UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

C-CORE: Uma Ferramenta de Programação para Construção e Reuso de Componentes

Raphael Marcilio de Souza Neto

São Carlos
Maio/2005

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

S719af

Souza Neto, Raphael Marcilio de.
C-CORE : uma Ferramenta de programação para construção e reuso de componentes / Raphael Marcilio de Souza Neto. -- São Carlos : UFSCar, 2007.
73 f.

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

1. Desenvolvimento de software baseado em componentes. 2. Ferramenta de desenvolvimento. 3. Catalysis/UML. 4. Componentes de software. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"C-CORE (Component Construction and Reuse): uma Ferramenta de Programação para Construção e Reuso de Componentes"

RAPHAEL MARCILIO DE SOUZA NETO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Prof. Dr. Antonio Francisco do Prado
(Orientador - DC/UFSCar)

Prof. Dr. Mauro Biajiz
(DC/UFSCar)

Profa. Dra. Silvia Regina Vergílio
(INF/UFPR)

São Carlos
Maio/2005

Agradecimentos

A Deus pela oportunidade de estudar.

Aos meus pais e irmãos pelo apoio e incentivo.

Ao meu orientador Antonio Francisco do Prado pela orientação dada durante todo o decorrer deste trabalho.

Aos colegas de trabalho pela troca de experiências.

Finalmente a todos aqueles que colaboraram direta ou indiretamente na realização deste.

Obrigado.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

O Desenvolvimento de Software compreende diversas atividades que nem sempre são suportadas por ferramentas que auxiliam o Engenheiro de Software e automatizam grande parte de suas atividades. Diferentes pesquisas têm sido realizadas e muitos recursos têm sido gastos com o objetivo de construir ferramentas que apóiam o Processo de Desenvolvimento de Software. Dentre essas ferramentas, destacam-se aquelas que oferecem apoio às atividades de projeto e implementação, e que são integradas com ferramentas que auxiliam outras atividades do processo de construção de software. Com essas ferramentas, pode-se obter uma maior consistência dos artefatos produzidos ao longo de todo o ciclo de desenvolvimento de software. Outro ponto importante do processo de desenvolvimento relaciona-se com a obtenção de software com melhor qualidade e menor custo. Uma das áreas que vem se destacando na produção de software com qualidade e menor custo é a que pesquisa o reuso de componentes de software. Assim, vem ganhando destaque as ferramentas de programação orientadas a componentes de software integradas com ferramentas de modelagem que visam principalmente a produtividade de software baseada no reuso. Motivados por estas idéias, este trabalho apresenta uma ferramenta de suporte ao projeto e implementação de componentes e suas aplicações, denominada C-CORE, integrada com outra ferramenta de modelagem, denominada MVCASE.

Abstract

The Software Development contains several activities that are not always supported by tools that aid the Software Engineer and automate great part of his activities. Different researches have been accomplished and many resources have been spent aiming to construct tools that support the Software Development Process. Among these tools, stand out those that support project and implementation activities, and that are integrated with tools that support other activities of the software construction process. With those tools, it can be obtained a larger consistence of the results that are produced along the whole software development cycle. Another important point of the development process is related to obtain a software with better quality and less cost. One of the areas that is highlighted in the software production with quality and less cost, is those that researches the software components reuse. Thus, the Software Component Oriented Programming Tools integrated with modeling tools that aim mainly to improve the software productivity based on reuse is standing out. Motivated by these ideas, this work presents a tool denominated C-CORE, that supports the project and implementation of components and their applications integrated with other modeling tool, denominated MVCASE.

Capítulo 1

Introdução

O esforço da comunidade de software e pesquisadores da área em desenvolver produtos de qualidade e que satisfaçam os requisitos exigidos da atual indústria de software tem motivado a construção de ferramentas de engenharia de software apoiadas por computador (*CASE – Computer-Aided Software Engineering*) [Pressman 2001] que auxiliem na construção de software com melhor qualidade e menor custo. Essas ferramentas, que auxiliam os Engenheiros de Software, permitem que atividades específicas do desenvolvimento de software passíveis de automatização sejam realizadas. Inúmeras ferramentas, construídas por diferentes fabricantes, têm surgido para apoiar essas atividades e para melhor identificar onde tais ferramentas podem ser aplicadas no processo de Engenharia de Software, uma classificação foi definida, permitindo agrupá-las segundo uma taxonomia que considera, principalmente suas origens e custos [Pressman, 2001; QED, 1989]. Estas ferramentas também podem ser categorizadas quanto à fase do processo de desenvolvimento a qual apóiam [Pressman, 2001]. Nesta última categorização, as ferramentas de análise e projeto e de programação têm sido bastante empregadas para obter um software com qualidade e com menor custo.

As ferramentas de análise e projeto permitem ao engenheiro de software modelar sistemas representando-os com diferentes técnicas que consideram dentre outros elementos, seus dados, funções e arquitetura. Estas ferramentas têm uma abordagem metodológica onde o software é o produto de uma seqüência racional de passos, produzindo artefatos em diferentes níveis de abstração, principalmente os de Requisitos, Análise e Projeto.

Por outro lado, dentre as ferramentas que dão suporte à fase de Implementação destacam-se os tradutores, compiladores e montadores. Todavia, como os computadores ficaram mais poderosos e os softwares ficaram maiores e mais complexos, a abrangência dessas ferramentas foi expandida. Uma dessas expansões ocorreu com o uso de sistemas "time-sharing" para o desenvolvimento de software, o que encorajou o desenvolvimento de editores de programas, depuradores, analisadores de código e geradores de relatório [Gane, 1990; Maia, 1999]. Outro impulso aconteceu com o surgimento do conceito de composição de

sistemas de software. Este conceito enfoca o uso de componentes [Pressman 2001], que é uma parte não-trivial de um sistema, praticamente independente e substituível, que preenche uma função clara no contexto de uma arquitetura bem-definida. Seguindo esta idéia, surgiram as ferramentas de programação que auxiliam a programação orientada a componentes, disponibilizando ao engenheiro de software, componentes de diferentes domínios e um meio eficiente de reutilizá-los. A disponibilidade de componentes de software nessas ferramentas é uma excelente forma de melhorar a qualidade no desenvolvimento de software principalmente na construção e reuso em diferentes domínios. Cita-se como exemplo, domínio GUI (*Graphical User Interface*), o qual possibilita a construção rápida de interfaces gráficas, através do reuso de componentes e da geração automática de código. Em outros domínios, como Banco de Dados e *Web*, essas ferramentas disponibilizam componentes e recursos que facilitam o reuso no desenvolvimento de aplicações que usam Banco de Dados, e acessam a Internet. Vários benefícios decorrem do reuso de componentes de software: aumento da produtividade; maior segurança, considerando que os mesmos já foram previamente testados; redução das linhas de código, visto que a experiência e a funcionalidade embutida nos componentes minimizam a ocorrência de erros nas diferentes fases do desenvolvimento, facilitando principalmente a manutenção; entre outros.

Assim, as ferramentas de análise e projeto e de programação provêem, em conjunto, suporte para as fases de desenvolvimento do software desde a Identificação e Especificação dos requisitos até sua Implementação e Teste. Entretanto, mesmo com o suporte a essas fases de desenvolvimento de software, ainda há carência de recursos que melhorem a qualidade do software e aumentem sua produtividade. No desenvolvimento de software ocorrem inconsistências entre os artefatos nas diferentes fases do seu ciclo de vida, e estas inconsistências são detectadas com apoio de métodos, técnicas e ferramentas que auxiliam o engenheiro de software durante todo o processo de desenvolvimento. As correções de inconsistências verificadas nas diferentes fases de desenvolvimento de software são bastante onerosas. Refinar e validar um componente de software ou aplicação antes destas serem entregues ao cliente constituem ações que normalmente não são ainda muito bem suportadas pelos ambientes de desenvolvimento.

Assim, por exemplo, no caso das ferramentas de programação que não estão integradas com outras ferramentas de modelagem, fica difícil manter uma documentação do sistema consistente com sua implementação.

Embora hoje existam diferentes ferramentas de programação como, por exemplo, JBuilder [Borland, 2003], NetBeans [NetBeans, 2003], Eclipse [Eclipse, 2003], estas muitas vezes não estão integradas para apoiar outras etapas do Processo de Desenvolvimento de Software que incluem, por exemplo, as fases de análise e projeto. Por exemplo, no caso do desenvolvimento baseado em componentes, esta integração é importante porque oferece mecanismos para: auxiliar na correção de inconsistências entre as fases de desenvolvimento de componentes e suas aplicações; e validar o produto em um ambiente de execução, que suporta a detecção de falhas e suas correções. Outro aspecto é que muitas vezes quando estão integradas, estas ferramentas não são gratuitas e apresentam restrições de funcionalidade e limitações do uso.

Assim, motivados em facilitar o processo de desenvolvimento de software, esta pesquisa apresenta uma ferramenta de programação orientada a componentes, denominada C-CORE (*Component Construction and Reuse*) [Souza Neto et al, 2004b], que apóia grande parte das tarefas de construção de componentes de software e reuso destes componentes por suas aplicações. Além de apoiar a fase de programação a C-CORE está integrada com a ferramenta de modelagem MVCASE [MVCASE, 2003], o que possibilita uma modelagem orientada para a implementação, reduzindo as inconsistências entre os artefatos de modelagem e de implementação. A ferramenta C-CORE também, assim como a ferramenta MVCASE, encontra-se introduzida no Ambiente de Desenvolvimento de Software Baseado em Componentes (ADSBC), provendo suporte para o desenvolvimento, manutenção e melhorias tanto de componentes como de suas aplicações.

A apresentação deste trabalho está organizada em 06 (seis) Capítulos, além da Introdução. No Capítulo 2, apresentam-se os principais conceitos e tecnologias envolvidas no desenvolvimento deste trabalho. No Capítulo 3, apresenta-se o ambiente ADSBC onde a C-CORE está integrada. No Capítulo 4, apresenta-se a ferramenta C-CORE. No Capítulo 5, têm-se os estudos de caso desenvolvidos com ferramenta C-CORE. Finalmente, no Capítulo 6 têm-se as Conclusões.

Capítulo 2

Principais Conceitos e Tecnologias

Neste capítulo, são abordados os principais conceitos e tecnologias estudados para o desenvolvimento desta pesquisa. Dentre estes conceitos e tecnologias destacam-se os relacionados com Componentes de Software, e suas tecnologias de implementação, como *Enterprise JavaBeans* (EJB) e *JavaBeans*, utilizadas pela ferramenta C-CORE. Outros conceitos relacionam-se com o método *Catalysis* para o Desenvolvimento de Software Baseado em Componentes (DBC), adotado na construção da C-CORE. Também foram realizados estudos sobre algumas ferramentas de programação orientadas a componentes que auxiliaram na identificação dos requisitos da C-CORE.

2.1 Componentes de Software

Atualmente, sistemas complexos e de alta qualidade precisam ser construídos em períodos de tempo extremamente curtos. Isso implica na necessidade de uso de uma abordagem mais organizada de reuso. A Engenharia de Software Baseada em Componentes é um processo que enfatiza o projeto e a construção de sistemas baseados em computador usando "componentes" de software reusáveis [Pressman, 2001].

Na literatura há várias definições do termo componente de software. Segundo Booch e outros [Booch et. al, 1999], componentes são partes físicas e substituíveis de um sistema que fornece a realização de um conjunto de interfaces.

Sametinger [Sametinger, 1997] define "componentes reusáveis" como artefatos autocontidos, claramente identificáveis, os quais realizam uma função específica e têm interfaces claras, em conformidade com um dado modelo de arquitetura de software, documentação apropriada e um grau de reuso definido.

Um melhor entendimento da definição dada por Sametinger pode ser alcançado através de uma discussão mais aprofundada dos termos indicados por sua definição [Spagnoli & Becker, 2003]:

- **Autocontido**: característica dos componentes de poderem ser reusados sem a necessidade de incluir/depender de outros componentes. Caso exista alguma dependência, então todo o conjunto deve ser visto como o componente reutilizável;
- **Identificação**: componentes devem ser facilmente identificados, ou seja, devem estar contidos em um único lugar ao invés de espalhados e misturados com outros artefatos de software ou documentação. Os componentes são tratados como artefatos por poderem assumir uma variedade de formas como, por exemplo, código fonte, documentação e código executável;
- **Funcionalidade**: componentes têm uma funcionalidade clara e específica que eles realizam;
- **Interface**: componentes devem ter uma interface clara que indica como podem ser reusados e conectados a outros componentes, e devem ocultar os detalhes que não são necessários para o reuso;
- **Documentação**: a existência de documentação é indispensável para o reuso. O tipo de componente e sua complexidade irão indicar a conveniência do tipo de documentação; e
- **Condição de reuso**: deve ser mantida a condição de reuso do componente que compreende diferentes informações como, por exemplo, quem é o proprietário, quem deve ser contatado em caso de problemas, qual é a situação de qualidade, entre outras.

D´Souza e Wills [D'Souza & Wills, 1998] definem um componente como um pacote coerente de software, desenvolvido independentemente e disponibilizado como uma unidade que possui interfaces, pelas quais o componente pode ser composto por outros componentes para prover e utilizar serviços.

Para Werner e Braga [Werner & Braga, 2000], os componentes são partes operacionais de software os quais desempenham totalmente suas funções, sendo empacotados com o objetivo de prover conjuntos de serviços acessíveis apenas através de uma interface bem definida.

Outra taxonomia classifica os componentes em dois grupos principais [Szyperski, 1998; CBSE, 2003]:

- **Componentes de negócio:** São aqueles reconhecidos pelo domínio da aplicação os quais estão inseridos, contendo as regras do negócio. São exemplos: Categoria, Cursos e Aluno, para um domínio de Educação a Distância (EAD);
- **Componentes de infra-estrutura**: São os que fornecem suporte aos componentes de negócio. São exemplos: segurança, distribuição, persistência em banco de dados e serviços da Internet (POP3, SMTP, dentre outros).

Com base nos conceitos de componentes descritos acima, define-se nesta pesquisa componentes como sendo: unidades de código, previamente testadas, que expõem suas funcionalidades por intermédio de interfaces bem definidas, e que são empacotadas em um arquivo para distribuição, podendo ser desenvolvidos segundo uma linguagem de programação e reusados por aplicações em diferentes domínios.

2.2 As Tecnologias *Enterprise JavaBeans* e *JavaBeans*

A Sun Microsystems [Sun Microsystems, 2003] disponibiliza duas tecnologias para construção de componentes, implementados com a linguagem Java: *Enterprise JavaBeans* e *JavaBeans* apresentadas a seguir.

2.2.1 A Tecnologia *Enterprise JavaBeans*

A tecnologia *Enterprise JavaBeans* (EJB) é um modelo de servidor de componentes para Java. Permite a criação de aplicações *multi-tier* (multi-camadas), que requerem serviços de gerenciamento de transações, segurança, conectividade e acesso a banco de dados [Valeskey, 1999; Raj, 2003]. Em EJB, as regras de negócio são implementadas em componentes disponibilizados em um servidor EJB, ficando independentes de plataforma.

Um sistema em EJB é dividido em três camadas físicas: Máquina do Cliente, Servidor EJB e Servidor de Banco de Dados, ou outro mecanismo de persistência conforme mostra a Figura 1 [Fukuda, 2000]. Estas camadas podem ou não residir em uma mesma máquina.

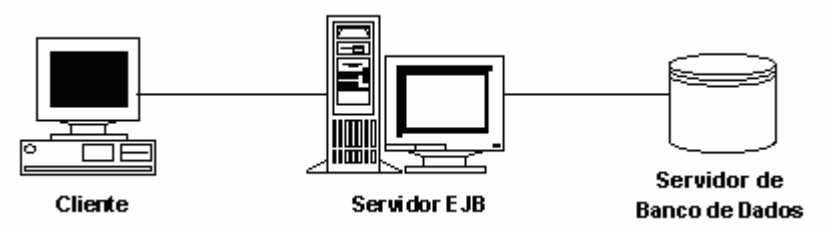


Figura 1 - Arquitetura do Sistema EJB

Um componente pode ser do tipo *Session Bean* ou *Entity Bean*. Os *Session beans* podem ser: *Stateless* e *Stateful*. Os *Stateless* são componentes que não possuem um estado interno e não mantêm qualquer informação de um método que invoca outro. Os *Stateful* são componentes que possuem um estado interno, que muda durante a sua execução, de acordo com suas variáveis de instância. Podem realizar uma tarefa para um cliente, e são criados e destruídos pelo próprio cliente. Cada componente *Session Stateful* permite o acesso ao Servidor EJB, de apenas um Cliente, tendo o tempo de vida de acordo com a sessão do cliente.

Um componente *Session Bean* pode ser acessado localmente ou remotamente por um cliente. Para acesso local devem-se usar as interfaces "LocalHome" e "Local". Para acesso remoto devem-se usar as intefaces "Home" e "Remote". As Interfaces "Home" e "LocalHome" servem como uma fábrica de componentes. Elas disponibilizam métodos para criar, localizar e destruir instâncias do componente.

Os componentes *Entity Beans* representam objetos em depósitos de dados persistentes. Cada componente *Entity* é associado a uma "tupla" de uma "relação" de um Banco de Dados Relacional. Os componentes *Entity* permitem o acesso compartilhado por muitos clientes, permanecendo ativo por tempo indeterminado, mantendo seu estado no banco de dados após sua utilização. Da mesma forma que os *Session Beans*, os componentes *Entity* também podem ser acessados localmente e remotamente por meio de suas interfaces.

A persistência do componente *Entity* no banco de dados pode ser realizada de duas formas: *Container-Managed Persistence* (CMP), gerenciado pelo EJB *Container*, ou *Bean Managed Persistence* (BMP), quando a persistência é delegada ao próprio componente. Os componentes *Entity CMP* ficam na camada intermediária, dentro de um *Container*, que reside no Servidor EJB, podendo acessar um Banco de Dados, residente na terceira camada, diretamente, via *Java DataBase Connectivity* (JDBC), ou indiretamente pelo *Container*, retornando uma resposta para o Cliente na primeira camada.

O EJB *Container* é o ambiente de execução dos componentes *EJBs*. Clientes utilizam o serviço *Java Naming and Directory Interface* (JNDI), para localizar a Interface "Home" desses componentes. Quando um Cliente acessa um componente EJB, o serviço JNDI retorna uma referência para o objeto implementado da interface "Home". O acesso local não usa o serviço JNDI, e sim uma referência direta para a interface "LocalHome".

2.2.2 A Tecnologia *JavaBeans*

Um *JavaBean* é um componente de software reutilizável que é escrito na linguagem de programação Java. Componentes *JavaBeans* podem ser reutilizados em diferentes aplicações, como por exemplo, *Servlets* [Sun Microsystems, 2003], *JSPs* [Sun Microsystems, 2003], *Applets* [Sun Microsystems, 2003], *Frames* [Sun Microsystems, 2003], entre outras.

Em ambientes como JBuilder, NetBeans, Eclipse, *JavaBeans* têm sido utilizado principalmente para tornar disponível componentes do domínio GUI (*Graphical User Interface*), Banco de Dados, e de outros domínios cujos componentes estão presentes nas aplicações em geral.

Nestes ambientes, o reuso de componentes *JavaBeans* ocorre através de sua interface de *designtime* e *runtime*. A interface de *designtime* disponibiliza métodos e eventos associados ao componente em tempo de *design* da aplicação. Esta interface permite ao Engenheiro de Software a construção interativa de uma aplicação, pois o componente é personalizado à medida que engenheiro de software efetua configurações no mesmo. Desse modo, pode-se acompanhar parte do comportamento da aplicação sendo construída. A interface de *runtime* disponibiliza métodos e eventos para utilização em tempo de execução da aplicação.

A arquitetura de componentes *JavaBeans* foi projetada para construção independente de componentes de software no qual programadores montam grandes componentes de aplicação, podendo ser compostos ou conectados a outros componentes *JavaBeans*.

Os componentes JavaBeans [Sun Microsystems, 2003] apresentam características como, por exemplo, Propriedades, Eventos, Introspecção, Customização, Persistência e Empacotamento em arquivos JAR (*Java ARchive*). Para se comunicarem, estes componentes utilizam o modelo de eventos denominado *Push* [Costa Neto, 2001]. Segundo este modelo (Figura 2), um *Source* gera eventos que notificam imediatamente todos os *Listeners* registrados. É mostrado na Figura 2 o funcionamento do modelo *Push*.

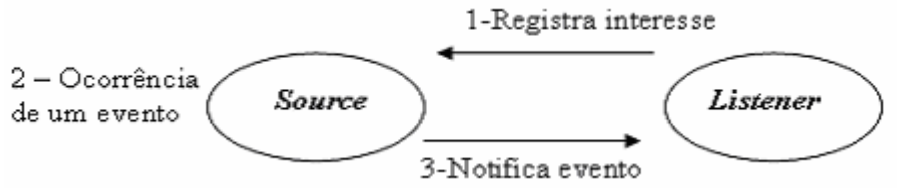


Figura 2 - Funcionamento do modelo Push [Costa, 2001]

Conforme a Figura 2, inicialmente um *Listener* registra interesse pela notificação de eventos ocorridos com um *Source* que, na ocorrência de um evento, notifica o *Listener* registrado.

Para suporte a este modelo, tem-se o pacote java.bean [Sun Microsystems, 2003] com classes e interfaces para o tratamento de eventos. Em especial têm-se as classes PropertyChangeSupport (usada para suporte a propriedades *bound*) e VetoableChangeSupport (suporte a propriedades *constrained*) para permitir o registro de *Listeners* interessados em receber notificações de eventos de um *Source*, e permitir que o *Source* solicite a notificação dos *Listeners* registrados.

O Modelo de Eventos segue o padrão de projeto *Observer* [Gamma et al, 1995], o qual define uma dependência entre objetos, tal que quando há mudança no estado de um objeto, todos os seus dependentes são notificados e atualizados automaticamente. Os termos *Source* e *Listener* são usados para descrever as fontes e os receptores de eventos. Um *Source* pode ser associado a qualquer quantidade de *Listeners*. A Figura 3 apresenta o modelo de eventos do componente "Pessoa", especificado segundo o padrão *Observer*, onde "PessoaEvent" e "PessoaListener" representam *Source* e *Listener*, respectivamente.

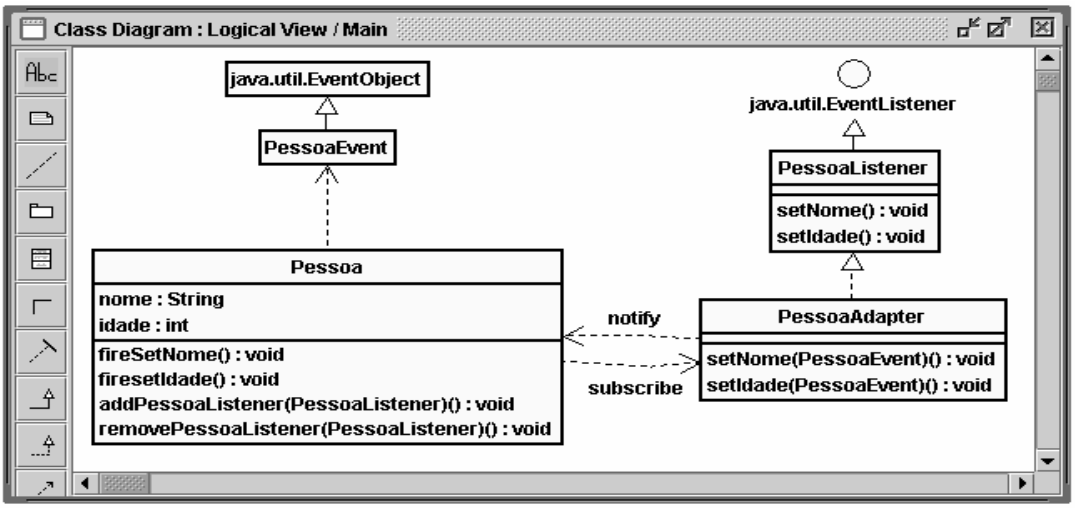


Figura 3 - Modelo de Eventos do componente Pessoa

Ferramentas de programação baseada em componentes podem suportar a personalização de um componente tornando disponível suas propriedades para alteração, em tempo de *design*. A personalização permite ao programador alterar a aparência e o comportamento do componente, com o uso de editores de propriedades. A figura 4 mostra, por exemplo, o reuso do componente Pessoa numa ferramenta de programação orientada a componentes como o JBuilder. À direita, a Figura mostra a janela de edição de propriedades. Ao se selecionar um componente, no caso Pessoa, esta janela atualiza os guias de propriedades e os eventos do componente. No guia de propriedades, são apresentadas as propriedades do componente que podem ser alteradas em tempo de *design* através do editor de propriedades e em tempo de *runtime* por meio da invocação de métodos. No guia de eventos, são apresentados os eventos gerados pelo componente selecionado. Para cada evento pode ser especificado um método a ser chamado quando ocorrer o evento.

Figura 4 - Reuso do componente Pessoa

Na personalização de um componentes *JavaBeans*, usa-se a serialização para armazenar e recuperar seu estado, após alterações de suas propriedades.

Depois de construídos, os componentes *JavaBeans* podem ser armazenados em um arquivo com extensão JAR. Um *JavaBean* pode ser composto por um conjunto de classes, arquivos de "help", ícones para o componente, objetos serializados, e outros elementos que o

definem. Um único arquivo JAR pode conter vários componentes. Este arquivo serve como uma unidade de distribuição e implantação de componentes *JavaBeans*.

As tecnologias EJB e *JavaBeans* citadas preocupam-se mais com a implementação dos componentes. Contudo, sabe-se que um componente precisa ser especificado e projetado conforme os requisitos de um domínio do problema. Neste trabalho estudou-se e utilizou-se o método *Catalysis* para a modelagem dos componentes.

2.3 O método *Catalysis*

Catalysis [D'Souza & Wills, 1998] é um método de desenvolvimento de software Orientado a Objetos, que integra técnicas para construir Sistemas Baseados em Componentes usando Padrões e *Frameworks*. Teve influências dos métodos OMT [Rumbaugh, 1991] e Fusion [Coleman et. al, 1994] e da UML [Fowler, 1997, Booch et. al, 1999], com base semântica forte para frameworks, e inclui novos recursos para sua extensibilidade, além dos estereótipos.

O método Catalysis fundamenta-se nos princípios de abstração, precisão e componentes "plug-in". A abstração orienta o Engenheiro de Software na busca dos aspectos essenciais do sistema, dispensando detalhes irrelevantes para o contexto do sistema. A precisão tem como objetivo descobrir erros e inconsistências na modelagem; e os componentes "plug-in" suportam o reuso de componentes para construir outros [D'Souza & Wills, 1998]. As principais características de Catalysis são:

- **Desenvolvimento baseado em componentes**: definem interfaces flexíveis para um conjunto de componentes de uma arquitetura, independentes da implementação;
- **Integridade e precisão**: definem modelos de negócio precisos em vários níveis de abstração que garantem um refinamento explícito até o nível de implementação;
- **Reuso**: baseia-se em padrões e *frameworks*, tornando o seu processo de desenvolvimento mais claro e adaptável em muitos contextos; e
- **Expansibilidade**: define consistência, refinamento e desenvolvimento incremental entre seus níveis de desenvolvimento.

O processo de desenvolvimento de software em *Catalysis* segue as características do modelo Espiral da Engenharia de Software [Pressman, 2001], e é dividido em três níveis lógicos: Domínio do Problema, Especificação dos Componentes e Projeto Interno dos Componentes como mostra Figura 5. Estes níveis correspondem às atividades tradicionais do ciclo de vida do software: Planejamento, Especificação e Projeto, que são executadas de forma incremental e evolutiva, resultando na geração de uma nova versão a cada ciclo.

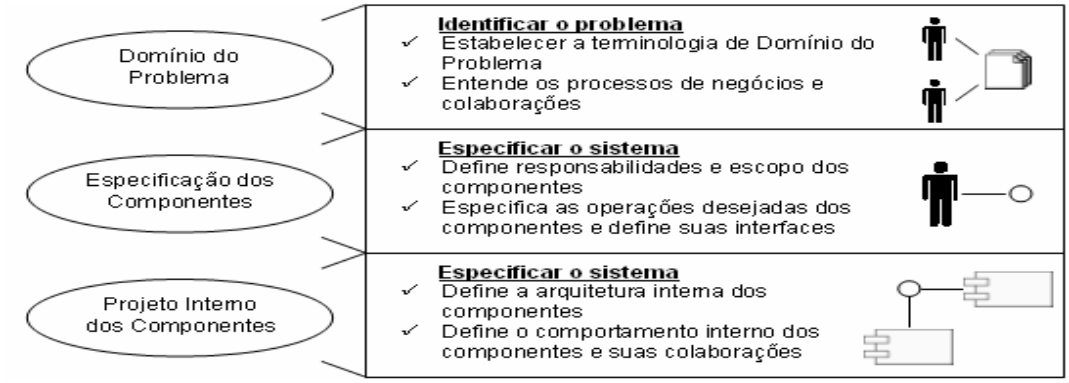


Figura 5 - Níveis Lógicos de Catalysis (Adaptado [D'Souza & Wills, 1998])

No nível Domínio do Problema é enfatizada a identificação dos requisitos do sistema no domínio do problema, especificando "o quê" o sistema deve fazer para solucioná-lo. Para tanto, identificam-se os tipos de objetos e ações do domínio, agrupando-os em diferentes visões por áreas de negócio, definindo o contexto do domínio do problema. Neste nível, utilizam-se técnicas de entrevista coletiva e informal com o usuário, relatando-as com *Storyboards* e/ou *Mind-maps* [D'Souza & Wills, 1998]. *Storyboards* são representações de diferentes situações e cenários no domínio do problema e *Mind*-maps são representações estruturadas dos termos importantes desse domínio relatados, na entrevista, pelos usuários. Por exemplo, na Figura 6 tem-se um *Mind-Map* que apresenta o requisito: objeto "Customer" tem a ação "Has" sobre o objeto "Account".

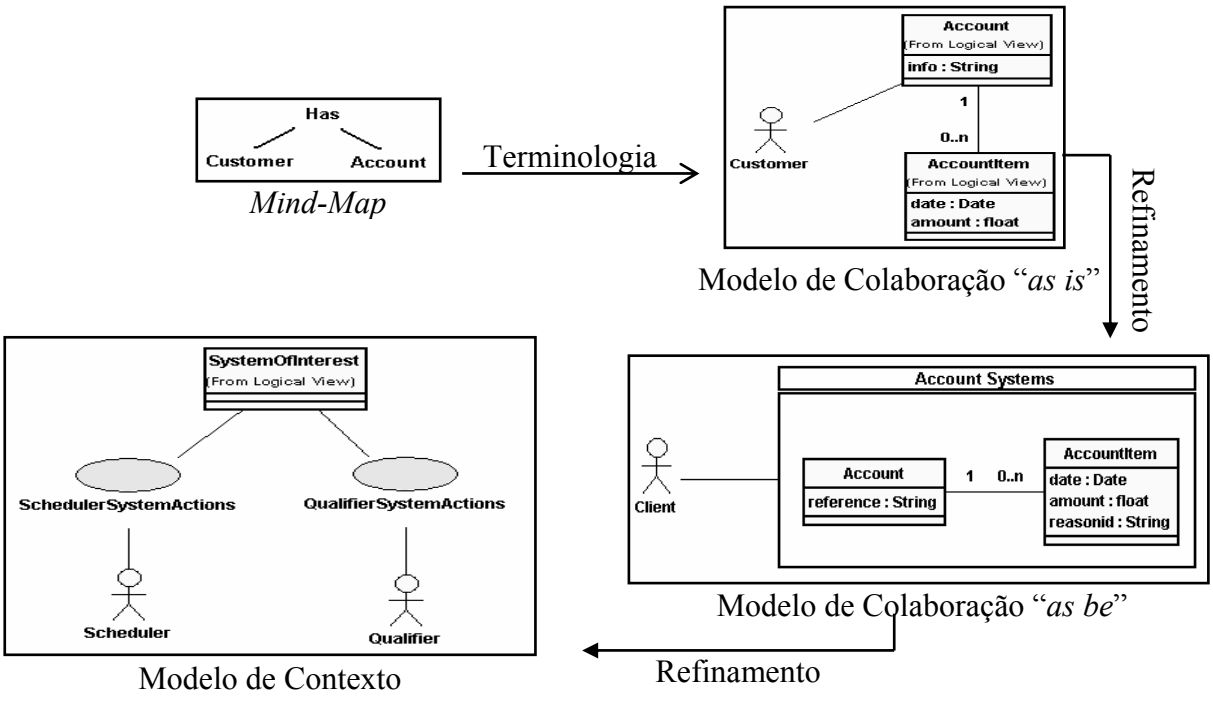
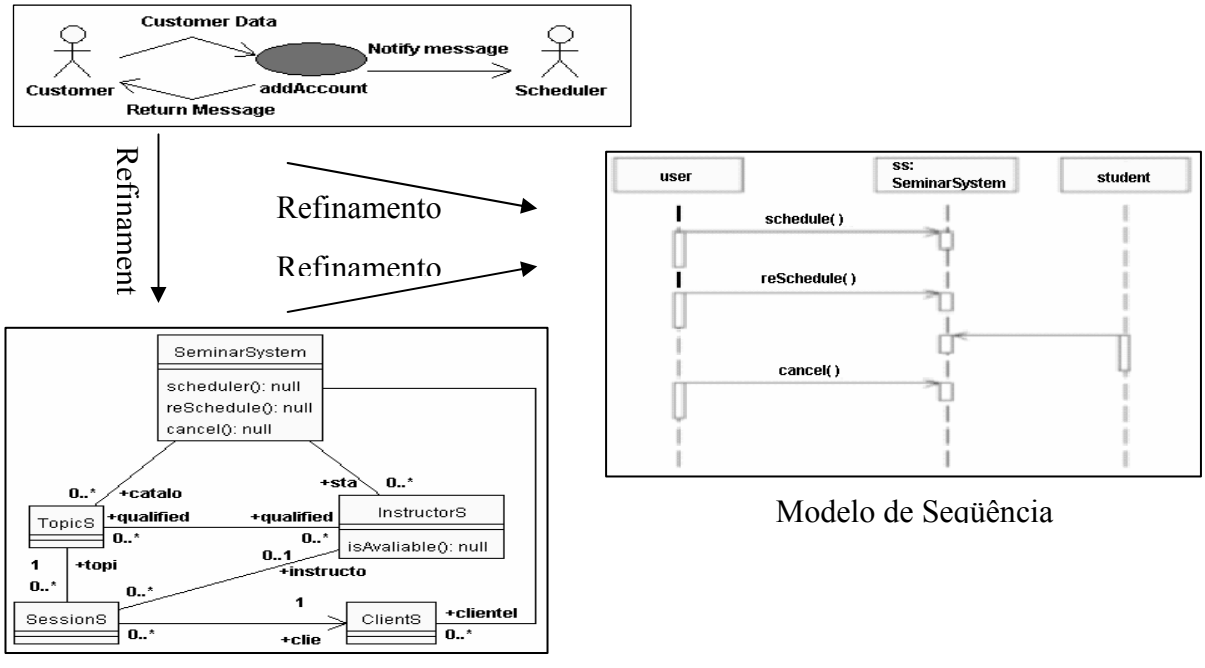


Figura 6 - Nível Domínio do Problema

Uma vez definido o contexto do domínio do problema, os requisitos coletados são transportados para o Modelo de Colaboração "as-is", que descreve como é o sistema e "as-be", que descreve como será o sistema, os quais representam uma coleção de ações e os objetos participantes. As ações de um propósito comum tanto no Modelo de Colaboração "as-is" quanto no de "as-be", são agrupadas dentro de uma colaboração. Em seguida, usando o princípio do refinamento, o Modelo de Colaboração é mapeado para um Modelo de Contexto, Figura 6 – Diagrama de Contexto, o qual pode ser representado por Modelos de Casos de Uso [Booch, 1999], [Larman, 1999], que representam os atores e suas interações com o sistema, podendo ainda utilizar *Object Constraint Language* (OCL) [Booch, 1999] para detalhar as especificações sem ambigüidades e o dicionário de dados para especificar cada tipo encontrado.

No nível Especificação dos Componentes, é dada ênfase à identificação, comportamento e responsabilidades dos componentes. Conforme Figura 7, esse nível tem início com o refinamento dos modelos obtidos no Domínio do Problema para o Modelo de Tipos que especifica o comportamento dos objetos, mostrando os atributos e as operações dos tipos de objetos, porém sem se preocupar com a implementação. Com base nos Modelos de Tipos e de Casos de Uso, especificam-se os Modelos de Interações que descrevem as seqüências das ações entre objetos relacionados que são representados por Modelos de Seqüência e de Estado.

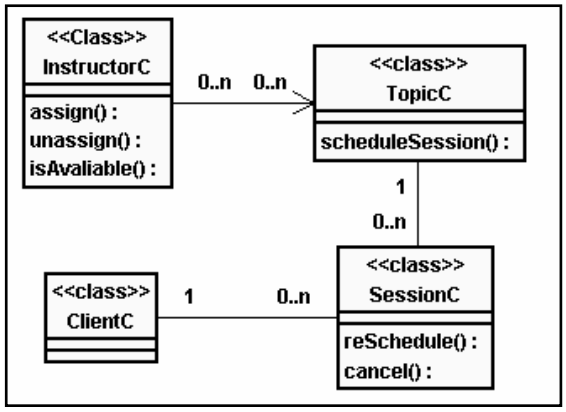


Modelo de Seqüência

Modelo de Tipos

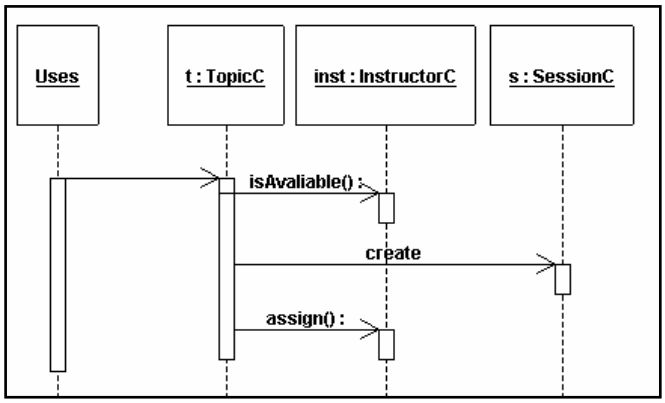
Figura 7 - Nível de Especificação dos Componentes

No Projeto Interno dos Componentes, detalha-se o comportamento interno do componente, preocupando-se com a sua distribuição física e com os requisitos não funcionais. Neste nível, conforme a Figura 8, utiliza-se o Modelo de Classes para representar as classes do componente com seus atributos, operações e relacionamentos. Para detalhar o comportamento interno pode-se usar o Modelo de Seqüência, conforme mostra a Figura 8. Outro modelo usado é o de Componentes para representar a arquitetura das plataformas física e lógica dos componentes.



Modelo de Classes

Refinamento



Modelo de Seqüência

Figura 8 - Nível de Projeto Interno dos Componentes

Além das tecnologias de implementação e de um método que orienta o processo de Desenvolvimento Baseado em Componentes, para facilitar a construção dos componentes é importante que se tenha um conjunto de ferramentas que apóiem todo ou pelo menos em parte o processo de desenvolvimento.

2.4 Ferramentas de programação baseadas na linguagem Java

Ferramentas de programação sempre se destacaram pela facilidade oferecida no desenvolvimento de aplicações de pequeno e médio porte. Com o lançamento das primeiras versões da linguagem Java pela *Sun Microsystems*, começaram a surgir ferramentas de programação baseadas nesta linguagem.

Atualmente, dentre essas ferramentas, tem-se aquelas que suportam o desenvolvimento de software orientado a componentes segundo diferentes tecnologias da linguagem Java. Este suporte à construção, vem de inúmeros recursos presentes nas ferramentas como, por exemplo, geração automática de código, compilação, execução, entre outros. Esses recursos tornam mais rápida a construção de componentes, além de proporcionar maior confiança, pois testes podem ser aplicados como forma de verificação e correção de erros durante toda a implementação. Depois de construídos, os componentes podem ser importados para a paleta de componentes da ferramenta, ficando disponíveis para o reuso das aplicações ou mesmo para a construção de novos componentes.

Para reuso, o Engenheiro de Software seleciona os componentes nas paletas de componentes e adapta-os conforme as necessidades da aplicação. À medida que novos componentes são selecionados, esses são adicionados ao projeto, e ao mesmo tempo a ferramenta gera o respectivo código. Para facilitar a programação, as ferramentas dispõem de um inspetor de objetos (*Object Inspector*), que suporta o acesso às propriedades e o tratamento de eventos dos componentes selecionados. Assim, pode-se construir uma aplicação refinando os componentes em tempo de *design*. Outras partes específicas como, por exemplo, implementação de um evento, pode exigir a interação direta no código da aplicação pelo Engenheiro de Software.

De uma maneira geral, essas ferramentas apresentam os mesmos itens estruturais: barra de ferramentas com as principais funcionalidades da ferramenta; paleta de componentes; editor de propriedades e eventos; editor visual da interface gráfica (*design*) e editor do código textual (*source*).

A seguir são apresentadas as principais características das ferramentas JBuilder, NetBeans e Eclipse, a partir das quais foram identificados os requisitos da ferramenta C-CORE.

2.4.1 JBuilder

O JBuilder é uma ferramenta da *Borland* para desenvolvimento de software na linguagem Java. É disponibilizado em diferentes versões voltadas a determinados perfis de desenvolvimento. Possui recursos para a implementação de componentes e aplicações através de *wizards* (assistentes), destacando-se: a) visualização das classes através de diagrama de classe UML; b) suporte a tecnologias como, *Web Services*, edição de arquivos XML [XML, 2003], JSP e SQL; e c) fornecimento de componentes e ferramentas para construção de aplicações GUI com acesso a banco de dados.

Para que se tenha uma idéia, tem-se na Figura 9 a interface principal da ferramenta JBuilder onde são destacados: a barra de ferramentas (1); o inspetor de objetos (2) e editor visual da interface gráfica (*design*) (3).

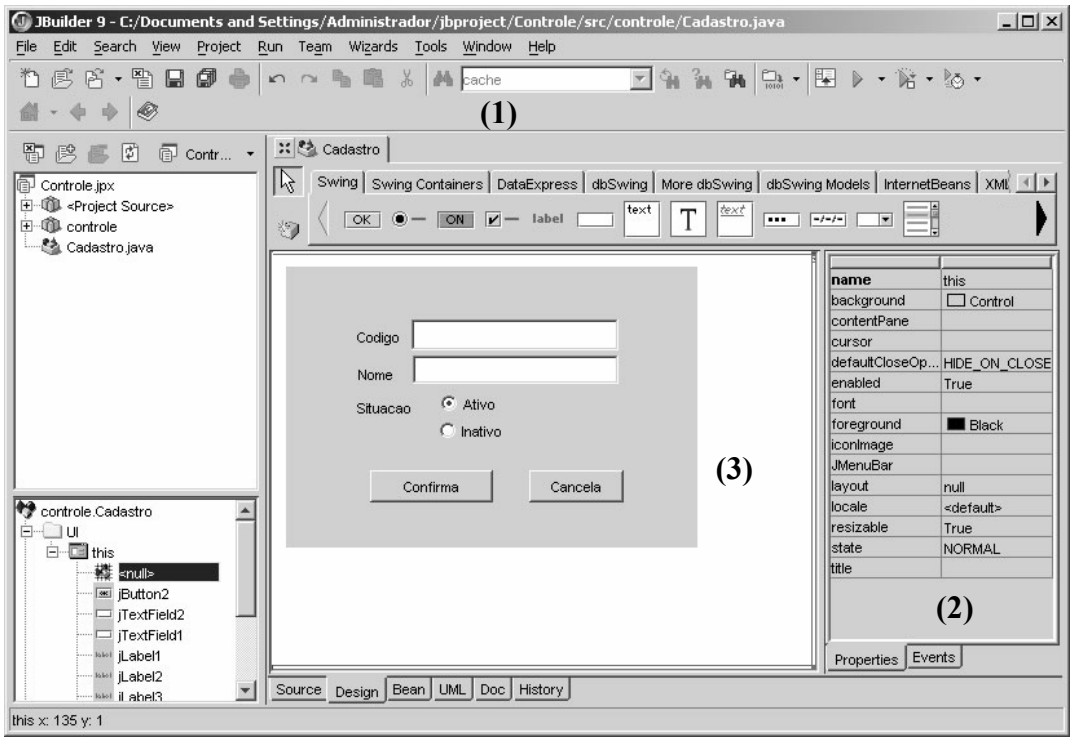


Figura 9 - Interface da ferramenta JBuilder

2.4.2 NetBeans

O NetBeans é uma ferramenta gratuita para o desenvolvimento de software na linguagem Java, permitindo o desenvolvimento rápido de aplicações com GUI. Seus principais recursos são: a) compilação e depuração do código *Java*; e b) *wizards* que permitem criar rapidamente diferentes tipos de aplicações Java como, por exemplo, RMI, *Applets*, *JavaBeans* entre outras.

A Figura 10 apresenta a interface principal da ferramenta *NetBeans* onde são destacados: a barra de ferramentas (1); o inspetor de objetos (2) e editor visual da interface gráfica (*design*) (3).

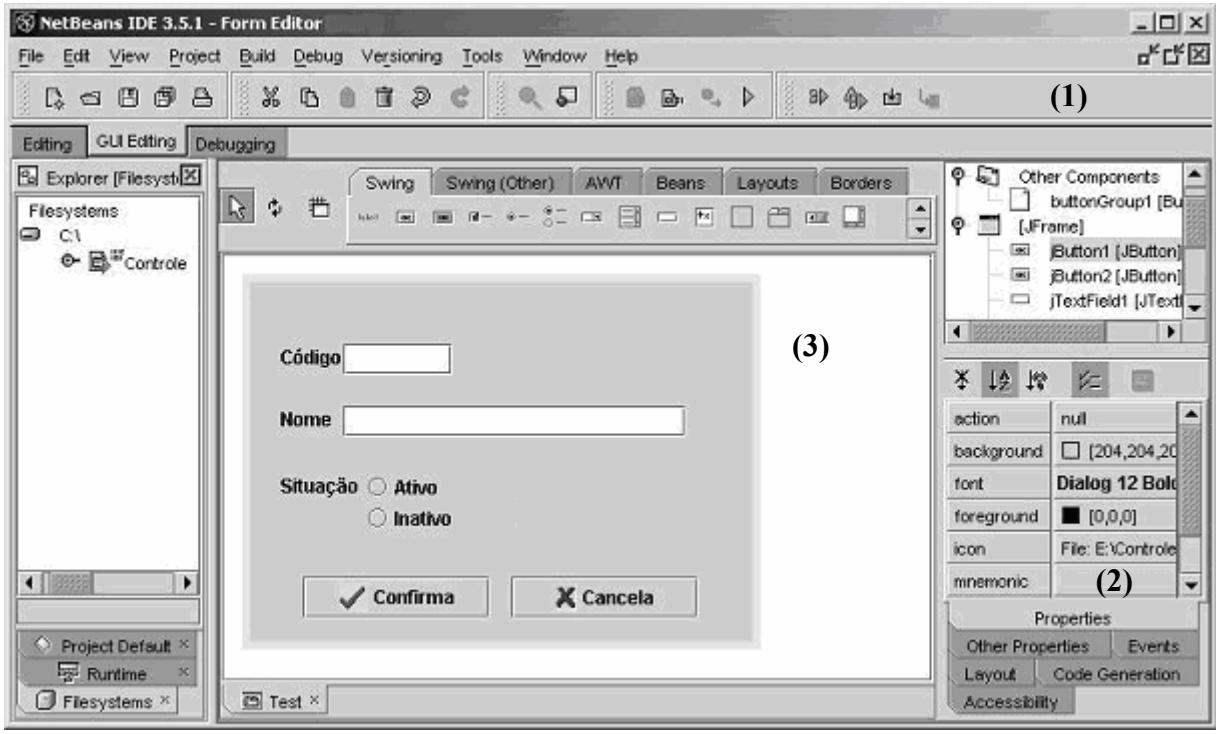


Figura 10 - Interface da ferramenta NetBeans

2.4.3 Eclipse

Eclipse é um projeto de desenvolvimento de software dedicado a prover uma plataforma para o desenvolvimento de ferramentas integradas. É formado pelo Projeto Eclipse, Projeto de Ferramentas Eclipse e Projeto de tecnologias Eclipse.

No projeto de ferramentas Eclipse, tem-se a ferramenta V4ALL GUI Designer [V4All, 2003] que permite o desenvolvimento de software utilizando a linguagem Java. Possui recursos básicos para o desenvolvimento de software como, por exemplo, componentes e aplicações *Web*. Dentre as tecnologias utilizadas citam-se *Servlet* e EJB. Sua principal característica é o suporte integrado a outras ferramentas que formam um ambiente com vários recursos para o desenvolvimento de software.

A Figura 11 ilustra a interface principal da ferramenta, onde são destacados: a barra de ferramentas (1); o inspetor de objetos (2) e editor visual da interface gráfica (*design*) (3).

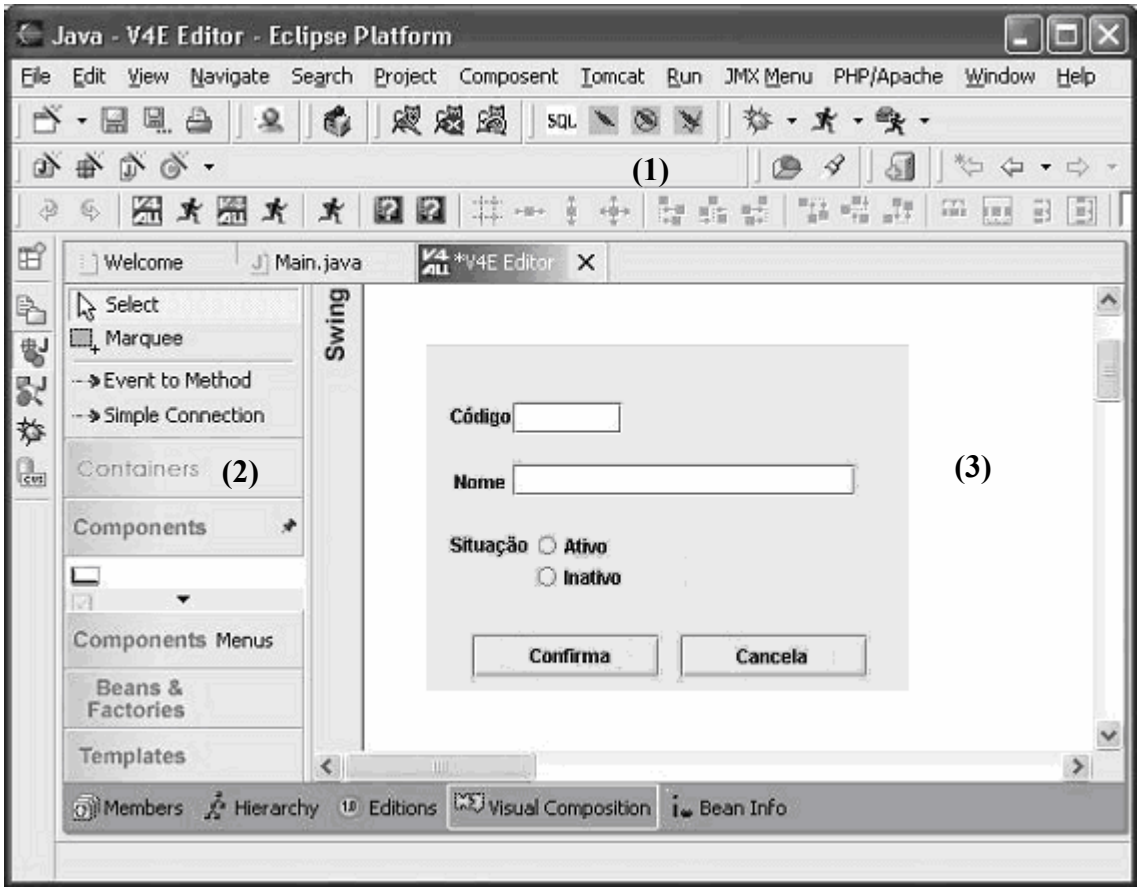


Figura 11 - Interface da ferramenta Eclipse V4All

Baseado nos estudos e utilização das ferramentas apresentadas identificaram-se os principais requisitos das ferramentas de programação. Segue-se uma apresentação dessas funcionalidades.

2.4.4 Principais requisitos das ferramentas de programação

Quando se utiliza uma ferramenta de programação no desenvolvimento de software, o Engenheiro de Software pode se beneficiar de um conjunto de funcionalidades as quais auxiliam efetivamente o desenvolvimento de aplicações, incluindo as móveis, web e outras. Dentre as diversas funções presentes nas ferramentas, algumas cooperam entre si e juntas provêm suporte tanto para o gerenciamento das aplicações por parte das ferramentas quanto para a implementação dos diferentes requisitos da aplicação.

Assim, os principais requisitos identificados nas ferramentas JBuilder, NetBeans e Eclipse foram:

- **Infrastructure**: que suporta o gerenciamento de projetos de software;

- **User Interface Configuration**: que permite personalizar fontes e cores da interface do usuário;
- **GUI Constructor**: que permite criar aplicações através do reuso de componentes disponíveis na ferramenta. Normalmente, o Construtor tem: um inspetor de objetos, que mostra as propriedades e eventos dos componentes; gerenciadores de *layout*; assistentes; e gerador de código, que gera automaticamente o código da interface à medida que alterações na mesma são efetuadas. Neste último, o código é apresentado no *Code Editor* por meio de sua integração com o construtor GUI;
- **Code Editor**: que permite escrever o código do software numa linguagem como, por exemplo, Java. O Editor oferece diferentes recursos para o Engenheiro de Software interagir durante a escrita do código;
- **Debugging**: que suporta a depuração de código;
- **Documentation**: que permite criar uma documentação para classes e interfaces selecionadas de um projeto;
- **Distributed Application Support**: que suporta a construção de aplicações distribuídas segundo diferentes tecnologias;
- **XML Support**: que suporta a escrita de código na linguagem XML e a XSLT, linguagem declarativa, baseada em XML, utilizada para apresentar ou transformar dados XML;
- **Refactoring Support**: que suporta o *Refactoring*, que analisa o código e realiza ações corretivas;
- **WebServices Support**: que permite criar, montar e consumir *WebServices*;
- **Web Support**: que permite desenvolver e manter, segundo as tecnologias *Web* da linguagem Java, aplicações cliente-servidor, as quais podem ser disponibilizadas em servidores Web suportados;
- **J2ME Support**: que oferece suporte para a tecnologia J2ME para realizar: escrita, compilação, otimização e apresentação de aplicações em dispositivos móveis;
- **J2EE Support**: que oferece suporte não somente para desenvolver e manter componentes e aplicações em camadas segundo a tecnologia J2EE, como também disponibiliza estes em servidores EJB suportados;
- **Tools Integration**: que torna possível a integração de ferramentas, visando cobrir todo o ciclo de vida do software;

- **Database Support**: que possibilita a integração com diferentes Bancos de Dados. Criação, recuperação, atualização e exclusão de registros de uma fonte de dados, com o uso da linguagem SQL, são exemplo de operações suportadas nesta integração. Dessa forma, o *Database Support* permite que, por exemplo, uma aplicação exerça ação sobre seus dados em um banco de dados. As ações podem ser realizadas com o uso de componentes, do domínio de banco de dados, disponíveis na ferramenta, os quais oferecem serviços para manipulação das tabelas e seus registros em banco de dados;
- **Version Control Support**: que suporta o controle de versão de todos os elementos pertencentes a um projeto;
- **Test Support**: que permite realizar testes, aplicados com a finalidade de encontrar erros com os softwares desenvolvidos;
- **Modeling Support**: que permite fazer a modelagem visando aumentar a representação das informações de um projeto de software. Dentre os principais modelos que podem contribuir para aumentar o nível de informação, tem-se: modelos textuais e gráficos da modelagem do software; e
- **MetaData Repository Support**: que oferece suporte a geração de metadados e o seu armazenamento em um repositório.

Nem todas as ferramentas estudadas apresentam essas funcionalidades. Além disso, o nível de suporte às funcionalidades enumeradas, varia entre as ferramentas. Segue-se uma comparação das funcionalidades encontradas nas ferramentas estudadas.

2.4.5 Comparação das principais funcionalidades das ferramentas de programação

A comparação realizada nesta seção visa tornar mais claro as funcionalidades encontradas em cada ferramenta estudada. Baseado nos requisitos apresentados na Seção 2.4.4, tem-se uma comparação (Tabela 1) das principais funcionalidades das ferramentas JBuilder, NetBeans e Eclipse. Essa tabela foi feita com base nos estudos e nas experiências com a utilização dessas ferramentas.

Tabela 1: Comparação das principais funcionalidades encontradas nas ferramentas JBuilder, NetBeans e Eclipse.

Funcionalidades	JBuilder	NetBeans	Eclipse
Infrastructure	Gerenciador completo e intuitivo. Permite a visualização distinta do projeto e seus elementos, o que facilita a organização.	Gerenciador apresenta não só as informações do projeto como também as dos seus elementos em um mesmo local, o que dificulta sua visualização e localização.	Gerenciador apresenta não só as informações do projeto como também as dos seus elementos em um mesmo local, o que dificulta sua visualização e localização.
User Interface Configuration	Muitos recursos e boa interface.	Muitos recursos e boa interface.	Muitos recursos e boa interface.
Graphical User Interface (GUI) Constructor	Construtor completo e intuitivo. Os componentes são apresentados em JTabbedPanel, e organizados em paletas, que representam os domínios dos componentes.	Construtor completo e intuitivo. Os componentes são apresentados em um Panel, e organizados em combobox, que representam os domínios dos componentes.	Construtor completo e intuitivo. Os componentes são apresentados em um Panel, e organizados em combobox, que representam os domínios dos componentes.
Code Editor	Editor é orientado à sintaxe, tem recursos como: *highlight*, que permite destacar palavras reservadas; e personalização de fontes e cores.	Editor é orientado à sintaxe, tem recursos como: *highlight*, que permite destacar palavras reservadas; personalização de fontes e cores; e ocultação de partes do código.	Editor é orientado à sintaxe, tem recursos como: *highlight*, que permite destacar palavras reservadas; personalização de fontes e cores; e ocultação de partes do código.
Debugging	Boa interface e de fácil utilização.	Boa interface e de fácil utilização.	Boa interface e de fácil utilização.
Documentation	Permite gerar a documentação através do JavaDoc.	Permite gerar a documentação através do JavaDoc.	Permite gerar a documentação através do JavaDoc.
Distributed Application Support	Oferece suporte a tecnologias como, por exemplo, CORBA, JNDI, RMI e outras.	Oferece suporte a tecnologias como, por exemplo, CORBA, JNDI, RMI e outras.	Oferece suporte a tecnologias como, por exemplo, CORBA, JNDI, RMI e outras.
XML Support	Vários recursos. Possibilita o desenvolvimento de software com XML via Editor de Código e também com assistentes.	Vários recursos. Possibilita o desenvolvimento de software com XML via Editor de Código e também com assistentes.	Vários recursos. Possibilita o desenvolvimento de software com XML via Editor de Código e também com assistentes.

Refactoring Support	Oferece suporte a alguns tipos de Refactoring.	Oferece suporte a alguns tipos de Refactoring.	Oferece suporte a alguns tipos de Refactoring.
WebServices Support	Vários recursos. Possibilita o desenvolvimento de WebServices via Editor de Código e também com assistentes.	Vários recursos. Possibilita o desenvolvimento de WebServices via Editor de Código e também com assistentes.	Vários recursos. Possibilita o desenvolvimento de WebServices via Editor de Código e também com assistentes.
Web Support	Recursos interativos provêm apoio para desenvolver software com as tecnologias Web: JSP, Servlet e outras.	Recursos interativos provêm apoio para desenvolver software com as tecnologias Web: JSP, Servlet e outras.	Recursos interativos provêm apoio para desenvolver software com as tecnologias Web: JSP, Servlet e outras.
J2ME Support	Vários recursos. Possibilita o desenvolvimento de aplicações J2ME via Editor de Código e também com assistentes.	Vários recursos. Possibilita o desenvolvimento de aplicações J2ME via Editor de Código e também com assistentes.	Vários recursos. Possibilita o desenvolvimento de aplicações J2ME via Editor de Código e também com assistentes.
J2EE Support	Bastantes recursos. Assistentes provêm apoio para desenvolver componentes EJB; recursos adicionais permitem construir visualmente componentes e seus relacionamentos, e a geração automática de código.	Bastantes recursos. Assistentes provêm apoio para desenvolver componentes EJB; recursos adicionais permitem a geração automática de código e a manipulação visual na construção de componentes EJB.	Assistentes provêm apoio para desenvolver componentes EJB, com a geração automática de código.
Tools Integration	Não é possível integrar com outras ferramentas, pois não se pode exportar o projeto para outro tipo de arquivo.	Permite a integração com outras ferramentas através da utilização de arquivos XMI [OMG, 2004].	Não é possível integrar com outras ferramentas, pois não se pode exportar o projeto para outro tipo de arquivo.
Database Support	Vários recursos. Apresenta suporte à integração da ferramenta e também das aplicações com o banco de dados.	Vários recursos. Apresenta suporte à integração da ferramenta e também das aplicações com o banco de dados.	Vários recursos. Apresenta suporte à integração da ferramenta e também das aplicações com o banco de dados.

Version Control Support	Apresenta integração com outras ferramentas como, por exemplo, Borland StarTeam, Rational ClearCase e Microsoft Visual SourceSafe.	Apresenta integração com outras ferramentas como, por exemplo, Rational ClearCase, CVS [CVS, 2004] Client Library e OpenVMS CMS.	Apresenta integração com outras ferramentas como, por exemplo, CVS Client e outras.
Test Support	Apresenta integração para realização de teste com outras ferramentas como, por exemplo, JUnit.	Apresenta integração para realização de teste com outras ferramentas como, por exemplo, JUnit.	Apresenta suporte ao projeto "*Test & Performance*", o qual oferece *frameworks* e serviços para o desenvolvimento de ferramentas de teste e performance, que podem ser usadas em todas as partes do ciclo de vida do software. Incluídas no Eclipse, oferecem uma coleção de serviços e interfaces de usuário como, por exemplo, *tracing* e *logging*.
Modeling Support	Apresenta recurso para visualização das classes e interfaces segundo notação UML.	Tem integração com ferramentas de modelagem através do intercambio de arquivos na linguagem XMI.	Possui recursos simples para a modelagem segundo notação UML.
MetaData Repository Support	Não apresenta suporte a geração, armazenamento de metadados em um repositório.	Apresenta recursos para geração de metadados segundo MOF e armazenamento em um repositório de metadados. Este é capaz de carregar metamodelos MOF armazenados em arquivos XMI, criar instancias deles, gerar interfaces JMI e exportar o conteúdo do repositório para documentos XMI.	Apresenta recursos para geração de metadados através do *framework* EMF. Este permite que modelos possam ser gerados a partir de arquivos Java, documentos XML, ou ferramentas de modelagem.

2.5 Considerações Finais

Este capítulo procurou abordar os principais trabalhos existentes sobre o assunto desta pesquisa expondo os principais conceitos e tecnologias utilizadas no desenvolvimento de software.

Na Seção 2.1 foi apresentado o conceito de componentes de software necessário ao entendimento dos artefatos produzidos pela ferramenta proposta. Além disso, esse conceito também é empregado na construção desta ferramenta visando maior qualidade. Com isto, poder-se-á compreender melhor seu projeto interno e, conseqüentemente, facilitar sua extensão com o acréscimo de novas funcionalidades, à medida que novos requisitos forem necessários.

Na Seção 2.2 foram apresentadas as tecnologias de componentes EJB e *JavaBeans,* as quais serão suportadas pela ferramenta proposta para a construção de componentes e aplicações.

Na Seção 2.3 foi apresentado o método *Catalysis*, que integra técnicas para o desenvolvimento de software baseado em componentes. Adotou-se *Catalysis* por ser um método atual e que cobre grande parte do DBC.

Nas Seções 2.4 foram apresentadas as ferramentas de programação orientadas a componentes JBuilder, NetBeans e Eclipse. Também foram apresentadas uma especificação e uma comparação das principais funcionalidades encontradas nestas ferramentas. Ao relacionar essas funcionalidades, verificou-se que *Infrastructure, Graphical User Interface Constructor, Code Editor* e *Debugging,* constituem os requisitos mínimos necessários à construção de ferramentas de programação capazes de auxiliar o Engenheiro de Software no desenvolvimento de sistemas com maior qualidade e maior produtividade.

O capítulo seguinte apresenta o Ambiente de Desenvolvimento de Software Baseado em Componentes (ADSBC) no qual está integrada a ferramenta C-CORE. Inicia-se esse capítulo apresentando o ambiente ADSBC. Em seguida, tem-se uma breve descrição das fases que compõem o desenvolvimento de software baseado em componentes no ADSBC. A fase de construção, na qual a C-CORE oferece suporte, é apresentada com mais detalhes. Por fim, é apresentada a ferramenta MVCASE, mecanismo de modelagem do ambiente e que está integrada com a ferramenta C-CORE.

Capítulo 3

Ambiente de Desenvolvimento de Software Baseado em Componentes (ADSBC)

Apresenta-se neste Capítulo o ADSBC – Ambiente de Desenvolvimento de Software Baseado em Componentes [Souza Neto et al, 2004a] no qual a ferramenta C-CORE está integrada.

3.1 Ambiente ADSBC

O ADSBC provê suporte para o desenvolvimento, manutenção e melhorias tanto de componentes como de suas aplicações. Contém um repositório que compartilha todas as informações relacionadas ao projeto ao longo do seu ciclo de vida, e as ferramentas de apoio MVCASE e C-CORE.

As principais características apresentadas pelo ADSBC são:

- Mecanismos que suportam principalmente as fases de construção e reuso de componentes;
- Controle de acesso e a evolução de objetos. Este módulo é conhecido como Repositório do ambiente e é implementado através de um Sistema de Gerenciamento de Banco de Dados (SGBD);
- Suporte para inclusão de novas ferramentas para permitir extensão do ambiente, sejam elas de apoio ao desenvolvimento de software ou com outras funções; e
- Integração entre as ferramentas que compõem o ambiente: Estas ferramentas do ambiente concordam sobre os tipos de dados, operações, métodos utilizados e o processo de desenvolvimento sendo seguido. Com a integração, as ferramentas compartilham informações, evitando redundâncias e inconsistências.

As ferramentas no ADSBC são integradas com o objetivo de prover suporte efetivo ao Engenheiro de Software na consistência entre os artefatos produzidos em todas as etapas do projeto e implementação. Segundo Thomas & Nejmeh [Thomas & Nejmeh 1992], as ferramentas podem ser integradas de diferentes formas. Dentre elas destacam-se as relacionados com a apresentação e com os dados.

Integração de apresentação refere-se à utilização de interfaces comuns a ambas as ferramentas, ou seja, interfaces construídas com base em uma mesma biblioteca, como, por exemplo, Motif [Sun Microsystems, 2003], permitindo a utilização alternada sem mudanças substanciais de estilo de interação, minimizando o impacto e tempo para aprender a utilizá-las.

Integração de dados é obtida por meio do compartilhamento dos mesmos dados, estabelecido por intermédio de arquivos ou banco de dados. As ferramentas do ADSBC são integradas por meio de arquivos com extensão XMI (*XML Metadata Interchange*), permitindo que especificações geradas entre elas sejam armazenadas textualmente nesses arquivos para intercambio de informações.

Para nortear o Engenheiro de Software na realização do desenvolvimento de software baseado em componentes, o ADSBC também provê um modelo de processo que é apresentado a seguir.

3.2 Modelo de processo suportado pelo ADSBC

O Engenheiro de Software constrói softwares que executam em computadores com diferentes plataformas, tecnologias e arquiteturas, seguindo um processo e um modelo de processo, adaptado às necessidades de negócio, que objetiva produzir software com qualidade e que seja capaz de atender às necessidades dos seus usuários.

Um modelo de processo organiza o conjunto de atividades para o desenvolvimento de software de acordo com uma filosofia de organização [Reis, 2001]. Existem diversos modelos de processo de software propostos como, por exemplo, cascata, espiral e concorrente [Pressman, 2001] os quais são aplicados a diferentes tipos de software. Cada software tem suas particularidades. Assim, um produto de software que seria incorporado em determinados tipos de hardware, não deveria seguir um modelo de processo que enfatiza a construção de protótipos. Um modelo de processo apropriado para a criação de software para o sistema de eletrônica de uma aeronave, pode não ser adequado para a criação de um site na Web [Pressman, 2001]. Para que um modelo de processo de software seja efetivo é importante que seja auxiliado por ferramentas que suportem suas atividades. Tais ferramentas são mecanismos que melhoram a produtividade e tornam os artefatos provenientes de cada atividade do processo, mais consistentes, e fáceis de serem mantidos. Em resumo, pode-se ter um software de forma mais rápida, eficiente e efetiva, com melhor tempo e custo.

Dentre os diferentes modelos de processos de desenvolvimento, tem-se o de

desenvolvimento baseado em componentes. Este modelo reúne muitas das características do modelo espiral [Boehm, 1986]. O modelo espiral é um modelo evolucionário para o desenvolvimento de software, gerando protótipos incrementais com base em uma seqüência de tarefas. Cada etapa do ciclo de vida é realizada por um conjunto de tarefas de trabalho que são adaptadas às características do projeto a ser desenvolvido. O conjunto de tarefas de trabalho e suas formalidades são ajustados de acordo com a grandeza do projeto de desenvolvimento.

O modelo de processo baseado em componentes do ADSBC (Figura 12) permite construir componentes e aplicações que podem reutilizar componentes de um domínio do problema. As ferramentas MVCASE e C-CORE são os principais mecanismos que auxiliam o Engenheiro de Software automatizando grande parte de suas tarefas. O método *Catalysis* é usado tanto para a Construção dos Componentes como das Aplicações.

A Figura 12, adaptada de [Pressman, 2001], mostra à esquerda, o modelo de Processo de Desenvolvimento Baseado em Componentes, que compreende 6 (seis) etapas: Comunicação com o Cliente; Planejamento; Análise de Risco; Construção; Entrega; e Avaliação pelo cliente. O processo é evolutivo e a cada ciclo tem-se um novo protótipo. Seguindo este modelo podem-se construir componentes de um domínio do problema e aplicações deste domínio que reutilizam estes componentes. À direita da Figura 12, tem-se, na notação SADT[1][Ross, 1997], uma visão mais detalhada da etapa Construção, onde a C-CORE atua, a qual é dividida em 2 fases: Construção de Componentes e Aplicações.

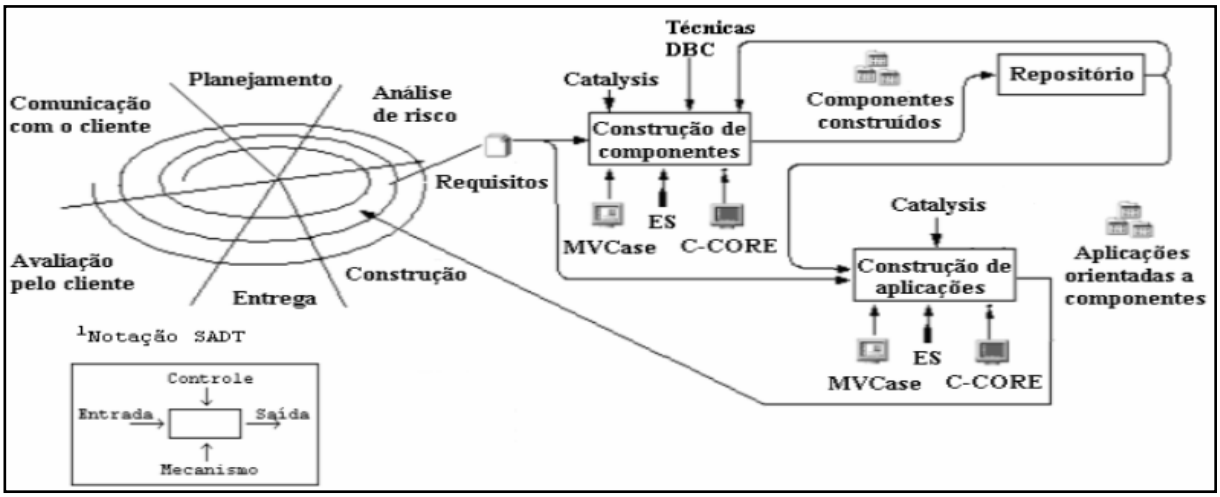


Figura 12 - Modelo de Processo de Desenvolvimento Baseado em Componentes

A seguir são descritas as etapas do modelo de processo de desenvolvimento apresentado. A etapa de construção é apresentada com mais detalhes, considerando sua importância neste projeto de pesquisa.

3.2.1 Comunicação com o cliente, Planejamento e Análise de riscos

Na etapa "Comunicação com o cliente" são realizadas tarefas com o objetivo de estabelecer uma comunicação entre o Engenheiro de Software e o cliente. Diferentes técnicas são empregadas nesta etapa como, por exemplo, reuniões, questionários entre outras, para identificar os requisitos do domínio do problema.

Na etapa "Planejamento" são realizadas tarefas relacionadas com o planejamento dos recursos e prazos do projeto. Para tanto, inicialmente é feita uma descrição do escopo do projeto, que em seguida é decomposto em partes menores. Para cada uma dessas partes, são estimados recursos e prazos com base em dados históricos e experiências. Na estimativa de custos e esforço de software deve ser levada em consideração as variáveis: humana, software reusável, ambiental, política entre outras, para realização de cada parte do projeto.

Na etapa "Análise de riscos" são avaliados os riscos técnicos e gerenciais. Os riscos constituem incertezas em estimativas, cronograma, carga de recursos entre outras. A identificação e a análise dos riscos é importante porque ajudam o Engenheiro de Software a entender e administrar a incerteza. Cada risco identificado é analisado e depois classificado por probabilidade e impacto, para determinar a possibilidade de que venha ocorrer e o dano que vai causar. Finalmente é elaborado um plano para administrar estes riscos. Também é analisada a viabilidade de desenvolvimento do software em cada versão do clico de vida. Em seguida tem-se fase de Construção de Componentes e Aplicações.

3.2.2 Construção

Nesta etapa são construídos os componentes e aplicações que reusam componentes de um domínio do problema semelhante às idéias do DBC. O DBC é divido em 2 fases. Na primeira fase, são construídos os componentes de um Domínio do Problema e na segunda são construídas as aplicações que reusam estes componentes.

Os componentes desenvolvidos e implementados numa linguagem orientada a componentes são armazenados num repositório para reuso. Por exemplo, podem-se ter componentes de domínios básicos, como GUI e BD, de domínios de aplicações como Educação a Distância, Comércio Eletrônico, e outros. Dessa forma, as aplicações podem reutilizar estes componentes, disponíveis no repositório, de forma visual e interativa.

A fase de Construção de Componentes é dividida em cinco passos (Figura 13): Domínio do Problema, Especificação dos Componentes, Projeto Interno dos Componentes,

Refinamento de Componentes (onde se tem a construção de componentes não disponíveis) e Implementação dos Componentes. Os principais mecanismos que auxiliam o Engenheiro de Software nos passos são as ferramentas MVCASE e C-CORE.

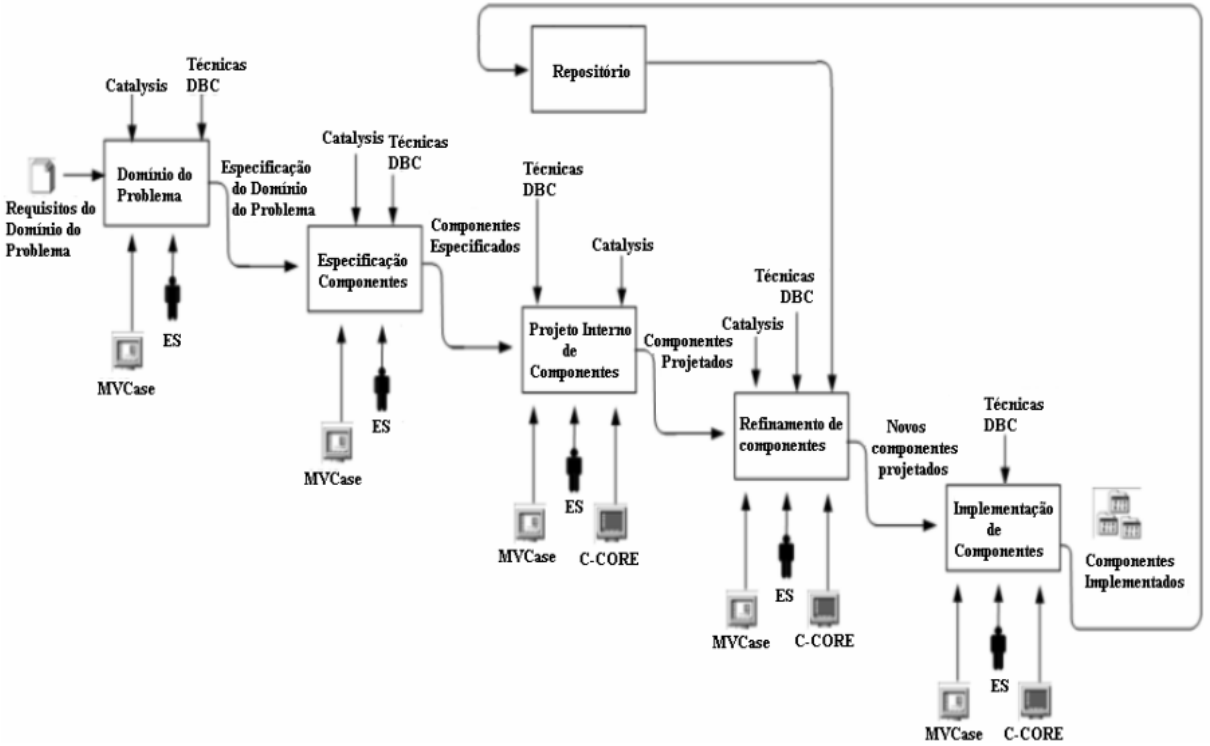


Figura 13 - Construção de Componentes

Conforme mostra a Figura 13, inicialmente tem-se a especificação os requisitos do domínio com o foco em componentes (Domínio do Problema). Em seguida, essas especificações são refinadas para obter os componentes especificados sem, contudo considerar os requisitos não funcionais (Especificação dos Componentes). No próximo passo, continua-se o processo de refinamento das especificações dos componentes, agora considerando os requisitos não funcionais, como a plataforma e a arquitetura adotada para o projeto (Projeto Interno de Componentes). Em seguida, refinam-se os componentes projetados, comparando-os com os que já se encontram disponíveis no repositório (Refinamento de Componentes). Neste passo, somente os componentes que não existem no repositório são mantidos no projeto e encaminhados para o próximo passo (Implementação de Componentes), onde se tem a implementação dos componentes em uma linguagem de programação.

Uma vez construídos os componentes de um domínio do problema, o Engenheiro de Software pode construir aplicações que reutilizam estes componentes, conforme mostra a

Figura 14. Parte-se dos requisitos da aplicação e segue-se o ciclo de vida normal de desenvolvimento de software, que compreende: Especificação, Projeto, Implementação e Teste da Aplicação. No passo Teste o Engenheiro de Software realiza testes com a aplicação, que permitem verificar se a mesma atende aos requisitos especificados. Os erros encontrados podem ser da aplicação ou dos componentes construídos na fase anterior. Os resultados de execução orientam o Engenheiro de Software no processo de refinamento e depuração tanto da aplicação como dos componentes do domínio construído. O método *Catalysis* e as ferramentas MVCASE e C-CORE são também usadas nesta fase para orientar e apoiar o engenheiro de software na execução do processo de desenvolvimento.

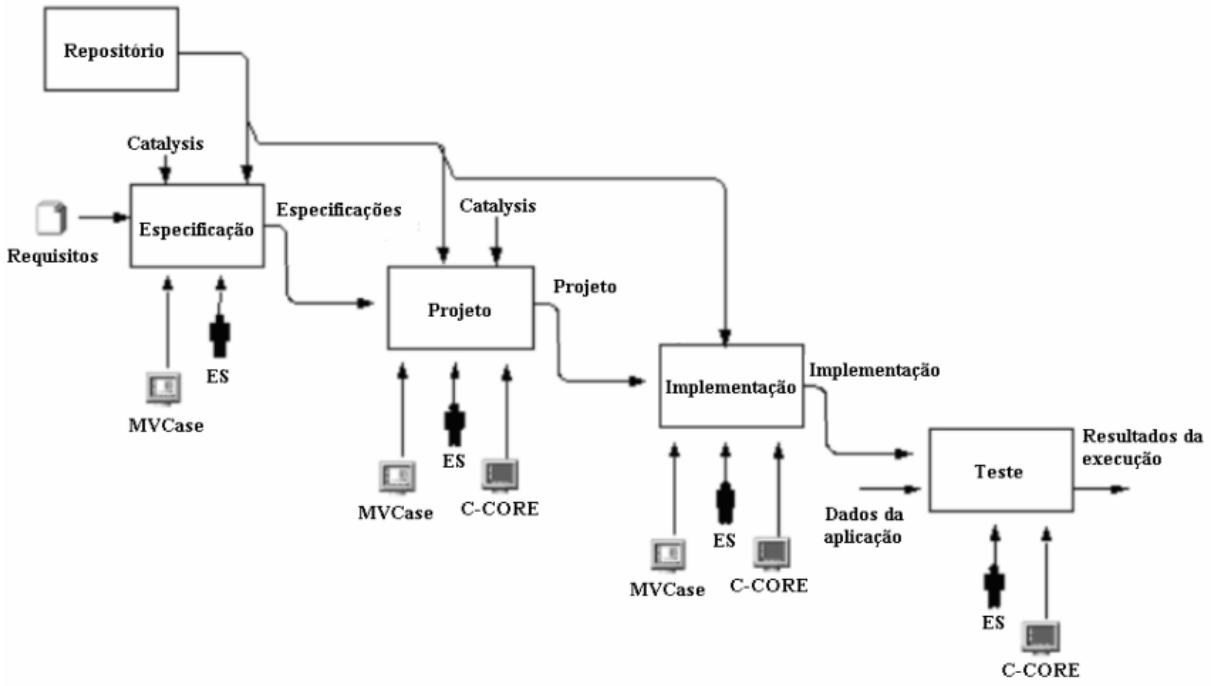


Figura 14 - Construção de Aplicações

O aumento de reuso dos componentes pode implicar em alterações nos seus projetos e implementações, requerendo muitas vezes experiência e bom conhecimento do domínio do problema. O modelo de desenvolvimento baseado em componentes visa principalmente o reuso de software, o que oferece ao Engenheiro de Software os benefícios de diminuir o tempo de desenvolvimento, os custos, além de aumentar a compatibilidade do software construído. As próximas etapas tratam da entrega e avaliação do processo de desenvolvimento, em cada versão do ciclo de vida dos componentes.

3.2.3 Entrega e Avaliação pelo Cliente

As tarefas da etapa "Entrega" destinam-se a entregar e instalar os artefatos produzidos e fornecer apoio ao usuário. Treinamentos dos usuários também podem ser realizados.

As tarefas da etapa "Avaliação pelo cliente" destinam-se a realizar avaliações dos artefatos entregues, o que constitui da análise da conformidade dos componentes em relação aos requisitos levantados na etapa de comunicação com o cliente. Problemas detectados podem ser corrigidos através de tarefas de manutenção. Avaliação do planejamento também pode ser realizada objetivando uma avaliação da produtividade, custos reais versus estimados e prazos.

3.3 Ferramenta MVCASE

Na Engenharia de Software, o uso de ferramentas CASE [Pressman, 2001], [Sommerville, 1995] no projeto vem ganhando destaque. Dentre as diferentes ferramentas tem-se a MVCASE, que é orientada a objetos, desenvolvida em Java e, portanto, multiplataforma, que suporta a especificação de requisitos usando UML. Os requisitos podem ser especificados usando técnicas gráficas e textuais em diferentes níveis de abstração. A especificação de requisitos utilizando técnicas gráficas facilita a comunicação entre o engenheiro de software e seus usuários.

Na MVCASE, o engenheiro de software especifica um sistema em UML segundo quatro visões:

- **Visão de Casos de Uso**: abrange os casos de uso que descrevem a parte comportamental do sistema conforme é visto pelos seus usuários finais, analistas e pessoal de teste. Nesta visão, além do diagrama de caso de uso, podem-se utilizar outros diagramas como, por exemplo, diagrama de interação, representado pelos diagramas de seqüência e de colaboração, e o diagrama de transição de estados [Booch et al, 1999];
- **Visão Lógica**: abrange as classes, interfaces e colaborações. Nesta visão, além do diagrama de classes, tem-se o diagrama de estados;
- **Visão de Componentes**: abrange os componentes utilizados para a construção do sistema. A principal técnica desta visão é o digrama de componentes [Booch et al, 999]; e
- **Visão de "*Deployment*"**: abrange os nós que formam a topologia de hardware em que o sistema é executado. A principal técnica desta visão é o diagrama de implantação [Booch et al, 1999].

A seguir são descritas algumas técnicas UML utilizadas para a construção de sistemas de software na ferramenta MVCASE:

- **Diagrama de Caso de Uso**: mostra um conjunto de casos de uso, agentes externos e seus relacionamentos. Um Caso de Uso é utilizado pelo engenheiro de software para especificação do comportamento do sistema ou parte do sistema. Ele descreve um conjunto de seqüência de ações efetuadas pelo sistema com o objetivo de realizar algo para um ou mais agentes externos. Os agentes externos, que podem ser até outros sistemas que possam interagir com o sistema que está sendo modelado são representados por atores.
- **Diagrama de Interação**: mostra um conjunto de objetos e seus relacionamentos em uma interação, incluindo as mensagens que poderão ser trocadas entre eles.
    - Diagrama de Seqüência: mostra uma interação organizada, em uma seqüência de tempo, entre os objetos participantes de um caso de uso, e suas trocas de mensagens. Esses diagramas podem ser utilizados para especificar sistemas de tempo real, onde a seqüência no tempo é relevante. Um Diagrama de Seqüência possui duas dimensões: vertical, que representa o tempo, e horizontal, que representa diferentes objetos.
    - Diagrama de Colaboração: mostra uma interação organizada, em uma estrutura, entre os objetos participantes de um caso de uso, e suas trocas de mensagens. Esses diagramas podem ser utilizados para especificar sistemas em que o contexto de colaboração é significante. Em um Diagrama de Colaboração, os objetos são tidos como vértices de um gráfico e os vínculos que conectam esses objetos são tidos como arcos do gráfico.
- **Diagrama de Classes**: mostra um conjunto de pacotes, classes, interfaces e seus relacionamentos;
- **Diagrama de Estados**: mostra os estados dos objetos de uma classe. Os eventos desse diagrama causam uma transição de um estado para outro e as ações resultam na mudança de estado. Cada Diagrama de Estados está associado a uma classe ou a um diagrama de estados de um nível mais alto.
- **Diagrama de Componentes**: mostra as dependências entre componentes de software, incluindo componentes de código fonte, componentes de código binário, e componentes executáveis. Um módulo de software é representado como um tipo de componente. Componentes são compostos pelos seguintes elementos: *packages*, programa

principal, subprogramas e tarefas. Cada diagrama de componentes fornece uma visão física de um modelo do software.

A Figura 15 mostra a interface principal da MVCASE, organizada nas visões *Use Case*, *Logical*, *Component* e *Deployment* e um exemplo de diagrama, no caso o de componentes, de um domínio de Educação a Distância.

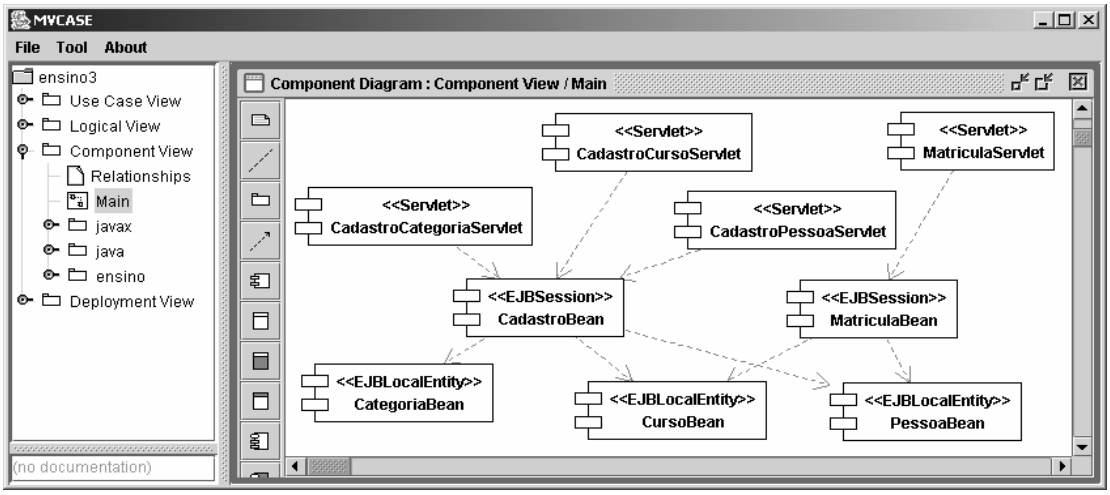


Figura 15 - Interface Principal da MVCASE e Um Exemplo do Diagrama de Componentes

- **Diagrama de *Deployment***: mostra as conexões físicas entre os processadores, dispositivos e a alocação dos processos aos processadores. Esse diagrama mostra a organização do hardware e a ligação do software com os dispositivos físicos. O tipo do dispositivo de hardware é dado pelo seu *stereotype*, tais como, processador, vídeo, memória, disco e outros dispositivos periféricos.

A MVCASE suporta também o Desenvolvimento Baseado em Componentes (DBC), e utiliza a tecnologia EJB para a implementação dos componentes. Os componentes de um projeto podem ser do tipo EJB Entity ou EJB Session. Pode-se também, através de um *wizard*, fazer o *deployment* dos componentes num servidor EJB, no caso J2EE, da Sun Microsystem. O *deployment* consiste em publicar os componentes no servidor, deixando-os disponíveis para serem acessados pelas diferentes aplicações.

3.4 Considerações Finais

Neste capítulo foi apresentado o ambiente ADSBC, no qual a ferramenta C-CORE está inserida. Também foi apresentada a MVCASE, uma ferramenta orientada a objetos, que também está inserida no ambiente e oferece suporte para a especificação dos requisitos de software, segundo a UML. O ADSBC e a ferramenta MVCASE pertencem ao grupo de

engenharia de software do qual este pesquisador e seu orientador fazem parte e, por isso, foram escolhidos para se relacionarem com a ferramenta C-CORE.

O próximo capítulo apresenta a construção da ferramenta C-CORE, com a sua arquitetura e principais artefatos de sua modelagem e implementação.

Capítulo 4

C-CORE – *Component Construction and Reuse*

A C-CORE é uma ferramenta de programação orientada a componentes de software, desenvolvida segundo o método *Catalysis* e implementada com a tecnologia de componentes *JavaBeans*. Suporta tanto a Construção quanto o Reuso de Componentes implementados na linguagem de programação *Java,* através de recursos textuais e gráficos, o que torna a implementação mais atrativa e interativa.

A C-CORE também suporta diversas funcionalidades, que foram definidas com base nos requisitos identificados a partir do estudo do domínio de ferramentas semelhantes à C-CORE, como o JBuilder, NetBeans e Eclipse. Na modelagem da C-CORE seguiu-se o método *Catalysis*, complementado com técnicas de implementação, resultando em protótipos os quais foram refinados até à versão final da C-CORE.

Para que se tenha uma idéia do processo de desenvolvimento da C-CORE, segue-se uma apresentação das atividades realizadas para sua construção.

4.1 Construção da ferramenta C-CORE

A construção da C-CORE foi realizada em 4 fases: Domínio do Problema, Especificação, Projeto Interno e Implementação.

4.1.1 Domínio da C-CORE

Seguindo *Catalysis*, no primeiro nível, que corresponde ao Domínio do Problema, foram identificados os requisitos para a ferramenta, com base nos estudos e nas experiências com a utilização das ferramentas JBuilder, NetBeans e Eclipse. A Tabela 2 apresenta o conjunto de funcionalidades propostas e uma breve descrição de como foram atendidos pela C-CORE. Esse conjunto de funcionalidades (Tabela 2) representa os requisitos necessários à construção de uma ferramenta de programação capaz de auxiliar no desenvolvimento de componentes de software e suas aplicações, implementados com algumas das principais tecnologias utilizadas pela indústria de software.

Tabela 2: Funcionalidades da C-CORE.

Funcionalidades	Descrição
Infrastructure	Gerenciador simples e intuitivo. Permite a visualização distinta e organizada do projeto e seus elementos.
User Interface Configuration	Alguns recursos incluindo configuração de fonte e cores interface.
Graphical User Interface (GUI) Constructor	Construtor simples e intuitivo, onde os componentes são apresentados em JTabbedPanel, e organizados em paletas, que representam os domínios dos componentes.
Code Editor	Editor simples com o recurso *highlight*, que permite destacar palavras reservadas.
Debugging	Depurador simples, que possibilita a execução passo-a-passo do código.
Documentation	Geração da documentação do código através do JavaDoc.
Distributed Application Support	Suporte ao desenvolvimento de software usando tecnologias como, por exemplo, CORBA, JNDI, RMI e outras.
XML Support	Suporte ao desenvolvimento de software com XML.
WebServices Support	Suporte ao desenvolvimento de WebServices.
Web Support	Suporte ao desenvolver de software com as tecnologias Servlet e Applet.
J2ME Support	Suporte ao desenvolvimento de aplicações J2ME.
J2EE Support	Suporte ao desenvolvimento de aplicações J2EE.
Tools Integration	Suporte à integração com outras ferramentas.
Database Support	Suporte a integração da ferramenta o banco de dados.
Modeling Support	Integração com ferramentas de modelagem através do intercambio de arquivos na linguagem XMI.
MetaData Repository Support	Integração com o módulo MDR da ferramenta NetBeans, o qual oferece suporte ao *MetaData Repository*.

Para modelar os requisitos foram inicialmente utilizados *mind-maps*, visando representar os termos mais importantes do domínio do problema. Os *mind-maps* dão origem aos modelos de abstração dos cenários de uso da ferramenta. São modelos que representam a coleção de ações e os objetos participantes. A Figura 16 mostra um Modelo de Colaboração identificando os atores e suas principais ações especificadas nesta etapa, obtido pelo refinamento dos *mind-m*ap. Os atores são representados por retângulos e suas ações por elipses [D'Souza & Wills, 1998].

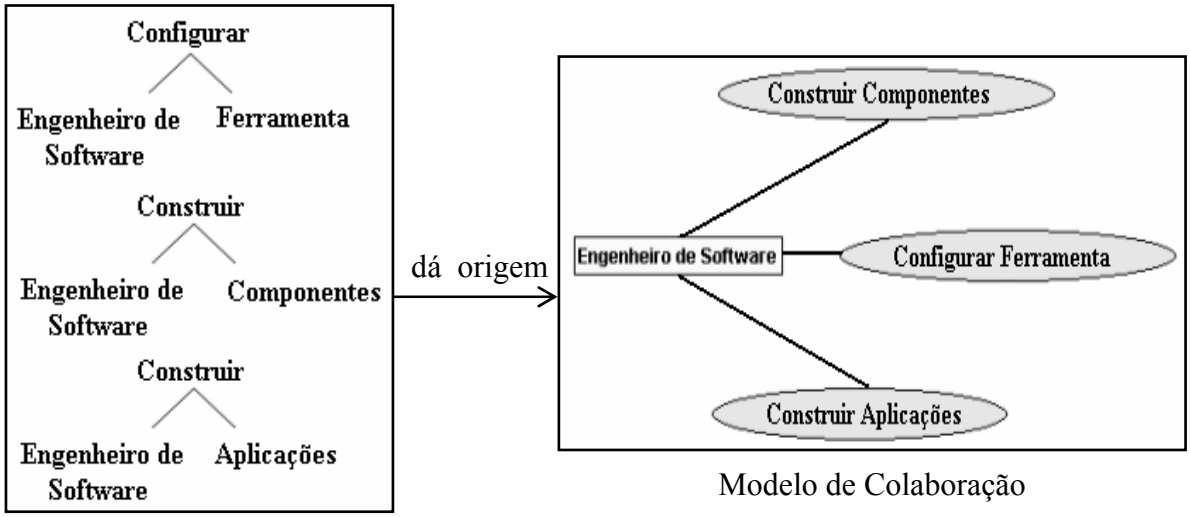


Figura 16 - Modelos: *Mind-Map* e de Colaboração

No caso, o ator "Engenheiro de Software" interage com a C-CORE para Configurar a Ferramenta, que representam as ações para personalizar o ambiente de trabalho. Além disso, o Engenheiro de Software também interage com a C-CORE para Construir Componentes e Aplicações, ou seja, para criar e alterar componentes e suas aplicações. Estas ações são refinadas em um conjunto de funcionalidades disponíveis na ferramenta de programação.

O Modelo de Colaboração é refinado, buscando melhor entendimento das funcionalidades, especificando-se seus Casos de Uso. Os Casos de Uso representam cenários das interações dos atores na ferramenta e são agrupados em contextos para melhorar a legibilidade e facilitar sua compreensão. A Figura 17 mostra um Modelo de Casos de Uso para a ação "Configurar Ferramenta" obtida do refinamento da ação "Configurar Ferramenta". Nesta etapa, ações podem ser modificadas ou adicionadas dando origem a novos comportamentos, como por exemplo, "Configurar Interface do Usuário" e "Integrar Ferramentas". Deste último deriva o Caso de Uso: "Integrar Ferramentas de BD".

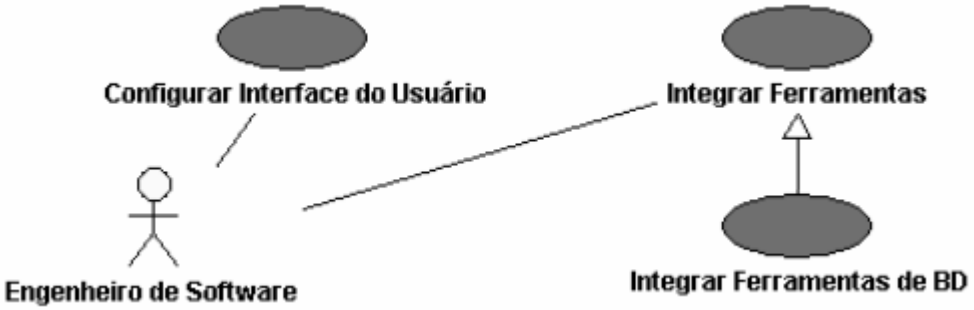


Figura 17 - Modelo de Caso de Uso para a ação "Configurar Ferramenta"

Através da personalização de fontes e cores de alguns dos itens estruturais da C-CORE, o Engenheiro de Software pode configurá-la (Configurar Interface do Usuário), para torná-la mais amigável. Também através de configurações, a ferramenta C-CORE pode ser integrada com outras ferramentas de software, ampliando, assim, o suporte oferecido ao Engenheiro de Software na realização de suas atividades.

Da mesma forma, a ação "Construir Aplicações", do Modelo de Colaboração da Figura 16, também foi refinada em um Modelo de Case de Uso, conforme mostra a Figura 18. Para esta ação, também foram adicionados novos comportamentos, como por exemplo, "Importar Projeto", "Criar Projeto", "Criar Artefato" (que deriva os Casos de Uso: "Criar Interface" que inclui "Usar Editor de Código", "Criar Classe" que inclui "Usar Editor de Código", "Criar GUI" que inclui "Usar GUI Constructor"), "Compilar Projeto", "Executar Projeto", "Depurar Projeto", "Exportar Projeto", "Empacotar Projeto", "Gerar JavaDoc", "Configurar Projeto" (que deriva os Casos de Uso: "Configurar Diretório de Trabalho" e "Configurar Biblioteca" que por sua vez, deriva "Incluir Suporte para Tecnologias"), identificados a partir da ação "Construir Aplicações".

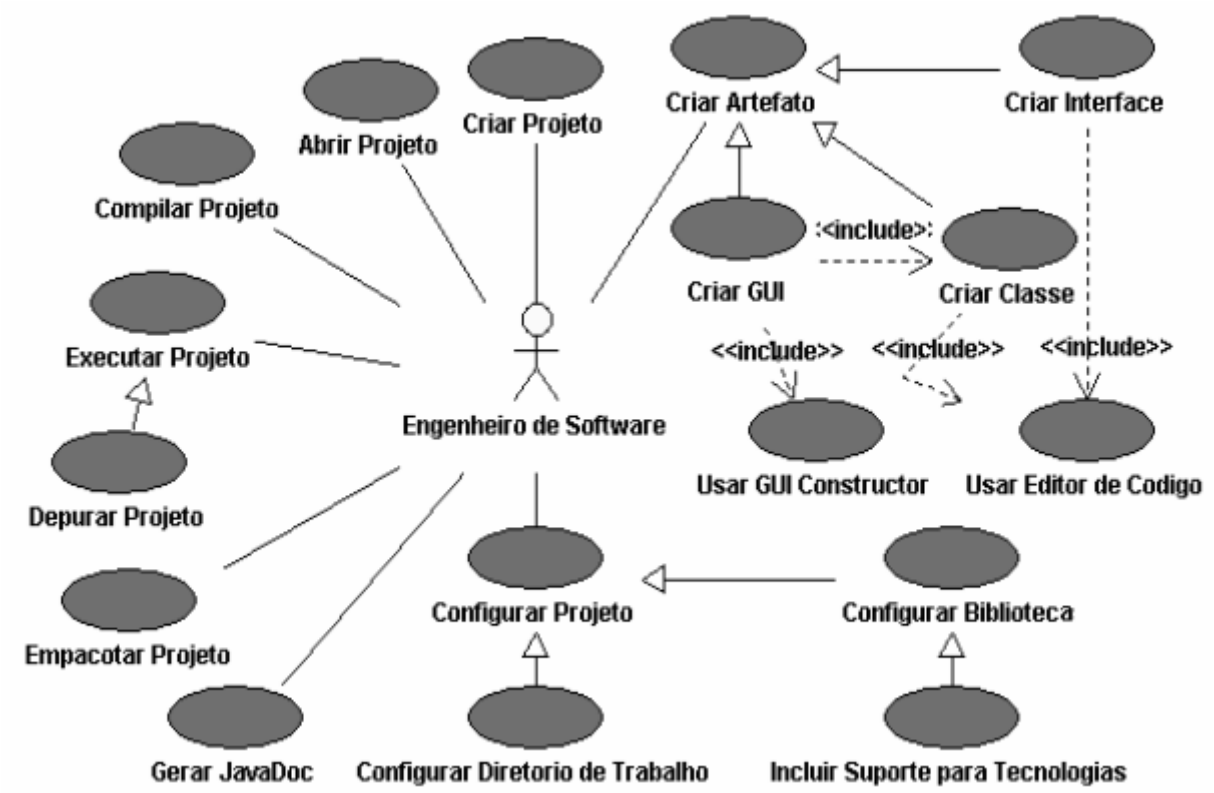


Figura 18 - Modelo de Caso de Uso para a ação "Construir Aplicações"

Inicialmente, o Engenheiro de Software cria um projeto, que pode ter suas configurações e artefatos. Dentre esses artefatos têm-se as interfaces e as classes. Na construção desses artefatos, o Engenheiro de Software tem o apoio do Editor de Código, que representa o recurso textual de implementação, onde pode-se importar outros recursos necessários para o projeto. Além de classes, têm-se os artefatos do tipo "interface GUI" que são construídos com o apoio do GUI *Constructor*, que suporta a construção de interfaces gráficas do usuário a partir do reuso de componentes do domínio GUI. Esse tipo de artefato possui uma representação gráfica, bem como uma classe que a implementa na linguagem Java. Além do cenário de criação, o Engenheiro de Software pode abrir um projeto já existente e dar prosseguimento à sua implementação. Nesse caso, considera-se que o projeto foi criado e persistido em um arquivo XMI pela MVCASE ou C-CORE.

Em seguida, o Engenheiro de Software realiza outras atividades de implementação que incluem, por exemplo, as mini-especificações dos métodos em cada classe. À medida que o Engenheiro de Software realiza essas atividades, o mesmo pode compilar o código e, em seguida, executá-lo para verificar a existência de erros. A C-CORE oferece suporte para execução passo-a-passo do código, inspeção do conteúdo de variáveis, e outros recursos. Esses recursos reduzem significativamente o tempo de implementação do software, uma vez que erros podem ser mais facilmente identificados.

Para finalizar o desenvolvimento do projeto, o Engenheiro de Software, opcionalmente, pode gerar o JavaDoc para classes e interfaces, para auxiliar no entendimento do software. Também, pode-se empacotar o software, agregando seus artefatos em um arquivo com extensão "JAR".

Outras técnicas podem ser utilizadas para identificar os requisitos neste nível do Domínio do Problema, como por exemplo, o *snapshot* que é a descrição de um conjunto de objetos e valores de seus atributos em um particular ponto no tempo. A Figura 19 mostra um *snapshot* dos objetos Interface GUI (Tela), Projeto e Biblioteca.

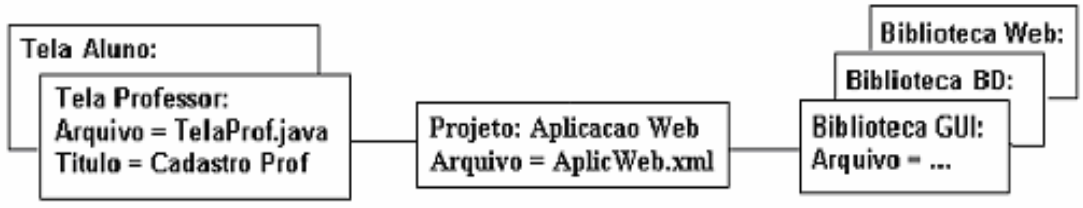


Figura 19 - *Snapshot* dos objetos Interface GUI (Tela), Projeto e Biblioteca

4.1.2 Especificação da C-CORE

Neste passo, correspondente ao segundo nível de *Catalysis*, foi descrito o comportamento externo da C-CORE de uma forma não ambígua. No caso, foram refinados os modelos do Domínio do Problema, especificando-se os Modelos de Tipos.

Os Modelos de Tipos da C-CORE foram obtidos pelo refinamento dos modelos de casos de uso, que representam os comportamentos dos objetos e suas conexões de mensagens, e dos modelos *snapshots* que representam os atributos dos objetos e seus relacionamentos.

Na Figura 20, por exemplo, apresenta-se um Modelo de Tipos com os principais tipos da ferramenta C-CORE, onde são destacados seus atributos e métodos usados para implementação de um projeto de software. A C-CORE suporta a implementação de um projeto de software de cada vez, o qual é composto de 01 (um) ou mais arquivos do tipo: interface, classe e *InterfaceGUI*. Todos os tipos são derivados do tipo File, exceto *InterfaceGUI* que deriva do tipo interface. Caso seja necessária alguma biblioteca específica para algum tipo de arquivo, o Engenheiro de Software pode importá-la no projeto.

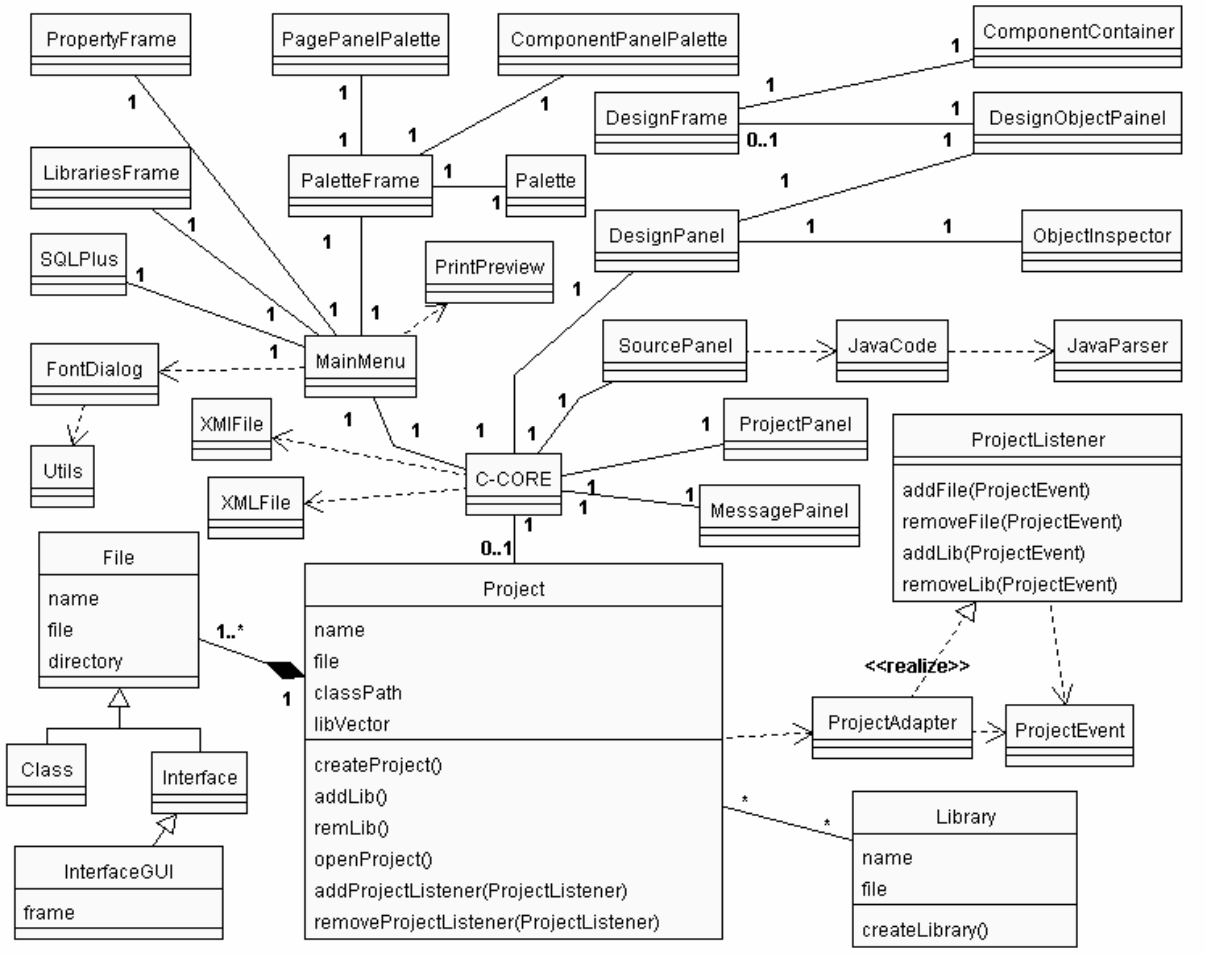


Figura 20 - Modelo de Tipos com os principais tipos da ferramenta C-CORE.

4.1.3 Projeto Interno da C-CORE

No terceiro nível de *Catalysis*, Projeto Interno dos componentes, foram modeladas as classes da C-CORE e seus relacionamentos, obtidas pelo refinamento dos modelos especificados no segundo nível. A Figura 21 mostra um Modelo de Classes, obtido a partir do refinamento do Modelo dos Tipos do nível anterior. Nesta etapa definiram-se as tecnologias adotadas para a implementação da C-CORE. Assim, baseado na tecnologia *JavaBeans* foram especificadas as classes que representam a C-CORE. Conforme mostra o modelo da Figura 21, o nível de detalhamento é maior e mais próximo da implementação.

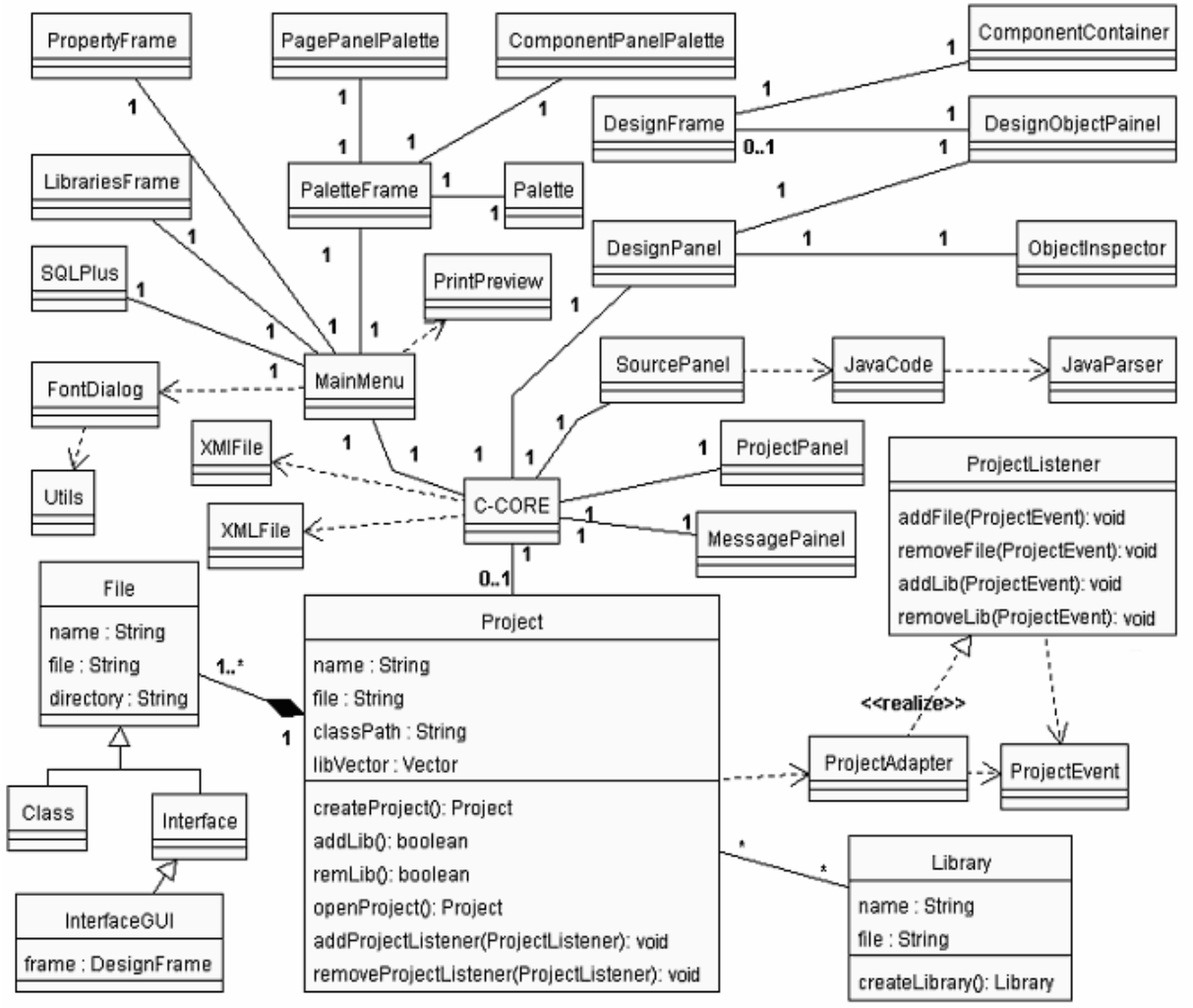


Figura 21 - Modelo de Classe obtido a partir do refinamento do Modelo dos Tipos do nível anterior

Finalmente, as classes são refinadas dando origem aos componentes que disponibilizam suas funcionalidades através de suas interfaces para prover serviços. Algumas classes foram agrupadas de modo a formar um único componente de software. Os componentes foram construídos e organizados, levando-se em conta uma determinada

arquitetura de software, que os organiza de acordo com as funções que desempenham. Dentre as diversas arquiteturas propostas e encontradas na literatura, tem-se a de 03 (três) camadas: *Interface*, que é responsável pela aparência da ferramenta; *Business Services*, que contém as regras de negócio do software; e *Data Service*, que se preocupa com acesso e manipulação de dados.

A Figura 22 mostra um modelo de componentes obtido pelo refinamento do modelo de classes construído no passo anterior, organizado segundo a arquitetura de 03 (três) camadas. Na camada "*Interface*" tem-se os componentes: SQLPLUS (classe SQLPLUS); MessagePanel (classe MessagePanel); PropertyFrame (classe PropertyFrame); ProjectPanel (classe ProjectPanel); LibraryFrame (classe LibraryFrame); PaletteFrame(classes: PaletteFrame, ComponentPanelPalette e PagePanelPalette); SourcePanel (classe SourcePanel); MainMenu (classe MainMenu); DesignPanel(classes: ComponentContainer, DesignFrame, DesignObjectPanel, DesignPanel e ObjectInspector); PrintPreview (classe PrintPreview) e C-CORE (classe C-CORE). Na camada "*Business Services*" tem-se os componentes: File (classes: File, Class, Interface e InterfaceGUI); Project (classes: Project, ProjectListener, ProjectAdapter e ProjectEvent); Library (classe Library); Palette (classe Palette) e Util (classes: FontDialog e Util). Na camada "*Data Services*" tem-se os componentes: XMIFile (classe XMIFile); XMLFile (classe XMLFile) Code (classes: JavaCode e JavaParser).

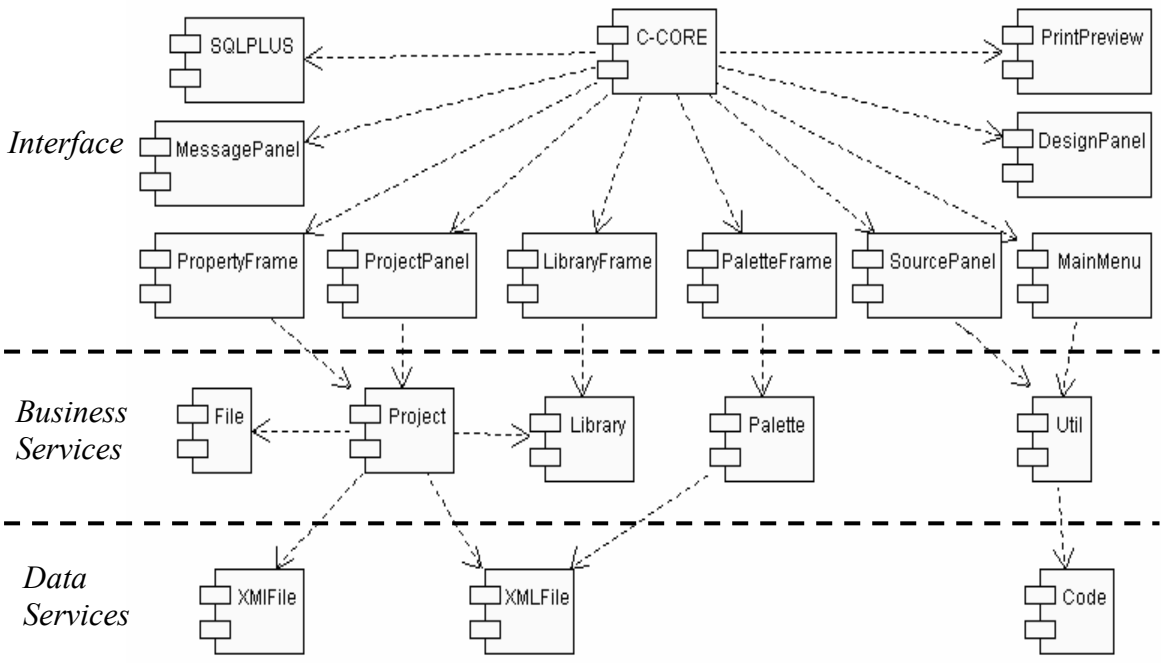


Figura 22 - Modelo de Componentes da Ferramenta, Organizado Segundo a Arquitetura de Três Camadas

No desenvolvimento da ferramenta, devido ao grande número de componentes e para que se tenha uma melhor visualização, os Modelos de Componentes foram organizados em pacotes, na arquitetura de 3 (três) camadas conforme mostra a Figura 23.

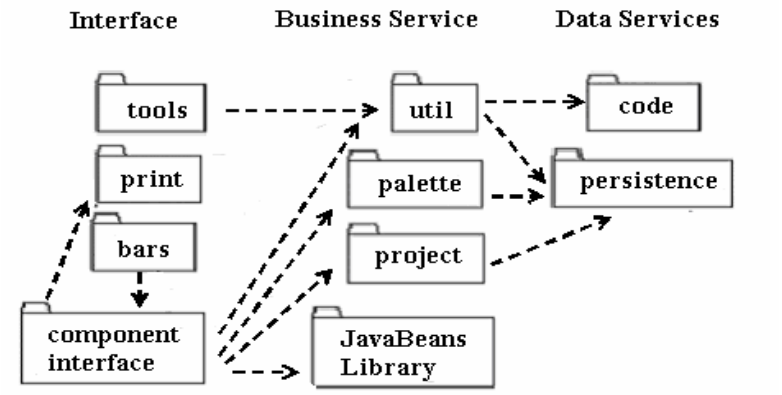


Figura 23 - Modelo de Componentes Organizados em Pacotes

Na camada *Interface* tem-se os pacotes: *tools*, que permite incluir ferramentas de apoio ao desenvolvimento, como por exemplo, as de acesso a banco de dados; *print*, que permite visualizar a impressão e imprimir os artefatos do projeto; *bars*, que contém tanto as funções disponíveis na barra de ferramentas quanto as na barra de menu; e *componentInterface* que agrupa os componentes da interface da ferramenta, incluindo painel de desenvolvimento – *DesignPanel* – e, também, o inspetor de objetos – *ObjectInspector*.

Na camada *Business Service* têm-se os pacotes: *util*, que reúne componentes que auxiliam na representação dos artefatos manipulados pela ferramenta; *code*, que é responsável por toda a geração de código das interfaces GUI criadas em um projeto; *palette*, que contém a paleta de componentes e os mecanismos para o gerenciamento dos componentes adicionados a ela; *project*, que contém o componente *project* bem como os mecanismos para o gerenciamento de seus artefatos e bibliotecas; e *JavaBeansLibrary*, que agrupa bibliotecas de componentes *JavaBeans* que podem ser reusadas no desenvolvimento de software.

Na camada *Data Services* tem-se o pacote *persistence*, que contém funções relacionadas à persistência do projeto e seus artefatos em banco de dados. Além disso, têm-se aquelas que permitem exportar o projeto para outros formatos.

Ainda na etapa de projeto interno dos componentes, também foram levados em consideração os requisitos não funcionais, com destaque para a persistência dos objetos construídos ou gerados pela C-CORE. No caso, a persistência dos objetos é feita na linguagem XMI, permitindo compartilhamento com demais ferramentas que ofereçam suporte para tal linguagem.

4.1.4 Implementação da C-CORE

Neste passo foi realizada a implementação dos componentes da C-CORE. Os componentes da ferramenta foram construídos segundo a arquitetura *JavaBeans* da linguagem Java. Os componentes que se encontram na camada de "Interface" foram implementados com bibliotecas do pacote "javax.swing". Dentre estes componentes tem-se o componente C-CORE que utiliza os outros componentes pertencentes a camada de "Interface", compondo, assim, a interface gráfica da ferramenta C-CORE. A Figura 24 exibe a interface principal da C-CORE com uma aplicação sendo construída, reutilizando componentes disponíveis nas paletas. Também, são destacados na Figura 24, os itens estruturais da ferramenta: barra de ferramentas (1); paleta de componentes (2); inspetor de objetos (3); editor de código (Source) (4); e editor visual de interfaces gráficas (Design) (5).

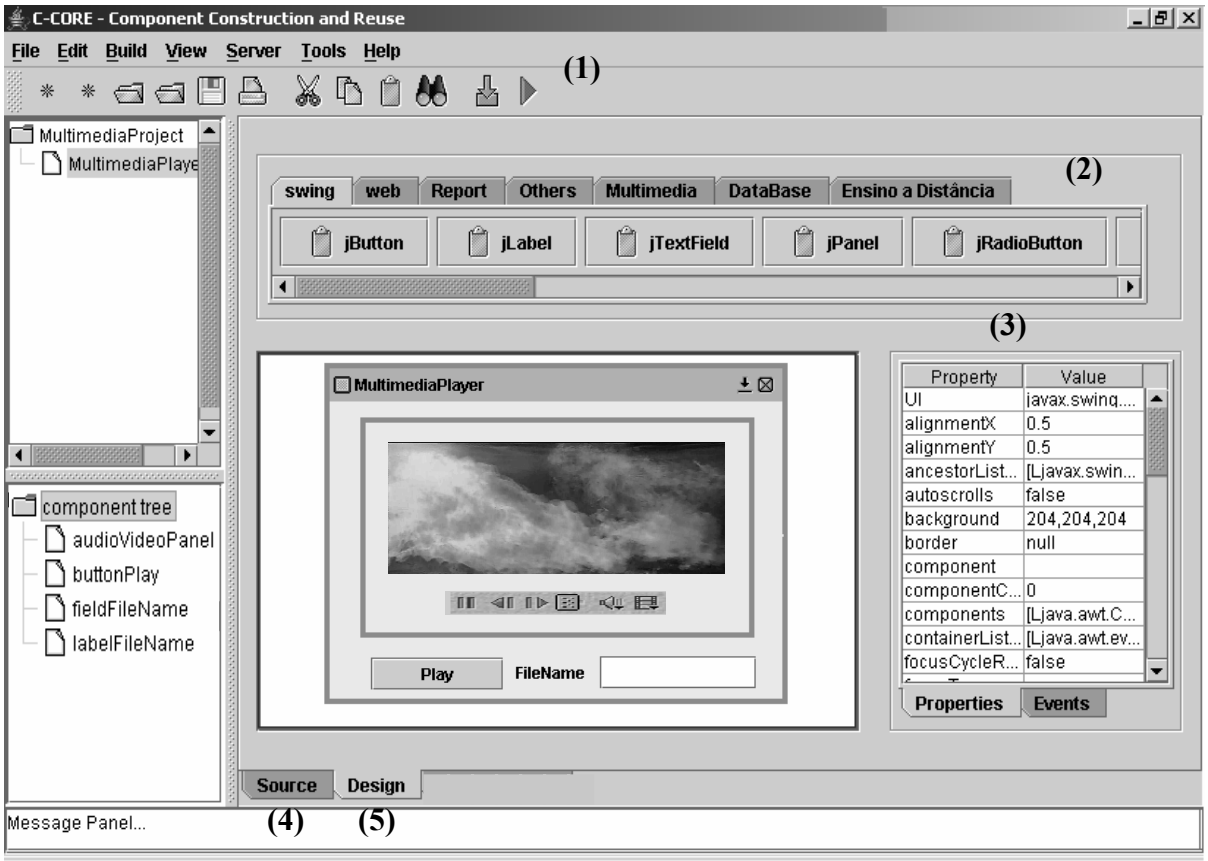


Figura 24 - Interface principal da C-CORE com uma aplicação sendo desenvolvida

Outro componente da ferramenta é o gerenciador de projeto (componente *ProjectPanel* e *Project*) que controla os artefatos de um projeto de software, incluindo o código gerado e as representações gráficas das interfaces GUI. Todas as informações de um projeto, com exceção das representações gráficas das interfaces GUI usadas na C-CORE e outras informações de projeto, são armazenadas em XMI (componente *XMIFile*), permitindo

o compartilhamento com outras ferramentas que ofereçam suporte nesta linguagem. Por exemplo, a Figura 25 mostra parte do arquivo XMI de uma Aplicação implementada na C-CORE, onde são destacados: a classe da Aplicação (1), seus atributos (2) e seus métodos (3). A Figura 26 mostra uma visão do mesmo arquivo XMI carregado na ferramenta MVCASE.

```
- <UML:Model xmi.id="a1" name="New Project" isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false">
  - <UML:Namespace.ownedElement>
    + <UML:Package xmi.id="a2" name="Use Case View" isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false">
    - <UML:Package xmi.id="a4" name="Logical View" isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false">
      + <UML:ModelElement.stereotype>
      - <UML:Namespace.ownedElement>
( 1 )   <UML:Class xmi.id="a6" name="MultimidiaPlayer" isSpecification="false" isRoot="false" isLeaf="false"
          isAbstract="false" isActive="false">
        + <UML:ModelElement.stereotype>
        + <UML:GeneralizableElement.generalization>
        - <UML:Classifier.feature>
( 2 )     + <UML:Attribute xmi.id="a10" name="audioVideoPanel" isSpecification="false" ownerScope="instance">
          + <UML:Attribute xmi.id="a13" name="buttonPlay" isSpecification="false" ownerScope="instance">
          + <UML:Attribute xmi.id="a16" name="fieldFileName" isSpecification="false" ownerScope="instance">
          + <UML:Attribute xmi.id="a19" name="labelFileName" isSpecification="false" ownerScope="instance">
          + <UML:Attribute xmi.id="a22" name="c" isSpecification="false" ownerScope="instance">
( 3 )     + <UML:Operation xmi.id="a25" name="buttonPlayClick" isSpecification="false" ownerScope="instance"
              isQuery="false" isRoot="false" isLeaf="false" isAbstract="false">
          + <UML:Method xmi.id="a29" isSpecification="false" isQuery="false">
          + <UML:Operation xmi.id="a31" name="initComponents" isSpecification="false" ownerScope="instance"
              isQuery="false" isRoot="false" isLeaf="false" isAbstract="false">
          + <UML:Method xmi.id="a34" isSpecification="false" isQuery="false">
          + <UML:Operation xmi.id="a36" name="MultimidiaPlayer" isSpecification="false" ownerScope="instance"
              isQuery="false" isRoot="false" isLeaf="false" isAbstract="false">
          + <UML:Method xmi.id="a39" isSpecification="false" isQuery="false">
          </UML:Classifier.feature>
        </UML:Class>
```

Figura 25 - Parte do arquivo XMI de uma aplicação

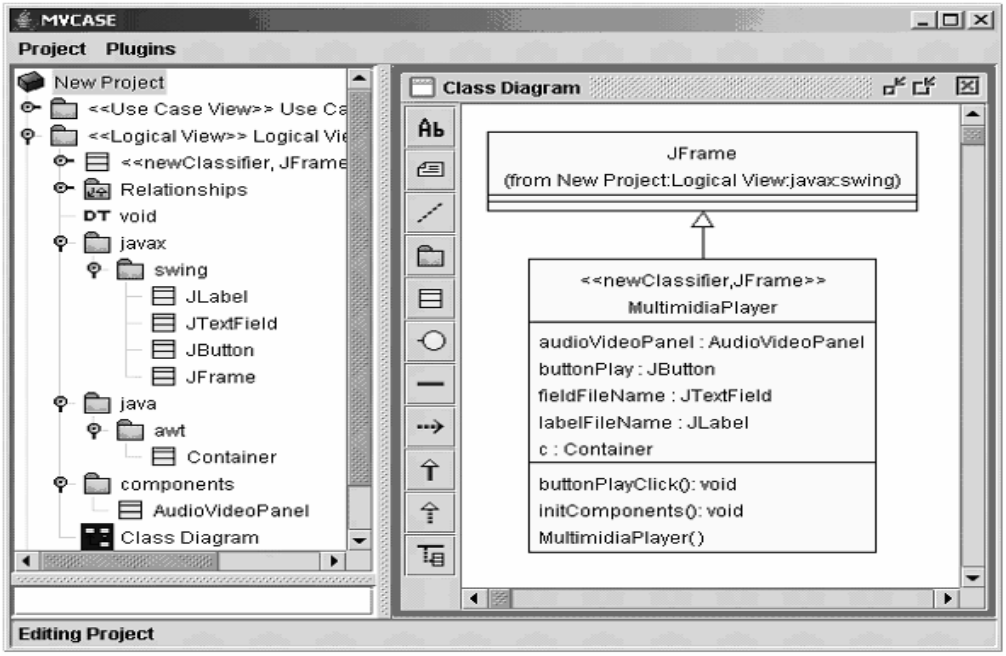


Figura 26 – Arquivo XMI de uma aplicação carregado na MVCASE

Para as representações gráficas das interfaces GUI usadas na C-CORE, e outras informações do projeto, a ferramenta C-CORE gera um arquivo XML (componente *XMLFile*). As especificações gráficas do projeto são armazenadas em XML para beneficiar-se deste suporte oferecido pela linguagem Java. Este arquivo é usado em conjunto com o arquivo XMI para representar um projeto de software na C-CORE. A Figura 27 mostra parte do arquivo XML de uma Aplicação implementada na C-CORE, onde são destacados: o "classPath" do projeto (1), seu conteúdo (2), a representação gráfica da interface GUI (3) e o nome do projeto (4).

```
<?xml version="1.0" encoding="UTF-8" ?>
- <java version="1.4.2_04" class="java.beans.XMLDecoder">
  - <object class="c_core.project.Project">
( 1 )
    + <void property="classPath">
( 2 )
    - <void property="content">
      - <void method="add">
        - <object class="java.util.Vector">
          - <void method="add">
              <string>C:\Ferramenta\projects\MultimediaPlayer.java</string>
            </void>
          + <void method="add">
          - <void method="add">
              <string>JFrame</string>
            </void>
          - <void method="add">
              <string>C:\Ferramenta\projects</string>
            </void>
          </object>
        </void>
      </void>
( 3 )
    - <void property="frames">
      - <void method="add">
        - <object class="c_core.componentInterface.DesignFrame">
          + <void property="m_contentPanel">
          + <void property="newSize">
          - <void property="title">
              <string>MultimediaPlayer</string>
            </void>
          </object>
        </void>
      </void>
( 4 )
    - <void property="name">
        <string>MultimediaProject</string>
      </void>
```

Figura 27 - Parte do arquivo XML de uma aplicação

Para o desenvolvimento de software, a ferramenta C-CORE dispõe de recursos gráficos e textuais, que facilitam o desenvolvimento de componentes e suas aplicações. Estes recursos podem ser usados em conjunto, tornando o desenvolvimento de componentes mais interativo e permitindo que Engenheiros de Software, com diferentes conhecimentos em DBC, possam realizar seu trabalho com melhor qualidade e maior produtividade.

Como recurso textual, tem-se o editor de texto (Figura 28), no qual o código do software é escrito diretamente na linguagem Java. O Editor possui recursos de orientação à sintaxe, no caso da linguagem Java, e outros como, por exemplo, *highlight*, que permite destacar palavras reservadas.

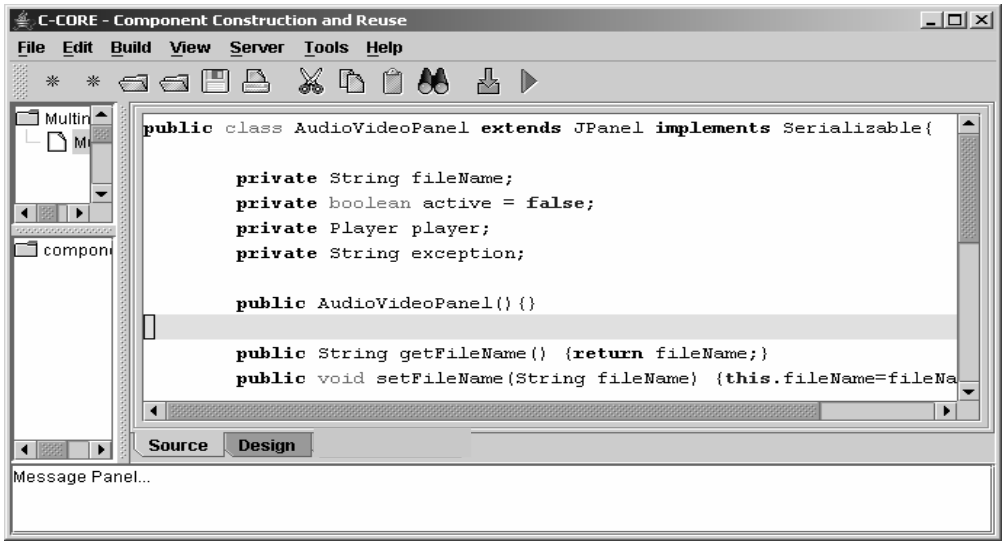


Figura 28 - Recurso Textual da ferramenta C-CORE

Como recurso gráfico, a ferramenta C-CORE oferece o "*DesignPanel*" (Figura 29) que permite o Engenheiro de Software desenvolver software de forma interativa, reusando componentes que já foram construídos. Estes componentes podem ser refinados com o apoio do *Object Inspector* que permite a interação direta em suas propriedades e eventos. À medida que o Engenheiro de Software realiza alterações nos componentes, é gerado automaticamente código (componente *Code* da ferramenta) no componente no Editor de Código. Esse código pode ser complementado conforme as necessidades específicas do componente, como por exemplo, no detalhamento do comportamento de um método.

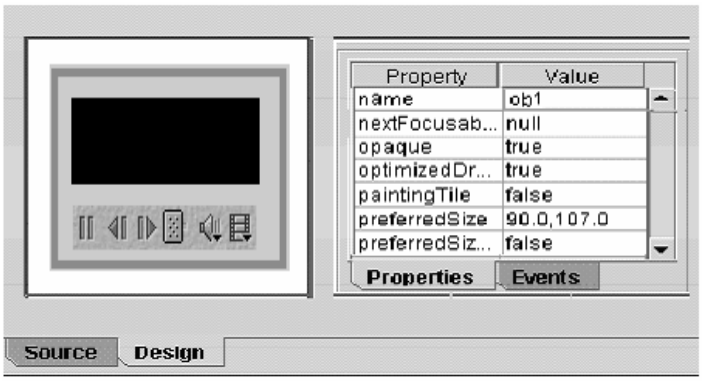


Figura 29 - Recurso Gráfico da ferramenta C-CORE

A Figura 30 apresenta na MVCASE tanto parte do modelo de Classe de uma Aplicação (a) quanto parte de sua representação XMI (b), além do repositório (c), que armazena efetivamente toda a especificação do projeto. Essa figura também apresenta código Java (d) representando parte do modelo de classe da Aplicação, o qual se encontra disponível para alteração no editor de texto da C-CORE. Essas informações são carregadas na ferramenta mediante seu gerador, e atualizadas no repositório através de seu interpretador.

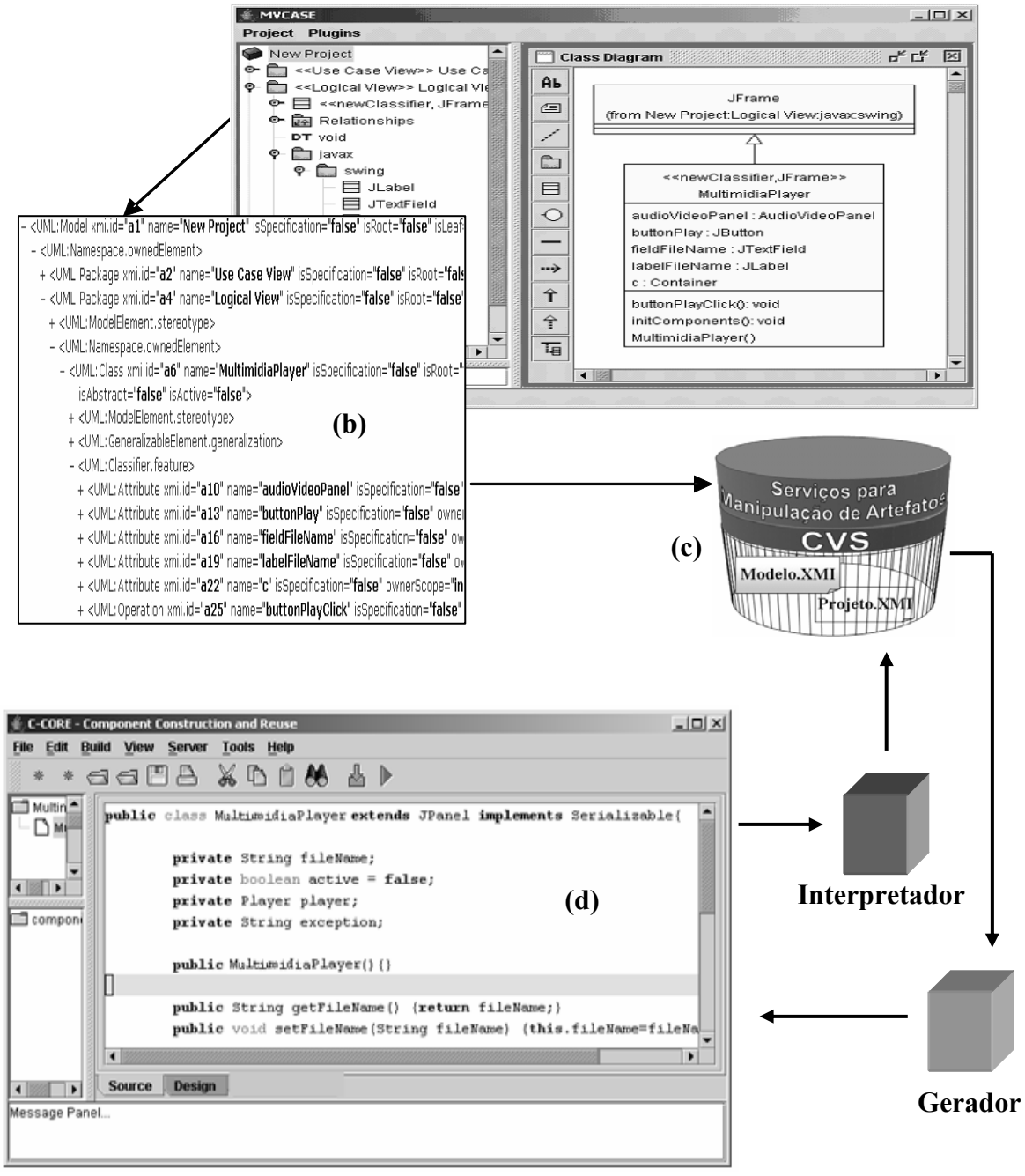


Figura 30 – Integração das Ferramentas MVCASE e C-CORE

A composição básica da C-CORE já inclui componentes para os domínios de GUI e de acesso a banco de dados. À medida que novos componentes são construídos, estes podem ser disponibilizados para reuso, importando-os para dentro da ferramenta C-CORE (componente *PaletteFrame*). No caso de componentes implementados segundo a tecnologia *JavaBeans*, faz-se o *deployment* dos componentes, disponibilizando-os em paletas, organizadas segundo seus domínios. Componentes construídos com outras tecnologias, como no caso de EJB, ficam disponíveis em bibliotecas, que são importadas diretamente pelas aplicações. A Figura 31 mostra, por exemplo, uma paleta de componentes *JavaBeans* do domínio Multimídia, disponíveis para reuso.

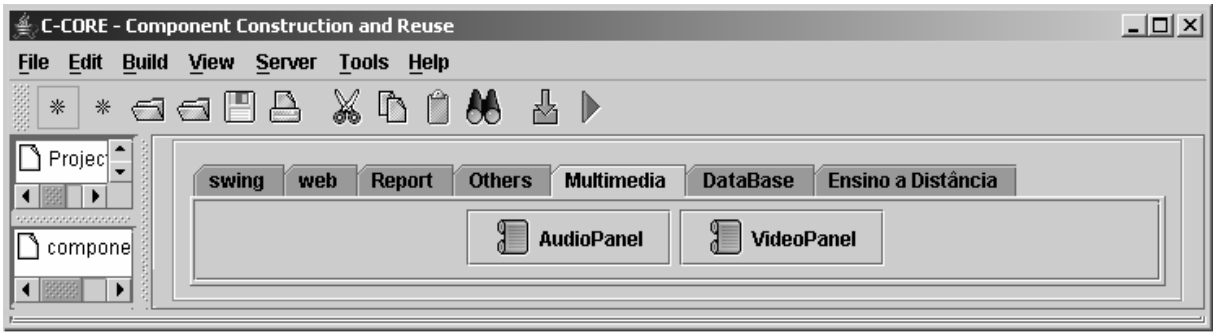


Figura 31 - Componentes do Domínio Multimídia disponíveis na C-CORE

Outros recursos disponíveis na C-CORE permitem compilar e executar suas aplicações em plataforma cliente e servidor. Estes recursos facilitam o desenvolvimento, pois testes podem ser aplicados como forma de detecção de erros durante a implementação. A ferramenta dispõe ainda de recursos para *deployment* dos componentes e aplicações.

4.2 Considerações Finais

Neste capítulo apresentou-se a C-CORE, uma ferramenta de programação orientada a componentes centrada na linguagem Java. Foi desenvolvida segundo *Catalysis* por ser um método atual e que cobre grande parte do desenvolvimento baseado em componentes.

Apesar do pouco tempo aplicado no desenvolvimento da C-CORE em relação ao de outras ferramentas de programação, a C-CORE apresenta auxilio a grande parte das funcionalidades normalmente encontradas nas ferramentas de programação. Destaca-se, no caso, a persistência do projeto em um arquivo XMI, o que permite a integração com ferramentas do domínio de modelagem. Esta característica tem sido bastante explorada, objetivando minimizar inconsistências entre os artefatos produzidos nas fases de projeto e

implementação do software. Essa exigência de mercado pode ser comprovada por meio da disponibilidade, mesmo que parcial, dessa funcionalidade em algumas ferramentas de programação disponíveis no mercado.

Uma outra parte das funcionalidades do domínio de ferramentas de programação, no entanto, ainda não faz parte do conjunto de funcionalidades da C-CORE. Essas incluem *Refactoring Support, que* oferece suporte automatizado na alteração e inclusão de código do projeto*; Version Control,* que permite vários desenvolvedores atuarem na construção de software, pois garante gerenciamento e controle eficaz dos diferentes artefatos produzidos na C-CORE; e *Test Support* que dá maior agilidade na verificação do comportamento do software produzido na ferramenta. Suportando essas funcionalidades faltantes, a C-CORE proporcionaria significativa redução de tempo e aumento da qualidade no desenvolvimento de componentes e suas aplicações.

No próximo capítulo, apresenta-se o desenvolvimento de 4 (quatro) estudos de caso, usados para testar a ferramenta C-CORE, e sua integração com a ferramenta MVCASE.

Capítulo 5

Estudos de Caso

A ferramenta C-CORE apresentada no Capítulo 4 e integrada no ambiente ADSBC, descrito no Capítulo 3, foi testada com o desenvolvimento de 4 (quatro) estudos de caso. Primeiro foi construído um *framework* de componentes do domínio EAD, implementado em EJB. Em seguida, foi desenvolvida uma aplicação reutilizando os componentes do *framework*. Outro estudo de caso foi a construção de uma biblioteca de componentes, implementada em *JavaBeans*, denominada *DBCLIB*. Finalmente foi desenvolvida outra aplicação que reutiliza componentes da biblioteca *DBCLIB*.

5.1 Framework EAD

Trata-se de um *framework* para Ensino a Distância que suporta a construção de sistemas para elaboração e gerenciamento de cursos, a partir do reuso de seus componentes. Atende principalmente aos requisitos relacionados com os recursos e serviços didáticos, administrativos e suporte aos cursos. Segue-se uma apresentação do seu desenvolvimento, sendo que os 3 primeiros passos baseiam-se no método Catalysis e foram realizados pelo Engenheiro de Software com apoio da ferramenta MVCASE. O passo da implementação foi realizado pelo Engenheiro de Software com apoio da C-CORE.

5.1.1 Domínio do Problema

Seguindo Catalysis, no primeiro nível, Domínio do Problema, foi realizado um estudo do domínio de Educação a Distância [Sanches, 2002]. Os requisitos identificados foram modelados através de técnicas como *storeboards* e *mind-maps*, que posteriormente foram refinados em um Modelo de Colaboração. A Figura 32 mostra um Modelo de Colaboração do *framework*, obtido do refinamento de *mind-map*, com os atores: Administrador, Autor, Professor e Aluno e as ações: Definir Ator, Elaborar Curso, Realizar Curso e Gerenciar Curso.

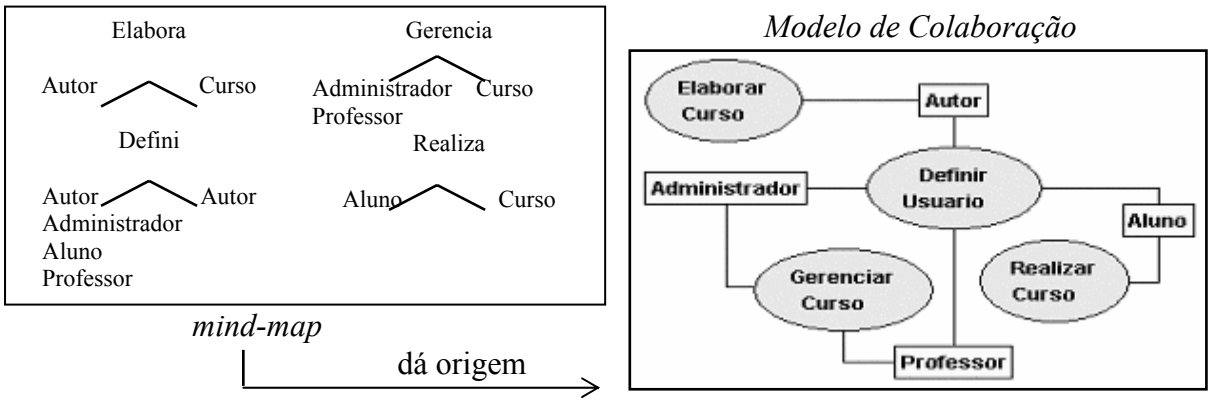


Figura 32 - Modelo de Colaborações do framework de Educação a Distância

Definido as principais ações, fez-se o refinamento do Modelo de Colaboração, buscando um melhor entendimento das funcionalidades do *framework*, especificando-se seus Casos de Uso. A Figura 33 mostra um Modelo de Casos de Uso, obtido do refinamento do Modelo de Colaboração da Figura 32. Ações podem ser modificadas ou adicionadas, como por exemplo, os Casos de Uso elaborarCurso, elaborarAulaCurso, elaborarConteudoAula e elaborarExercicios, identificados a partir da ação Elaborar Curso.

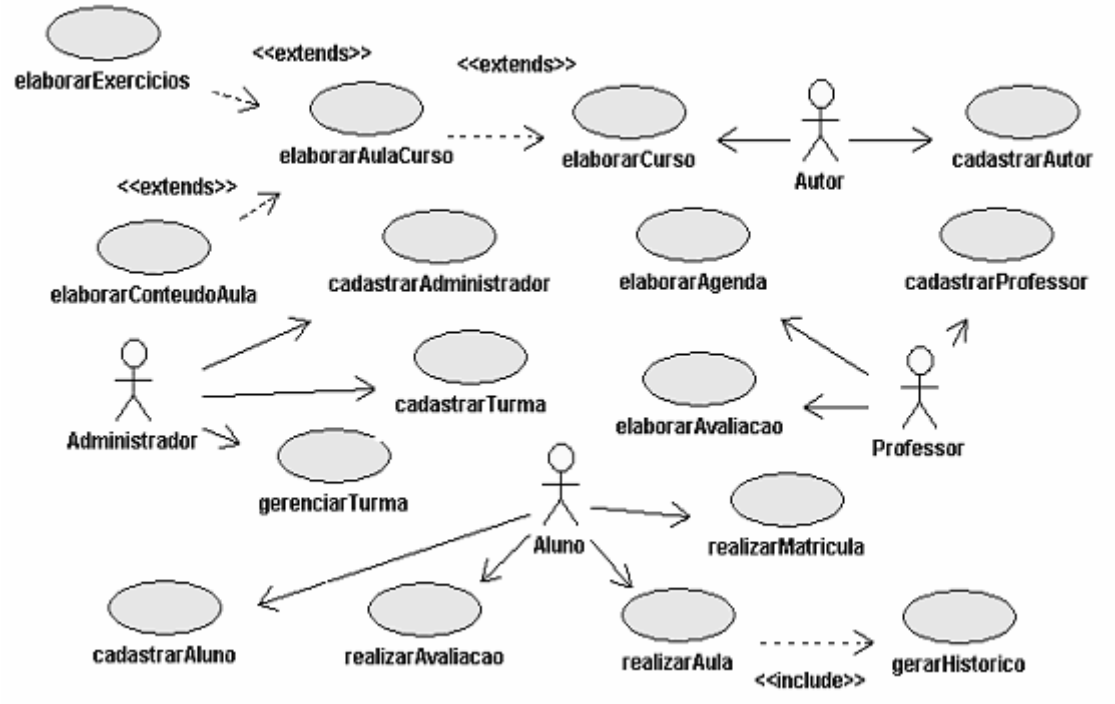


Figura 33 - Modelo de Use Cases do *framework* de Educação a Distância

Outra técnica utilizada neste nível do Domínio do Problema foi o *snapshot* que é a descrição, geralmente como um desenho, de um conjunto de objetos e os valores de alguns de

seus atributos em um particular ponto no tempo. A Figura 34 mostra um *snapshot* dos objetos Curso, Aula e Conteúdo.

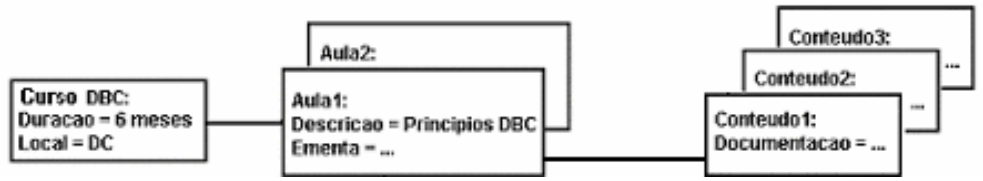


Figura 34 - *Snapshot* dos objetos: Curso, Aula e Conteúdo

5.1.2 Especificação dos Componentes

Neste segundo nível de Catalysis, foram refinados os modelos do Domínio do Problema, especificando-se o Modelo de Tipos do *Framework*. O modelo de tipos do *framework* foi obtido pelo refinamento dos modelos de Casos de Uso, que representa os comportamentos dos objetos e suas conexões de mensagens, e dos modelos *snapshot* que representa os atributos dos objetos e seus relacionamentos. A Figura 35 apresenta o Modelo de Tipos que mostra os principais tipos do *framework*, destacando os atributos e comportamento dos tipos FrCurso, FrAula e FrConteudo. Os símbolos "< >" indicam que os tipos são genéricos para o domínio da aplicação, visando o reuso.

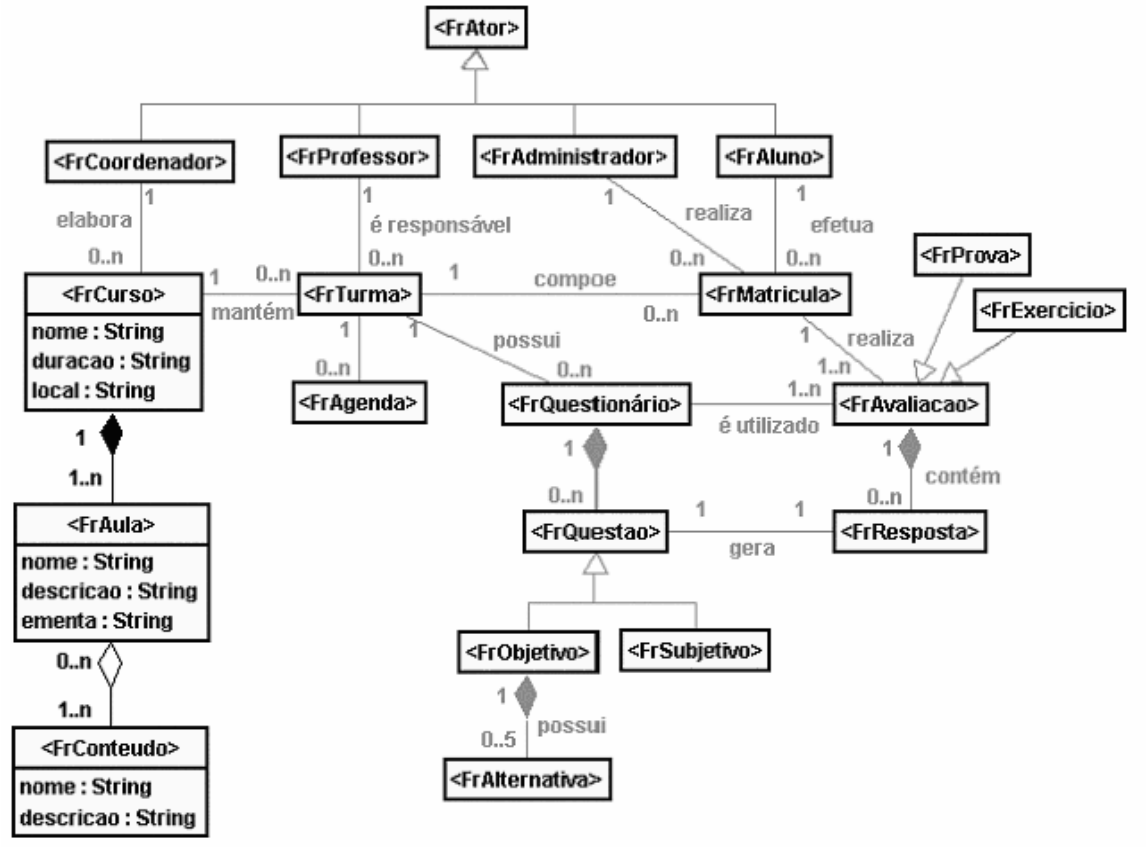


Figura 35 - Modelo de Tipo do Framework de Educação a Distância

5.1.2 Projeto Interno dos Componentes

Neste terceiro nível de Catalysis foram modeladas as classes, obtidas pelo refinamento dos tipos do *framework* especificado no passo anterior. Em seguida, as classes deram origem aos componentes. Devido ao grande número de componentes e para uma melhor visualização, o Modelo de Classes do *framework* foi agrupado em pacotes, especificando-se o Modelo de Pacotes do *framework*. A Figura 36 mostra o Modelo de Pacotes do *framework*. O pacote EAD reutiliza componentes dos pacotes *java* e *ejb*, e contém outros pacotes agrupados em três camadas: *Interface*, *Business Service* e *Data Service*. Na camada *Interface* tem-se o pacote Aplicação que suporta a construção de aplicações que utilizam serviços da camada *Business Service*. A camada *Business Service* corresponde às regras de negócio do *framework*, e a camada *Data Service* mantém os serviços de acesso a Banco de Dados. O pacote Básico contém componentes de tratamento de exceções e outras funções básicas do *framework*. O pacote Aplicação tem componentes para acesso aos beans do tipo Session. O pacote Regras de Negócio é composto dos pacotes Definição Ator, Elaboração Curso, Realização Curso e Gerenciamento Curso. Estes pacotes contêm componentes bean do tipo Entity. O pacote Conexão, da camada *Data Service*, contém os componentes que gerenciam a conexão e acesso a banco de dados relacionais.

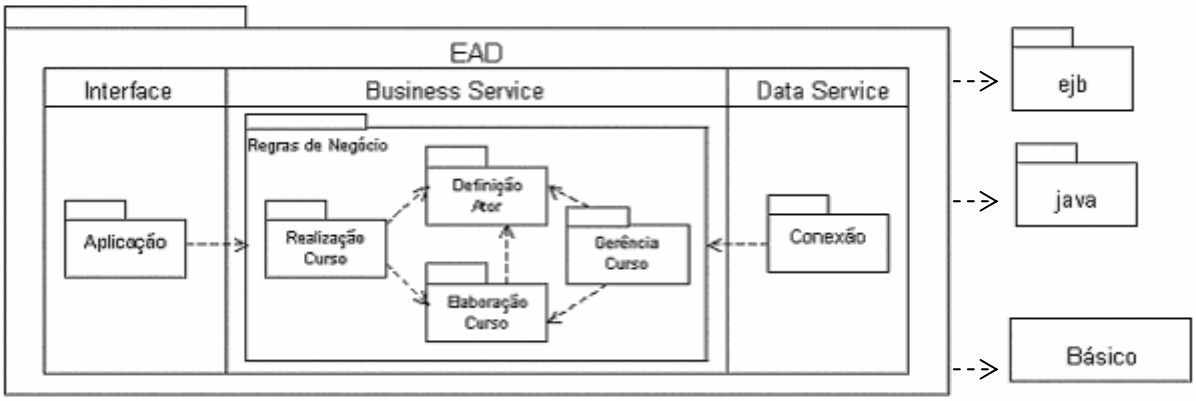


Figura 36 - Modelo de Pacotes de Componentes do *Framework*

5.1.3 Implementação dos Componentes

Neste nível, foram implementados os componentes do *framework* segundo a tecnologia EJB. Este passo foi realizado com o auxilio da C-CORE. Assim, dando continuidade ao projeto modelado na MVCASE, o Engenheiro de Software abre o projeto na C-CORE e continua refinando-o, agora detalhando o seu código. A Figura 37 mostra o projeto do *framework* (1) aberto na C-CORE. Esta figura também mostra parte do código do componente

Administrador no Editor de Código da ferramenta (2). No caso, para a implementação deste componente foi importada a biblioteca J2EE para o projeto.

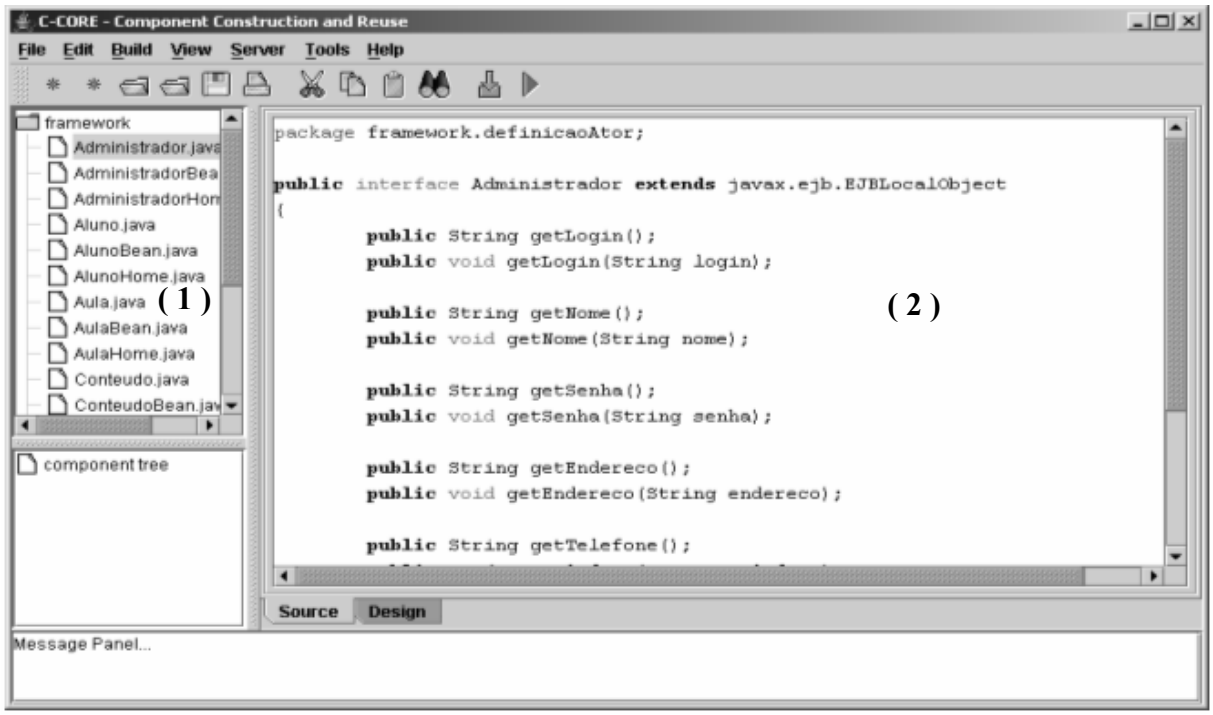


Figura 37 - Projeto do *framework* na C-CORE

À medida que se desenvolve o projeto na C-CORE, pode-se, através da interação com a MVCASE, obter a sua modelagem na MVCASE. A Figura 38 mostra, por exemplo, o projeto do *framework* lido novamente na MVCASE.

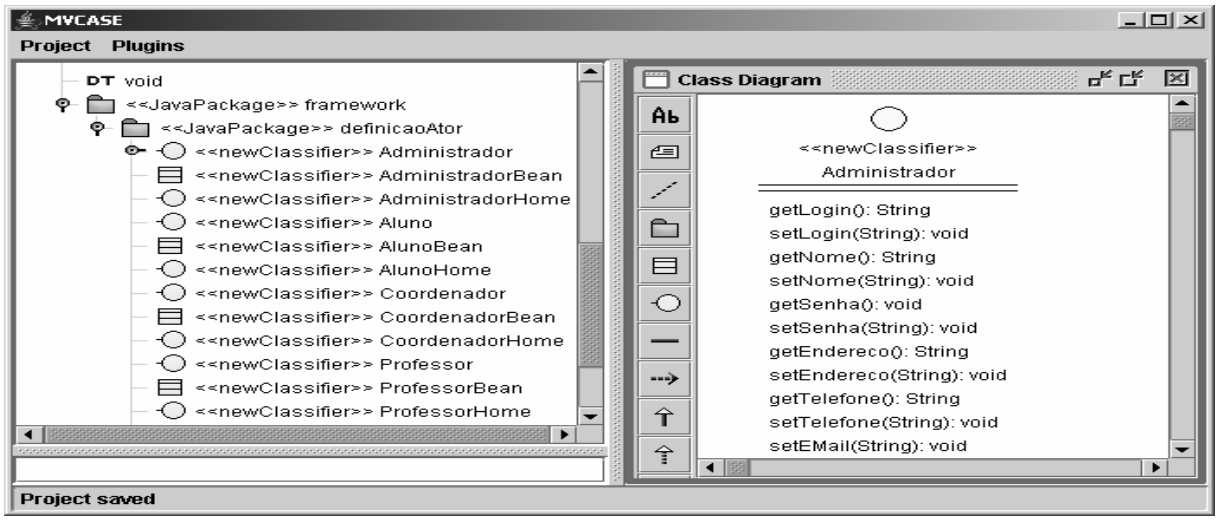


Figura 38 - Projeto do *Framework* carregado na MVCASE

Ao término da implementação faz-se o *deployment* dos componentes no servidor de componentes J2EE da SUN, tornando-os disponíveis para a construção de aplicações.

5.2 Aplicação EAD reutilizando o *framework* EAD

Diferentes aplicações do domínio EAD podem ser desenvolvidas fazendo reuso dos componentes do *framework* de EAD construído. A Figura 39 resume uma das aplicações, que ilustra o reuso dos componentes.

> Trata-se de uma aplicação que suporta o desenvolvimento de cursos para ensino, de aplicativos, como, HTML, Flash e outros. O sistema tem uma interface interativa que possibilita que os interessados se matriculem em um ou mais cursos que são oferecidos periodicamente. Um curso tem um responsável, autor do curso, e um ou mais professores que acompanham o curso. Todas as aulas de um curso são registradas pela data e horário. A matrícula de um aluno é aprovada ou não pelo administrador do sistema, após a aprovação do cadastro pessoal do aluno. O sistema deve controlar o tipo de permissão de acesso dos usuários: administrador, autor, professor, e aluno. O sistema controla ainda as avaliações e exercícios dos cursos.

Figura 39 - Aplicação de Educação a Distancia

Da mesma forma, os 3 primeiros passos do desenvolvimento, correspondentes à modelagem, foram realizados com o apoio da MVCASE. Em seguida, prossegue-se na C-CORE com a implementação da aplicação.

5.2.1 Domínio do Problema

Neste passo foi estudado o domínio do problema, identificando-se os requisitos da aplicação. Antes de iniciar a especificação dos requisitos da aplicação, na MVCASE, importou-se os componentes do domínio do problema, no caso Educação a Distancia, que serão reutilizados pela aplicação. A Tabela 3 lista os principais Casos de Uso desta aplicação. Para cada Caso de Uso tem-se uma breve descrição, as entradas e respostas da aplicação.

Tabela 3: Lista de Casos de Uso da Aplicação de Ensino a Distância

Descrição	Entrada	Caso de Uso	Resposta
Administrador faz cadastro	DadosAdministrador	cadastrarAdministrador	Msg01
Administrador gerencia curso	DadosGerenciamento	gerenciarCurso	Msg07
Autor faz cadastro	DadosAutor	cadastrarAutor	Msg02
Professor faz cadastro	DadosProfessor	cadastrarProfessor	Msg03
Aluno faz cadastro	DadosAluno	cadastrarAluno	Msg04
Autor elabora curso	DadosCurso	elaborarCurso	Msg05
Autor elabora agenda das aulas	DadosAgenda	elaborarAgenda	Msg06
Aluno realiza matrícula	DadosMatrícula	realizarMatrícula	Comprovante de Matrícula
Aluno realiza aula do curso	DadosAula	realizarAula	Msg08, Certificado
Aluno realiza avaliações do curso	DadosAvaliação	realizarAvaliação	Msg09, Nota

Outro modelo construído nesta etapa foi o de Casos de Uso, mostrado na Figura 40, que representa os cenários de interação dos atores, Administrador, Aluno, Autor e Professor, com a aplicação.

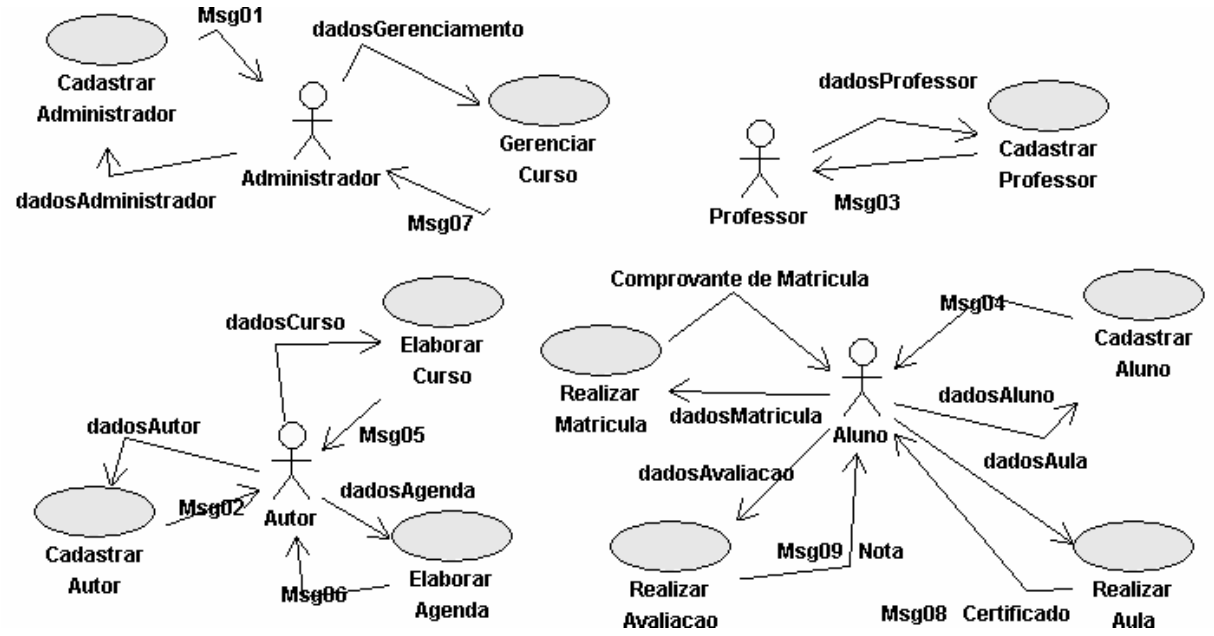


Figura 40 - Modelo de Casos de Uso da Aplicação de Educação a Distância

Técnicas, como *mind-maps*, *snapshots* e outras, também podem ser usadas na especificação dos requisitos neste primeiro passo.

5.2.2 Especificação da Aplicação

Neste passo foram refinados os modelos especificados no passo anterior, fazendo reuso dos componentes do *framework*. Por exemplo, a Figura 41 mostra um Modelo de Tipos que gerencia os participantes de um curso, especificado neste passo. Os tipos sombreados (entre < >), são reutilizados do *framework* construído na primeira fase.

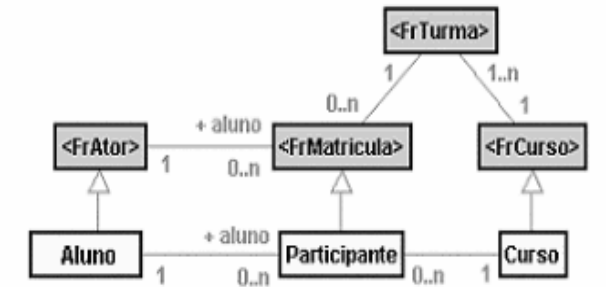


Figura 41 - Modelo de Tipos da Aplicação de Educação a Distância

5.2.3 Projeto da Aplicação

Neste passo foi desenvolvido o projeto interno dos componentes da aplicação. A Figura 42 mostra um Modelo de Classes da aplicação, onde Administrador reutiliza o componente Ator do *framework*. O Modelo de classes é obtido pelo refinamento do Modelo de Tipos especificado no passo anterior.

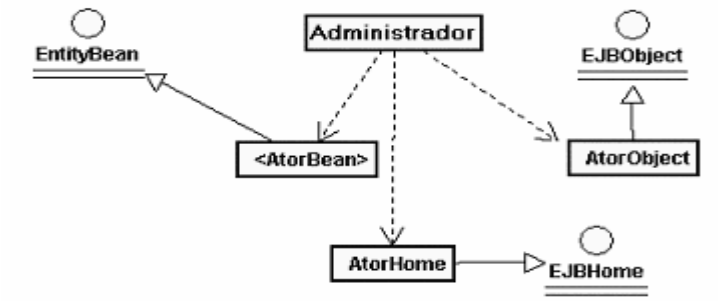


Figura 42 - Administrador reutilizando o componente Ator

5.2.4 Implementação e testes da Aplicação

Neste passo fez-se a implementação da aplicação com base no seu projeto, e numa primeira versão da implementação gerada na MVCASE, a partir dos Modelos de Classe e Componentes. A Figura 43 mostra o código EJB gerado para o componente Administrador. Em seguida, prosseguiu-se com a implementação na C-CORE. A Figura 44mostra detalhes da implementação da interface da aplicação na ferramenta C-CORE.

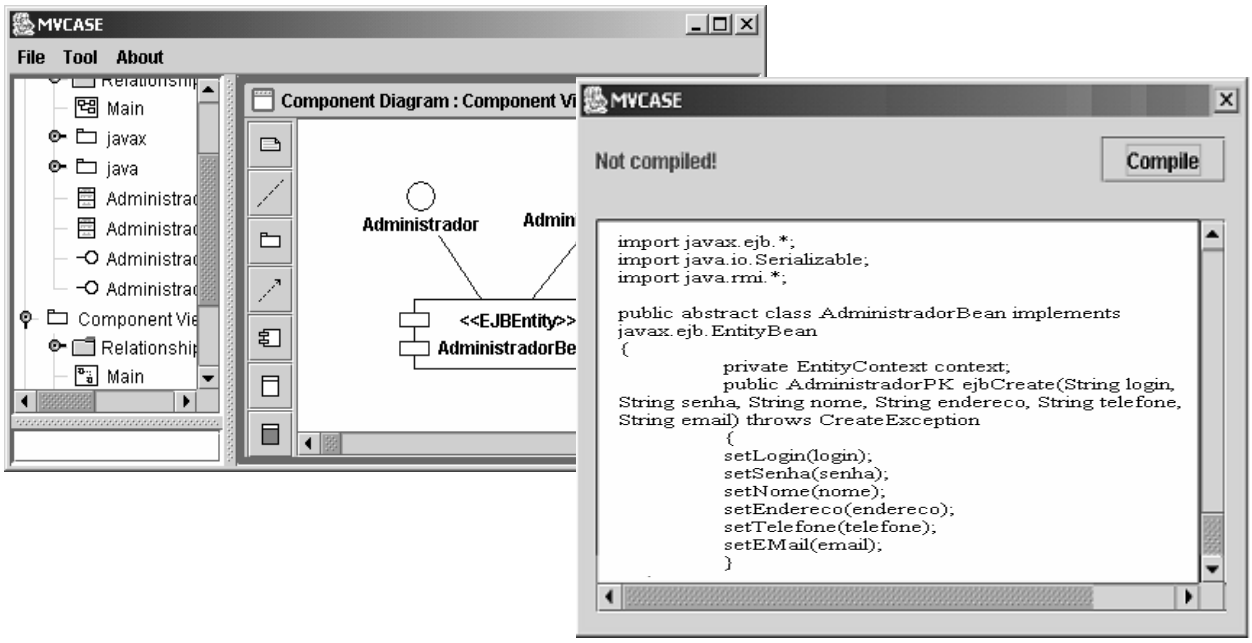


Figura 43 - Código gerado para o componente Administrador

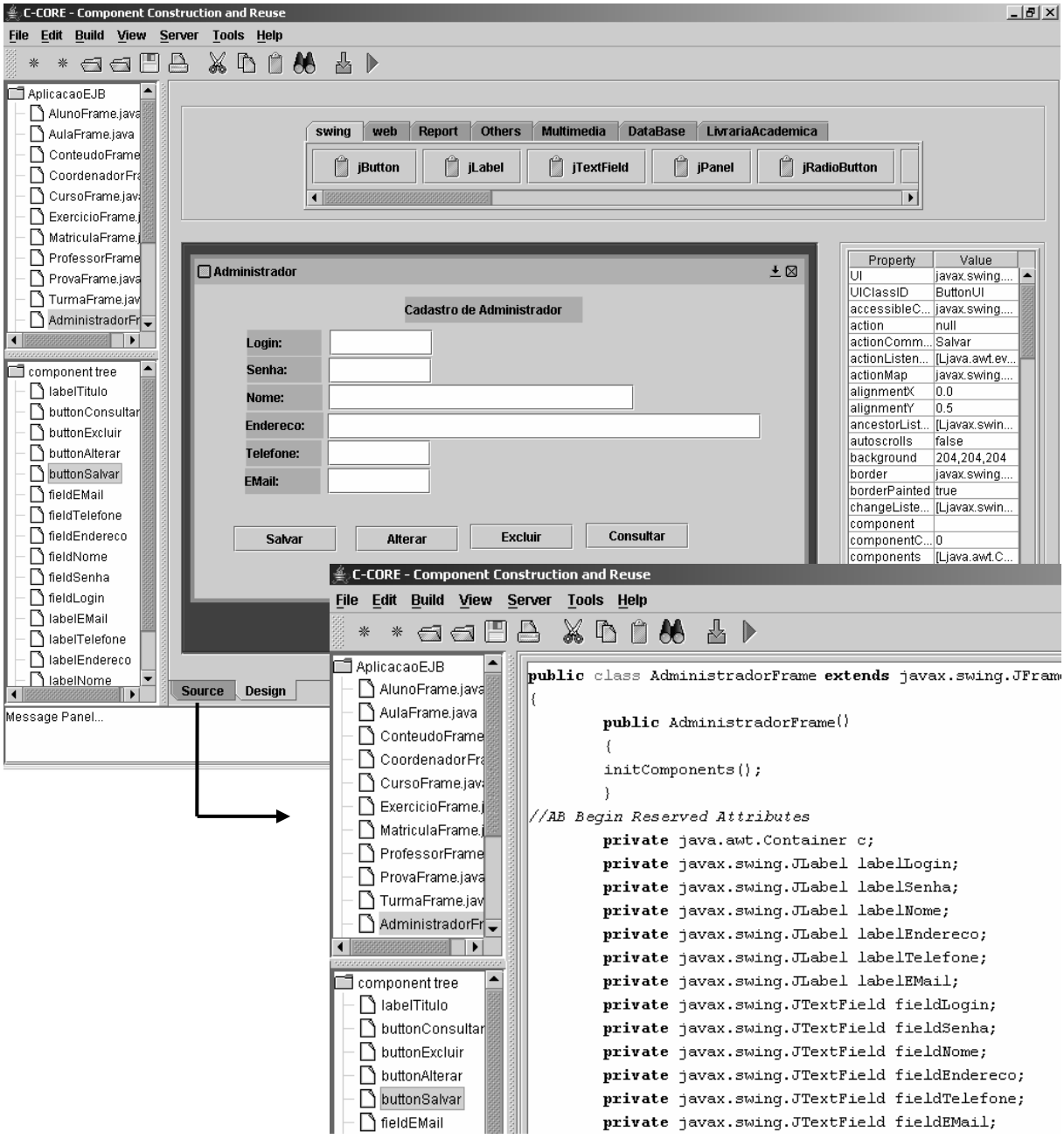


Figura 44 - Detalhes da implementação da interface da aplicação EJB na C-CORE

À medida que se desenvolve o projeto na C-CORE, pode-se, através da interação com a MVCASE, obter sua modelagem. A Figura 45 mostra o projeto da aplicação lido novamente na MVCASE.

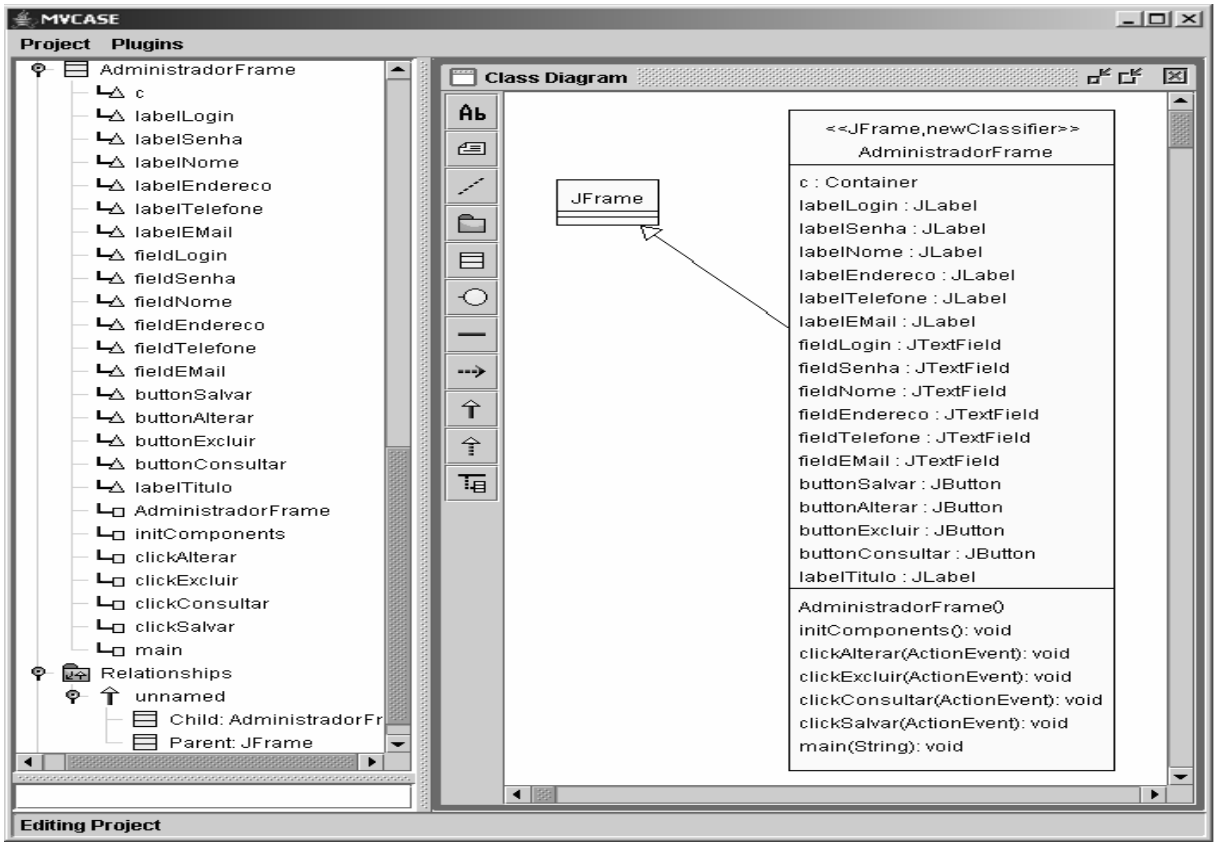


Figura 45 - Projeto da aplicação EJB carregado na MVCASE

Os testes com a aplicação possibilitam identificar erros da aplicação, dos componentes do *framework* ou da própria ferramenta C-CORE. Dependendo do erro retorna-se aos passos anteriores para corrigi-los. O processo de refinamento prossegue até que se tenha uma versão correta seja da aplicação, dos componentes ou da própria C-CORE.

5.3 Biblioteca de componentes para banco de dados (*DBCLIB)*

Outro estudo de caso consistiu na construção de uma biblioteca de componentes, denominada *DBCLIB,* que tratam do acesso e persistência de dados em SGBDs relacionais, e que podem ser reutilizados em aplicações de diferentes domínios. Foi desenvolvida segundo o método *Catalysis*, e implementada em *JavaBeans* na C-CORE.

5.3.1 Domínio do Problema

No primeiro nível de Catalysis foram identificados os requisitos do *DBCLIB* a partir de estudos do domínio de acesso e persistência em banco de dados.

Os requisitos identificados inicialmente foram modelados usando *mind-maps*, visando representar as diferentes situações e cenários de sua utilização. Os *mind-maps* dão origem aos

modelos dos cenários de uso do *DBCLIB*. São modelos que representam as ações que podem ocorrer no domínio do problema e os objetos participantes.

O Engenheiro de Software é um dos atores que utiliza o *DBCLIB* em aplicações de diferentes domínios, que acessam e persistem suas informações em bancos de dados. Duas principais ações identificadas são: definir o acesso e o controle de dados. A Figura 46 mostra um Modelo de Colaboração, obtido do refinamento de um Mind-*Map* que representa o Engenheiro de Software e suas ações "Definir Acessos aos Dados" e "Definir Controles dos Dados". Os atores são representados por retângulos e suas ações por elipses.

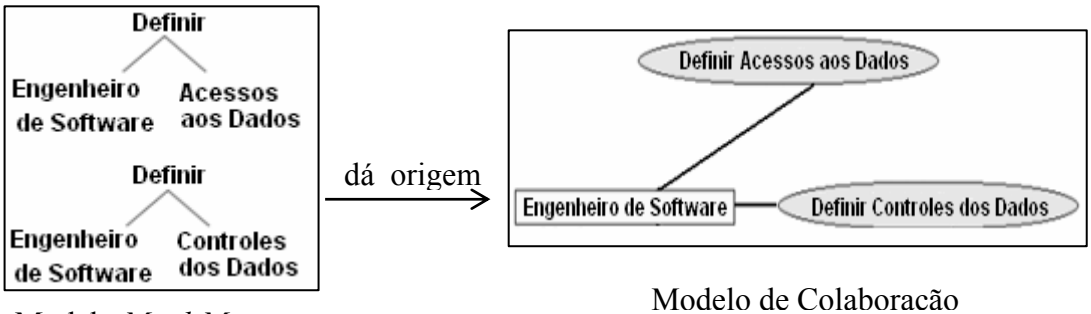


Figura 46 - Modelos: Mind-Map e Colaboração das ações do Engenheiro de Software da *DBCLIB*

O Modelo de Colaboração é refinado, buscando um melhor entendimento das funcionalidades, especificando-se seus Casos de Uso. Os Casos de Uso representam cenários das interações dos atores com aplicações que reusam componentes da biblioteca, e são agrupados em contextos, como por ator, para facilitar sua compreensão. A Figura 47 mostra um Modelo de Casos de Uso para as ações "Definir Acesso aos Dados" e "Definir Controles dos Dados", obtida do refinamento do Modelo de Colaboração da Figura 46.

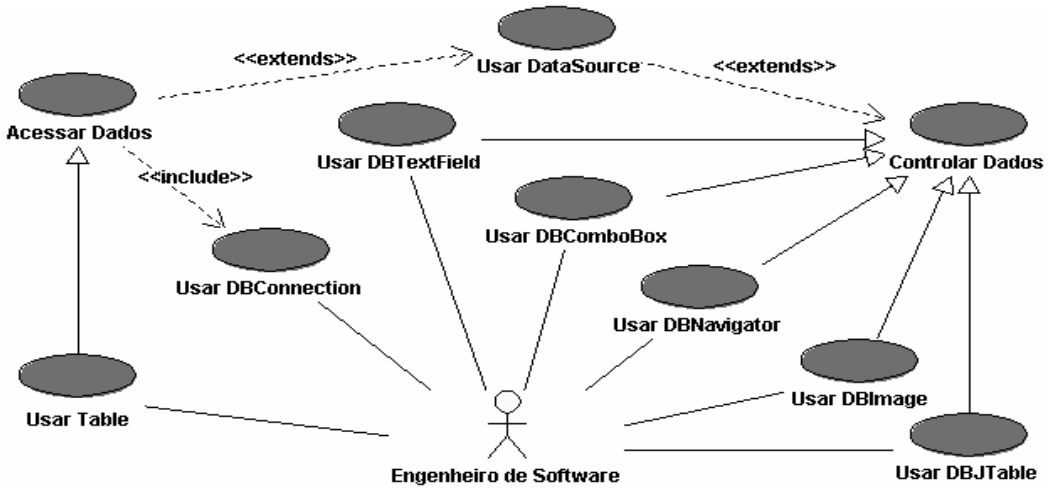


Figura 47 - Modelo de Casos de Uso da *CDBLIB* obtido a partir do refinamento do Modelo de Colaboração

Nesta etapa do desenvolvimento, funcionalidades podem ser modificadas ou adicionadas, como por exemplo, os Casos de Uso relacionados com estereótipo "<<include>>", e que representam funcionalidades adicionais para as ações inicialmente identificadas. Outros modelos podem ser utilizados neste nível do Domínio do Problema como, por exemplo, o modelo de *snapshots* que é a descrição, geralmente com um desenho, de um conjunto de objetos identificados no Domínio do Problema, com valores de seus atributos em um particular ponto no tempo. A Figura 48 mostra um modelo de *snapshots* do conjunto de objetos que fazem conexão (*DBConnection*), e acesso (*Table*) a banco de dados.

Figura 48 - Snapshots dos objetos de conexão e acesso a banco de dados do DBCLIB

5.3.2 Especificação dos Componentes

No segundo nível de *Catalysis*, foi descrito o comportamento externo dos componentes da biblioteca de uma forma não ambígua. Os modelos de Casos de Uso, que representam os comportamentos dos objetos e suas conexões de mensagens, e os modelos de *snapshots* que representam os atributos dos objetos e seus relacionamentos, foram refinados em um Modelo de Tipos. A Figura 49 mostra um modelo de Tipos destacando, da biblioteca, os principais tipos e atributos relacionados com a ação "Definir Acesso aos Dados".

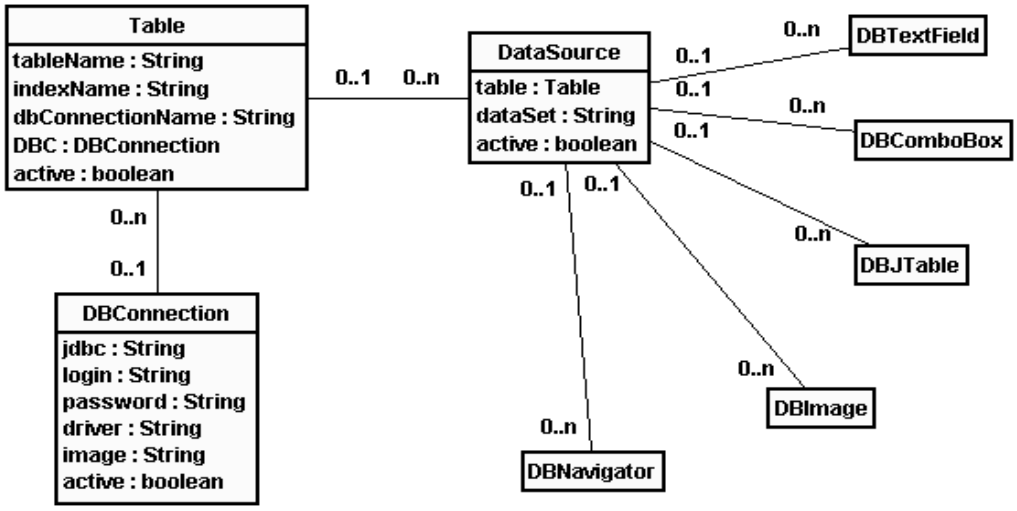


Figura 49 - Modelo de Tipos da DBCLIB

5.3.3 Projeto Interno dos Componentes

No terceiro nível de *Catalysis* foram modeladas as classes e seus relacionamentos, refinando os modelos de tipos construídos no segundo nível.

A Figura 50 mostra um modelo de classes, obtido do refinamento do modelo com os tipos: *DBConnection*, *Table* e *DataSource*. Considerando a plataforma de software adotada para implementação com a tecnologia *JavaBeans*, todos os objetos são derivados da classe javax.swing.JPanel do Java, e têm um relacionamento de dependência com a classe java.awt.BorderLayout, pois serão componentes que podem ser alterados em tempo de design em uma ferramenta de composição, no caso a C-CORE.

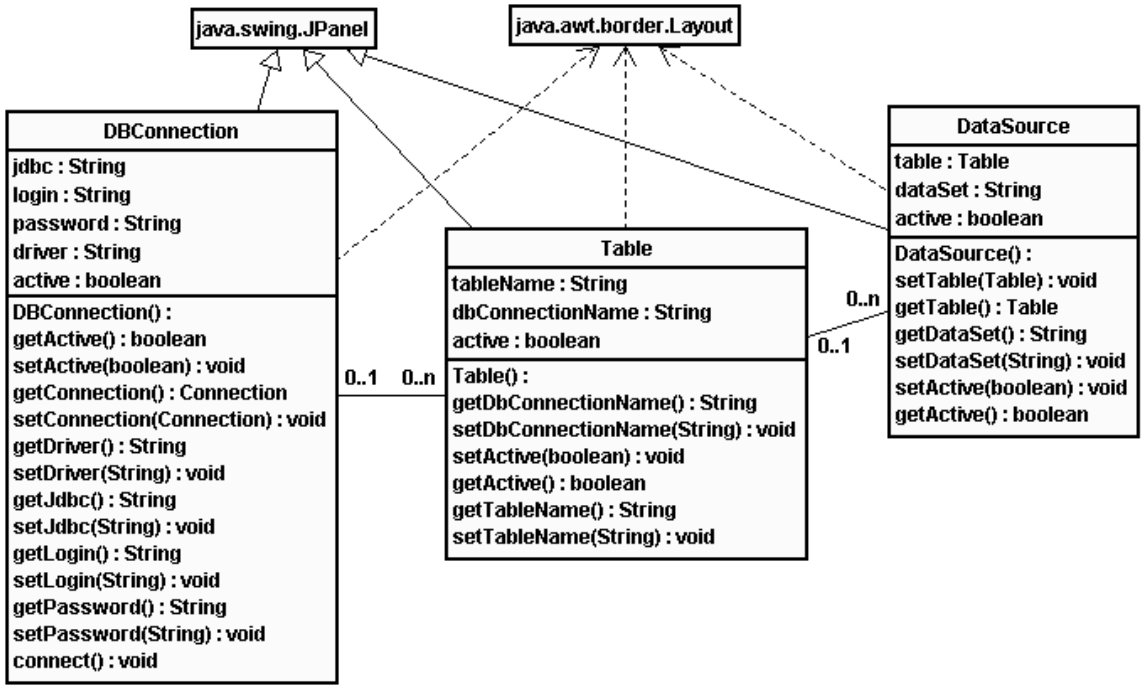


Figura 50 - Modelo de classes dos componentes: DBConnection, Table e DataSource

Finalmente, os modelos de classes dos componentes deram origem aos componentes, que disponibilizam suas funcionalidades através de suas interfaces. No caso, foram implementados os seguintes componentes do *DBCLIB*: DBConnection, Table DataSource, DBJTable, DBNavigator, DBComboBox, DBTextField e DBImage.

Para melhor visualização, o Modelo de Classes dos componentes foi organizado em pacotes conforme mostra a Figura 51. O pacote DataAccess tem componentes que acessam e manipulam dados de um banco de dados como, por exemplo, DBConnection, Table, DataSource e outros. O pacote DataControls tem componentes que controlam e visualizam os dados como, por exemplo, DBJTable, DBTextField, DBComboBox e outros.

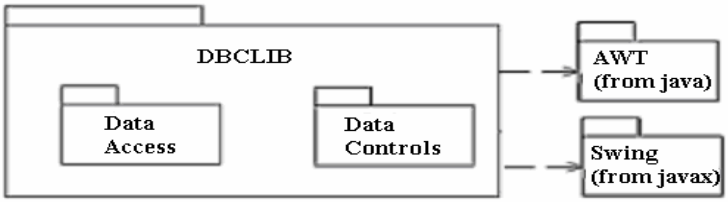


Figura 51 - Modelo de Pacotes da biblioteca *DBCLIB*

5.3.4 Implementação e "deployment" dos Componentes

Neste passo fez-se a implementação dos componentes da biblioteca com o apoio da ferramenta C-CORE. Os componentes foram construídos segundo a arquitetura *JavaBeans* e usam classes reflexivas, para suportar introspecção. A introspecção analisa o código objeto do componente e passa para ferramentas de composição (como a C-CORE), as propriedades, que representam o estado e a apresentação do componente, e os eventos que o estimulam. A edição dos valores das propriedades e das ações dos eventos (interface *runtime*) é suportada por editores próprios para cada caso. Para que se tenha uma idéia deste passo, a Figura 52 mostra parte da implementação do componente Table, onde é destacada parte do código que ativa o componente. No método "setActive" são definidas as ações que o componente deve executar em modo de Design (1) ou Execução (2) .

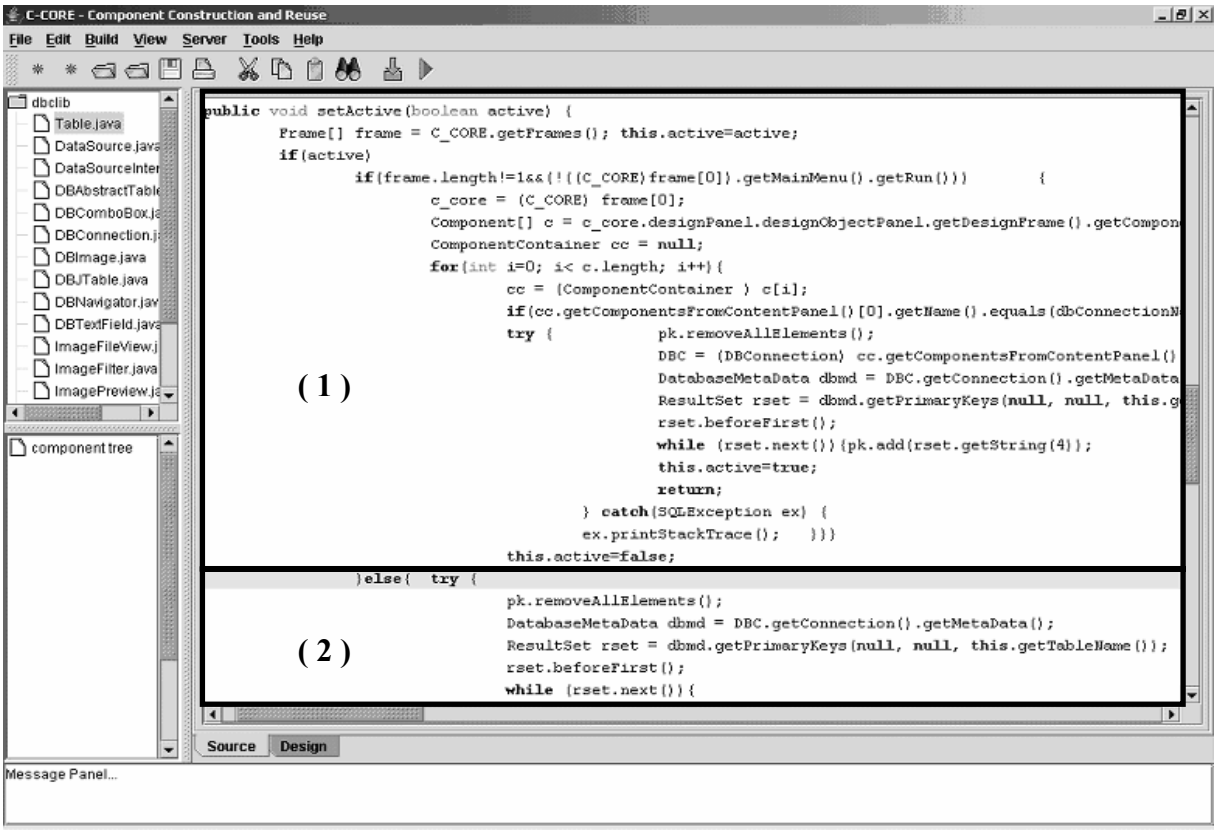


Figura 52 - Implementação do componente Table

Uma vez construídos, os componentes do *DBCLIB* podem ser reutilizados no desenvolvimento de diferentes aplicações baseadas em banco de dados. Para se reutilizar os componentes da biblioteca na ferramenta C-CORE faz-se o seu *deployments*, disponibilizando-os em paletas. A Figura 53 mostra uma paleta na C-CORE com componentes do *DBCLIB*.

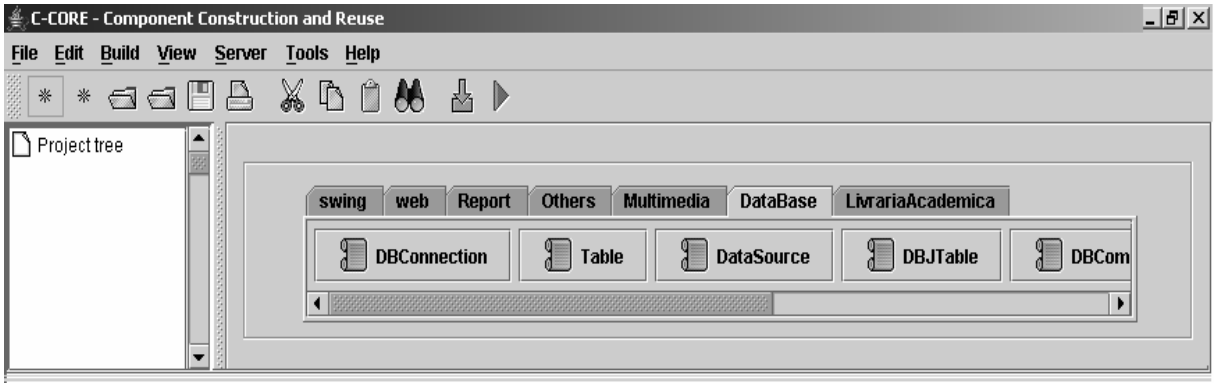


Figura 53 - Componentes do *DBCLIB* disponíveis na C-CORE

5.4 Aplicação EAD reutilizando componentes da *DBCLIB*

Diferentes Aplicações baseadas em banco de dados podem ser desenvolvidas fazendo reuso dos componentes da *DBCLIB*. A Figura 39 resume uma das aplicações, que será usada para mostrar o desenvolvimento com componentes.

Segue-se uma apresentação sobre o processo de desenvolvimento fazendo reuso dos componentes da *DBCLIB*. Para o desenvolvimento do Estudo de Caso, foram utilizadas as ferramentas MVCASE, para modelagem, e a C-CORE para implementação.

5.4.1 Domínio do Problema

Considerando que a aplicação é a mesma que foi implementada com componentes EJB, nesta fase, a modelagem é a mesma apresentada na seção 5.2.1.

5.4.2 Especificação da aplicação

Neste passo foram refinados os modelos do domínio do problema, fazendo reuso dos componentes da *DBCLIB*. Por exemplo, do refinamento dos Modelos de Casos de Uso e *snapshot*, do primeiro passo, foi obtido o Modelo de Tipos da Aplicação. A Figura 54 mostra,

por exemplo, parte do Modelo de Tipos da Aplicação, fazendo reuso de classes dos componentes da *DBCLIB*: DBConnection, Table, DBJTable, DBTextField, DBNavigator e DataSource.

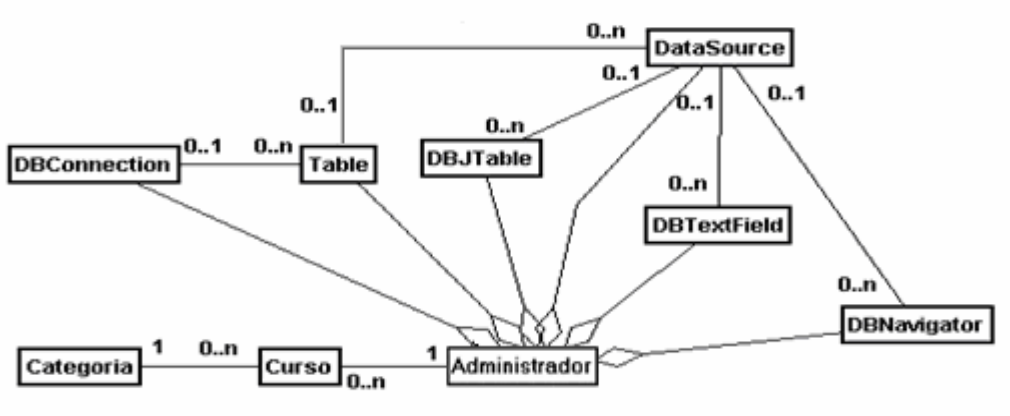


Figura 54 - Parte do Modelo de Tipos da Aplicação

5.4.3 Projeto da Aplicação

Neste passo foi desenvolvido o projeto da aplicação considerando os requisitos não funcionais, conforme a plataforma de hardware e software adotada. A Figura 55 mostra um Modelo de Classes da aplicação, obtido do refinamento do Modelo de Tipos. No modelo as classes dos componentes de controles visuais da *DBCLIB* (DBTextField, DBJTable e DBNavigator) são reutilizadas pela classe AdministradorFrame do componente da aplicação.

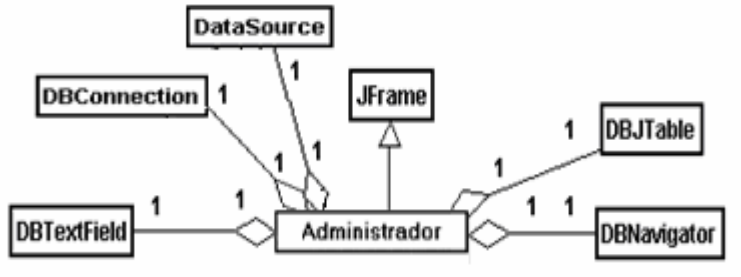


Figura 55 - Parte do Modelo de Classes da Aplicação

5.4.4 Implementação e testes da Aplicação

Finalmente na C-CORE, a aplicação foi implementada e testada. A Figura 56 mostra a interface GUI do cadastro de Administrador, reutilizando componentes da *DBCLIB* disponíveis na C-CORE.

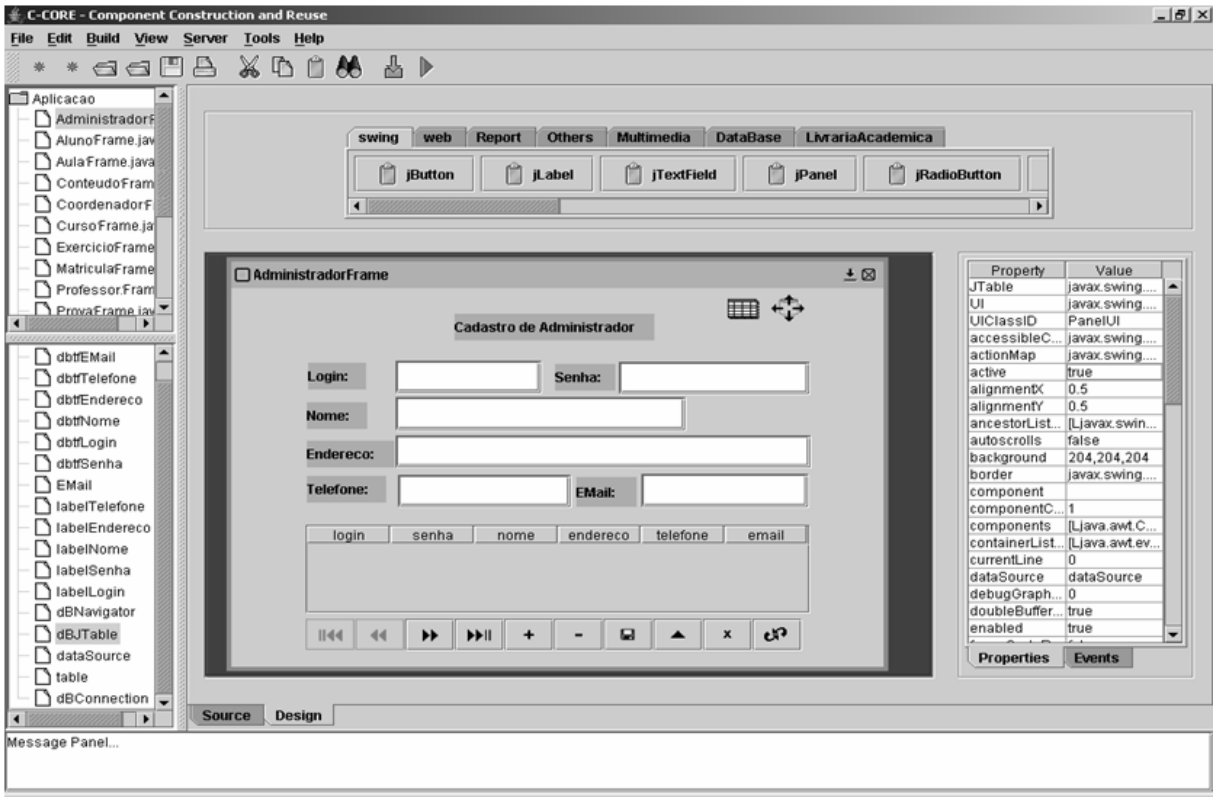


Figura 56 - Interface GUI do Cadastro de Administrador, reutilizando componentes da *DBCLIB*

À medida que se desenvolve o projeto na C-CORE, pode-se, através da interação com a MVCASE, obter a sua modelagem. A Figura 57 mostra o projeto da aplicação lido novamente na MVCASE.

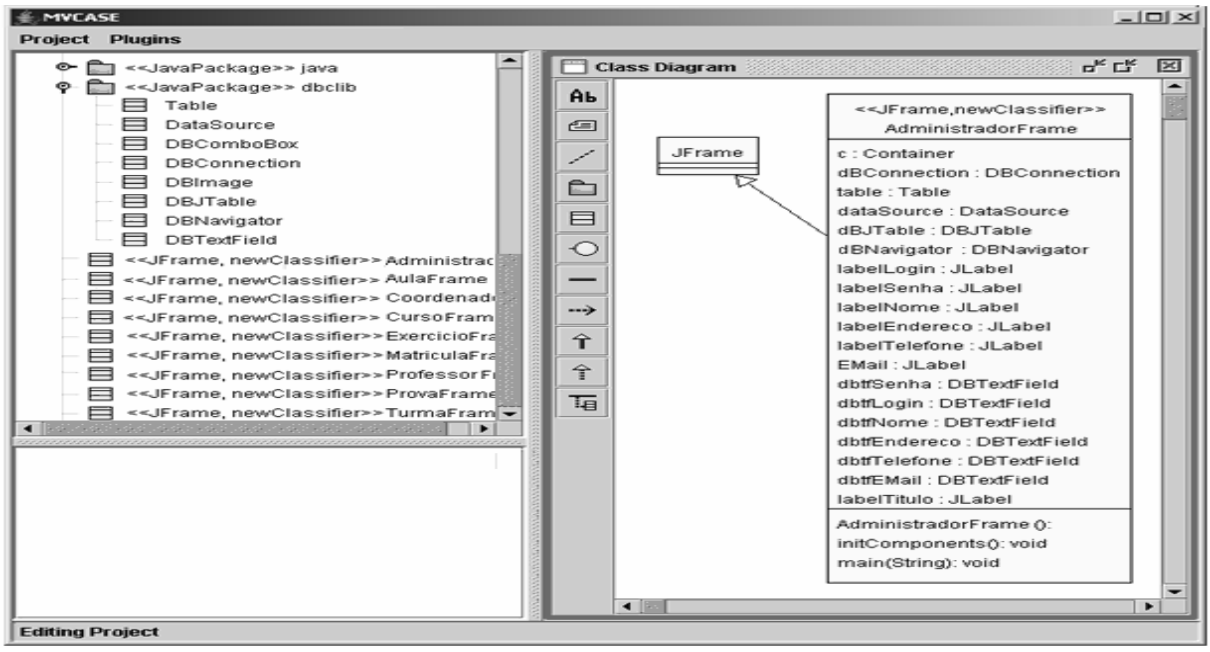


Figura 57 - Projeto da aplicação *JavaBeans* carregado na MVCASE

Os testes com a aplicação permitem verificar se a mesma atende aos requisitos identificados no domínio do problema. Neste momento o Engenheiro de Software pode retornar aos passos anteriores do desenvolvimento da aplicação ou mesmo dos componentes para corrigir possíveis erros.

5.5 Considerações Finais

Neste capítulo foram apresentados quatro estudos de caso do domínio de Ensino a Distância, construídos com o apoio das ferramentas C-CORE e MVCASE. Esta foi utilizada nas fases de análise e projeto; aquela foi usada na fase de implementação. Para esta fase, a C-CORE apresenta diferentes recursos, como por exemplo, o desenho de GUI com geração automática de código. A codificação do software por intermédio desse recurso interativo reduz drasticamente os trabalhos realizados na fase de implementação. Ainda nesta fase, o Engenheiro de Software pode realizar livremente alterações em um dado artefato de software, pois C-CORE está integrada com a MVCASE, aumentando a consistência dos artefatos de software das fases de projeto e de implementação. Esses recursos minimizam o esforço despendido pelos Engenheiros de Software com tarefas de garantia de qualidade do software produzido.

Os estudos de caso forneceram “feed-back”, o que permitiu a evolução do protótipo da ferramenta. Os resultados obtidos com a construção dos estudos de caso também mostraram a viabilidade da C-CORE na construção tanto de componentes quanto de aplicações que reusam componentes.

O próximo capítulo apresenta as conclusões, contribuições e trabalhos futuros deste trabalho.

Capítulo 6

Conclusão

Este trabalho apresentou uma ferramenta de programação orientada a componentes, denominada C-CORE. A C-CORE, desenvolvida segundo o método *Catalysis* e a tecnologia de componentes *JavaBeans*, é capaz de auxiliar o Engenheiro de Software na construção de componentes e suas aplicações com maior produtividade e melhor qualidade.

Embora já se tenha conhecimento de ferramentas programação semelhante a C-CORE, esta se diferencia principalmente pelo suporte à implementação dirigida pelos modelos de projeto, e à construção e reuso de componentes implementados em EJB e *JavaBeans*. A implementação dirigida pelos modelos torna mais consistente a documentação com o código dos componentes e suas aplicações. O suporte ao DBC possibilita tanto a construção dos componentes quanto o reuso. O reuso pode ser através de *framework* de componentes ou de bibliotecas de componentes instalados nas paletas da C-CORE.

A construção de ferramentas CASE demanda tempo, experiência e o conhecimento de diferentes técnicas, empregadas desde a análise do domínio do problema até sua implementação. Assim, a C-CORE foi documentada com técnicas UML, seguindo o método *Catalysis*.

6.1 Contribuições deste trabalho

A principal contribuição é a ferramenta C-CORE, que prove auxílio ao Engenheiro de Software na construção de componentes e aplicações com maior produtividade e melhor qualidade. Utiliza recursos textuais e gráficos, o que torna o desenvolvimento mais atrativo e interativo. Outras contribuições deste trabalho são:

a) Validação das idéias de desenvolvimento de componentes e aplicações, o reuso de biblioteca de componentes, e artefatos consistentes entre as fases de desenvolvimento de software baseado em componentes;
b) Apoio que a ferramenta oferece a novas pesquisas orientadas para a construção e reuso de componentes, incluindo o desenvolvimento de *frameworks* [Gamma et al., 1995], utilização de padrões de projeto [Alexander et al., 1977] e separação em aspectos [Kiczales et al., 1997];

c) Integração das ferramentas C-CORE e MVCASE na qual permite que as ferramentas cooperem entre si, minimizando os prazos e custos do desenvolvimento de software. A integração da C-CORE com a ferramenta MVCASE, também possibilita que se faça a modelagem e geração parcial do código dos componentes na MVCASE, e que se complemente e refine o projeto e a implementação dos componentes e suas aplicações na C-CORE. Nesta última também podem ser implementados os requisitos não funcionais, como os de acesso a banco de dados, interface, tratamento de exceções, e outros. Dessa forma têm-se artefatos consistentes entre as fases de desenvolvimento;
d) Biblioteca *DBCLIB* para o domínio banco de dados, modelado com técnicas de DBC e implementado com a tecnologia *JavaBeans*; e
e) Validação da idéia de construir componentes reutilizáveis para um domínio de banco de dados;

6.2 Trabalhos Futuros

Dentre as idéias para dar continuidade a este trabalho, destacam-se:

a) Ampliação das funcionalidades da ferramenta para suportar: *refactoring*, aspectos, e outros.
b) Desenvolvimento de biblioteca de componentes para outros domínios, incluindo o de multimídia, redes, e outros.
c) Suporte a edição e a geração de código para outras linguagens de programação como, por exemplo, C, C++, e outras;
d) Suporte para o controle de versões, permitindo que o gerenciamento das diferentes versões dos componentes construídos facilite o processo de manutenção e evolução das aplicações; e
e) Por fim, sugere-se a realização de novos estudos de caso visando melhorar a ferramenta, aumentando sua confiabilidade.

Referências Bibliográficas

[Alexander et al., 1977] Alexander, C., Ishikawa, S., Silverstein, M.; **A pattern language**. New York, Oxford University Press. 1977

[Boehm, 1988] Boehm, B.; **A Spiral Model for Software Development and Enhancement**. Computer. 1988

[Booch et al., 1999] Booch, G., Rumbaugh, J., Jacobson, I.; **The Unified Modeling Language – User Guide**. Addison Wesley, USA. 1999

[Borland, 2003] Borland; **JBuilder Tool**; Disponível no site: http://www.borland.com. Consultado em 28/04/2003.

[CBSE, 2003] CBSE; **Component-Based Software: Engineering Workshops Series**. Disponível no site: http://www.sei.cmu.edu/cbs. Consultado em 20/09/2003

[Coleman et al., 1994] Coleman, D., Arnold, P., Bodoff, S., Dollin, D., Gilchrist, H., Hayes, F., Jeremas, P.; **Object-Oriented Development – the Fusion Method**. Prentice Hall, New Jersey. 1994

[Costa Neto, 2001] Costa Neto, A.; **Projeto e Implementação de um Serviço de Eventos para o Desenvolvimento de Aplicações Baseadas em Componentes.** Dissertação de Mestrado – Programa de Pós-Graduação em Informática – Departamento de Sistemas e Computação – Universidade Federal da Paraíba, Campina Grande, PB – Brasil. 2001

[CVS, 2004] CVS; **CVS Home**. Disponível no site: http://www.cvshome.org. Consultado em 14/03/2004

[D'Souza & Wills, 1998] D'Souza, D.; Wills, A; **Objects, Components and Frameworks with UML – the Catalysis Approach.** Addison Wesley, USA. 1998

[Eclipse, 2003] Eclipse; **Eclipse Project**. Disponível no site: http://www.eclipse.org, Consultado em 03/12/2003.

[Fowler, 1977] Fowler, M.; **UML Destilled - Applying the Standard Object Modeling Language**. Addison Wesley, England. 1997

[Fukuda, 2000] Fukuda, A. P.; **Refinamento Automático de Sistemas Orientados a Objetos Distribuídos.** Dissertação de Mestrado – Programa de Pós-Graduação em Ciência da Computação – Departamento de Computação – UFSCar, São Carlos, São Paulo – Brasil. 2000

[Gamma, 1995] Gamma, E., Helm, R., Johnson, R., Vlissides, J.; **Design Patterns – Elements of Reusable Object-Oriented Software.** Addison-Wesley, USA. 1995

[Gane, 1990] Gane, Chris. **CASE**, **O Relatório Gane**. Livros Técnicos e Científicos Editora Ltda. 1990.

[Kiczales et al., 1997] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Videira Lopes, C., Loingtier, J.-M., Irwin, J; **Aspect-Oriented Programming (AOP)**. 11th European

Conference on Object-Oriented Programming (ECOOP), Lecture Notes in Computer Science (LNCS), Springer-Verlag, Finland, 1997.

[Larman, 1999] LARMAN, C. **Utilizando UML e Padrões**. Prentice Hall , Inc, 1999.

[Maia, 1999] Maia, F. A. A.. **Uma Ferramenta CASE para Metodologia FADO**. Dissertação de Mestrado em Informática – Universidade Federal da Paraíba, Campina Grande, Fevereiro, UFP. 1999

[MVCASE, 2003] MVCASE; **MVCASE Tool**; Disponível no site: http://www.recope.dc.ufscar.br/mvcase. Consultado em 03/12/2003.

[NetBeans, 2003] NetBeans; **NetBeans Tool**; Disponível no site: http://www.netbeans.org. Consultado em 20/09/2003.

[OMG, 2004] OMG - Object Management Group. **XML Metadata Interchange (XMI) - Version 1.2.** Disponível no site: http://www.omg.org/technology/documents/formal/xmi.htm. Consultado em 5/2/2004.

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

[QED, 1989] QED. **CASE: The Potential and the Pitfalls**. QED Information Sciences. 1989

[Raj, 2004] Raj, Gopalan; **Enterprise JavaBeans**. Disponível no site: http://www.execpc.com/~gopalan/java/ejb.html. Consultado em 20/10/2004

[Reis, 2001] Reis, C; **Caracterização de um Modelo de Processo para Projetos de Software Livre.** Monografia de Qualificação – Instituto de Ciências Matemáticas e de Computação – USP, São Carlos, SP – Brasil. 2001

[Ross, 1997] Ross, D. T.; **Structure Analysis (SA): A language for communicating Ideas**. IEEE Transaction on Software Engineering. 1997

[Rumbaugh, 1991] Rumbaugh, J.; **Object-Oriented Modeling and Design.** Prentice-Hall, Englewood Cliffs. 1991

[Sametinger, 1997] Sametinger, J.; **Software Engineering with Reusable Components**. 5th International Conference on Software Reuse, ACM/IEEE. Vitoria, Canadá. 1997

[Sanches, 2002] Sanches, I.C. ***Framework* para Ensino a Distância via Web**. Dissertação de Mestrado – Programa de Pós-Graduação em Ciência da Computação – Universidade Federal de São Carlos, São Carlos, SP – Brasil. 2002

[Sommerville, 1995] Sommerville, I. **Software Engineering. Fifth Edition**. Addison-Wesley, 1995.

[Souza Neto et al., 2004a] Souza Neto, R. M., Lucredio, D., Bossonaro, A. A., Cunha, J. R. D. D., Catarino, I. C. S., Souza, A. M., Prado, A. F.; **Component-Based Software Development Environment**. International Conference on Enterprise Information Systems 6th, ICEIS – Porto - Portugal. 2004

[Souza Neto et. al., 2004b] Souza Neto, R. M., Cunha, J. R. D. D., Bossonaro, A. A., Souza, A. M, Souza Junior, J. M., Catarino, I. C. S., Prado, A. F.; **C-CORE - Component Construction and Reuse - Ferramenta para Desenvolvimento de Software Baseado em Componentes** - 18o Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas – Brasília - DF. 2004

[Spagnoli & Becker, 2003] Spagnoli, L. A., Becker, K. "**Um Estudo sobre o Desenvolvimento Baseado em Componentes"**. PPG-CC – PUCRS – Brasil. 2003.

[Sun Microsystems, 2003] Sun Microsystems; **Java Technology**. Disponível no site: http://java.sun.com. Consultado em 14/03/2003

[Szyperski, 1998] Szyperski, C.; **Component Software: Beyond Object-Oriented Programming**. Addison-Wesley, USA. 1998

[Thomas & Nejmeh, 1992] Thomas, I., Nejmeh, B. **Definitions of tool integration for environments**, IEEE Software 9 (3). 1992

[V4All, 2003] V4All GUI Designer**; Eclipse Plugins** Disponível no site: http://www.assisiplugins.com/index_start.html. Consultado em 03/12/2003.

[Valeskey, 1999] Valeskey, T.; **Enterprise Javabeans – Developing Component-Based Distributed** Aplication. Addison-Wesley, USA. 1999

[Werner & Braga, 2000] Werner, C. M. L.; Braga, R. M. M.; **Desenvolvimento Baseado em Componentes**. XIV Simpósio Brasileiro de Engenharia de Software. João Pessoa – PB, Brasil. 2000

[XML, 2003] XML**; W3C - Extensible Markup Language (XML**) 1.0 Second Edition. Disponível no site: http://www.w3.org/XML. Consultado em 14/04/2003.