Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós Graduação em Ciência da Computação

Extensão da Ferramenta MVCASE com serviços Remotos de Armazenamento e Busca de Artefatos de Software

Daniel Lucrédio

São Carlos
Abril de 2005

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós Graduação em Ciência da Computação

Extensão da Ferramenta MVCASE com serviços Remotos de Armazenamento e Busca de Artefatos de Software

Dissertação apresentada ao Programa de pós-graduação em Ciência da Computação do Departamento de Computação da Universidade Federal de São Carlos como parte dos requisitos para obtenção do título de Mestre na área de Engenharia de Software.

Orientador: Dr. Antonio Francisco do Prado

São Carlos
Abril de 2005

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

L942ef

Lucrédio, Daniel.
Extensão da ferramenta MVCASE com serviços remotos de armazenamento e busca de artefatos de software / Daniel Lucrédio. -- São Carlos : UFSCar, 2005.
103 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2005.

1. Engenharia de software. 2. Engenharia de software auxiliada por computador. 3. UML. 4. Reuso. I. Título.

CDD: 005.1 (20ª)

a DEUS,

à Alessandra, minha querida esposa
e companheira neste mundo

e a meus pais e irmãos.

Agradecimentos

Algumas pessoas me ajudaram muito no desenvolvimento desta dissertação, direta ou indiretamente, e não posso deixar de agradecê-los.

Agradeço aos meus professores de graduação e pós-graduação da UFSCar, por compartilhar seu conhecimento, que foi valioso no desenvolvimento deste trabalho.

Ao meu orientador, Antonio Francisco do Prado, pela amizade, pela orientação, pelo incentivo, pela cobrança, e principalmente por me proporcionar a oportunidade de desenvolver um trabalho de mestrado.

Obrigado aos membros da banca examinadora, Dra. Claudia Maria Lima Werner e Dra. Renata Pontin de Mattos Fortes, pelas sugestões e discussões durante a defesa, que enriqueceram muito este trabalho.

Obrigado ao professor Silvio Romero de Lemos Meira, da UFPE, por ter aberto o espaço do C.E.S.A.R. para que eu pudesse divulgar meu trabalho no maior centro de desenvolvimento de software do nordeste, proporcionando oportunidade para trabalhos em conjunto.

Ao Departamento de Computação da UFSCar, à CAPES, à FAPESP e ao CNPq, pelo auxílio financeiro que tornou possível o desenvolvimento deste trabalho.

Gostaria também de agradecer a algumas pessoas que trabalharam comigo na publicação de artigos: Eduardo Kessler Piveta, Alan Gavioli, Mauro Biajiz e Claudio Haruo Yamamoto, obrigado pelas importantes contribuições vindas de outras áreas de pesquisa.

Obrigado também a todos os meus amigos e colegas do mestrado. Considero-os minha maior conquista nesse período de mestrado, ainda que não dê para escrever um artigo sobre isso.

Agradeço aos meus colegas do laboratório. Obrigado Eduardo, que praticamente co-orientou este trabalho, sempre lendo novos artigos e colocando novas idéias, pegando no pé para não deixar a gente desanimar. Obrigado Vinicius, pela amizade, pelas discussões produtivas (e não produtivas) e por trazer sua alegria da Bahia até o interior de São Paulo. Alexandre “Clô” Alvaro, pelos trabalhos em conjunto, pela amizade, por tudo o mais, mas principalmente, pelo “Soldat”! Darley “Biriguey”, pela amizade e pelo talento musical com a voz e a viola. Obrigado Val, sempre reclamando da vida mas nos alegrando com sua presença. Valeu amigo João “pelão”, que ajudou demais

na parte do controle de versões. Valeu Raphael, pela amizade e ajuda em diversos momentos. E obrigado Evandro, pela ajuda no estudo de caso.

Obrigado também a todo o pessoal do DC da UFSCar, em especial Bel, Renatinho, RAR, Genta, Jukinha, Dinho, Escovar, Mo-moacir, Bocudo, Jéssica, Fabiana, Matheus, Mairum, e outros, pelas boas risadas, pelo tereré na hora do almoço e por tudo o mais. Também agradeço pela honra de ter participado do Saravá Saci Soccer (SSS), primeiro e único time da pós-graduação campeão do torneio InterAnos.

E obrigado, Alessandra, pelo amor que sempre me dá conforto em todos os momentos.

Obrigado a meus pais, José Horácio e Dalila, meus sogros, Vivaldo e Ivani, e meus irmãos e cunhados, Rafael, Amanda, Maurício, Andréa, Fábio e Andréa. Sua presença ao meu redor é muito importante para mim.

E obrigado DEUS, sempre, pela minha vida.

“Sonhar mais um sonho impossível...
Lutar quando é fácil ceder...
Vencer o inimigo invencível...
Negar, quando a regra é vender.

Sofrer a tortura implacável,
Romper a incabível prisão,
Voar no limite improvável,
Tocar o inacessível chão.

É minha lei, é minha questão
Virar esse mundo, cravar esse chão,
Não me importa saber, se é terrível demais
Quantas guerras terei que vencer por um pouco de paz.

E amanhã, se esse chão que eu beijei for meu leito e perdão.
Vou saber que valeu delirar e morrer de paixão
E assim, seja lá como for, vai ter fim, a infinita aflição
E o mundo vai ver uma flor...
Brotar do impossível chão.”

Chico Buarque / Rui Guerra

Resumo

A reutilização de software busca promover aumento na produtividade e redução de prazo de entrega de produtos de software, através da otimização do trabalho, evitando-se a duplicação desnecessária de esforço. Nesse contexto, duas áreas se destacam: componentes de software e engenharia de software assistida por computador (CASE). A abordagem de desenvolvimento baseado em componentes busca oferecer maneiras para se construir software mais “reutilizável”. O uso de CASE busca reduzir o esforço, delegando parte do trabalho para ser realizado pelo computador, e aumentar a qualidade, auxiliando o Engenheiro de Software nas tarefas de desenvolvimento. Combinando as idéias de componentes de software e CASE, esta dissertação apresenta uma extensão da ferramenta MVCASE, uma ferramenta acadêmica desenvolvida na UFSCar - Universidade Federal de São Carlos, com serviços remotos de armazenamento e busca de artefatos de software, buscando melhorar a produtividade e reduzir tempo e custos no desenvolvimento de software.

Abstract

Software reuse aims to increase productivity and to decrease time-to-market, through work optimization and avoiding effort duplication. In this context, two areas arise: Software Components and Computer-Aided Software Engineering (CASE). The Component-Based Development approach aims to offer ways to build more “reusable” software. The use of CASE seeks to reduce effort, by delegating part of the tasks to be performed by the computer, and to improve quality, by helping the Software Engineer to perform some tasks. Combining the ideas of software components and CASE, this dissertation presents an extension of MVCASE tool, an academic tool developed in UFSCar - Federal University of São Carlos, with remote services for storage and search for software artifacts, aiming to improve productivity and to reduce time and costs in software development.

Lista de Figuras

Sumário

1 Introdução

1.1 Visão geral

Atualmente, a disseminação da Tecnologia de Informação (TI), junto com os avanços na área da computação, tem obrigado muitas indústrias, de praticamente todas as áreas, a adotarem sistemas computadorizados para se manterem competitivas e sobreviverem no mercado. Essa tendência tem resultado em sistemas grandes e complexos, dos quais essas empresas passaram a depender cada vez mais. Isso acabou gerando uma demanda por qualidade de software, estimulando a criação de modelos e normas de qualidade de software, como CMM (PAULK et al., 1993) e ISO (ISO, 1987), e estimulando as empresas a investirem no seu processo de desenvolvimento.

Conforme (WEBER; NASCIMENTO, 2002), no contexto brasileiro essa demanda pode ser vista claramente. Neste artigo, os autores apresentam os resultados de várias pesquisas realizadas com fábricas de software de todo o território nacional. Os resultados mostram que o aumento da preocupação com qualidade, principalmente visando às normas e padrões internacionais, é uma realidade para as empresas brasileiras. Por exemplo, em 1995, apenas 2% das empresas possuíam certificado ISO/9000. Já em 2001, esse número cresceu para 21%. Outro exemplo: em 1995, 3% das empresas já haviam tido experiência com CMM. Em 2001, esse número cresceu para 23%.

Além da qualidade, o alto nível de competição também exige que as empresas atinjam maior produtividade e menor tempo de desenvolvimento, para que possam vender seus produtos e obter a lucratividade necessária para se manter no mercado. É com base nessas idéias que surge o conceito de reutilização de software, que busca atingir esses objetivos através da otimização do esforço. Trata-se de uma idéia simples, que pode ser resumida como: “tudo o que já foi feito, não deve ser feito novamente”. Porém, aplicar essa idéia na prática não é tão simples. São necessários métodos, técnicas e ferramentas que consigam efetivamente promover a reutilização de software.

Existem diversas formas de reutilização. A orientação a objetos é uma tentativa de facilitar a reutilização. Princípios como encapsulamento e herança podem ajudar a promover a reutilização de código. Outra forma de reutilização são os padrões, que buscam encapsular o conhecimento que

foi obtido em soluções bem sucedidas, de modo que o mesmo possa ser reutilizado posteriormente. Uma abordagem que está sendo bastante pesquisada atualmente é o Desenvolvimento Baseado em Componentes (DBC).

É interessante notar que um assunto tão amplamente discutido atualmente tenha suas raízes décadas atrás. Em 1968, durante uma conferência realizada pelo comitê de ciência da OTAN (Organização do Tratado do Atlântico Norte) sobre Engenharia de Software, McIlroy apresentou sua tese de doutorado, intitulada "Componentes de Software Produzidos em Massa" (MCILROY, 1968).

Neste trabalho pioneiro, McIlroy apresenta sua idéia de que a indústria de software necessitava de um mercado de rotinas (componentes), classificadas de modo que ao se construir um software as mesmas pudessem ser encontradas, de acordo com o problema específico, e reutilizadas.

Então, o que vem a ser Desenvolvimento Baseado em Componentes? Numa tradução livre das palavras de McIlroy: "*Quando vamos escrever um compilador, começamos dizendo: 'Que mecanismo de tabela devemos construir?', ao invés de 'Que mecanismo devemos utilizar?'. Eu digo que já fizemos muito isso, e devemos começar a tirar esse tipo de coisa da prateleira.*"

Com isso McIlroy criticava uma prática comum na época, quando se pensava no software como um único bloco monolítico. A idéia principal do Desenvolvimento Baseado em Componentes é, ao invés de construir todo o software como um único bloco, reutilizar partes já prontas, tirando-as da "prateleira" e compondo-as de modo a atender os requisitos do software. Essa abordagem pode trazer vários benefícios, como:

- **Aumento da produtividade**: Através da reutilização, poupa-se trabalho e tempo, já que não é necessário construir o software a partir do zero;
- **Menor número de erros**: Caso se utilizem apenas componentes certificados, com características definidas de desempenho e execução, o software irá apresentar menos erros;
- **Maior manutenibilidade**: A modularização e baixo acoplamento alcançado através da utilização de componentes facilita a manutenção do software; e
- **Maior flexibilidade**: Sendo fracamente acoplados entre si, componentes de software podem ser modificados e estendidos, de modo a agregar novas funcionalidades sem a necessidade de modificação de outras partes do software.

Neste ponto cabe uma questão. Se a idéia de DBC é relativamente antiga, e pode trazer vários benefícios, porque é discutida até hoje, e ainda não é efetivamente utilizada na prática? Em

(ALMEIDA; ALVARO; MEIRA, 2004), os autores apresentam um interessante compêndio sobre as experiências com reutilização em diversas empresas de software. Em um deles (BASS et al., 2000), realizado na *Carnegie Mellon University / Software Engineering Institute* (CMU/SEI), os autores concluem que os principais inibidores para a prática do DBC são, em ordem decrescente de importância:

- **Falta de componentes disponíveis**: as empresas sentiram falta de um conjunto de componentes disponíveis para poder compor seu produto a partir deles. Isso implica em duas possibilidades: ou esses componentes ainda não existem, ou os consumidores não estão sendo capazes de encontrá-los;
- **Falta de padrões estáveis para tecnologia de componentes**: a maior parte das empresas pesquisadas considera as principais tecnologias de componentes atuais (*Enterprise JavaBeans* (DEMICHIEL, 2002), CORBA (OMG, 2002a), COM/DCOM/COM+ (MICROSOFT, 2005)) como sendo muito instáveis para dar suporte a seus produtos. Com medo de que uma mudança muito grande nessas tecnologias possa implicar em gastos excessivos para manter os produtos atualizados, as empresas preferem aguardar pela maturidade maior nessas tecnologias;
- **Falta de componentes certificados**: as empresas pesquisadas consideram como um dos inibidores a falta de componentes certificados, para garantir que o produto composto por estes componentes possua atributos de qualidade; e
- **Falta de um método consistente de engenharia para produzir, de maneira consistente, sistemas com qualidade utilizando componentes**: da mesma maneira que a falta de estabilidade das atuais tecnologias de componentes, as empresas consideram que nenhum dos atuais métodos para DBC é suficientemente estável e consistente para ser adotado no seu processo produtivo. Esse problema se relaciona com o item anterior, já que para se produzir componentes com qualidade, buscando a certificação, é necessário um método consistente de engenharia.

Esses quatro itens são alvo das atuais pesquisas na área de Desenvolvimento Baseado em Componentes, que atualmente busca soluções que ataquem diretamente estes inibidores. A relação entre esses quatro inibidores e este trabalho de pesquisa será discutida mais adiante neste capítulo.

Dentro da Engenharia de Software, existe um ramo que pesquisa o uso de soluções automatizadas para o DBC. A CASE (*Computer-Aided Software Engineering*, ou Engenharia de Software Assistida por Computador) é um termo utilizado para definir um conjunto de ferramentas automatizadas para dar suporte a um processo de Engenharia de Software.

Segundo (FUGGETTA, 1993), o processo de software é composto por um processo de produção de software, que contém todas as atividades, regras, metodologias, estruturas organizacionais e ferramentas utilizadas para conceber, projetar, desenvolver, entregar e manter um produto de software. Esse processo de produção é definido através de um metaprocesso, responsável pela representação, melhoramento e inovação dos procedimentos e regras utilizados na produção.

O suporte para esse processo de software é formado pelo suporte ao processo de produção e o suporte ao metaprocesso. Esse suporte é constituído por regras e atividades específicas de suporte, e uma infra-estrutura. Essa infra-estrutura consiste da tecnologia (ferramentas) do processo de produção e do metaprocesso, que utilizam serviços de uma tecnologia de apoio (serviços de rede, serviços do sistema operacional, banco de dados, e outros). A CASE é justamente este conjunto de ferramentas de suporte ao processo de produção e do metaprocesso.

É muito comum a confusão entre o **conceito** de CASE e as **ferramentas** CASE. A Engenharia de Software Assistida por Computador (CASE) consiste na utilização do computador para auxiliar no desenvolvimento de software. As ferramentas CASE são as ferramentas utilizadas pelo Engenheiro de Software no desenvolvimento, dentro da CASE.

Mas quais são os benefícios da utilização da CASE? Em (CRONHOLM, 1995), é apresentado um estudo, realizado em 1994 na Suécia, que buscou identificar os principais motivos que levam os fabricantes de software a investir em ferramentas CASE, e os benefícios trazidos. Apesar de antigo, esse estudo ainda se mostra válido, evidenciando uma preocupação ainda presente nas empresas atuais.

Os resultados desse estudo mostraram que a principal motivação que leva as empresas a investir em CASE é a busca por maior competitividade, dado que elas se encontram inseridas em um mercado dinâmico. Segundo a pesquisa, a CASE pode ajudar em dois pontos:

- **Rapidez no desenvolvimento**: a utilização de CASE torna o desenvolvimento mais simples e fácil de ser seguido, além de tornar o gerenciamento da documentação envolvida no processo mais eficaz; e
- **Qualidade do produto**: além de agilizar o processo, a CASE ajuda a melhorar a qualidade do mesmo, o que contribui para aumentar a qualidade do produto, através do aumento da flexibilidade do trabalho, aumento da consistência e qualidade da documentação.

No GOES (Grupo de Engenharia de Software), do Departamento de Computação da UFSCar (Universidade Federal de São Carlos), vem sendo desenvolvida uma ferramenta de modelagem, denominada MVCASE (*Multiple Views CASE*). Essa ferramenta dá suporte ao desenvolvimento de software orientado a objetos, cobrindo as fases de análise, projeto e implementação.

Recentemente (PRADO; LUCRÉDIO, 2001; ALMEIDA et al., 2002b, 2002a), a ferramenta foi modificada para atender aos requisitos do Desenvolvimento Baseado em Componentes. Atualmente, o Engenheiro de Software pode utilizar a MVCASE para desenvolver sistemas orientados a objetos, utilizando componentes, realizando a modelagem na linguagem UML (OMG, 2003b), o projeto e implementação de componentes segundo as tecnologias EJB (DEMICHIEL, 2002) e CORBA (OMG, 2002a).

1.2 Motivação e Justificativa

Considerando os quatro pontos que inibem a ampla adoção do Desenvolvimento Baseado em Componentes por parte da indústria de desenvolvimento de software (falta de um mercado de componentes; falta de padrões estáveis de tecnologias de componentes; falta de componentes certificados; e falta de métodos de engenharia para DBC), e os benefícios que podem ser alcançados pela CASE, esta última pode auxiliar em dois destes pontos: **falta de um mercado de componentes** e **falta de métodos consistentes de engenharia para DBC**, principalmente porque:

- A criação de um mercado de componentes envolve diversos fatores. Os mecanismos para fazer com que isso seja possível deverão utilizar algum tipo de ferramenta automatizada, visto que a complexidade e a dimensão dos fatores envolvidos faz com que seja inviável um processo manual. Para se ter uma idéia de tal complexidade, em (RAVICHANDRAN; ROTHENBERGER, 2003), são discutidos assuntos relacionados a mercados de componentes;
- Conforme a seção 1.1, um processo de software pode ser apoiado por uma determinada tecnologia CASE. No caso do Desenvolvimento Baseado em Componentes, a CASE pode auxiliar na análise, projeto, implementação, testes, disseminação, documentação e manutenção dos componentes, bem como auxiliar na composição desses componentes em produtos de software completos; e
- Conforme a seção 1.1, tem-se que uma das razões pelas quais as empresas sentem a falta de componentes disponíveis pode ser justamente a dificuldade em encontrá-los. Neste sentido, a CASE pode ser utilizada, oferecendo mecanismos para armazenamento e recuperação de componentes.

E foi justamente este último item que motivou esta pesquisa, junto com o fato de que a ferramenta MVCASE é desenvolvida no GOES, o que ofereceu uma excelente oportunidade para pesquisar a utilização de serviços remotos de armazenamento e busca de artefatos de software.

1.3 Escopo e objetivos

Os problemas apresentados são alvo de inúmeras pesquisas atualmente, e não poderiam ser todos cobertos em um único trabalho de mestrado. Sendo assim, é necessário definir o que foi tratado de forma específica nesta dissertação.

A partir de algumas limitações existentes na MVCASE, descritas nesta seção, foram desenvolvidas novas funcionalidades para a ferramenta, buscando ao mesmo tempo superar estas limitações e contribuir para o avanço nas pesquisas relacionadas à reutilização de software, mais precisamente com relação à dificuldade em armazenar e recuperar artefatos de software, visando a reutilização.

Para compreender os pontos tratados nesta dissertação, considere algumas características comuns aos ambientes de desenvolvimento de software:

- No processo de desenvolvimento de software está envolvida grande quantidade e variedade de informações. Requisitos de software, modelos gráficos de análise e projeto, documentos, anotações, rascunhos, e até o código executável, são alguns exemplos das informações envolvidas. Essas informações constituem parte importante do processo de desenvolvimento de software, influenciando diretamente na qualidade do processo e do produto;

- Para manipular esta variedade de artefatos, são necessárias diferentes ferramentas, cada uma sendo responsável por auxiliar o Engenheiro de Software em uma determinada etapa do processo de desenvolvimento. Neste cenário, a questão da interoperabilidade entre ferramentas se torna importante. Segundo Wasserman, citado em (SOMMERVILLE, 2000), diferentes ferramentas CASE devem possuir diversos níveis de integração. Entre eles, destaca-se a integração de dados, que permite que diferentes ferramentas possam compartilhar dados ao longo do processo de software;

- Outra característica importante dos ambientes de desenvolvimento é que muitas vezes podem existir vários Engenheiros de Software envolvidos em um mesmo projeto, cada um atuando em uma ou mais etapas do processo, e que precisam compartilhar informações. Possivelmente, estes Engenheiros de Software estarão trabalhando em diferentes estações de trabalho, conectadas a uma rede, seja ela uma rede local ou mesmo a Internet; e

- A reutilização de software exige um eficiente mecanismo de armazenamento e busca de informações. Segundo (VILLELA, 2000), o sucesso do Desenvolvimento Baseado em Componentes está intrinsecamente ligado aos mecanismos para disponibilização e recuperação de componentes reutilizáveis.

A ferramenta MVCASE é uma ferramenta de modelagem. Nela, o Engenheiro de Software utiliza a UML para criar modelos dos sistemas em desenvolvimento. Estes modelos são então armazenados em arquivos texto, de modo que possam ser posteriormente recuperados na ferramenta, para exibição e/ou edição. Antes da realização do trabalho descrito nesta dissertação, os modelos eram persistidos neste arquivo texto segundo a linguagem MDL (*Modeling Domain Language*), que é a mesma utilizada pela ferramenta comercial Rational Rose (IBM, 2004). Além disso, não existia nenhum tipo de mecanismo que facilitasse o compartilhamento destes arquivos texto, além daqueles existentes na maioria dos sistemas operacionais.

Estas limitações causavam alguns problemas com relação ao processo de desenvolvimento de software:

- Com a MDL somente era possível representar modelos UML. Outros tipos de artefatos não podiam ser representados. Apesar de a MVCASE trabalhar somente com modelos UML, era desejável que se tivesse um formato mais flexível de representação dos artefatos, visando facilitar futuras expansões da ferramenta ou mesmo da própria UML;

- Sendo uma linguagem proprietária, não existem muitas ferramentas capazes de reconhecer a MDL. Para que outras ferramentas possam reutilizar os artefatos produzidos pela MVCASE, era necessário algum tipo de adaptador. E como existe pouca documentação disponível com relação a esta linguagem, a construção deste adaptador é muito trabalhosa. Isto prejudicava tanto a reutilização dos artefatos como a cooperação da MVCASE com outras ferramentas;

- A ausência de um mecanismo que facilitasse o compartilhamento dos artefatos produzidos pela MVCASE dificultava o trabalho em equipe. Dois Engenheiros de Software não podiam realizar alterações simultaneamente no mesmo arquivo, a menos que cada um obtivesse sua própria cópia para trabalhar. Neste caso, alterações simultâneas precisavam ser posteriormente avaliadas e manualmente combinadas em uma única versão final do arquivo; e

- O armazenamento puro e simples de artefatos em arquivos texto não é apropriado à reutilização. Isto porque bibliotecas de software tendem a crescer e se tornar imensas. À medida que mais artefatos vão sendo produzidos, realizar a busca manual sobre este conjunto de artefatos, de modo a encontrar aquele que possa ser reutilizado, se torna impraticável.

Com base nestas limitações, pesquisou-se a extensão da ferramenta MVCASE com serviços remotos para armazenamento e busca de artefatos de software. Buscou-se principalmente fazer com que os artefatos produzidos pela ferramenta pudessem:

- ser representados em um formato padrão extensível, capaz de representar não somente modelos UML, mas outros tipos de artefato;
- ser interpretados por outras ferramentas;
- ser compartilhados por diferentes membros de uma equipe de desenvolvimento; e
- ser armazenados de tal forma que se pudesse realizar busca e recuperação, visando a reutilização.

Além de solucionar os problemas com a MVCASE, este trabalho de pesquisa também teve como objetivo contribuir com o avanço na pesquisa em reutilização de software. Oferecendo suporte automatizado para armazenamento e busca de artefatos de software, buscou-se desenvolver mecanismos que pudessem ser aplicados em outros cenários, com outros tipos de artefatos e ferramentas, e não somente a MVCASE. O desenvolvimento e a descoberta das tecnologias necessárias para o funcionamento destes mecanismos também foram objetivo deste trabalho.

Para melhor divulgação dos resultados desta pesquisa e possibilitar a continuidade de trabalhos futuros, a MVCASE foi disponibilizada sob licença de software livre, proporcionando benefícios para a comunidade de desenvolvimento de software, ajudando na disseminação da tecnologia produzida na universidade, e captando também contribuições de outros desenvolvedores.

1.4 Estrutura da dissertação

Este primeiro capítulo apresentou o contexto, a motivação, o escopo e os objetivos desta dissertação de mestrado.

No capítulo 2 faz-se uma introdução aos principais conceitos relacionados a esta dissertação. É feita uma discussão sobre ferramentas CASE, foco principal deste trabalho. Para tratar dos problemas de interoperabilidade entre ferramentas em ambientes heterogêneos e distribuídos, são apresentados o padrão XMI para representação de artefatos, e o controle de versões, primordial para o trabalho distribuído. Finalmente, é discutida a questão da busca, para possibilitar que os artefatos armazenados sejam mais facilmente reutilizados.

No capítulo 3 apresenta-se brevemente a ferramenta MVCASE, suas principais características e funcionalidades.

No capítulo 4 é apresentada em detalhes a extensão da MVCASE e os serviços remotos de armazenamento e busca de artefatos de software.

No capítulo 5 apresenta-se um estudo de caso, juntamente com as análises realizadas para avaliar os resultados da extensão.

Finalmente, no capítulo 6 são apresentadas as considerações finais e os trabalhos futuros.

2 *Revisão Bibliográfica*

Neste capítulo são apresentados os principais conceitos envolvidos com esta pesquisa. São discutidos assuntos relacionados à CASE, o padrão XMI, para intercâmbio de dados, e o controle de versões. Finalmente, são apresentados os principais trabalhos relacionados à busca e recuperação de artefatos reutilizáveis.

2.1 Ferramentas CASE

Ao se observar a história da humanidade, nota-se que a evolução humana está diretamente relacionada ao uso de ferramentas para aumentar as habilidades naturais dos seres humanos. Desde as ferramentas mais primitivas, destinadas à caça e à pesca, até os modernos satélites de comunicação, o homem sempre explorou o produto de sua inteligência na construção de artefatos para sobreviver e evoluir através das eras.

Na área da computação, mais especificamente na Engenharia de Software, não podia ser diferente. O desenvolvimento de software se apóia em uma grande diversidade de atividades que devem ser executadas para se produzir software com qualidade. Estas atividades exigem diferentes habilidades. Por exemplo, na análise de requisitos, o Engenheiro de Software deve agir como um interrogador, consultor, solucionador de problemas e um negociador (PRESSMAN, 2001). Já a implementação exige um raciocínio lógico, voltado a construir linhas de código para instruir o computador a executar exatamente o que é esperado.

Devido a esta diversidade inerente à Engenharia de Software, ferramentas sempre foram utilizadas para auxiliar na execução das tarefas relacionadas ao desenvolvimento de software. Essas ferramentas, denominadas CASE (*Computer-Aided Software Engineering* ou Engenharia de Software Assistida por Computador), existem desde os primórdios da computação, com os primeiros cartões perfurados, editores, compiladores e depuradores, até as modernas ferramentas para coleta de requisitos, projeto, construção de GUIs (*Graphical User Interface* ou Interface Gráfica com o Usuário), modelagem e gerenciamento de banco de dados, gerenciamento de configuração de software, entre outras, formando ambientes completos que cobrem todo ou a maior parte do ciclo

de vida de desenvolvimento de software (HARRISON; OSSHER; TARR, 2000).

Existe um grande número de ferramentas CASE no mercado, atuando nas diferentes atividades do processo de desenvolvimento de software. Muitas empresas as utilizam para reduzir o esforço despendido na construção de software e para aumentar a qualidade do seu processo. Na comunidade científica, ferramentas CASE são propostas para ilustrar, testar e validar novas idéias na área da Engenharia de Software, buscando aprimorar o conhecimento humano nessa área.

A CASE pode atuar desde a identificação e modelagem dos requisitos até as fases de implementação e testes. Atividades que são executadas ao longo de todo o processo (atividades guarda-chuva), como por exemplo o gerenciamento de configuração e gestão de projetos de software (PRESSMAN, 2001), também podem obter auxílio dessas ferramentas.

Os diferentes tipos de CASE existentes apresentam características muito particulares. Por exemplo, ferramentas para testes e depuração trabalham em um nível de abstração baixo, manipulando instruções e comandos em uma linguagem executável. Ferramentas para auxílio ao planejamento efetuam cálculos de estimativas de tempo, custo, esforço e outras métricas.

Ferramentas que oferecem suporte às atividades iniciais do ciclo de vida do projeto de software (como análise de requisitos) são algumas vezes chamadas de ferramentas CASE *front-end*, ou *upper* CASE. Já as ferramentas que são utilizadas em fases posteriores do ciclo de vida (como compiladores e ferramentas de teste) são chamadas de *back-end*, ou *lower* CASE.

Um aspecto importante que deve ser considerado é a interoperabilidade entre diferentes ferramentas CASE. Segundo (PRESSMAN, 2001), o grande potencial da Engenharia de Software Assistida por Computador só é alcançado integrando-se diversas ferramentas individuais. A idéia é que uma ferramenta complemente a funcionalidade da outra, contribuindo com todo o processo. Essa abordagem oferece várias vantagens em relação ao uso de ferramentas individuais, incluindo: a transferência de informação entre as ferramentas; redução do esforço necessário para realizar atividades guarda-chuva; e aumento do controle de projeto, entre outros (PRESSMAN, 2001).

Porém, essa integração é difícil de ser obtida, pois envolve diversos fatores, como por exemplo o uso de repositórios compartilhados, padronização de entradas e saídas, integração baseada em eventos, entre outros (HARRISON; OSSHER; TARR, 2000). Assim, é necessária uma arquitetura completa de integração, para que todas as ferramentas possam oferecer o máximo de seu potencial e contribuir para o melhoramento do processo de desenvolvimento como um todo.

Um tipo particular de ferramenta CASE, amplamente utilizado na indústria, são as ferramentas de modelagem. Essas ferramentas auxiliam na criação de modelos de diversos aspectos de um sistema de software, como o seu comportamento, sua arquitetura ou a estrutura do banco de dados.

Além de ser simplesmente um editor gráfico, para facilitar a criação desses modelos, essas ferramentas normalmente oferecem mecanismos que auxiliam em outras tarefas do desenvolvimento.

Para construir modelos de um sistema, o Engenheiro de Software usa uma linguagem de modelagem. Em 1998, foi proposta a UML (*Unified Modeling Language*), posteriormente aceita como um padrão (OMG, 2003b). Atualmente, a UML é amplamente utilizada, sendo por isso a linguagem mais suportada pelas ferramentas de modelagem atuais.

Com a utilização de ferramentas CASE, em especial as ferramentas de modelagem, as informações envolvidas no processo de desenvolvimento de software podem ser criadas, modificadas e reutilizadas mais facilmente do que em um processo manual. Documentos de requisitos, modelos gráficos e textuais, entre outras informações, são manipulados por essas ferramentas, proporcionando mais flexibilidade e rapidez no processo de desenvolvimento de software. As demais características dessas ferramentas, como capacidade de geração de código e consistência entre modelos, ajudam a aumentar a eficiência e qualidade do processo.

Dentre as ferramentas comerciais disponíveis, destacam-se:

- **Borland Together**[1]: A ferramenta *Together* surgiu como uma ferramenta livre. Hoje, comercializada pela *Borland*, ela evoluiu como uma das principais ferramentas de modelagem do mercado. Além da modelagem, a *Borland Together* dá suporte a outras tarefas, como geração de código, trabalho cooperativo, entre outras;

- **IBM Rational Rose**[2]: Originalmente concebida pelos criadores da UML, a *Rational Rose* foi a pioneira a utilizar esta linguagem de modelagem. Com recursos não só de modelagem, mas gestão de requisitos, trabalho cooperativo, engenharia reversa, entre outros, a *Rose* é hoje comercializada pela IBM;

- **GentleWare Poseidon UML**[3]: A exemplo da *Borland Together*, a *Poseidon* também surgiu a partir de um software livre. A *Poseidon* evoluiu a partir da ferramenta *ArgoUML* (apresentada mais adiante), com recursos mais poderosos de modelagem e geração de código;

- **Microsoft Visio**[4]: A *Microsoft Visio* permite a modelagem em diversas notações, incluindo a UML, porém não possui nenhum suporte para outras tarefas, sendo essencialmente um editor gráfico;

[1] http://www.borland.com/together/
[2] http://www-306.ibm.com/software/awdtools/developer/plus/
[3] http://www.gentleware.com/index.php
[4] http://office.microsoft.com/en-us/FX010857981033.aspx

- **ERWin**[5] e **ER/Studio**[6]: Estas duas ferramentas também são amplamente utilizadas, porém mais voltadas para a modelagem de dados. Possuem recursos para modelagem e criação de estruturas de bancos de dados para diferentes Sistemas Gerenciadores de Banco de Dados; e

- **Omondo Eclipse UML - versão comercial**[7]: Integrada à plataforma *Eclipse* (ECLIPSE, 2004), a ferramenta *Eclipse UML* permite a edição de diagramas UML, geração de código e engenharia reversa. Também é oferecido suporte ao trabalho cooperativo, com controle de versões e gerenciamento de configuração de software, através da integração com outras ferramentas.

Algumas dessas ferramentas podem ser oferecidas gratuitamente para instituições de ensino, podendo ser utilizadas com propósito didático. Porém, por motivos comerciais, não é disponibilizado o seu código-fonte. Isso impede que trabalhos de pesquisa utilizem esse código, modificando-o para inserir novos avanços tecnológicos. Para isso, normalmente são utilizadas ferramentas gratuitas, das quais se destacam:

- **Violet**[8]: Totalmente escrita em Java, permite a edição de alguns diagramas UML, como diagrama de classes, e de seqüência. Não possui suporte para geração de código ou outras tarefas do desenvolvimento;

- **UMLSculptor**[9]: Uma ferramenta simples para edição de diagramas de classes da UML;

- **EclipseUML - versão gratuita**[10]: Essa ferramenta é um *plug-in* para a plataforma Eclipse (ECLIPSE, 2004), permitindo a edição de diversos diagramas da UML e geração de código. O suporte para trabalho em equipe e acesso a repositórios é oferecido em uma versão comercial da ferramenta; e

- **ArgoUML**[11]: A ArgoUML é totalmente escrita em Java e permite a edição de todos os diagramas da UML. Também possui mecanismos para geração de código e engenharia reversa, sendo atualmente a ferramenta gratuita mais utilizada.

As ferramentas *Violet* e *UMLSculptor* permitem fazer a modelagem segundo a UML, e apenas alguns diagramas estão disponíveis. A versão gratuita da ferramenta *EclipseUML* é um *plug-in*

[5] http://www3.ca.com/Solutions/Product.asp?ID=260
[6] http://www.embarcadero.com/products/erstudio/
[7] http://www.omondo.com/
[8] http://www.horstmann.com/violet/
[9] http://umlsculptor.sourceforge.net/
[10] http://www.omondo.com/index.jsp
[11] http://argouml.tigris.org

para a plataforma *Eclipse* (ECLIPSE, 2004), que permite a edição de vários diagramas da UML e geração de código. Uma versão não gratuita da ferramenta dá suporte ao trabalho cooperativo, com integração a mecanismos de controles de versões e rastreamento de requisitos. Apesar desse suporte a ferramenta não é disponibilizada em regime de software livre. Já a ferramenta *ArgoUML* é um software livre, que permite a modelagem em UML e a geração de código em Java, sendo atualmente a mais conhecida das ferramentas gratuitas.

2.1.1 CASE e reutilização

É relativamente simples encontrar exemplos de ferramentas que facilitem a reutilização. Os primeiros ambientes integrados de desenvolvimento (IDEs), como Delphi, C++ Builder e JBuilder, dão suporte à reutilização de componentes.

Nesses ambientes, o Engenheiro de Software normalmente dispõe, em uma barra de ferramentas ou um menu, de uma série de componentes pré-fabricados, que podem ser integrados ao software sendo construído em uma ação simples com o *mouse* ou teclado. É também possível construir componentes personalizados e instalá-los na ferramenta, tornando-os disponíveis para reutilização.

Nestes ambientes, a reutilização envolve basicamente o código-fonte, e normalmente os componentes reutilizados são relacionados à interface gráfica com o usuário. Porém, para que os desafios e inibidores da reutilização (vide capítulo 1) sejam vencidos, há uma série de aspectos que esses ambientes não consideram.

Por exemplo, a reutilização em larga escala envolve artefatos produzidos em ambientes distribuídos e heterogêneos. A instalação manual de componentes em ferramentas não é prática neste cenário. Além disso, com componentes sendo produzidos em diferentes lugares, a sua manutenção e evolução envolve constantes mudanças, que precisam ser gerenciadas.

Outro aspecto é que a reutilização apenas de código-fonte despreza a maior parte dos benefícios associados a essa abordagem (KRUEGER, 1992; GRISS, 1995; FRAKES; ISODA, 1994; JACOBSON; GRISS; JONSSON, 1997).

As atuais tentativas de se alcançar a reutilização através da CASE são principalmente acadêmicas. Lüer & Rosenblum (LÜER; ROSENBLUM, 2001) descrevem um ambiente para Desenvolvimento Baseado em Componentes, chamado WREN[12]. Porém, o ambiente não dá suporte para a construção de componentes. Sua ênfase é maior na construção das aplicações, em um processo

[12]Este ambiente voltado à reutilização faz referência a Sir Christopher Wren (1632-1723), lembrado por fazer uso da reutilização no projeto de 51 igrejas reconstruídas em Londres por ocasião do grande incêndio de 1666. Cada projeto era único, mas uma variante reconhecível de um novo e elegante estilo arquitetural.

de 5 etapas: busca, seleção, configuração, projeto da funcionalidade faltante e implementação da funcionalidade faltante.

Em (LUCRÉDIO et al., 2004), apresentamos um ambiente para construção e reutilização de componentes distribuídos. O ambiente é composto por ferramentas que auxiliam nas tarefas de modelagem, codificação, implantação e execução dos componentes. O ambiente fornece recursos para automaticamente recuperar a topologia da rede, facilitando a implantação dos componentes em ambientes distribuídos de maneira a maximizar o desempenho e melhorar a distribuição da carga de processamento.

Em (BRAGA; WERNER; MATTOSO, 1999), os autores apresentam Odyssey, um ambiente de desenvolvimento de software orientado à reutilização, que oferece suporte ao Desenvolvimento Baseado em Componentes em certos domínios. O Odyssey tem como objetivo dar suporte a todo o ciclo de desenvolvimento de software, desde os modelos conceituais até a implementação dos componentes. Os modelos conceituais incluem modelos de casos de usos, modelos de características, sistemas de padrões de domínio e de análise e modelos orientados a objetos. Os modelos de implementação são os conjuntos de componentes reutilizáveis. A criação e manipulação destes modelos segue um processo de engenharia de domínio, chamado Odyssey-DE. Após criados, todos os modelos são armazenados de forma distribuída e heterogênea. Para que a manipulação destes dados possa ser realizada de maneira uniforme, existe uma camada de mediação, responsável por prover a integração dos dados de modo que o usuário tenha acesso a eles de forma transparente.

Outra utilização da CASE para facilitar a reutilização são os sistemas de repositório, que consistem em ferramentas para armazenar e recuperar artefatos reutilizáveis. A literatura é muito rica nessa área, como pode ser constatado na seção 2.4, onde são apresentados os principais trabalhos relacionados ao armazenamento e busca de componentes.

Uma abordagem que vem ganhando força, principalmente por aliar os benefícios da reutilização aos da CASE, é a MDA (*Model-Driven Architecture*) (KLEPPE; WARMER; BAST, 2003). Na MDA, os modelos não são "apenas papel" que auxiliam nas tarefas de desenvolvimento. Ao invés disso, eles são parte constituinte do software, assim como o código. Sendo assim, a reutilização pode acontecer em níveis mais altos de abstração, aumentando também os níveis de reutilização.

Na MDA, a CASE desempenha um papel fundamental, pois é responsável pelos mecanismos de transformação que possibilitam que os modelos sejam integrados ao software. É uma idéia nova, e por isso ainda não existem ferramentas que dêem suporte completo a essa abordagem (KLEPPE; WARMER; BAST, 2003). Porém, as ferramentas existentes[13] dão uma idéia de como a CASE pode contribuir com a reutilização no futuro.

[13] http://www.omg.org/mda/

2.2 O Padrão XMI

Conforme explícito no nome deste padrão, o XMI (XML Metadata Interchange) consiste na utilização da XML (*eXtensible Markup Language*) para possibilitar o intercâmbio de metadados. De acordo com a especificação do XMI (OMG, 2002d), este padrão surgiu como um meio de se proporcionar o intercâmbio de metadados entre 1) ferramentas de modelagem baseadas em UML (OMG, 2003b); e 2) entre repositórios de metadados baseados no MOF (Meta-Object Facility) (OMG, 2002b).

No primeiro caso, o XMI é utilizado para representar modelos UML, como modelos de casos de uso, modelos de classes, entre outros. No segundo caso, o XMI é utilizado para representar quaisquer tipos de dados e metadados que sejam compatíveis com o MOF.

O padrão MOF, também definido pelo OMG, oferece um *framework* orientado a objetos para a definição de modelos de metadados. O MOF segue a arquitetura clássica de meta-modelagem, conforme mostra a Figura 1.

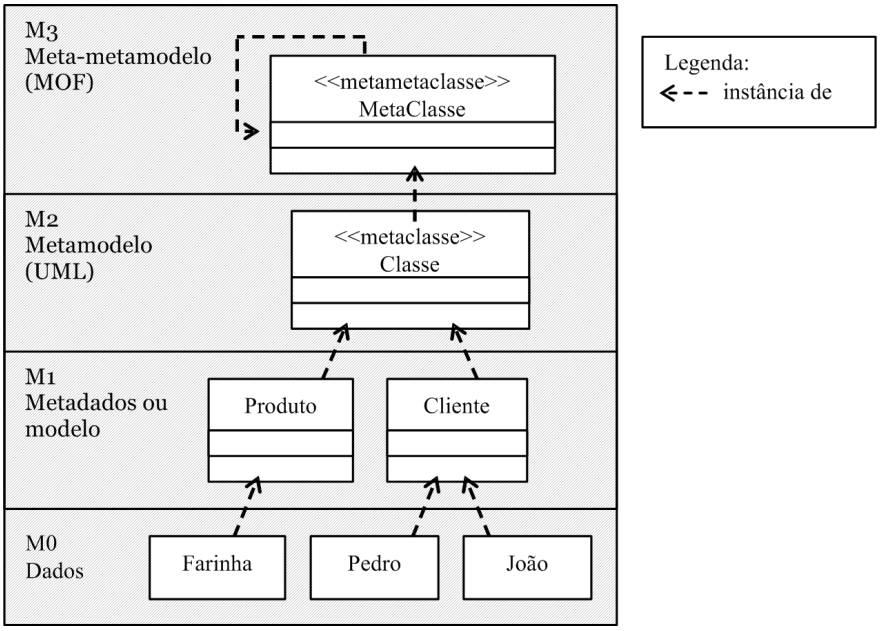

Figura 1: Arquitetura de metadados do MOF.

O primeiro nível (M0) corresponde aos dados propriamente ditos. O segundo nível (M1) corresponde aos metadados, ou modelo. São os dados que descrevem os dados. O terceiro nível (M2) é o metamodelo para definição de modelos. A especificação UML é um exemplo de metamodelo. O padrão MOF encontra-se no quarto nível (M3). Nesse nível estão os modelos que definem metamodelos, ou seja, MOF é uma linguagem para definição de linguagens de modelagem, como a UML, por exemplo. Como o MOF é um metamodelo, ele próprio é instância de si mesmo, e por isso não existe um quinto nível. O XMI é utilizado para representar textualmente elementos de

quaisquer destes níveis, em XML.

Nesta arquitetura, tanto dados como metadados podem ser representados em XMI, desde que estejam em conformidade com o MOF. Várias vantagens derivam do uso da linguagem XML como meio de representação de dados (OMG, 2002d): é uma linguagem aberta e um padrão já consolidado; dá suporte a um conjunto padrão internacional de caracteres; e existem várias APIs disponíveis, dando suporte à criação, visualização e integração de informação XML.

Além destas vantagens da XML, o padrão XMI oferece outras (OMG, 2002d):

- **Interoperabilidade entre ferramentas**: Num ambiente de desenvolvimento, existem diversas ferramentas para auxiliar os Engenheiros de Software. Na maioria dos casos, essas ferramentas trabalham com soluções próprias, e dificilmente podem ser integradas. Se cada ferramenta armazenar e recuperar artefatos no formato XMI, elas podem trocar informações sem a necessidade de mecanismos especiais de exportação e importação. Além disso, o escopo da informação descrita em XMI não é fixo, podendo se estender conforme necessário. Portanto, cada ferramenta pode adicionar suas próprias informações, sem prejudicar as já existentes;

- **Trabalho em ambientes interconectados**: Na maior parte dos ambientes de desenvolvimento, as estações de trabalho estão interconectadas, seja através de uma rede local ou a Internet. Freqüentemente há a necessidade de troca de modelos e informações entre as estações. Utilizando XMI, esses modelos são representados como seqüências de caracteres, que podem ser facilmente enviada pela rede, através de mecanismos simples de envio e recebimento de texto, como por exemplo, o correio eletrônico. Isso oferece uma grande vantagem, pois normalmente as redes possuem mecanismos de segurança, que bloqueiam certos tipos de mensagens; e

- **Reutilização**: Quando trabalhando em diferentes projetos para clientes distintos, o Engenheiro de Software pode se deparar com ferramentas diferentes em cada cliente. Através da interoperabilidade entre ferramentas obtida com o padrão XMI, é possível, por exemplo, reutilizar os modelos de um projeto A, em um outro projeto B, sem a necessidade de nenhuma conversão.

Apesar de teoricamente o padrão XMI apresentar inúmeras vantagens, algumas delas só serão alcançadas na prática com a sua adoção em larga escala por parte dos fabricantes de ferramentas de modelagem e de repositórios. Porém, existe atualmente um grande número de ferramentas que utiliza esse padrão, como por exemplo, ArgoUML (ARGOUML, 2005), EclipseUML (OMONDO,

2005) e Borland Together (BORLAND, 2005), e a tendência é que esse padrão se estabeleça, acompanhando a necessidade de meios mais flexíveis para o armazenamento de informações.

2.3 Controle de versões

Durante o ciclo de desenvolvimento, os artefatos produzidos evoluem até que atinjam um estado em que satisfaçam os requisitos exigidos pelo usuário, tornando-se o que é conhecido como item de configuração (CI ou *Configuration Item*). Portanto, define-se o CI como sendo cada um dos elementos de informação que são criados durante o desenvolvimento de um produto de software, ou que para este desenvolvimento sejam necessários, que são identificados de maneira única e cuja evolução é passível de rastreamento (PRESSMAN, 2001).

O controle da evolução dos CIs exige, entre outras coisas, o controle das diferentes versões que são geradas com as alterações realizadas sobre estes artefatos. Estas versões devem ser devidamente armazenadas e identificadas. Este processo é chamado de controle de versões, e o seu principal objetivo é oferecer meios para que diferentes membros de uma equipe possam trabalhar em um mesmo projeto, evitando ao máximo a ocorrência de conflitos decorrentes de alterações sendo efetuadas simultaneamente sobre os itens de configuração.

Porém, armazenar e gerenciar diferentes versões de todos os artefatos de um projeto envolve uma grande quantidade de informação. Para se ter uma idéia, em (CONRADI; WESTFECHTEL, 1998), os autores apresentam os principais conceitos envolvidos no controle de versões. A quantidade e complexidade das informações faz com que seja inviável um controle manual das mesmas. Por isso, normalmente elas são controladas por ferramentas automatizadas. Existem diversas ferramentas de controle de versões disponíveis. Dentre as ferramentas comerciais, destacam-se:

- **ClearCase**[14]: A *ClearCase* era originalmente comercializada pela *Rational*, fabricante da ferramenta de modelagem *Rose*. Hoje, a *ClearCase* é de responsabilidade da IBM. É uma ferramenta de gerenciamento de artefatos de software, que oferece recursos e processos que podem ser implementados e personalizados sob demanda. Pode ser integrada com a *Rational ClearQuest*, uma ferramenta flexível de controle de mudanças e defeitos. Ela unifica o processo de mudança sobre o ciclo de vida de desenvolvimento e é escalonável, a todo tamanho de empresa, sem mudança de ferramentas ou processos;

- **BitKeeper Source Management**[15]: Comercializada pela BitMover, Inc., esta ferramenta faz controle de mudanças, dando suporte à visualização de mudanças, detecção e resolução

[14] http://www-306.ibm.com/software/awdtools/clearcase/
[15] http://www.bitkeeper.com/

de conflitos de mudanças, e combinação de mudanças em um arquivo;

- **Visual Source Safe**[16]: Fabricada pela *Microsoft Corporation*, é uma ferramenta que trabalha junto com outro produto da *Microsoft*, o *Visual Studio .NET*. Controla arquivos-fonte que podem ser classificados como valiosos, controla o uso concorrente de arquivos, uso de "etiquetas" em arquivos para controle de versão, controle de mudanças em nível de linhas de código, entre outros;

- **AllChange**[17]: *AllChange* é um sistema integrado de suporte ao Gerenciamento de Configuração. As principais funcionalidades são: rastreabilidade de mudanças; segurança de acesso a artefatos controlados; trabalho cooperativo; e disponibilização de informação. É comercializado pela *Intasoft*; e

- **ChangeMan**[18]: Comercializada pela *Serena*, a *ChangeMan* gerencia alterações paralelas em todas as aplicações e oferece recursos como o gerenciamento de configuração de sistemas distribuídos e construção automática da aplicação e implantação.

Porém, o custo destas ferramentas é elevado. Para um projeto de pesquisa, é mais interessante a utilização de software livre. Dentre as ferramentas livres de controle de versões, destacam-se:

- **CVS - Concurrent Versions System**[19]: O CVS é um software livre de controle de versões. Ele mantém um repositório de artefatos, e dá suporte à combinação e efetivação de mudanças, e permite a verificação de diferenças entre as versões. Atualmente, é um software livre amplamente utilizado e conhecido;

- **Subversion**[20]: A proposta do *Subversion* é ser um substituto para o CVS, melhorando algumas de suas características. Possui a maioria das funcionalidades do CVS, inclusive a sua interface; e

- **CBE - Code Building Environment**[21]: Também similar ao CVS, o CBE é um sistema totalmente escrito em Java com funções para o controle de revisões. Possui algumas outras funcionalidades, como renomear arquivos ou a utilização de banco de dados.

Dentre estas ferramentas, o CVS se destaca por possuir uma característica que o coloca à frente dos demais. Ele é amplamente utilizado. Praticamente todas as comunidades de software

[16] http://msdn.microsoft.com/vstudio/previous/ssafe/
[17] http://www.intasoft.net/
[18] http://www.serena.com/Products/changeman/Home.asp
[19] http://www.cvshome.org/
[20] http://subversion.tigris.org/
[21] http://sourceforge.net/projects/cbe

livre, como Sourceforge[22], Java.net[23], Tigris[24], e os projetos abertos, como Netbeans[25], mantêm seus artefatos organizados em repositórios CVS. As empresas também o utilizam, principalmente por ser uma ferramenta simples, aberta e já consolidada. Também por sua popularidade, é grande a quantidade de APIs e outros softwares de suporte ao CVS.

O CVS (*Concurrent Versions System*, ou Sistema de Versões Concorrentes) (CVS, 2005) é um sistema amplamente utilizado para auxiliar no controle de versões durante o desenvolvimento de software. Tendo se originado como um conjunto de comandos para ser utilizado no sistema operacional Unix, o CVS faz parte de um projeto aberto, podendo ser livremente utilizado e modificado.

No CVS, um conjunto de arquivos é mantido em um único local, em uma área comum. Quando deseja modificar algum arquivo, o Engenheiro de Software o recupera da área comum para uma área de trabalho local. Ao finalizar as modificações, o Engenheiro de Software o retorna à área comum. Ao fazer isso, o CVS armazena a versão antiga do arquivo, mantendo assim um histórico das diferentes versões de cada arquivo.

Caso dois desenvolvedores modifiquem um mesmo arquivo ao mesmo tempo, o CVS tentará mesclar as duas versões em uma única. Porém, isso só é possível caso as modificações tenham sido feitas em partes diferentes do arquivo. Caso contrário, ocorrerá um conflito, que deverá ser resolvido manualmente.

O CVS também oferece recursos para evitar a ocorrência de conflitos. Opcionalmente, o CVS pode manter uma lista das pessoas que estão trabalhando com o arquivo, e notificar cada uma delas na ocorrência de uma mudança. Também é possível visualizar, no momento da retirada do arquivo, todos os desenvolvedores que o estão modificando. Uma outra opção, mais drástica, é bloquear o arquivo para escrita. Dessa maneira, somente um desenvolvedor pode modificar o arquivo a cada vez. Porém, essa não é uma solução muito produtiva, visto que pode prejudicar o trabalho de outro desenvolvedor.

Com as informações contidas no CVS, é possível examinar cada modificação realizada em cada arquivo, ao longo do tempo. Isso facilita, por exemplo, a descoberta de erros, permitindo saber em qual momento e em qual versão os mesmos foram inseridos.

Além disso, com o CVS é possível cadastrar usuários, atribuindo permissões individuais de acesso, de forma que se possa controlar o acesso aos artefatos. Outro recurso importante é que o CVS não armazena integralmente todas as versões de todos os artefatos, o que consumiria muito espaço em disco. Ao invés disso, ele armazena integralmente apenas a primeira versão do artefato,

[22] http://sourceforge.net
[23] http://www.java.net/
[24] http://www.tigris.org/
[25] http://www.netbeans.org/

e para as versões subseqüentes, armazena apenas a diferença em relação à anterior.

2.4 Busca de componentes

Dentro do contexto deste trabalho, que busca estudar a reutilização através do uso de componentes de software e CASE, uma importante questão deve ser levantada: uma vez que se tenha um meio de armazenar uma grande quantidade de componentes reutilizáveis, como possibilitar, de modo eficiente, que o Engenheiro de Software encontre aquele que satisfaz suas necessidades? E isso deve ser possível sem depender de uma busca manual, o que, dependendo do tamanho deste repositório, pode ser inviável.

Por esse motivo, foi feito um levantamento bibliográfico (LUCRÉDIO; ALMEIDA; PRADO, 2004) dos trabalhos relacionados à busca e recuperação de artefatos reutilizáveis. Foram pesquisados trabalhos desde o começo da década de 90 até os dias atuais. A Figura 2 ilustra a evolução dessa área.

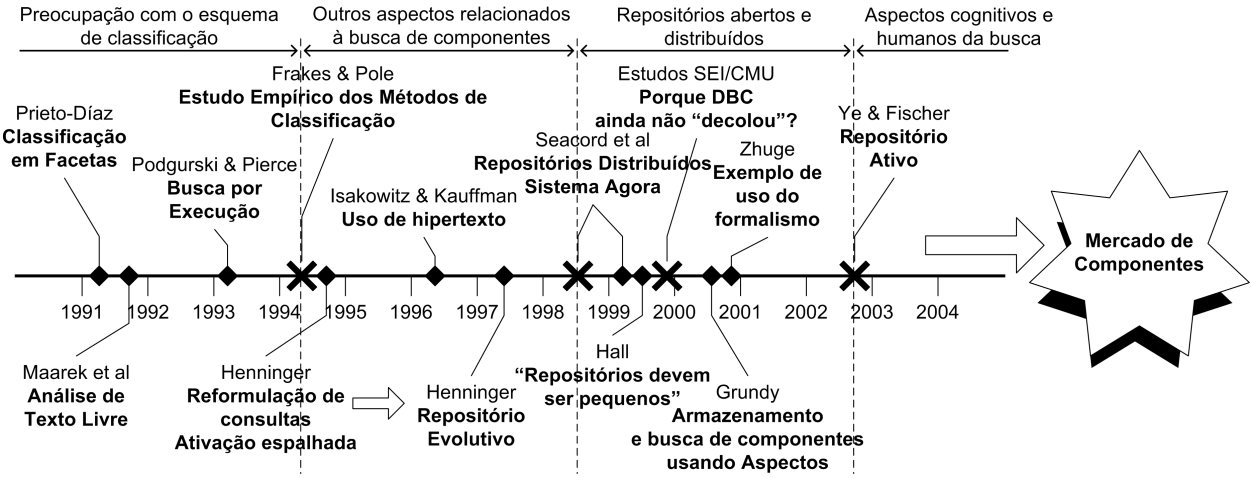

Figura 2: Evolução das pesquisas na área de armazenamento e busca de componentes de software.

Existem alguns trabalhos-chave (representados por um "X" na Figura 2) que marcam as principais mudanças na pesquisa dessa área. Inicialmente, o foco estava no esquema de classificação utilizado.

Em (PRIETO-DÍAZ, 1991), o autor apresenta a utilização de um esquema baseado em facetas para classificar componentes de uma biblioteca de software. Nesta abordagem, define-se um número limitado de características que um componente pode possuir (facetas). A cada característica é associado um conjunto de possíveis termos. Um componente é então descrito escolhendo-se os termos que melhor o enquadrem em cada faceta. Outra abordagem é apresentada em (MAAREK; BERRY; KAISER, 1991), onde os autores utilizam o conceito de indexação automática, que consiste

em automaticamente extrair, a partir de descritores em texto livre, termos que melhor representem o componente em questão. Em (PODGURSKI; PIERCE, 1993), os autores exploram a busca por componentes através de sua execução. O mecanismo funciona do seguinte modo: dada uma série de valores de entrada, e um conjunto de valores esperados de saída, os componentes são executados, e aqueles que produzirem a resposta esperada são recuperados.

Em 1994, o trabalho de (FRAKES; POLE, 1994) mostrou que os métodos de classificação existentes na época não eram muito diferentes entre si. A partir deste marco, as pesquisas começaram a considerar outros aspectos relacionados à busca de componentes. Nesse sentido, (HENNINGER, 1994) propõe a utilização de técnicas para que o usuário possa refinar suas expressões de busca, a fim de obter melhores resultados, e também recuperar não somente componentes que atendam exatamente à consulta, mas também componentes próximos a ela, aumentando a chance de encontrar o componente desejado. Em uma evolução deste trabalho (HENNINGER, 1997), o autor descreve um repositório de componentes que evolui com o tempo. À medida que o reutilizador formula suas consultas, os índices são automaticamente revistos e modificados, se necessário, de forma que em futuras consultas os componentes sejam encontrados mais facilmente. Outra tentativa interessante de se aprimorar os resultados da busca é apresentada em (ISAKOWITZ; KAUFFMAN, 1996). Neste trabalho os autores propõem a utilização de um mecanismo de navegação baseado em hipertexto, onde o reutilizador pode "navegar" através do repositório para encontrar o componente desejado.

O período seguinte é marcado por uma série de questionamentos. Em 1998, primeiramente levantou-se a questão de que repositórios deveriam ser abertos e distribuídos (SEACORD; HISSAM; WALLNAU, 1998; SEACORD, 1999). Em (HALL, 1999), o autor questiona se é verdadeira a idéia de que grandes repositórios de componentes são um fator decisivo para o sucesso da reutilização. Segundo o autor, é mais importante manter um conjunto restrito de componentes reutilizáveis, de um domínio de aplicações específico, do que manter e gerenciar um grande número de componentes variados. O estudo realizado no SEI/CMU entre 1999 e 2000 (BASS et al., 2000) também foi importante, pois buscou identificar áreas-chave para as pesquisas envolvendo Engenharia de Software Baseada em Componentes.

Outros trabalhos desse período incluem (GRUNDY, 2000), onde o autor propõe exportar as propriedades dos componentes, os eventos relacionados e seus métodos, em várias unidades, que representam os diferentes aspectos do componente. Estes aspectos são indexados em uma estrutura em facetas, utilizada para busca. Um outro tipo de abordagem pode ser visto em (ZHUGE, 2000). Este é um exemplo, entre outros trabalhos, que utiliza formalismo para descrever o comportamento do componente. A vantagem é que se tem a garantia de que os componentes recuperados irão atender aos requisitos especificados durante a busca. Além disso, a informação contida na descrição do componente vai além de palavras-chave ou termos isolados, permitindo conhecer exatamente a

semântica de sua funcionalidade, sem ambigüidade.

Finalmente, em 2002, o trabalho de (YE; FISCHER, 2002) reuniu vários conceitos, técnicos e não-técnicos. A idéia defendida é que o repositório deve ser responsável por antecipar a necessidade do usuário e entregar essa informação mesmo que ela não seja explicitamente requisitada. Esse tipo de repositório é chamado de repositório ativo.

Considerando-se o estado atual da pesquisa na área de Engenharia de Software Baseada em Componentes, a pesquisa atual e futura relacionada ao armazenamento e busca de componentes de software está convergindo para auxiliar no surgimento de um mercado de componentes.

Esse mercado exigirá repositórios distribuídos pela Internet, que possam ser facilmente acessados por desenvolvedores. Deverá permitir o armazenamento de componentes em diversos formatos, que serão executados em diferentes plataformas, e em diferentes modelos de componentes.

O papel que os mecanismos de busca irão desempenhar nesse mercado de componentes será muito importante, visto que eles deverão ser capazes de atuar em repositórios de escala global, sendo responsáveis por encontrar componentes que atendam às necessidades do Engenheiro de Software, possibilitando uma reutilização efetivo e em larga escala.

Mais detalhes sobre este assunto, como por exemplo algumas limitações dessas abordagens e as possíveis contribuições nesta área, podem ser obtidos em (LUCRÉDIO; ALMEIDA; PRADO, 2004), que detalha o estudo sobre busca de componentes que foi realizado como parte deste trabalho de pesquisa.

3 A Ferramenta MVCASE

A ferramenta MVCASE é uma ferramenta de modelagem baseada na UML, que vem sendo desenvolvida desde 1997 (BARRÉRE, 1999) por alunos de mestrado e iniciação científica do Departamento de Computação da Universidade Federal de São Carlos, auxiliando no andamento de diversos trabalhos de pesquisa, conforme pode ser visto em (PRADO; LUCRÉDIO, 2000, 2001; NOVAIS, 2002; ALMEIDA et al., 2002a, 2002b; LUCRÉDIO et al., 2003b; MORAES, 2004; GARCIA et al., 2004).

Totalmente escrita em Java, a MVCASE auxilia o Engenheiro de Software nas atividades relativas à análise, projeto, implementação e implantação de sistemas de software orientados a objetos. A Figura 3 mostra como se encaixa a MVCASE dentro da classificação de tecnologia CASE proposta por (FUGGETTA, 1993).

Classes de CASE	
Planejamento e modelagem de negócio	MVCASE
Análise e projeto	
Programação	
Desenvolvimento de interface com o usuário	
Verificação e validação	
Manutenção e engenharia reversa	
Gerenciamento de configuração	
Gerenciamento de projetos	

Figura 3: Classificação da MVCASE.

Essa classificação de Fuggetta considera as diferentes atividades do ciclo de vida do software, e classifica as tecnologias CASE de acordo com as atividades onde são utilizadas. Nesse contexto, a MVCASE pode ser classificada como uma CASE que atua no planejamento e modelagem de negócio, servindo para criar modelos complexos de negócio, identificando requisitos e fluxos de informação (FUGGETTA, 1993). Também oferece recursos gráficos para criar modelos de análise e projeto e auxílio à codificação. A seguir são apresentados os principais recursos disponíveis na MVCASE.

3.1 Modelagem Textual e Gráfica

A Figura 4 mostra a tela principal da interface gráfica da ferramenta.

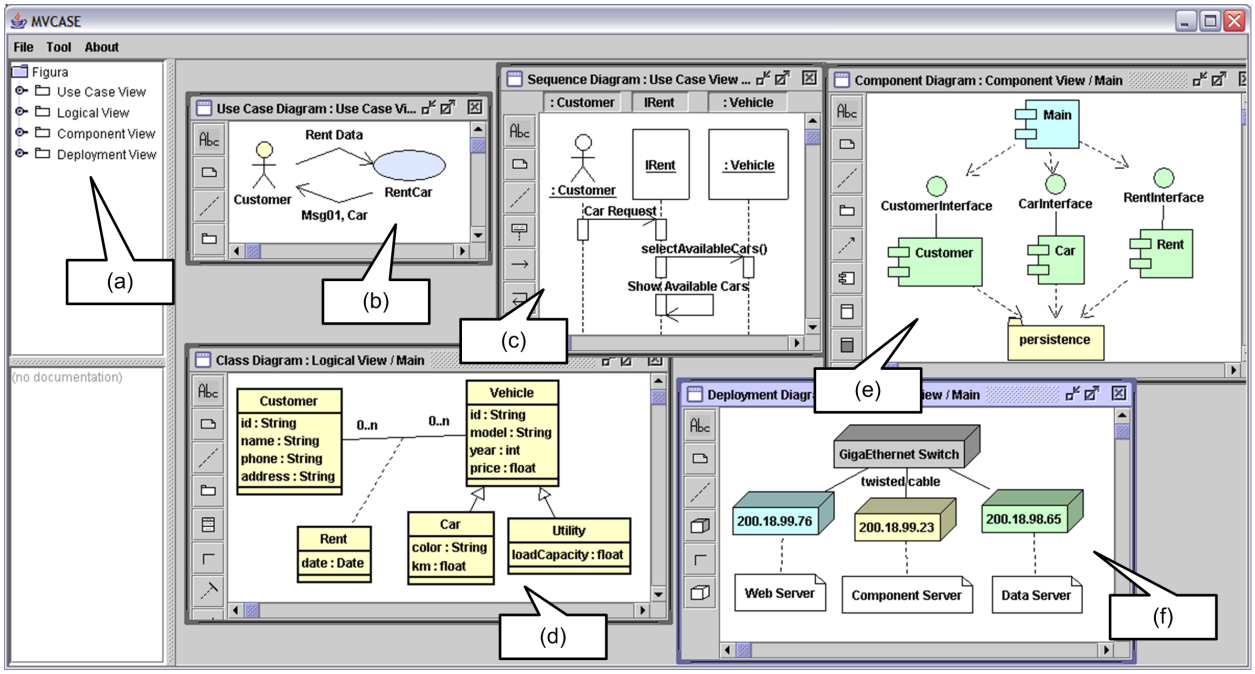

Figura 4: Interface gráfica da MVCASE.

A MVCASE oferece suporte à modelagem segundo a UML em quatro visões (a): Visão de Casos de Uso, Visão Lógica, Visão de Componentes e Visão de implantação.

A Visão de Casos de Uso mostra uma visão comportamental, do ponto de vista dos atores externos. Envolve diferentes técnicas de representação dos casos de uso, destacando-se: o diagrama de casos de uso (b), que mostra um conjunto de casos de uso, atores e seus relacionamentos; e o diagrama de seqüência (c), que detalha um curso, normal ou alternativo, de um caso de uso, mostrando um conjunto de objetos e as mensagens por eles enviadas e recebidas (BOOCH; JACOBSON; RUMBAUGH, 1999). Outras técnicas, como o Diagrama de Colaboração (BOOCH; JACOBSON; RUMBAUGH, 1999), também podem ser utilizadas nesta visão.

A Visão Lógica oferece uma visão estruturada em uma coleção de pacotes, classes e relacionamentos. Dentre as técnicas utilizadas nesta visão destacam-se: o diagrama de classes (d), que fornece uma visão estática, estruturada em um conjunto de classes, interfaces, colaborações e seus relacionamentos; e o diagrama de estados, que fornece uma visão dinâmica, representada através de uma máquina de estados (BOOCH; JACOBSON; RUMBAUGH, 1999).

A Visão de Componentes fornece uma visão estática, estruturada em módulos. A principal técnica dessa visão é o diagrama de componentes (e), que mostra componentes e seus relaciona-

mentos (BOOCH; JACOBSON; RUMBAUGH, 1999).

A Visão de Implantação fornece uma visão estática da implantação de uma arquitetura. Destaca-se nesta visão o diagrama de implantação (f), que mostra um conjunto de elementos processadores e seus relacionamentos (BOOCH; JACOBSON; RUMBAUGH, 1999).

3.2 Desenvolvimento Baseado em Componentes Distribuídos

A MVCASE também oferece suporte para o Desenvolvimento Baseado em Componentes Distribuídos (DBCD). Guiado pelo modelo incremental de processo proposto por (ALMEIDA et al., 2004), o Engenheiro de Software pode utilizar a ferramenta nas diversas etapas do desenvolvimento, desde as especificações do domínio do problema até a implementação, adicionando, a cada passo, requisitos funcionais e não funcionais, como por exemplo, persistência e distribuição. Conforme mostra a Figura 5, o modelo propõe duas fases distintas.: Desenvolver Componentes Distribuídos e Desenvolver Aplicações Distribuídas.

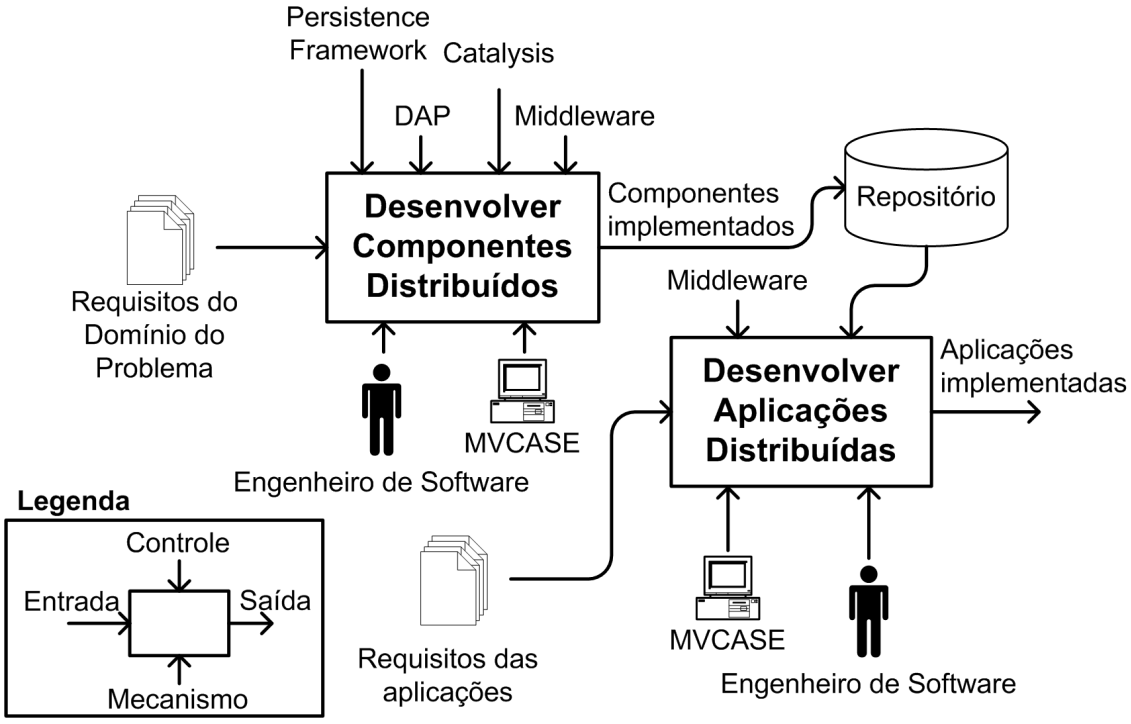

Figura 5: Modelo Incremental de Processo para DBCD (ALMEIDA et al., 2004).

Na primeira fase, **Desenvolver Componentes Distribuídos**, parte-se dos requisitos do domínio do problema, e produzem-se componentes implementados em uma linguagem orientada a objetos (desenvolvimento "para reutilização"). Uma vez implementados, os componentes são armazenados em um repositório. Na segunda fase, **Desenvolver Aplicações Distribuídas**, o Engenheiro de Software consulta no repositório os componentes disponíveis para um domínio do

problema. Depois de identificados os componentes necessários, as aplicações que os reutilizam são desenvolvidas (desenvolvimento “com reutilização”) (ALMEIDA et al., 2004).

3.3 Utilização de Padrões de Projeto

Na MVCASE, também é possível utilizar padrões de projeto (LUCRÉDIO et al., 2003b). A partir da estrutura de um padrão, a MVCASE importa os padrões e os disponibiliza ao Engenheiro de Software, para que ele possa modificá-la para a situação específica. A Figura 6 ilustra esse processo, onde o padrão *Adapter* (GAMMA et al., 1995) é importado na MVCASE.

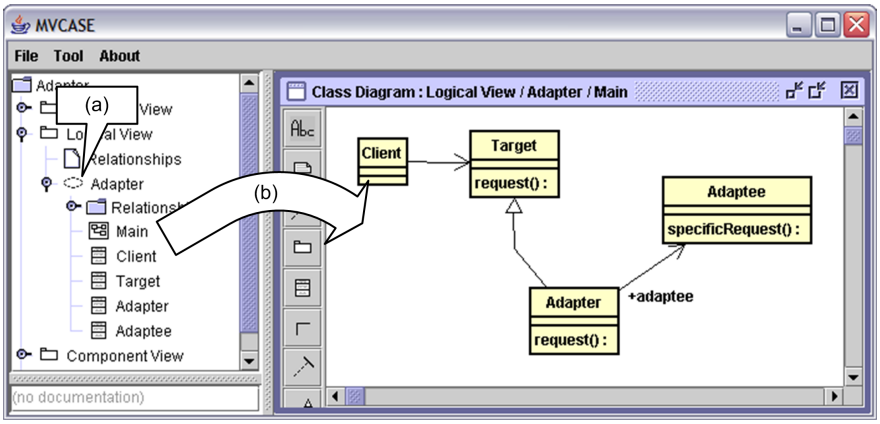

Figura 6: Padrão *Adapter* sendo importado na MVCASE.

Na Figura 6, os elementos do padrão *Adapter* são importados para o *browser* da MVCASE (a). Em seguida, estes elementos são “arrastados” para o projeto do software (b). Se existem relacionamentos entre os elementos, os mesmos são automaticamente inseridos. Uma vez inseridos no projeto, o Engenheiro de Software pode modificar os elementos para uma situação específica.

Obviamente, esse modelo de aplicação de padrões é limitado aos padrões de projeto que possuam uma estrutura bem definida e que possa ser modificada facilmente. No caso do *Adapter*, por exemplo, a aplicação do padrão pode ser feita apenas substituindo-se os nomes dos elementos. Em outros casos, a aplicação do padrão é mais complexa, e sua reutilização fica restrita à visualização da estrutura. Outra limitação é que a MVCASE foca a abordagem *top-down* (FLORIJN; MEIJERS; WINSEN, 1997), ou seja, os elementos do padrão são sempre importados, não importa se os mesmos já existam no projeto do software. Sendo assim, elementos já existentes do projeto devem ser modificados e/ou descartados, o que exige um trabalho extra.

3.4 Suporte para Diferentes Modelos de Componentes

A MVCASE possui recursos para auxiliar no projeto de componentes, seguindo os padrões definidos em um modelo de componentes, e automaticamente gerando os elementos relacionados ao modelo de componentes utilizado. A Figura 7 ilustra este processo para o modelo de componentes EJB (DEMICHIEL, 2002).

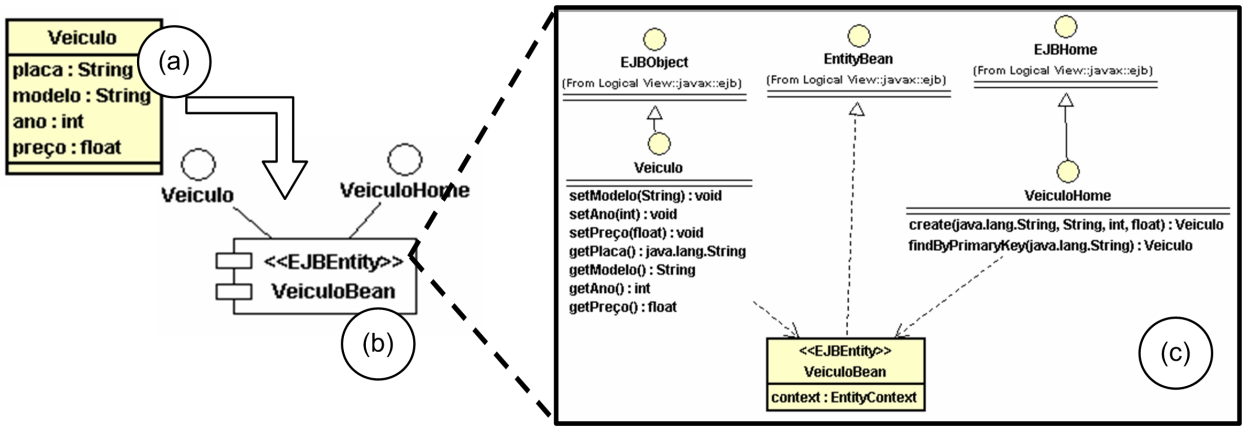

Figura 7: Projeto de um componente na MVCASE, segundo a tecnologia EJB.

A partir de uma especificação de classe em alto nível (a), a MVCASE gera um componente (b) de acordo com o modelo EJB. A estrutura interna do componente (c) também é gerada, com duas interfaces, no caso **Veiculo**, que contém as regras de negócio, e **VeiculoHome**, que contém operações relativas ao ciclo de vida do componente, incluindo operações para criar, localizar e destruir instâncias.

Até o presente momento, a MVCASE oferece suporte para automatizar o projeto segundo os modelos EJB, CORBA (OMG, 2002a) e um modelo de componentes baseado em *Delphi*, conforme pode ser visto em (MORAES, 2004).

3.5 Desenvolvimento de Software Orientado a Aspectos

A MVCASE possui suporte para os conceitos básicos do paradigma orientado a aspectos (GARCIA et al., 2004), como definidos na linguagem *AspectJ* (KICZALES, 2001).

A principal construção em AspectJ é o *aspecto*. Um aspecto define uma funcionalidade específica que pode afetar diferentes partes de um sistema, como, por exemplo, persistência em banco de dados. Além disso, os aspectos podem alterar a estrutura dinâmica de um sistema por meio dos *pontos de junção* ou, ainda, alterar a estrutura estática por meio de uma *declaração intertipos*. Um aspecto normalmente define um conjunto de pontos de junção e adiciona comportamentos a eles

por meio dos *adendos* [1].

Sabe-se que estes conceitos foram originalmente propostos considerando-se apenas a sua implementação, por meio de uma linguagem de programação (KICZALES, 2001). Contudo, a definição de um padrão para o projeto de software orientado a aspectos, a partir de modelos e técnicas de modelagem orientadas a aspectos independentes da linguagem de programação, vem despertando o interesse de pesquisadores.

Com essa motivação, foi feito um estudo das diferentes propostas para a modelagem orientada a aspectos, onde alguns pontos em comum foram identificados, indicando uma conformidade na representação de alguns conceitos do paradigma orientado a aspectos. A seguir, foi elaborada uma notação para representação dos conceitos do paradigma orientado a aspectos.

A MVCASE dá suporte a esta notação através de sua interface visual, com menus e botões que permitem a criação dos elementos referentes ao paradigma orientado a aspectos. A Figura 8 mostra uma tela da MVCASE com um diagrama contendo o padrão *Observer* (PIVETA; ZANCANELLA, 2003), utilizado neste exemplo para monitorar variações de temperatura, modelado segundo a notação definida em (GARCIA et al., 2004).

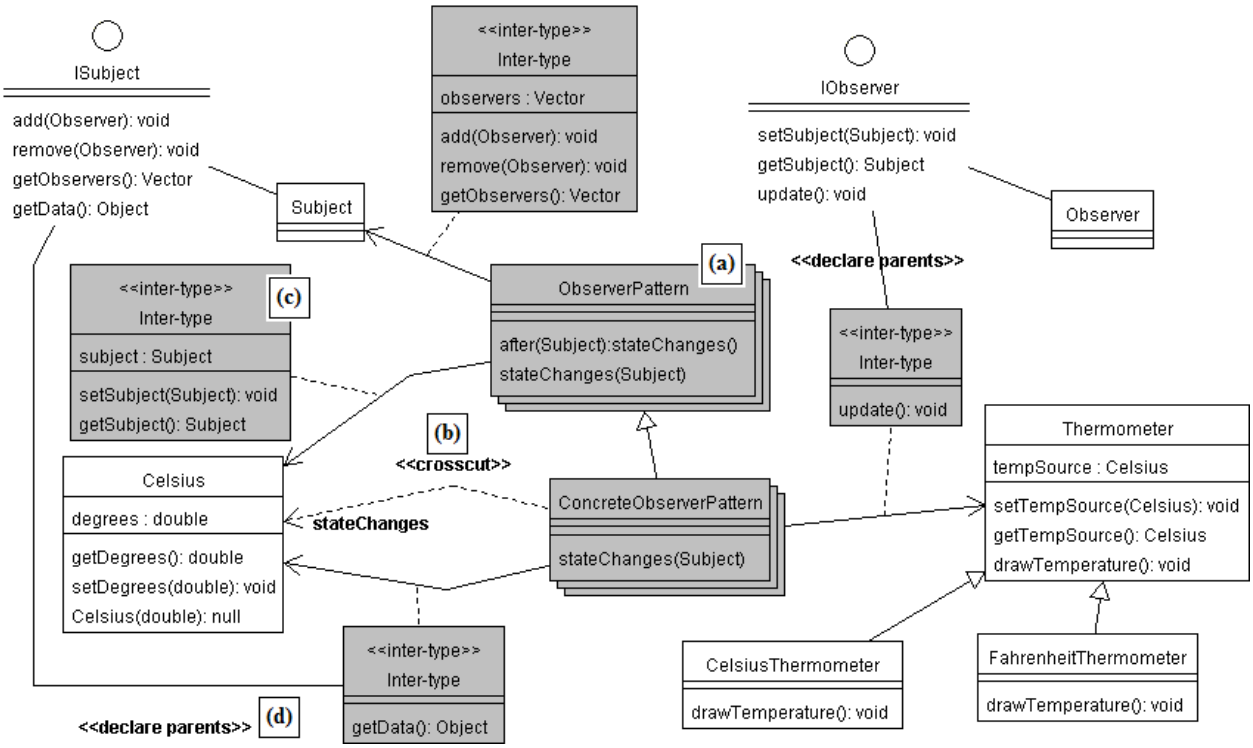

Figura 8: Padrão *Observer* modelado na MVCASE.

[1] As sugestões de tradução dos termos comumente utilizados na POA, definidas pela Comunidade Brasileira de Desenvolvimento de Software Orientado a Aspectos por meio da sua *mailing list* (aosd-br@yahoogrupos.com.br) e no Primeiro Workshop Brasileiro de Desenvolvimento de Software Orientado a Aspectos, podem ser encontrados em: http://twiki.im.ufba.br/bin/view/AOSDbr/TermosEmPortugues

Nessa notação, um aspecto é representado por vários retângulos sobrepostos (a). Dentro do aspecto existem três compartimentos. A exemplo da notação UML convencional, os dois primeiros são relativos aos atributos e métodos. Esta notação introduz um terceiro compartimento, com os adendos e conjuntos de pontos de junção. Além do aspecto, é possível representar os relacionamentos de entrecorte (b) entre aspectos e classes. Para isso, foi utilizada uma abstração da UML (OMG, 2003b), com o estereótipo "**crosscuts**". Modificações na estrutura de uma classe são representadas através de uma classe de associação com o estereótipo "**inter-type**" (c). Finalmente, modificações na estrutura hierárquica das classes são representadas através do relacionamento de realização da UML (OMG, 2003b), com o estereótipo "**declare parents**" (d).

Além da modelagem, a MVCASE também gera automaticamente código em *AspectJ* (KICZALES, 2001) para os modelos orientados a aspectos. O código gerado é testado e caso erros sejam encontrados, ou novos requisitos sejam identificados, o Engenheiro de Software pode trabalhar no modelo e novamente gerar o código em *AspectJ*.

3.6 Geração de Código

Para auxiliar nas atividades de implementação, a MVCASE provê assistentes para automatizar tarefas como geração de código, empacotamento e publicação de componentes. Uma das principais características da MVCASE é que ela pode gerar código completamente funcional, e não apenas estruturas de classes e assinaturas de operações. A MVCASE permite que o Engenheiro de Software insira código executável diretamente nas operações dos modelos de classes. Quanto mais completas estiverem essas operações, mais completo será o código gerado.

Esse recurso ajuda a acelerar as iterações que ocorrem em modelos de processo evolutivos. A Figura 9 mostra uma comparação entre um modelo de processo evolutivo típico e um modelo de processo auxiliado pela MVCASE. Em um típico modelo de processo evolutivo, as fases de análise e projeto são seguidas por uma fase de implementação, onde é construído um protótipo, que é depois testado. Com a MVCASE, o código deste protótipo é automaticamente gerado, exigindo menos esforço nesta fase. Alguns ajustes podem ser feitos após a geração de código, porém a maior parte do trabalho já está pronto.

Atualmente, a MVCASE oferece suporte para a geração de código nas linguagens Java, AspectJ (GARCIA et al., 2004), XML (W3C, 2005) e SQL (ASSÁO, 2003).

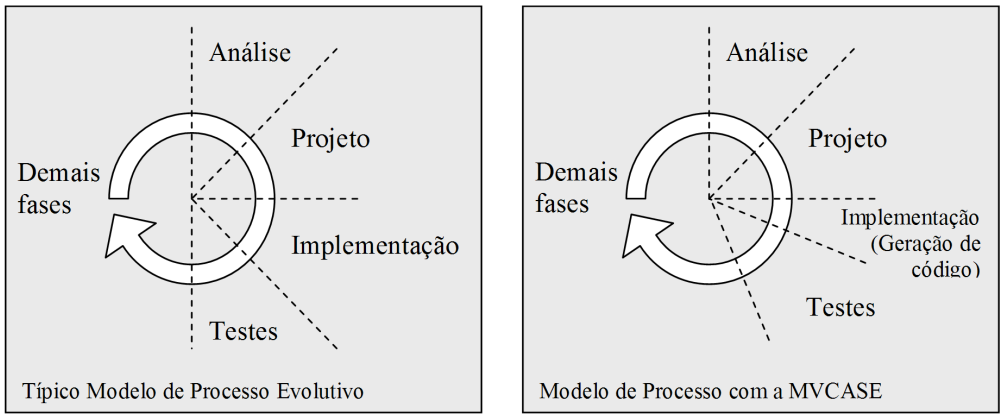

Figura 9: Comparação entre um modelo de processo evolutivo típico e um modelo de processo utilizando a MVCASE.

3.7 Dados sobre a Utilização da Ferramenta

Desde 1997, a MVCASE vem sendo utilizada em diferentes cenários. Um total de vinte projetos de pesquisa, envolvendo três universidades brasileiras, se relacionam com a MVCASE, incluindo:

- 7 projetos de iniciação científica;
- 11 projetos de mestrado;
- 2 projetos de doutorado

Desde 2000, vem sendo constantemente utilizada em aulas de Engenharia de Software, em diferentes cursos de graduação e pós-graduação. Os estudantes utilizam a MVCASE durante as aulas para praticar conceitos de modelagem e desenvolvimento baseado em componentes, e também durante seus trabalhos.

A divulgação da MVCASE envolveu várias apresentações, em empresas e conferências, no Brasil (ALMEIDA et al., 2002b; GARCIA et al., 2004; LUCRÉDIO et al., 2003b), Canadá (ALVARO et al., 2003), Coréia (ALMEIDA et al., 2002a), França (LUCRÉDIO; ALMEIDA; PRADO, 2004), Hong Kong (ALMEIDA et al., 2004) e Estados Unidos (LUCRÉDIO et al., 2004). Em três ocasiões a ferramenta foi premiada:

- Melhor ferramenta do SBES'2001 - XV Simpósio Brasileiro de Engenharia de Software - Sessão de ferramentas (PRADO; LUCRÉDIO, 2001)
- Terceira melhor ferramenta do SBES'2002 - XVI Simpósio Brasileiro de Engenharia de Software - Sessão de ferramentas (ALMEIDA et al., 2002b)

- Entre as melhores ferramentas do SBES'2004 - XVIII Simpósio Brasileiro de Engenharia de Software - Sessão de ferramentas (GARCIA et al., 2004)

4 *Extensão da Ferramenta MVCASE com Serviços Remotos de Armazenamento e Busca de Artefatos de Software*

Nos capítulos anteriores foram apresentados os principais conceitos envolvidos nesta pesquisa, que fornecem o embasamento teórico para o desenvolvimento dos serviços remotos de armazenamento e busca de artefatos de software. Neste capítulo é apresentada a extensão da MVCASE para prover estes serviços.

Conforme já discutido no capítulo 1, as extensões desenvolvidas nesta pesquisa tiveram como principal objetivo superar algumas limitações da ferramenta MVCASE, principalmente quanto à sua capacidade de representação de artefatos e disponibilizá-los para reutilização. Também foi objetivo desta pesquisa explorar os mecanismos envolvidos nesta extensão, de modo a contribuir com o avanço científico na área da reutilização de software.

Com estes objetivos em mente, foram definidos alguns requisitos a serem atendidos, apresentados a seguir.

4.1 Requisitos

Os requisitos considerados no desenvolvimento da extensão da ferramenta MVCASE se dividem em algumas áreas. Os requisitos dizem respeito: ao **formato para representação dos artefatos**, à possibilidade de **interoperabilidade com outras ferramentas**, ao **trabalho em equipe**, à **reutilização dos artefatos produzidos pela ferramenta**, à **disponibilização da MVCASE como software livre**, e à **contribuição em termos de avanço científico**.

Com relação ao formato para representação dos artefatos, considerou-se os seguintes requisitos:

- **Ser um padrão bem estabelecido**: Sendo um padrão, é mais provável que um número maior de pessoas conheçam o formato, fazendo com que a ferramenta se torne mais amigável

com relação aos seus desenvolvedores. Além disso, um padrão só se estabelece após ter sido colocado em prática, testado e validado em inúmeros cenários. Outra característica importante de um padrão é que o mesmo agrega conhecimentos e experiências valiosos adquiridos de diversas pessoas e organizações. Utilizando tal padrão na ferramenta, tais conhecimentos e experiências contribuem com a qualidade da ferramenta;

- **Ser flexível**: A flexibilidade é um requisito indispensável. Novas tecnologias surgem a todo momento, e tecnologias antigas evoluem para se adaptar a novas tendências. Sendo assim, o formato com que os artefatos da MVCASE são representados e armazenados deveria ser passível de extensão e/ou adaptação para acompanhar estas mudanças sem a necessidade de grande retrabalho; e
- **Ser genérico**: Apesar de ser utilizado para representar modelos UML, o formato utilizado pela MVCASE deveria ser genérico, capaz de representar outros tipos de artefato. Isto facilitaria a extensão da ferramenta com novas funcionalidades, e sua integração com outras ferramentas.

Outro requisito considerado nesta pesquisa foi tornar a MVCASE apta a cooperar com outras ferramentas em um ambiente CASE de desenvolvimento de software. Neste sentido, pode-se destacar dois níveis de interoperabilidade:

- **Integração de dados**: Os dados produzidos pela MVCASE deveriam ser compreensíveis por outras ferramentas, sem a necessidade de conversores ou adaptadores complexos; e
- **Integração de funcionalidades**: As funções desempenhadas pela ferramenta deveriam poder ser requisitadas por outras ferramentas, de modo a facilitar a cooperação.

O compartilhamento de artefatos entre os membros de uma equipe também foi um requisito considerado nesta pesquisa. Estando inserida em um ambiente de desenvolvimento típico, a MVCASE deveria oferecer mecanismos para facilitar o intercâmbio dos artefatos nela produzidos, de modo que diferentes Engenheiros de Software pudessem trabalhar simultaneamente sobre um mesmo conjunto de artefatos sem as dificuldades existentes na versão original da MVCASE, conforme descrito no capítulo 1.

Outro requisito foi a possibilidade de busca e recuperação dos artefatos produzidos na MVCASE, visando sua reutilização. A idéia foi possibilitar que, uma vez armazenados, se fizesse a busca sobre os artefatos de forma automática. Este requisito é essencial à reutilização, uma vez que o número de artefatos tende a crescer, dificultando a busca manual. Além disso, este mecanismo de busca deveria:

- ser simples e fácil de usar (PRIETO-DÍAZ, 1991), diminuindo as barreiras para sua adoção pelos desenvolvedores de software;
- ser aberto, inclusivo e distribuído (SEACORD, 1999), de modo a aumentar o nível de reutilização que pode ser alcançado, incorporando um maior número de artefatos reutilizáveis e reduzindo os problemas dos repositórios centralizados;
- possibilitar um certo relaxamento na busca, idéia que é defendida por diversos autores da área de reutilização, como (PRIETO-DÍAZ, 1991), (ISAKOWITZ; KAUFFMAN, 1996) e (HENNINGER, 1994). Isto é necessário porque normalmente o reutilizador nem sempre conhece exatamente os requisitos do artefato que ele deseja encontrar, e um mecanismo muito exato poderia deixar de lado possíveis candidatos à reutilização.

Como também foi objetivo desta pesquisa disponibilizar a MVCASE como software livre, um importante requisito a ser atendido foi modificar a ferramenta para que a mesma tivesse uma arquitetura flexível e preparada para captar contribuições de diversos desenvolvedores. A ferramenta teria que ser mais bem dividida do que a sua versão original, de modo a minimizar as dificuldades enfrentadas pelos desenvolvedores.

Finalmente, por se tratar de um trabalho de pesquisa, todos estes requisitos deveriam ser atendidos de forma a contribuir com o avanço científico na área de reutilização de software, frente aos seus atuais desafios. As tecnologias desenvolvidas deveriam ser genéricas o suficiente para poder ser utilizadas fora deste cenário, utilizando outras ferramentas além da MVCASE, e com outros tipos de artefatos.

4.2 Esboço da solução

A Figura 10 esboça a solução empregada para atender aos requisitos descritos na seção anterior:

A MVCASE agora trabalha com um formato padronizado e flexível para representar seus artefatos (a). Dessa forma, é possível que outras ferramentas (b) possam compreender os dados por ela produzidos, facilitando a interoperabilidade. Nesta pesquisa, foi utilizado para isso o XMI, que possui as características necessárias para atender aos requisitos, conforme descrito mais adiante neste capítulo.

A MVCASE possui uma arquitetura em *plug-ins* (c), de modo que os membros da comunidade de software livre possam mais facilmente desenvolver extensões e contribuir com a evolução da ferramenta. O serviço de armazenamento (d), que também é um *plug-in* da MVCASE (e), é capaz

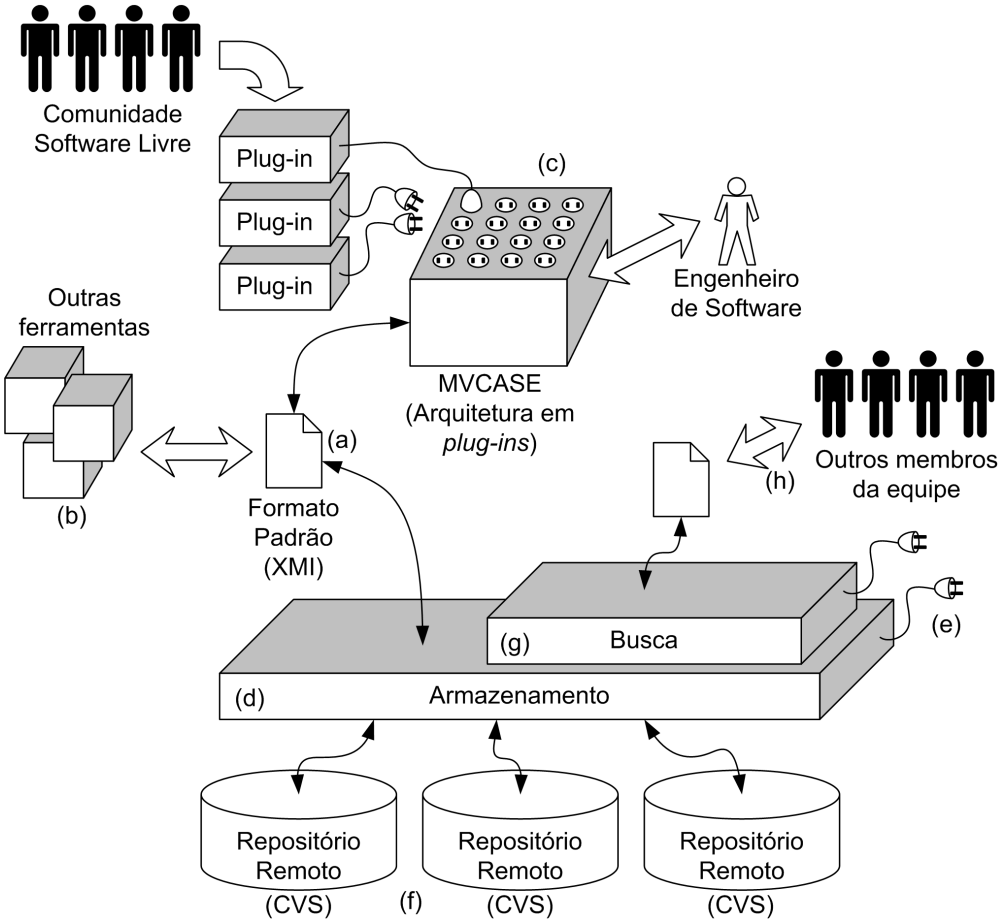

Figura 10: Esboço da nova arquitetura da MVCASE.

de armazenar artefatos em repositórios distribuídos e remotos (f). Nesta pesquisa, foram utilizados repositórios CVS. Uma vez armazenados, os artefatos podem ser recuperados com a ajuda do serviço de busca (g), que também atua sobre diversos repositórios. O serviço de busca também é um *plug-in* da ferramenta, mas que pode ser usado separadamente. Uma vez recuperado, outros membros da equipe podem trabalhar simultaneamente com este artefato (h).

Com essa abordagem, a maioria dos requisitos foi atendida, conforme apresentado no final deste capítulo. A seguir apresenta-se com mais detalhes como foi realizada a extensão da ferramenta.

4.3 Extensão da MVCASE

A extensão envolveu as seguintes alterações na ferramenta:

1. **Suporte para XMI**: Para atender aos requisitos referentes ao formato de armazenamento dos artefatos e interoperabilidade entre ferramentas, modificou-se a MVCASE para que a

mesma possa persistir seus modelos segundo o padrão XMI (OMG, 2002d);

2. **Serviço de armazenamento remoto**: Foi adicionado à MVCASE um módulo para acesso remoto ao CVS, de modo que o Engenheiro de Software possa armazenar e recuperar os artefatos produzidos em um repositório remoto, sem a necessidade de se preocupar com a transferência física dos seus arquivos. Este armazenamento remoto exige também um certo controle, uma vez que duas ou mais pessoas podem reutilizar e/ou modificar um dado artefato ao mesmo tempo, resultando em conflitos e inconsistência. Como este trabalho utiliza o CVS, a MVCASE também foi modificada para utilizar os recursos desse sistema para tentar reduzir a ocorrência destes conflitos e facilitar a reutilização; e

3. **Serviço de busca**: Uma vez que os artefatos estejam armazenados, é desejável que se possa realizar buscas, que pode ser o fator decisivo para que a reutilização aconteça. Isto porque podem existir diversos repositórios, com um número excessivo de artefatos reutilizáveis, de modo que uma busca manual e sequencial se torna proibitiva.

Para melhor compreender a natureza e o alcance dessas alterações, na próxima seção será apresentada a arquitetura interna original da MVCASE, antes da extensão.

4.3.1 Arquitetura original da MVCASE

A Figura 11 ilustra a arquitetura original da MVCASE, antes da extensão realizada nesta pesquisa.

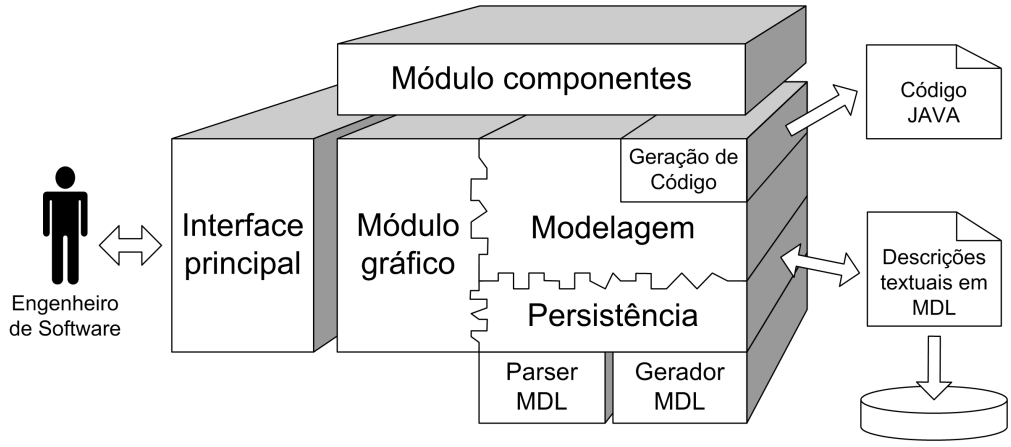

Figura 11: Arquitetura original da MVCASE.

O Engenheiro de Software acessava a ferramenta através de sua interface gráfica. Existia um módulo gráfico, responsável pela criação e edição dos modelos. Esse módulo gráfico tinha acesso a um módulo central, de modelagem, responsável pela representação dos modelos na memória. Para

persistir essa informação em disco, existia o módulo de persistência, composto por um gerador MDL, que criava descrições textuais referentes ao modelo em uma linguagem de descrição de modelos (MDL), e um *parser* MDL, que interpretava as descrições textuais e instanciava os modelos correspondentes na memória, tornando-os disponíveis para edição. O suporte para componentes era de responsabilidade de outro módulo, que acessava o módulo de modelagem.

Havia alguns problemas com essa arquitetura, dentre os quais se destacam:

- O módulo de modelagem era excessivamente carregado. Idealmente, ele teria de ser uma implementação do metamodelo da UML, para manipulação dos modelos criados pela MV-CASE. Porém, ele possuía funcionalidades diversas, como a geração de código e persistência, por exemplo;
- Os módulos eram muito dependentes entre si. O módulo gráfico estava excessivamente integrado ao módulo de modelagem. Mas principalmente, o módulo de modelagem estava excessivamente integrado ao módulo de persistência. Ou seja, os elementos do metamodelo eram os responsáveis por sua própria persistência em disco. Isso dificultava a realização de modificações, como por exemplo a mudança do metamodelo ou a mudança do mecanismo de persistência;
- Como havia uma concentração excessiva de código no módulo de modelagem, a inserção de novas funcionalidades era trabalhosa, e exigia cuidado especial para manter as funcionalidades existentes; e
- Havia pouco suporte para compartilhamento dos artefatos e reutilização. Para acessar um mesmo artefato simultaneamente através da rede, os Engenheiros de Software precisavam utilizar os serviços do sistema operacional. Não era possível que alterações fossem efetuadas no mesmo arquivo, sendo necessário que cada Engenheiro de Software obtivesse uma cópia do arquivo, e a combinação das alterações fosse realizada posteriormente. Este processo frequentemente resultava em inconsistência de dados, além de dificultar a reutilização.

O principal foco deste trabalho é o último item. A construção dos serviços remotos de armazenamento e busca de artefatos de software tiveram como motivação facilitar a reutilização. Porém, como pode ser visto mais adiante neste capítulo, as alterações também causaram impacto nos outros itens.

As próximas seções contêm uma descrição detalhada de cada uma das alterações efetuadas. As alterações foram realizadas exatamente na ordem em que são aqui apresentadas.

4.3.2 Suporte para XMI

O padrão XMI consiste na utilização da XML para representar dados. Conforme já descrito no capítulo 2, o XMI contém não só a representação dos dados, mas também os dados que o descrevem (metadados). Dessa forma, um documento XMI pode ser compreendido fora do contexto onde foi criado, facilitando a interoperabilidade e o intercâmbio de informações.

Como um documento XMI é um documento XML, interpretar os dados nele contidos é uma tarefa relativamente simples. Existem diversas APIs específicas para XML, destacando-se a SAX (*Simple Api for XML*) (SAX, 2005) e o DOM (*Document Object Model*) (W3C, 2004). Implementações para estas APIs existem em diferentes linguagens, incluindo Java.

Porém, carregar as informações contidas em um documento XMI para a memória é apenas parte da solução. A maneira com que os dados e metadados são manipulados é igualmente importante. Uma ferramenta que utilize o XMI como meio de persistência deve prover interfaces de acesso aos dados e metadados, além de métodos para escrever e ler documentos XMI.

O MOF (*Meta-Object Facility*) (OMG, 2002b) (vide capítulo 2) contém a definição dessas interfaces na linguagem IDL (*Interface Definition Language*) (OMG, 2002a). Sendo uma linguagem genérica, a IDL pode ser mapeada para qualquer linguagem de programação. A *Sun Microsystems* definiu então o padrão JMI (*Java Metadata Interface*) (DIRCKZE, 2002), que define regras de mapeamento dessas interfaces para a linguagem Java. Dessa forma, ferramentas escritas em Java podem acessar os dados e metadados do MOF de maneira padronizada.

O MOF (e por consequência o JMI) possui duas formas de acesso à informação. Uma delas é o acesso direto, que permite ler e modificar as informações através de métodos do tipo *set* e *get*, para atributos e referências monovaloradas, e métodos do tipo *add* e *remove* para atributos e referências multivaloradas.

A segunda forma de acesso é através do módulo reflexivo, que permite a introspecção sobre os objetos. Com esse módulo, é possível, por exemplo, descobrir o tipo de um objeto, seus atributos, métodos e relacionamentos. Na arquitetura do MOF (vide capítulo 2) isto significa navegar entre diferentes meta-níveis. É desta maneira que o MOF define a ligação entre os dados e os metadados.

Para que a MVCASE pudesse então dar suporte completo ao XMI e ao MOF, seria necessário implementar o padrão JMI para o metamodelo da UML. Porém, tais implementações já existem. A *homepage* do JMI[1] cita três implementações: a implementação de referência (UNISYS, 2002), da *Unisys Corporation*, o MDR (MetaData Repository) (NETBEANS, 2005b), parte do projeto *Netbe-*

[1] http://java.sun.com/products/jmi/implementations.html (Acesso em janeiro/2005)

ans (NETBEANS, 2005c), e o *SAP Netweaver*, da SAP [2].

O *SAP Netweaver* é um repositório de metadados utilizado em vários projetos da SAP. Porém, por não se tratar de um software livre, optou-se por não utilizá-lo nesta pesquisa. Já a implementação de referência da *Unisys Corporation* e o MDR do projeto *Netbeans* podem ser livremente utilizadas, de acordo com seu termo de licença de software livre. Ambas implementações seguem o padrão JMI e MOF, e possuem suporte para leitura e escrita de documentos XMI. A implementação da *Unisys*, apesar de ser livre, é uma versão simplificada de um produto comercial, e portanto possui algumas limitações. O MDR não possui essa característica, sendo por isso mais utilizado. Algumas ferramentas, como a *ArgoUML* e sua versão comercial, a *Poseidon*, utilizam o MDR, o que comprova sua estabilidade e confiabilidade. Por estes motivos, decidiu-se utilizar o MDR na extensão da MVCASE. Com essa decisão, poupou-se tempo e esforço de implementação, além de se reutilizar um software já testado, o que resultou em uma maior confiabilidade na ferramenta.

4.3.2.1 Integração do MDR na MVCASE

A Figura 12 ilustra a integração do MDR na MVCASE.

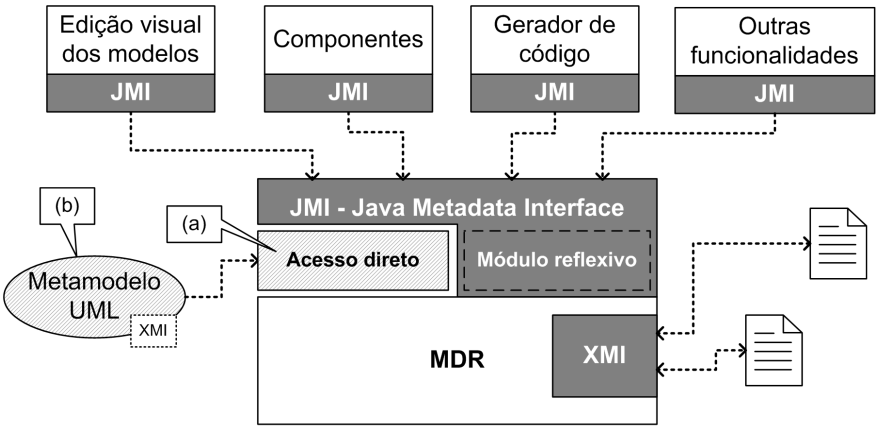

Figura 12: Integração do MDR na MVCASE.

O MDR implementa a leitura e escrita de documentos XMI. Também implementa o padrão JMI, de modo que as diferentes funcionalidades da MVCASE (edição visual de modelos, geração de código, etc.) acessam os dados através de suas interfaces padronizadas. As duas formas de acesso aos dados (reflexivo e direto) são implementadas.

O módulo reflexivo é implementado pelo MDR, e não exige nenhum tipo de configuração extra para que os dados sejam acessados através de métodos reflexivos. Porém, utilizar somente acesso reflexivo é trabalhoso, visto que é necessário realizar a introspecção sobre os objetos a fim de se descobrir seus atributos e métodos.

[2] http://www.sap.com/solutions/netweaver/index.aspx

O acesso direto é mais simples, porém exige que se tenha interfaces específicas para o metamodelo utilizado, no caso o da UML. Essas interfaces específicas e suas implementações são automaticamente geradas (a) pelo MDR, a partir da especificação em XMI do metamodelo (b). Utilizou-se uma especificação do metamodelo da UML 1.5 disponível na *homepage*[3] do próprio MDR, que deu origem a interfaces específicas para a UML. A Figura 13 ilustra essa geração para o elemento **Ator** da UML. A especificação do metamodelo da UML define a classe **Actor**, que dá origem à interface **ActorClass**, com métodos **createActor** para criação de um ator da UML.

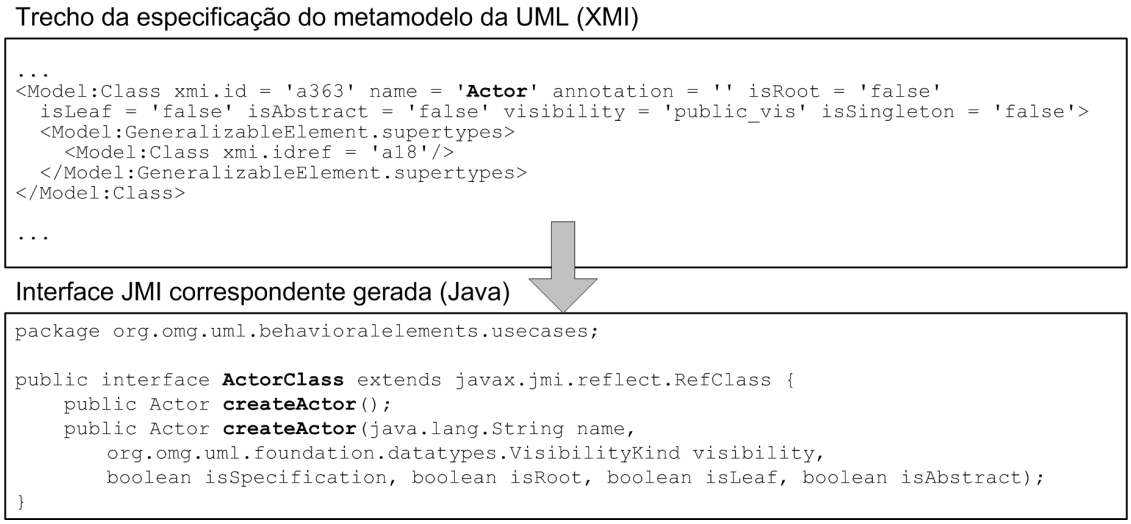

Figura 13: Geração da interface JMI para acesso direto a um ator da UML.

O metamodelo da UML 1.5 utilizado é completo, com todos os elementos da especificação formal conforme definido pelo OMG em (OMG, 2003b). Porém, a UML 1.5 não define como são representadas as informações visuais dos modelos. A MVCASE, como qualquer outra ferramenta de modelagem, precisa armazenar posições de elementos em diagramas, além de outras informações referentes à representação gráfica de diagramas. A UML 2.0 contém a proposta de um padrão para intercâmbio de diagramas (OMG, 2003a). Porém a UML 2.0 ainda não está finalizada.

Por este motivo, decidiu-se utilizar uma versão simplificada deste metamodelo, já que ainda podem ocorrer algumas modificações com o decorrer do tempo, o que exigiria retrabalho no futuro. Vale ressaltar que o metamodelo utilizado é suficiente para os recursos gráficos da MVCASE, e é consideravelmente semelhante à proposta do OMG, visando sua adaptação para conformidade com o futuro padrão, caso este venha a se estabelecer, o que é muito provável.

Dessa forma, tem-se dois metamodelos: o metamodelo não-gráfico da UML 1.5, e um metamodelo gráfico, desenvolvido especialmente para esta pesquisa. No entanto, para utilizar o metamodelo gráfico na MVCASE, teve-se que criar as suas especificações em XMI e, com o auxílio do MDR, gerar as interfaces JMI correspondentes. Dessa forma, o código da MVCASE pode

[3]http://mdr.netbeans.org/metamodels.html

acessar estas interfaces, utilizando o MDR para ler, modificar e escrever informações visuais em documentos XMI.

4.3.2.2 Arquitetura baseada em *plug-ins*

A utilização do MDR proporcionou para a MVCASE, além das funcionalidades de interoperabilidade e intercâmbio de dados almejadas nesta pesquisa, a possibilidade de desenvolvimento de uma arquitetura baseada em *plug-ins*.

Como na MVCASE todo acesso aos dados é realizado via JMI, que é um padrão já consolidado, qualquer desenvolvedor que conheça o padrão pode desenvolver seu código de manipulação de dados e integrá-lo facilmente à MVCASE, sem prejudicar ou modificar as funcionalidades já existentes.

No contexto desta pesquisa, que teve como um dos objetivos a disponibilização da MVCASE como software livre, tal característica é muito importante, já que o processo de software livre tem como princípio o envolvimento de inúmeros desenvolvedores. Em uma arquitetura baseada em *plug-ins* é mais fácil adicionar novas funcionalidades à ferramenta, o que agiliza sua evolução e disseminação na comunidade de desenvolvimento. Esse fenômeno pode ser observado com a plataforma *Eclipse* (ECLIPSE, 2004), que possui inúmeros *plug-ins* disponíveis.

Para dar à MVCASE esta característica, foi desenvolvida uma API de comunicação entre os *plug-ins* e a ferramenta, baseada no modelo de eventos do Java. A Figura 14 mostra como a MVCASE implementa a arquitetura em *plug-ins*.

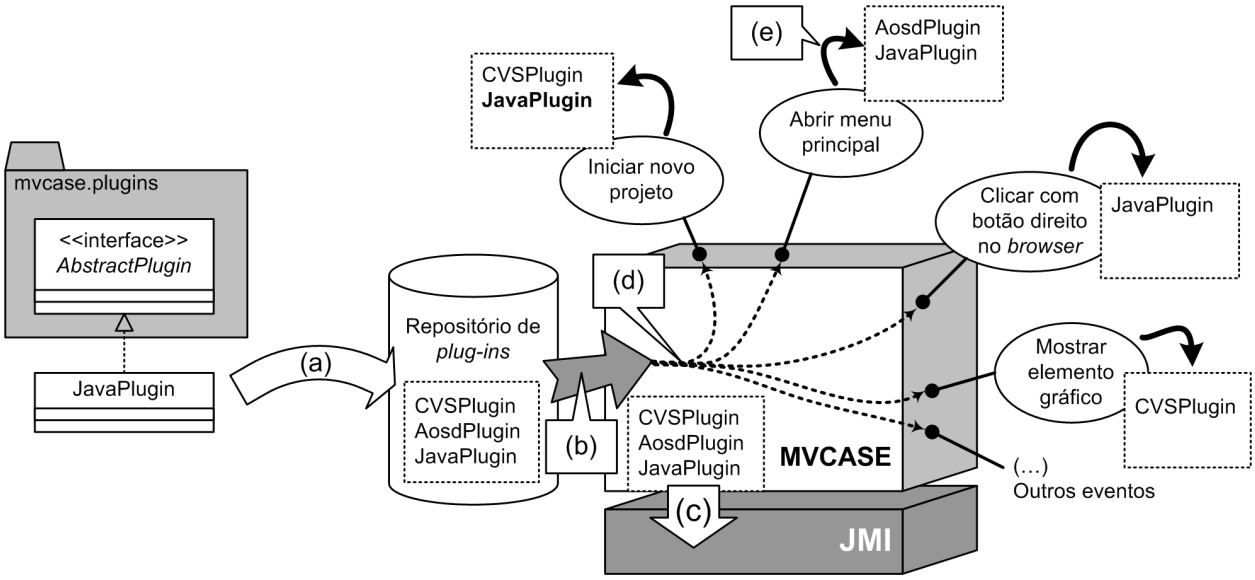

Figura 14: Arquitetura em *plug-ins* da MVCASE.

Todo *plug-in* da MVCASE implementa a interface **AbstractPlugin**. Esta interface é uma

representação genérica de um *plug-in*, e é utilizada pela MVCASE sem que a mesma saiba o que o *plug-in* faz. Para ser instalado na MVCASE, um *plug-in* é simplesmente copiado para um repositório de *plug-ins* (a). Ao ser iniciada, a MVCASE percorre esse repositório e carrega todos os *plug-ins* para a sua instância da máquina virtual Java (b). Estando na mesma máquina virtual, os *plug-ins* passam a ter acesso às mesmas interfaces JMI e aos modelos manipulados pela MVCASE (c).

Ao ser carregado, o *plug-in* se registra em um dos possíveis eventos da MVCASE (d). Ao gerar o evento, a MVCASE então notifica todos os *plug-ins* que estão nele registrados. Por exemplo, quando o usuário clicar sobre o menu principal da ferramenta (e), o *plugin* **JavaPlugin** pode exibir um menu com a opção para gerar código Java, enquanto que o **AosdPlugin** exibe uma opção para gerar código em *AspectJ*. Caso não tenha nenhum *plug-in* registrado, nenhum comportamento adicional ao da ferramenta é executado.

Dessa forma, é possível que a ferramenta execute sua função básica sem que nenhum *plug-in* esteja instalado. A dependência entre *plug-ins* também é possível. Por exemplo, o *plug-in* para *AspectJ* pode utilizar os métodos de geração de código Java do **JavaPlugin**.

Em tese, caso exista um número grande de *plug-ins* no repositório, ocorrerá um decréscimo no desempenho da ferramenta, devido ao fluxo extra de controle necessário para registrar os *plug-ins* e para redirecionar os eventos. Porém, a quantidade de *plug-ins* dificilmente será excessiva, já que o usuário pode escolher apenas os *plug-ins* efetivamente utilizados para instalar na ferramenta. Na prática, este decréscimo de desempenho não foi observado.

Os benefícios da arquitetura em *plug-ins* incluem a alta flexibilidade, manutenibilidade e extensibilidade da ferramenta, pois novas funcionalidades podem ser desenvolvidas totalmente à parte e modificações em um *plug-in* não envolvem modificações na ferramenta. Da mesma forma, modificações na ferramenta não envolvem modificações nos *plug-ins*, a menos que estas modificações envolvam a API relativa ao seu suporte à arquitetura baseada em *plug-ins*.

Após essa alteração, a MVCASE se tornou menor em termos de linhas de código e melhor modularizada. O módulo principal da ferramenta possui somente funcionalidades para ler, manipular e salvar modelos UML, enquanto que outras funcionalidades foram encapsuladas em *plug-ins*. As demais funcionalidades da ferramenta foram encapsuladas em *plug-ins*:

- **Geração de código Java**: *Plug-in* responsável pela geração do código Java relativo aos modelos criados na ferramenta;
- ***Plug-in* para Desenvolvimento de Software Orientado a Aspectos**: *Plug-in* que dá suporte à modelagem de sistemas orientados a aspectos e à geração de código em *AspectJ*;

- **Modelagem de dados no esquema relacional**: *Plug-in* responsável pela modelagem de estruturas de dados e geração de *scripts* SQL para criação de tabelas em Sistemas Gerenciadores de Banco de Dados relacionais; e

- ***Plug-in* EJB**: *Plug-in* de suporte para o desenvolvimento baseado em componentes distribuídos segundo a tecnologia EJB.

As demais alterações realizadas na MVCASE como parte deste trabalho de pesquisa também foram desenvolvidos em forma de *plug-ins*, conforme descrito no restante deste capítulo.

4.3.2.3 Resumo do suporte para XMI na MVCASE

Comparando-se a nova arquitetura da MVCASE com a arquitetura original, pode-se notar que alguns dos problemas foram resolvidos:

- O módulo de modelagem era excessivamente carregado. Agora, como é implementado pelo MDR, ele contém somente a implementação do metamodelo da UML;

- Os módulos eram muito acoplados entre si. Agora, toda dependência entre os módulos é feita através das interfaces JMI, o que resulta em maior flexibilidade e extensibilidade. Novas funcionalidades podem ser inseridas de maneira independente. De fato, a utilização do MDR possibilitou o desenvolvimento de uma arquitetura baseada em *plug-ins*, conforme descrito na seção 4.3.2.2; e

- Apesar de não ser completamente responsável por um suporte mais completo para ambientes distribuídos de desenvolvimento, a utilização do padrão XMI facilita a interoperabilidade e o intercâmbio de dados com outras ferramentas.

Outro ponto que merece ser ressaltado é que esta pesquisa também contribuiu com o projeto do MDR. A integração com a MVCASE revelou um problema com relação à operação de *roll-back* oferecida pelo MDR. O problema foi notificado por este pesquisador, e foi posteriormente corrigido pela equipe do MDR, conforme pode ser constatado na sua *homepage*[4].

4.3.3 Serviço de Armazenamento Remoto

O armazenamento remoto de artefatos envolve a utilização da rede para comunicação com algum tipo de servidor de artefatos. Nesta pesquisa, este servidor é o CVS (*Concurrent Versions*

[4]http://www.netbeans.org/issues/show_bug.cgi?id=47487

System, apresentado no capítulo 2). O motivo pelo qual se decidiu utilizar o CVS nesta pesquisa é principalmente sua popularidade. Trabalhos que utilizam o CVS como base possuem maior chance de serem disseminados na comunidade de desenvolvimento. Outro motivo é a existência de diversas APIs e outros sistemas de suporte ao CVS, o que facilita sua integração com a MVCASE. Além disso, as outras ferramentas que propõem melhorias em relação ao CVS possuem basicamente os mesmos comandos. Sendo assim, uma possível adaptação para essas ferramentas não seria muito trabalhosa.

Para armazenar seus artefatos no CVS, a MVCASE teria de implementar o lado "cliente", conforme o protocolo de comunicação do CVS, ou utilizar um software "cliente" já existente.

Pelo fato de ser amplamente utilizado, existem inúmeros softwares livres ligados ao CVS. Dentre estes, existe um cliente CVS para Java, denominado *javacvs* (NETBEANS, 2005a). A exemplo do MDR, o *javacvs* também é parte do projeto *Netbeans*.

O *javacvs* consiste de um conjunto de classes Java que implementam as principais funcionalidades do CVS, permitindo que programadores desenvolvam código de acesso a repositórios CVS facilmente. Na MVCASE, foi desenvolvido o **CVSPlugin**, um *plug-in* de acesso ao CVS que utiliza o *javacvs*, segundo a arquitetura baseada em *plug-ins* adotada.

O *plug-in* permite que o Engenheiro de Software utilize as principais funções do CVS diretamente na MVCASE, conforme mostra a Figura 15.

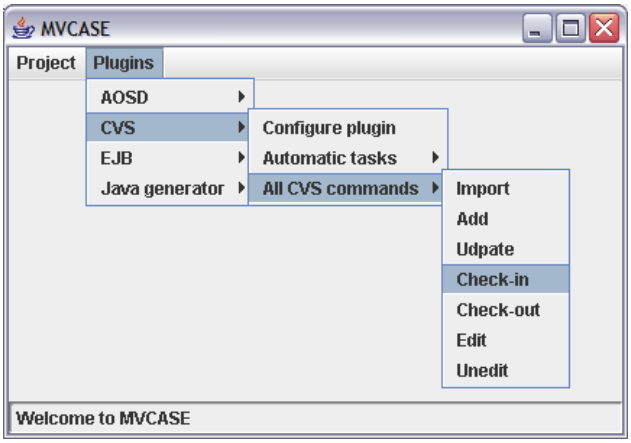

Figura 15: *Plug-in* para acesso ao CVS a partir da MVCASE.

A configuração do repositório CVS, acessado remotamente, é feita em uma interface gráfica (Figura 16) que permite que o Engenheiro de Software insira diferentes informações como o endereço do servidor, o protocolo a ser utilizado, o módulo relacionado, usuário e senha, e a pasta a ser utilizada para armazenar os artefatos localmente.

Com o **CVSPlugin**, os artefatos produzidos na MVCASE não precisam ser armazenados lo-

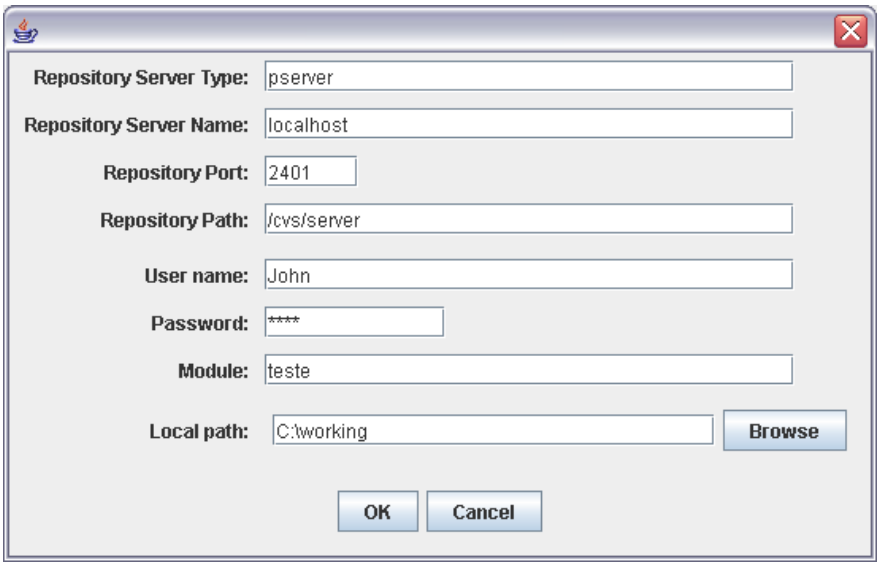

Figura 16: Tela de configuração do CVS na MVCASE.

calmente, mas sim em repositórios remotos. O trabalho de transferência manual dos arquivos não é mais necessário, agilizando o processo de desenvolvimento em equipes.

A Figura 17 ilustra o funcionamento do serviço de armazenamento remoto da MVCASE.

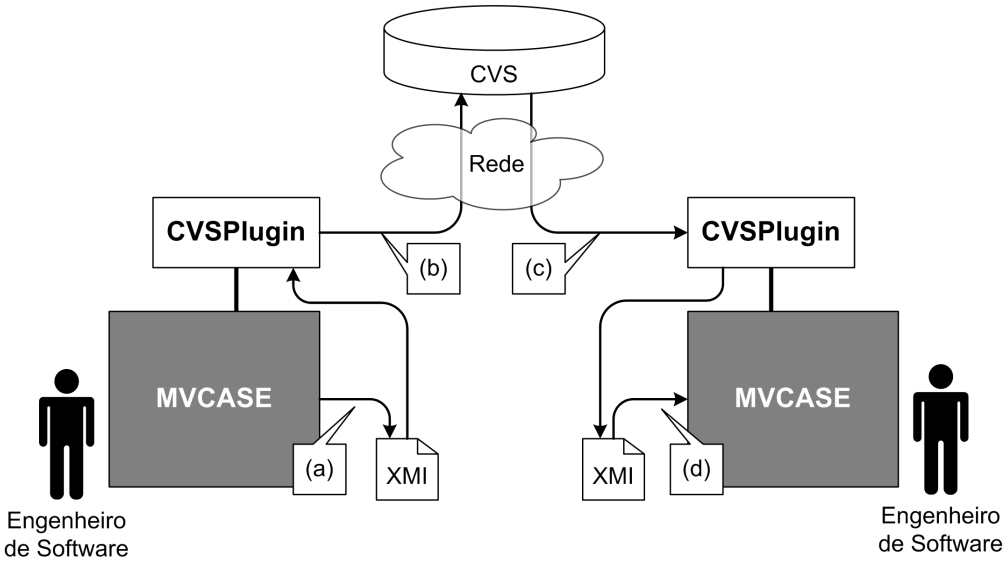

Figura 17: Serviço de armazenamento remoto da MVCASE.

Por exemplo, um Engenheiro de Software trabalha normalmente com a MVCASE, salvando seus modelos em um arquivo XMI (a). Em seguida, ele utiliza o **CVSPlugin** para enviar este arquivo ao CVS (b). Outro Engenheiro de Software, trabalhando em outro local da rede, pode recuperar este arquivo (c) utilizando sua cópia da MVCASE com o **CVSPlugin**. Recuperado o arquivo, ele o abre normalmente na MVCASE (d).

Comparando-se com a arquitetura original da MVCASE, o **CVSPlugin** facilita o trabalho em

equipe, visto que diferentes indivíduos podem trabalhar com um mesmo conjunto de artefatos sem a necessidade de transferência manual dos arquivos. Porém, este compartilhamento de artefatos exige um controle maior sobre as alterações realizadas. Por exemplo, no caso de dois Engenheiros de Software realizarem alterações simultâneas, é necessário um tratamento mais completo. O CVS possui algumas funcionalidades que auxiliam a resolver estes problemas. Sendo assim, o **CVSPlugin** foi modificado para fazer uso destas funcionalidades na MVCASE.

Os principais comandos do CVS são:

- **import**: Utilizado para inserir artefatos no CVS pela primeira vez. Este comando cria um módulo de armazenamento contendo todos os artefatos inseridos;
- **add**: Utilizado para inserir um novo artefato no CVS, quando já existir um módulo criado por um comando **import**;
- **update**: Recupera os artefatos do servidor e os atualiza no cliente;
- **commit** ou **check-in**: Envia os artefatos que estão no cliente para o repositório. Caso haja alguma diferença entre o artefato sendo enviado e o que está armazenado no repositório, é gerada uma nova versão no repositório sendo que a versão antiga permanece armazenada;
- **check-out**: Recupera os artefatos do servidor para o cliente. A diferença entre o **check-out** e o **update** é que o primeiro considera que os artefatos não existem no cliente. Porém, caso os mesmos existam, seu funcionamento é semelhante ao **update**;
- **edit**: Recupera os artefatos do servidor para o cliente, marcando-os para edição; e
- **unedit**: Elimina a marca para edição de um determinado artefato, descartando as alterações.

Esse conjunto de comandos permitem que diferentes versões de artefatos sejam gerenciadas no servidor. A maneira e a sequência com que são utilizados é importante. Por exemplo, para iniciar o uso do CVS é necessário primeiro realizar um **import**, para criar o módulo no servidor, e em seguida realizar um **check-out**, para que a área de trabalho local seja configurada. Outro exemplo é a ocorrência de alterações simultâneas.

Considere a seguinte situação: um usuário “A” realiza um **check-out**, obtendo uma determinada versão do artefato para si. Outro usuário “B” também realiza o mesmo **check-out**, obtendo a mesma versão do artefato. Num determinado momento, “A” finaliza suas alterações e realiza o **commit**. Após este comando, o artefato se encontra em uma versão diferente daquela recuperada por “B”. Por esse motivo, quando “B” finalizar suas alterações e tentar realizar um **commit**, o

comando não será executado. “B” deverá então recuperar a versão atual do artefato, e replicar suas alterações nesta versão. Isso é feito com o comando **update**, que tentará automaticamente combinar as alterações feitas por “B” com as alterações feitas por “A”. Caso as alterações tenham ocorrido em partes diferentes do arquivo, essa operação é finalizada sem problemas, e “B” finaliza o processo com um **commit**, que gera uma versão final do artefato, contendo ambas as alterações realizadas por “A” e “B”.

Porém, pode acontecer de ambos terem alterado a mesma parte do arquivo, gerando um conflito. Neste caso, o comando **update** notifica “B” que há um conflito, e que este deve ser resolvido manualmente: “B” deve então contactar “A” para juntos decidirem sobre a resolução do conflito, trabalhando na MVCASE, ou alterando diretamente o correspondente documento XMI.

Existem outras situações onde os comandos devem ser usados em uma determinada sequência. Para facilitar o processo, o **CVSPlugin** foi modificado para implementar algumas destas tarefas automaticamente, conforme ilustra a Figura 18.

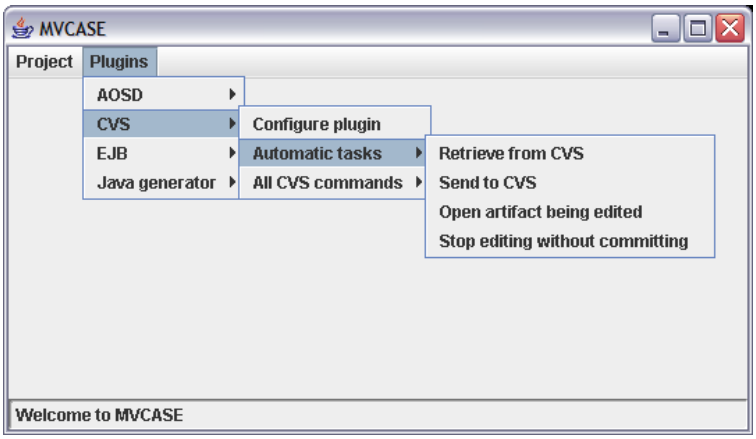

Figura 18: Tarefas automáticas para acesso ao CVS na MVCASE.

Estas tarefas foram desenvolvidas para facilitar a utilização do CVS. Primeiramente, elas eliminam a necessidade de manualmente ler e salvar os modelos para arquivos XMI, como descrito na seção 4.3.3. Elas automaticamente criam um arquivo temporário dentro da estrutura de diretórios da MVCASE para realizar as operações de acesso ao CVS. Além disso, elas implementam algumas sequências de comandos mais comuns quando se utiliza o CVS.

A tarefa **Retrieve from CVS** realiza a recuperação de artefatos do CVS, tanto para edição como visualização. A Figura 19 ilustra o algoritmo implementado por esta tarefa.

Inicialmente, o comando **check-out** é executado. Caso ocorra alguma falha, existem duas possibilidades. Ou ocorreu conflito com a versão atual na máquina cliente, e neste caso é exibida uma mensagem para resolvê-lo manualmente, ou ocorreu outra falha qualquer. Em ambos os casos, a tarefa não é realizada completamente.

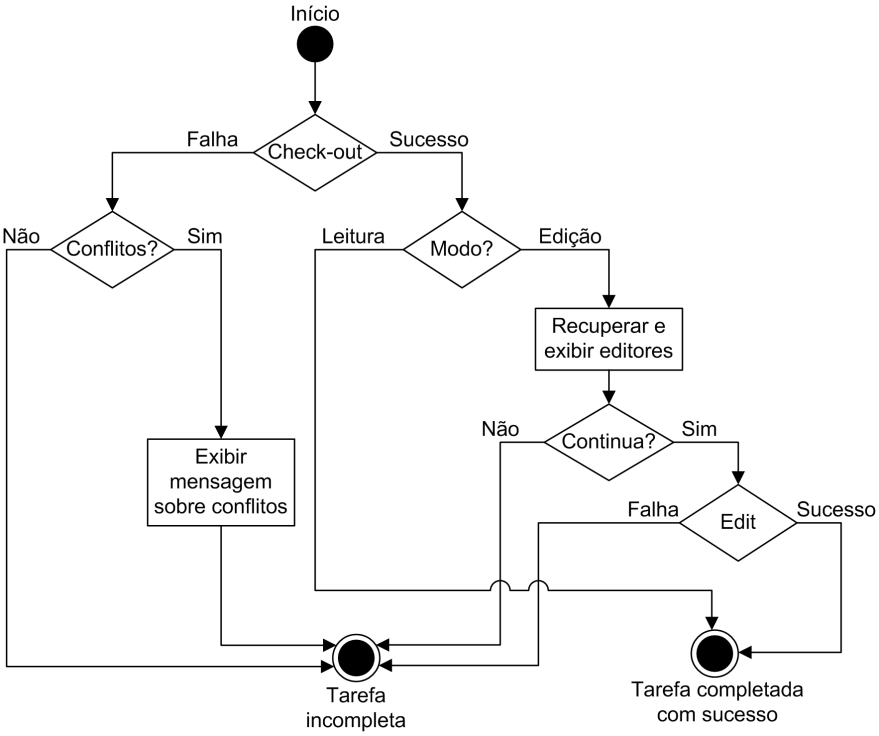

Figura 19: Algoritmo seguido pela tarefa **Retrieve from CVS**.

Caso o comando **check-out** tenha sucesso, verifica-se o modo solicitado. Caso o Engenheiro de Software tenha solicitado o artefato somente para leitura, a tarefa termina com sucesso. Caso ele tenha solicitado o artefato para edição, são exibidos todos os usuários que estão atualmente editando o artefato (Figura 20). O Engenheiro de Software verifica a lista e escolhe se deve continuar ou não a tarefa de edição. Caso positivo, é executado o comando **edit**, que marca o artefato como sendo editado. O Engenheiro de Software é então incluído na lista dos editores, e a tarefa é concluída.

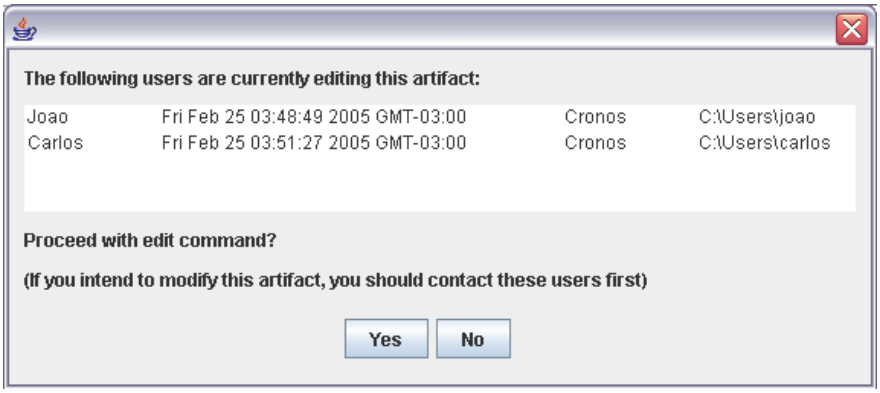

Figura 20: Tela que exibe todos os usuários que estão atualmente editando o artefato.

A tarefa **Send to CVS** realiza o envio dos artefatos ao CVS. A Figura 21 ilustra o algoritmo implementado por esta tarefa.

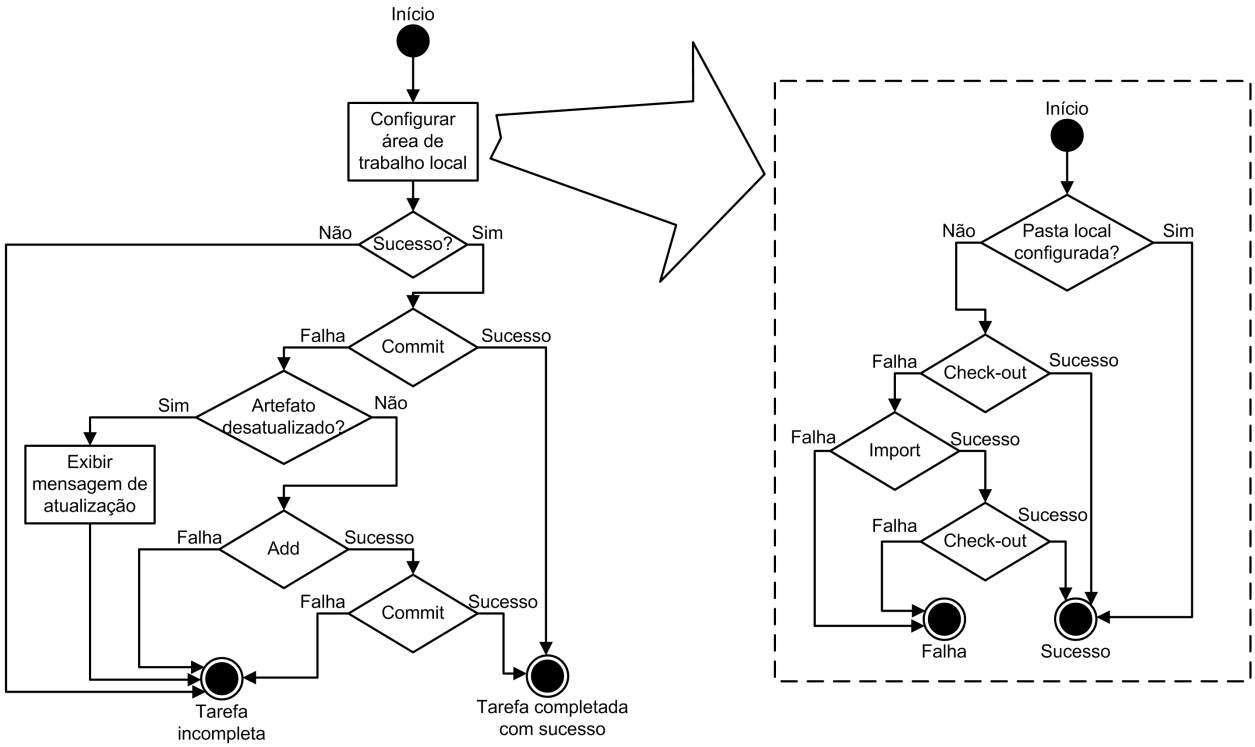

Figura 21: Algoritmo seguido pela tarefa **Send to CVS**.

Inicialmente, é necessária a configuração da área de trabalho local, pois os comandos do CVS só funcionam quando a área de trabalho local está configurada para trabalhar com o CVS. A configuração é realizada da seguinte maneira: Primeiro verifica-se se a pasta local já está configurada. Caso positivo, não é necessário nenhum procedimento extra. Caso contrário, tenta-se realizar um **check-out**. Caso este comando seja bem sucedido, significa que a área local foi corretamente configurada. Caso contrário, significa que não existe nenhum módulo no servidor para fazer o **check-out**. Então, é necessário fazer primeiro um **import**, e depois um **check-out**. Caso estas operações sejam bem sucedidas, a área local foi corretamente configurada. Caso contrário, a operação de configuração da área de trabalho local falhou por algum outro motivo.

Uma vez configurada a área de trabalho local, a tarefa **Send to CVS** tenta realizar um **commit**. Caso o **commit** tenha sucesso, a tarefa termina corretamente. Caso contrário, existem dois motivos pelos quais este comando pode falhar. O primeiro é que a versão do artefato que está no CVS é mais recente do que a que está na área de trabalho local. Neste caso, uma mensagem é exibida ao Engenheiro de Software, para que o mesmo tente executar a operação **Retrieve from CVS** para atualizar a área de trabalho local, e a tarefa é terminada. Caso não seja este o problema, o **commit** pode ter falhado porque o artefato ainda não existe no CVS. Neste caso, tenta-se uma operação **add** seguida por um **commit**. Caso estas operações sejam bem sucedidas, a tarefa termina com sucesso. Caso contrário, a tarefa termina incompleta, por algum outro motivo.

A tarefa **Open artifact being edited** simplesmente abre para edição o último artefato que foi recuperado do CVS. Esta tarefa é necessária porque nem sempre o Engenheiro de Software irá iniciar e completar um processo de modificação de uma única vez. A tarefa **Stop editing without committing** simplesmente executa o comando **unedit**, indicando que o artefato em questão não está mais sendo alterado, descartando as alterações.

Mesmo com a implementação destas tarefas automatizadas, um outro problema surgiu da utilização do CVS em conjunto com a MVCASE. O padrão XMI, utilizado pela MVCASE para persistência dos modelos, contém um identificador único para cada elemento. Da maneira como foi implementado pelo MDR, este identificador é sequencial e único somente dentro do arquivo XMI. Isso quer dizer que, caso duas pessoas, em estações de trabalho diferentes, criem elementos diferentes, pode acontecer de serem atribuídos a estes elementos o mesmo identificador. Quando o CVS realizar a combinação de duas versões do artefato, pode ocorrer uma duplicação do identificador, resultando em erro.

Outro problema é que este identificador é descartado na leitura do XMI, e gerado novamente durante a escrita. Isto significa que, a cada edição na MVCASE, os elementos do XMI poderão possuir identificadores diferentes, não importa se foram ou não modificados. Neste caso, torna-se impossível para o CVS realizar o processo de combinação, pois sempre ocorrerá um conflito, mesmo que as modificações tenham ocorrido em partes diferentes do modelo. A Figura 22 ilustra esses problemas.

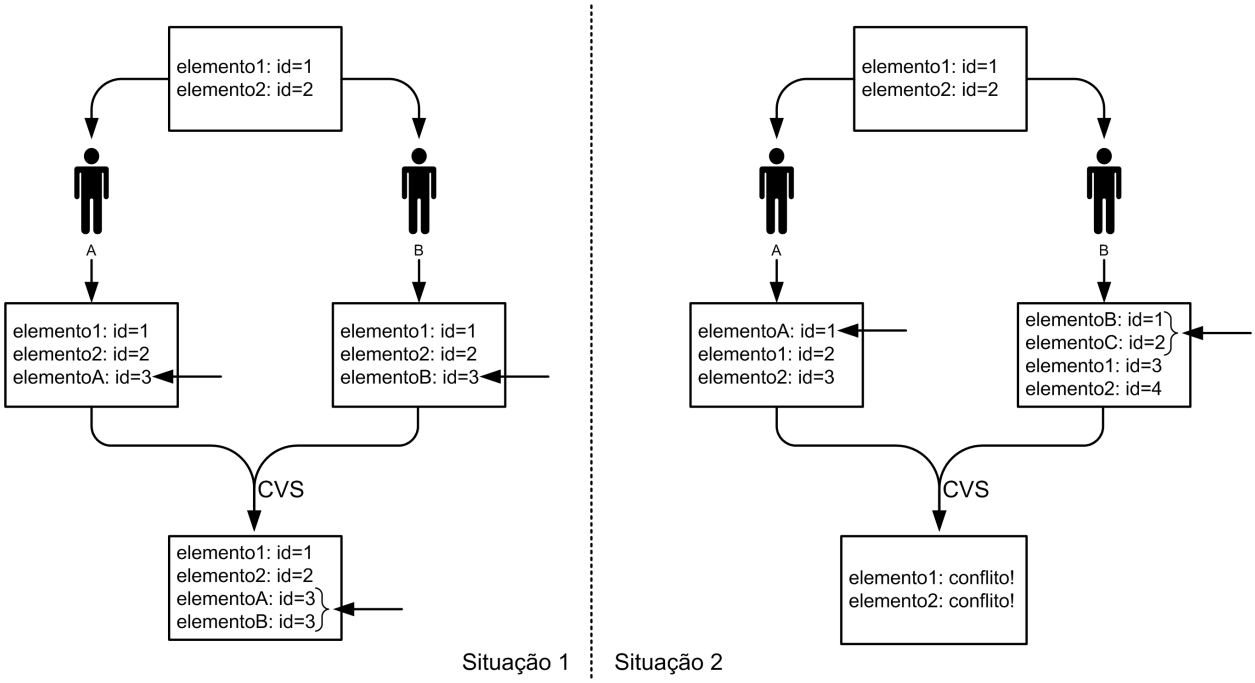

Figura 22: Problemas causados pelo identificador único do XMI.

Na situação 1, um mesmo modelo é alterado por dois Engenheiros de Software diferentes. O

Engenheiro de Software A adiciona um novo elemento **elementoA**, enquanto que o Engenheiro de Software B adiciona um novo elemento **elementoB**. Ao salvar em XMI, tanto **elementoA** como **elementoB** irão possuir um mesmo identificador, de forma que ao combinar as duas versões, o CVS irá produzir um documento XMI inválido, onde dois elementos possuem o mesmo identificador.

Na situação 2, um novo elemento **elementoA** é inserido pelo Engenheiro de Software A, e dois novos elementos **elementoB** e **elementoC** pelo Engenheiro de Software B. Ao realizar a combinação entre os dois arquivos XMI, o CVS irá detectar conflitos nos elementos **elemento1** e **elemento2**, sendo que esse conflito não existe, pois nenhum dos dois elementos foi modificado.

Para resolver estes problemas, três ações foram tomadas:

- Cada elemento do XMI corresponde a um objeto da MVCASE. Portanto, a cada instanciação de um objeto na MVCASE é atribuído um identificador global único, composto do endereço da máquina na rede, a hora local, com precisão de milissegundos, e um número aleatório. Dessa forma, é virtualmente impossível que dois objetos, e por consequência dois elementos do XMI, tenham um mesmo identificador;
- Na geração do XMI, utiliza-se este identificador ao invés do identificador padrão sequencial gerado pelo MDR; e
- Uma tabela intermediária é utilizada para armazenar os identificadores lidos, uma vez que os mesmos são descartados logo após a leitura.

Dessa maneira, todos os elementos possuem um identificador que é globalmente único, ao invés de ser único somente dentro do XMI. Além disso, esse identificador não é transiente, permanecendo o mesmo após sucessivas leituras e escritas em XMI. Dessa forma, caso um Engenheiro de Software altere somente parte de um modelo, o correspondente arquivo XMI será alterado somente no trecho correspondente. O efeito disso sobre o controle de versões é que os conflitos causados desnecessariamente pela utilização de um identificador não único não irão mais ocorrer.

Porém, mesmo com estas alterações, ainda podem existir situações problemáticas com relação ao uso do XMI em conjunto com o CVS. Por exemplo, outras ferramentas de modelagem que não tenham realizado esse tratamento sobre o identificador único sobre o XMI podem enfrentar este mesmo problema, prejudicando a interoperabilidade com a MVCASE. Além disso, existem algumas situações onde a abordagem adotada não é capaz de detectar conflitos existentes. Por exemplo, considere um modelo de classes sendo alterado simultaneamente por dois Engenheiros de Software. Um primeiro Engenheiro de Software remove uma determinada classe **X**, enquanto que o segundo Engenheiro de Software adiciona um relacionamento com essa classe **X**. Nenhum

conflito é detectado pelo CVS, apesar de o mesmo existir. Como neste trabalho o foco não foi especificamente o controle de versões, não aprofundou-se neste assunto, que exigiria um tratamento mais completo e dedicado.

4.3.3.1 Resumo do serviço de armazenamento remoto

Com o **CVSPlugin**, o controle de versões é facilitado porque faz uso do CVS, além de estender sua capacidade. Os Engenheiros de Software podem trabalhar simultaneamente sobre um mesmo conjunto de artefatos, sob um certo controle das alterações realizadas.

Um tratamento mais completo, não abordado nesta primeira versão, é necessário, já que o controle de versões é apenas parte da área da Engenharia de Software conhecida como Gerenciamento de Configuração, responsável pelo controle da evolução do software. Um trabalho semelhante ao da MVCASE e que dá um tratamento mais específico ao Gerenciamento de Configuração pode ser visto em (CUNHA, 2005).

Quanto ao trabalho cooperativo, esta primeira versão já oferece importantes contribuições. Comparando-se com a arquitetura original da MVCASE, pode-se notar as vantagens do uso do **CVSPlugin** em trabalhos em equipe. Porém, o trabalho cooperativo envolve muitas outras questões, como por exemplo a troca de mensagens entre indivíduos, e o conhecimento mútuo sobre o trabalho sendo realizado. Novas versões em trabalhos futuros poderão estender a ferramenta com tais funcionalidades.

A reutilização com o **CVSPlugin** é facilitada, visto que o Engenheiro de Software pode automaticamente recuperar artefatos remotamente armazenados, bastando fornecer o seu caminho. Porém, encontrar esse caminho pode não ser simples. Por essa razão, foi desenvolvido um serviço de busca, como parte da extensão da MVCASE.

4.3.4 Serviço de Busca

A última alteração efetuada na MVCASE foi o desenvolvimento de um serviço de busca. Uma vez que o repositório esteja populado, é desejável que se tenha um mecanismo de busca para facilitar a reutilização dos artefatos nele armazenados. Conforme discutido no Capítulo 2, uma busca eficiente pode significar o sucesso da reutilização, aumentando a produtividade e reduzindo custos de desenvolvimento.

Como parte desta pesquisa, foram realizados estudos sobre mecanismos de busca, conforme apresentamos em (LUCRÉDIO; ALMEIDA; PRADO, 2004). Com base nestes estudos, foi desenvolvido um protótipo baseado na estrutura de facetas, inicialmente apresentado em (LUCRÉDIO et al.,

2003a), e posteriormente em (LUCRÉDIO et al., 2004).

Dos resultados destes estudos, partiu-se para a implementação de um mecanismo de busca baseado em facetas. Decidiu-se utilizar esta estrutura pelo fato de a mesma ser intuitiva, adequada ao domínio de artefatos de software, de fácil manutenção e uso, e passível de automação (PRIETO-DÍAZ, 1991). Também satisfaz a maioria dos requisitos apresentados na seção 4.1, conforme detalhado mais adiante neste capítulo. Além disso, inúmeros trabalhos a utilizam, o que comprova sua praticidade e aplicabilidade na área da reutilização.

Para facilitar o entendimento da organização em facetas, considere um artefato denominado **Conta bancária**. Esse artefato pertence ao domínio financeiro, foi desenvolvido em EJB e contém funcionalidades relativas ao armazenamento em banco e mineração de dados. Uma possível estrutura hierárquica de classificação para este artefato é apresentada na Figura 23.

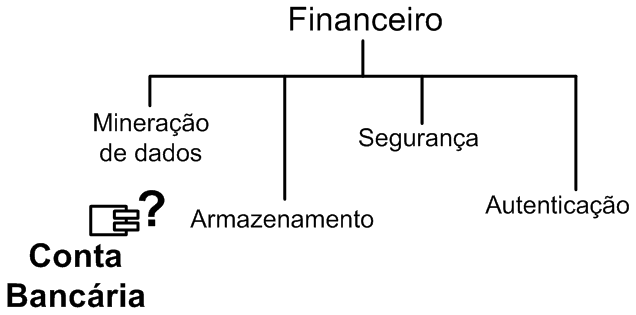

Figura 23: Organização hierárquica.

Como pode ser observado, em uma organização hierárquica podem ocorrer situações de ambigüidade. No exemplo da Figura 23, o artefato **Conta Bancária** poderia ser inserido tanto no nó **Mineração de dados** como **Armazenamento**.

Diferente da organização hierárquica, a organização em facetas permite que um determinado artefato seja classificado em mais de um lugar da estrutura. A Figura 24 ilustra como seria classificado o artefato **Conta Bancária** em um esquema em facetas.

Ao invés de ser associado a um único nó de uma árvore, um artefato pode possuir diferentes características ao mesmo tempo. No exemplo da Figura 24, é utilizada uma estrutura com quatro facetas: **Domínio**, **Funcionalidade**, **Modelo de componente** e **Tipo**. Para classificar um artefato, associa-se então um ou mais possíveis valores para cada faceta. No caso, escolheu-se o valor **Financeiro** para a faceta **Domínio**, os valores **Mineração de dados** e **Armazenamento** para a faceta **Funcionalidade**, e assim por diante. Dessa forma, reduz-se a ambigüidade, oferecendo maior poder descritivo para classificar os artefatos reutilizáveis.

A busca utilizando a organização em facetas é realizada escolhendo-se valores possíveis para

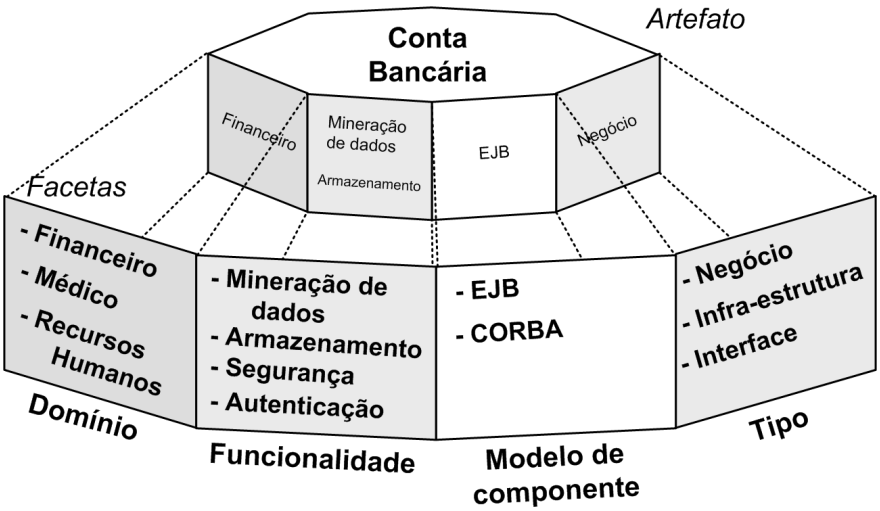

Figura 24: Organização em facetas.

as facetas, ou especificando uma expressão de consulta envolvendo os valores. Também é possível navegar na estrutura, através de técnicas de hipertexto. Ambas as possibilidades foram utilizadas neste trabalho de pesquisa, conforme apresentado mais adiante nesta seção.

Decidiu-se implementar o serviço de busca em uma interface *Web*, ao invés de implementá-lo na MVCASE, pois a reutilização não deve se restringir aos artefatos produzidos pela ferramenta. Dessa forma, é possível recuperar outros artefatos, como código-fonte em java, por exemplo, utilizando um *Web browser* qualquer. Ao encontrar um determinado artefato, é mostrada ao reutilizador uma *string* que descreve o caminho para o artefato, de modo que qualquer cliente CVS pode ser usado para recuperar este artefato do seu respectivo repositório.

Porém, para facilitar seu uso e evitar a necessidade de se interpretar manualmente essa *string* com o caminho até o artefato, o serviço de busca foi integrado à MVCASE através de um *plug-in* que chama automaticamente a página *Web* do serviço de busca, dentro da própria interface visual da MVCASE. A Figura 25 mostra uma tela deste *plug-in*.

O *plug-in* de acesso ao serviço de busca, além de mostrar a sua interface *Web* e permitir sua navegação sem a necessidade de se utilizar um *Web browser* externo à MVCASE, detecta quando o reutilizador clica sobre um artefato que deseja recuperar do repositório CVS. Na ocorrência deste evento, o *plug-in* de busca envia uma mensagem ao *plug-in* de acesso ao CVS, para que este último recupere automaticamente o artefato selecionado.

A Figura 26 mostra como foi implementada a arquitetura do serviço de busca utilizando a organização em facetas.

O serviço pode percorrer vários repositórios CVS em busca de artefatos. Existe um indexador manual, que consiste em uma interface gráfica onde o Engenheiro de Software pode cadastrar

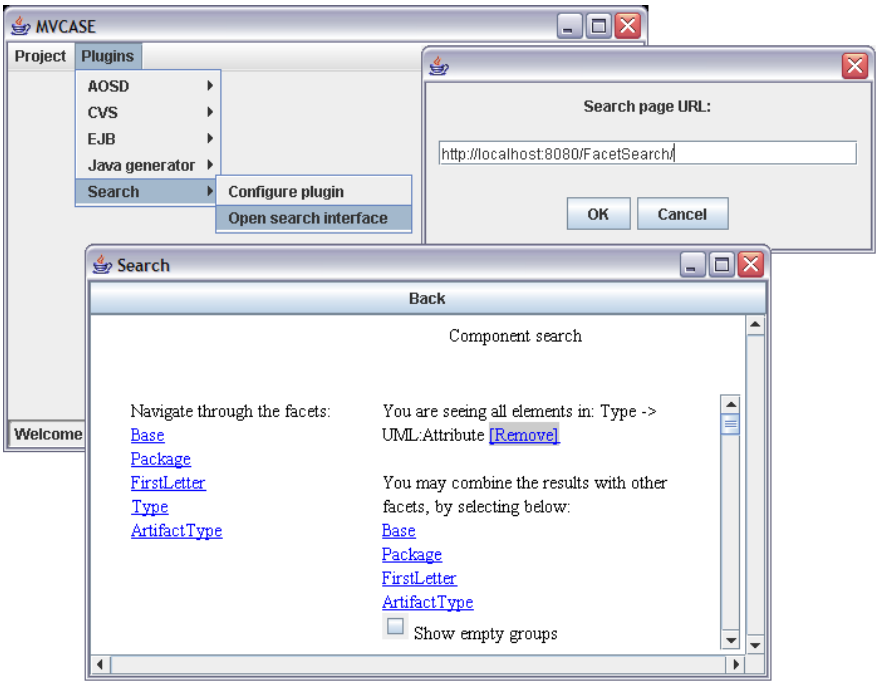

Figura 25: *Plug-in* da MVCASE para acesso ao serviço de busca.

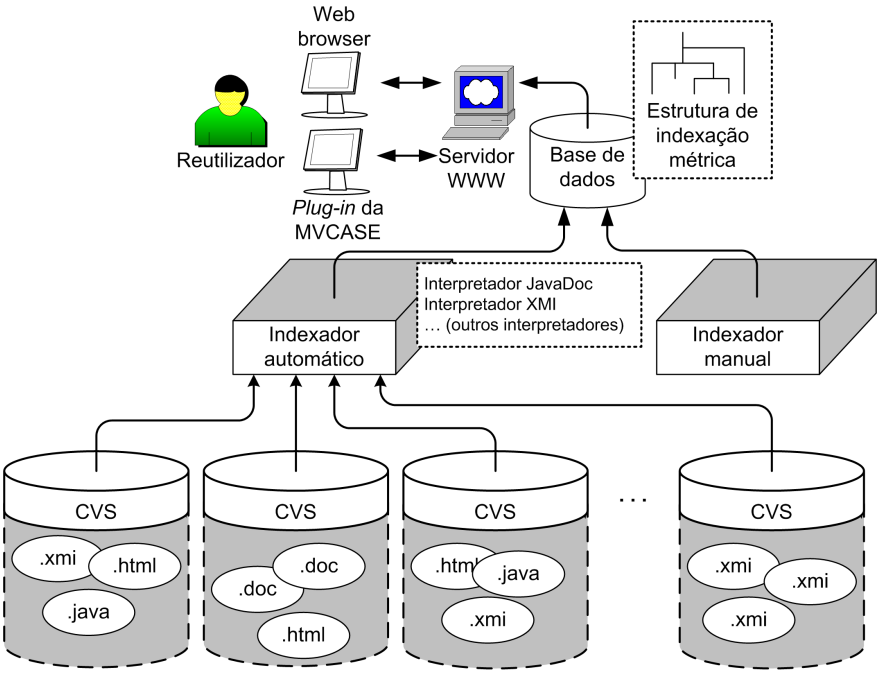

Figura 26: Serviço de busca.

manualmente as informações sobre as facetas, armazenando-as diretamente no banco de dados. A Figura 27 mostra o modelo de dados utilizado para armazenar estas informações.

Uma faceta (*Facet*) possui um nome (*name*), e um conjunto de possíveis valores (*FacetValue*) para aquela faceta. Um valor para uma faceta é descrito por uma sequência de caracteres (*value*), e está associado a exatamente uma faceta. Um artefato (*Artifact*) possui uma *URL*, que é um caminho

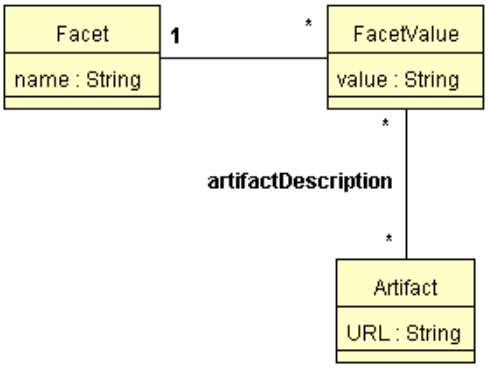

Figura 27: Modelo de dados utilizado para armazenar as informações sobre as facetas no Banco de Dados.

que identifica onde o artefato está armazenado. A descrição do artefato consiste na associação entre artefatos e conjuntos de valores para facetas. Para esta pesquisa, a base de dados foi implementada no Sistema Gerenciador de Banco de Dados PostgreSQL[5].

Porém, na maioria dos casos, a classificação manual exige muito esforço. Portanto, foi construído um indexador automático, responsável por interpretar automaticamente as informações contidas nos artefatos e popular essa base de dados.

A interpretação dos artefatos é feita de acordo com o tipo do artefato. Para esta pesquisa, foram implementados dois interpretadores: um interpretador simples de documentação Java, do tipo *javadoc*, e um interpretador XMI. Porém, vale ressaltar que outros tipos de interpretadores poderiam fazer uso da estrutura em facetas.

O interpretador *javadoc* percorre a documentação Java em busca de três facetas, obtidas diretamente do arquivo de documentação: O pacote que contém o artefato java, A primeira letra do nome do artefato, e o tipo do artefato (*classe*, *interface*, *enumeration*, etc.).

O interpretador XMI utiliza o MDR para percorrer arquivos XMI, em busca das mesmas três facetas: O pacote que contem o artefato, a primeira letra do nome do artefato, e o tipo do artefato (no caso, pode ser qualquer tipo da UML, como classe, ator, caso de uso, componente, etc.).

Uma quarta faceta é inserida para indicar o tipo do interpretador utilizado, no caso Javadoc ou XMI. E uma quinta faceta para indicar em qual repositório se encontra o artefato. A cada artefato é associado um localizador, que consiste em um caminho para realizar o *check-out* do artefato do CVS. O Engenheiro de Software utiliza então este caminho para recuperar o artefato.

O conjunto de facetas utilizado foi escolhido com base na facilidade para extração automática de informação. Porém, este conjunto pode ser estendido para incluir outras qualidades de um

[5]http://www.postgresql.org/

artefato. Poderiam ser incluídas facetas para classificar, por exemplo, artefatos de acordo com seu tamanho, seu autor ou data de sua última modificação. Outras informações contidas no XMI, como por exemplo hierarquias de classe, relacionamentos entre elementos, ou até especificações formais sobre os comportamentos dos métodos, também são bons candidatos para o esquema em facetas.

A apresentação para o reutilizador é feita através de uma interface *Web*, que pode ser visualizada em qualquer *Web browser* ou através do *plug-in* da MVCASE para busca. A vantagem de se utilizar o *plug-in* é que o mesmo detecta quando o reutilizador seleciona um artefato e automaticamente o recupera do repositório.

Uma estrutura de indexação métrica é utilizada para realizar a busca por similaridade, conforme apresentamos em (LUCRÉDIO et al., 2004). Foi definida uma métrica de semelhança entre artefatos, que considera o quão similares são seus conjuntos de valores para cada faceta. Assim é possível recuperar não somente os artefatos que atendem exatamente à expressão de consulta, mas também artefatos semelhantes, aumentando a chance de reutilização.

A métrica definida foi a *K-Metric*, que mede o quão similares são os valores das facetas de diferentes componentes, da seguinte maneira:

Conforme a Figura 28, considere um conjunto finito de artefatos, denominado *R*, um conjunto *F*, contendo *n* facetas (f_1, f_2, ... f_n), e *n* conjuntos finitos (P_1, P_2, ... P_n) de possíveis valores (palavras-chave) para cada faceta. Cada conjunto P_i de possíveis valores pode ter de 0 a *X* elementos, onde *X* (representado na Figura por *A*, *B*, *C*, ... *Z*) é o número máximo de valores que a faceta f_i pode ter. A cada artefato *a* contido em *R* é associado um descritor D_a, composto de *n* conjuntos de valores p_i, um para cada faceta, onde p_i é um subconjunto de P_i.

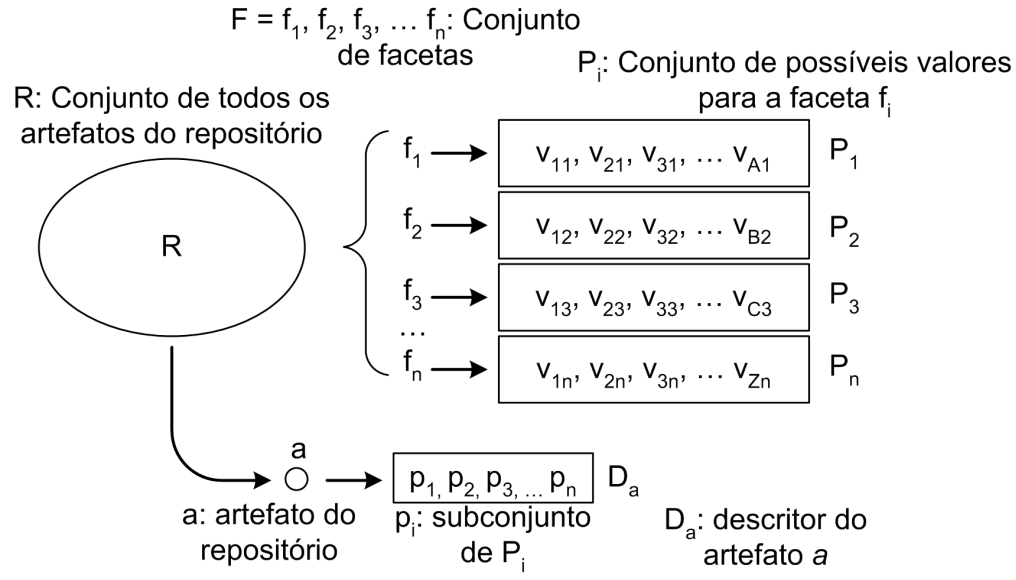

Figura 28: Estrutura de classificação em facetas.

A *K-Metric* calcula, como um valor de dissimilaridade entre dois artefatos A1 e A2, o número de inserções e remoções de valores que são necessários para fazer com que os descritores de A1 e A2 se tornem iguais (uma substituição conta como uma remoção seguida de uma inserção). A similaridade entre dois artefatos é inversamente proporcional ao valor obtido.

Por exemplo, considere três componentes da API Java: *Button*, *Menu* e *CardLayout*. *Button* é um componente para criar botões para interface gráfica com o usuário, que pode ser clicado e produzir eventos. *Menu* gerencia menus gráficos, e o usuário pode selecionar itens deste menu e produzir eventos. *CardLayout* gerencia a localização visual dos elementos em janelas da interface com o usuário.

Considere que estes componentes foram classificados em um esquema com três facetas: *tipo de dados*, *tipo do componente*, e *ação desempenhada*, com os seguintes conjuntos de valores associados a estas facetas:

Button:
1. Tipo de dados = {`Gráfico, Botão, Evento`}
2. Tipo do componente = {`Visual`}
3. Ação desempenhada = {`Criar, Enviar`}

Menu:
1. Tipo de dados = {`Gráfico, Menu, Evento`}
2. Tipo do componente = {`Visual`}
3. Ação desempenhada = {`Criar, Gerenciar, Enviar`}

CardLayout:
1. Tipo de dados = {`Layout, Gráfico`}
2. Tipo do componente = {`Visual`}
3. Ação desempenhada = {`Gerenciar`}

A distância entre os componentes *Button* e *Menu*, denominada *d(Button, Menu)*, é calculada da seguinte forma: Primeiro escolhe-se qualquer um dos componentes para realizar operações de inserção e remoção de valores. Escolhendo-se o componente *Menu*, por exemplo, para fazer com que seu conjunto de valores associados seja igual ao conjunto de valores associados ao componente *Button*, é necessário remover o valor *Menu* da faceta *Tipo de dados* (1 operação de remoção), remover o valor *Gerenciar* da faceta *Ação desempenhada* (1 operação de remoção), e adicionar o valor *Botão* à faceta *Tipo de dados* (1 operação de inserção). No total, foram necessárias três

operações para igualar os conjuntos de valores de cada componente, e portanto *d(Button, Menu)* = 3. Realizando o mesmo processo para os componentes *Button* e *CardLayout*, tem-se que *d(Button, CardLayout)* = 6.

Esse resultado condiz com a realidade, pois em uma primeira impressão, um programador Java instintivamente identifica que o componente *Button* é mais similar ao componente *Menu* (distância = 3) do que ao componente *CardLayout* (distância = 6).

Esse valor é então utilizado como entrada em uma estrutura de indexação métrica. No caso a estrutura utilizada foi a árvore HCS (YAMAMOTO; BIAJIZ; TRAINA, 2003), que permite realizar consultas por abrangência ou consulta aos *k* vizinhos mais próximos. Na primeira, dado um artefato de exemplo, são recuperados todos os artefatos cuja distância ao exemplo é menor ou igual a um valor determinado. Na consulta por vizinhança, dado um artefato de exemplo, são recuperados os *k* artefatos mais próximos, sendo *k* um valor arbitrário.

Para mais informações sobre a busca por similaridade que foi implementada, consulte (LUCRÉDIO et al., 2004).

Além da busca por similaridade, é possível percorrer a estrutura em facetas utilizando navegação sobre hipertexto ou elaboração de consultas. A Figura 29 mostra a interface *Web* construída para navegação sobre as facetas, sendo visualizada no *plug-in* da MVCASE.

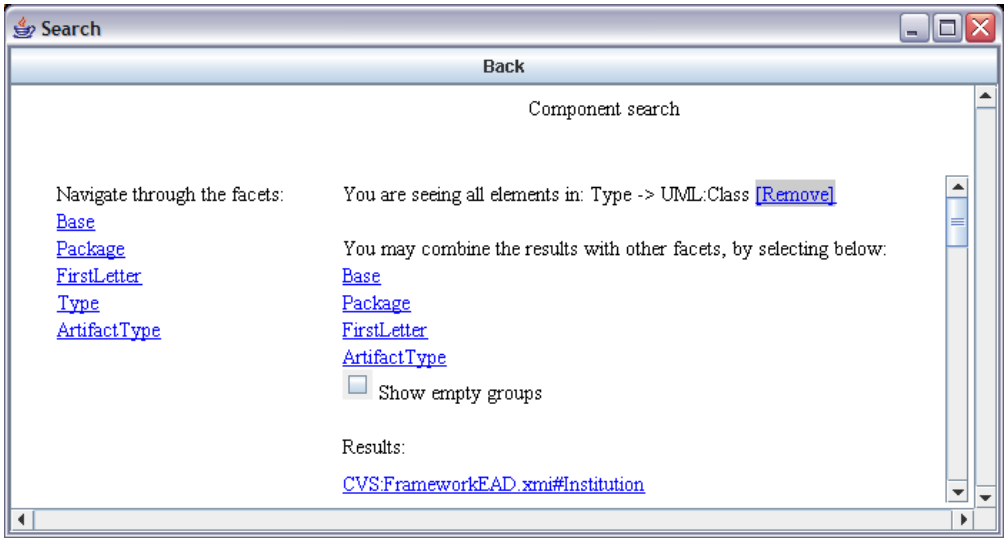

Figura 29: Interface do serviço de busca por navegação, sendo visualizada no *plug-in* da MVCASE.

No lado esquerdo, estão todas as facetas disponíveis. Ao clicar sobre uma delas, são mostrados ao Engenheiro de Software todos os possíveis valores para a faceta. No exemplo da Figura 29, foi selecionada a faceta **Type**, e o valor **UML:Class**. São então mostrados todos os artefatos do tipo **UML:Class** encontrados. É possível combinar estes resultados com outras facetas, selecionando-as conforme mostra a parte central da Figura 29. Pode-se, por exemplo, mostrar todos os artefatos

do tipo **UML:Class**, que comecem com a letra **“F”**.

Também é possível, além de realizar a busca por navegação, recuperar artefatos através da elaboração de uma consulta. O Engenheiro de Software utiliza a interface mostrada na Figura 30, e escolhe possíveis valores para as facetas. São então mostrados todos os artefatos classificados nos valores escolhidos para as facetas.

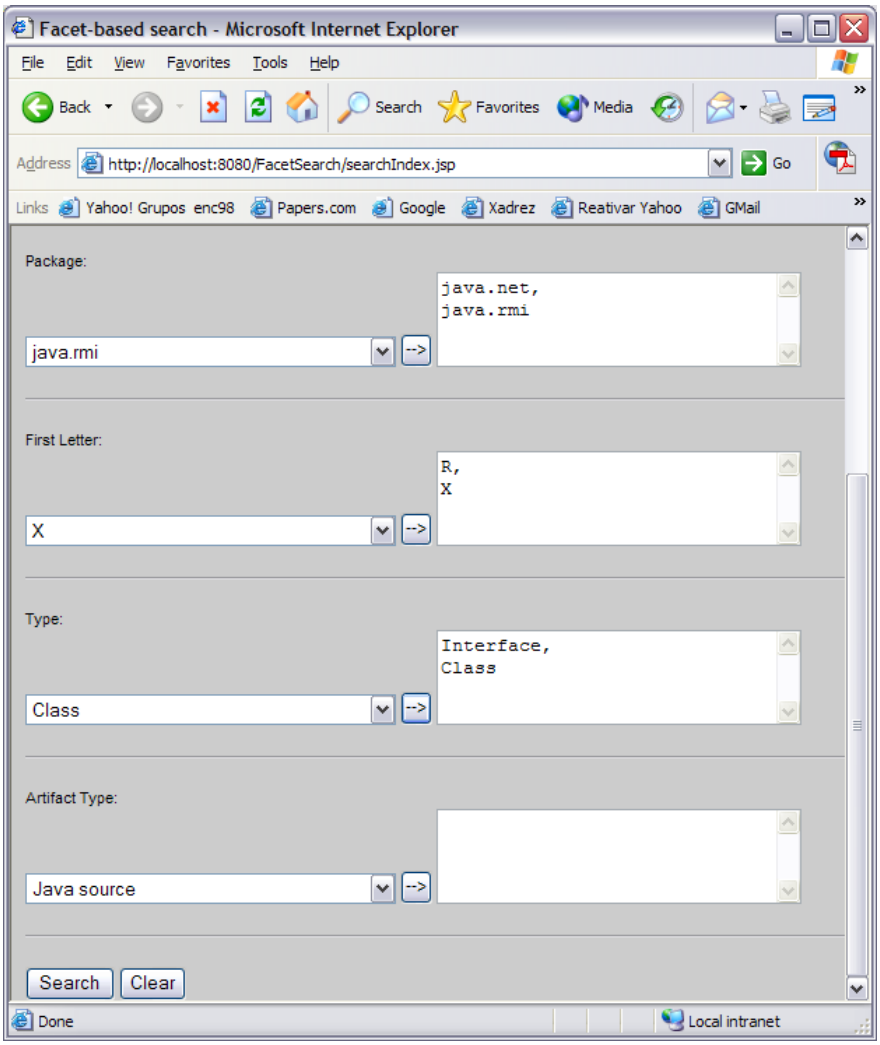

Figura 30: Interface do serviço de busca através de consulta, sendo visualizada no *Internet Explorer 6.0*.

4.3.4.1 Resumo do serviço de busca

Com o serviço de busca, é possível varrer grandes repositórios e encontrar artefatos sem a necessidade de uma busca exaustiva, o que poupa o trabalho do reutilizador. Nesta pesquisa, foram construídos apenas dois indexadores automáticos, um para recuperar informações a partir de documentação java (*javadoc*), e outro para extrair informações a partir de documentos XMI. Porém, a arquitetura utilizada é extensível, e qualquer outro tipo de indexador poderia ser construído como,

por exemplo, extratores de texto automáticos.

Com o *plug-in* da MVCASE, a interface do serviço de busca fica integrada à MVCASE. O reutilizador pode simplesmente utilizar o serviço através de um menu da ferramenta, e ao clicar sobre um artefato, o mesmo é automaticamente recuperado do seu respectivo local.

4.3.5 Integração da MVCASE com o ambiente *Orion*

Após a extensão da MVCASE, principalmente devido ao suporte para o XMI, é possível realizar a sua integração com outras ferramentas. Por exemplo, é possível utilizar a MVCASE em conjunto com a ferramenta *Poseidon*, que também utiliza o XMI como meio de persistência.

No GOES - Grupo de Engenharia de Software do Departamento de Computação da UFSCar - Universidade Federal de São Carlos, onde foi desenvolvida esta pesquisa, existem outras duas ferramentas de auxílio ao desenvolvimento de software, denominadas *SoCManager* e *C-CORE*.

A ferramenta *SoCManager* (CUNHA, 2005) auxilia no gerenciamento de projetos, atividades, prazos de execução, equipes de desenvolvimento, e no gerenciamento de configuração. A ferramenta *C-CORE* (NETO et al., 2004) auxilia nas atividades de codificação, depuração e projeto de interface gráfica com o usuário. Juntas, essas duas ferramentas e a MVCASE formam o ambiente *Orion*, ilustrado na Figura 31.

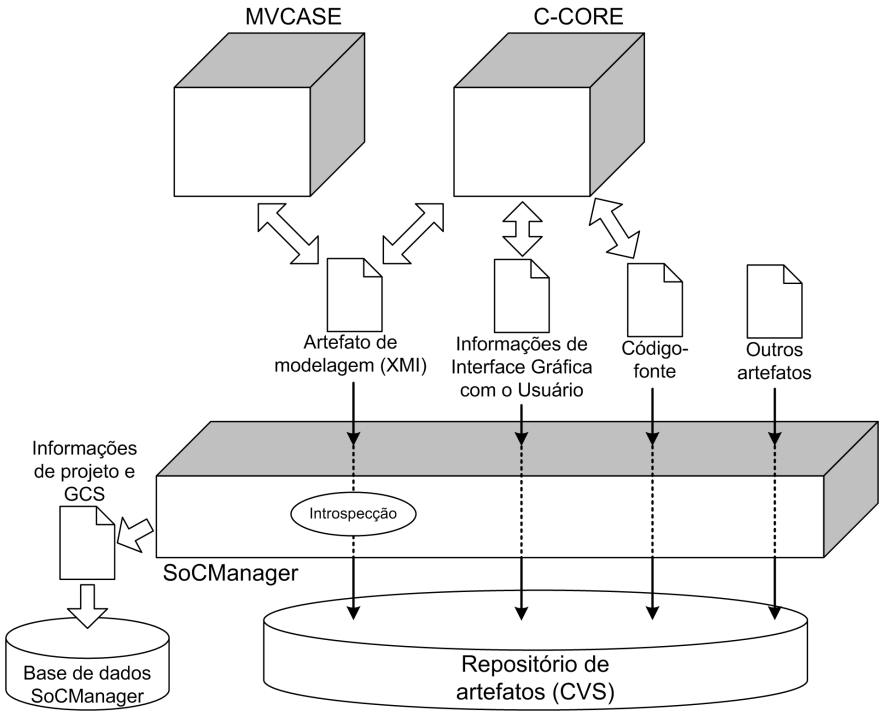

Figura 31: Ambiente *Orion*.

A ferramenta MVCASE produz artefatos de modelagem, persistidos em XMI. A ferramenta

C-CORE tem foco na implementação, e trabalha principalmente com código-fonte e informações relativas à interface gráfica com o usuário, mas também é capaz de persistir as informações contidas no código-fonte em XMI, como estruturas de classes, hierarquias, atributos, métodos e até mesmo o corpo dos métodos. Dessa forma, artefatos do tipo XMI produzidos pela MVCASE podem ser diretamente manipulados pela *C-CORE* e vice-versa.

A ferramenta *SoCManager* gerencia projetos e auxilia em algumas tarefas do Gerenciamento de Configuração de Software (GCS). Dessa forma, ela é responsável pelo armazenamento dos artefatos em um repositório, no caso o CVS. A *SoCManager* utiliza uma base de dados própria para armazenar informações de controle de artefatos de qualquer tipo, incluindo XMI, código-fonte ou informações relativas à interface gráfica com o usuário, produzidos pelas ferramentas do *Orion*, ou outros artefatos quaisquer.

Porém, quando o artefato é do tipo XMI, a *SoCManager* pode realizar a introspecção sobre o mesmo, pois ela possui um interpretador XMI. Sendo assim, é possível realizar o controle sobre artefatos tanto no nível de arquivo como no nível de sua estrutura interna. A *SoCManager* utiliza para isso o conceito de Item Interno (ITAMI, 1997), que permite um controle mais fino sobre as alterações e os relacionamentos entre os artefatos. Por exemplo, é possível realizar o controle sobre elementos da UML representados internamente em um documento XMI, como atores, casos de uso, classes, entre outros.

A idéia por trás desta arquitetura é possibilitar que:

- Se tenha o uso independente de cada uma das ferramentas relacionadas ao repositório;
- As ferramentas compartilhem as informações dos artefatos armazenados no repositório, garantindo uma maior consistência e evitando a duplicação dessas informações; e
- Novas ferramentas possam ser integradas ao ambiente.

O funcionamento e a utilização conjunta das ferramentas do ambiente *Orion* pode ser melhor visualizado no capítulo 5, onde é apresentado um estudo de caso desenvolvido no ambiente.

4.4 Resumo da extensão da MVCASE

As alterações efetuadas na MVCASE facilitam o trabalho em ambientes distribuídos e a reutilização. As possibilidades de ganhos em termos de produtividade e redução de esforço são compensadoras. A MVCASE agora auxilia nas tarefas de transferência física dos arquivos entre os membros da equipe, as tarefas de coordenação de esforço e gerenciamento de alterações e de

versões, e a exaustiva tarefa da busca por um artefato reutilizável em repositórios grandes e distribuídos.

A Figura 32 mostra uma comparação entre a arquitetura original da MVCASE e sua nova versão.

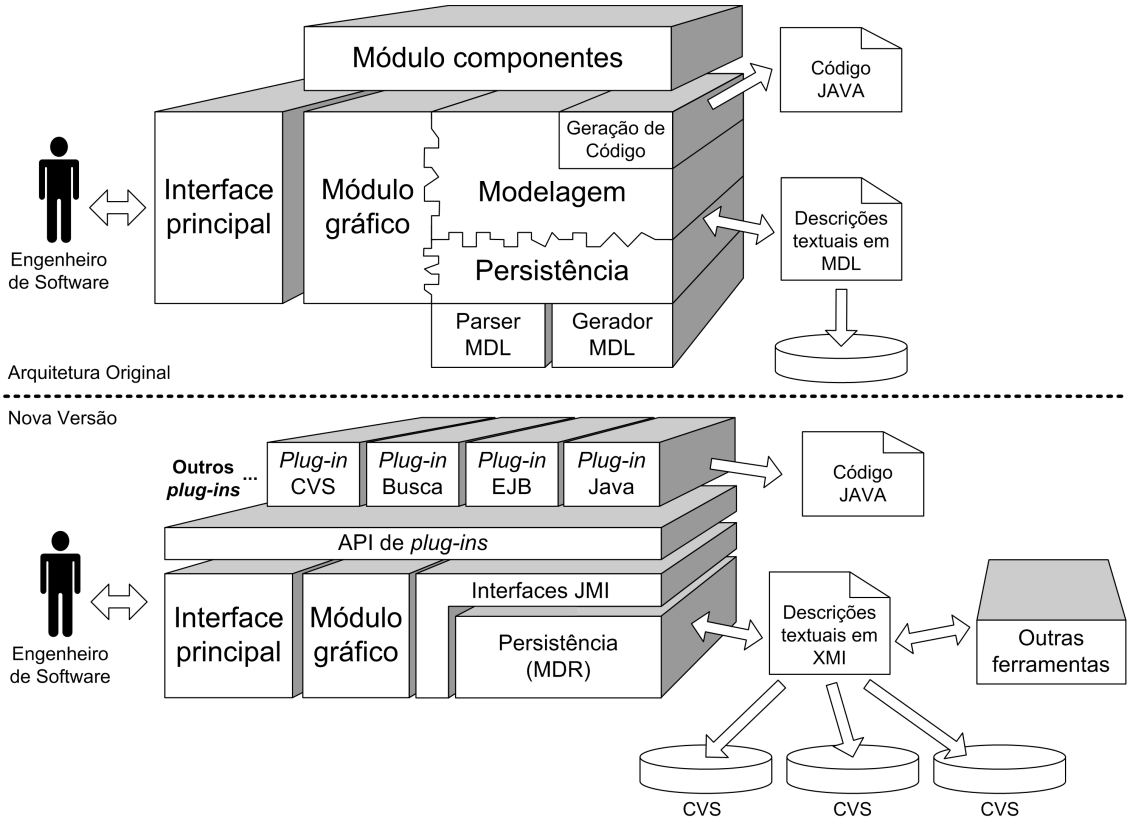

Figura 32: Comparação entre a arquitetura original da MVCASE e sua nova versão.

A nova versão da ferramenta é melhor modularizada, com as funcionalidades pouco acopladas entre si. A persistência é totalmente gerenciada pelo módulo de persistência (MDR), e o mesmo é responsável pela persistência e nada mais. O acesso aos metadados é feito somente através das interfaces JMI, diminuindo a dependência entre os módulos.

A API de *plug-ins* possibilita a fácil integração de novas funcionalidades na MVCASE, em forma de *plug-ins*. Com exceção das relacionadas à modelagem, as demais funcionalidades da ferramenta estão separadas em *plug-ins*, que podem ser instalados separadamente.

Além de se obter uma melhor arquitetura interna para a ferramenta, os requisitos apresentados no início deste capítulo foram em sua maioria atendidos:

- **Padrão de armazenamento flexível e genérico**: O XMI é um padrão flexível, podendo ser estendido facilmente, bastando para isso modificar o correspondente metamodelo. Também é genérico, permitindo que sejam representados artefatos de quaisquer natureza, desde que

compatíveis com o MOF. Sendo um padrão já estabelecido, com diversas ferramentas que o reconhecem, o XMI já foi testado e validado, sendo uma solução confiável para o armazenamento na MVCASE;

- **Interoperabilidade com outras ferramentas**: A integração de dados foi alcançada, tanto através do uso conjunto do XMI com outras ferramentas de modelagem, como com outros tipos de ferramentas. A integração da MVCASE com o ambiente Orion é um exemplo desta integração. A integração de funcionalidade também foi alcançada. O uso do JMI, um padrão para as interfaces de acesso aos dados, em conjunto com a API de *plug-ins* da MVCASE, facilita o acesso de outras ferramentas às funcionalidades da MVCASE. O uso de *plug-ins* desenvolvidos por terceiros é um exemplo dessa possibilidade;

- **Compartilhamento de artefatos**: Utilizando os serviços do CVS integrados à MVCASE, é possível que diversos Engenheiros de Software compartilhem artefatos. O CVS também oferece algum suporte ao controle de versões, auxiliando na combinação de alterações simultâneas e na resolução de conflitos;

- **Possibilidade de busca**: Com o serviço de busca, é possível automaticamente recuperar artefatos armazenados em repositórios CVS. O mecanismo em facetas utilizado é simples e de fácil utilização, é aberto, permitindo a inclusão de novos artefatos facilmente. Dá suporte ao CVS, um sistema amplamente utilizado, fazendo com que sua utilização seja mais facilmente disseminada, e pode ser utilizado com diversos repositórios distribuídos. A busca por similaridade oferece a possibilidade de um certo relaxamento na busca, aumentando as chances de reutilização;

- **Disponibilização como software livre**: A MVCASE foi disponibilizada como software livre. Sua arquitetura interna, bem modularizada e baseada em *plug-ins*, facilita a captação de contribuições da comunidade de software livre. Outros desenvolvedores e pesquisadores podem agora desenvolver seu trabalho utilizando a MVCASE. Além disso, as tecnologias aqui pesquisadas podem se disseminar mais facilmente, sendo distribuídas junto com a ferramenta; e

- **Contribuição científica**: Alguns aspectos desta extensão, como a busca por similaridade utilizando indexação métrica aplicada à classificação de software para o reuso, e a utilização conjunta do XMI com a estrutura em facetas, por exemplo, envolveram importantes inovações tecnológicas. A validade e importância dos resultados destas inovações podem ser comprovados através das publicações em periódicos e eventos bem conceituados (LUCRÉDIO; ALMEIDA; PRADO, 2004; LUCRÉDIO et al., 2004, 2004; ALMEIDA et al., 2004). Fi-

nalmente, os mecanismos construídos, e em especial o mecanismo de busca, podem ser facilmente estendidos para serem utilizados em outros cenários.

Como outras conclusões a respeito destas extensões, pode-se citar que a reutilização de sistemas de software livres já consolidados, como o MDR e o *javacvs*, poupou trabalho de implementação durante o desenvolvimento da extensão, além de proporcionar maior confiabilidade na ferramenta. A integração da MVCASE com as ferramentas do ambiente *Orion* propiciaram uma cobertura maior sobre as fases do ciclo de vida do software. O Engenheiro de Software pode utilizar as ferramentas do ambiente desde os requisitos até as fases de execução e testes, facilitando seu trabalho.

A MVCASE foi cadastrada na comunidade *java.net*, que reúne projetos de software livre em Java. Para dar mais força ao projeto, o mesmo foi vinculado ao grupo SouJava[6], a maior comunidade brasileira de desenvolvimento em Java. Maiores informações sobre o projeto MVCASE como software livre se encontram em sua *homepage*[7] do *Java.net*

O próximo capítulo é dedicado à ilustração da extensão apresentada neste capítulo com um estudo de caso, e à análise dos resultados através de testes com a nova versão da ferramenta.

[6]http://www.soujava.com.br/index.jsp
[7]http://mvcase.dev.java.net/

5 *Estudo de caso e avaliação*

Neste capítulo é apresentado um estudo de caso para ilustrar a utilização dos serviços remotos de armazenamento e busca de artefatos de software. Em seguida, é feita uma avaliação dos resultados, com base nos dados obtidos na execução do estudo de caso. O estudo de caso compreende a construção e reutilização de componentes dos domínios de Aplicações Multimídia e de Educação à Distância (EAD), e faz parte de outro projeto de pesquisa desenvolvido no Departamento de Computação da UFSCar, denominado DBCM - Desenvolvimento Baseado em Componentes Multimídia (UFSCAR, 2004).

5.1 Descrição dos domínios

Esta seção descreve brevemente os domínios envolvidos com o estudo de caso, que são o domínio de Aplicações Multimídia e o de Educação à Distância. As descrições a seguir foram extraídas de (SILVA, 2004).

Aplicações com recursos multimídia vêm se tornando mais comuns em ambientes computacionais, pois oferecem interfaces áudio-visuais que facilitam a compreensão e permitem uma grande interatividade com o usuário. A utilização de recursos multimídia nas aplicações de diferentes domínios tem exigido novas técnicas, métodos e ferramentas para apoiar seus desenvolvimentos e um maior conhecimento de programação para realizar suas implementações.

Informações multimídia podem ser classificadas com base nos tipos de mídias envolvidas. Mídia se refere ao tipo de informação, como dados alfanuméricos, imagens, áudio e vídeo. Existem muitas formas de classificar mídias. Uma forma proposta por Guojun (GUOJUN, 1996) classifica as mídias conforme a existência ou não da dimensão de tempo para a mídia. Nessa classificação, existem duas classes de mídias: estáticas e dinâmicas.

Mídias estáticas não têm a dimensão de tempo e, portanto, seus conteúdos e significados não são dependentes do tempo de apresentação. Mídias estáticas incluem dados alfanuméricos, gráficos, e imagens. Mídias dinâmicas têm a dimensão de tempo, e seus significados e exatidão

dependem da taxa com que são apresentadas. Mídias dinâmicas incluem animação, áudio e vídeo.

Do ponto de vista lingüístico, qualquer sistema capaz de manipular mais de um tipo de mídia pode ser chamado de “sistema multimídia”. Guojun (GUOJUN, 1996) define sistema multimídia como aquele capaz de manipular ao menos um tipo de mídia dinâmica no formato digital, assim como mídia estática. Nessa definição, a combinação sincronizada de múltiplos tipos de mídias com ao menos uma mídia dinâmica denomina-se “objeto multimídia”.

Com o avanço tecnológico, a Educação a Distância é vista hoje não só como um sistema especial, mas como uma parte integrante do aprendizado que prepara profissionais nas diversas áreas de ensino e pesquisa. Muitas Instituições de Ensino estão aperfeiçoando e desenvolvendo seus programas de EAD. Recursos como redes de trabalho, videoconferência, computação colaborativa, Internet, objetos multimídia e outros são utilizados para maior interação entre estudantes e professores, melhoria da qualidade do ensino e difusão da educação em diferentes locais e regiões do mundo (SILVA, 2004).

No caso do domínio EAD foram identificados três atores principais: Administrador, Professor e Estudante. Na tabela 1 são apresentadas as principais funcionalidades que estudantes, professores e administradores dos cursos podem realizar.

Ator	**Ação (funcionalidade)**
Administrador	Cria cursos
Administrador	Gerencia cursos
Professor	Cria disciplinas
Professor	Cria aulas (para determinada disciplina)
Professor	Cria exercícios e provas (para determinada disciplina)
Professor	Armazena mídias (como material de aula)
Professor	Cria objetos multimídia (como material de aula)
Professor	Gerencia disciplinas
Estudante	Matricula-se no curso
Estudante	Realiza aulas
Estudante	Realiza exercícios
Estudante	Realiza provas
Estudante	Acessa materiais (de determinada aula)

Tabela 1: Funcionalidades do domínio EAD.

5.2 Preparação do estudo de caso

O estudo envolveu uma equipe de quatro alunos (A,B,C e D) de pós-graduação, com experiência na linguagem Java e nos domínios EAD e multimídia. A equipe foi dividida da seguinte forma:

O aluno A foi designado gerente de projeto, responsável por controlar as informações do projeto, definição da equipe, controle dos módulos do projeto, designação de tarefas, e reutilização de projetos. O gerente de projeto também define quais são as *baselines* do projeto. Uma *baseline* (IEEE, 1990) consiste em um conjunto de artefatos controlados, que serve de base para as próximas atividades de desenvolvimento. Além de gerente de projeto, o aluno A também foi designado bibliotecário, responsável pela definição dos repositórios do projeto e associação de módulos do projeto aos repositórios.

O aluno B foi designado como único membro do comitê de controle de configuração. Este comitê é responsável pela definição dos tipos de artefatos a serem controlados, avaliação de requisições de mudança, solicitação de modificações, avaliação de planejamentos de modificação. O aluno B também foi designado como auditor, responsável por avaliar os artefatos entregues para controle, e os artefatos que foram modificados.

Todos os alunos A,B,C e D foram designados Engenheiros de Software, responsáveis pelas tarefas de desenvolvimento, envolvendo a modelagem, codificação, testes e empacotamento do software.

Os alunos do estudo utilizaram as ferramentas do ambiente *Orion*, sendo que a *SoCManager* foi utilizada nas tarefas de Gerenciamento de Configuração de Software, a MVCASE nas tarefas de modelagem e construção dos componentes, e a C-CORE nas tarefas de codificação e projeto de interface com o usuário.

O estudo foi desenvolvido nos laboratórios de pesquisa do Departamento de Computação da UFSCar, com as equipes distribuídas em duas salas diferentes. A comunicação entre os membros da equipe foi realizada através de correio eletrônico, e aplicativos de *Instant Messaging* e teleconferência, quando a situação exigia iterações mais imediatas.

O estudo foi dividido em quatro projetos: 1) Construção de um *framework* de componentes para educação à distância; 2) Construção de uma aplicação que reutilizou os componentes do *framework* para educação à distância; 3) Construção de um *framework* para aplicações multimídia; e 4) Construção de uma aplicação que reutilizou os componentes do *framework* multimídia.

Foi utilizado neste estudo o método *Catalysis* (D'SOUZA; WILLS, 1999). *Catalysis* compreende três níveis: Domínio do Problema, Especificação dos Componentes e Projeto Interno dos Componentes. Esses níveis são realizados conforme o modelo espiral de ciclo de vida de software (PRESSMAN, 2001). Sempre que necessário retorna-se aos passos anteriores para refinar ou remover inconsistências dos artefatos produzidos em cada passo.

Inicialmente, o gerente de projeto criou um novo projeto de software na *SoCManager*, e ca-

dastrou os demais membros da equipe, atribuindo-lhes seus papéis conforme descrito nesta seção. O bibliotecário criou um repositório CVS para armazenar os artefatos do projeto.

O levantamento de requisitos foi realizado pelos Engenheiros de Software, com base em experiências anteriores (SILVA, 2002a; SILVA; VIEIRA, 2001; SILVA, 2002b; CATARINO, 2002) envolvendo o próprio grupo de pesquisa onde foi desenvolvido o presente trabalho.

Com o propósito de ilustrar a execução do estudo de caso, e a utilização dos serviços remotos de armazenamento e busca de artefatos de software, a seguir apresenta-se dois dos projetos do estudo de caso: a construção de um *framework* EAD e sua reutilização em uma aplicação. A construção do *framework* e de uma aplicação do domínio multimídia seguiu a mesma metodologia, com quatro alunos formando a equipe, utilizando as ferramentas do ambiente *Orion* para desenvolver o projeto segundo o método *Catalysis*.

5.3 Construção do *framework* para ensino à distância

Seguindo *Catalysis*, partiu-se do domínio do problema, onde foram identificados os principais objetos e ações envolvidos. Em seguida, estas primeiras especificações foram refinadas em modelos de casos de uso e modelos de tipos. A Figura 33 mostra este modelo de tipos construído na MVCASE. Nesta figura não são mostrados os atributos de cada tipo, para enfatizar os relacionamentos entre eles.

Um professor (*Teacher*) pode criar várias disciplinas (*Discipline*) para determinado curso (*Course*) e definir várias aulas (*Lesson*) e provas (*Test*) para cada disciplina. Se o professor desejar, pode também definir exercícios (*Exercise*) de múltipla escolha (*Option*) para cada aula, com até cinco alternativas. As provas da disciplina agregam várias questões (*Question*) e cada questão pode agregar até cinco alternativas (*Option*). As respostas dos exercícios (*ReplyExercise*) e das questões das provas (*Question*) são armazenadas para que o professor possa definir uma média final da disciplina para o estudante. Em relação à prova, o professor pode definir várias questões e atribuir um número de questões para que cada estudante responda. Por exemplo, um professor define trinta questões para uma prova de determinada disciplina e define que os estudantes precisam responder apenas dez questões. Quando o estudante acessar a prova, são escolhidas, aleatoriamente, dez questões entre as trinta definidas pelo professor. A prova gerada para o estudante é armazenada em uma nova classe.

Ao definir uma disciplina, podem ser indicadas várias bibliografias (*Bibliography*). Cada disciplina também pode estar associada a requisitos (*Requirements*), que são outras disciplinas obrigatórias para realização da disciplina. Quando um estudante matriculado acessa e realiza os exer-

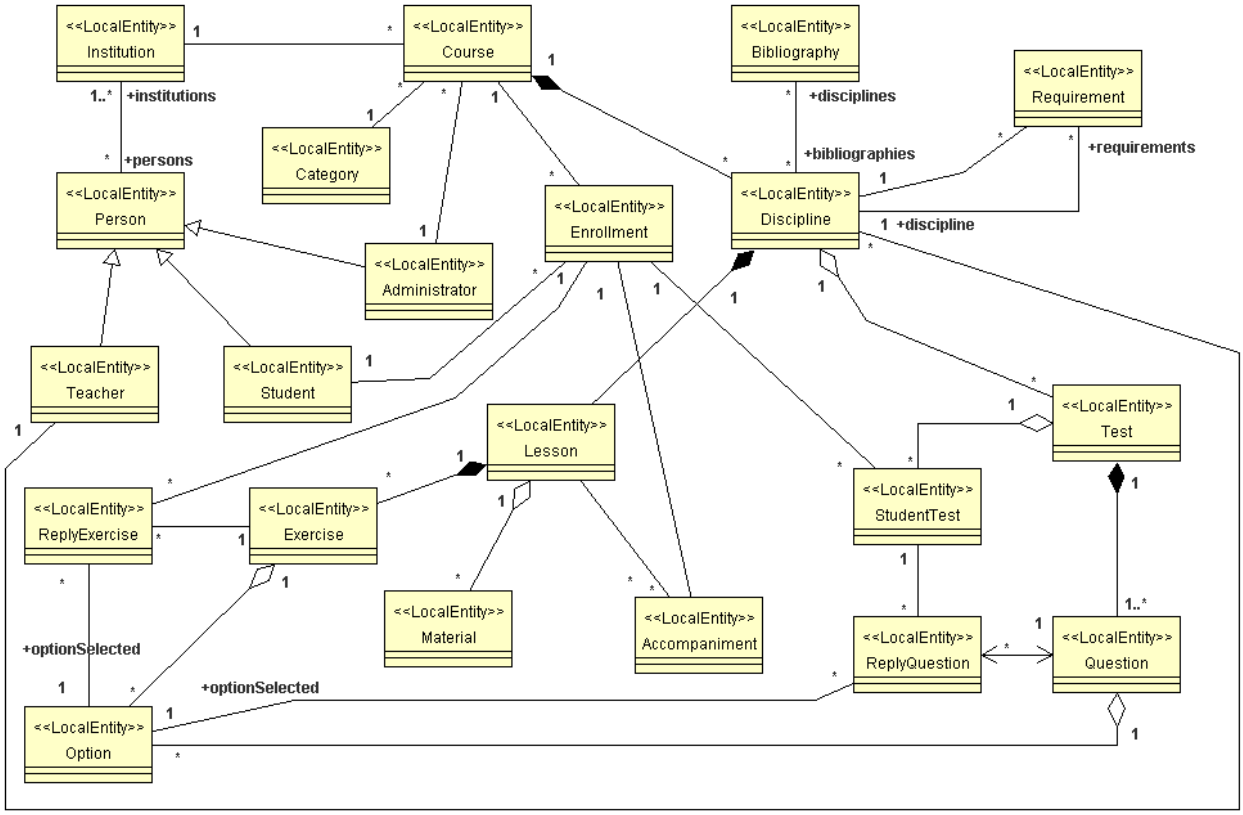

Figura 33: Tela da ferramenta MVCASE com o modelo de tipos.

cícios de uma aula de determinada disciplina, é gerado um acompanhamento (*Accompaniment*) do estudante para a aula. Uma aula pode agregar vários materiais (*Material*). O material pode ser uma mídia ou uma composição de cenas (objeto multimídia). Uma composição de cenas pode agregar uma ou mais cenas, e cada cena agrega uma ou mais mídias, que podem ser do tipo imagem, texto, áudio ou vídeo.

Durante a criação dos modelos, os Engenheiros de Software trabalharam em conjunto sobre os artefatos de modelagem. O *plug-in* para o CVS da MVCASE foi utilizado para acesso remoto ao repositório do projeto, auxiliando na transferência dos artefatos através da rede e no trabalho distribuído. Para ilustrar a utilização deste *plug-in*, considere o modelo da Figura 34.

Dois Engenheiros de Software eram responsáveis pelo modelo da Figura 34, modificando-o ao mesmo tempo. Um deles identificou que, ao invés de armazenar a idade do aluno (a), seria mais interessante armazenar a sua data de nascimento. O outro identificou a necessidade de um campo para que o professor insira comentários gerais sobre uma determinada prova (b). A seguir, cada Engenheiro de Software obteve uma cópia deste artefato, e realizou a sua mudança, conforme ilustra a Figura 35. Do lado esquerdo, o Engenheiro de Software A modificou o atributo **age** para **birthDate** (a). Do lado direito, o Engenheiro de Software B adicionou um novo atributo: **comments**.

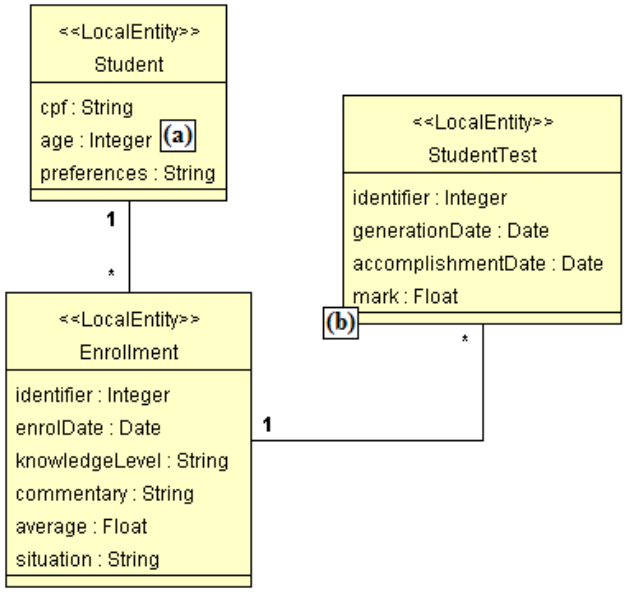

Figura 34: Modelo com algumas classes do *Framework* EAD.

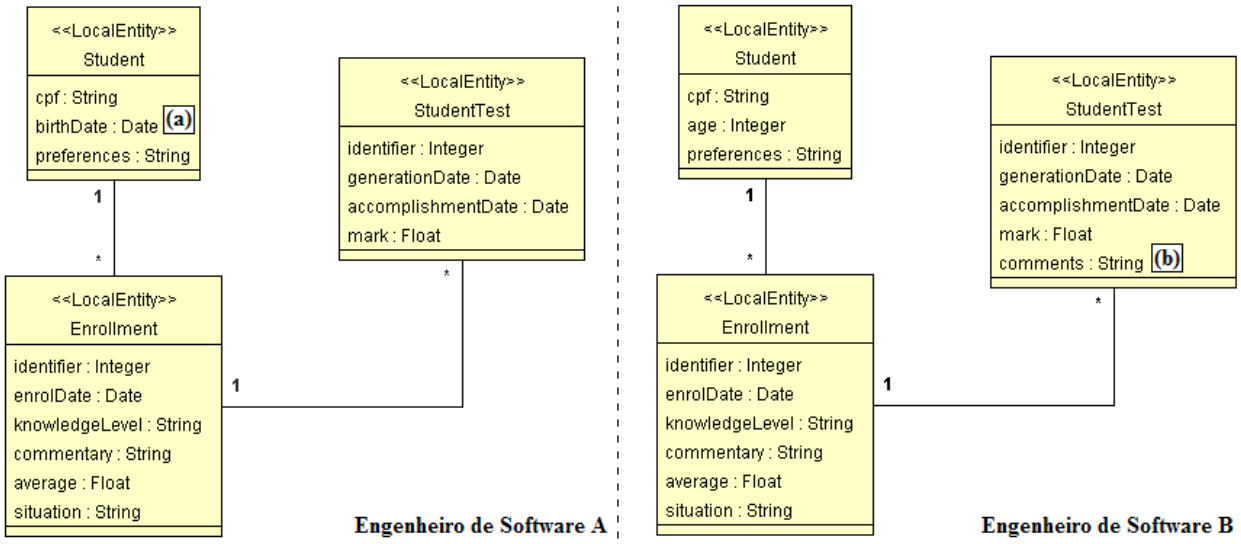

Figura 35: Mudanças sendo efetuadas por diferentes Engenheiros de Software ao mesmo tempo.

Após terem realizado as mudanças, os Engenheiros de Software utilizaram o *plug-in* do CVS para enviar os artefatos alterados novamente para o repositório. O primeiro Engenheiro de Software a enviar os artefatos alterados para o repositório não teve nenhum problema, pois a versão anterior que se encontrava no repositório é a mesma que ele havia recuperado. Já o segundo recebe uma mensagem informando que o artefato que está no repositório (que é a nova versão posta pelo primeiro Engenheiro de Software) é diferente daquele que havia sido por ele recuperado. É necessário então realizar a combinação (*merge*) das duas versões. Para isso, o Engenheiro de Software utiliza o *plug-in* para CVS da MVCASE, que tenta realizar a combinação automaticamente. Como neste caso cada Engenheiro de Software alterou parte diferente do modelo, não ocorre nenhum conflito, e esta operação termina sem problemas. O artefato final é então enviado para o

repositório, e passa a conter ambas as alterações (Figura 36). Neste caso, não ocorreu conflito, mas caso este tivesse ocorrido, o Engenheiro de Software deveria então resolvê-lo manualmente, diretamente no arquivo XMI.

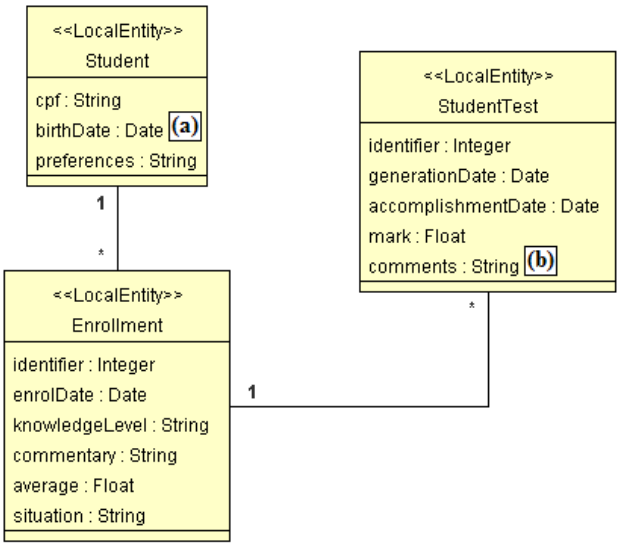

Figura 36: Versão final do artefato, que contém ambas as alterações: (a) e (b).

Prosseguindo com *Catalysis*, partiu-se para o segundo nível: Especificação dos Componentes. Neste passo, os Engenheiros de Software identificaram quais seriam os componentes do *framework*, adicionando também quais as operações a serem implementadas. Foram identificados neste *framework* 21 componentes de acesso a banco, e 7 componentes de negócio. As dependências e relacionamentos entre os componentes também foram especificadas.

No terceiro nível de *Catalysis*, realizou-se o projeto interno dos componentes. Foi utilizada para isso a tecnologia EJB (DEMICHIEL, 2002), que auxilia na construção de componentes distribuídos com suporte à persistência em Banco de Dados. Neste caso, a MVCASE possui um *plug-in* que gera automaticamente a estrutura de componentes EJB a partir dos modelos de mais alto nível. A Figura 37 mostra um diagrama de componentes com alguns dos componentes EJB gerados pela MVCASE. A dependência entre os componentes transientes (estereótipo *EJBSession*) e persistentes (estereótipo *EJBLocalEntity*), formando uma arquitetura em camadas, também é especificada na ferramenta.

Finalmente, foi utilizado o *JavaGeneratorPlugin*, um *plug-in* da MVCASE para a geração de código Java. O código gerado foi compilado e executado. Foi utilizado o servidor de componentes JBoss (JBOSS, 2005) para execução. Testes foram realizados, utilizando a *C-CORE* para auxiliar nas tarefas de depuração. Correções e alterações foram realizadas diretamente nos modelos, utilizando a MVCASE para gerar novamente o código correspondente, ou diretamente no código, utilizando a C-CORE para persistir as mudanças em XMI, mantendo os modelos atualizados. Após

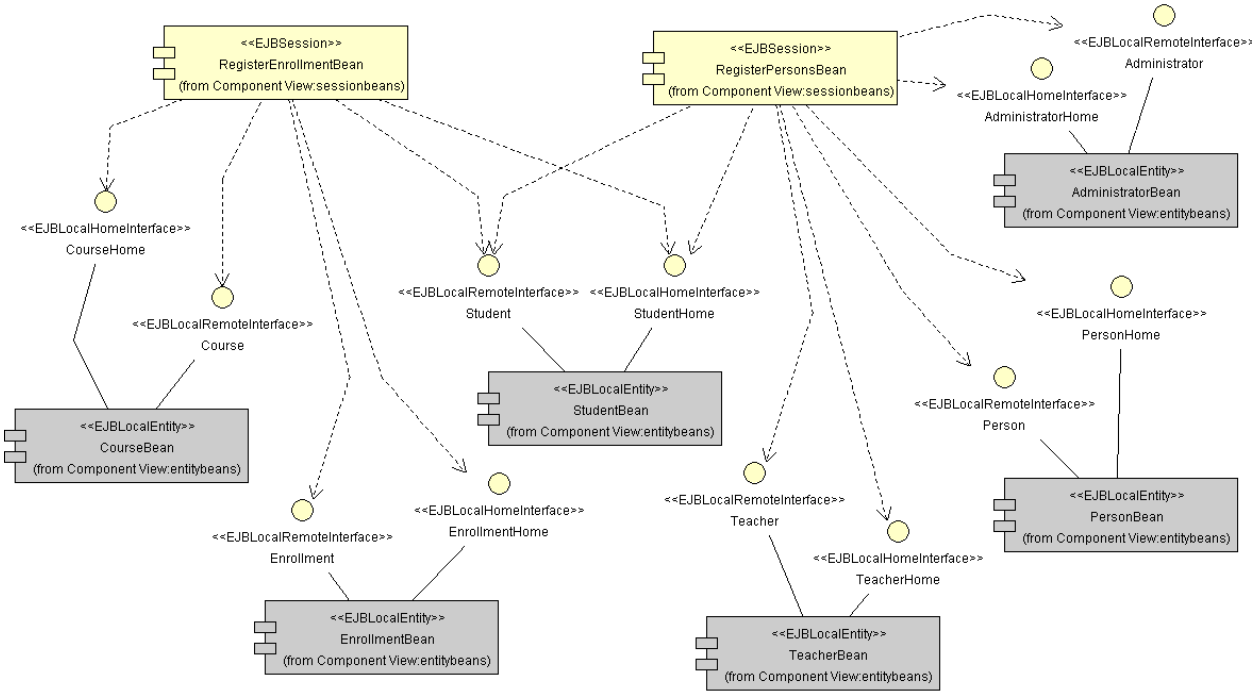

Figura 37: Diagrama que mostra parte dos componentes do *framework* EAD.

a finalização do *framework*, a ferramenta *SoCManager* ficou responsável pelo gerenciamento das alterações sobre os artefatos construídos, seguindo o processo definido em (CUNHA, 2005).

Uma vez finalizado, esse *framework* pode ser então reutilizado na construção de aplicações do domínio de educação à distância. Na próxima seção é apresentada a segunda parte deste estudo de caso, onde uma aplicação que reutiliza os componentes deste *framework* foi desenvolvida.

5.4 Construção de uma aplicação para ensino à distância

Construiu-se uma aplicação onde administradores podem criar cursos, e professores podem criar disciplinas para os cursos. Ao criar as disciplinas, os professores devem também criar aulas e definir materiais de apoio às aulas. Os materiais definidos podem ser objetos multimídia que o professor também deverá criar. Os estudantes acessam as disciplinas e visualizam os materiais disponíveis.

A aplicação foi desenvolvida reutilizando os componentes EJB do *framework* EAD, e a tecnologia JSP (*Java Server Pages*) para construção de páginas *Web*.

Para facilitar a reutilização dos componentes, foi utilizado o mecanismo de busca da MV-CASE, que realiza a busca sobre artefatos XMI e código-fonte em Java. O mecanismo realiza a indexação automática sobre os artefatos XMI, e os disponibiliza para busca utilizando navegação, consulta ou busca por similaridade. A Figura 38 ilustra a utilização do mecanismo de busca através

da navegação.

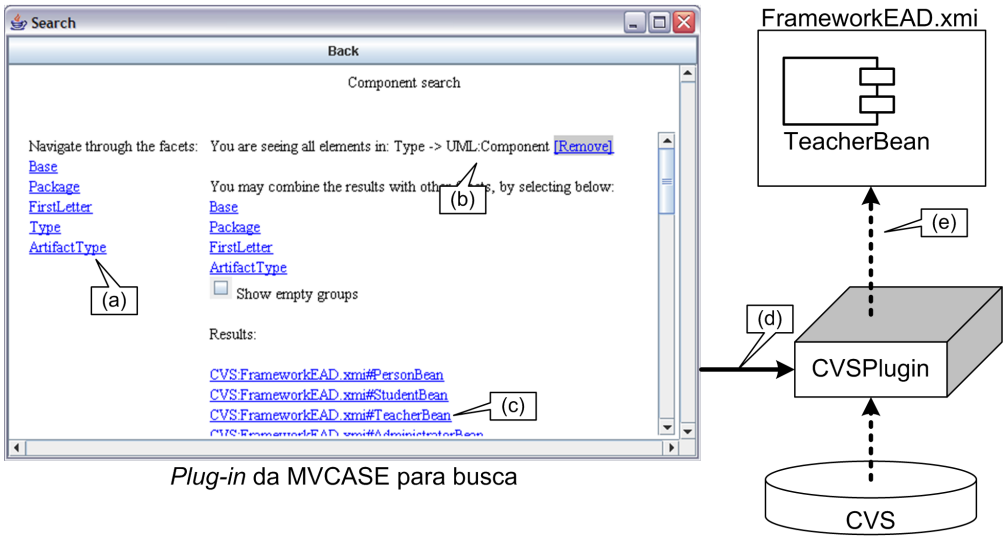

Figura 38: Utilização do mecanismo de navegação para recuperar artefatos reutilizáveis.

Na Figura 38, são mostradas, no lado esquerdo, as cinco facetas extraídas automaticamente (a). No lado direito são mostradas as facetas selecionadas. No caso, estão sendo selecionados todos os artefatos do tipo **UML:Component** (b). Para recuperar um artefato, o Engenheiro de Software clica sobre ele na interface do serviço de busca (c). O *plug-in* da MVCASE para busca, que está mostrando a interface do serviço de busca e monitorando as ações do usuário, automaticamente envia uma mensagem (d) ao *CVSplugin*, para que o mesmo recupere o artefato referente ao elemento selecionado (e). No caso, está sendo recuperado o artefato **FrameworkEAD.xmi**, que contém o componente **TeacherBean** selecionado.

Vale ressaltar que é possível combinar outras facetas para reduzir o número de componentes visualizados, facilitando a tarefa de se encontrar o componente desejado. Também foram indexados todos os componentes da API Java, para facilitar a construção da aplicação.

Uma vez recuperados os componentes necessários, foram construídas páginas JSP que implementam as funcionalidades requeridas para a aplicação. A Figura 39 mostra um diagrama com alguns componentes da aplicação. No caso, são mostradas três páginas JSP: **ManageTeacher.jsp**, **RegisterDiscipline.jsp** e **RegisterCourse.jsp**, que reutilizaram os componentes **TeacherBean**, **DisciplineBean** e **CourseBean** do *framework* EAD, respectivamente, e o pacote **java.util** da API da linguagem Java.

A Figura 40 ilustra uma tela da aplicação implementada, para cadastro de aulas. Neste exemplo, está sendo cadastrada uma aula introdutória ao assunto de Desenvolvimento Baseado em Componentes.

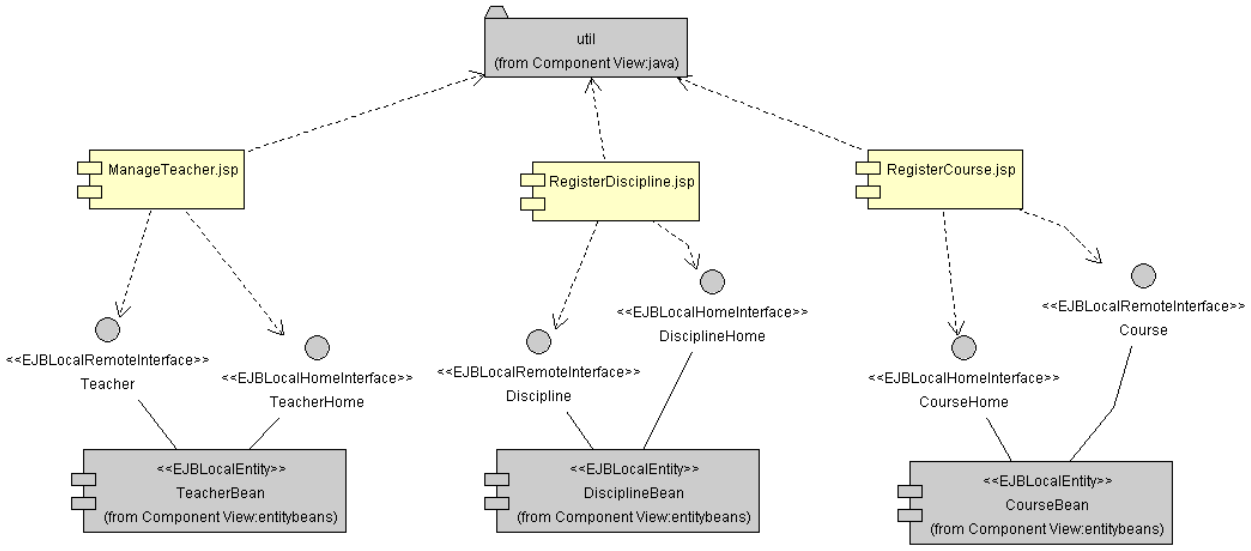

Figura 39: Diagrama de componentes da aplicação, onde são mostradas páginas JSP que reutilizam componentes do *framework* EAD e da API Java.

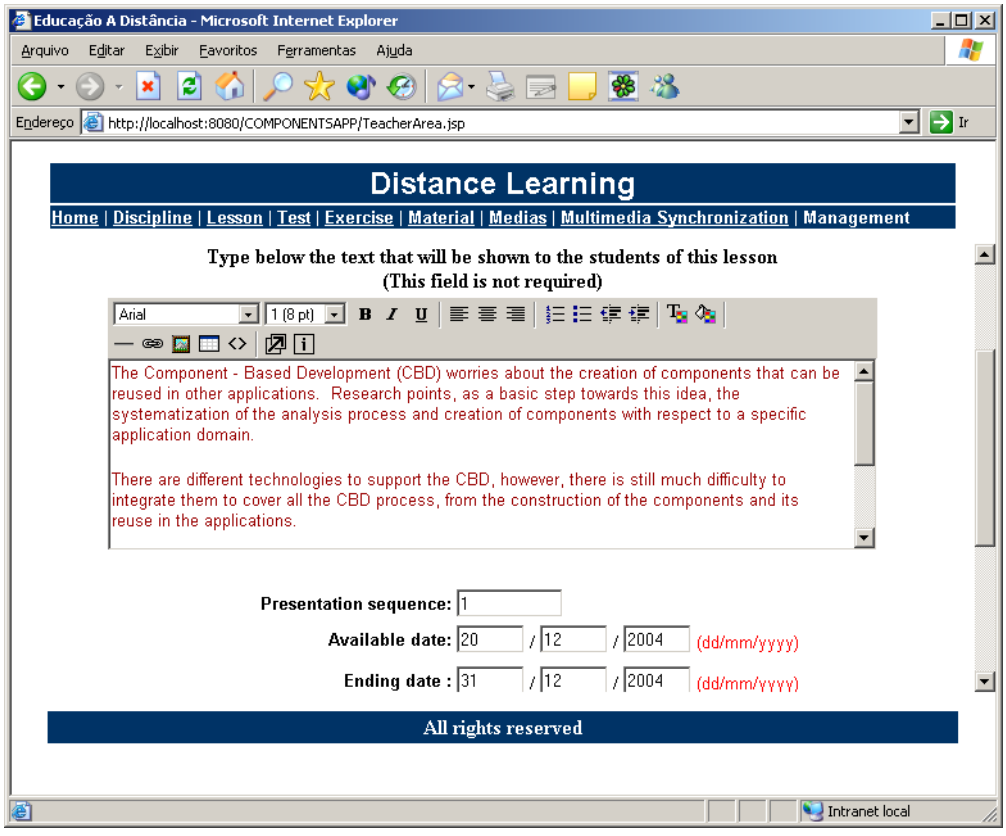

Figura 40: Tela da aplicação construída.

5.5 Discussão

Na construção dos *frameworks* EAD e multimídia, foram geradas 41 classes de análise, que deram origem a 41 componentes EJB. Foram gerados, para esses componentes, 123 arquivos Java.

A construção das aplicações envolveu 49 páginas JSP, reutilizando estes componentes.

Os alunos foram questionados, após a execução do estudo de caso, sobre os possíveis benefícios observados. Dentre as respostas obtidas, destacam-se:

- a facilidade de intercâmbio dos artefatos, através dos serviços de armazenamento remoto utilizando o CVS, o que proporcionou o trabalho em equipe;
- a introspecção sobre os artefatos XMI, permitindo encontrar artefatos internos a ele utilizando o serviço de busca;
- facilidade de reutilização, proporcionada tanto pelo serviço de busca como pelo serviço de armazenamento remoto; e
- possibilidade de intercâmbio com outras ferramentas.

Portanto, os resultados esperados para este trabalho puderam ser observados de forma qualitativa. Isso indica que os benefícios em termos de reutilização e facilidade de trabalho em equipe foram alcançados, devido à utilização dos mecanismos remotos de armazenamento e busca de artefatos de software.

Os alunos também foram questionados quanto a possíveis limitações e outras sugestões. Foram citados principalmente:

- o serviço de busca não considera os descritores em texto dos artefatos. No caso, um extrator automático de texto livre poderia ser uma solução; e
- também foi notada uma queda de desempenho da ferramenta, quando comparada com a sua versão original, antes da extensão. Analisando-se o código-fonte da ferramenta, e as situações onde a queda de desempenho foi observada, identificou-se o MDR como causa deste problema, pois os momentos de acesso constante ao metamodelo correspondem exatamente aos picos de baixo desempenho.

Este estudo de caso permitiu observar as vantagens dos novos serviços da MVCASE, porém foram produzidos relativamente poucos componentes, o que não exigiu muito do serviço de busca. Por isso, com o objetivo de melhor avaliar seu desempenho, foi realizado um estudo comparativo (LUCRÉDIO et al., 2004) entre o mecanismo de busca em facetas e um mecanismo de busca para *Web*.

5.6 Avaliação da busca

A maneira mais simples de se buscar componentes é utilizando um mecanismo de busca para *Web*. Existem vários mecanismos livres, como o *Jakarta Lucene*[1], que exigem pouco esforço para serem utilizados, sendo facilmente integrados a repositórios.

Estes mecanismos de busca são uma boa base de comparação para se avaliar um mecanismo de busca de componentes. Se um mecanismo possui desempenho pior do que um mecanismo para *Web*, então ele possui pouco valor prático, já que é mais simples utilizar o mecanismo *Web*.

Esse estudo comparativo foi realizado com o mecanismo em facetas utilizado na MVCASE e o *Jakarta Lucene*. Primeiramente, o *Jakarta Lucene* foi instalado. Cerca de duzentos componentes da API Java foram indexados, tanto no *Jakarta Lucene* como no mecanismo em facetas. Porém, como já citado, os extratores automáticos que foram construídos nesta pesquisa não são capazes de compreender texto livre[2], ao contrário do *Lucene*. Sendo assim, o nível semântico alcançado com a busca através do *Lucene* seria naturalmente maior. Portanto, utilizou-se para esta comparação a classificação manual, onde um Engenheiro de Software foi responsável por inserir as informações semânticas nas facetas. Dessa forma, pode-se avaliar o desempenho da estrutura em facetas quanto à sua capacidade de facilitar a formulação de consultas e sua maneira de realizar a busca, frente ao mecanismo *Web*.

Foi utilizado um esquema de quatro facetas: **Pacote**, **Domínio**, **Tipo principal de dados** e **Ação realizada**. Ambos os mecanismos foram implantados no servidor *Jakarta Tomcat*[3].

Em seguida, foram definidas algumas possíveis situações de reutilização, de modo que um único artefato fosse capaz de resolver o problema:

1. Dada uma sentença, extrair as palavras que a compõem;
2. Dada uma coleção, percorrer todos seus elementos;
3. Converter uma *String* hexadecimal em um inteiro; e
4. Formatar uma data em um objeto do tipo *String*.

Os alunos deste estudo foram oito estudantes de pós-graduação da Universidade Federal de São Carlos e da Universidade Federal de Pernambuco, que foram solicitados a realizar buscas no

[1] http://jakarta.apache.org/lucene/docs/index.html

[2] A complexidade exigida para tais extratores fez com que sua construção se tornasse inviável para este trabalho de mestrado, considerando-se seu escopo e restrições de tempo. Conforme apresentado na seção 6.3 desta dissertação, essa é uma possibilidade de trabalhos futuros

[3] http://jakarta.apache.org/tomcat/

Jakarta Lucene e no mecanismo em facetas. Cada aluno recebeu duas das situações, uma para ser solucionada com o *Lucene* e outra com o mecanismo em facetas. Dessa forma, os resultados da primeira situação não influenciariam a segunda. Cada situação foi fornecida a um mesmo número de pessoas, de modo que todas foram testadas um mesmo número de vezes. Os alunos foram observados enquanto realizavam as buscas, e suas ações foram registradas. Usando o mecanismo em facetas, as consultas foram formuladas escolhendo-se, de uma lista pré-definida, uma ou mais palavras para cada faceta, enquanto que no *Jakarta Lucene*, os alunos forneceram conjuntos quaisquer de palavras. Os resultados eram classificados quanto à sua similaridade à consulta, através da busca por similaridade, no caso do mecanismo em facetas, e quanto à relevância, no caso do *Lucene*.

Foram medidos o número de consultas consecutivas necessárias para se encontrar o artefato correto e o número de *links* seguidos para que o aluno tivesse certeza de que o artefato era (ou não) o artefato correto. Em termos práticos, quanto maior o número de consultas, e quanto maior o número de *links*, pior será o mecanismo, pois exigirá mais trabalho para realizar a busca. A Tabela 2 mostra as médias de consultas e *links* para cada situação, em ambos mecanismos.

	Jakarta Lucene		**Mecanismo em facetas**	
Situação	Consultas	Links	Consultas	Links
1	2,5	4,5	1,5	1,5
2	3	4	1	1
3	2	2,5	1,5	1,5
4	2,5	3,5	1,5	1,5
Total	**2,5**	**3,625**	**1,375**	**1,375**

Tabela 2: Resultados do estudo comparativo (LUCRÉDIO et al., 2004).

Conforme pode ser visto, os alunos utilizando o mecanismo em facetas precisaram de menos consultas para encontrar o artefato correto, o que pode ser explicado pela facilidade de formular consultas escolhendo-se palavras dentro de conjuntos pré-definidos, ao invés de escolher palavras quaisquer. Porém, podem existir alguns casos onde a escolha aleatória de palavras, como no caso do Lucene, é mais vantajosa. Um exemplo é quando o Engenheiro de Software não conhece muito bem o domínio, e deseja fazer uma busca mais abrangente.

Os alunos também seguiram menos *links* para definir se o artefato era ou não o desejado. Isso pode ter acontecido porque o *Jakarta Lucene* apresenta um número grande de artefatos como resultado, enquanto que a busca em facetas apresenta um número menor de artefatos, o que facilita a exploração do resultado. Outra explicação é que a busca por similaridade do mecanismo em facetas apresenta os artefatos mais relevantes primeiro, enquanto que o *Lucene* apresenta os artefatos relevantes somente depois, exigindo que o reutilizador percorra um número maior de artefatos irrelevantes antes de encontrar o desejado.

Porém, mesmo que os resultados deste estudo tenham demonstrado que nestes casos específicos o mecanismo da MVCASE obteve melhor desempenho, é difícil afirmar que este mecanismo é melhor do que o *Lucene*, pois a avaliação não considerou todos os aspectos da busca. Por exemplo, a classificação manual dos artefatos, tendo sido feita por um especialista, naturalmente resultou em um maior nível semântico do que a classificação automática feita pelo *Lucene*. Para isso, se fazem necessários estudos mais completos e abrangentes, de forma a avaliar de forma mais precisa quais os pontos fortes e fracos deste mecanismo. Trabalhos futuros podem ser realizados neste sentido.

6 *Considerações finais e trabalhos futuros*

Este capítulo apresenta algumas considerações finais, destacando alguns trabalhos relacionados, as principais contribuições deste trabalho e direções para trabalhos futuros.

6.1 Trabalhos relacionados

O projeto Agora (SEACORD; HISSAM; WALLNAU, 1998), do *Carnegie Mellon University/-Software Engineering Institute* (CMU/SEI), utiliza uma abordagem para descoberta e recuperação de componentes através da *Web*. Utilizando introspecção, o protótipo Agora identifica e recupera componentes Java automaticamente, tornando-os disponíveis para reutilização. O princípio utilizado é que repositórios de componentes voltados à reutilização devem ser distribuídos, abertos e pouco restritos. A abordagem para recuperação e busca de componentes utilizada no presente trabalho é semelhante à do Agora, por também utilizar introspecção sobre os artefatos para classificá-los para a busca. A presente abordagem também pode ser utilizada com diversos repositórios CVS, classificando artefatos automaticamente, o que vai ao encontro da idéia de repositórios distribuídos do Agora. Porém, este último considera apenas a informação contida no código-fonte, enquanto que o presente trabalho pode utilizar as informações de modelagem contidas no XMI, aumentando o nível semântico que pode ser obtido.

O trabalho de Hummel & Atkinson (HUMMEL; ATKINSON, 2004) também envolve a descoberta e classificação de componentes para busca e reutilização, utilizando casos de teste como entrada. Essa abordagem é semelhante à de Podgurski & Pierce (PODGURSKI; PIERCE, 1993), explorando a execução como meio de descoberta de componentes. No trabalho de Hummel & Atkinson, são combinadas técnicas de recuperação de componentes em um processo de seis etapas: Na primeira etapa o componente desejado é descrito sintaticamente, através de descrições UML ou Java. Na segunda etapa, a informação semântica é inserida através da definição de casos de teste que avaliam se um componente satisfaz ou não suas expectativas. Em uma terceira etapa, a busca é realizada, utilizando mecanismos de *Web search* e comparação de assinatura. Uma vez recuperados, os componentes são compilados, e os casos de teste são executados para verificar se os componentes

atendem aos requisitos. Essa abordagem possui um problema de considerar apenas os casos de teste como informação semântica. Em casos mais simples, onde os componentes realizam cálculos ou outras computações, isto é suficiente. Porém, para componentes mais complexos, a construção de casos de teste pode ser inviável. A presente abordagem utiliza a informação semântica contida no XMI, que não possui essa restrição.

Quanto à utilização do XMI, outras ferramentas também se baseiam na mesma API utilizada nesta pesquisa: o MDR. As ferramentas ArgoUML (ARGOUML, 2005) e Poseidon (GENTLEWARE, 2005) também dão suporte à modelagem em UML. Porém, elas apresentam o problema referente ao identificador único dos elementos, descrito na seção 4.3.3, dificultando sua utilização em sistemas de controle de versões. Além disso, não possuem uma arquitetura em *plug-in*, como a MVCASE, dificultando sua extensão e evolução.

Em (OLIVEIRA; MURTA; WERNER, 2004), os autores utilizam o XMI para realizar o controle de versões em modelos baseados no MOF. O sistema proposto pode ser utilizado por qualquer ferramenta CASE que seja capaz de exportar/importar documentos XMI, como por exemplo a MVCASE. Sendo baseado no MOF, ele permite um controle mais fino sobre os elementos controlados, e não somente sobre arquivos-texto, como é o caso do CVS. Isso facilita as tarefas de rastreamento de mudanças e detecção e resolução de conflitos. Nesta abordagem, uma ferramenta como a MVCASE poderia automaticamente mostrar pontos de conflito em sua própria interface visual. Atualmente, a resolução dos conflitos que ocorrem em documentos XMI armazenados no CVS deve ser feita diretamente sobre sua forma textual. Segundo os autores, uma restrição deste trabalho é que todas as versões dos artefatos são armazenadas integralmente, e não apenas as diferenças entre as versões, como é o caso do CVS. Os autores justificam essa escolha pelo atual barateamento dos recursos de memória e evolução das técnicas de compactação de arquivos.

6.2 Contribuições

Este trabalho oferece diferentes contribuições, tanto para a comunidade acadêmica como para a comunidade de desenvolvimento em geral.

Após a extensão, a MVCASE se tornou apta a dar suporte a projetos de desenvolvimento em ambientes distribuídos. É possível que equipes compostas por vários membros trabalhem com um mesmo conjunto de artefatos simultaneamente.

Comparando-se com a arquitetura original da MVCASE, o nível de reutilização alcançado por meio do uso da sua nova versão foi acrescido, em parte graças ao serviço de busca, que oferece oportunidades de reutilização para o Engenheiro de Software. É possível encontrar artefatos pre-

viamente construídos em meio a diferentes repositórios, utilizando mecanismos de navegação ou consulta. Os dados do estudo comparativo entre o mecanismo de busca em facetas e um mecanismo de busca para *Web* indicam esse aumento.

Como contribuição, vale ressaltar a combinação entre a estrutura em facetas e o MOF, visando a busca. Através da introspecção em arquivos XMI e da disponibilização da informação em forma de facetas, o reutilizador dispõe de mais uma visão sobre os artefatos, podendo navegar sobre os repositórios de maneira a encontrar o artefato desejado para reutilização.

Após a disponibilização da MVCASE como software livre, a comunidade acadêmica passa a contar com uma plataforma para implementar e testar idéias relacionadas à tecnologia CASE, modelagem, desenvolvimento baseado em componentes, desenvolvimento de software orientado a aspectos e reutilização.

Do mesmo modo, a comunidade de desenvolvimento dispõe de uma ferramenta livre, podendo utilizá-la em seus projetos de diferentes naturezas, e inclusive modificá-la para melhor atender às suas necessidades. Dessa forma, a ferramenta poderá evoluir e se tornar cada vez mais robusta e preparada para ser utilizada em projetos reais.

A arquitetura baseada em *plug-ins* permite que novas funcionalidades sejam mais facilmente adicionadas à MVCASE. A ferramenta pode ser utilizada tanto sem nenhum *plug-in*, servindo apenas para modelagem, ou com *plug-ins* para tarefas específicas, como geração de componentes EJB e código Java. A forma de instalação dos *plug-ins* na ferramenta, que não exige nenhum tipo de configuração extra, também facilita sua utilização.

Esta pesquisa também contribuiu com o projeto do MDR. Durante a integração do MDR na MVCASE, o aluno identificou um problema com o MDR, e o notificou à equipe do projeto *Netbeans*, que o corrigiu (vide seção 4.3.2.3).

A forma de utilização do XMI em um sistema de controle de versões também é contribuição deste trabalho. Foi sugerida uma nova idéia para se resolver os problemas com o identificador único dos elementos representados no documento. A solução utilizada é simples, exigindo pouco esforço de implementação, além de proporcionar os resultados desejados.

Ainda a respeito da utilização do CVS, foram definidas tarefas automatizadas que facilitam a utilização desse tipo de repositório. Utilizando os algoritmos descritos na seção 4.3.3, o Engenheiro de Software pode utilizar um número reduzido de comandos, facilitando o compartilhamento de artefatos.

Também é contribuição deste trabalho a forma de integração entre ferramentas que foi explorada. De acordo com os níveis de integração enumerados por Wasserman, citado em (SOMMER-

VILLE, 2000), a MVCASE pode ser integrada a outras ferramentas no nível de dados, através do padrão XMI, e em um certo grau no nível de controle, através do padrão JMI e do mecanismo de comunicação em eventos utilizado na arquitetura baseada em *plug-ins*. Ambas as formas de integração foram comprovadas no caso do ambiente *Orion*.

6.3 Trabalhos futuros

Outra forma de contribuição importante em um trabalho de pesquisa é a geração de trabalhos futuros. Com relação à presente dissertação, uma série de possibilidades foi identificada.

O modo com que a MVCASE dá suporte ao desenvolvimento baseado em componentes (vide seção 3.4, e (PRADO; LUCRÉDIO, 2001; ALMEIDA et al., 2002b)) é muito semelhante às idéias da MDA (*Model-Driven Architecture* ou Desenvolvimento Orientado a Modelos) (KLEPPE; WARMER; BAST, 2003). Proposta pelo OMG, essa abordagem busca aumentar o nível de abstração no desenvolvimento, dando aos modelos importância maior do que meramente uma documentação de referência. É uma idéia recente, sendo ainda bastante discutida e pouco consolidada. Aliando-se este cenário conturbado, que estimula a pesquisa científica, à experiência com a MVCASE, uma série de trabalhos futuros pode ser identificada.

Dentre as promessas da MDA, destaca-se a rapidez de desenvolvimento, maior manutenibilidade, flexibilidade e reutilização. Porém, pouca ênfase é dada neste último item. Sendo assim, torna-se interessante um trabalho que tenha como objetivo investigar o papel da MDA em processos voltados à reutilização. Como motivação adicional, vale destacar que os principais pesquisadores da área de reutilização, como Krueger (KRUEGER, 1992), Griss (GRISS, 1995), Frakes & Isoda (FRAKES; ISODA, 1994), Jacobson et al. (JACOBSON; GRISS; JONSSON, 1997), sempre ressaltaram que os grandes benefícios oriundos da reutilização só são alcançados quando são reutilizados artefatos de alto nível de abstração, e não somente código-fonte. O presente momento, que testemunha o surgimento das idéias de desenvolvimento baseado em modelos e suas tecnologias de apoio, como a MDA, é o momento certo de se investigar uma abordagem deste tipo.

Entrando em mais detalhes na abordagem, outra possibilidade de trabalho futuro pode ser identificada. Um conceito importante na MDA é o de transformação. Consiste em automaticamente gerar um modelo específico para uma plataforma a partir de um modelo genérico, independente de plataforma. A MVCASE já implementa esta idéia para alguns modelos de componentes. Porém, o ideal seria oferecer um suporte para implementação de transformações em modelos, utilizando tecnologias como a QVT (OMG, 2002c), que é uma chamada de proposta do OMG para uma linguagem voltada à consulta e transformações envolvendo modelos. Dessa forma, organizações

poderiam desenvolver transformações específicas para sua própria plataforma.

Outro aspecto relacionado à MDA, e que é ainda bastante discutido, é o poder representativo da UML enquanto linguagem de modelagem. Alguns autores, como Dave Thomas (THOMAS, 2004) e Martin Fowler (FOWLER, 2004), defendem a idéia de que a UML não é suficiente para atender a requisitos reais de projetos. Sendo assim, uma idéia de trabalho futuro é aproveitar o suporte para modelagem da MVCASE na construção de uma meta-CASE, ou seja, um *framework* para construção de ferramentas CASE específicas, dando suporte a outro mecanismo de modelagem que não seja a UML. Dessa forma, organizações poderiam aplicar a MDA utilizando sua própria linguagem de modelagem. Seguindo as idéias de desenvolvimento baseado em linhas de produto (CLEMENTS; NORTHROP, 2002), este possível trabalho envolve a definição de quais características da MVCASE são genéricas para qualquer ferramenta de modelagem e quais outras características seriam necessárias para permitir a instanciação de outras ferramentas, além da definição completa do processo de instanciação da meta-CASE para CASEs específicas.

Fora do contexto da MDA, outra possibilidade é a investigação de um mecanismo de busca mais elaborado. Outras áreas da ciência da computação, como Recuperação da Informação e Bancos de Dados, possuem histórico e experiência na área de busca. O presente trabalho chegou a explorar essas áreas, como apresentamos em (LUCRÉDIO et al., 2003a) e (LUCRÉDIO et al., 2004). Porém, é necessário um tratamento mais completo, principalmente com relação à extração automática de informações com base em descrições textuais. Trabalhos futuros neste sentido podem oferecer importantes contribuições.

A realização de análises quantitativas dos benefícios alcançados com os serviços remotos de armazenamento e busca de artefatos de software também é um possível trabalho futuro. Os dados qualitativos apresentados nesta dissertação já são uma importante indicação destes benefícios, mas dados quantitativos podem efetivamente contribuir para aumentar essa certeza.

Finalmente, com relação à queda de desempenho observada na execução do estudo de caso (vide seção 5.5), trabalhos futuros podem pesquisar outros meios de acesso aos metadados alternativos ao MDR, ou mesmo modificar esta API para torná-la mais eficiente.

Referências

ALMEIDA, E. S. d. et al. Distributed Component-Based Software Development. In: *The 28th IEEE Annual International Computer Software and Applications Conference (COMPSAC)*. Hong Kong: IEEE/CS Press, 2004. v. 1, p. 4–9.

ALMEIDA, E. S. d.; ALVARO, A.; MEIRA, S. R. d. L. Key Developments in the Field of Software Reuse. *Submitted to the IEEE Software*, 2004.

ALMEIDA, E. S. d. et al. MVCase: An Integrating Technologies Tool for Distributed Component-Based Software Development. In: *Asia - Pacific Network Operations and Management Symposium - Poster Session*. Jeju Island, Korea: IEEE/CS Press, 2002.

ALMEIDA, E. S. d. et al. Ferramenta MVCase - Uma Ferramenta Integradora de Tecnologias para o Desenvolvimento de Componentes Distribuídos. In: *XVI Simpósio Brasileiro de Engenharia de Software - Sessão de Ferramentas*. Gramado - RS - Brazil: [s.n.], 2002.

ALVARO, A. et al. Orion-RE: A Component-Based Software Reengineering Environment. In: *WCRE 2003 - 10th Working Conference on Reverse Engineering*. Victoria - British Columbia - Canada: [s.n.], 2003. p. 248–257.

ARGOUML. *ArgoUML tool*. ArgoUML, 2005. Disponível em: <http://argouml.tigris.org>. Acesso em: 29 Mar 2005.

ASSÁO, F. M. *Extensão da MVCASE para suportar a geração do modelo de dados a partir do modelo de objetos*. [S.l.], 2003. UFSCar - Universidade Federal de São Carlos.

BARRÉRE, T. S. *CASE com Múltiplas Visões de Requisitos de Software e Implementação Automática em Java - MVCASE*. Tese (Dissertação de Mestrado) — UFSCar - Universidade Federal de São Carlos, São Carlos - SP - Brasil, 1999.

BASS, L. et al. *Volume I - Market Assessment of Component-Based Software Engineering*. [S.l.], 2000. CMU/SEI - Carnegie Mellon University/Software Engineering Institute.

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. *The Unified Modeling Language Reference Manual*. [S.l.]: Addison-Wesley, 1999.

BORLAND. *Borland Together*. Borland Software Corporation, 2005. Disponível em: <http://www.borland.com/together/>. Acesso em: Mar 2005.

BRAGA, R. M.; WERNER, C.; MATTOSO, M. Odyssey: A Reuse Environment based on Domain Models. In: *IEEE Symposium on Application-Specific Systems and Software Enginerring Technology (ASSET'99)*. Richardson, Texas: IEEE/CS Press, 1999. p. 50–57.

CATARINO, I. C. S. *Framework para Ensino a Distância via Web*. Tese (Dissertação de Mestrado) — UFSCar - Universidade Federal de São Carlos, São Carlos, São Carlos - SP - Brasil, 2002.

CLEMENTS, P.; NORTHROP, L. *Software Product Lines: Practices and Patterns*. [S.l.]: Addison-Wesley, 2002.

CONRADI, R.; WESTFECHTEL, B. Version Models for Software Configuration Management. *ACM Computing Surveys*, v. 30, n. 2, p. 232–282, 1998.

CRONHOLM, S. Why CASE Tools in information Systems Development? - an Empirical Study Concerning Motives for Investing in Case Tools. In: *18th Information Systems research In Scandinavia (IRIS 18)*. Gjern, Denmark: [s.n.], 1995.

CUNHA, J. R. D. D. *Uma Ferramenta de Apoio ao Processo de Gerenciamento de Configuração de Software*. Tese (Dissertação de Mestrado) — UFSCar - Universidade Federal de São Carlos, São Carlos - SP - Brazil, 2005.

CVS. *CVS - Concurrent Versions System*. CVS, 2005. Disponível em: <http://www.cvshome.org>. Acesso em: Mar 2005.

DEMICHIEL, L. G. *Enterprise JavaBeans Specification, Version 2.1*. [S.l.], 2002. Sun Microsystems.

DIRCKZE, R. *Java Metadata Interface (JMI) Specification*. [S.l.], 2002. Java Community Process.

D'SOUZA, D.; WILLS, A. *Objects, Components and Frameworks with UML: The Catalysis Approach*. [S.l.]: Addison-Wesley, 1999. (Object Technology Series).

ECLIPSE. *Eclipse Platform*. 2004. Disponível em: <http://www.eclipse.org/>. Acesso em: 17 June 2004.

FLORIJN, G.; MEIJERS, M.; WINSEN, P. v. Tool Support for Object-Oriented Patterns. In: *European Conference on Object-Oriented Programming (ECOOP'97)*. Jyväskylä, Finland: Springer-Verlag, 1997. p. 472–495.

FOWLER, M. *Model Driven Architecture*. 2004. Disponível em: <http://martinfowler.com/bliki/ModelDrivenArchitecture.html>. Acesso em: December, 2004.

FRAKES, W. B.; ISODA, S. Success Factors of Systematic Software Reuse. *IEEE Software*, v. 11, n. 01, p. 14–19, 1994.

FRAKES, W. B.; POLE, T. P. An Empirical Study of Representation Methods for Reusable Software Components. *IEEE Transactions on Software Engineering*, v. 20, n. 8, 1994.

FUGGETTA, A. A Classification of CASE Technology. *IEEE Computer*, v. 26, n. 12, p. 25–38, 1993.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

GARCIA, V. C. et al. Uma Ferramenta CASE para o Desenvolvimento de Software Orientado a Aspectos. In: *XVIII Simpósio Brasileiro de Engenharia de Software - Sessão de Ferramentas*. Brasília - DF - Brasil: [s.n.], 2004.

GENTLEWARE. *Poseidon for UML*. Gentleware, 2005. Disponível em: <http://www.gentleware.com (29 Mar 2005)>. Acesso em: 29 Mar 2005.

GRISS, M. Making Software Reuse Work at Hewlett-Packard. *IEEE Software*, v. 12, n. 01, p. 105–107, 1995.

GRUNDY, J. Storage and retrieval of Software Components using Aspects. In: *2000 Australasian Computer Science Conference*. Canberra, Australia: IEEE/CS Press, 2000. p. 95–103.

GUOJUN, L. *Communication and Computing for Distributed Multimedia Systems*. [S.l.]: Artech House, 1996.

HALL, P. Architecture-driven component reuse. *Information and Software Technology*, v. 41, n. 14, p. 963–948, 1999.

HARRISON, W.; OSSHER, H.; TARR, P. Software Engineering Tools and Environments: A Roadmap. In: *The Future of Software Engineering*. New York: ACM, 2000. p. 261–277.

HENNINGER, S. Using Iterative Refinement to Find Reusable Software. *IEEE Software*, v. 11, n. 5, p. 48–59, 1994.

HENNINGER, S. An Evolutionary Approach to Constructing Effective Software Reuse Repositories. *ACM Transactions on Software Engineering and Methodology*, v. 6, n. 2, p. 111–140, 1997.

HUMMEL, O.; ATKINSON, C. Extreme Harvesting: Test Driven Discovery and Reuse of Software Components. In: *IEEE International Conference on Information Reuse and Integration (IRI - 2004)*. Las Vegas, NV, USA: IEEE Systems, Man, and Cybernetics Society (SMC), 2004. p. 66–72.

IBM. *Rational Rose XDE Developer Plus*. IBM, 2004. Disponível em: <http://www-306.ibm.com/software/awdtools/developer/plus/>. Acesso em: 17 June 2004.

IEEE. *IEEE Standard Glossary of Software Engineering Terminology*. [S.l.], 1990. The Institute of Electrical and Electronics Engineers.

ISAKOWITZ, T.; KAUFFMAN, R. J. Supporting Search for Reusable Software Objects. *IEEE Transactions on Software Engineering*, v. 22, n. 6, 1996.

ISO. *Quality Systems - Model for Quality assurance in design/development, production, installation and servicing: ISO 9001*. [S.l.], 1987. International Organization for Standardization.

ITAMI, S. *Proposta de um Esquema de Controle de Elementos de Software para a Abordagem Orientada a Objetos*. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos - SP, 1997.

JACOBSON, I.; GRISS, M.; JONSSON, P. *Reuse-driven Software Engineering Business (RSEB)*. [S.l.]: Addison-Wesley, 1997.

JBOSS. *JBoss*. 2005. Disponível em: <http://www.jboss.org/index.html>. Acesso em: Mar 2005.

KICZALES, G. An Overview of AspectJ. In: *15th European Conference on Object-Oriented Programming (ECOOP'2001)*. Budapest, Hungary: Lecture Notes in Computer Science (LNCS), 2001. v. 2072, p. 327–353.

KLEPPE, A.; WARMER, J.; BAST, W. *MDA Explained - The Model Driven Architecture: Practice and Promise*. [S.l.]: Addison-Wesley, 2003. (Object Technology Series).

KRUEGER, C. Software Reuse. *ACM Computing Surveys*, v. 24, n. 02, p. 131–183, 1992.

LUCRÉDIO, D. et al. Orion - A Component Based Software Engineering Environment. *JOT - Journal of Object Technology*, v. 3, n. 4, p. 51–74, 2004.

LUCRÉDIO, D.; ALMEIDA, E. S. d.; PRADO, A. F. d. A Survey on Software Components Search and Retrieval. In: STEINMETZ, R.; MAUTHE, A. (Ed.). *30th IEEE EUROMICRO Conference, Component-Based Software Engineering Track*. Rennes - France: IEEE/CS Press, 2004. p. 152–159.

LUCRÉDIO, D. et al. Abordagens para Recuperação Eficiente de Componentes utilizando Indexação Métrica. In: *3o. WDBC - Workshop De Desenvolvimento Baseado em Componentes*. São Carlos - SP - Brazil: [s.n.], 2003.

LUCRÉDIO, D. et al. MVCASE Tool - Working with Design Patterns. In: *SugarLoafPLoP'2003 - The Third Latin American Conference On Pattern Languages Of Programming*. Porto de Galinhas - PE - Brazil: [s.n.], 2003.

LUCRÉDIO, D. et al. Component retrieval using metric indexing. In: *The 2004 IEEE International Conference on Information Reuse and Integration (IEEE IRI-2004)*. Las Vegas - USA: IEEE / CS Press, 2004.

LÜER, C.; ROSENBLUM, D. WREN: An Environment for Component-Based Development. *ACM SIGSOFT Software Engineering Notes*, v. 26, n. 5, p. 207–217, 2001.

MAAREK, Y. S.; BERRY, D. M.; KAISER, G. E. An Information Retrieval Approach for Automatically Constructing Software Libraries. *IEEE Transactions on Software Engineering*, v. 17, n. 8, 1991.

MCILROY, M. D. 'Mass Produced' Software Components. In: *NATO Software Engineering Conference*. [S.l.: s.n.], 1968. p. 138–155.

MICROSOFT. *Microsoft Component Object Model (COM) Technology*. Microsoft Corporation, 2005. Disponível em: <http://www.microsoft.com/com/default.asp>. Acesso em: April 2005.

MORAES, J. L. C. d. *Reutilizacão de Componentes de um Framework do Domínio de Cardiologia*. Tese (Dissertação de Mestrado (em andamento)) — UFSCar - Universidade Federal de São Carlos, São Carlos - SP - Brazil, 2004.

NETBEANS. *Javacvs*. NetBeans project, 2005. Disponível em: <http://javacvs.netbeans.org (29 Mar 2004)>. Acesso em: 29 Mar 2005.

NETBEANS. *MDR - MetaData Repository*. NetBeans project, 2005. Disponível em: <http://mdr.netbeans.org (29 Mar 2005)>. Acesso em: 29 mar 2005.

NETBEANS. *The Netbeans project*. NetBeans project, 2005. Disponível em: <http://www.netbeans.org (29 Mar 2005)>. Acesso em: 29 Mar 2005.

NETO, R. M. d. S. et al. Component-Based Software Development Environment (CBDE). In: *6th ICEIS - International Conference on Enterprise Information Systems*. Porto - Portugal: [s.n.], 2004. p. 338–343.

NOVAIS, E. R. A. *Reengenharia de Software Orientada a Componentes Distribuídos*. Tese (Dissertação de Mestrado) — UFSCar - Universidade Federal de São Carlos, São Carlos - SP - Brazil, 2002.

OLIVEIRA, H.; MURTA, L.; WERNER, C. Odyssey-VCS: Um Sistema de Controle de Versões para Modelos Baseados no MOF. In: *XVIII Simpósio Brasileiro de Engenharia de Software - XI Sessão de Ferramentas*. Brasília - DF - Brasil: [s.n.], 2004.

OMG. *Common Object Request Broker Architecture: Core Specification*. [S.l.], 2002. Object Management Group.

OMG. *Meta Object Facility (MOF) Specification*. [S.l.], 2002. Object Management Group.

OMG. *Request for Proposal: MOF 2.0 Query / Views / Transformations RFP*. [S.l.], 2002. Object Management Group.

OMG. *XML Metadata Interchange (XMI) Specification*. [S.l.], 2002. Object Management Group.

OMG. *UML 2.0 Diagram Interchange Specification*. [S.l.], 2003. Object Management Group.

OMG. *Unified Modeling Language Specification*. [S.l.], 2003. Object Management Group.

OMONDO. *Eclipse UML Tool*. Omondo, 2005. Disponível em: <http://www.eclipseuml.com/>. Acesso em: Febryary 2005.

PAULK, M. C. et al. *Capability Maturity Model for Software, Version 1.1*. [S.l.], 1993. CMU/SEI - Carnegie Mellon University/Software Engineering Institute.

PIVETA, E. K.; ZANCANELLA, L. C. Observer Pattern using Aspect-Oriented Programming. In: *The Third Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP 2003)*. Porto de Galinhas - PE - Brazil: [s.n.], 2003.

PODGURSKI, A.; PIERCE, L. Retrieving Reusable Software By Sampling Behavior. *ACM Transactions on Software Engineering and Methodology*, v. 2, n. 3, p. 286–303, 1993.

PRADO, A. F. d.; LUCRÉDIO, D. MVCASE: Ferramenta CASE Orientada a Objetos. In: *XIV Simpósio Brasileiro de Engenharia de Software - Sessão de Ferramentas*. João Pessoa - PB - Brazil: [s.n.], 2000.

PRADO, A. F. d.; LUCRÉDIO, D. Ferramenta MVCASE - Estágio Atual: Especificação, Projeto e Construção de Componentes. In: *XV Simpósio Brasileiro de Engenharia de Software - Sessão de Ferramentas*. Rio de Janeiro - RJ - Brazil: [s.n.], 2001.

PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. [S.l.]: McGraw-Hill, 2001.

PRIETO-DÍAZ, R. Implementing Faceted Classification for Software Reuse. *Communications of the ACM*, v. 34, n. 5, 1991.

RAVICHANDRAN, T.; ROTHENBERGER, M. A. Software Reuse Strategies and Component Markets. *Communications of the ACM*, v. 46, n. 8, p. 109–114, 2003.

SAX. *SAX 2.0: The Simple API for XML*. The SAX Project, 2005. Disponível em: <http://www.saxproject.org/>. Acesso em: 29 Mar 2005.

SEACORD, R. C. Software Engineering Component Repositories. In: *International Workshop on Component-Based Software Engineering, Held in conjunction with the 21st International Conference on Software Engineering (ICSE)*. Los Angeles, CA, USA: [s.n.], 1999.

SEACORD, R. C.; HISSAM, S. A.; WALLNAU, K. C. *Agora: A Search Engine for Software Components*. [S.l.], 1998. CMU/SEI - Carnegie Mellon University/Software Engineering Institute.

SILVA, D. *Uma Ferramenta para Descoberta de Conhecimento com Suporte de Data Warehousing e sua Aplicação para Acompanhamento do Aluno em Educação a Distância*. Tese (Dissertação de Mestrado) — UFSCar - Universidade Federal de São Carlos, São Carlos, São Carlos - SP - Brasil, 2002.

SILVA, D. Using Data Warehouse and Data Mining Resources for Ongoing Assessment of Distance Learning. In: *IEEE International Conference on Advanced Learning Technologies*. Kazan, Tatarstan, Russia: IEEE/CS Press, 2002.

SILVA, D.; VIEIRA, M. T. P. An Ongoing assessment model in distance learning. In: *Internet and Multimedia Systems and Applications*. Honolulu, USA: [s.n.], 2001.

SILVA, E. A. d. *Tutorial - Reutilização de Componentes (Projeto DBCM)*. 2004. Projeto DBCM - Desenvolvimento Baseado em Componentes Multimídia - DC - USFCar.

SOMMERVILLE, I. *Software Engineering*. 6. ed. [S.l.]: Pearson Education, 2000.

THOMAS, D. MDA: Revenge of the Modelers or UML Utopia? *IEEE Software*, v. 21, n. 3, p. 15–17, 2004.

UFSCAR. *DBCM - Desenvolvimento Baseado em Componentes Multimídia*. 2004. Projeto financiado pelo CNPq. Vigência: 12/2003 a 12/2004.

UNISYS. *JMI-RI Documentation - CIM Version 1.3*. [S.l.], 2002. Unisys Corporation.

VILLELA, R. *Busca e Recuperação de Componentes em Ambientes de Reutilização de Software*. Tese (Tese de doutorado) — Universidade Federal do Rio de Janeiro, Rio de Janeiro - RJ - Brazil, 2000.

W3C. *Document Object Model (DOM) Level 3 Core Specification*. World Wide Web Consortium, 2004. Disponível em: <http://www.w3.org/DOM/>. Acesso em: 17 June 2004.

W3C. *Extensible Markup Language (XML)*. World Wide Web Consortium, 2005. Disponível em: <http://www.w3.org/XML/>. Acesso em: 29 Mar 2005.

WEBER, K. C.; NASCIMENTO, C. J. d. Brazilian Software Quality in 2002. In: *ICSE 2002 - 24th International Conference on Software Engineering*. Orlando, Florida, USA: [s.n.], 2002. p. 634–638.

YAMAMOTO, C. H.; BIAJIZ, M.; TRAINA, C. Aumento da Eficiência das Estruturas de Indexação Métricas com Uso de Conceitos da Lógica Nebulosa. In: *XVIII Simpósio Brasileiro de Banco de Dados (SBBD 2003)*. Manaus - AM - Brazil: [s.n.], 2003. p. 185–199.

YE, Y.; FISCHER, G. Supporting Reuse By Delivering Task-Relevant and Personalized Information. In: *ICSE 2002 - 24th International Conference on Software Engineering*. Orlando, Florida, USA: [s.n.], 2002. p. 513–523.

ZHUGE, H. A problem-oriented and rule-based component repository. *The Journal of Systems and Software*, v. 50, p. 201–208, 2000.