as modificações realizadas como: quem fez o quê, quando, onde, como e por que.

Para a utilização da ferramenta é necessário a execução de seis atividades:

A primeira atividade é a de **modelagem do processo** (1), para cada software produzido na empresa, o gerente de configuração, deve modelar um processo de controle de modificações. Segundo o autor, esse processo deve ser adaptado do processo padrão definido para a organização.

Na atividade de **modelar formulário (2)**, o gerente de configuração, define quais informações serão coletadas durante a realização de controle das modificações. Assim, os envolvidos na modificação realizarão as atividade e tomarão as decisões necessárias.

A próxima atividade é **configurar o processo (3)**, nesse momento serão atribuídas as responsabilidades e os papéis definidos no processo, e também é configurado o envio de notificações por e-mail em relação ao conteúdo, destinatários e momento.

É disponibilizado para os usuários **verificarem quais atividades e decisões estão pendentes para a execução (4)**. O usuário pode verificar as suas pendências relativas ao projeto que participa e também pode receber notificações por e-mail sobre

o progresso de execução dos processos.

Na atividade de **realizar as atividades e decisões pendentes de execução (5)** o usuário informa que está executando uma atividade ou tomando uma decisão e, assim, as outras pessoas envolvidas serão notificadas. Nessa atividade o usuário também consegue visualizar as informações coletadas anteriormente.

Ao **finalizar uma atividade ou decisão (6),** o usuário deve fornecer informações necessárias para que a modificação possa ser compreendida no futuro. Após esta ação as informações são disponibilizadas para os usuários responsáveis pelas novas pendências.

A ferramenta Odyssey-CCS trabalha em conjunto com a ferramenta Odyssey-VCS, permitindo que as versões dos artefatos sejam associadas à modificação e possam ser consultadas posteriormente.

3.8 IMART

IMART é um modelo de interoperabilidade para ADDS, que permite manipular as informações contidas nos artefatos por meio de um metamodelo. Assim, é possível que sejam armazenados em diferentes formatos e manipulados de diferentes formas (WIESE, 2006).

Segundo WIESE (2006), o IMART permite que um artefato gerado por uma ferramenta seja importado ou exportado no ADDS. Os problemas referentes aos conflitos que possam surgir entre as versões de um artefato não é tratado pelo IMART. Um metamodelo é definido para representar os artefatos que são produzidos pelas ferramentas que desejam interoperar.

O IMART não se preocupa em como e onde os artefatos serão persistidos. Portanto, ele não trata políticas de concorrência, fusão entre duas versões,

fragmentação e distribuição dos artefatos.

A Figura 13 apresenta os componentes responsáveis por oferecer as funcionalidades do IMART.

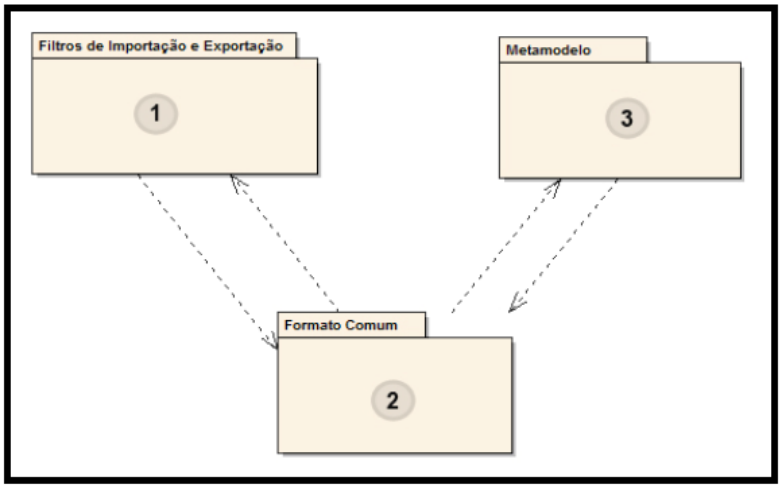

Figura 13: Componentes do modelo de interoperabilidade
Fonte: WIESE (2006)

O componente **Filtros de Importação e Exportação (1)** é responsável por atribuir as informações de um artefato ao metamodelo. Para que esse procedimento ocorra, é utilizado um mapeamento do artefato para o metamodelo. O mapeamento descreve as relações entre as classes, que representam os artefatos, e ele é utilizado pelo *framework* Castor (http://www.castor.org) para que as informações do artefatos sejam atribuídas ao metamodelo.

O componente **Formato comum (2)** executa o refinamento do artefato antes e após a sua transformação. Esse procedimento é necessário pois diferentes ferramentas podem conter *"namespaces"* distintos antes das *tag's* do arquivo XML/XMI.

Os metamodelos permitem descrever os dados por meio de estruturas bem-definidas. Dessa maneira, o componente **Metamodelo (3)** é responsável por representar a semântica dos artefatos, possibilitando que sejam manipulados. Com a

utilização do metamodelo é possível implementar diferentes formas para persistir os artefatos, como por exemplo, persisti-lo em um SCV ou em banco de dados.

3.9 Considerações Finais

O estudo realizado sobre os trabalhos apresentados neste capítulo mostrou várias características que podem fazer parte de um SCV, tais como: implementação de técnicas de percepção, extração de informações sobre um repositório de artefatos, controle de modificações de software, controle de versões de artefatos, resolução de conflitos entre versões de artefatos, entre outras. Entretanto, estas ferramentas não possibilitam a utilização de repositórios distribuídos, a resolução de conflitos em artefatos no formato XMI, utilizando a própria ferramenta CASE que gerou o artefato e nem controle sobre o tempo que o artefato fica alocado para o desenvolvedor.

A construção do DiSEN-SCV baseou-se nessas características, com foco de utilização em um ADDS. Além disso, foi adicionado um controle sobre o período de tempo na alocação dos artefatos e um mecanismo diferenciado para resolução de conflitos em artefatos no formato XMI.

Capítulo 4 - DiSEN-SCV

O Gerenciador de Artefatos é um componente do Ambiente Distribuído de Desenvolvimento de Software – DiSEN. Em Pascutti (2002) é proposto um gerenciador de artefatos, com a finalidade de gerenciar as versões dos artefatos que são gerados durante a execução das atividades do projeto.

Além de gerenciar as versões dos artefatos, o Gerenciador de Artefatos - DiSEN-SCV é responsável por controlar o acesso aos artefatos e, também, resolver os conflitos, dentre as versões, que possam ocorrer durante o desenvolvimento.

Para resolução de conflitos foram seguidas as definições apresentadas por Borghoff e Schlichter (2000), conforme explicadas na Seção 2.3. Apesar da técnica pessimista diminuir o paralelismo na execução das atividades, a estratégia adotada no presente trabalho é a alocação de artefato por um período de tempo determinado. Assim, cada desenvolvedor autorizado pode alocar um artefato para realizar as alterações necessárias, iniciando a contagem do tempo correspondente. Tão logo este tempo expire ou quando a submissão, no repositório, de uma nova versão do artefato é realizada, ocorre a desalocação do artefato para este desenvolvedor. A partir deste momento, este artefato estará disponível para um outro desenvolvedor que esteja interessado em realizar alterações.

O DiSEN-SCV implementa a técnica pessimista com controle de tempo de alocação, devido à dificuldade em garantir que as alterações, executadas por diferentes pessoas, sejam coerentes, uma vez que essas pessoas podem estar geograficamente dispersas, em se tratando de um ADDS.

Devido a essa possibilidade, o DiSEN-SCV implementa técnicas de percepção para manter os participantes da equipe atualizados quanto às modificações que

ocorrerem nos artefatos. As técnicas implementadas foram: notificação por e-mail e mensagens instantâneas, utilização de cores e imagens para definir os estados dos artefatos.

Para o armazenamento dos artefatos, o DiSEN-SCV utiliza como repositório um SGBD, por garantir que as transações são executadas de forma segura, seguindo as propriedades ACID (Atomicidade, Consistência, Isolamento, Durabilidade) e, também, por manter os artefatos e seus metadados em um único repositório. Detalhes sobre a utilização do SGBD, como repositório de artefatos, são apresentados na Seção 4.3.

A seguir são descritas, com mais detalhes, as funcionalidades do DiSEN-SCV.

4.1 Funcionalidades

As funcionalidades do DiSEN-SCV foram definidas pelo grupo de pesquisa de engenharia de software do Departamento de informática (DIN) da Universidade Estadual de Maringá (UEM). As funcionalidades são: (1) armazenar os artefatos, (2) recuperar os artefatos, (3) controlar as versões, (4) controlar a requisição dos desenvolvedores sobre os artefatos, (5) permitir melhor equilíbrio no tempo de alocação dos artefatos, (6) fornecer informações sobre os artefatos, (7) resolver conflitos de versões de artefatos no formato XMI e (8) controlar os repositórios locais. Elas são ilustradas pelo diagrama de caso de uso apresentado na Figura 14.

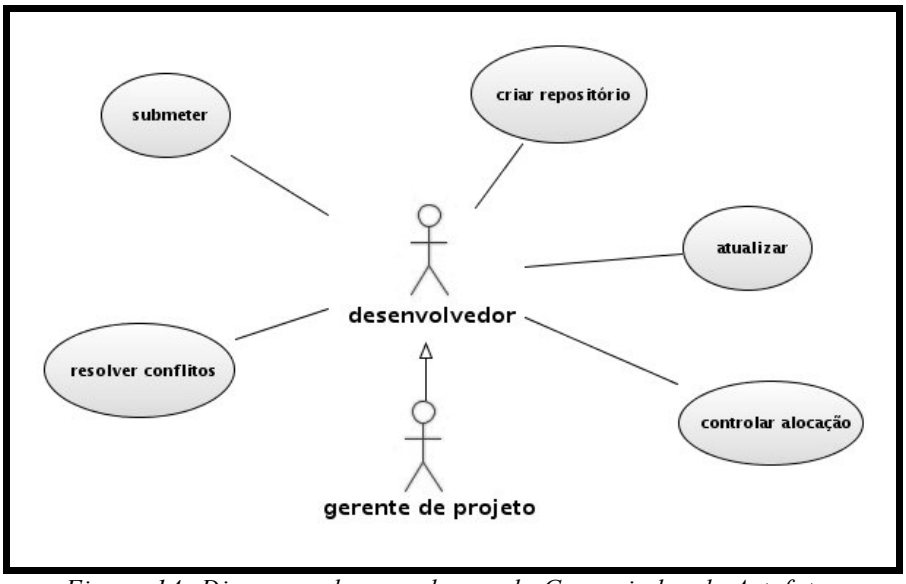

Figura 14: Diagrama de caso de uso do Gerenciador de Artefatos

As funcionalidades (1 e 3) são atendidas pelo caso de uso ***submeter***, as funcionalidades (2 e 6) pelo caso de uso ***atualizar***, as funcionalidades (4 e 5) pelo caso de uso ***controlar alocação***, a funcionalidade (7) pelo caso de uso ***resolver conflitos***, a funcionalidade (8) pelo caso de uso ***criar repositório*** e são descritas a seguir:

Armazenar artefatos (1) é o responsável por armazenar os artefatos, enviados pelos desenvolvedores, no repositório de artefatos.

Recuperar os artefatos (2), retorna a última versão dos artefatos requisitados pelo desenvolvedor.

O **controle das versões** (3) é realizado, basicamente, em três tipos de artefatos: código-fonte, binário e artefatos de modelos UML. Essa funcionalidade é responsável, também, por manter o histórico das modificações que foram realizadas pelos desenvolvedores em cada versão dos artefatos.

Os artefatos são associados a uma atividade, dessa maneira o **acesso aos artefatos** (4) é permitido somente aos desenvolvedores que são designados para realizá-la. Caso seja necessário, um artefato pode ser alocado a um desenvolvedor e, para que a desalocação não dependa somente da vontade do desenvolvedor, é atribuído

um **período de tempo para a alocação** (5). Portanto, um artefato pode ser desalocado pela ação do desenvolvedor, que alocou o artefato, ou pelo término do período de alocação. O tempo de alocação é definido pelo gerente de projeto. Durante esse tempo, o desenvolvedor pode realizar suas modificações no artefato sem a preocupação de que outro desenvolvedor também altere o artefato. Também, durante esse tempo, qualquer outro desenvolvedor que solicitar o artefato, será avisado de que o mesmo não poderá ser modificado (arquivo somente leitura), por já existir um outro desenvolvedor que alocou o artefato para realizar modificações. Decorrido o tempo determinado, se o desenvolvedor não submeter o artefato com suas modificações para o repositório, ele corre o risco de um outro desenvolvedor alocar o artefato e assim terá que aguardar até que consiga novamente alocá-lo, possivelmente, tendo retrabalho para fundir suas modificações.

O desenvolvedor pode obter **informações sobre cada versão do artefato** (6): Quem modificou determinada versão?, Quando?, O que foi modificado? e Porquê?. Com isso, é possível determinar quem realizou mais modificações em um determinando artefato; identificar a quantidade de linhas que foram modificadas, em artefatos no formato texto, por um determinado desenvolvedor e aplicar métricas para avaliar o desempenho do desenvolvedor, entre outras. Essas informações são armazenadas pelo DiSEN-SCV, apesar de ser importante a extração dessas informações, está além do escopo dessa dissertação.

A funcionalidade de **resolução de conflitos de versões de artefatos** no formato XMI (7) é responsável por auxiliar o desenvolvedor quando ocorre conflito entre duas versões de um artefato. Essa funcionalidade compara os elementos das duas versões e permite que o desenvolvedor consiga distinguir, na própria ferramenta CASE, os elementos que estão divergentes. Assim, o desenvolvedor poderá aceitar ou rejeitar as modificações realizadas.

A funcionalidade **controlar os repositórios locais (8)** é responsável por criar

e controlar os repositórios locais, não permitindo que o desenvolvedor crie um novo repositório e sobreponha um existente.

4.2 Arquitetura do DiSEN-SCV

O DiSEN-SCV é constituído pelos componentes: Alocação, Percepção, Tratador de Conflito e Artefato, como podem ser visualizados na Figura 15.

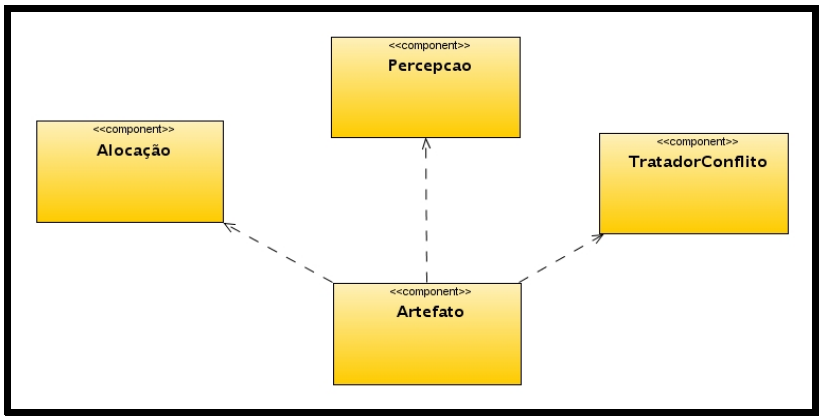

Figura 15: Diagrama de Componentes

Cada componente é descrito a seguir:

Componente Alocação: responsável por gerenciar a alocação sobre os artefatos. O desenvolvedor pode alocar um artefato por um determinado período de tempo. Esse tempo é determinado pelo gerente de projetos com base em seu conhecimento e experiência. Para determinar o tempo de alocação, ele deve levar em consideração o grau de dificuldade para realização da atividade e a experiência dos desenvolvedores. Por exemplo, uma atividade com grau de dificuldade baixo e desenvolvedores com experiência alta, o período de alocação deve ser pequeno.

O diagrama de classes desse componente é apresentado na Figura 16.

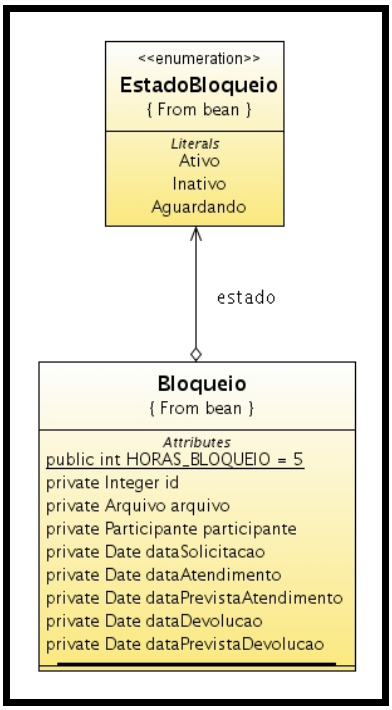

Figura 16: Diagrama de classes do componente Alocação

A classe **Bloqueio** é uma entidade para representar as solicitações de alocação sobre um artefato, que são realizadas pelos participantes do projeto. Cada instância dessa classe identifica, o participante que solicitou a alocação, qual é o artefato que o participante deseja alocar, a data e a hora da solicitação. Quando o participante solicita uma alocação é definido um estado para essa solicitação. Se o artefato, que o participante deseja alocar, não estiver alocado para algum participante, o estado é definido como **Ativo**, senão o estado é definido como **Aguardando**.

As solicitações, que estão com o estado **Aguardando,** ficam em espera até que o artefato seja liberado. As solicitações que estão com o estado **Aguardando** são mantidas ordenadas, crescentemente, pela data de solicitação. Quando um artefato é desalocado, a primeira solicitação que está com o estado **Aguardando** é colocada em estado **Ativo.**

O estado **Ativo** identifica para qual participante o artefato está alocado. Portanto, cada artefato deve possuir somente uma instância de **Bloqueio** com o estado **Ativo**.

A classe **EstadoBloqueio** é do tipo *enum*[4] e é utilizada para definir o estado da alocação, que pode ser:

- **Ativo:** esse estado indica qual instância da classe **Bloqueio** está ativa para cada artefato.
- **Inativo:** indica que já expirou o tempo de alocação.
- **Aguardando:** indica que a instância está aguardando para adquirir a alocação.

Componente Percepção: responsável por disseminar informações entre os participantes do projeto. Esse componente notifica os usuários, em relação aos artefatos modificados, de duas formas:

- **por aviso:** caso o usuário esteja autenticado no ambiente, ele recebe um aviso do tipo *instant messenger*, constituindo-se na forma mais rápida do usuário ter conhecimento sobre uma mudança no artefato.
- **por e-mail:** é a forma mais garantida de que o usuário receberá a informação, entretanto, não é possível garantir o momento em que o usuário irá ler o e-mail.

Componente Tratador Conflito: responsável por auxiliar o desenvolvedor na resolução de conflitos entre duas versões de um artefato. Esse componente trata os conflitos de artefatos no formato texto e XMI. Em artefatos no formato texto, os conflitos são identificados comparando linha à linha de cada versão do artefato. No

4 É um tipo cujo os campos consistem de um conjunto fixo de constantes. http://java.sun.com/docs/books/tutorial/java/javaOO/enum.html.

caso dos artefatos no formato XMI, os eventuais conflitos são identificados comparando os elementos do artefato.

As classes que constituem esse componente estão apresentadas na Figura 17.

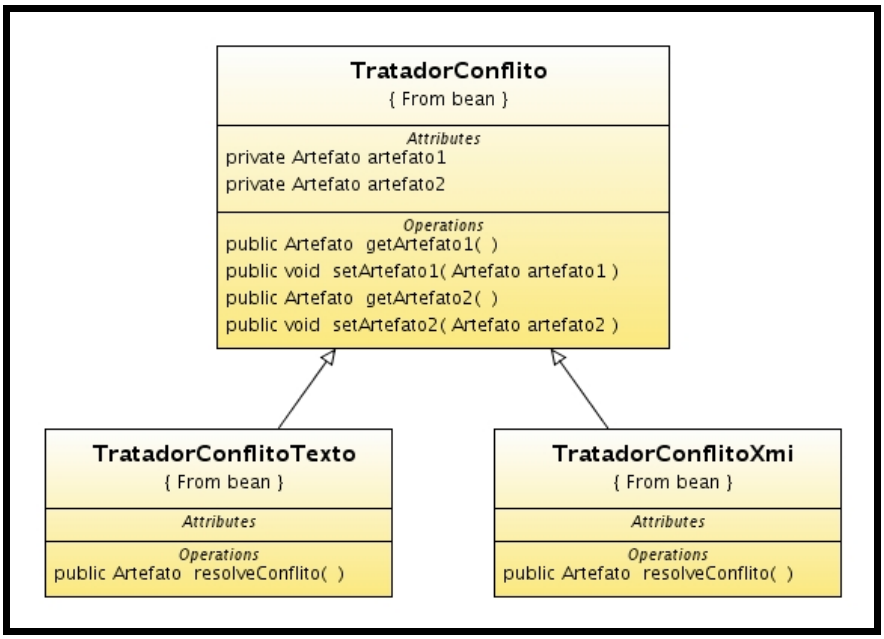

Figura 17: Diagrama de classes do componente Tratador Conflito

A classe **TratadorConflito** é especializada em outras duas classes: **TratadorConflitoTexto** e **TratadorConflitoXmi**.

Foi necessário fazer a especialização da classe **TratadorConflito** porque os conflitos podem ser resolvidos diferentemente para cada tipo de artefato.

A classe **TratadorConflitoTexto** é responsável por resolver conflitos em artefatos no formato do tipo texto, como por exemplo, arquivos de código-fonte. A classe **TratadorConflitoXmi** resolve conflitos em artefatos no formato XMI. Inicialmente, foi implementado para diagramas de use-case, uma vez que existe um metamodelo definido por Wiese (2006).

Componente Artefato: responsável pelo controle e armazenamento das versões dos artefatos nos repositórios. Esse componente controla os repositórios, central e local. O repositório central armazena as versões dos artefatos e o repositório local armazena a versão que o desenvolvedor está trabalhando atualmente. Seguindo o *framework* definido por Schiavoni (2007), foram criadas duas classes para gerenciar os repositórios: *ServiçoArtefato* e *SuporteArtefato*.

A classe *ServiçoArtefato* é responsável pelo gerenciamento do repositório central. Os métodos dessa classe controlam o acesso ao repositório, armazenam novas versões de artefatos, recuperam uma determinada versão, solicitam ao componente **Alocação** o bloqueio de um artefato, enviam as notificações ao componente **Percepção** para que sejam enviadas aos desenvolvedores e utilizam o componente **TratadorConflito** para auxiliar na resolução de conflitos.

A classe *SuporteArtefato* é responsável pelo gerenciamento do repositório local. Dentre as suas responsabilidades, pode-se citar: validação do repositório informado pelo desenvolvedor, verificando se realmente é um repositório local; controle dos artefatos localmente, mantendo a versão, data, nome e *checksum* dos artefatos no momento que eles foram armazenados no repositório local.

O *SuporteArtefato* cria um arquivo no repositório local para controlar os artefatos que estão armazenados localmente. Um exemplo desse arquivo é apresentado na Figura 18.

```
<disen.infraestrutura.suporte.artefato.Descritor>
  <arquivos>
    <disen.infraestrutura.suporte.artefato.DescritorCorpo>
      <nome>use-case diagram.xmi</nome>
      <data>2008-07-23 08:52:00.672 BRT</data>
      <checkSum>1679146560</checkSum>
      <versao>2</versao>
      <idTarefa>1</idTarefa>
    </disen.infraestrutura.suporte.artefato.DescritorCorpo>
    <disen.infraestrutura.suporte.artefato.DescritorCorpo>
      <nome>presentation.pdf</nome>
      <data>2008-07-23 08:52:00.672 BRT</data>
      <checkSum>4093417219</checkSum>
      <versao>4</versao>
      <idTarefa>2</idTarefa>
    </disen.infraestrutura.suporte.artefato.DescritorCorpo>
    <disen.infraestrutura.suporte.artefato.DescritorCorpo>
      <nome>document.txt</nome>
      <data>2008-07-23 08:52:00.672 BRT</data>
      <checkSum>1440763004</checkSum>
      <versao>5</versao>
      <idTarefa>2</idTarefa>
    </disen.infraestrutura.suporte.artefato.DescritorCorpo>
  </arquivos>
</disen.infraestrutura.suporte.artefato.Descritor>
```

Figura 18: Descritor artefatos locais

Quando for necessário comunicar-se com o repositório central para enviar, buscar ou alocar uma versão do artefato, o *SuporteArtefato* solicitará a execução dessas ações ao *ServiçoArtefato,* que é o responsável pelo repositório central.

Esse componente também possui um conjunto de classes que representam as entidades para manipulação dos artefatos. O diagrama de classes é apresentado na Figura 19.

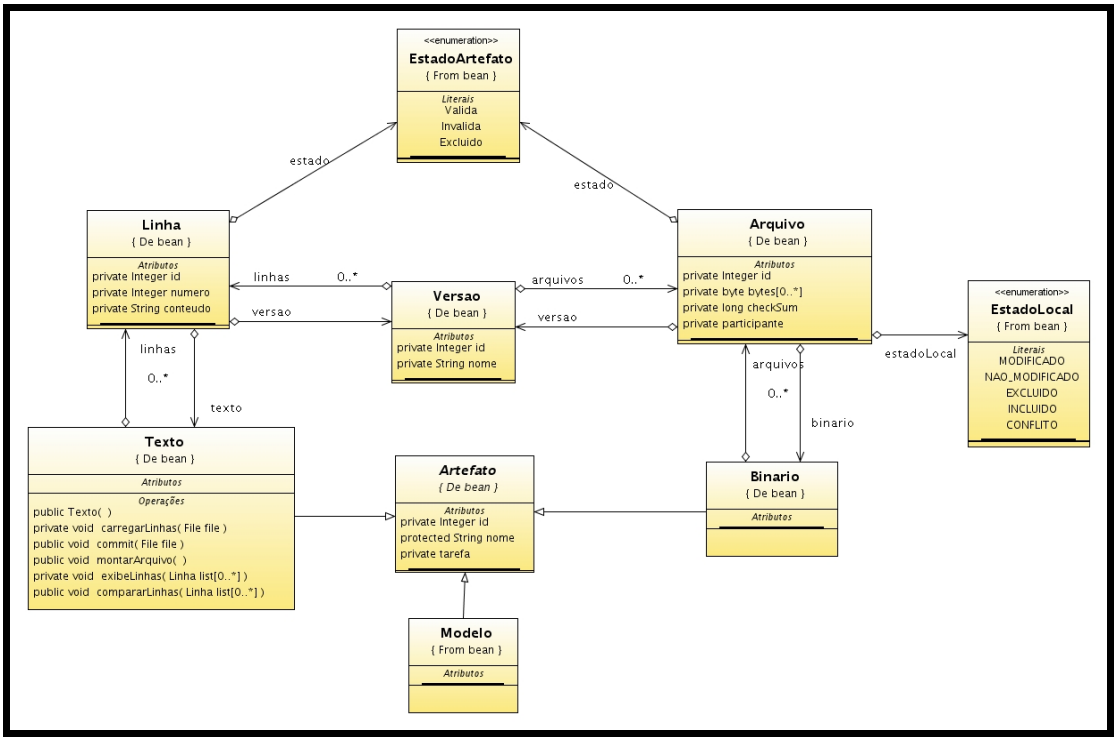

Figura 19: Diagrama de classes do componente artefato

4.3 Implementação

O DiSEN-SCV foi implementado utilizando a tecnologia Java e segue a arquitetura do *framework* FRADE, permitindo que seja integrado ao ambiente DiSEN como um *plugin.*

A sua interface gráfica foi construída de maneira que fosse simples para os usuários. Ela permite que o desenvolvedor escolha o lugar onde os artefatos serão armazenados localmente. Esse lugar é chamado de repositório local. Dessa maneira, o desenvolvedor pode trabalhar desconectado do servidor. Após escolhido o repositório local, o desenvolvedor deverá fazer o "*Checkout*", que consiste em fazer uma cópia dos artefatos que estão no repositório central (servidor) para o repositório local. Com isso, o desenvolvedor pode trabalhar com as cópias dos artefatos, fazendo as modificações necessárias.

Caso o desenvolvedor queira alocar um artefato por um período de tempo, ele deverá selecionar o artefato e acionar o botão "*Lock*". A fim de mostrar os diferentes estados de um artefato, foi adotada a política de cores para identificar cada um deles. Assim, o artefato aparecerá com o cadeado na cor vermelha indicando que o mesmo está alocado para o próprio desenvolvedor (Figura 20). O cadeado na cor azul indica que o artefato está alocado para um outro desenvolvedor, portanto, as próximas requisições de alocação serão colocadas em uma lista de espera e assim que o artefato for desalocado, o DiSEN-SCV envia um e-mail e uma mensagem de aviso ao primeiro da lista de espera.

Os artefatos que são modificados pelo desenvolvedor são apresentados na cor azul e os que são adicionados na cor verde. Caso um artefato seja excluído ele aparecerá na cor cinza.

Após as modificações, os artefatos deverão ser enviados novamente ao repositório central. Para a realização dessa ação o desenvolvedor deverá acionar o

botão "*Commit*". Essa funcionalidade sincroniza os artefatos do repositório local com os artefatos do repositório central. Caso um artefato modificado esteja alocado para outro desenvolvedor, ele será ignorado, e só poderá ser submetido quando o desenvolvedor conseguir a sua alocação.

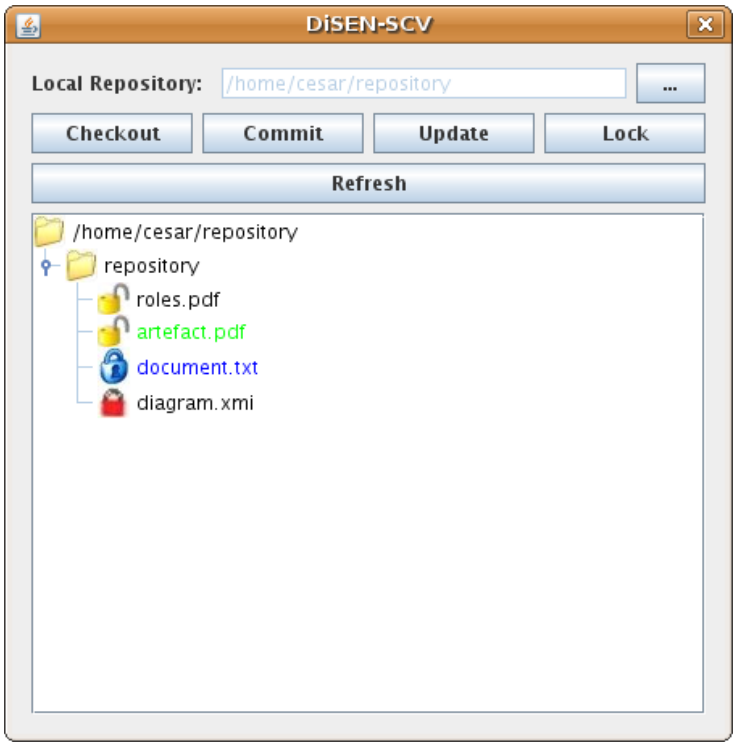

Figura 20: Gerenciador de Artefato - DiSEN-SCV

O DiSEN-SCV também possui a funcionalidade para resolver os conflitos de versões de artefatos no formato XMI. Esses artefatos são produzidos por ferramentas de modelagem UML, tais como: Poseidon[5], Enterprase Architect[6], Requisite (BATISTA, 2003), IBM Rational Rose[7], entre outras.

O ambiente DiSEN permite que essas ferramentas possam interoperar por meio da ferramenta IMART (WIESE, 2006) e o controle das versões dos artefatos

5 Poseidon for UML. http://www.gentleware.com

6 Enterprise Architec. http://www.sparxsystems.com.au

7 IBM Rational Rose. http://www-306.ibm.com/software/rational

produzidos é gerenciado pelo DiSEN-SCV. Esses artefatos também podem entrar em conflito de versões. Artefatos nesse formato são mais difíceis de resolver conflitos do que em artefatos no formato de código-fonte, porque eles são gerados automaticamente pelas ferramentas de modelagem, enquanto que os artefatos de código-fonte são gerados pelo próprio desenvolvedor. Essa funcionalidade é exemplificada a seguir.

Suponha que dois desenvolvedores estão trabalhando na mesma atividade e que ambos possuam o artefato na primeira versão. O primeiro desenvolvedor realiza suas modificações (Figura 21) e submete o artefato para o repositório central, gerando a segunda versão do artefato.

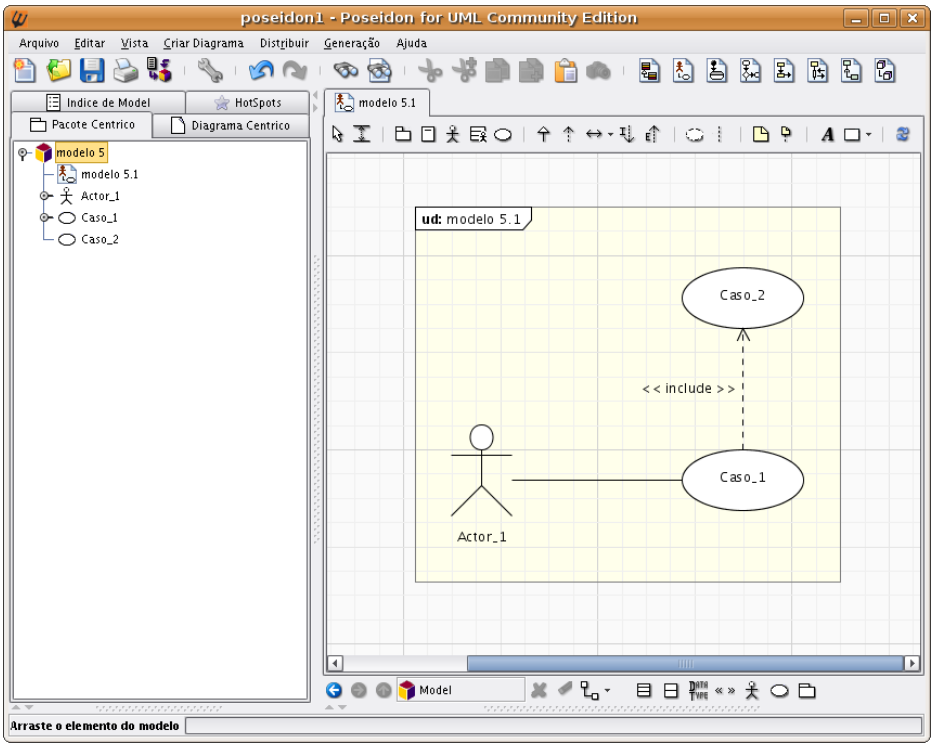

Figura 21: Modificações realizadas pelo primeiro desenvolvedor

O segundo desenvolvedor, que começou a modificar o artefato (Figura 22), ao mesmo tempo, que o primeiro, recebe um aviso de que o artefato que está modificando possui uma nova versão. Nesse momento, o segundo desenvolvedor sincroniza sua cópia local (que está sendo modificada) com a versão que está no repositório central que, nesse caso, foi modificada pelo primeiro desenvolvedor.

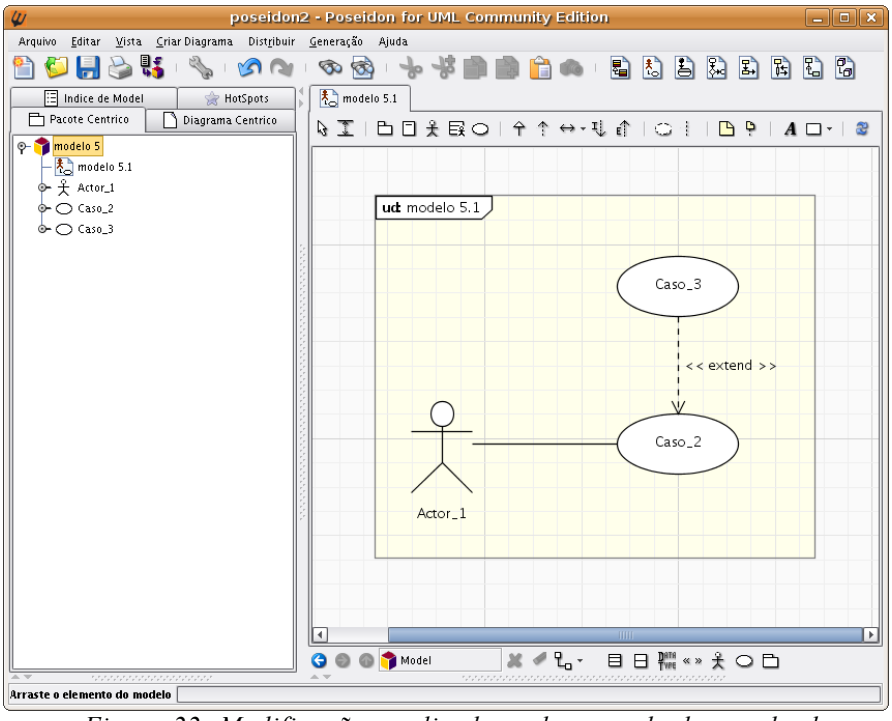

Figura 22: Modificações realizadas pelo segundo desenvolvedor

Após a sincronização, o artefato resultante com as modificações de ambos desenvolvedores é ilustrado pela Figura 23.

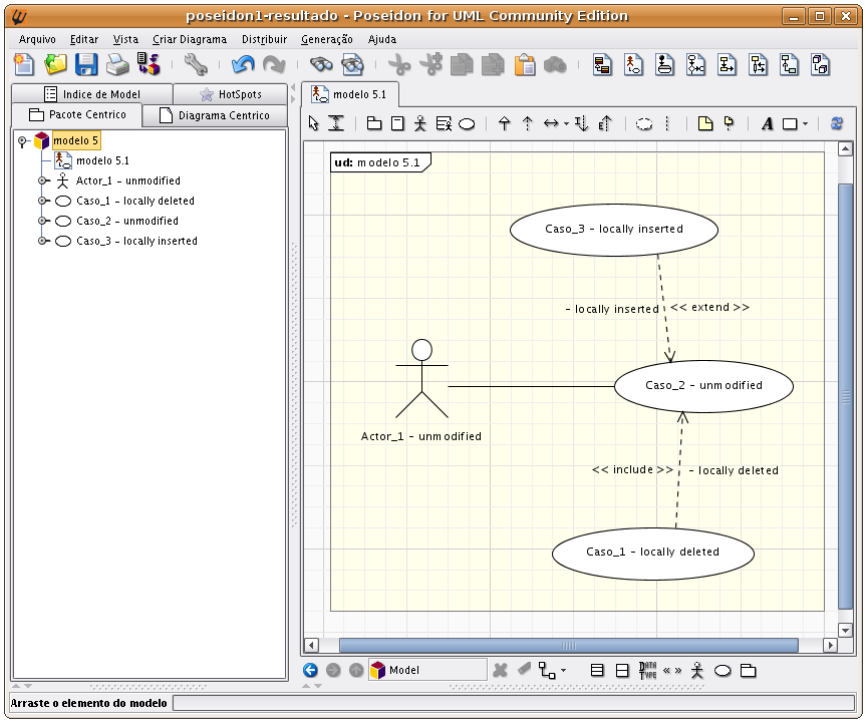

Figura 23: Resultado da junção das versões

Os elementos (atores, *use-case* e associações) podem ser identificados como:

- **unmodified:** significa que não foi realizada modificação alguma nas versões comparadas.
- **locally deleted:** significa que o elemento foi excluído na versão local.
- **remotely deleted:** significa que o elemento foi excluído na versão remota.
- **locally inserted:** significa que o elemento foi inserido na versão local.
- **remotely inserted:** significa que o elemento foi inserido na versão remota.
- **conflict:** significa que o elemento foi modificado nas duas versões e possui divergência entre elas.
- **warning:** significa que existe dois ou mais elementos (use-case, ator e/ ou associação) com o mesmo nome.

Dessa maneira, o desenvolvedor perceberá os elementos que foram modificados no diagrama e poderá aceitar ou não as modificações realizadas.

Para o armazenamento dos artefatos foram utilizados dois tipos de repositórios: Subversion e um SGBD.

1) Utilizando o Subversion como repositório

Na primeira implementação do DiSEN-SCV foi utilizado o Subversion como repositório de artefatos e o servidor de aplicações Apache como meio de comunicação. O Subversion é um Sistema de Controle de Versões (SCV) *open-source* e o armazenamento dos artefatos é centralizado.

O Subversion foi escolhido, inicialmente, por fornecer: versionamento de diretórios, histórico das versões, atomicidade no processo de *commit*, permissão para

realizar *branching* e *merge*. Partindo dessas funcionalidades, o que faltaria para completar a idéia, seria o gerenciamento de artefatos no formato XMI e um mecanismo para tratar os possíveis conflitos, que podem ocorrer durante o desenvolvimento do projeto, para artefatos desse tipo.

A arquitetura do Subversion pode ser visualizada na Figura 24.

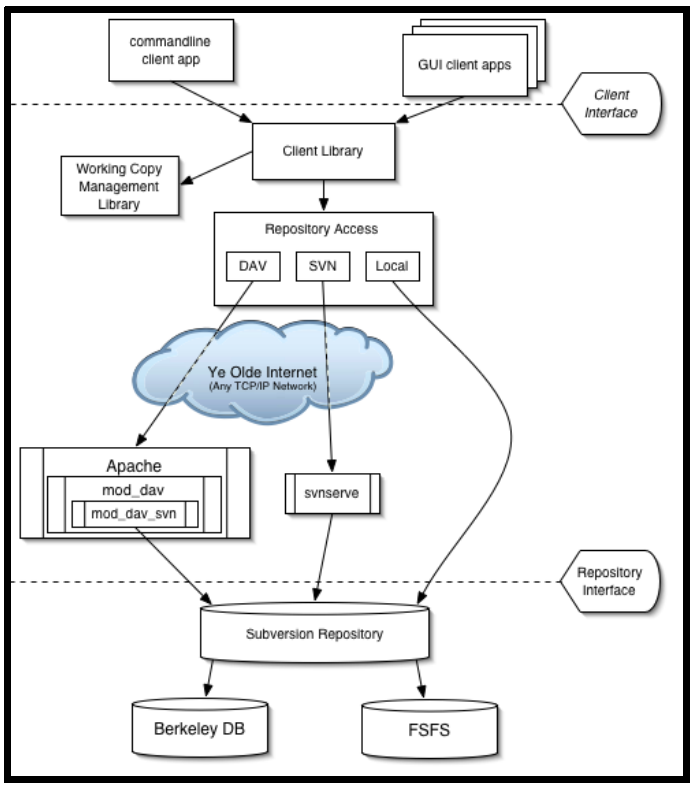

Figura 24: Arquitetura do Subversion
Fonte: COLLINS-SUSSMAN, FITZPATRICK e PILATO (2004)

A arquitetura do Subversion disponibiliza uma interface para acesso ao repositório e uma biblioteca para as aplicações "cliente" acessarem o repositório. A biblioteca fornece três formas para acesso ao repositório: utilizando o servidor de aplicações Apache, o servidor *svnserver* e acesso local.

Além da utilização do Subversion como repositório de artefatos, foi utilizado um SGBD para armazenar os metadados dos artefatos.

O DiSEN possui um módulo para gerenciamento de projetos, que planeja e controla a execução dos projetos. Esse módulo controla: as atividades que deverão ser realizadas, o prazo, as dependências, qual o participante do projeto que irá executar uma atividade, quais os artefatos gerados, entre outras. Mais detalhes podem ser encontrados em (ENAMI, 2006) (SCHIAVONI, 2007).

Além de manter os artefatos associados a uma atividade de projeto, o DiSEN-SCV deve armazenar os metadados dos artefatos. Os metadados são armazenados em um SGBD e, devido à utilização do Subversion, os artefatos são armazenados no repositório do Subversion.

Diante do contexto de utilização do DiSEN-SCV, em um ADDS, essa situação pode gerar problemas de sincronização entre os repositórios, de metadados e de artefatos. Em um ADDS, os repositórios de metadados e de artefatos podem estar em servidores diferentes, portanto, deve existir um mecanismo para garantir o sincronismo desses repositórios (Figura 25).

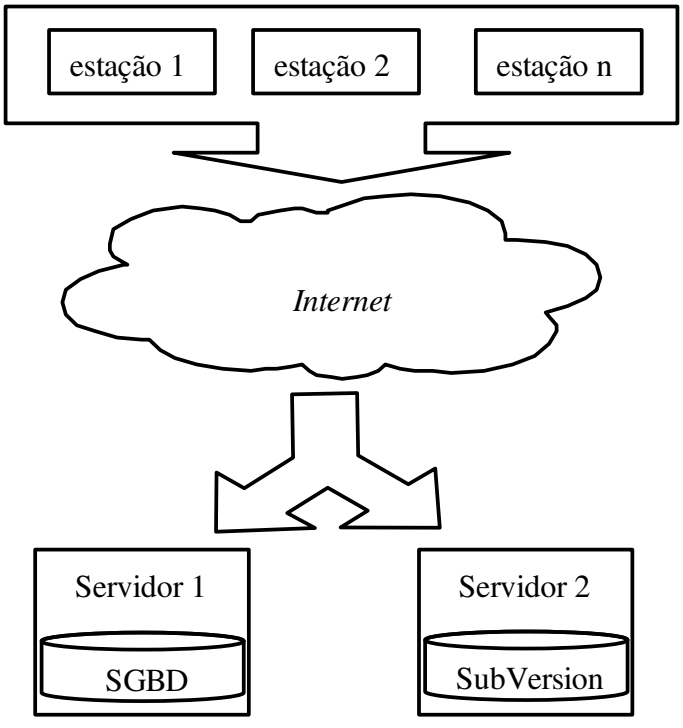

Figura 25: Um exemplo de localização de repositórios em ADDS

Devido ao problema de sincronismo entre os repositórios e também pelo fato de que o Subversion não considera modelos UML, a sua utilização ficou inviabilizada. Assim, foi implementada uma segunda versão para o Gerenciador de Artefatos, utilizando outro repositório de artefatos, que é detalhado a seguir.

2) Utilizando SGBD como repositório

Devido à necessidade de manter sincronizados os artefatos com seus metadados, foi decidido utilizar um único repositório para o armazenamento de ambos. Como os metadados do projeto são armazenados em um SGBD e é necessário, também, armazenar os metadados dos artefatos, foi utilizado o SGBD como repositório de artefatos.

Além do SGBD armazenar os artefatos e seus metadados, ele controla as transações, permitindo a concorrência entre elas. A utilização do SGBD, como repositório de artefatos, possibilitou a configuração de repositórios distribuídos.

No ambiente DiSEN é utilizado o Sequoia para gerenciamento dos SGBDs. Sequoia é um *middleware* que oferece *clustering*, balanceamento de carga e serviços de gerenciamento de falhas. Os *databases* são distribuídos e replicados entre vários *host*, também fornece suporte para manutenção e recuperação *online* (CONTINUENT, 2007).

Um exemplo de aplicação do Sequoia é apresentada na Figura 26.

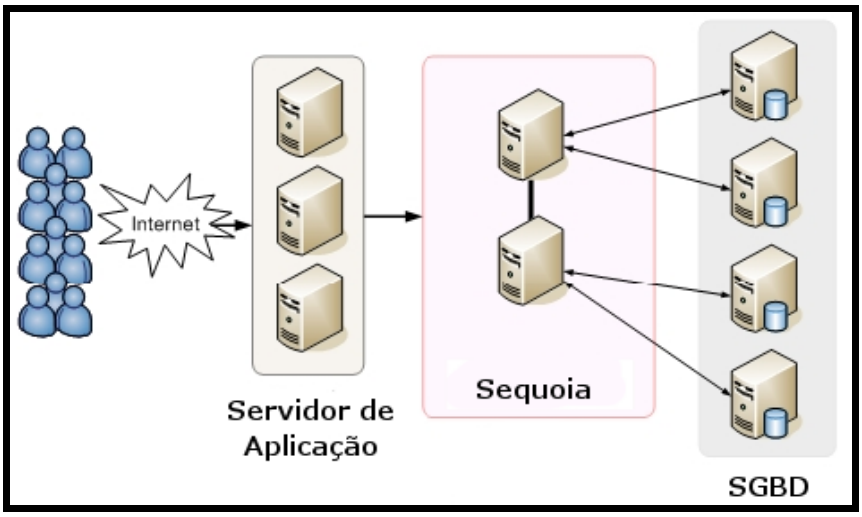

Figura 26: Sequoia
Fonte: CONTINUENT, 2007

Portanto, a utilização do SGBD como repositório de artefatos dentro do ambiente DiSEN e a utilização do *middleware* Sequoia, permitem que os artefatos sejam armazenados em repositórios distribuídos e não ocorrendo problemas durante a sincronização dos artefatos com seus metadados.

A Tabela 1 apresentam as características que foram consideradas na escolha entre a utilização do Subversion e o SGBD como repositório de artefatos.

Tabela 1: Comparação do uso do Subversion e SGBD como repositório de artefatos

	SGBD	**SubVersion**
Versionamento de diretórios	Sim	Sim
Alocação de artefatos por um período de tempo	Sim	Não
Branching	Sim	Sim
Integração ao ambiente DiSEN	Fácil	Difícil
Adição de novos estados para os artefatos (ex. Beta)	Sim	Não
Controle de concorrência	Sim	Sim
Sincronização com os metadados (atividades e artefatos)	Fácil	Difícil

Com base nas dificuldades encontradas para utilizar, dentro do ambiente DiSEN, o Subversion como repositório de artefatos e das características apresentadas na Tabela 1, optou-se em utilizar um SGBD como repositório de artefatos.

4.4 Comparação com os trabalhos relacionados

A Tabela 2 apresenta uma comparação do DiSEN-SCV com alguns trabalhos apresentados no capítulo 3. Os trabalhos que são comparados são identificados como segue: (A) DiSEN-SCV, (B) o sistema OSCAR (NUTTER et al., 2002), (C) ADAMS (BRUEGGE et al., 2006), (D) WebAPSEE (SALES, REIS e LIMA, 2007), (E) Odyssey-VCS (OLIVEIRA, 2005), (F) Gconf (FIGUEIREDO, 2004), (G) Odyssey-CCS (LOPES, MURTA e WERNER, 2006).

Na escolha dos trabalhos levou-se em consideração aqueles que apresentam características de gerenciamento de versões de artefatos. Portanto, os trabalhos, (MEIRE et al., 2003), (BOULILA et al., 2003), (SHUMMMER E SCHUMMER, 2000), (MOLLI et al., 2002), (ZAFFER et al., 2001), (SARMA et al., 2003), (KOBLYLINSHI et al., 2002), (LOPES, 2005), (FROEHLICH e DOURICH, 2004), (EL-JAICK, 2004), (SANTOS, 2003) e (WIESE, 2006), não são comparados.

Tabela 2: Comparativo entre os trabalhos relacionados

	A	B	C	D	E	F	G
Controle de versões	X	X	X	X	X		
Histórico de modificações de versões	X	X	X	X	X	X	X
Notificação de eventos	X	X	X	X	X		
Alocação de artefato com período de tempo	X						
Técnicas de percepção com cores	X						
Merge de artefatos no formato XMI	X						

Os itens avaliados são descritos a seguir:

- **Controle de versões:** nesse item foi observado se as ferramentas controlam as versões dos artefatos.
- **Histórico de modificações:** foi observado se as ferramentas mantêm o histórico das modificações das versões dos artefatos, por exemplo, quem efetuou a modificação, quando, onde e porquê.
- **Notificação:** foi verificado se as ferramentas realizam algum tipo de notificação quando ocorre alguma modificação nos artefatos.
- **Alocação de artefato com período de tempo:** nesse item foi verificado se as ferramentas permitem que os artefatos sejam alocados, por um determinado período de tempo, a um desenvolvedor e que a partir desse momento, mais nenhum outro desenvolvedor consegue realizar modificações no artefato alocado.
- **Técnicas de percepção com cores:** verificou nesse item se as ferramentas implementam técnicas de percepção com a utilização de cores.
- **Merge de artefatos no formato XMI:** nesse item foi verificado se as ferramentas auxiliam os desenvolvedores na resolução de conflitos em artefatos no formato XMI, permitindo que os conflitos sejam resolvidos visualmente na ferramenta CASE.

A comparação apresentada na Tabela 2 mostra a contribuição do DiSEN-SCV em relação aos demais trabalhos, destacando, a alocação do artefato por um período de tempo, a utilização de técnicas de percepção com cores e o suporte para tratamento de artefatos no formato XMI.

4.5 Avaliação

Um estudo da importância e das hipóteses de experimentos que podem ser realizados para avaliação pode ser encontrado em Travassos et al. (2002), sendo os três deles discutidos a seguir:

- **Survey:** determina a distribuição de atributos ou características (descritivo), explicar o porquê de os desenvolvedores terem escolhido uma técnica (explanatória) ou o estudo preliminar para uma investigação mais profunda (explorativa). O survey permite levantar um grande número de variáveis a serem avaliadas (variáveis qualitativas e quantitativas).
- **Estudo de caso:** é utilizado para monitorar projetos, atividades e atribuições. Estudos de caso visam observar um atributo específico e estabelecer relação com atributos diferentes.
- **Experimento:** normalmente, é realizado em laboratório e oferece um nível maior de controle sobre os resultados. O seu objetivo é manipular uma ou algumas variáveis e manter as outras fixas a fim de medir o efeito do resultado. Os experimentos são utilizados para confirmar teorias, confirmar o conhecimento convencional, explorar os relacionamentos, avaliar a predição dos modelos ou as medidas.

Travassos, et al. (2002), explicam que o tipo de experimento mais apropriado em uma determinada situação vai depender dos objetivos do estudo, das propriedades do processo de software utilizado durante a experimentação, ou dos resultados finais esperados com o experimento.

Nesta dissertação optou-se por fazer um estudo de caso qualitativo, tendo em vista a necessidade de observar se as funcionalidades foram atendidas após a implementação dos componentes do DiSEN-SCV. Anteriormente, o ambiente DiSEN

não oferecia suporte para gerenciar as versões dos artefatos produzidos durante o desenvolvimento.

Para realização dos testes foram utilizados cinco computadores, sendo três utilizados como servidores e dois como clientes. Em dois servidores foi instalado o banco de dados PostgreSQL 8.2, no outro servidor foi instalado o Sequoia e uma instância do DiSEN-SERVER, responsável por prover os serviços para o ambiente. Nos computadores clientes foi instalado o DiSEN-CLIENT, que faz acesso aos serviços fornecidos pelo DiSEN-SERVER. A Figura 27 ilustra o ambiente para a realização dos testes.

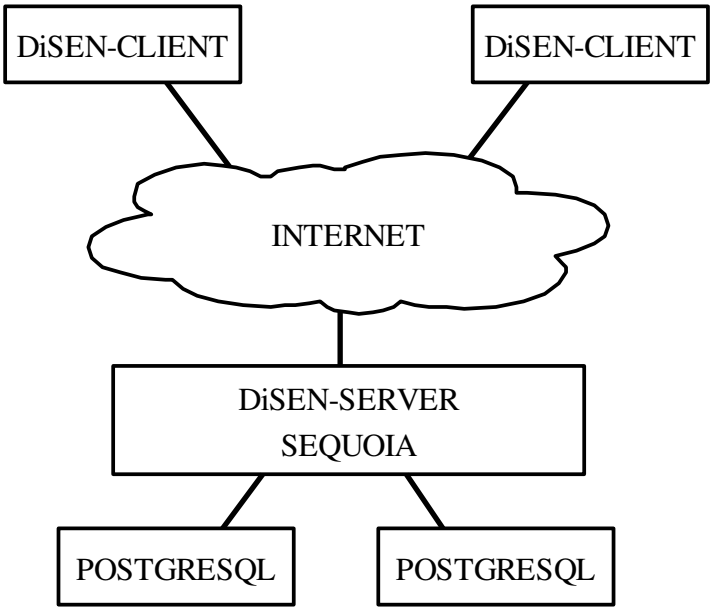

Figura 27: Ambiente para realização dos testes

Os computadores possuem as seguintes configurações:

Servidores: Pentium 4 – 2.8 GHZ, 1.5 GB Ram, 80 GB HD e sistema operacional Linux Ubuntu 7.10.

Clientes: Pentium 4 – 2.8 GHZ, 1.0 GB Ram, 80 GB HD e sistema operacional Linux Ubuntu 7.10.

A seguir são descritos os cenários de aplicação dos testes.

Primeiro cenário: Gerenciamento de versões

No primeiro cenário foram avaliadas as funcionalidades da ferramenta DiSEN-SCV: *checkout, commit* e *update*. Neste cenário a ferramenta deve permitir ao desenvolvedor criar um repositório local (*checkout*); submeter novos artefatos e os artefatos modificados (*commit*); atualizar o repositório local (*update*); e controlar as versões dos artefatos.

Os testes foram realizados com a utilização de artefatos nos formatos texto, binário e XMI. Os artefatos nos formatos binário e XMI utilizam a mesma estrutura para armazenamento, a Figura 28 apresenta essa estrutura.

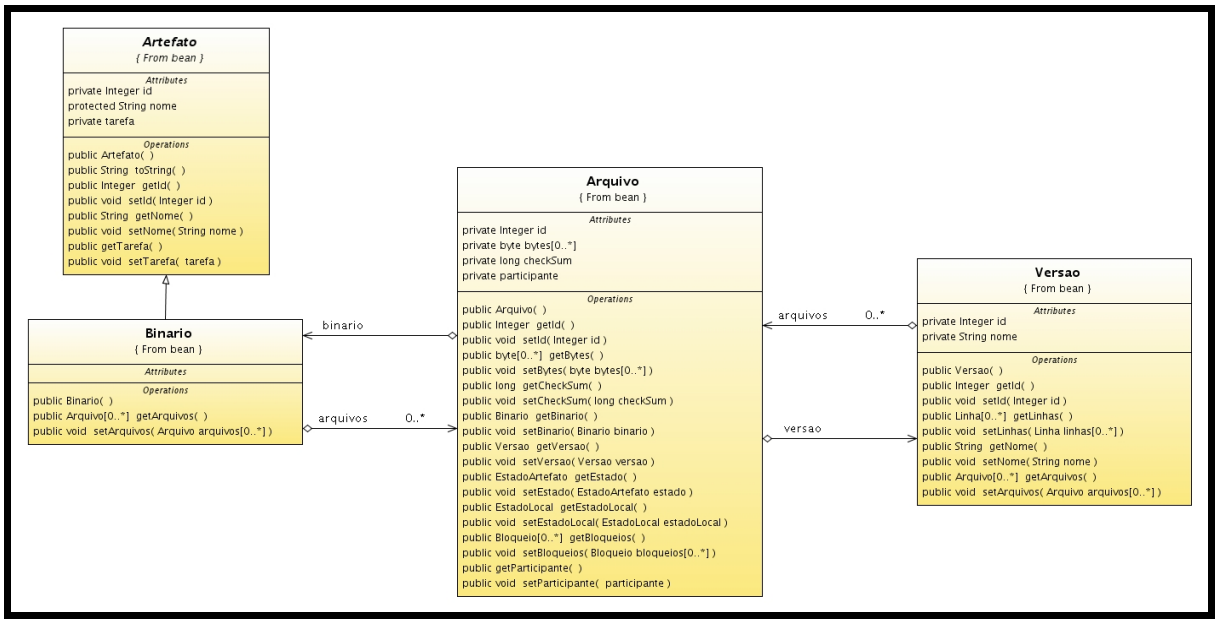

Figura 28: Estrutura para armazenamento de artefatos binários e XMI

Os artefatos no formato texto utilizam uma estrutura para armazenamento diferente dos artefatos binários e XMI. Artefatos nesse formato possibilitam que sejam armazenados em uma granularidade menor, nesse caso são armazenadas as linhas do artefato. Essa estrutura é apresentada na Figura 29.

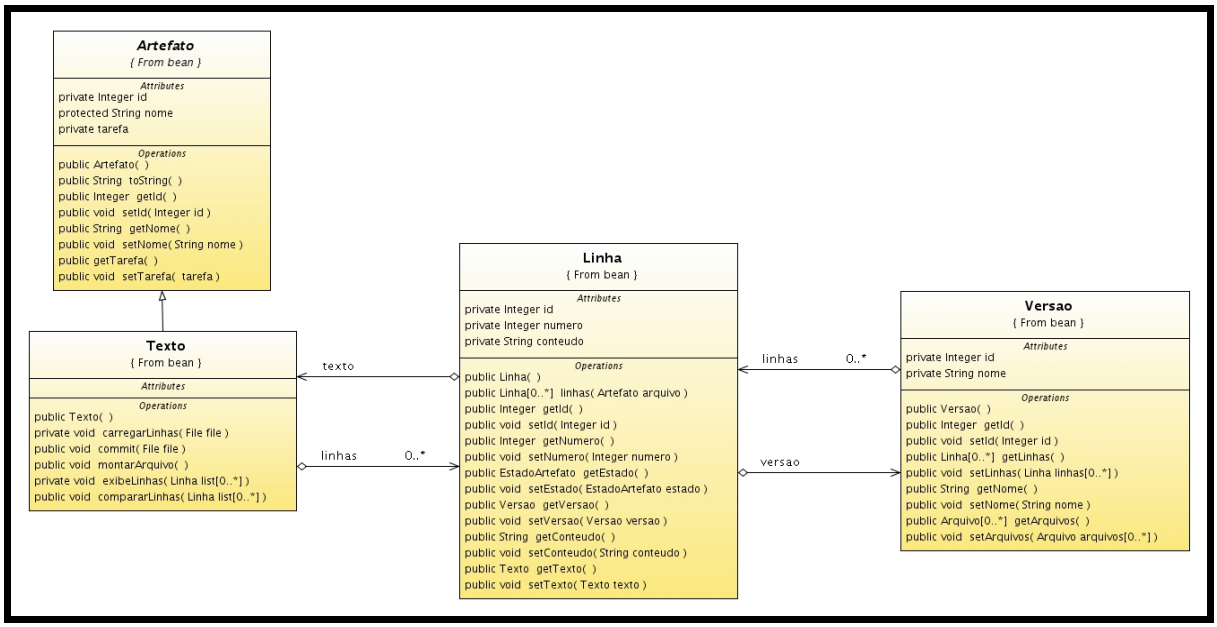

Figura 29: Estrutura para armazenamento de artefatos texto

As funcionalidades de *checkout*, *commit* e *update* foram executadas com êxito para ambos os tipos de artefatos. A associação dos artefatos, com a respectiva atividade, é realizada no momento que um novo artefato é submetido ao repositório central. Essa associação é realizada pelo desenvolvedor.

Segundo cenário: Alocação de artefatos

Neste cenário é avaliada a funcionalidade de alocação de artefatos. O DiSEN-SCV deve permitir ao desenvolvedor requisitar a alocação de qualquer artefato que esteja associado com uma de suas atividades. Caso o artefato esteja alocado para outro desenvolvedor, a solicitação do desenvolvedor é armazenada em uma fila de espera, senão, o artefato é alocado ao desenvolvedor que solicitou a alocação. A alocação do artefato é realizada por um período de tempo determinado pelo gerente de projeto.

Quando o tempo de alocação de um artefato expira ou, quando o desenvolvedor, que está com o artefato alocado, submete o artefato com qualquer modificação, automaticamente, o artefato é alocado para o primeiro da fila de espera e uma notificação é enviada avisando que o artefato está alocado para ele.

O DiSEN-SCV apresenta visualmente ao desenvolvedor os artefatos que estão alocados e distingue os que estão alocados ao próprio desenvolvedor, para outros desenvolvedores e os que não possuem alocação (Figura 20).

A funcionalidade atendeu com sucesso os testes realizados, tanto no gerenciamento de alocação, controle da fila de espera e no envio de aviso de notificação.

Terceiro cenário: Percepção

Neste cenário o DiSEN-SCV foi avaliado quanto às técnicas de percepção implementadas.

O DiSEN-SCV utiliza as técnicas de percepção com cores para distinguir os estados dos artefatos e a utilização de imagens para representar se o artefato esta alocado e, se está alocado para o próprio desenvolvedor ou para outro.

Na Figura 20, percebe-se essas técnicas implementadas, exibindo os artefatos em diferentes cores representando seus estados respectivamente. Na mesma figura, também pode-se observar os artefatos que estão alocados, representados pela imagem de um cadeado.

Outra técnica de percepção utilizada é a de notificação por mensagem do tipo *instant messenger* e por e-mail. Essa técnica é utilizada para avisar um desenvolvedor que está aguardando a alocação de um artefato. É enviado ao desenvolvedor um e-mail de aviso e se ele estiver *on-line* no ambiente, é avisado por envio de mensagem do tipo *instant messenger*.

Também pode ser observado, na Figura 23, a utilização de técnicas de percepção, com o objetivo de avisar os desenvolvedores em relação aos elementos do diagrama que estão divergentes.

A utilização de técnicas de percepção na ferramenta DiSEN-SCV mostrou-se importante para melhorar o trabalho em grupo. A utilização dessas técnicas tem como

objetivo manter os desenvolvedores atentos ou cientes de toda a situação da equipe, dos outros desenvolvedores e dos artefatos envolvidos.

Quarto cenário: Tratamento de conflitos em artefatos no formato XMI

Neste cenário é avaliada a funcionalidade de tratamento de conflitos que podem ocorrem em artefatos no formato XMI. Para a realização dos testes foi utilizada a ferramenta Poseidon for UML Community Edition 6.0.2. Essa ferramenta foi escolhida por ser de fácil acesso, fornecer uma versão sem custo para uso e por exportar os modelos no formato XMI.

Os testes foram realizados com diagramas de *use-case,* aproveitando o modelo definido por Wiese (2006). Inicialmente, foi criado um diagrama de *use-case* e submetido ao repositório central, gerando a primeira versão do artefato. Nesse diagrama procurou-se explorar todos os elementos, atores, *use-case*, associação, *include*, *extends*. A primeira versão do artefato é apresentada pela Figura 30.

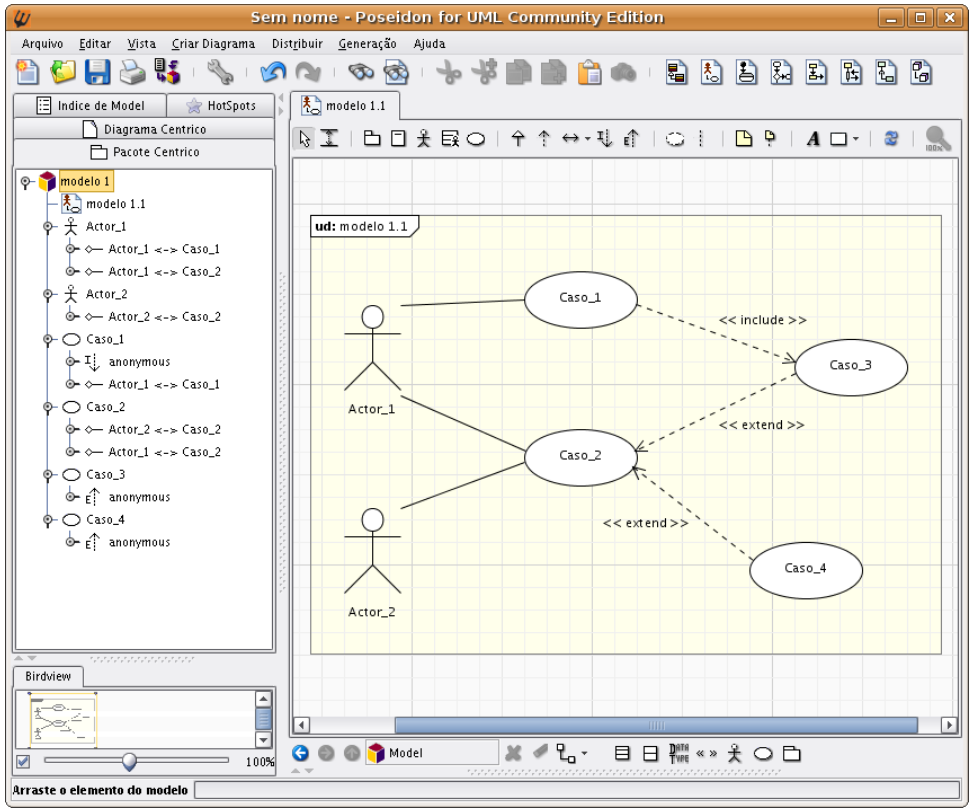

Figura 30: Artefato inicial para realização dos testes

Para a realização dos testes, foi criada uma atividade e dois desenvolvedores foram alocados para executá-la. Nessa atividade foi disponibilizado o artefato da Figura 30 e cada desenvolvedor deveria realizar algumas modificações e enviar o artefato novamente para o repositório central.

O desenvolvedor (A) realizou as seguintes modificações: inclusão de um novo *use-case* (Caso_5), inclusão de um *extend* entre os *use-cases* (Caso_5 e Caso_1), exclusão da associação entre o ator (Actor_1) e o *use-case* (Caso_2), renomeou o *use-case* (Caso_3) para (Caso_3_1) e substituiu o *extend* entre (Caso_3 e Caso_2) por um *include*. Essas modificações são apresentadas na Figura 31.

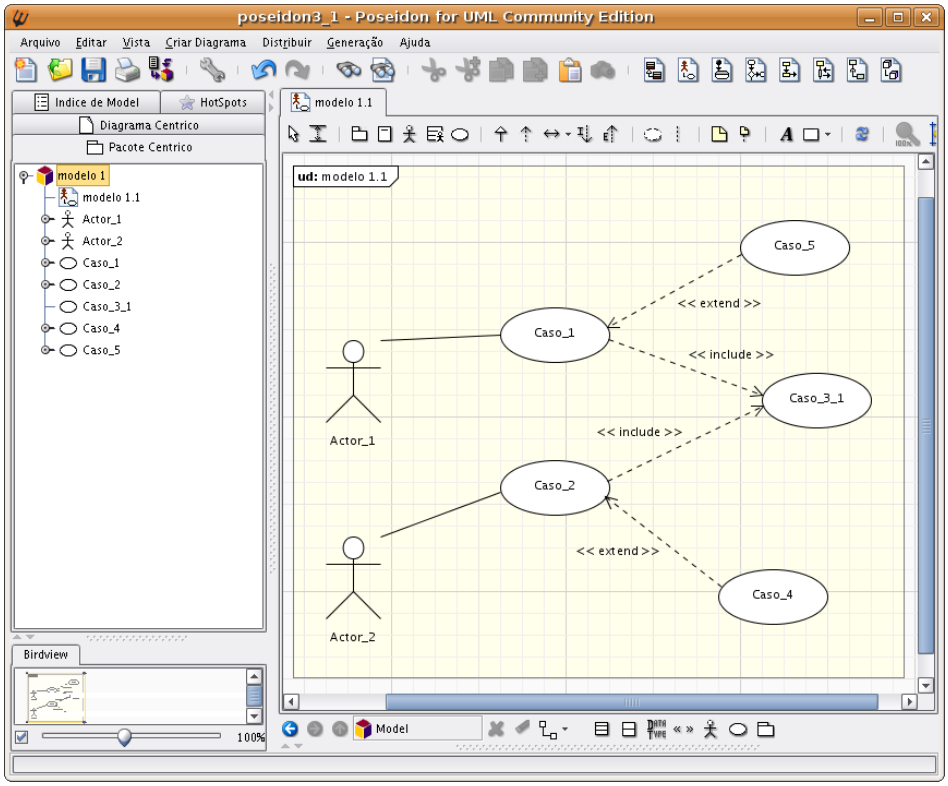

Figura 31: Artefato modificado pelo desenvolvedor (A)

As modificações realizadas pelo desenvolvedor (B) foram: exclusão do *use-case* (Caso_3), inclusão do *use-case* (Caso_6), inclusão de um *include* entre os *use-cases* (Caso_2 e Caso_1), inclusão de um *extend* entre os *use-cases* (Caso_1 e Caso_6),

exclusão da associação entre o ator (Actor_1) e o *use-case* (Caso_2) e inclusão da associação entre o ator (Actor_2) e *use-case* (Caso_1). Essas modificações são apresentadas na Figura 32.

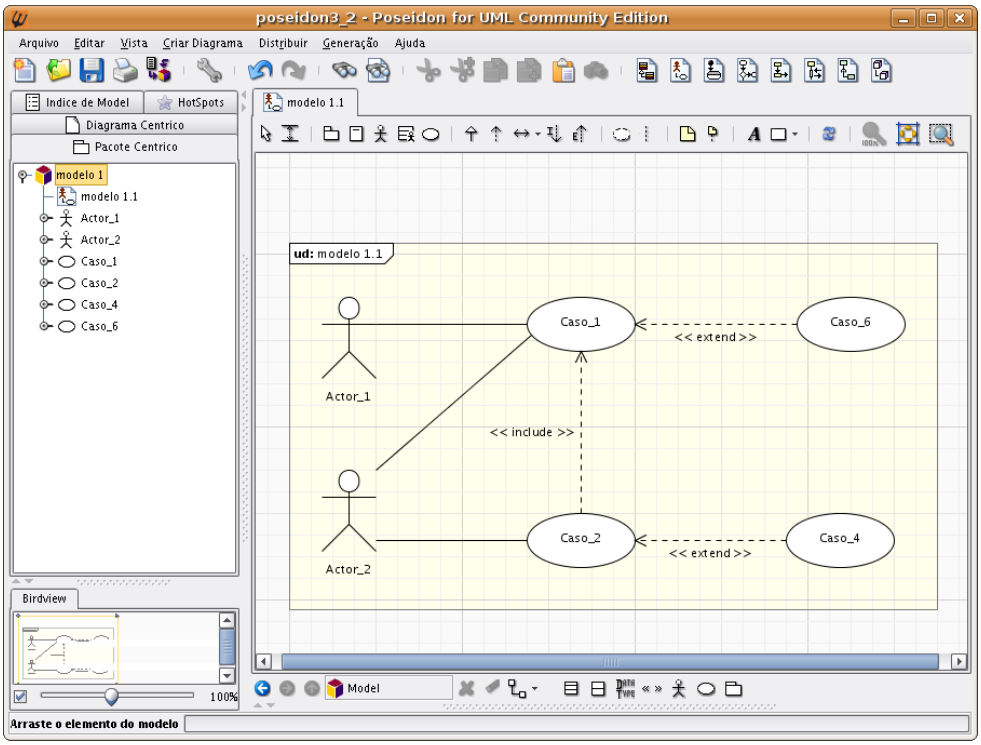

Figura 32: Artefato modificado pelo desenvolvedor (B)

Após as modificações, os desenvolvedores enviaram os artefatos com suas modificações ao repositório central. Somente o primeiro desenvolvedor que enviar o artefato, ao repositório central, conseguirá gerar a segunda versão com suas modificações. O outro desenvolvedor não conseguirá, devido ao fato do artefato se encontrar em uma versão diferente da qual ele possuía em seu repositório local.

Na realização desse teste, o desenvolvedor (A) foi o primeiro a enviar o artefato ao repositório central, portanto o desenvolvedor (B) terá que atualizar o artefato que está no seu repositório local. Nesse caso, após a atualização do artefato, no repositório local do desenvolvedor (B), o artefato atualizado deverá conter as modificações de ambos os desenvolvedores (A e B).

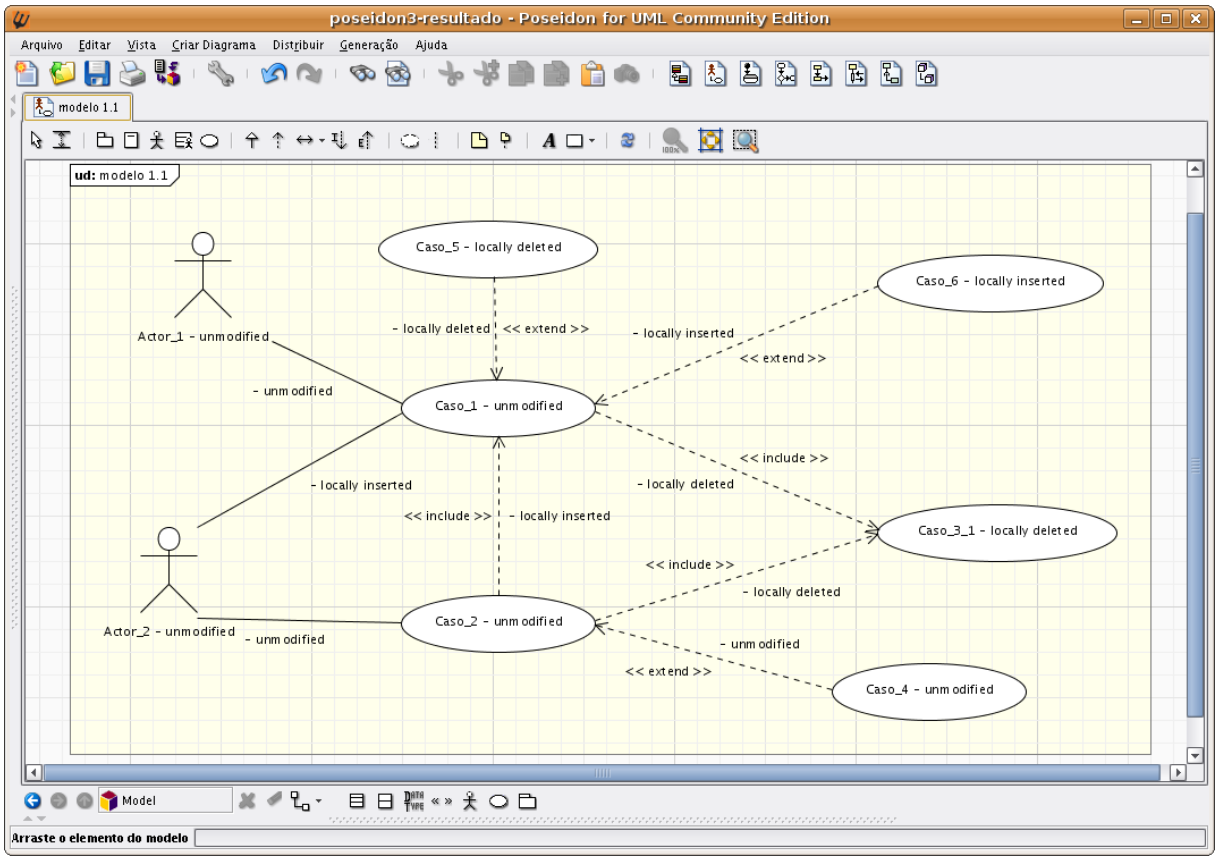

Figura 33: Merge entre as modificações dos desenvolvedores (A e B)

O artefato resultante da junção das modificações dos dois desenvolvedores é apresentado na Figura 33. Nesse artefato, o desenvolvedor consegue identificar os elementos que foram incluídos, excluídos e os que não possuem modificações. Assim, o desenvolvedor (B) teve que analisar cada elemento do diagrama e decidir se ambas as modificações deverão permanecer.

Para destacar a real contribuição dessa funcionalidade, esse mesmo cenário foi executado utilizando o Subversion como repositório de artefatos e o NetBeans 6.0 para facilitar o acesso ao repositório e a visualização do conflito.

Após os desenvolvedores A e B terem realizados suas modificações e o desenvolvedor A ter submetido seu artefato ao repositório central, o desenvolvedor B teve que atualizar o seu repositório local. Quando o artefato foi atualizado, ocorreu conflito entre as modificações de ambos os desenvolvedores.

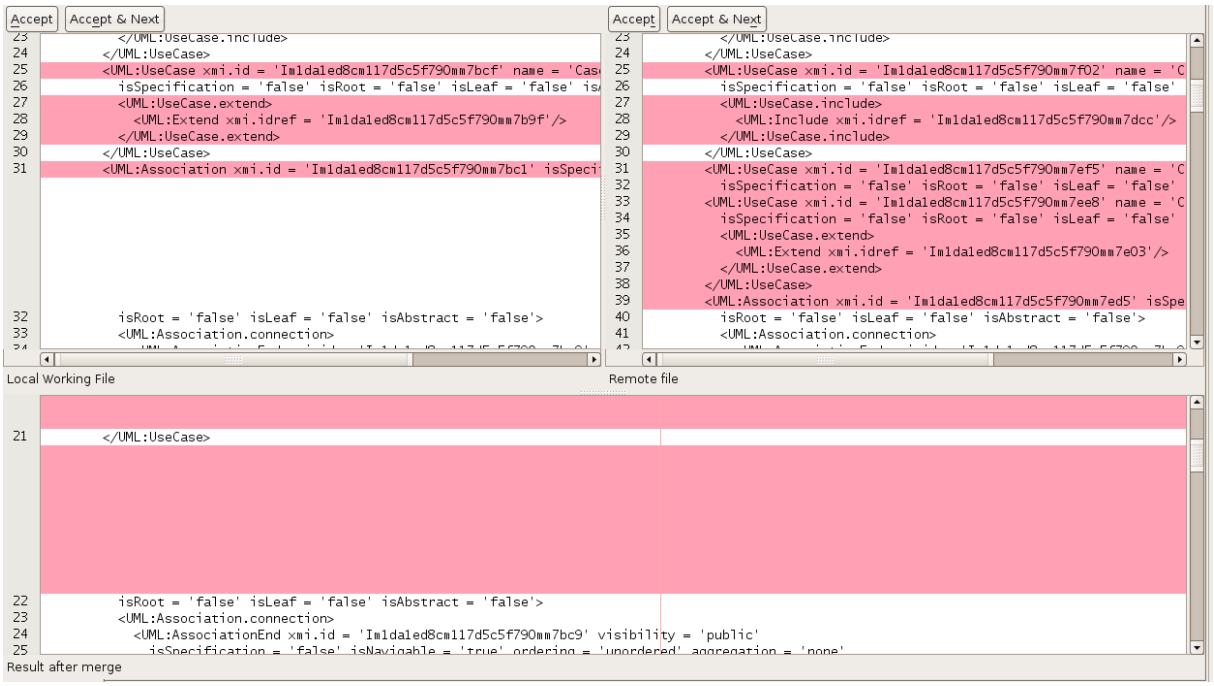

Figura 34: Visualização do conflito utilizando o NetBeans

A Figura 34 ilustra o conflito gerado quando o desenvolvedor B atualizou seu artefato com as modificações do desenvolvedor A. A região que está destacada está em conflito e o desenvolvedor B deverá resolvê-lo. Nesse momento, o desenvolvedor B terá que identificar o que foi alterado por ambos os desenvolvedores e aceitar uma das modificações. Percebe-se que o desenvolvedor B terá maior dificuldade para realizar essa ação, em comparação com a funcionalidade que é oferecida pelo DiSEN-SCV (Figura 33).

Considerações Finais

Os cenários apresentados puderam avaliar o DiSEN-SCV quanto: controle de versões de artefatos, controle de concorrência, controle de alocação, técnicas para melhorar trabalho em grupo e suporte no tratamento de conflitos em artefatos no formato XMI.

Capítulo 5 - Conclusões

Esta dissertação apresenta uma solução para o gerenciamento de versões de artefatos para um ADDS, tratando os possíveis conflitos que podem surgir durante a execução das atividades de desenvolvimento de software, principalmente em artefatos no formato XMI.

Quando dois ou mais desenvolvedores estão trabalhando na mesma atividade e modificando o mesmo artefato, existe uma grande possibilidade, durante a produção do artefato, que ocorra algum conflito, considerando que eles podem estar geograficamente dispersos.

Existem diversas ferramentas que tratam conflitos em artefato de código-fonte, como CVS, Subversion e as apresentadas no Capítulo 3. Entretanto, os artefatos no formato XMI são de difícil entendimento para o desenvolvedor, por serem produzidos por ferramentas CASE. O DiSEN-SCV auxilia o desenvolvedor na resolução de conflitos em artefatos desse tipo, permitindo que ele visualize, na própria ferramenta CASE, o que está divergente entre duas versões do artefato.

Para prevenir a ocorrência de conflitos, o DiSEN-SCV implementa técnicas de percepção, distinguindo os estados dos artefatos (modificado, incluído, excluído, alocado, etc.) com a utilização de cores e enviando avisos de notificações aos desenvolvedores, quando ocorre uma alteração no artefato. Por exemplo, quando uma nova versão do artefato é criada, por um desenvolvedor, os demais desenvolvedores, envolvidos na mesma atividade, são avisados sobre a nova versão. Dessa maneira, os desenvolvedores estão sempre com informações atualizadas em relação às modificações ocorridas nos artefatos da atividade que está realizando.

Uma outra medida utilizada para evitar a ocorrência de conflitos é a alocação de artefatos. Quando um artefato é alocado para um desenvolvedor, somente ele pode

submeter modificações para o repositório central. Entretanto, a alocação de artefatos diminui o paralelismo na execução das atividades. Portanto, o DiSEN-SCV utiliza a alocação de artefatos por um período de tempo determinado. O tempo de alocação é estipulado pelo gerente de projeto, de acordo com o grau de dificuldade para a execução da atividade e o conhecimento do desenvolvedor.

O controle de alocação de artefatos por um período de tempo, evita que a desalocação do artefato seja realizada somente quando o desenvolvedor desejar, permitindo melhor equilíbrio no tempo de alocação do artefato entre os membros de uma equipe.

A percepção também é utilizada na alocação dos artefatos. Os artefatos são identificados com a imagem de um cadeado. Quando um artefato não está alocado, ele é identificado com a imagem de um cadeado aberto, quando ele está alocado para o próprio desenvolvedor, ele é identificado com a imagem de um cadeado fechado e na cor vermelha e, quando o artefato está alocado para outro desenvolvedor, ele é identificado com a imagem de um cadeado fechado e na cor azul (Figura 20).

Os artefatos são armazenados em granularidade diferentes de acordo com seu tipo. Artefatos no formato texto são divididos por linhas, para permitir uma granularidade mais fina. Os artefatos nos formatos binários e XMI são armazenados por inteiro, mas como existe um modelo para os artefatos no formato XMI, eles são comparados a nível de elementos, no momento de tratamento de conflitos.

Inicialmente, esses foram os níveis de granularidades implementados, mas pode-se obter granularidade mais fina para os artefatos no formato XMI. Artefatos desse tipo podem ser armazenados com base no modelo utilizado para o tratamento de conflitos, que foi definido por Wiese (2006).

A utilização de um SGBD como repositório de artefatos, possibilitou algumas vantagens:

- o ambiente DiSEN possui um mecanismo de persistência que está

consolidado e integrado ao *middleware* do ambiente. Portanto, facilitou o uso do SGBD como repositório de artefatos.

- o uso do SGBD garante o controle de concorrência das transações.
- o armazenamento dos artefatos no mesmo repositório de seus metadados, facilitou o sincronismo entre os mesmos.
- a utilização do *middleware* Sequoia permitiu o uso de repositórios distribuídos.

Como desvantagens da utilização do SGBD, em relação a utilização do Subversion, pode-se citar:

- necessidade de implementar o controle de versões de artefatos, mantendo o histórico de atualizações e versionamento de diretórios, ao invés de utilizar o controle do Subversion.
- necessidade de implementar o algoritmo de *merge* para artefatos no formato texto, ao invés de utilizar o algoritmo do Subversion.

A concretização dessa dissertação contribui em relação ao desenvolvimento distribuído de software, permitindo:

- melhor disseminação das informações entre os desenvolvedores, com a utilização de técnicas de percepção;
- maior disponibilidade de acesso aos repositórios de artefatos, devido a existência de repositórios distribuídos;
- melhor equilíbrio no tempo de alocação dos artefatos entre os desenvolvedores.

Pode-se destacar como principais contribuições do DiSEN-SCV:

(1) o controle na alocação de artefatos por um período de tempo determinado;

(2) o agrupamento das modificações de duas versões de um artefato no formato XMI;

(3) a visualização gráfica, na ferramenta CASE, do que foi modificado em um artefato no formato XMI;

(4) a implementação de técnicas de percepção provendo o desenvolvimento de software com mais qualidade;

(5) a replicação dos repositórios de artefatos.

Dificuldades encontradas

Em se tratando de um ADDS, onde podem existir diversos servidores provendo um ou mais serviços, a maior dificuldade foi em garantir a sincronização dos artefatos com os metadados das atividades do projeto, conforme explicado na Seção 4.3.

Trabalhos Futuros

A realização deste trabalho de pesquisa levou à construção de um gerenciador de artefatos para ADDS. A sua implementação abre novas perspectivas de pesquisa, que podem ser exploradas em trabalhos futuros. Alguns desses trabalhos são apresentados a seguir:

- O tratamento de conflitos em artefatos no formato texto e o versionamento de diretórios não foram implementados, devido a existência de várias ferramentas que tratam desses problemas. Para a adição dessas funcionalidades pode-se construir um módulo ou integrar uma ferramenta existente para resolver esse tipo de conflito.
- A resolução de conflitos em artefatos no formato XMI foi implementada somente para diagramas de *use-case*. Dessa forma, pode-se criar novos modelos para representar outros artefatos, por exemplo, diagrama de classes, diagrama de seqüências, diagrama de estados, documentos de requisitos, entre outros.

- O DiSEN-SCV armazena o histórico de modificações dos artefatos, portanto, pode-se criar um módulo para explorar esses dados, extraindo informações úteis para o gerente de projeto e até mesmo para o desenvolvedor, por exemplo, pode-se estimar a produtividade de um desenvolvedor em relação à quantidade de versões de artefatos que ele produziu.
- Implementar um sistema de gerenciamento de modificações, permitindo assim o controle da evolução do desenvolvimento de software. Esse sistema dever permitir o controle do estado das modificações e o relacionamento entre as modificações e os artefatos.
- Integração do DiSEN-SCV com a ferramenta VIMEE (GUILHERMINO, 2007). A ferramenta VIMEE permite o agendamento e gerenciamento de uma reunião em um ADDS. Durante a reunião, os membros podem analisar, modificar e propor alterações em relação os artefatos que foram produzidos pelos desenvolvedores. Por isso é necessário que haja integração entre o DiSEN-SCV e a ferramenta VIMEE.

Referências Bibliográficas

ARAÚJO, R. M.; DIAS, M. S.; BORGES, M. R. S. Suporte por Computador ao Desenvolvimento Cooperativo de Software: Classificação e Propostas. In: XI SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 1997, Fortaleza, **Anais...** p. 299–314.

BATISTA, S. M. **Uma ferramenta de apoio à fase de requisitos da MDSODI no contexto do ambiente DiSEN**. 2003. 83 f. Dissertação de Mestrado – Programa de Pós- Graduação em Informática, Universidade Federal do Paraná, Curitiba, 2003.

BORGES, M. R. S.; CAVALCANTI, M. C. R.; CAMPOS, M. L. M. Suporte por computador ao trabalho cooperativo. In: XV JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, 1997, Canela, RS, **Anais...** Canela: Congresso Nacional da SBC, 1997.

BORGHOFF, U. M.; SCHLICHTER, J. H. **Computer-Supported Cooperative Work: Introduction to Distributed Applications.** Springer-Verlag New York, USA. 2000. 544 p. ISBN: 3540669841.

BOULILA, N.; DUTOIT, A. H.; BRUEGGE, B. D-Meeting: An Object-Oriented Framework for supporting distributed modelling of software, ICSE 2003. International Workshop on Global Software Development. p. 34-38, Maio, EUA. 2003.

BRODSKY, S. A. XMI Opens Application Interchange. 1999. Disponível em: <http://www-306.ibm.com/software/awdtools/standards/xmiwhite0399.pdf>. Acesso em 15 de abril 2007.

BRUEGGE B. et al. Supporting Distributed Software Development with fine-grained Artefact Management. In: IEEE INTERNATIONAL CONFERENCE ON GLOBAL SOFTWARE ENGINEERING, 2006, Washington. **Proceedings...** Washington, 2006. 213-222.

CARMEL, E. Global Software Teams – Collaborating Across Borders and Time-Zones. Prentice Hall, USA, 1999, 269p.

CARRIERO, N.; GELERNTER, D. Linda in context. In: COMMUNICATIONS OF THE ACM, 1989, New York, USA, vol. 32, n. 4, p. 444-458. 1989.

CHRISSIS, M. B.; KONRAD, M.; SHRUM, S. CMMI: Guidelines for Process Integration and Product Improvement, Boston, MA, Addison-Wesley, 2003.

CHRISTENSEN, M. J.; THAYER, R. H. The Project Manager's Guide to Software

Engineering's Best Practices. In: IEEE COMPUTER SOCIETY PRESS AND JOHN WILEY & SONS. 2002.

COLLINS-SUSSMAN, B.; FITZPATRICK, B. W.; PILATO, C. M. **Version Control with Subversion.** Editora O`REILLY. 1ª edição. 2004. 319 p. ISBN 10: 0-596-00448-6 | ISBN 13:9780596004484.

CONRADI, R.; WESTFECHTEL, B. Version Models for Software Configuration Management. In: ACM COMPUTING SURVEYS, vol. 30, June, 1998. p. 232-282.

CONTINUENT, Middleware Sequoia. Disponível em: http://sequoia.continuent.org. Acesso em 12/10/2007.

DART, S. Concepts in Configuration Management Systems. In: INTERNATIONAL WORKSHOP ON SOFTWARE CONFIGURATION MANAGEMENT, 1991, p. 1-18, Trondheim, Norway, 1991.

DOURISH, P.; BELLOTTI, V. Awareness and Coordination in Shared Workspaces. In: ACM CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK, 1992, Toronto, Ontario, Canada, Proceedings... p. 107-114. 1992.

EL-JAICK, D. **MIMIX: Sistema de Apoio à Modelagem Cooperativa de Software Utilizando Ferramentas CASE.** 2004. Dissertação de Mestrado, COPPE, UFRJ, Rio de Janeiro, RJ, Brasil. 2004.

ELLIS, C. A.; GIBBS, S. J.; REIN, G. L. Groupware - Some Issues and Experiences. In: COMMUNICATIONS OF THE ACM, 1991. p., 38-58.

ENAMI, L. N. M. **Um Modelo de Gerenciamento de Projetos para um Ambiente de Desenvolvimento Distribuído de Software.** 2006. 217 p. Dissertação de Mestrado – Programa de Pós-Graduação em Ciência da Computação, Universidade Estadual de Maringá, Maringá, 2006.

ESTUBLIER, J. Software Configuration Management: a Roadmap. In: 22nd INTERNATIONAL CONFERENCE ON SOFTWARE ENGENEERING, 2000, Limerick, Ireland. **Proceedings...** Limerick, Ireland, 2000. p. 279-289.

ESTUBLIER, J.; LEBLANG, D.; CLEMM, G.; CONRAD, R.; TICHY, W.; HOEK, A. W. D. Impact of the research community on the field of software configuration management: summary of an impact project report. ACM SIGSOFT Software Engineering Notes, v.27 n.5, September, 2002.

ESTUBLIER, J.; LEBLANG, D.; VAN DER HOEK, A. et al. Impact of Software Engineering Research on the Practice of Software Configuration Management, ACM Transactions on Software Engineering and Methodology (TOSEM), v. 14, n. 4 (October), 2005. p. 1-48.

FIGUEIREDO, S.; SANTOS, G.; ROCHA, A. R. Gerência de Configuração em Ambientes de Desenvolvimento de Software Orientados a Organização. In: SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, 2004, Brasília. **Anais...** Brasília, 2004. ISBN 858844276-0.

FREITAS, A.; MAIA, A.; NUNES, D. Um Modelo para Interação entre Processos de Software. In: CONGRESSO BRASILEIRO DE COMPUTAÇÃO, CBCOMP, 4, 2004, Itajaí. **Anais...** Itajaí: Univali, 2004. p. 149-154.

FROEHLICH, J.; DOURISH, P. Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE 2004, Edinburgh, UK. **Procedings...** Edinburgh, UK, 2004, p. 387-396.

FUKS, H.; RAPOSO, A.; GEROSA, M. Do modelo 3C à Engenharia de Groupware. 2003. Disponível em: <www.tecgraf.puc-rio.br/publications>. Acesso em 25 março de 2007.

GUILHERMINO, D. F. et al. Um modelo para gerenciar a comunicação em um ambiente distribuído de desenvolvimento de software. In: XIII CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACION, Corrientes e Resistencia – Argentina. Outubro de 2007.

GUTWIN, C.; GREENBERG, S. A Descriptive Framework of Workspace Awareness for Real-Time Groupware, Journal of Computer Supported Cooperative Work, v. 11, n. 3, 2002. p. 411-446.

HASS, A. M. J. Configuration Management Principles and Practices, Boston, MA, Pearson Education, Inc. 2003.

HERBSLEB, J. D.; MOITRA, D. Guest Editors' Introduction: Global Software Development, IEEE Software, vol. 18, no. 2, March/April, 2001. p. 16-20.

HUZITA, E. H. M. **Uma Metodologia par a Desenvolvimento Baseado em Objetos Distribuídos Inteligentes.** Projeto de pesquisa, Universidade Estadual de Maringá, Departamento de Informática, 1999.

IEEE, Std 1042 - IEEE Guide to Software Configuration Management, Institute of Electrical and Electronics Engineers. 1987.

IEEE, Std 610.12 - IEEE Standard Glossary of Software Engineering Terminology, Institute of Electrical and Electronics Engineers. 1990.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. The Unified Software Development Process. Addison-Wesley, 1999.

KIEL, L. Experiences in Distributed Development: A Case Study, In: WORKSHOP ON

GLOBAL SOFTWARE DEVELOPMENT. ICSE 2003, Oregon, EUA. 2003.

KOBYLINSKI, R.; CREIGHTON, O.; DUTOIT, A.; BRUEGGE, B. Building awareness in distributed software enginering: Using issues as context. In: INTERNATIONAL WORKSHOP ON DISTRIBUTED SOFTWARE DEVELOPMENT, Orlando, EUA, 2002.

LANUBILE, F.; DAMIAN, D.; OPPENHEIMER, H. L. Global Software Development: Technical, Organizational, and Social Challenges. ACM SIGSOFT Software Engineering Notes. Volume 28. n. 6, November, 2003.

LEON, A. A Guide to Software Configuration Management Norwood, MA, Artech House Publishers. 2000.

LOPES, L. G. B.; MURTA, L. G. P.; WERNER, C. M. L. Odyssey-CCS: Uma ferramenta flexível para o controle de modificações em software, In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE. Sessão de Ferramentas, Florianópolis, SC, Brasil, outubro 2006.

LOPES, L. G. B.; MURTA, L. G. P.; WERNER, C. M. L. Odyssey-CCS: A Change Control System Tailored to Software Reuse. In: INTERNATIONAL CONFERENCE ON SOFTWARE REUSE. (ICSR), Torino, Italy, June. p. 170-183.

MAIR, Q. Technical Issues in the Design of a Virtual Software Corporation. ECSCW'97 OOGP workshop. 1997.

MEIRE, A. P.; BORGES, M. R. S.; ARAÚJO, R. M. Supporting Collaborative Drawing with the Mask Versioning Mechanism. Lecture Notes in Computer Science, Vol. 2806, Berlim, Alemanha. 2003. p. 208-223.

MOLLI, P.; SKAF-MOLLI, H.; OSTER, G.; JOURDAIN, S. SAMS: Synchronous, Asynchronous, Multi-Synchronous Environments. In: CONF. ON COMPUTER SUPPORTED COOPERATIVE WORK IN DESIGN. CSCWD´2002, Rio de Janeiro, Brasil, 2002. p. 80-84.

MORANDINI, M. Critérios e Requisitos para Avaliação da Usabilidade de Interfaces em Groupware – CSCW. Faculdade de Engenharia Elétrica e de Computação – FEEC. Unicamp. 1998.

MURTA, L. G. P. **Gerência de Configuração no Desenvolvimento Baseado em Componentes.** 2007. 229 p. Tese de Doutorado. COPPE, UFRJ. Rio de Janeiro, Brasil. 2007.

NUTTER, D. et al. An Artefact Repository to Supported Distributed Software Engineering, In COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, 2002. **Proceedings...** 26th Annual International, 2002, p. 1081-1086.

ODYSSEY, Projeto Odyssey. 2004. Disponível em: <http://www.cos.ufrj.br/~odyssey>.

Acesso em 05 de fevereiro de 2007.

OLIVEIRA, H. L. R. **Odyssey-VCS: Uma Abordagem de Controle de Versões para Elementos da UML.** 2005. 104 p. Dissertação de Mestrado, COPPE, UFRJ, Rio de Janeiro, Brasil. 2005.

OMG XMI. XMI Specification 2.1. 2005. Disponível em: <http://www.omg.org/technology/documents/formal/xmi.htm>. Acesso em 27 de Março de 2007.

OMG, Unified Modeling Language (UML) Specification, version 1.4, formal/01-09-67, Object Management Group. 2001.

OMG, XML Metadata Interchange (XMI) Specification, version 1.2, formal/02- 01-01, Object Management Group. 2002.

PASCUTTI, M. C. D. **Uma proposta de arquitetura de um ambiente de desenvolvimento de software distribuído baseado em agentes.** 2002, 102 f. Dissertação (Mestrado) - Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre.

PINHEIRO, M.K.; LIMA, J. V.; BORGES, M. R. S. Awareness em Sistemas de Groupware. In: IDEAS, 2001, San Jose, Costa Rica. **Proceedings...** Costa Rica, 2001, p. 323-335.

POZZA, R. S. **Proposta de um Modelo para Cooperação baseado no Gerenciador de Workspace no Ambiente DiSEN.** 2005, 97 p. Dissertação de Mestrado – Programa de Pós-Graduação em Ciência da Computação, Universidade Estadual de Maringá, Maringá, 2005.

PRESSMAN, R. S. **Software Engineering: A Practitioner's Approach.** McGraw-Hill International Editions, 5th ed. 2001.

PRIKLADNICKI, R.; LOPES, L.; AUDY, J. L. N.; EVARISTO, R. Desenvolvimento Distribuído de Software: Um Modelo de Classificação dos Níveis de Dispersão dos Stakeholders. In: I SBSI – SIMPÓSIO BRASILEIRO DE SISTEMAS DE INFORMAÇÃO, 2004, Porto Alegre. p. 253-262. 2004.

SALES, E. O.; REIS, C. A. L.; LIMA, A. M. Gestão de Configuração integrada a Gerência de Processos de Software no Ambiente WebAPSEE, In: CONFERENCIA LATINOAMERICANA DE INFORMÁTICA, 2007, Costa Rica. **Proceedings...** Costa Rica, 2007.

SANTOS, V. V. **ARIANE: Um Mecanismo de Apoio à Percepção em Bases de Dados Compartilhadas.** 2003. 114 p. Dissertação de Mestrado. COPPE/UFRJ. Rio de Janeiro, Brasil. 2003.

SARMA, A.; NOROOZI, Z.; VAN DER HOEK, A. Palantír: Raising Awareness among Configuration Management Workspaces. In: 25th INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2003, EUA. **Proceedings...** EUA, 2003. p. 444-454.

SCHIAVONI, F. L. **FRADE – Framework para infra-estrutura de um Ambiente Distribuído de Desenvolvimento de Software**. 2007. 181 p. Dissertação de Mestrado – Programa de Pós-Graduação em Ciência da Computação, Universidade Estadual de Maringá, Maringá, 2007.

SCHUMMER, T.; SCHUMMER, J. Support for Distributed Teams in eXtreme Programming. In: EXTREME PROGRAMMING AND FLEXIBLE PROCESSES SOFTWARE ENGINEERING, 2001, Addison-Wesley Longman Publishing Co., Boston, MA, USA, 2001.

SOARES, M. D. **Gerenciamento de Versões de Páginas Web**. 2000. 114 p. Dissertação de Mestrado do ICMC-USP, Universidade de São Paulo, São Carlos, 2000.

SOFTEX, MPS.BR - Melhoria de Processo do Software Brasileiro - Guia Geral (Versão 1.1), Associação para Promoção da Excelência do Software Brasileiro. 2006.

TICHY, W. RCS: a system for version control, Software - Practice and Experience, v. 15, n. 7 (July), 1985. p. 637-654.

TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. A. G. G. Introdução à engenharia de software experimental. Relatório Técnico RT-ES-590/02. Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 2002. 52 p.

VILLELA, K.; OLIVEIRA, K. M.; SANTOS, G.; ROCHA, A. R. C.; TRAVASSOS, G. H. Cordis-FBC: an Enterprise Oriented Software Development Environment In: WORKSHOP LEARNING SOFTWARE ORGANIZATION, 2003, Luzern.

WERNER, C. M. L.; MANGAN, M. A. S.; MURTA, L. G. P.; et al., OdysseyShare: an Environment for Collaborative Component-Based Development. In: IEEE CONFERENCE ON INFORMATION REUSE AND INTEGRATION, 2003, Las Vegas, USA, October, p. 61-68, 2003.

WIESE, I. S. **Um Modelo de Interoperabilidade para Ambientes de Desenvolvimento Distribuído de Software**. 2006. 90 f. Dissertação de Mestrado – Programa de Pós- Graduação em Ciência da Computação, Universidade Estadual de Maringá, Maringá, 2006.

ZAFFER, A.; SHAFFER, C.; EHRICH, R.; PEREZ, M. NetEdit: A Collaborative Editor, TR-01-13, Computer Science, Virginia Tech, EUA. 2001.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo