ULBRA CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Recredenciado pela Portaria Ministerial nº 3.607 - D.O.U. nº 202 de 20/10/2005

Leandro Oliveira Ferreira

DESENVOLVIMENTO DE MIDDLEWARE PARA INTEGRAÇÃO DO REPOSITÓRIO COMPWEB COM SISTEMAS DE CONTROLE DE VERSÃO

Palmas
2010

ULBRA CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Recredenciado pela Portaria Ministerial nº 3.607 - D.O.U. nº 202 de 20/10/2005

Leandro Oliveira Ferreira

DESENVOLVIMENTO DE MIDDLEWARE PARA INTEGRAÇÃO DO REPOSITÓRIO COMPWEB COM SISTEMAS DE CONTROLE DE VERSÃO

"Monografia apresentada como requisito da disciplina Trabalho de Conclusão de Curso II (TTC II) do curso de Sistemas de Informação, orientado pelo Professor *Msc*. Michael Schuenck dos Santos".

Palmas
2010

Leandro Oliveira Ferreira

DESENVOLVIMENTO DE MIDDLEWARE PARA INTEGRAÇÃO DO REPOSITÓRIO COMPWEB COM SISTEMAS DE CONTROLE DE VERSÃO

"Monografia apresentada como requisito da disciplina Trabalho de Conclusão de Curso II (TTC II) do curso de Sistemas de Informação, orientado pelo Professor *MSc*. Michael Schuenck dos Santos".

BANCA EXAMINADORA

_____

Prof. *MSc*. Michael Schuenck dos Santos

_____

Prof. *MSc*. Jackson Gomes de Souza

_____

Prof. *MSc*. Fernando Luiz de Oliveira

Palmas
2010

DEDICATÓRIA

Dedico esse trabalho de Conclusão de Curso ao Grande Arquiteto do Universo, que com sua imensa misericórdia me brindou com o dom da vida; aos meus pais, que não mediram esforços para que mais esta etapa fosse concretizada e a Gabriela, meu grande amor, que nunca deixou de me apoiar.

AGRADECIMENTO

Agradeço a Deus por todas as bênçãos derramadas sobre a minha vida, por estar comigo a cada segundo, por permitir que eu concluísse mais essa etapa da minha caminhada.

Aos meus pais por serem exemplos para mim e por me apoiarem em cada momento difícil que passei. A minha mãe, por tudo que fez por mim durante toda a minha vida e ao meu pai, que não mediu esforços para a concretização desse sonho.

A minha irmã, pelo carinho e dedicação.

A minha namorada Gabriela, que presenciou as batalhas travadas durante esse período me apoiando e me incentivando a nunca desistir e sempre fazer o melhor.

Ao meu orientador o professor *MSc*. Michael Schuenck, que colaborou de forma ímpar para a conclusão desse trabalho.

A todos os professores, que de alguma forma contribuíram para o meu aprendizado.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ACID	*Atomic, Consistent, Isolated and Durable*
API	*Application Programming Interface*
CVS	*Concurrent Version System*
DBCS	Desenvolvimento Baseado em Componentes de Software
GCS	Gerenciamento de Configuração de Software
HD	*Hard Disk (Disco Rígido)*
HTML	*HyperText Markup Language*
ICS	Itens de Configuração de Software
ISO	*International Organization for Standardization*
JMS	*Java Message Service*
JSF	*Java Server Faces*
JVM	*Java Virtual Machine*
MOM	*Middleware Orientando a Mensagem*
MOO	*Middlewares* orientados a objetos
OMG	*Object Management Group*
ORB	*Object Request Brokers*
OSI	*Open System Interconnection*
PHP	*Hypertext Pre-processor*
REST	*Representational State Transfer*
RPC	*Remote Procedure Call*
SCCS	*Source Code Control System*
SCM	*Software Configuration Management*
SCV	Sistema de Controle de Versão
SVN	*Subversion*
URL	*Uniform Resource Locator*
XML	*Extensible Markup Language*

RESUMO

O desenvolvimento de software baseado em componentes tem sido uma solução para a minimização de custos e tempo de um projeto. Para gerenciar esses componentes e assim estimular seu reuso, utilizam-se repositórios locais apropriados para o armazenamento e busca de componentes. Com a utilização de repositórios surgirão novas necessidades, dentre elas a de integrar um Sistema de Controle de Versão (SCV) e uma área do Gerenciamento de Configuração de Software (GCS) com o repositório. O SCV é responsável por controlar todas as modificações ocorridas em um artefato bem como disponibilizar versões desse. Assim, este trabalho propõe um *middleware* que integre um repositório a diferentes sistemas de controle de versão, minimizando assim, a tarefa de enviar uma versão a cada nova correção ou evolução de um artefato.

Palavras-chave: repositório, *middleware*, componentes, versionamento.

1. INTRODUÇÃO

A demanda por sistemas cada vez mais complexos e confiáveis tem feito com que o desenvolvimento de software consuma cada vez mais tempo e dinheiro. O Desenvolvimento Baseado em Componentes de Software (DBCS) é uma técnica que otimiza o processo de desenvolvimento de software porque pode reutilizar componentes já utilizados em outros projetos, além de vários grupos poderem trabalhar em partes diferentes de uma aplicação simultaneamente devido à separação entre implementação do componente e a sua especificação [BRECHÓ, 2009, online]. Os desenvolvedores de aplicações podem ainda, se abster de conhecer detalhes de implementação dos componentes e se concentrarem nos requisitos fundamentais do sistema. No caso da manutenção, é mais fácil e rápido trocar um componente, que fazer o mesmo procedimento em sistemas monolíticos.

Neste contexto, o desenvolvimento de sistemas baseados em componentes de software tem se destacado como uma das principais abordagens de reaproveitamento. Nesta abordagem, há a necessidade de existir o compartilhamento de componentes, fato motivador da existência de diversos repositórios, que são o elo entre desenvolvedor e utilizador de componentes de software [TIZZEI, 2007, p. 4]. Tais ferramentas permitem o armazenamento e busca com o propósito de auxiliar no reuso de componentes e, assim, melhorar as metas de qualidade, custo e produtividade [OLIVEIRA, 2008, p.7]. Uma vez que esses componentes armazenados são bem organizados, divulgados e gerenciados, essa ferramenta torna-se um poderoso instrumento na política do reuso.

Dentro do contexto de DBCS, podem existir dificuldades na gerência de novas versões de componentes, uma vez que o desenvolvedor pode corrigir ou aperfeiçoar o componente várias vezes, tendo que posteriormente acessar o repositório e cadastrar a

nova versão, tornando essa tarefa onerosa e podendo até atrasar as demais tarefas. Além disso, o desenvolvedor pode não se lembrar de realizar essa tarefa, principalmente se não houver uma política bem definida na empresa, já que ele, muitas vezes, está preocupado com o produto final funcionando.

Por outro lado, a marcação de versões como releases em um Sistema de Controle de Versão (SCV) como o SVN (*Subversion*) [SVN, 2009, online], é uma tarefa mais corriqueira e amplamente difundida.

Sendo assim, é desejável que essa tarefa seja automatizada o máximo possível, dado que ela será repetida cada vez que uma nova distribuição de um componente estável estiver disponível, ou seja, pronta para ser utilizada em outro contexto.

Existem diversas ferramentas de controle de versão, algumas até com uso gratuito como é o caso do CVS (*Concurrent Version System*) [CVS, 2009, online], SVN (Subversion) [SVN, 2009, online], Git [GIT, 2009, online], Mercurial [MERCURIAL, 2009, online] entre outras, que são populares e úteis entre os desenvolvedores.

O uso de SCV torna-se imprescindível no controle de artefatos produzidos em um projeto e principalmente no desenvolvimento de software em equipe. Isso porque sua principal função é manter relatórios sobre como, quando e por quem estes artefatos foram alterados além de possuir um histórico de todas as versões geradas, permitindo a recuperação de uma versão antiga a qualquer momento durante o ciclo de vida do projeto. O SCV ainda permite o acesso simultâneo a um artefato por mais de um membro da equipe, agilizando o processo de desenvolvimento.

Por outro lado, o SCV não possui características como opções diferentes de busca de artefatos, mais de uma possibilidade de acesso, como por exemplo, uma interface HTML e outra Web Service, e um local centralizado com maiores informações como o número de downloads de um artefato ou versão de um artefato, características essas que são encontradas em alguns repositórios de componentes.

Diante desses fatos, procurou-se unir as funcionalidades identificadas nos SCV, cruciais para um projeto, com as características de um repositório, as quais são de grande importância principalmente no DBCS.

Essa integração dar-se-á através de um *middleware* que, de forma transparente, ou seja, sem que se perceba que está sendo usado um SCV, o desenvolvedor possa guardar seus artefatos no repositório e posteriormente, quando novas versões forem criadas, ele possa enviá-las do seu próprio ambiente de desenvolvimento ou ainda

utilizando um cliente de SCV. Essas versões poderão ser recuperadas por utilizadores de componentes através do repositório.

É importante ressaltar que, com a utilização da solução proposta alguns outros problemas surgiram como a necessidade do SCV ter um meio de acesso externo. Isso pode ser resolvido atribuindo um endereço na internet para o SCV que possa ser acessado por outros utilizadores.

Assim, a solução proposta é recomendada para ambientes corporativos, onde o CompWeb, o *middleware* e os SCV's estejam hospedados em servidores da empresa e disponíveis somente para os funcionários dessa empresa. Essa solução ajudaria no amadurecimento da empresa em relação a reutilização dos seus artefatos armazenados no CompWeb.

Esse trabalho se subdivide da seguinte forma: a Seção 2 traz a revisão de literatura com os conceitos e as definições das tecnologias utilizadas nesse trabalho como por exemplo: *Middleware* e SCV. A Seção 3 traz os materiais e métodos utilizados no desenvolvimento desse trabalho. Na Seção 4 aborda os resultados e discussões. Por fim, tem-se a conclusão e as referências bibliográficas nas Seções 5 e 6 respectivamente.

2. REVISÃO DE LITERATURA

Esse capítulo apresentará os principais conceitos e tecnologias utilizados para o desenvolvimento do *middleware* proposto por esse projeto. A Seção 2.1 abordará os conceitos de componentes e Desenvolvimento Baseado em Componentes de Software (DBCS). A Seção 2.2 explanará sobre repositórios de componentes. A Seção 2.3 trará informação sobre a área de Gerência de Configuração de Software (GCS) de forma geral, abordando também sobre Sistemas de Controle de Versão (SCV). Na Seção 2.4 será abordado sobre *Middleware*s.

2.1. Componentes e Desenvolvimento Baseado em Componentes

O modelo de DBCS é uma técnica que otimiza o processo de desenvolvimento de software porque permite reutilizar componentes já criados e testados em outros projetos evitando assim o retrabalho, além de possibilitar que vários grupos possam trabalhar na construção de partes diferentes de uma aplicação simultaneamente.

É possível notar que o DBCS tem se tornado uma prática constante nas equipes de desenvolvimento devido à utilização de novas bibliotecas e frameworks baseados em componentes. Podem-se destacar alguns frameworks como Joomla (Gerenciador de Conteúdo Web de código aberto desenvolvido utilizando a linguagem PHP) [JOOMLA, 2009, online] e JSF (framework Java para desenvolvimento de aplicações web) [JSF, 2009, online]. Esses têm como base o DBCS, fomentando o reuso de software e o desenvolvimento colaborativo.

Boa parte dos softwares disponíveis no mercado, principalmente os mais antigos, foram desenvolvidos em blocos monolíticos, formados por partes interrelacionadas, sendo que esses relacionamentos não têm uma interface bem definida, dificultando a reutilização desses blocos [BRAGA, 2000, p.3].

O DBCS propicia o reuso de componentes. Componentes são definidos, no contexto da Engenharia de Software Baseada em Componentes, como unidades de software desenvolvidas e testadas separadamente e que podem ser integradas com outros componentes para compor aplicações finais [SZYP, 1998, p. 27]. Componentes implementam uma ou mais interfaces que identificam os serviços oferecidos a outros componentes, além de declarar os serviços necessários para o seu funcionamento. Um aspecto muito importante é sempre ressaltar que um componente deve encapsular dentro de si seu projeto e implementação, além de oferecer interfaces bem definidas para o meio externo.

O reuso de componentes reduz a complexidade no desenvolvimento, assim como os custos, através da reutilização de componentes exaustivamente testados em outros contextos [BRECHÓ, 2009, online]. É necessária a gerência desses componentes, uma vez que esses podem ser modificados e/ou corrigidos se houver a necessidade. É interessante que haja esse histórico de modificações além das novas versões do componente disponível para que o desenvolvedor possa, em uma próxima reutilização, analisar quais das versões atende melhor o contexto do seu projeto atual.

2.2. Repositórios de Componentes

Um mecanismo de armazenamento e busca de componentes, ou de artefatos de software reusáveis, é bem usual para os desenvolvedores no processo, muitas vezes exaustivo, de localização de componentes adequados para um software em desenvolvimento [SCHUENCK, 2005, p.4]. Um repositório de componentes pode facilitar essa busca tornando essa tarefa menos onerosa além de incentivar a reutilização de componentes.

Dentro dessa perspectiva, um repositório de componentes surge com o intuito de organizar e facilitar a busca de componentes, que nem sempre estão centralizados e organizados. Um repositório será considerado, para o propósito desse trabalho, uma ferramenta onde componentes de software e demais artefatos reusáveis serão armazenados e classificados de modo a facilitar a busca e recuperação dos mesmos. No entanto, algumas outras funcionalidades podem incentivar o seu uso.

De acordo com Ezran [1999, p.185] existem algumas funcionalidades que um repositório deve prover:

- **Identificação e descrição**: para identificar e descrever um componente de software são necessárias algumas características tais como nome, domínio, palavras-

chave, dentre outras que os identificam e os diferenciam dos demais componentes de software que compõem esse mesmo repositório.

- **Inserção:** um repositório deve permitir que usuários autorizados insiram novos componentes de software ou ainda, novas versões de componentes já cadastrados.
- **Exploração do catálogo:** aos usuários de um repositório deve ser permitido que explorem a lista de componentes para que possam conhecer e analisar as características dos componentes disponíveis.
- **Pesquisa textual:** um repositório deve permitir que seus usuários façam pesquisas mais específicas na descrição dos componentes. Como resultados da pesquisa, serão obtidos um ou mais componentes que satisfaçam as condições desejadas.
- **Recuperação:** após a identificação do componente desejado, um repositório deve permitir que seus usuários recuperem esse componente para que possam posteriormente utilizá-lo num processo de reuso.
- **Histórico:** é importante para o gerenciamento de um repositório que ele possa armazenar informações de uso, modificações, criação e exclusão de cada um dos ativos disponíveis. Essas informações devem montar uma base histórica que facilitará a análise e a reutilização dos mesmos.
- **Controle de acesso:** um repositório pode adotar uma política de segurança para que determinadas funcionalidades só estejam acessíveis a pessoas autorizadas. Por exemplo, pode-se definir uma política de segurança onde a pesquisa e utilização de componentes sejam públicas, já o envio de componentes e de versões de componentes seja restrito aos usuários cadastrados no sistema.
- **Gerenciamento de versões:** um repositório pode conter várias versões de um mesmo componente e, sendo assim, é recomendável que haja algum mecanismo para controlar essas versões e estabelecer o relacionamento entre elas.

Com a crescente motivação para o reuso de componentes de software no processo de desenvolvimento, diversas empresas estão investindo na construção de repositórios com suporte ao reuso. Alguns para uso interno, outros com propósitos comerciais [HOLANDA & SOUZA, 2006, p. 5]. Dentre esses repositórios podemos destacar o Sourceforge e Sourceforge Enterprise Edition [SOURGEFORGE, 2009, online], o Maven (um gerente de repositórios para projetos Java) [MAVEN, 2009, online], Koders [KODERS, 2009, online] e Google Code [GOOGLECODE, 2009 online].

A próxima Seção abordará sobre o CompWeb, um repositório web de componentes, sobre o qual o *middleware* proposto trabalhará com o objetivo de interligá-lo a um SCV.

2.2.1. O Repositório CompWeb

O CompWeb é um repositório web de componentes de software desenvolvido utilizando a tecnologia Ruby on Rails que disponibiliza uma interface HTML e outras WebService REST, possibilitando o armazenamento e busca de componentes ou artefato de software [FERREIRA, 2009, p. 6] .

O CompWeb disponibiliza um sistema de busca por nome do componente, *tags* relacionadas, linguagem e plataforma. É importante destacar que sistemas de busca com *tags* têm se popularizado e sido adotado por vários sites como o registrador de favoritos Delicious.com [DELICIOUS, 2009, online], o site de vendas Submarino [SUBMARINO, 2009, online], entre outros, como forma de classificação de conteúdo, facilitando as buscas posteriores. A Figura 1 mostra a busca de componentes no CompWeb através da interface HTML.

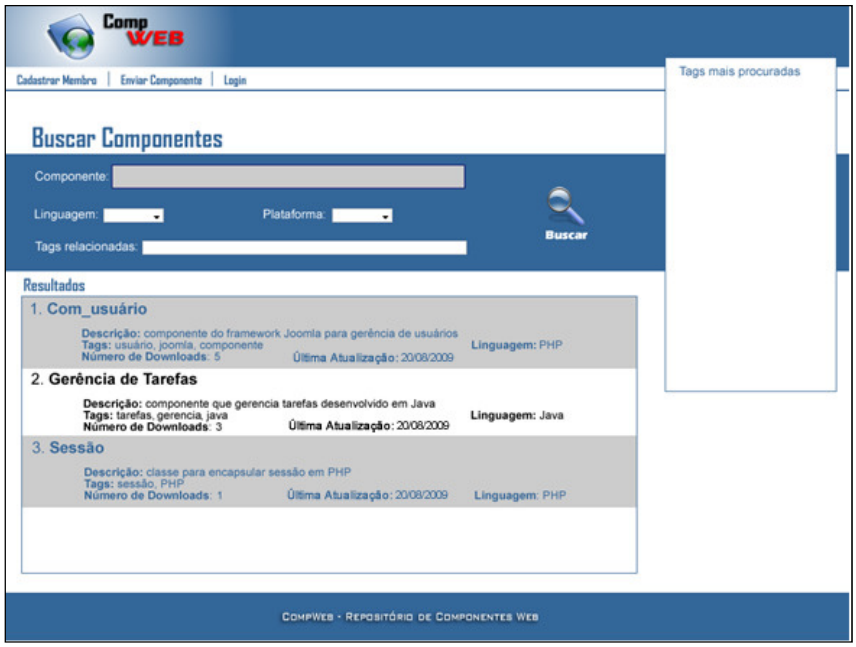

Figura 1 – Sistema de Busca do CompWeb

O utilizador de componentes pode buscar componentes de diversas formas como, por exemplo, especificando a linguagem, plataforma ou *tags* (separadas por

espaço em branco) relacionadas com o componente que ele deseja. O utilizador de componentes não necessariamente precisa ser membro ou ter sido autenticado no sistema para poder buscar algum componente, facilitando assim o acesso.

Outro interessante recurso que o CompWeb disponibiliza é a comunicação entre utilizadores e desenvolvedores de componentes. Através desse contato é possível *feedbacks* sobre erros bem como sugestões de melhorias para os componentes compartilhados.

O resultado de uma busca no CompWeb disponibiliza também informações como o número de downloads e data de atualização do componente. A data de atualização se refere à data em que a versão mais recente do componente foi submetida.

É possível o acesso ao CompWeb através de interfaces Web Services REST para busca e download de arquivos como demonstra a Figura 2.

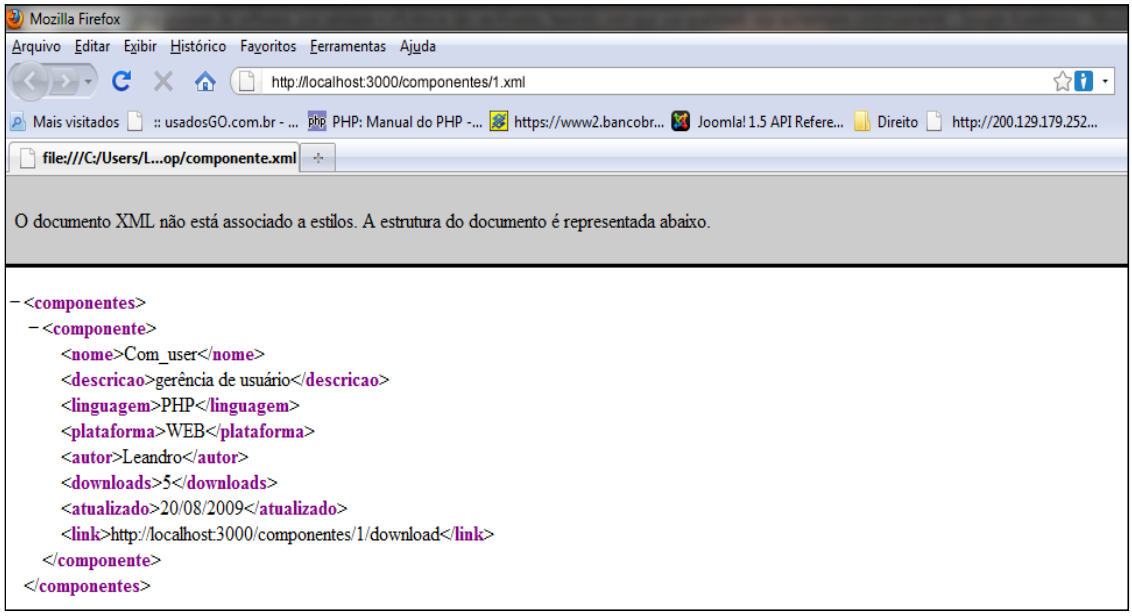

Figura 2 – Resultado de uma busca utilizando a interface Web Service REST

A Figura 2 demonstra o resultado de um consulta por meio da interface Web Service REST. Pode-se observar que a URL (*Uniform Resource Locator*) informada termina com “xml”, o que indica que o servidor retornará um recurso do tipo XML para essa solicitação.

2.3. Gerenciamento de Configuração de Software

O Gerenciamento de Configuração de Software (*Software Configuration Management* – SCM) ou GCS é uma atividade que deve ser aplicada em todo o processo de engenharia de software. Segundo Babich [1986, p. vi], o GCS é a arte de identificar, organizar e controlar modificações no software que está sendo construído por uma equipe de programação. A meta é maximizar a produtividade minimizando os erros decorrentes da integração das várias partes desenvolvidas em paralelo por diferentes membros da equipe.

Uma vez que mudanças podem ocorrer a qualquer tempo dentro do ciclo de vida do projeto, o GCS procura identificar, controlar e garantir que as mudanças estejam sendo adequadamente implementadas e relatadas as pessoas que possam ter interesse nelas [CHRISTENSEN & THAYER, 2002p.432-435]. Esse processo se faz necessário para evitar inconsistências e perda de informação. Somente a organização e descrição disciplinada dos artefatos de um projeto garantem que estes manterão a qualidade, a consistência e a atualidade ao longo das mudanças que acontecem durante a vida dos projetos e a vida dos produtos [FILHO, 2001, p.452-453].

Os artefatos que compreendem todas as informações produzidas no processo da engenharia de software como documentação de análise e especificação, modelos e o próprio código (fontes e executáveis) são chamados de itens de configuração de software (ICS) [PRESSMAN, 1995, p. 916-940]. O software produzido durante todo o projeto é visto como um conjunto de ICS que foram sendo modelados para atender os requisitos solicitados. Alguns exemplos de itens de configuração gerenciados em um processo de GCS são resumidos na Tabela 1.

Assim que um ou mais ICS são entregues, revisados, corrigidos e depois aprovados tornam-se uma linha de base. O ponto central da GCS é o conceito de **linha de base** (*baselines*) definido por Peters e Pedrycz [2001, 67-81] como sendo um artefato que assinala o término de uma atividade e o início de outra. Um documento *baseline* é composto pelos conjuntos de configurações de um ICS após terem sido devidamente revisados e aprovados.

Tabela 1 – Exemplos de Itens de Configuração [Peters & Pedrycz, 2001, p. 69]

Item de configuração	Exemplos
Plano de Gestão	Plano do processo, guia de Engenharia de Software, plano de GCS, plano de testes, plano de manutenção
Especificação	Requisitos, projeto, especificação de testes
Projeto	Código-Fonte
Teste	Projeto de testes, casos, procedimentos, dados e geração de testes
Dicionário de Dados	Especificação dos requisitos de software
Código	Fonte, Executável, requisito
Bibliotecas	Componentes, bibliotecas reutilizáveis
Banco de dados	Banco de dados de auditoria

O GCS é obtido através da identificação de cada *baseline* e do acompanhamento de todas as alterações subseqüentes daquela *baseline* [PETERS & PEDRYCZ, 2001, p.67-81]. As *baselines* são colocadas em um banco de dados de projetos (também chamados de biblioteca de projetos ou repositório de software). Quando um membro da equipe quer fazer uma modificação num ICS já definido como *baseline,* este é copiado do banco de dados de projetos para o ambiente de trabalho do solicitante. É feito um registro dessa atividade e o membro pode trabalhar na cópia até que as mudanças exigidas sejam contempladas. Depois que os procedimentos de controle de mudanças foram contemplados, ou seja, revisados, corrigidos e aprovados, essa copia atualizará o ICS anteriormente copiado. Em alguns casos, ICS copiado poderá ser bloqueado de forma que ninguém mais possa trabalhar nele até que as mudanças tenham sido implementadas, revistas e aprovadas [PRESSMAN, 1995, p. 916-940].

Segundo Peters e Pedrycz [2001, p.67-81] o GCS é composto por quatro elementos básicos: Identificação da Configuração de Software, Controle de Configuração de Software, Auditoria de Configuração de Software e Relatório sobre o Estado de Configuração de Software. A Tabela 2 resume esses elementos básicos.

Tabela 2 – Elementos Básicos do GCS [PETERS & PEDRYCZ, 2001, p.70]

Elementos do GCS	Métodos
Identificação da configuração de software	Definir componentes *baseline.*
Controle de configuração de software	Mecanismo para iniciar, preparar, avaliar, aprovar ou reprovar todas as propostas de alteração.
Auditoria de configuração de software	Mecanismo para a determinação do grau de correspondência entre o estado atual de um sistema de software e seus *baselines* (requisitos e documentos de planejamento).
Relatório sobre o estado de configuração de software	Mecanismo para manter um registro de como um sistema está evoluindo e do estado de um sistema em relação a documentos publicados e acordos redigidos.

O elemento **Identificação da Configuração de Software** tem o objetivo de identificar os ICS, definir o esquema de nomes e números de forma clara e sem que haja equívocos na identificação dos ICS, na descrição dos ICS e na definição dos ICS que formaram um documento *baseline*.

O **Controle de Configuração de Software** é responsável pela gestão das alterações ocorridas nos documentos *baselines*. Essa gestão de alterações é descrita em cinco atividades: 1) Requisições de alteração para um ICS são iniciadas pelos membros da equipe e/ou pelos clientes do projeto; 2) Análise das solicitações em relação à definição de impacto que irá causar em termos de tempo, custo e cronograma; 3) Os resultados das alterações propostas para cada ICS são distribuídos aos membros da equipe; 4) Avaliação das alterações propostas a um ICS e realização das alterações pelos membros da equipe. As alterações rejeitadas são arquivadas para consultas futuras; 5) Feedback para os clientes e desenvolvedores de software dos resultados da avaliação de uma alteração solicitada.

- **Auditoria de Configuração de Software** verifica se aquilo que foi projetado realmente foi implementado e se os testes aplicados ao ICS provam que os requisitos foram atendidos.

- O **Relatório Sobre o Estado da Configuração de Software** registra todas as alterações feitas para um documento *baseline*. Fornece informações sobre o estado do desenvolvimento do software e sua evolução para a equipe do projeto.

Contudo, Murta [MURTA, 2004, p. 6-8] descreve o GCS divido em três sistemas sendo eles: controle de modificações, controle de versões e controle de construções e liberações.

- **O sistema de controle de modificações** controla de forma sistemática os ICS, armazena todas as informações geradas durante o andamento das requisições de modificação de um ICS e relata essas informações aos membros interessados.
- **O sistema de controle de versões** permite que eles evoluam de forma distribuída e concorrente, porém disciplinada. Essa característica é necessária para que diversas requisições de modificação possam ser tratadas em paralelo, sem corromper o sistema de GCS como um todo.
- **O sistema de controle de construções e liberações** automatiza o complexo processo de transformação dos diversos artefatos de software, que compõem um projeto, no sistema executável propriamente dito, de forma aderente aos processos, normas, procedimentos, políticas e padrões definidos para o projeto. Além disso, esse sistema estrutura as configurações de referência selecionadas para liberação, conforme necessário para a execução da função de auditoria da configuração.

Existem diversas técnicas para a integração do ambiente de trabalho com os recursos de GCS, uma vez que as atividades de Engenharia de Software lidam com problemas complexos e que se agravam com preocupações referentes ao controle de modificações, controle de versões e controle de construções e liberações. Essas técnicas visam prover ao desenvolvedor um ambiente onde os recursos de GCS não interfiram profundamente na rotina de trabalho [MURTA, 2004, p. 8].

Desta forma, ambientes de programação, como, por exemplo, o Eclipse [ECLIPSE, 2009, online], o NetBeans [NETBEANS, 2009, online], fornecem recursos de controle de versões (no caso do Eclipse isso é possível através de *plugins* adicionado a ele). Esses recursos permitem que o programador acesse as versões compatíveis de arquivos de código-fonte sem ter que sair do ambiente de programação. Usualmente, o acesso a essas versões compatíveis é feito usando o conceito de projeto, que agrupa esses artefatos, facilitando a manutenção da consistência dos mesmos.

Apesar da existência de processos e ferramentas de GCS adequado para todos os níveis de formalismos desejados, ainda existem projetos de desenvolvimento de software que ignoram completamente a automação do GCS [MURTA, 2004, p. 37].

O GCS torna-se uma atividade indispensável para controlar a complexidade existente quando equipes numerosas manipulam, simultaneamente, um conjunto de artefatos, evitando erros e retrabalho e, conseqüentemente, aumentando a qualidade e a produtividade [MURTA, 2004, p.37].

Atualmente, o GCS é muito utilizado em diversas empresas, principalmente em grupos de desenvolvimento de software de código aberto. A utilização de SCV, ferramentas que auxiliam o GCS, facilita consideravelmente o controle da evolução, a correção de *bugs*, a manutenção de diversas versões simultâneas de um mesmo software e o desenvolvimento distribuído em equipes. Os SCV são fundamentais na gerencia de versões de ICS que são criados durante o ciclo de vida de um projeto e serão abordados na próxima Seção.

2.3.1. Sistemas de Controle de Versão (SCV)

O uso dos Sistemas de Controle de Versão (SCV) tem como principal função manter um relatório sobre como, quando e por quem estes artefatos foram alterados. Além disso, sistemas deste tipo ainda possuem um histórico de todas as revisões geradas, permitindo a recuperação de uma versão antiga sem erros ou sem alguma funcionalidade [WIRES & FEELEY, 2007, p. 3].

O primeiro sistema de controle de versão foi o *Source Code Control System* (SCCS) criado na Bell Labs em 1972 por Mark J. Rochkind para um IBM System/370 e depois portado para sistemas Unix [MARC, 1975, p. 364-369].

2.3.1.1. Funcionamento Básico

Alguns conceitos básicos são compartilhados entre a maioria dos softwares de controle de versões, o que não quer dizer que não existam outros softwares de controle de verões que funcione de maneira totalmente diferente do explicado a seguir.

Um sistema de controle de versão é composto, basicamente, por duas partes: **repositório** (*repository*) e **área de trabalho**.

O repositório é o local são armazenados todos os arquivos do projeto utilizados pela equipe bem como o histórico gerado pelas alterações realizadas em cada arquivo. Geralmente, esse armazenamento é feito em um sistema de arquivos ou em um banco de dados qualquer de forma persistente. Um sistema de controle de verões pode ter vários repositórios possibilitando o versionamento de vários projetos independentes.

A área de trabalho (*working copy*) é uma cópia local da última versão dos arquivos de um projeto e que é monitorada para identificar as alterações realizadas. A cada nova alteração realizada pelo desenvolvedor é necessário atualizar os arquivos do servidor submetendo as alterações, esse processo é denominado *commit*. Após a realização do *commit* as alterações são enviadas para o servidor que as guarda gerando uma nova **revisão** no repositório, contendo além das modificações feita a data e o autor. Uma revisão é uma cópia de todos os arquivos e diretórios em um determinado momento da evolução do projeto. Elas são mantidas e podem ser recuperadas e analisadas sempre que desejado.

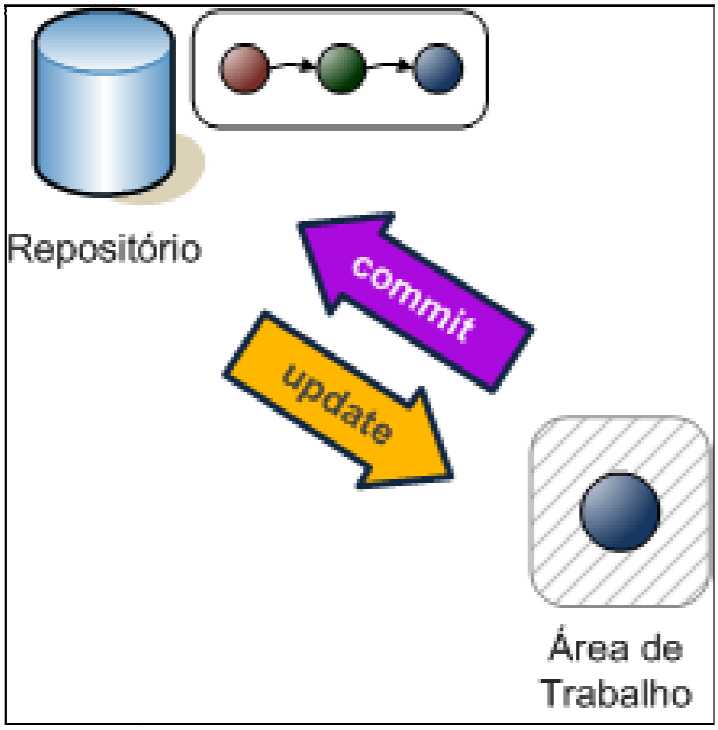

Figura 3 - Sincronização da área de trabalho com o repositório através do *commit* e *update*.

O desenvolvedor pode baixar para sua área de trabalho local os arquivos de um repositório ou mesmo atualizar os arquivos já existentes em sua área de trabalho. Esse processo é denominado *update*. Assim a comunicação entre repositório e área de trabalho é realizada através desses dois processos: *commit* e *update*.

A forma como ele permite alterar esses arquivos pode seguir dois modelos: o distribuído e o cliente-servidor [DONOVAN & GRIMSON, 1990, p. 2-4]. No modelo distribuído, cada desenvolvedor trabalha diretamente em uma cópia local de arquivo. A atualização da cópia no repositório compartilhado só é feita ao final de todas as modificações desejadas. Já no modelo cliente-servidor não existe a necessidade de uma cópia local. Os desenvolvedores acessam remotamente o repositório compartilhado e fazem as modificações diretamente, sem a necessidade de atualização posterior da cópia.

Cada software de controle de versões guarda diferentes informações sobre o usuário responsável pelas modificações. As mais comuns são a versão do arquivo (cada alteração gera uma nova versão), o dia e hora da alteração, o nome do usuário e um comentário a respeito do que foi modificado. A cada nova versão criada é estabelecido um histórico, bem como a possibilidade de resgate de versões anteriores de um documento ou um código. Esse resgate de arquivos passados denominado *roll-back* [LEYTON, 2005, p.6] pode ser realizado a qualquer versão anteriormente armazenada.

Alguns softwares de controle de versões ainda possuem a característica de permitir a criação de ramos (*branches*). Esta funcionalidade é utilizada quando se deseja criar versões diferentes de um software, mas que contenham a mesa base. Por exemplo, uma versão *shareware* (versão de um software com funcionalidades reduzidas) e uma versão comercial do software. Ambas possuem o mesmo núcleo, mas enquanto a versão comercial disponibiliza todas as funcionalidades, a *shareware* inclui apenas algumas [TROMBETTA, 2008, p. 3]. Existem ainda, softwares de controle de versões que dispõem de um sistema de *tags,* como é o caso do *Subversion* (SVN), para identificar versões de um artefato.

2.4. Middleware

A computação distribuída cresceu exponencialmente, principalmente, depois que órgãos governamentais e empresas, geralmente localizados em regiões diferentes, necessitavam trocar informações. Muitas vezes, essas informações estavam armazenadas em aplicações com diferentes linguagens e plataforma. Dessa necessidade surgiu o conceito de *middleware* que em um primeiro instante tinha a função básica de unir os componentes de um programa distribuído, determinando a maneira pela qual estes

componentes se comunicariam. Em seguida percebeu-se que era possível integrar aplicações completas entre e dentro de organizações utilizando *middlewares*. Por outro lado, isso acabou aumentando gradativamente a complexidade dos *middlewares* [CHARLES, 1999, p. 17-19].

Outro importante fator que impulsionou a popularização e utilização de *middlewares* foi à expansão e, conseqüentemente, a evolução da internet que trouxe uma gama de informações e recursos disponíveis, geralmente, de forma distribuída. Essas informações e recursos acabaram gerando um grande número de serviços para os consumidores.

Para que esses serviços sejam mais úteis e atrativos aos consumidores, muitas vezes é necessária a junção de um ou mais sistemas. Esses sistemas precisam ter um alto desempenho para receber um grande volume de requisições, característica essa que deve ser levada em consideração quando um engenheiro de software for escolher o melhor tipo de *middleware* para essa integração. Além do alto desempenho, existem outros atributos que devem ser observados como: flexibilidade, adaptabilidade, disponibilidade entre outros.

O termo *middleware* tem sido usado para descrever diversos tipos de softwares. Segundo Toni Bishop [BISHOP, 2002, p. 1], *middleware* é uma camada de software que conecta duas ou mais aplicações. Essa é uma das diversas definições de *middleware* encontradas na literatura.

Em resumo, um sistema de *middleware* caracteriza-se como uma camada de software localizada entre a aplicação e o sistema operacional que permite comunicação entre aplicações distribuídas, tendo por objetivo diminuir a complexidade e heterogeneidade de diferentes sistemas [BERNSTEIN, 1996, p. 86-98].

A Figura 4 exemplifica a comunicação entre aplicações distribuídas através de *middlewares*.

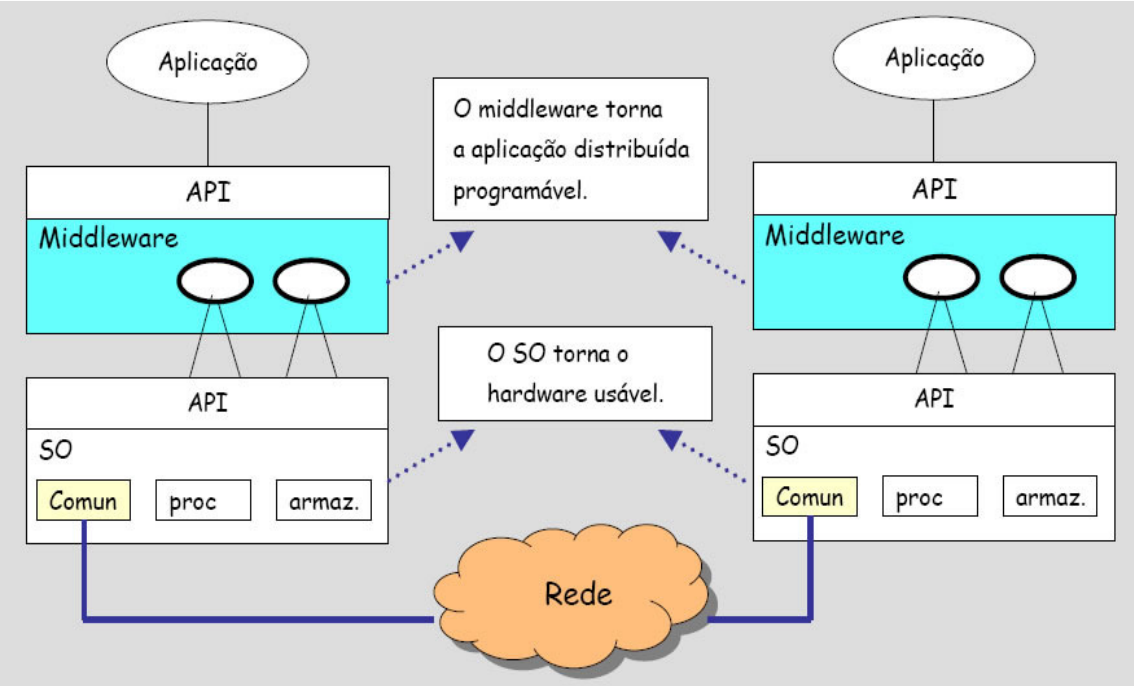

Figura 4 – Comunicação de aplicações distribuídas através de um *middleware* [Bernstein, 1996].

A Figura 4 demonstra, de maneira geral, com é feito a comunicação de aplicações distribuídas utilizando um *middleware*. A aplicação do Host 1 informa os dados para o *middleware* que em seguida passa esses dados para o sistema operacional que, através da rede, os envia para o Host 2. O sistema operacional do Host 2, por sua vez, passa os dados para o *middleware* que enviará esses dados para a aplicação do Host 2.

Middleware tem como objetivo resolver o problema da heterogeneidade e prover transparência e simplicidade na construção de sistemas distribuídos.

Duas características importantes na construção de um *middleware* são sua flexibilidade e performance. Porém, estes dois pontos são mutuamente exclusivos, já que o alto nível de flexibilidade acaba impedindo um alto nível de performance, sendo ideal um balanceamento entre os dois.

2.4.1. História do Middleware

O primeiro *middleware* foi desenvolvido pela *Sun Microsystems* em meados dos anos 80 para comunicação de dois programas em computadores distintos sem a preocupação com detalhes da rede [CHARLES, 1999, 17-19]. Antes, os produtos de *middleware* não

eram conhecidos como *middleware* propriamente dito, e sim como ferramentas de comunicação entre aplicações.

Assim, com o crescimento dos sistemas distribuídos devido ao fato das indústrias estarem migrando seus sistemas baseado em cliente/servidor para modelos baseados em aplicações distribuídas, os protocolos *middleware* começaram a ser utilizado em maior escala. Outro fator que impulsionou essa utilização foram os grandes avanços tecnológicos na área de telecomunicação [GEIHS, 2001, p. 24-31].

A evolução e disseminação de sistemas abertos (*open systems*) trouxeram padronizações que acarretariam na competição e diminuição dos preços desses produtos de *middleware*.

Em dezembro de 1980 o primeiro esboço da especificação de *middleware* foi publicado pelo *Open System Interconnection* (OSI). Esse esboço foi autorizado pela *International Organization for Standardization* (ISO) mas levou alguns anos para que este padrão fosse ratificado. Vale ressaltar que esse processo falhou em conseguir um impacto significativo na evolução dos *middlewares* [BISHOP, 2002, p. 1].

Avanços foram realizados na padronização das plataformas de *middlwares* para aumentar a comunicação assíncrona por volta dos anos 90. O principal aspecto foi à manipulação de eventos – pois eles deveriam prover respostas em um determinado tempo- condições de erro, comunicações de grupos e aplicações multimídia [Bacon, 2002, p. 59-64].

Fundada em 1989 por onze companhias (incluindo Hewlett-Packard, IBM, Sun Microsystems, Apple Computer, American Airlines e Data General) a *Object Management Group* (OMG) criou as primeiras especificações para *middleware*. Hoje a OMG é formada por mais de 800 entidades dentre elas empresas, órgãos governamentais e universidade [OMG, 2009, online]. A partir dessas especificações, o foco principal passou a ser no detalhamento de funções específicas dos produtos de *middleware* para a inclusão de perguntas (*queries*), transações, segurança, etc. Essas especificações continuaram a melhorar até que em 1996, controles de acesso e de autorização foram incluídos e em 1998, especificações para aplicativos em tempo real foram adicionadas [CERUTI, 1999, p. 1-2].

2.4.2. Serviços de Middleware

Um serviço de *middleware* é um serviço de uso geral que fica entre plataformas e aplicações (ver Figura 3) [BERNSTEIN, 1996, p. 86-98]. Entenda como plataforma um conjunto de serviços de baixo nível formados pela arquitetura do processador e a API (*Application Programmer Interface*) do sistema operacional, como por exemplo, Intel x86 e Win-32. Já um serviço de *middleware* é definido pelas APIs e protocolos que suporta.

Como muitos sistemas de alto nível, *middlewares* são difíceis de definir de uma maneira tecnicamente precisa. No entanto, os componentes de *middleware* têm várias propriedades que, juntas, geralmente tornam claro que esses não são um aplicativo ou plataforma de serviços específicos. Algumas dessas propriedades são: genéricos entre aplicações, suportam múltiplas plataformas, são distribuídos, e provê suporte padrão para interfaces e protocolos [BERNSTEIN, 1996, p. 86-98].

Para que um serviço possa ser considerado um *middleware* ele deve atender às necessidades de várias aplicações de um determinado contexto, não fornecendo somente serviços para uma aplicação específica, suporte a pelo menos uma API padrão. Além disso, sua implementação deve ser independente de plataforma.

Os seguintes serviços, normalmente, são fornecidos por *middlewares*:

- **Gestão de Apresentação**: Gerenciamento de formulários, gráficos, impressora e hipermídia.
- **Computação**: Ordenação, dispositivos matemáticos, serviços internacionalizáveis, conversores de dados e serviço de tempo.
- **Gestão de informação**: Servidor de diretórios, gerenciador de log, gerenciador de arquivos.
- **Comunicação**: Mensagens peer-to-peer, chamada remota de procedimento, fila de mensagem, mensagem eletrônica e exportação eletrônica de dados.
- **Controle**: Gerenciamento de transações, gerenciamento de threads.
- **Gestão do sistema**: Serviço de notificação de eventos, serviço de contas, gerenciamento de configuração, detectores de falhas.
- **Comunicação entre processos:** Pode ser considerado o coração do middleware. Podemos citar como exemplo os *Object Request Broker* (ORB) do CORBA, que tem como objetivo a integração entre as aplicações remotas.

- **Interface com o usuário**: HTML e outros formatos multimídia aceitos pela Internet.

O serviços de gerenciamento de qualidade de serviço (*QoS*) e segurança da informação também fazem parte dessa lista. Porém, há uma necessidade de padronização, entre a integração do *QoS* e o *middleware*, para que esse procedimento possa ser realizado [GEIHS, 2001, p. 24-31].

2.4.3. Taxonomia dos Middlewares

Existe uma grande dificuldade para classificar *middlewares*. Dentre essas dificuldades podemos destacar o fato dos *middlewares* não serem somente um serviço, mas uma combinação de vários. Para esse trabalho, foi utilizada a classificação proposta por Tony Bishop [BISHOP, 2002, p. 1], R. Nunn [NUNN, 2009, online] e W. Emmerich [EMMERICH, 2000, p. 117-129] onde os *middlewares* são divididos nas seguintes categorias: monitor de processamento de transação, *middlewares* orientado a mensagem, *middlewares Remote Procedure Call* (RPC), *middlewares* orientado a objetos, os quais serão detalhados abaixo:

2.4.3.1. Monitor de Processamento de Transação

Normalmente, monitores de transação de processo (TP) não são usados para comunicação entre aplicações. Esse tipo de *middleware* é mais indicado para transações de aplicação que acessa bancos de dados relacionais por proverem um ambiente completo para esse tipo de transação. Essas transações podem incluir diversos comandos onde todos estes podem ser executados com sucesso ou todos falharem. Uma transação deve obrigatoriamente suportar propriedades ACID (*Atomic, Consistent, Isolated and Durable*). Atômica diz respeito ao fato de em uma transação ser executada completamente ou nada ser executado, ou seja, não existem resultados parciais. A consistência nos diz que independentemente do estado da transação ela deve sempre estar consistente com o sistema. Deve ser isolada, pois uma transação deve funcionar independentemente das outras quando estiverem sendo executadas na mesma máquina. Deve ser durável para que após a conclusão da transação os resultados desta não sejam perdidos. A Figura 5 demonstra o modelo de interação do Monitor de Processamento de Transação.

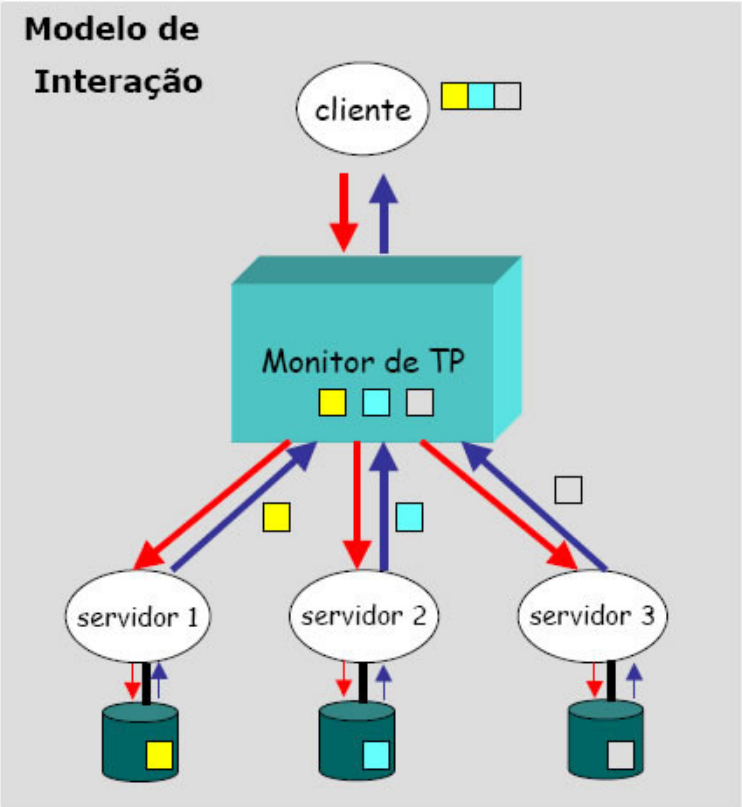

Figura 5 – Modelo de interação do Monitor de Processamento de Transação [Rosa, 2009, online]

As vantagens desse tipo de *middleware* estão no fato dos componentes poderem ser mantidos num estado consistente devido às transações fazerem uso de propriedades *ACID*, que são muito confiáveis por serem tolerantes a falhas. Por outro lado esse tipo de *middleware* tende a ter o tempo de processamento mais longo devido à utilização da propriedade *ACID* e requerer uma grande perícia na implementação [Talarian, 2009, online]. Esse tipo de *middleware* é adequado quando transações precisam ser coordenadas e sincronizadas em múltiplos bancos de dados.

2.4.3.2. *Middleware* Orientando a Mensagem (MOM)

É baseado na troca de mensagens entre programas e pode ser divido em duas categorias: mensagens enfileiradas e mensagens passadas.

Mensagens enfileiradas é um modelo de comunicação indireta, pois a comunicação acontece utilizando-se filas, ou seja, mensagens de um programa são enviadas para uma fila específica até que seja mandada para o seu receptor. Por outro lado, as mensagens passadas têm um modelo de comunicação direto pelo fato da mensagem ser entregue diretamente à parte interessada. A Figura 6 exemplifica o envio de mensagens em um *middleware* orientado a mensagem utilizando fila.

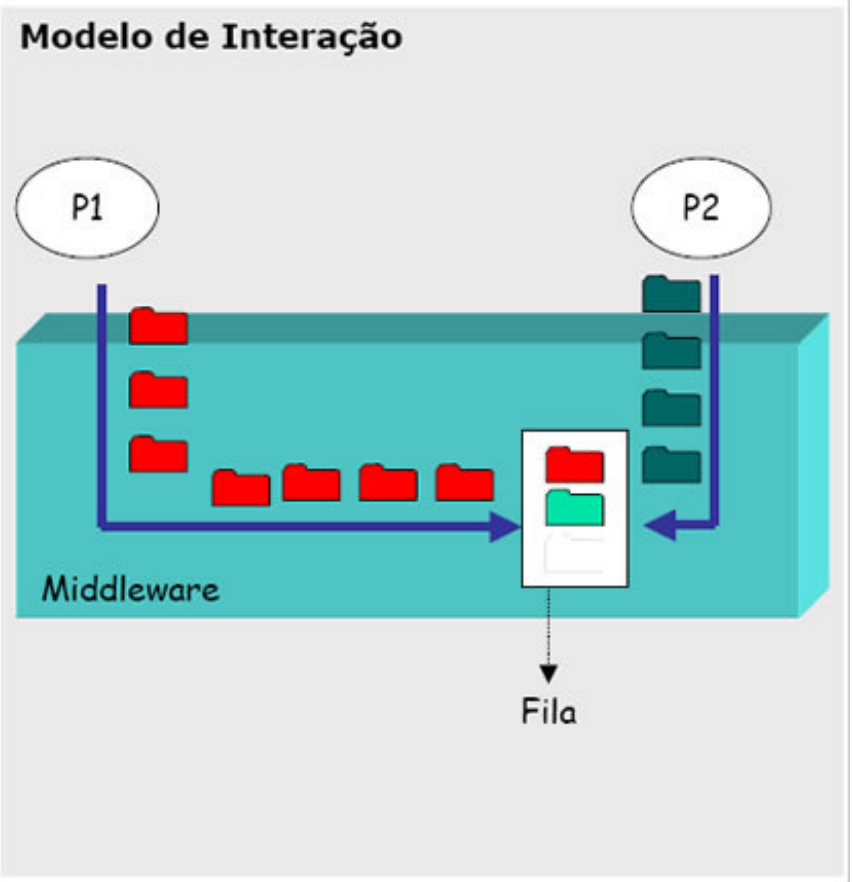

Figura 6 - Modelo de interação do *middleware* orientado a mensagem [Rosa, 2009, online]

Podemos apontar algumas vantagens: primeiro MOM suporta comunicações de grupos, ou seja, é possível o envio de mensagem para varias aplicações, sendo que se alguma aplicação não receber a mensagem, todas as outras não receberam. Segundo, o uso de filas persistentes, ou seja, é enviada uma notificação informando quando há falha no processamento da mensagem, permitindo assim que seja reiniciado o processo, aumenta a confiabilidade. Terceiro, suportam mais protocolos de redes que os *middlewares* RPC [Hartwich, 2003, p. 46-57].

2.4.3.3. Remote Procedure Call (RPC)

Desenvolvidos pela Sun nos anos 80, é geralmente usado para prover chamadas remotas a procedimentos (RPC) sendo uma das principais formas de comunicação entre processos remotos, operando em baixo nível. Esse *middleware* está disponível para vários sistemas operacionais, incluindo distribuições Unix e Microsoft Windows.

A comunicação acontece da seguinte forma, se um cliente que deseja receber um serviço faz uma requisição ao servidor. Essa requisição é codificada em um

determinado formato (*marshaling*) para que possa ser transmitido. No outro lado o servidor recebe a mensagem e executa o processo reverso, ou seja, converte os dados codificados (*unmarshaling*) para o formato específico do serviço. Em seguida ele executa o serviço requisitado e envia o resultado para o cliente. A Figura 7 exemplifica o processo de interação de um *middleware* RPC.

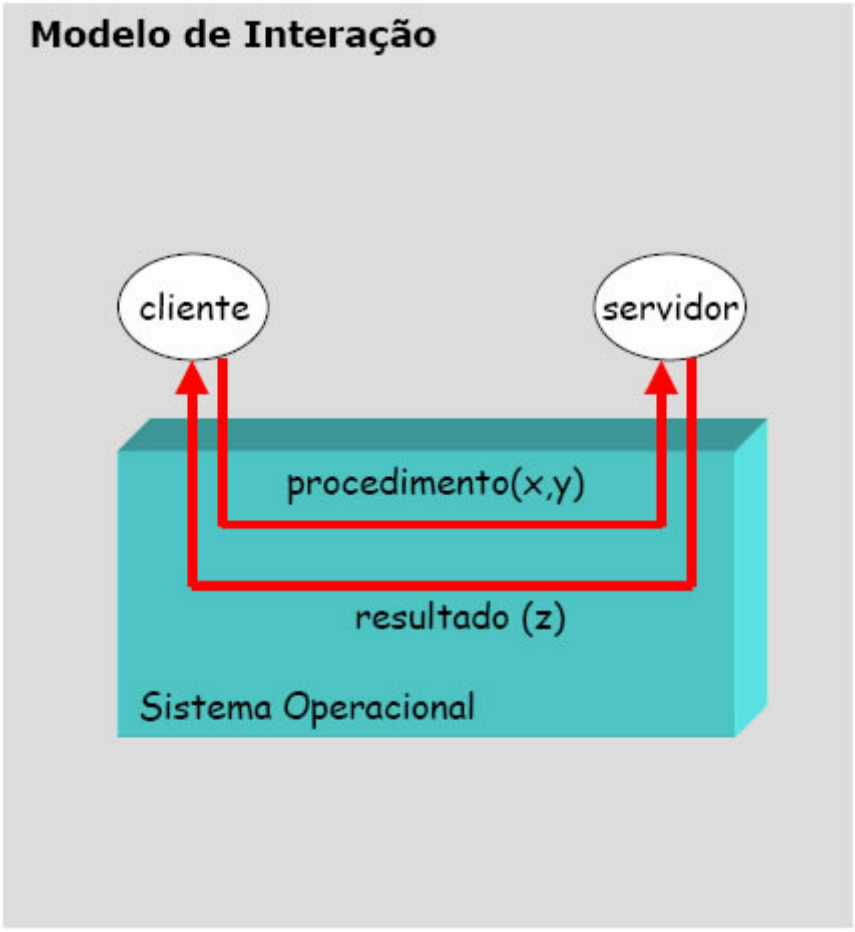

Figura 7 - Modelo de interação do *middleware* RPC [Rosa, 2009]

Marshaling/Unmarshaling de mensagens são implementados por *stubs* que são automaticamente criados por um compilador que permite a criação de RPC´s. Essa abordagem suporta uma comunicação síncrona entre um cliente e um servidor, mas não suporta diretamente comunicação assíncrona e *multicast*.

Podemos dizer que eles são melhores que os *middlewares* monitor de processamento de transação e os orientados a mensagens porque as implementações fazem com que o *marshaling/unmarshaling* seja realizado pelos *stubs*, e não manualmente como nos dois casos anteriores. Outra grande vantagem dessa abordagem é o fato do bom suporte a heterogeneidade, pois como já foi mencionado, vários sistemas operacionais e linguagens

de programação podem ser utilizados [Nunn, 2009, online]. De acordo com [Talarian, 2009, online], RPC´s devem ser utilizados em aplicações simples e pequenas com comunicação primária ponto-a-ponto. Esse tipo de *middleware* não seria uma boa escolha para se construir grandes aplicativos empresariais que necessitem de um alto grau de performance e confiabilidade.

2.4.3.4. Middlewares Orientados a Objetos (*MOO*)

Middlewares orientado a objeto são uma evolução dos *RPC´s*. A idéia desse tipo de *middleware* é fazer com que os princípios de orientação a objetos possam ser utilizados no desenvolvimento de sistemas distribuídos. *MOO's* suportam requisições de objetos distribuídos, o que nos permite dizer que um objeto cliente requisita a execução de uma operação em um objeto servidor que pode residir em um outro local. Assim como nos RPC's, nesse tipo de abordagem são os *stubs* que realizam o *marshaling/unmarshaling* das mensagens. Quanto à comunicação, as mensagens são enviadas de forma transparente aos componentes, podendo suportar comunicação síncrona e assíncrona. A Figura 8 demonstra esse tipo de comunicação.

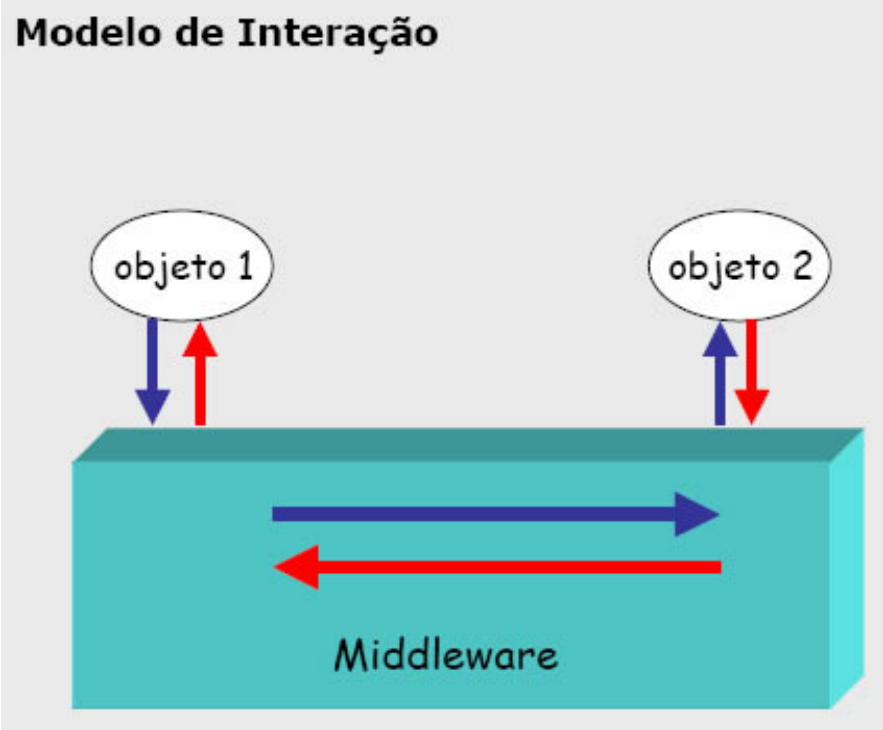

Figura 8 - Modelo de interação de *middleware* orientado a objeto [Rosa, 2009]

MOO apresenta um conjunto de vantagens: primeiro, em termos de confiabilidade, MOO suporta tratamento de exceções. O cliente captura essas exceções para verificar se houveram falhas durante a execução da requisição. Segundo, MOO

suporta vários tipos de heterogeneidade, pois eles podem ser codificados em vários tipos de linguagem, e servidores podem estar em diferentes plataformas de hardware e sistema operacional. Terceiro, a maioria das implementações dão suporte a mensagens e a transações. Por conseguir agrupar todas essas características, MOO tem a capacidade de substituir os outros três tipos de *middleware* em diferentes aspectos, isto faz de MOO um poderoso e flexível tipo de *middleware* [Hartwich, 2003, p. 46-57] [Nunn, 2009, online].

2.4.4. Tipos de Middleware

Embora a tecnologia de *middleware* convencionais favoreça o desenvolvimento de aplicações distribuídas, ele não oferece um apoio adequado para lidar com os aspectos dinâmicos da nova infra-estrutura computacional. A próxima geração de aplicações requer um *middleware* que pode ser adaptado às mudanças no ambiente e personalizado para caber em dispositivos variando de PDA's e sensores para poderosos desktops e supercomputadores. Essa nova necessidade fez com que surgissem novos tipos de *middleware*, dentre eles o reflexivo e o adaptativo, que serão abordados a seguir [Kon, 2002, p. 33-38].

2.4.4.1. *Midleware* reflexivo

É um tipo de *middleware* implementado como uma biblioteca de componentes que podem ser configurados e reconfigurados em tempo de execução pela aplicação [Kon, 2002, p. 33-38]. A interface do *middleware* é adaptável e pode ser usado por aplicativos desenvolvidos para *middlewares* tradicionais. Além disso, o aplicativo pode inspecionar as configurações internas do *middleware* e, se necessário, configurá-lo para se adaptar as mudanças do ambiente. Nesse modo, é possível selecionar os protocolos de rede, políticas de segurança, algoritmos de codificação e vários outros mecanismos para otimizar o desempenho do sistema para diferentes contextos e situações.

A nova geração de aplicativos requer *middlewares* que possam se adaptar a mudanças no ambiente e nos dispositivos em que ele funciona contrapondo-se as tecnologias de *middlewares* tradicionais.

Além das características mencionadas acima, uma arquitetura reflexiva deve também fornecem suporte para personalizar o comportamento do middleware de forma dinâmica bem como a gestão de recursos disponíveis.

Ao contrário do *middleware* tradicionai, que é construído como uma caixa preta monolítica, *middleware* reflexivo é organizado como um grupo de componentes que colaboram entre si. Essa organização permite que os motores do *middleware,* geralmente muito pequenos, sejam configurados permitindo a interação com os *middlewares* tradicionais [Kon, 2002, p. 33-38]. *Middlewares* convencionais incluem todas as funcionalidades que qualquer aplicativo pode precisar, no entanto, na maioria das vezes, as requisições solicitadas usam apenas um pequeno subconjunto destas funcionalidades.

Enquanto *middlewares* convencionais como Java Virtual Machine (JVM) requerem vários megabytes de memória, *middlewares* reflexivos baseado em componentes podem consumir poucos kilobytes de memória.

As técnicas de reflexão tornam possível que a implementação em um certa linguagem seja aberta sem que sejam revelados os detalhes de implementação ou de portabilidade; as técnicas de orientação a objetos permitem como modelo resultante que o comportamento e a linguagem de programação sejam ajustadas localmente e incrementalmente.

2.4.4.2. *Middleware* adaptativo

Surge como uma nova concepção de *middleware* que tem como chaves três tecnologias: separação de interesses, reflexão computacional e projetos baseados em componentes.

Esse tipo de *middleware* tem como foco dissolver limites tradicionais em como: quando e onde os seres humanos e os computadores estão interagindo. Seu objetivo geral é ser dinamicamente adaptável.

As seguintes características devem ser encontradas nesse tipo de *middleware*:

- **Suporte a aplicações sensíveis ao contexto e adaptativas**: Facilita na construção de aplicações adaptativas e sensíveis ao contexto por um *middleware*.
- **Suporte em tempo de execução de comunicação *ad hoc* entre aplicações**: Considerando que o custo da comunicação em ambientes sem fio é maior que o custo da computação neste mesmo ambiente, cabe ao *middleware* utilizar de maneira inteligente o recurso da comunicação baseado no contexto e nas necessidades reais das aplicações.

- **Serviços adaptativos específicos da aplicação:** Além dos serviços disponibilizados pelo *middleware*, uma aplicação pode ter serviços específicos, que envolvem segurança e gerenciamento de grupo.
- **Interoperabilidade com *middleware* de outros domínios**: Com o objetivo de fornecer a interoperabilidade necessária com *middlewares* de outros domínios, o *middleware* adaptativo deve ter suporte à interoperabilidade, o que permite que aplicações móveis se comuniquem com as aplicações já existentes sem muito esforço.

Middleware adaptativo tem sido utilizado em vários sistemas incluindo controle de mísseis, computação de aviões não tripulados e sistemas de radar [SCHMIDT, 2001, p. 2-8].

3. MATERIAIS E MÉTODOS

Nesse capítulo serão apresentados os materiais e métodos utilizados no desenvolvimento desse trabalho.

3.1. Local e período

Os locais de trabalho utilizados nesse projeto foram: os laboratórios do CEULP/ULBRA e Diretoria de Tecnologia da Informação da Universidade Federal do Tocantins. Cada local contribuiu de forma ímpar para o bom andamento do projeto. O período de desenvolvimento de todo o projeto teve início em julho de 2009 e foi concluído em junho de 2010 na cidade de Palmas-TO.

3.2. Materiais

Durante o desenvolvimento desse trabalho foram utilizados os seguintes materiais que foram divididos em três categorias: *hardware*, *software* e fontes bibliográficas.

3.2.1. Hardware

Foi utilizado o notebook Semp Toshiba equipado com um processador Intel(R) Core Duo 2.0 GHz, 2 GB de memória *RAM* e um HD (*Hard Disk*) com capacidade de 100 GB de armazenamento. Um computador desktop equipado com um processador Intel Core 2 Quad Q8200 2,33 GHz, 8 GB de memória RAM e dois HDs SATA com capacidade de 350 GB de armazenamento.

3.2.2. Software

Os softwares utilizados no desenvolvimento desse projeto foram:

- Microsoft Word 2007 para edição;
- Microsoft Windows Seven como sistema operacional;
- Adobe Acrobat Pro para leitura de material de apoio encontrados na internet;
- Microsoft Power Point para desenvolvimento da apresentação do trabalho proposto;
- Mozila FireFox utilizado como navegador para as pesquisas na internet;
- Como IDE para desenvolvimento do *middleware* foi utilizado o NetBeans 6.8;
- Para a implementação do *middleware* proposto foi utilizado a plataforma Java versão 6.

3.2.2.1. **SVNKit** [SVNKIT, 2009]

A SVNKit é uma API desenvolvida utilizando a plataforma Java e traz diversas funcionalidades para acessar e manipular repositórios SVN. Essa API não necessita de bibliotecas Java adicionais ou outros aplicativos para ser utilizada nem a necessidade de um sistema operacional ou código específico. Ela é portátil, *Open Source* e compatível com a última versão do SVN. O SVNKit pode ser encontrada para download no endereço http://svnkit.com/index.html.

Através da SVNKit se tem acesso a uma série de interfaces que permitem realizar praticamente tudo o que um usuário pode precisar em um repositório SVN. Isto inclui fazer check-out, updating, committing, busca de históricos e versões.

A API SVNKit será utilizada nessa trabalho na implementação do middleware para conexão com repositórios SVN's.

3.2.3. Fontes Bibliográficas

Os materiais coletados e estudados foram: páginas Web, artigos, dissertações de mestrados, livros e trabalhos de conclusão de curso e especializações. Foram utilizados também como fontes de dados alguns livros encontrados na biblioteca do Centro Universitário Luterano de Palmas - CEULP/ULBRA.

3.2.4. Softwares de Controle de Versões

Existem hoje no mercado diversas ferramentas que implementam o controle de versão, automatizando o processo de forma a agilizar e facilitar o trabalho do desenvolvedor. Essas ferramentas são responsáveis por gerenciar os artefatos em um repositório mantendo informações sobre como, quando e por quem estes artefatos foram alterados. Por exemplo, quando dois arquivos são alterados por usuários diferentes simultaneamente, a ferramenta de controle de versão utilizada, de modo geral, faz uma comparação entre eles e procura por eventuais conflitos. Se encontrado algum, a ferramenta solicita a intervenção do usuário para indicar que modificações devem ser armazenadas. Caso contrário, ele aceita apenas alterações feitas na versão mais atualizada do arquivo. A cada operação de atualização da cópia do repositório, a ferramenta incrementa a versão do arquivo. A ferramenta também adiciona informações a um histórico que contém o nome do autor, data e hora da modificação, bem como uma breve descrição do que foi alterado. Algumas ferramentas permitem ainda o desenvolvimento de ramos (*branches*). Desta forma, é possível elaborar, em paralelo, duas ou mais versões diferentes de um mesmo software.

Esta seção tem como objetivo descrever algumas destas ferramentas como o CVS, o Subversion, o GIT e o Mercurial.

3.2.4.1. *Concurrent Version System* – CVS [CVS, 2009, online]

O *Concurrent Version System*, mais conhecido como CVS, é um sistema de código aberto para controle de versões que foi desenvolvido no final da década de 80. Entretanto, devido a algumas limitações, tais como problemas para renomear ou mover arquivos, o CVS perdeu espaço. Porém, ainda existem diversas distribuições de clientes para esse sistema de controle de versão, tais como, TortoiseCVS e Eclipse (*plugin* do CVS para o eclipse). Diversas outras ferramentas foram implementadas com o propósito de substituí-lo, como o sistema *Subversion*, que será apresentado na seção a seguir.

3.2.4.2. *Subversion* – SVN [SVN, 2009, online]

Iniciado em 2000 com a CollabNet [COLLABNET, 2009, online] que ainda patrocina uma grande parte dos trabalhos, o SVN acabou substituindo o uso do CVS para gerenciar o desenvolvimento de projetos de código aberto, tais como o Apache, KDE e Samba.

O *Subverion* (SVN) é um sistema de controle de versões gratuito e com código aberto (*open-source*) que resolveu problemas como, por exemplo, a impossibilidade de renomear e mover arquivos mantendo um histórico das ações realizadas, comuns em outros sistemas como CVS [COLLINS-SUSSMAN, 2003, cap. 1, p.1-3]. Além disso, ele possui mecanismos que evitam que os arquivos ou diretórios sejam corrompidos durante uma operação de *commit* (atualização no repositório) e suporta o armazenamento de arquivos binários. Por último, como era um dos objetivos da equipe que o desenvolveu, o SVN inclui todas as funcionalidades providas pelo CVS (para que qualquer usuário do CVS pudesse migrar de sistema com pouco esforço) através de uma interface mais simples, facilitando a criação, edição e remoção de arquivos e diretórios.

Outra importante funcionalidade que o SVN disponibiliza é o sistema de *tags* para rotular versões e dizer se aquela versão é um *release*, ou seja, está pronta para uso, ou se ele ainda está em fase de desenvolvimento.

3.2.4.3. Git [GIT, 2009, online]

Git é um sistema de controle de versão distribuído, gratuito e de código aberto projetado para lidar com tudo, desde pequenos projetos a projetos de grande dimensão, com rapidez e eficiência.

Desenvolvido por Linus Torvalds inicialmente para dar suporte ao desenvolvimento do Kernel do Linux. O primeiro protótipo foi divulgado por Torvalds em 2005, como uma alternativa aos sistemas de controle de versão existentes, pois nenhum deles atendia ao requisito de eficiência exigido por ele [TORVALDS, 2009, online].

O Git foi projetado para ter o seu repositório distribuído. Ou seja, cada desenvolvedor possui uma cópia completa do repositório, com cada versão dos arquivos do sistema gerenciado e seu histórico. Assim, o Git contempla duas características muito importantes: a primeira é a velocidade, pois qualquer operação de *commit*,

branch, ou consulta ao histórico é realizada apenas na máquina local, sem acessar a rede (apenas as operações de sincronização com outros repositórios acessam a rede). A segunda é a tolerância a falhas, pois, como o repositório é replicado, problemas em um repositório são facilmente recuperáveis a partir das outras cópias existentes.

3.2.4.4. Mercurial

O Mercurial é um projeto de software livre que surgiu em abril de 2005. O nome "mercurial" é um adjetivo que significa "relativo a ou que tenha características (eloqüência, rapidez, inteligência) atribuídas ao deus Mercúrio" [SULLIVAN, 2009, online].

Surgiu originalmente para Linux, tendo sido portado para Windows, Mac OS X e outros sistemas. Seu código foi escrito praticamente todo em Python à exceção da implementação do diff, feita em C. O Mercurial foi concebido para projetos grandes, seus maiores objetivos incluem a alta performance, escalabilidade, descentralidade, ser totalmente distribuído de maneira colaborativa, ter um suporte robusto tanto para arquivos de texto quanto binários e capacidades avançadas de *branching* e *merging* (processo de fundir cópias de um mesmo arquivo modificado simultaneamente por mais de uma pessoa), mantendo-se conceitualmente simples [SULLIVAN, 2009, online]. Isso não significa que pequenas equipes de desenvolvimento não possam ou não devam usá-lo. Mercurial é extremamente rápido, e com o desempenho como a característica mais importante.

O Mercurial funciona basicamente via linha de comando. Seus comandos sempre começam com "hg", uma referência ao elemento químico Mercúrio, cujo símbolo é o Hg.

As ferramentas brevemente apresentadas anteriormente são exemplos de alguns SCV que poderão ser acessados pelo CompWeb através do middleware proposto nesse trabalho. Na seção 4 serão descritos os resultados obtidos na implementação do *middleware* e como ele interage com esses SCV.

3.3. Metodologia

Para o desenvolvimento desse trabalho, primeiramente, foi necessário realizar pesquisas sobre os conceitos envolvidos. Neste ponto, foi utilizado a internet, livros e artigos na busca de fontes bibliográficas. Posteriormente, foram realizadas reuniões com o

professor orientador para definição do escopo do projeto e os caminhos que deveriam ser seguidos.

Em seguida iniciou-se os estudos para identificar a carência do CompWeb em relação ao seu SCV. Várias soluções foram discutidas e em reuniões com professor orientador chegou-se ao consenso que a utilização do *middleware* seria a mais adequada.

A próxima etapa foi o estudo das tecnologias que seriam utilizadas na solução. Um amplo estudo sobre a API SVNKit foi realizado já que essa seria utilizada no desenvolvimento do *middleware.* Em paralelo com o estudo da API SVNKit foi desenvolvido o diagrama de seqüência e, em seguida, o de classes. O próximo passo foi a implementação do *middleware* propriamente dito. A implementação do *middleware* foi dividida em duas partes: primeiro foram desenvolvidas as classes de conexão com o SVC e posteriormente a parte do WebService que comunica com o CompWeb.

Por fim, foram realizados alguns testes a fim de verificar o funcionamento do *middleware* e correção de possíveis falhas.

4. RESULTADOS E DISCUSSÃO

Neste capítulo serão demonstrados os resultados obtidos no desenvolvimento do *middleware* proposto nesse trabalho.

Observou-se que o repositório de componentes CompWeb disponha de um SCV não muito prático, pois para que o desenvolvedor pudesse disponibilizar seus componentes e versões no CompWeb ele teria que cadastrar cada versão à medida que essas fossem criadas tornando essa tarefa onerosa, uma vez que esse componente já foi cadastrado. Esse re-trabalho dificultava o versionamento e amadurecimento dos componentes.

Com os estudos realizados percebeu-se que já existem vários SCV's disponíveis no mercado e comuns entre os desenvolvedores, que resolveriam essa problemática. No entanto, chegou-se a outra problemática: todos os SCV's verificados foram desenvolvidos com tecnologias diferentes do CompWeb, o que dificultaria a integração com esses sistemas. Assim, iniciaram-se novos estudos com o objetivo de buscar uma solução eficiente para a integração do CompWeb com um SCV. Após esses estudos concluiu-se que a criação de um *middleware* resolveria essa problemática devido à sua característica de interligar aplicações diferentes, como foi descrito no 2.4 desse trabalho. Portanto, optou-se por construir um *middleware* que interligasse o CompWeb a um SCV qualquer.

A seguir será detalhado como foi modelado o *middleware* para que fosse possível essa interligação.

4.1. Modelagem do Middleware

A modelagem do *middleware* proposto teve como objetivo interligar o CompWeb a um SCV qualquer, facilitando a busca de novas versões de componentes e, principalmente, tornando mais simples a tarefa de disponibilizar novas versões de um determinado componente já cadastrado no CompWeb.

A seguir será detalhado como funcionará todo o processo de busca e recuperação de componentes e suas versões, utilizando também a Figura 9 para ilustrar este processo.

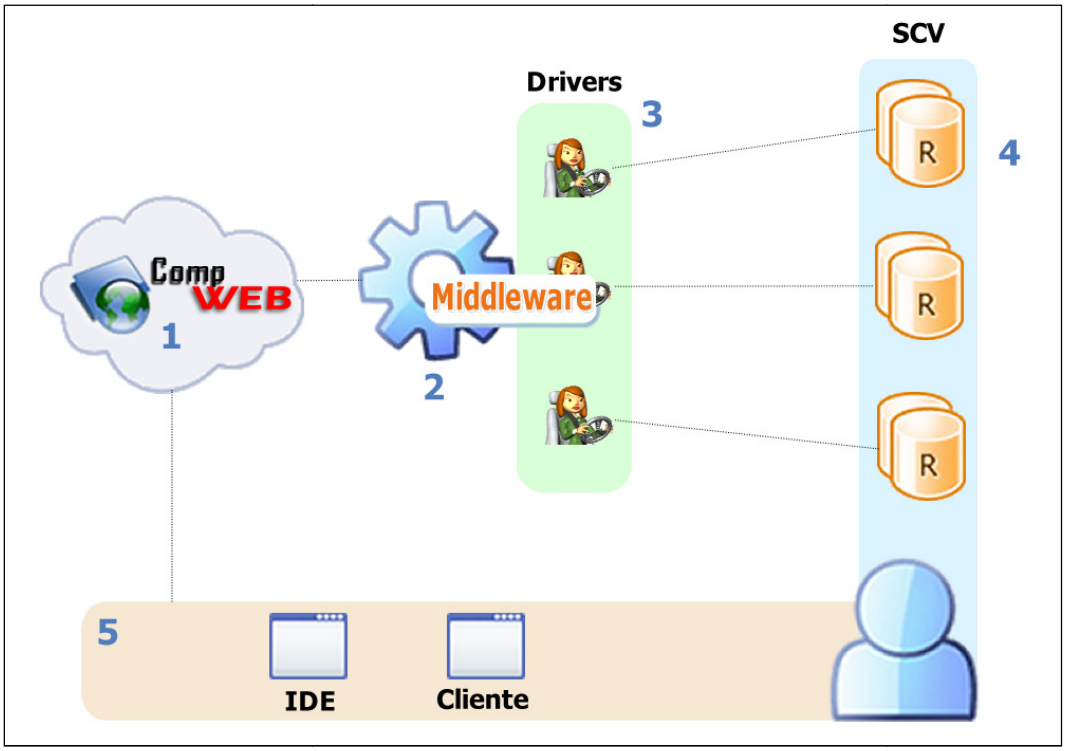

Figura 9 – Interação do CompWeb com o Middleware, os SCVs e o ambiente do desenvolvedor.

O desenvolvedor poderá criar normalmente seus componentes utilizando o seu ambiente de trabalho (5) e o SCV de sua preferência, ressaltando que não é exigido nenhum ambiente ou configuração especial para a utilização do CompWeb ou mesmo do *middleware*. Assim que terminada a primeira versão do componente, o desenvolvedor acessará o CompWeb (1), através da sua interface web e fará o cadastro do componente informando também a URL, usuário e senha do SCV onde se encontra o componente.

No momento de uma busca, o CompWeb, por usa vez, se comunicará com o *middleware* (2) através de uma interface WebService REST para obter os dados do

componente. Para que o *middleware* possa ter acesso aos SCV's (4), ele usará os *drivers* (3), interfaces específicas que farão a comunicação com cada SCV. A próxima seção detalhará cada atividade descrita na Figura 9.

4.2. Seqüência das Atividades

Essa seção apresentara detalhadamente cada atividade no processo de cadastro, busca e recuperação de componentes e suas versões.

A Figura 10 mostra a tela onde será feito o cadastro do componente, informando dados como: o nome do componente, a linguagem, a plataforma, uma breve descrição do componente, as tags relacionadas, a URL do SCV utilizado, o usuário e a senha para acesso ao SCV. Se o componente estiver em algum diretório específico dentro do SCV, será necessário informá-lo no momento do cadastro do componente.

Figura 10 - Tela de cadastro de componentes no CompWeb.

Depois de efetuado o cadastro, o CompWeb armazenará essas informações e, assim, quando um utilizador de componente buscar por componentes ele o fará na área de busca do CompWeb, ilustrada na Figura 11. Para isso, o utilizador de componentes informará o nome, a linguagem, as tags ou a plataforma do componente, não sendo obrigatório informar todos esses dados, mas pelo menos um desses. Assim que o utilizador de componentes fizer a busca, o CompWeb buscará em sua base de dados os componentes relacionados com os dados informados, retornando uma lista de componentes para o utilizador.

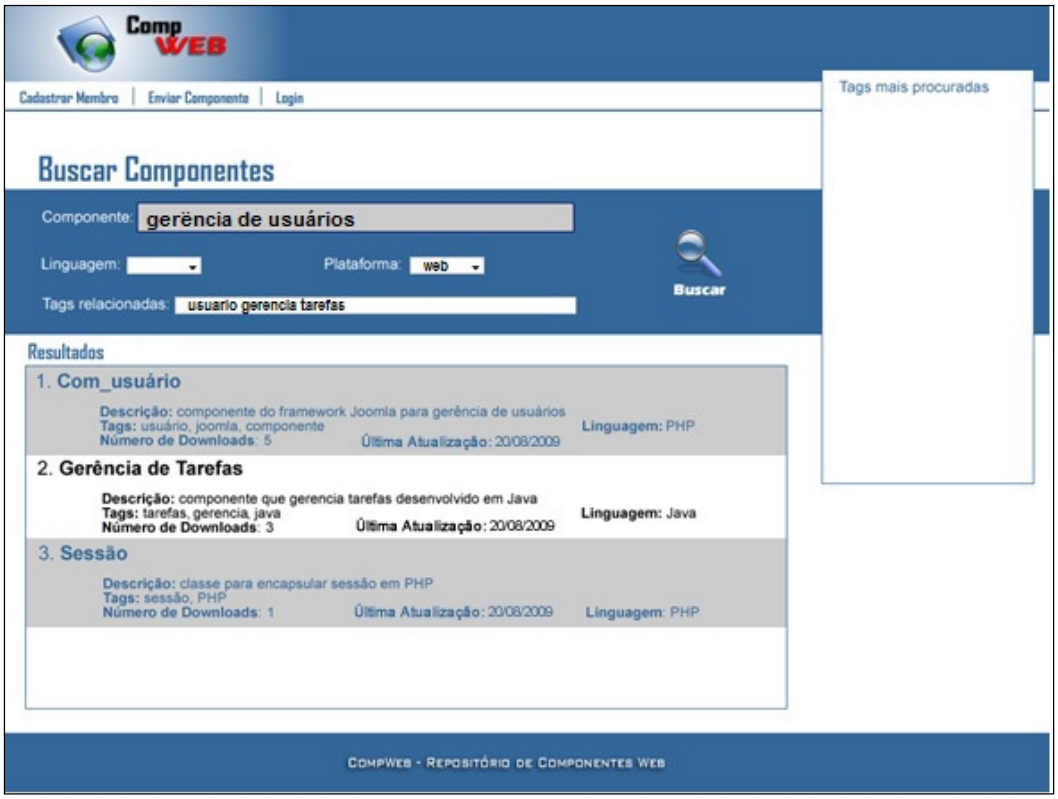

Figura 11 - Área de Busca do CompWeb, com o resultado de uma busca de componentes.

O utilizador poderá selecionar um dos componentes da lista, ação que fará com que o CompWeb informe para o *middleware* o nome do componente, o nome e a URL do SCV, o usuário, a senha e o diretório onde se encontra o componente, caso haja. A senha passará por uma criptografia para maior segurança e será descriptografada pelo middleware. Esses dados são enviados na forma de um arquivo XML, exemplificado na Figura 12, que é recebido pelo *middleware* por meio de sua interface Web Service REST.

```
- <componentes>
  - <componente>
      <nome>Com_user.rar</nome>
      <scv>SVN</scv>
      <url>http://200.129.179.71/svn/intranet/</url>
      <usuario>Lefer</usuario>
      <senha>qwe34fd^43</senha>
      <diretorio>producao</diretorio>
    </componente>
  </componentes>
```

Figura 12 - XML que contém as informações do componente solicitado.

O *middleware*, por sua vez, interpretará o XML recebido e verificará qual o SCV informado no elemento <*scv*> e o acessará através de uma classe específica para esse SCV, passando como parâmetro para essa classe a URL, usuário e senha contidos nos elementos <*url*>, <*usuario*> e <*senha*> respectivamente. As classes específicas para cada SCV foram chamadas de *drivers*, pois elas funcionaram como interfaces de conexão com cada SCV. Assim, para cada SCV haverá um *driver* específico. Depois de selecionado o *driver* adequado ao SCV informado, o *middleware* acessará o SCV através do *driver* e buscará os arquivos disponíveis no diretório informado pelo elemento <*diretório*> dentro do repositório informado pelo desenvolvedor daquele componente.

A Figura 13 mostra o processo de busca e recuperação de componentes, e as interações entre CompWeb, o *middleware* e o SCV.

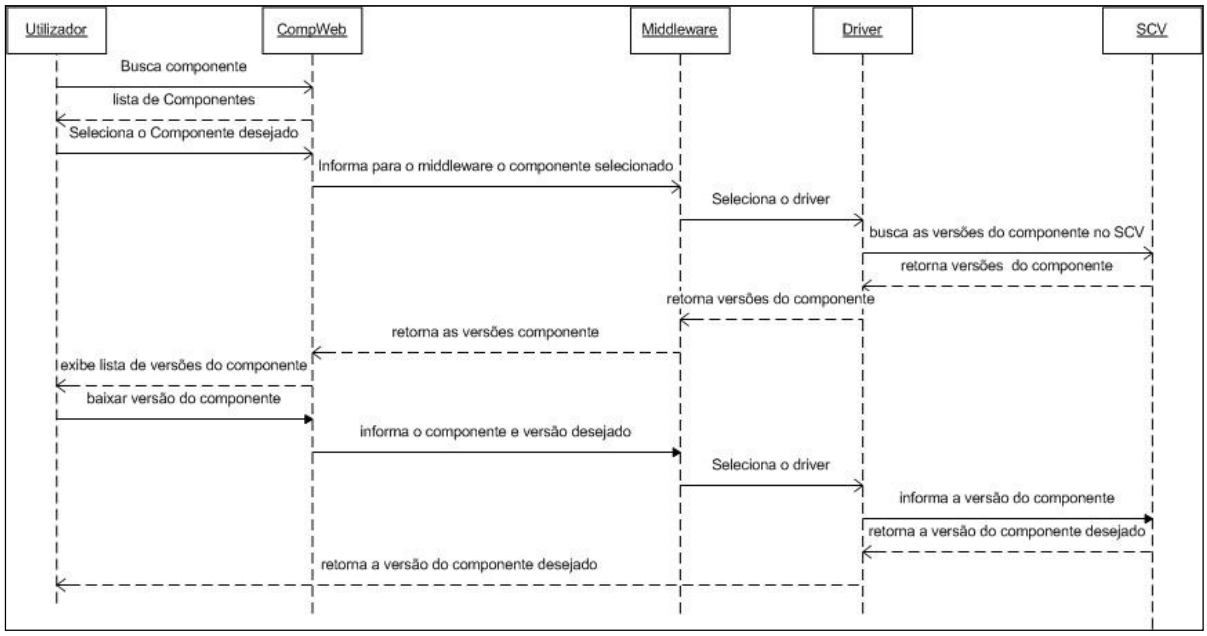

Figura 13 - Diagrama de Seqüência do processo de busca e recuperação de componente.

No processo de busca de componentes, o *middleware*, após conectar com o SCV, retornará as versões disponíveis do componente informado em forma de arquivo XML para o CompWeb, que exibirá esses dados através da sua interface web. Além das versões, o *middleware* trará informações como a data da versão, dado esse que poderá ser importante para o utilizador de componentes.

Já no processo de recuperação de versão, após ver as versões disponíveis, o utilizador selecionará a versão que desejar. O CompWeb enviará novamente para o *middleware* um arquivo XML semelhante ao utilizado na busca, acrescentando apenas o

elemento <*versao*>, que informará qual a versão desejada pelo utilizador de componentes.

4.3. Diagrama de Classes

A implementação das classes seguiu alguns padrões de projeto como o *Factory*, como pode ser observado na classe *DriverFactory*. Foram utilizados também conceitos de orientação a objetos como herança e polimorfismo que podem ser notados nas classes *Driver*,*SVN* e *GIT*. Essas características podem ser observadas no diagrama de classes representando na Figura 14.

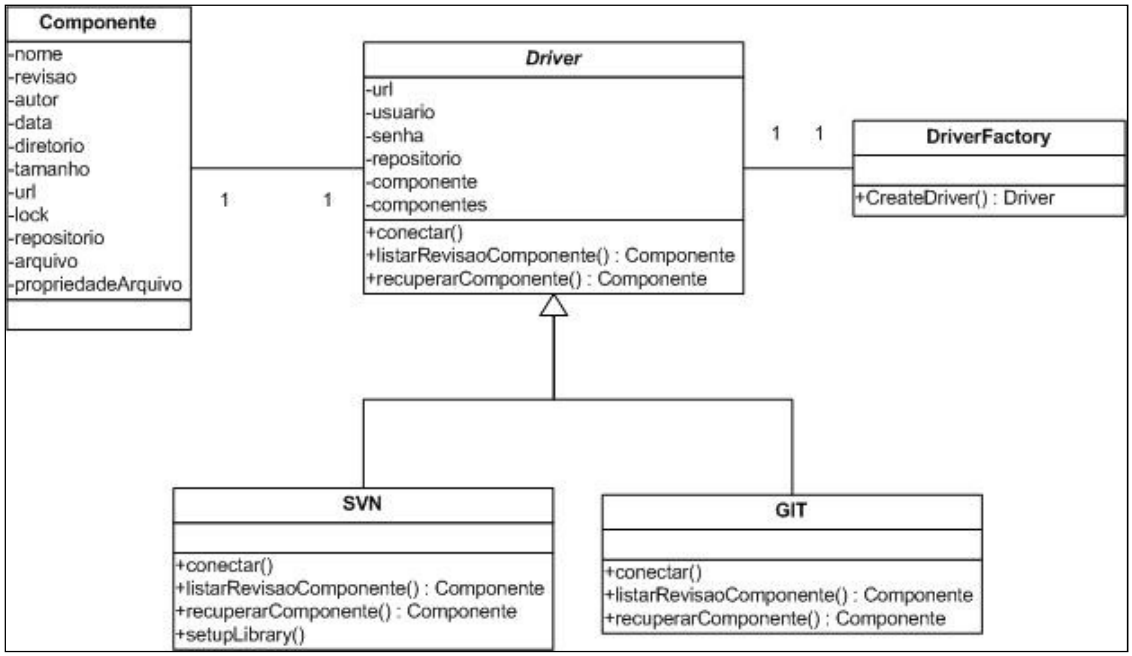

Figura 14 - Diagrama de Classes do Middleware.

O *middleware* proposto foi criado de forma que pudesse acessar qualquer SCV. Como pode ser observado no diagrama de classes da Figura 14, foi implementada uma classe abstrata denominada *Driver*, que generaliza todas as classes que conectam aos SCV. Assim, basta que essas classes de conexão estendam a classe *Driver* e sobrescrevam os métodos abstratos *conectar*, *listarRevisaoComponente* e *recuperarComponente.* O método *conectar* estabelece a conexão com o repositório do SCV. O método *listarRevisaoComponente*, que recebe um objeto do tipo *Componente*, retorna uma lista das versões de um componente, informando também a data de alteração. O método *recuperarComponente,* que recebe como parâmetro um objeto do

tipo *Componente* e a versão que deseja recuperar, é responsável por recuperar uma versão de um componente.

A classe *DriverFactory* é responsável por instanciar a classe de um determinado SCV informado pelo CompWeb. A classe *DriverFactory* tem um método chamado *createDriver* que recebe como parâmetro o nome do SCV, a URL do SCV, um usuário e uma senha para acesso ao repositório do SCV. Ao chamar esse método, a classe *DariverFactory* já faz a instância do *driver* desejado e a conexão ao repositório informado. A implementação da classe *DriverFactory* pode ser visualizada na Figura 15.

```
1  package middleware.main;
2
3  import driver.*;
4  import driver.SVN;
5
6  /**
7   * Classe responsável por selecionar e instanciar a classe informada
8   * @author Leandro
9   */
10 public class DriverFactory {
11
12     /**
13      *
14      * Método responsável por criar uma instância do driver desejado
15      *
16      * @param name String
17      * @param url String
18      * @param usuario String
19      * @param senha String
20      * @return Driver
21      */
22     public Driver CreateDriver(String name, String url, String usuario, String senha) {
23
24         if (name.equalsIgnoreCase("GIT")) {
25             return new Git(url, usuario, senha);
26         }
27         else if (name.equalsIgnoreCase("SVN")){
28            return new SVN(url, usuario, senha);
29         }
30         else{
31            return new SVN(url, usuario, senha);
32         }
33     }
34 }
```

Figura 15 - Implementação da classe DriverFactory

A classe *Componente* representa o componente propriamente dito. Ela tem como atributos: nome, revisão, autor, data, local, diretório, tamanho, url, repositório, arquivo e propriedadeArquivo.

A implementação das demais classes estará disponível nos anexos desse trabalho.

4.4. A implementação do *driver* SVN

Nesse projeto foi implementado somente o *driver* para o SVN. Mas o *middleware* está preparado para utilizar qualquer *driver* para qualquer SCV desde que esses sigam o padrão proposto que é estender a classe abstrata *Driver* e implementar os seus métodos abstratos.

A classe ou *driver* SVN foi implementada usando uma API (*Application Programming Interface*) chamada SVNKit [SVNKIT, 2009] que foi criada utilizando a linguagem Java. A SVNKit é gratuita e implementa todas a funcionalidades para trabalhar com o SVN. Com essa API é possível acessar e manipular qualquer repositório SVN.

A seguir é exibida a implementação do *driver* SVN.

```
1  public class SVN extends Driver {
2
3      public SVN(){
4          super();
5          setupLibrary();
6      }
7      public SVN(String url, String usuario, String senha) {
8          super(url, usuario, senha);
9
10         try {
11             this.conectar();
12         }
13         catch (SVNException ex) {
14             System.out.println(ex.getMessage());
15         }
16     }
17     public void conectar() throws SVNException{
18        this.getComponente().setRepositorio( SVNRepositoryFactory.create
19        ( SVNURL.parseURIDecoded( this.getUrl() ) ) );
20        ISVNAuthenticationManager authManager = SVNWCUtil.createDefaultAuthenticationManager
21        ( this.getUsuario() , this.getSenha() );
22        this.getComponente().getRepositorio().setAuthenticationManager( authManager );
23     }
24
25    public ArrayList<Componente> listarRevisaoComponente(Componente c){
26        ArrayList<Componente> comp = new ArrayList<Componente>();
27         try{
28             Collection entries = c.getRepositorio().getDir(c.getDiretorio()+c.getNome(),
29             -1, null, (Collection) null);
30             Iterator iterator = entries.iterator();
31             while (iterator.hasNext()) {
32                 SVNDirEntry entry = (SVNDirEntry) iterator.next();
33                 if (entry.getName().equals(c.getNome()) ) {
34                     comp.add(new Componente(entry.getName(), entry.getRelativePath(), entry.getAuthor(),
35                     entry.getRevision(), entry.getSize(), String.valueOf(entry.getURL()), entry.getDate()));
36                 }
37             }
38         }
39                                                    .
40                                                    .
41                                                    .
```

Figura 16 - Implementação da classe SVN

A classe SVN, como descrito anteriormente, é formada pelos métodos: *conectar*, *listarRevisaoComponente*, *recuperarComponente*; sendo esses herdados da classe *Driver*.

O método conectar é responsável pela conexão com o repositório SVN. Esse método cria a conexão com o repositório através da classe *SVNRepositoryFactory* que chama o método *create* que recebe como parâmetro a URL do repositório. É importante ressaltar que essa classe *SVNRepositoryFactory* faz parte da API do SVNKit. Logo em seguida é feita a autenticação do usuário no repositório através da instrução *SVNWCUtil.createDefaultAuthenticationManager* que recebe como parâmetro o usuário e a senha de acesso ao repositório. Assim, a conexão está estabelecida e autenticada com o repositório. A autenticação é salva em um objeto do tipo *repository* que será utilizado para buscar e recuperar componentes.

O método *listarRevisaoComponente* recebe como parâmetro um objeto do tipo *Componente* e retorna uma lista de componentes. Essa lista conterá todas as versões disponíveis do componente. Isso é possível porque a API SVNKit permite recuperar todos os dados de um item do repositório incluindo a versão, autor, data de modificação e o arquivo propriamente dito. Esses dados são recuperados através do método *getDir* da classe *SVNRepository* que recebe como parâmetro o diretório onde está o componente, o nome do componente, a versão do componente e o tipo de retorno, que nesse caso será uma lista de Componentes.

O método *recuperarComponente* é responsável por recuperar uma versão de um componente armazenada no repositório. Ele recebe como parâmetro um objeto do tipo *Componente* e a versão que deseja recuperar e retorna um objeto do tipo *Componente*. Esse objeto retornado tem todos os dados incluindo o arquivo binário desse componente. Esse arquivo binário, em uma operação de recuperação de componente, será retornado para o utilizador de componentes.

5. CONSIDERAÇÕES FINAIS

O DBCS tem contribuído na busca por maior produtividade e menor custo no desenvolvimento de software. Essa busca fez com que surgissem inúmeras necessidades como: aperfeiçoamento das técnicas de reutilização de software; desenvolvimento de ambientes propícios para o armazenamento e busca de artefatos produzidos durante o ciclo de vida do projeto; sistemas que permitissem o mapeamento juntamente com o histórico de modificações dos artefatos e principalmente, a utilização do conhecimento adquirido em projetos já finalizados em novos contextos.

O aperfeiçoamento das técnicas de reutilização é percebido através de mudanças no modo de se desenvolver software, como por exemplo: aplicações antes construídas em blocos de códigos monolíticos estão sendo substituídas por softwares baseado em componentes. Essa mudança implica em softwares mais organizados, mais fáceis de fazer manutenção além de poder reutilizar componentes já construídos em novos projetos.

Atualmente, existe um grande número de equipes de desenvolvimento de software utilizando o DBCS. Isso fez com que fossem criados vários repositórios de códigos, componentes e softwares disponíveis na internet. Esses repositórios nem sempre atendem a todas as necessidades dos desenvolvedores deixando a desejar em alguns pontos como, por exemplo, o Gerenciamento de Configuração de Software.

Gerenciamento de Configuração de Softwares (GCS) é uma disciplina da engenharia de software responsável por identificar, organizar e controlar modificações no software que está sendo construído.

Sistema de Controle de Versão (SCV), uma importante área do GCS responsável por historiar todas as modificações e permitir que em qualquer momento

dentro do projeto seja possível acessar versões de um artefato, nem sempre são contempladas por repositórios.

O presente trabalho trouxe alguns exemplos de repositórios e algumas funcionalidades que facilitassem a utilização desses. Funcionalidades essas que não impactassem no ambiente de trabalho do desenvolvedor, mas que automatizasse a liberação de novas versões estáveis por meio da integração com SCV's.

Foi utilizado o repositório CompWeb nesse trabalho, por se tratar de um ambiente web, que permite o versionamento de qualquer tipo de artefato. O CompWeb disponibiliza mais de um meio de acesso e possui um sistema de busca simples que permite algumas opções de filtros. Porém, como foi verificado em inúmeros repositórios, o CompWeb não dispõe de um SCV, sendo apenas possível indicar as versões do artefato.

Dentro deste contexto foi proposto um *middleware* que integrasse as funcionalidades de um Sistema de Controle de Versão (SCV), como SVN (*Subversion*) com o repositório CompWeb. Essa integração permite que o desenvolvedor crie versões de seus componentes, direto do seu ambiente de trabalho ou ainda utilizando um cliente de SCV.

Assim, o desenvolvedor poderá acessar os componentes por ele submetido ao CompWeb e fazer correções e atualizações sem se preocupar com versões, uma vez que essa tarefa será executada pelo sistema de controles de versão, e ainda poderá disponibilizar o componente para download.

Por outro lado, os utilizadores de componentes podem pesquisar e utilizar os componentes do CompWeb que através do *middleware* acessa a área de versionamento e busca pelos componentes e suas verões disponíveis para download.

É proposto como trabalho futuro a adaptação do *middleware* para que esse possa carregar os *drivers* na forma de plugins. Isso faria com que não fosse necessário implementar os *drivers* estaticamente. Devido a limitação de tempo, dada a necessidade de estudar a API SVNKit [SVNKit, 2009] para que fosse possível a implementação do *driver* SVN, não foi possível a adequação da arquitetura do *middleware* para suporte a plugins.

6. REFERÊNCIAS BIBLIOGRÁFICAS

[BAB, 1986] Babich, W. A., Software Configuration Management, Addison-Wesley, 1986.

[BACON, 2002] Bacon, J. and Moody, K., Toward Open, Secure, Widely Distributed Services, Communication of the ACM,Vol. 45, No. 6, Jun 2002.

[BERNSTEIN, 1996] Bernstein, Philip A. Middleware: A Model for Distributed System Services Communications of the ACM. V.39, nº 2.p.86-98. Fev 1996.

[BISHOP, 2002] Toni A. A Survey Of Middlewares. A thesis presented to the Faculty of Towson University. Tonson, Maryland – EUA. August 2002.

[BRAGA, 2000] Braga, R.; Werner, C.; Mattoso, M.; - Uma infra-estrutura de Reutilização baseada em Modelos de Domínio - João Pessoa, outubro 2000.

[BRECHÓ, 2009] Site do Laboratório de Engenharia de Software – Equipe de Reutilização de Software. Disponível em: HTTP://reuse.cos.ufrj.br/site/index.php ?option=com_content&task=view&id=37&Itemid=47. Acesso em 27 de setembro de 2009.

[CERUTI, 1999] Ceruti, M. and Thuraisingham, B., Dependable Objects for Databases, Middleware and Methodologies: A Position Paper, *Proceedings of the 5th International Workshop* on Object-Orient Real-Time Dependable Systems, IEEE, Nov 1999.

[CHARLES, 1999] Charles, John. *Middleware Moves to the Forefront.* Computer, Maio 1999. Páginas. 17-19.

[CHRISTENSEN & THAYER, 2002] Christensen, M. J.; Thayer, R. H. – *The Project Manager's Guide to Software Engineering's Best Practices*. 1 ed., IEEE Computer Society Press and John Wiley & Sons.

[COLLABNET, 2009] Site da empresa Collab.ner – 55HTTP://www.collab.net – acessado em 20/11/2009

[COLLINS-SUSSMAN, 2003] Collins-Sussman, B.; Fitzpatrick B. W., and Pilato C.M.; Version Control with Subversion. O'Reilly Media, 2003.

[CVS, 2009] Site do CVS – disponível em: HTTP://www.cvshome.org/ - acesso em 27 de setembro de 2009.

[DELICIOUS, 2009] Site do delicius.com – disponível em: HTTP://delicious.com/- acesso em 27 de setembro de 2009.

[DONOVAN & GRIMSON, 1990] Donovan, B. and Grimson, J. B. A distributed version control system for wide area networks. Software Engineering Journal, Setembro 1990.

[ECLIPSE, 2009] "Eclipse.org Main Page". In: http://www.eclipse.org, Acessado em 10/11/2009.

[EMMERICH, 2000] Emmerich, W. Software engineering and middleware: A roadmap. Communications of the ACM, pages 117-129, 2000.

[EZRAN, 1999] Ezran, M., et al.. Practical Software Reuse: The Essencial guide.1999.185 p

[FERREIRA, 2009] Ferreira, L.; CompWeb – Repositório de Componentes Web – Relatório Final de Estágio – Centro Universitário Luterano de Palmas – julho de 2009 – Palmas, Tocantins, Brasil

[FILHO, 2001] Filho, W. P. P.; Engenharia de Software – Fundamentos, Métodos e Padrões – ISBN 9-788521-612605. Páginas 452-453. Rio de Janeiro – 2001.

[GEIHS, 2001]. GEIHS, K., Middleware Challenges Ahead, IEEE Computer, Vol. 34, Issue 6, Jun 2001.

[GIT, 2009] Site do GIT – disponível em: HTTP://git-scm.com/ - acesso em 27 de setembro de 2009.

[GOOGLECODE, 2009] Site do GoogleCode – 56HTTP://code.google.com/intl/HT-BR/ - acessado em 15/11/2009

[HARTWICH, 2003] HARTWICH, Christoph. A Middleware Architecture for Transactional, Object-Oriented Applications. Dissertação defendida na Universidade de Berlin.
Nov-2003.

[HOLANDA & SOUZA, 2006] Holanda, Caroline Batista Spencer, Souza, Clarissa Angélica de Almeida de, Walcélio L. Melo. ProReuso: Um Repositório de Componentes para Web Dirigido por um Processo de Reuso – XV Simpósio Brasileiro de Engenharia de Software

[JOOMLA, 2009] Site do Joomla – 56HTTP://www.joomla.org/ - acessado em 20/11/2009

[JSF, 2009] Site da Sun – 56HTTP://java.sun.com/javaee/javaserverfaces/?intcmp=3278 – acessado em 20/112009

[KODERS, 2009] Site do Koders – 56HTTP:// koders.com/ - acessado em 15/11/2009

[KONN, 2002] Konn, F. Costa, F. Blair, G. and Campbell, R. H.. The case of reflective middleware. Communication. ACM, 45(6):33–38, 2002

[LEYTON, 2005] Leyton R. Version Control – A Misunderstood technology. Report, 2005.

[MARC, 1975] Marc, J. – The source code control system. IEEE Transactions on Software Engineering. SE-1 (4), Jan 1975.

[MAVEN, 2009] Site do Apache Maven – 56HTTP://maven.apache.org/ - acessado em 16/11/2009

[MERCURIAL, 2009] Site do Mercurial – disponível em: HTTP://mercurial.selenic.com/wiki/ - acesso em 27 de setembro de 2009.

[MURTA, 2004] Leonardo, G. P. – ODYSSEY-SCM: Uma Abordagem de Gerência de Configuração de Software para o Desenvolvimento Baseado em Componentes – Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia - Universidade Federal do Rio de Janeiro – Maio de 2004.

[NETBEANS, 2009] "Welcome to NetBeans". In: HTTP://www.netbeans.org, Acessado em 10/11/2009.

[NUNN, 2009] NUNN, R.. Distributed software architectures using middleware. HTTP://www.cs.ucl.ac.uk/staff/W.Emmerich/lectures/3C05-02-03/aswe18-essay.pdf. Acessado em 11/11/2009.

[OLIVEIRA, 2008] Oliveira, João Paulo Freitas de. X-CORE: um serviço de repositório distribuído e compartilhado de componentes de software – Dissertação de Mestrado do Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba. João Pessoa, 2008.

[OMG, 2009] Site da OMG – 57HTTP://www.omg.org – acessado em 21/11/2009

[PETERS & PEDRYCZ, 2001] Peters, F. J., Pedrycz W.; tradução de Ana Patrícia Garcia – Engenharia de Software . ISBN 85-352-0746-5. Páginas. 67-81. Rio de Janeiro: Elsevier, 2001.

[PRESSMAN, 1995] Pressman, Roger S.; tradução José Carlos Barbosa dos Santos – Engenharia de Software. ISBN 9-788534-60237-2. Páginas 916-940. São Paulo 1995.

[ROSA, 2009] Rosa, N. Souto – Ambientes de Middlewares – Universidade Federal de Pernambuco. Disponível em 57HTTP://www.cin.ufpe.br/~sd/disciplinas/sd/pos/aulas/Middleware.pdf acessado em 19/11/2009

[SCHMIDT, 2001] Schmidt, D. C.; Schantz , R. E.; Masters, M. W.; Cross, J. K.; Sharp, D. C.; DiPalma, L. P.; - Toward Adaptive and Reflective Middleware for Network-Centric Combat Systems – Naval Surface Warfare Center Dahlgren – publicado em 01-Nov-2001.

[SCHUENCK, 2005] Schuenck, M.; Tanure, C.; Dias, J. L.; Miranda, S.; Negócio, Y.; Elias, G. – Análise de Repositórios de Componentes para a Identificação das Soluções mais Adotadas – Workshop de Desenvolvimento Baseado em Componentes. Juiz de Fora, Minas Gerais, 2005.

[SOURGEFORGE, 2009] Site do Sourgeforge – 57HTTP://sourceforge.net/ - acessado em 15/11/2009

[SUBMARINO, 2009] Site do submarino – disponível em: HTTP://.submarino.com.br – acessado em 08 de novembro de 2009

[SULLIVAN, 2009] O'Sullivan, Bryan. Mercurial: The definitive guide. http://hgbook.redbean.com/. Publicado em 2009 por O'Reilly Media.

[SVN, 2009] Site do Subversion – disponível em: HTTP://subversion.tigris.org/ - acesso em 27 de setembro de 2009.

[SVNKIT, 2009] Site da API SVNKit - HTTP://svnkit.com/ - acessado em 29/11/2009

[SZYP, 1998] C. SZYPERSKI, *Component Software: Beyond Object-Oriented Programming,* Addison-Wesley, New York, 1998.

[TALARIAN, 2009] TALARIAN: Everything you need to know about middleware. A Guide To Selecting A Real-Time Infrastructure HTTP://searchwebservices.techtarget.com/searchWebServices/downloads/Talarian.pdf. Acessado em 10/11/2009.

[TIZZEI, 2007] Tizzei , L. P.; Uma infra-estrutura de suporte à evolução para repositórios de componentes – redação da Dissertação apresentada para a Banca Examinadora antes da defesa da dissertação. – Campinas 2 de março de 2007 – SP – Brasil

[TORVALDS, 2009] Torvalds, L.; "The Linux Home Page at Linux Online". In: HTTP://linux.org/, Acessado em 12/11/2009.

[TROMBETTA, 2008] Trombetta, André. , Korogi, Genilson., Peralta, Karine; artigo Sistema de Controle de Versões – Faculdade de Informática – Pontifícia Universidade Católica do Rio Grande do Sul – Porto Alegre – RS – Brasil – 2008

[WIRES, FEELEY, 2007] Wires, J. and Feeley, M.; Secure File System Versioning at the Block Level. ACM SIGOPS Operating Systems Review. Proceedings of the 2007 – Conference on EuroSys EuroSys '07, Volume 41 Issue 3.

ANEXOS

Anexo I – Código fonte da classe Componente

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package middleware.main;

import java.io.ByteArrayOutputStream;
import java.util.Date;
import org.tmatesoft.svn.core.SVNProperties;
import org.tmatesoft.svn.core.io.SVNRepository;

/**
 *
 * @author Leandro
 */
public class Componente {

    private String nome;
    private String diretorio;
    private String autor;
    private long revisao;
    private long tamanho;
    private String url;
    private Date data;
    private boolean lock;
    private SVNRepository repositorio;
    private       ByteArrayOutputStream       arquivo       =       new
ByteArrayOutputStream();
    private String urlArquivo;
    private SVNProperties propriedadeArquivo = new SVNProperties();

    public Componente() {
    }

    public  Componente(String  nome,  String  diretorio,  String  autor,
long revisao, long tamanho, String url, Date data) {
        this.nome = nome;
        this.diretorio = diretorio;
        this.autor = autor;
        this.revisao = revisao;
        this.tamanho = tamanho;
        this.url = url;
        this.data = data;
    }

    /**
     * @return the nome
     */
    public String getNome() {
        return nome;
    }

    /**
     * @param nome the nome to set
```

```
     */
    public void setNome(String nome) {
        this.nome = nome;
    }

    /**
     * @return the diretorio
     */
    public String getDiretorio() {
        return diretorio;
    }

    /**
     * @param diretorio the diretorio to set
     */
    public void setDiretorio(String diretorio) {
        this.diretorio = diretorio;
    }


    /**
     * @return the autor
     */
    public String getAutor() {
        return autor;
    }

    /**
     * @param autor the autor to set
     */
    public void setAutor(String autor) {
        this.autor = autor;
    }



    /**
     * @return the tamanho
     */
    public long getTamanho() {
        return tamanho;
    }

    /**
     * @param tamanho the tamanho to set
     */
    public void setTamanho(int tamanho) {
        this.setTamanho(tamanho);
    }

    /**
     * @return the url
     */
    public String getUrl() {
        return url;
    }

    /**
     * @param url the url to set
     */
```

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

    /**
     * @return the data
     */
    public Date getData() {
        return data;
    }

    /**
     * @param data the data to set
     */
    public void setData(Date data) {
        this.data = data;
    }

    /**
     * @return the lock
     */
    public boolean isLock() {
        return lock;
    }

    /**
     * @param lock the lock to set
     */
    public void setLock(boolean lock) {
        this.lock = lock;
    }

    /**
     * @return the revisao
     */
    public long getRevisao() {
        return revisao;
    }

    /**
     * @param revisao the revisao to set
     */
    public void setRevisao(long revisao) {
        this.revisao = revisao;
    }

    /**
     * @param tamanho the tamanho to set
     */
    public void setTamanho(long tamanho) {
        this.tamanho = tamanho;
    }

    /**
     * @return the repositorio
     */
    public SVNRepository getRepositorio() {
        return repositorio;
    }
```

```
    /**
     * @param repositorio the repositorio to set
     */
    public void setRepositorio(SVNRepository repositorio) {
        this.repositorio = repositorio;
    }

    /**
     * @return the arquivo
     */
    public ByteArrayOutputStream getArquivo() {
        return arquivo;
    }

    /**
     * @param arquivo the arquivo to set
     */
    public void setArquivo(ByteArrayOutputStream arquivo) {
        this.arquivo = arquivo;
    }

    /**
     * @return the urlArquivo
     */
    public String getUrlArquivo() {
        return urlArquivo;
    }

    /**
     * @param urlArquivo the urlArquivo to set
     */
    public void setUrlArquivo(String urlArquivo) {
        this.urlArquivo = urlArquivo;
    }

    /**
     * @return the propriedadeArquivo
     */
    public SVNProperties getPropriedadeArquivo() {
        return propriedadeArquivo;
    }

    /**
     * @param propriedadeArquivo the propriedadeArquivo to set
     */
    public           void           setPropriedadeArquivo(SVNProperties
propriedadeArquivo) {
        this.propriedadeArquivo = propriedadeArquivo;
    }


}
```

Anexo II – Código fonte da classe Driver

```
package middleware.main;

import java.util.ArrayList;
import org.tmatesoft.svn.core.io.SVNRepository;

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
/**
 *
 * @author Leandro
 */
public abstract class Driver {

    private String url;
    private String usuario;
    private String senha;
    private SVNRepository repositorio;
    private String diretorio;
    private Componente componente = new Componente();
    private ArrayList<Componente> componentes;

    public Driver() {
    }

    /**
     * @param String url, String usuario, String senha, SVNRepository
repository
     *
     */
    public Driver(String url, String usuario, String senha) {
        this.url = url;
        this.usuario = usuario;
        this.senha = senha;
    }

    /**
     * @return the url
     */
    public String getUrl() {
        return url;
    }

    /**
     * @param url the url to set
     */
    public void setUrl(String url) {
        this.url = url;
    }

    /**
     * @return the usuario
     */
    public String getUsuario() {
        return usuario;
    }
```

```
    /**
     * @param usuario the usuario to set
     */
    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }

    /**
     * @return the senha
     */
    public String getSenha() {
        return senha;
    }

    /**
     * @param senha the senha to set
     */
    public void setSenha(String senha) {
        this.senha = senha;
    }

    /**
     * @return the repository
     */
    public SVNRepository getRepositorio() {
        return repositorio;
    }

    /**
     * @param repository the repository to set
     */
    public void setRepositorio(SVNRepository repositorio) {
        this.repositorio = repositorio;
    }

    /**
     * @return the diretorio
     */
    public String getDiretorio() {
        return diretorio;
    }

    /**
     * @param diretorio the diretorio to set
     */
    public void setDiretorio(String diretorio) {
        this.diretorio = diretorio;
    }

    /**
     * @return the componente
     */
    public Componente getComponente() {
        return componente;
    }

    /**
     * @param componente the componente to set
     */
    public void setComponente(Componente componente) {
```

```
        this.componente = componente;
    }

    /**
     * @return the componentes
     */
    public ArrayList<Componente> getComponentes() {
        return componentes;
    }

    /**
     * @param componentes the componentes to set
     */
    public void setComponentes(ArrayList<Componente> componentes) {
        this.componentes = componentes;
    }

    public abstract ArrayList<Componente> listaRevisaoDiretorio(
Componente c);

    public abstract ArrayList<Componente> listaRevisaoComponente(
Componente c);

    public abstract Componente recuperaComponente(Componente c, long
revisao);

    public abstract void conectar();
}
```

Anexo III – Código fonte da classe Factory

```
package middleware.main;

import driver.*;
import driver.SVN;

/**
 * Classe responsável por selecionar e instanciar a classe informada
 * @author Leandro
 */
public class Factory {

    /**
     *
     * Método responsável por criar uma instância do driver desejado
     *
     * @param name String
     * @param url String
     * @param usuario String
     * @param senha String
     * @return Driver
     */
    public Driver CreateDriver(String name, String url, String
usuario, String senha) {

        if (name.equalsIgnoreCase("GIT")) {
            return new Git(url, usuario, senha);
        }
        else if (name.equalsIgnoreCase("SVN")){
           return new SVN(url, usuario, senha);
        }
        else{
           return new SVN(url, usuario, senha);
        }
    }
}
```

Anexo IV – Código fonte da classe SVN

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package driver;

import middleware.main.Componente;
import middleware.main.Driver;
import java.io.*;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.tmatesoft.svn.core.*;
import org.tmatesoft.svn.core.io.*;
import org.tmatesoft.svn.core.auth.ISVNAuthenticationManager;
import org.tmatesoft.svn.core.internal.io.dav.DAVRepositoryFactory;
import org.tmatesoft.svn.core.internal.io.fs.FSRepositoryFactory;
import
org.tmatesoft.svn.core.internal.io.svn.SVNRepositoryFactoryImpl;
import org.tmatesoft.svn.core.wc.*;

/**
 *
 * @author Leandro
 */
public class SVN extends Driver {

    public SVN(){
        super();
        setupLibrary();
    }

    public SVN(String url, String usuario, String senha) {
        super(url, usuario, senha);

        try {
            this.conectar();
        }
        catch (SVNException ex) {
            Logger.getLogger(SVN.class.getName()).log(Level.SEVERE,
null, ex);
            System.out.println(ex.getMessage());
        }
    }

     /*
     * faz a conexão com o repositorio informado
     * @param
     */
    public void conectar() throws SVNException{
       this.getComponente().setRepositorio(
SVNRepositoryFactory.create( SVNURL.parseURIDecoded( this.getUrl() ) )
);
       ISVNAuthenticationManager             authManager             =
SVNWCUtil.createDefaultAuthenticationManager(    this.getUsuario()    ,
this.getSenha() );
```

```
       this.getComponente().getRepositorio().setAuthenticationManager(
authManager );
    }

   public ArrayList<Componente> listaRevisaoComponente(Componente c){
       ArrayList<Componente> comp = new ArrayList<Componente>();
        try{
            Collection                    entries                    =
c.getRepositorio().getDir(c.getDiretorio()+c.getNome(),    -1,    null,
(Collection) null);
            Iterator iterator = entries.iterator();
            while (iterator.hasNext()) {
                SVNDirEntry entry = (SVNDirEntry) iterator.next();
                if (entry.getName().equals(c.getNome()) ) {
                    comp.add(new           Componente(entry.getName(),
entry.getRelativePath(),    entry.getAuthor(),    entry.getRevision(),
entry.getSize(), String.valueOf(entry.getURL()), entry.getDate()));
                }
            }
        }
        catch(SVNException ex){
            ex.printStackTrace();
        }
        return comp;
   }

    public Componente recuperaComponente(Componente c, long revisao) {
        try {
            SVNNodeKind                   nodeKind                    =
c.getRepositorio().checkPath(c.getDiretorio()+c.getNome(), -1);

            if (nodeKind == SVNNodeKind.NONE) {
                System.err.println("There   is   no   entry   at   '"   +
c.getUrl()+c.getDiretorio()+c.getNome() + "'.");
                System.exit(1);
            } else if (nodeKind == SVNNodeKind.DIR) {
                System.err.println("The     entry     at     '"     +
c.getUrl()+c.getDiretorio()+c.getNome()  +  "'  is  a  directory  while  a
file was expected.");
                System.exit(1);
            }

            c.getRepositorio().getFile(c.getDiretorio()+c.getNome(),
revisao, c.getPropriedadeArquivo(), c.getArquivo());

            FileOutputStream            arq           =            new
FileOutputStream("C:\\Users\\Leandro\\Desktop\\"+c.getNome());
            PrintStream out = new PrintStream(arq);
            c.getArquivo().writeTo(out);

            c.setRevisao( revisao);

        }
        catch (SVNException ex) {
            ex.printStackTrace();
            System.out.println(ex.getMessage());
        }
        catch (IOException ioe) {
            ioe.printStackTrace();
        }
```

```
        return c;
    }
}
```