UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃOEM CIÊNCIA DA COMPUTAÇÃO

Reengenharia de Interfaces utilizando *Wrapping*

FRANK JOSÉ AFFONSO

São Carlos - SP
2003

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

A257ri

Affonso,Frank José.
Reengenharia de interfaces utilizando Wrapping / Frank José Affonso. -- São Carlos : UFSCar, 2003.
136 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2003.

1. Engenharia de software. 2. Reengenharia de interface. 3. Sistema legado. 4. Componentes de software. 5. Técnicas de empacotamento. 6. World Wide Web (sistema de recuperação da informação). I. Título.

CDD: 005.1 (20ª)

“Aprenda sempre...

Aprendi que se aprende errando;
Que crescer não significa fazer aniversário;
Que o silêncio é a melhor resposta, quando se ouve uma bobagem;
Que trabalhar não significa ganhar dinheiro;
Que sonhos estão aí para serem alcançados;
Que amigos a gente conquista mostrando o que somos;
Que os verdadeiros amigos sempre ficam com você até o fim;
Que a maldade se esconde atrás de uma bela face;
Que não se espera a felicidade chegar, mas se procura por ela;
Que quando penso saber de tudo ainda não aprendi nada;
Que a natureza é a coisa mais bela na vida;
Que amar significa se dar por inteiro;
Que um só dia pode ser mais importante que muitos anos;
Que se pode conversar com estrelas;
Que se pode confessar com a lua;
Que se pode viajar além do infinito;
Que ouvir uma palavra de carinho faz bem a saúde;
Que dar um carinho também faz...
Que sonhar é preciso;
Que se deve ser criança a vida toda;
Que nosso ser é livre;
Que o julgamento alheio não é importante;
Que o que realmente importa é a paz interior.

Não podemos viver apenas para nós mesmos. Mil fibras nos conectam com outras pessoas e por essas fibras nossas ações vão como causas e voltam pra nós como efeitos.”

Autor desconhecido

Dedico este trabalho

a meus pais Antonio e Odila, meu irmão Alex
e a minha namorada Cátia.

Agradecimentos

Agradeço a Deus por ter me guiado até o caminho do mestrado e por ter me dado saúde e forças para que eu chegasse até aqui.

Agradeço a Profa. Dra. Rosângela Aparecida Dellosso Penteado pela oportunidade de fazer um mestrado, pela orientação e conduta no desenvolver deste trabalho, pela amizade conquistada nesses anos, pela compreensão e paciência nos bons e maus momentos. Obrigado por você ter sido Rosângela. Obrigado por você ter sido orientadora, amiga, companheira, "mãe", etc. Meu sincero e eterno agradecimento.

Agradeço a Profa Dra Júnia Coutinho Anacleto Silva pela participação e contribuição para a realização deste trabalho. Pela amizade adquirida nesses anos. Obrigado por você ter sido Júnia.

Agradeço a meus pais pelo esforço feito a vida toda para que eu chegasse até aqui, pelo incentivo e força nesses anos.

Agradeço ao meu irmão pela paciência e compreensão nesses anos pelos bons e maus momentos.

Agradeço a minha namorada pela compreensão, ajuda, força, palavras de conforto, tolerância e amor nesses anos.

Agradeço a todos os professores e funcionários do PPGCC pelo convívio e dedicação nesses anos.

Agradeço a todas as amizades formadas nesses anos, não citarei nomes para não correr o risco de esquecer alguém. Agradeço também aos meus inimigos, pois me ensinaram como enxergar a vida de outra maneira, proporcionando outra fonte de crescimento.

Agradeço a todos as pessoas que me ajudaram a chegar até aqui.

Agradeço a todos que direta ou indiretamente contribuíram para a realização deste trabalho.

Meu muito obrigado.

Resumo

Com a evolução tecnológica e com a crescente utilização da Internet, empresas e instituições governamentais desejam migrar seus sistemas desenvolvidos com recursos computacionais antigos (legados) para mais modernos. No entanto, essa é uma tarefa que requer investimentos elevados, podendo o processo de reengenharia ser utilizado nesses casos. Uma forma de modificar esses sistemas é por meio da reengenharia da sua interface, através do empacotamento de sua lógica (*wrapping)*. Essa técnica preserva o ambiente nativo do sistema e suas funcionalidades, reduzindo em tempo e custo o processo de reengenharia. Para apoiar a migração de sistemas legados propõe-se um Processo de Reengenharia de Interface (PRI) que apóia a migração de sistemas legados de maneira geral, realizando o empacotamento de suas funcionalidades e acoplando uma nova interface do usuário para *Web*. A nova interface do usuário será disponibilizada para *Web*, que se comunica com os componentes identificados no sistema legado de acordo com os recursos especiais que eles oferecem. O PRI resume-se no estudo da interface e da lógica do sistema. Com isso, pode-se realizar a organização do sistema para que seja realizado seu empacotamento, que corresponde ao revestimento das funcionalidades legadas por uma camada de software, viabilizando a comunicação com a nova interface do usuário, desenvolvida segundo critérios de usabilidade. Os sistemas utilizados como estudo de casos são desenvolvidos no ambiente *Delphi* com ou sem características da orientação a objetos. Neste trabalho somente os desenvolvidos sem características da orientação a objetos são apresentados em detalhes. Com a aplicação desse processo, somente a interface passa a ser desenvolvida em um outro paradigma, mas o código legado permanece como o original, facilitando a tarefa dos mantenedores do sistema.

Palavras Chaves: Reengenharia de Interface, Técnicas de Empacotamento, Sistemas Legados, Usabilidade, Camada de Software, Componentes de Software, Web.

Abstract

With the technological evolution and the increasing utilization of the Internet, companies and governmental institutions have been looking for modern solution to replace and improve their legacy systems. These kind of solutions require high investments, being able to use the reengineering process in these cases. A maner to modify these systems is through reengineering of their interfaces, by wrapping of their logic. This technique preserves the native environment of system and its functionalities, reducing in time and cost the reengineering process. To support the migration of legacies systems in a general way, there is an Interface Reengineering Process (IRP), whose accomplish the wrapping of its functionalities and connect them with a new user Web interface. This Web interface connects itself with the identified components of the legacy system following the special resources that they offer. The IRP is summarized by the study of the interface and logic of the system. In this way, it possible to accomplish the system organization for the wrapping to be done, which is composed by the legacies functionalities covered by a new software layer. This new layer makes possible the communication with the new user interface, which has developed according to the usability criteria. The systems used as case studies were developed using Delphi environment, with or without object orientation characteristics. In this work only systems developed without object orientation characteristics are presented in details. With the use of the new process proposed here, the interface can be developed in any paradigm, while the legacy code remains as the original, simplyfing the system maintenance.

Keywords: Interface Reengineering, Wrapping Technique, Legacy Systems, Usability, Software Layer, Software Component, Web.

Sumário

CAPÍTULO 1 - INTRODUÇÃO

CAPÍTULO 2 - REVISÃO BIBLIOGRÁFICA

CAPÍTULO 3 - O PROCESSO DE REENGENHARIA DE INTERFACE

CAPÍTULO 4 - ESTUDO DE CASO

CAPÍTULO 5 - CONCLUSÕES E TRABALHOS FUTUROS

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Lista de Abreviaturas e Siglas

API - Application Programming Interface.

CORBA - Common Object Request Broker Architecture.

DII - Dynamic Invocation Interface

DLL - Dynamic Link Library

EJB - Enterprise JavaBeans

FaPRE/OO - Família de Padrões para Reengenharia Orientada a Objetos

GOMS - Goals Operators Methods and Selection rules.

IDL - Interface Definition Language.

IHC - Interação Humano-Computador.

JNI - Java Native Interface.

JSP - JavaServer Pages.

MAS - Modelo de Análise do Sistema.

MASA - Modelo de Análise do Sistema Atual.

MORPH - Model Oriented Reengineering Process for Human-Computer Interface

OMG - Object Management Group.

ORB - Object Request Broker.

UML - Unified Modeling Language.

XML - eXtensible Markup Language.

WIMP - Windows, Icon, Mouse and Pull down menu.

CAPITULO 1

INTRODUÇÃO

1.1 - CONSIDERAÇÕES INICIAIS

Atualmente é crescente o número de empresas e instituições governamentais que desejam migrar seus sistemas desenvolvidos sem recursos computacionais atualmente essenciais para a aprendizagem, utilização e memorização por parte dos usuários. Esses sistemas passaram por diversas manutenções sem que a documentação correspondente fosse elaborada. Assim, detém regras de negócio que não podem ser desprezadas pela organização. De agora em diante, tais sistemas\software serão chamados de sistemas\softwares legados.

A necessidade de migrar esses sistemas legados relaciona-se com o ambiente em que esses podem ser disponibilizados, tais como: para *Web*, para um sistema cliente-servidor ou para um sistema local. O alvo principal das empresas é a migração para *Web*, devido à quantidade elevada de usuários que se pode atingir, além de que a Internet representa um mercado potencial de compra e venda. Nesse tipo de migração, vários cuidados devem ser tomados, pois o público alvo desse sistema deixa de ser individual passando para diversos usuários trazendo por isso um conjunto próprio de requisitos de usabilidade que devem ser atendidos no projeto de suas interfaces.

Para disponibilizar sistemas legados para *Web*, existem algumas técnicas de reengenharia que atuam apenas na mudança da interface do usuário, preservando a lógica do sistema através do empacotamento (*wrapping*) de suas funcionalidades. Essas técnicas

amenizam o impacto da reengenharia quanto ao esforço, o tempo e o custo, pois o ambiente nativo do sistema e o conhecimento das pessoas com ele familiarizadas são preservados.

O empacotamento de software (*wrapping*) consiste em decompor as funcionalidades de um sistema em pacotes, que são envolvidos por uma "camada de software" formando os componentes de software do sistema. O objetivo dessa camada é permitir a comunicação entre o código legado empacotado e a nova interface do usuário.

A preocupação deste trabalho está centrada também na confecção da nova interface do usuário, pois se tratando de uma migração para *Web*, essa deve ser preparada para atender usuários leigos, médios e avançados. Para que isso ocorra, recomendam-se cuidados quanto à usabilidade tanto na transição da interface legada como no desenvolvimento da nova interface. Esses critérios auxiliam na identificação dos problemas de usabilidade presentes na interface legada e que podem ser corrigidos na confecção da nova interface, melhorando a comunicação com o usuário.

Para realização deste trabalho são considerados sistemas legados desenvolvidos no ambiente *Delphi* com ou sem características da orientação a objetos, com interfaces gráficas, mas que apresentam problemas quanto à usabilidade.

1.2 - OBJETIVO

O objetivo deste trabalho é elaborar um processo para conduzir a reengenharia de interface de sistemas legados desenvolvidos no ambiente *Delphi* com ou sem características da orientação a objetos para *Web*. Esse processo pode ser resumido no entendimento da lógica e da usabilidade da interface do usuário do sistema legado, no empacotamento (*wrapping*) de suas funcionalidades, no desenvolvimento da nova interface do usuário para *Web* e na integração desses dois últimos.

As diretrizes que atendem ao empacotamento do sistema atuam tanto nos sistemas orientados a procedimentos, organizando o código legado para ser empacotado, quanto naqueles que possuem características da orientação a objetos, que permite uma forma mais natural de empacotamento, devido à organização do código legado.

A migração da interface do usuário para a *Web* é apoiada por um conjunto de diretrizes que atuam: 1) na avaliação da interface legada por métodos de avaliação de usabilidade, 2) na migração dos componentes dessa interface para componentes *Web* e 3) na construção da nova interface apoiada pelos critérios de usabilidade. Essas diretrizes permitem o aprimoramento

da interface, oferecendo aos usuários maior conforto e segurança em sua utilização.

1.3 - MOTIVAÇÃO

Ao longo dos anos foi desenvolvido um conjunto de sistemas, sendo empregados em seu desenvolvimento os recursos disponíveis na época. No entanto, esses estão operando até hoje. Com o passar dos anos esses sistemas estão se tornando obsoletos, deixando de atender as novas necessidades das empresas ou de seus usuários. A motivação para a realização deste trabalho está centrada nos sistemas legados existentes atualmente e na necessidade de migrá-los para *Web*.

Atualmente, existe uma série de estudos que apontam um conjunto de alternativas para adequar sistemas legados para a realidade atual, a *Web*. Uma delas é realizar a reengenharia do sistema, que é composta pela engenharia reversa e pela engenharia avante. A engenharia reversa é utilizada para a reconstituição dos modelos de análise do sistema, que são utilizados em sua reconstrução na etapa de engenharia avante, em que podem-se empregar recursos atuais de desenvolvimento. Essa alternativa é considerada uma tarefa árdua, pois exige tempo e recursos financeiros elevados para sua realização.

Outra alternativa é fazer apenas a reengenharia da interface do usuário, que consiste em empacotar as funcionalidades do sistema legado e desenvolver uma nova interface do usuário que é acoplada ao sistema empacotado. Com isso, o ambiente nativo do sistema é preservado, as funcionalidades empacotadas são reutilizadas total ou parcialmente e, conseqüentemente, o tempo e os custos para realização dessa reengenharia são reduzidos.

Este trabalho apresenta um processo em que sistemas legados procedimentais sejam disponibilizados para *Web*, empacotando suas funcionalidades, preservando a linguagem de implementação original, mas fazendo a reengenharia da sua interface com o usuário. Com isso, preserva-se o ambiente do sistema, mantendo a experiência das pessoas familiarizadas com ele e preservando as regras de negócio das empresas que o utilizam.

1.4 - ORGANIZAÇÃO DO TRABALHO

O Capítulo 2, Revisão Bibliográfica, apresenta um conjunto de trabalhos que fornecem embasamento teórico necessário para a realização deste trabalho. Os assuntos abordados envolvem a Linguagem UML, a Engenharia Reversa e Reengenharia, a IHC - Interação

Humano-Computador, o Processo de Migração de Interface, os Componentes de Software, os Recursos de *Middleware*, os Recursos de Software para Implementação. Além desses assuntos, ênfase especial é dada as Técnicas e as Soluções de Padrões para o Empacotamento de Software, as quais oferecem um conjunto de diretrizes que auxiliam na preparação do código legado para que possa ser realizado seu empacotamento e na sua disponibilização para *Web*, para um sistema distribuído ou, para um sistema local.

O Capítulo 3, Processo de Reengenharia de Interface, apresenta o conjunto de diretrizes que auxiliam os engenheiros de software na migração de sistemas legados desenvolvidos no ambiente *Delphi* para *Web*. Esses sistemas foram desenvolvidos no paradigma procedimental ou com características orientadas a objetos, cujas interfaces do usuário são de baixa usabilidade.

O Capítulo 4 apresenta um estudo de caso aplicando as diretrizes do processo de reengenharia de interface. O sistema tomado como exemplo foi desenvolvido no ambiente *Delphi* de forma orientada a procedimentos.

Finalmente, no Capítulo 5, são apresentadas as conclusões deste trabalho, ressaltando as suas principais contribuições e perspectivas para trabalhos futuros.

CAPITULO 2

REVISÃO BIBLIOGRÁFICA

2.1 - CONSIDERAÇÕES INICIAIS

Neste capítulo são apresentados os assuntos da literatura especializada que contribuem para a realização deste trabalho. A linguagem UML, *Unified Modeling Language* , é composta por um conjunto de técnicas que oferecem representação tanto para extração da lógica do sistema legado quanto para o projeto de um novo sistema. A engenharia reversa e a reengenharia oferecem como contribuição, respectivamente, a recuperação da lógica através reconstituição dos modelos de análise e a reconstrução do sistema baseada nesses modelos. Os métodos de avaliação de usabilidade fornecem diretrizes que atuam na identificação de problemas de usabilidade, no desenvolvimento e na avaliação da interface como produto final. O processo MORPH, *Model Oriented Reengineering Process for Human-Computer Interface* **(MOORE, 1998),** oferece diretrizes que apóiam a migração de interfaces orientadas a caracteres para gráficas. Os recursos de *middleware* permitem a integração entre o sistema legado empacotado e a nova interface do usuário, atendendo a três categorias de sistema: locais, distribuídos e *Web*. Os recursos de implementação contribuem com o empacotamento do sistema legado através da construção de uma "camada de software", cujo objetivo é viabilizar a comunicação entre o sistema legado e a nova interface do usuário.

Os assuntos relacionados ao empacotamento de software abordados neste capítulo envolvem um conjunto de técnicas que atuam na extração da lógica dos sistemas legados e no

empacotamento desses, disponibilizando-os para diferentes categorias de sistemas, como para *Web*, para um ambiente cliente-servidor ou para um sistema local. Além dessas técnicas, são apresentados alguns padrões de empacotamento, que também fornecem soluções para integração de sistemas.

As técnicas e padrões apresentados neste capítulo, existentes na literatura especializada, são direcionados a sistemas desenvolvidos em linguagens como COBOL ou C/C++, cujas interfaces do usuário são orientadas a caracteres. No entanto, o foco deste trabalho é o empacotamento de sistemas desenvolvidos no ambiente *Delphi*, cujas interfaces do usuário são gráficas. Com isso, as técnicas fornecidas por esses trabalhos podem ser aplicadas parcialmente, pois a identificação do código legado e o empacotamento dos sistemas são realizados de formas distintas, devido à organização do código quanto à interface do usuário e a lógica do sistema.

Este capítulo apresenta na seção 2.2 os conceitos da Linguagem UML; na seção 2.3 os aspectos que envolvem a Engenharia Reversa e a Reengenharia, como conceitos, métodos e família de padrões; na seção 2.4 os conceitos da IHC (Interação Humano-Computador) e os métodos de avaliação de interface; na seção 2.5 o Processo de Migração de Interface; na seção 2.6 os conceitos de reuso componentes de software; na seção 2.7 alguns recursos de *middleware* que podem ser utilizados na implementação do trabalho; na seção 2.8 o recurso de software para a implementação do empacotamento (*wrapping*); na seção 2.9 o empacotamento de software e na seção 2.10 as considerações finais.

2.2 - A LINGUAGEM UML

A UML **(BOOCH, 2000)** é um conjunto de técnicas que apóiam a elaboração de projetos de sistemas orientados a objetos. Essas técnicas podem ser aplicadas na visualização, na especificação, na construção e na documentação de sistemas. A Figura 1 mostra a contribuição de diversos métodos na Linguagem UML.

As técnicas da UML estão organizadas em diagramas que atuam desde a representação dos requisitos até o projeto do sistema. Atualmente, existem algumas ferramentas que implementam os recursos da UML, tais como: *Rational Rose* **(RATIONAL, 2003)**, *ArgoUML* **(ARGOUML, 2003)**, *Together* **(TOGETHER, 2003)**, entre outras.

A ferramenta Rational Rose **(RATIONAL, 2003)** implementa os recursos da UML segundo quatro visões **(BOOCH, 2000)**: Visão de Caso de Uso (*Use Case View*), Visão

Lógica (*Logical View*), Visão de Componentes (*Component View*) e Visão de Implementação Física (*Deployment View*). Além disso, essa ferramenta oferece suporte aos sistemas voltados para *Web* através de representações para Páginas, *Formulários, Servlets*, *Beans*, entre outros. A Tabela 1 mostra as principais representações da *Rational Rose* para a modelagem de sistemas *Web*.

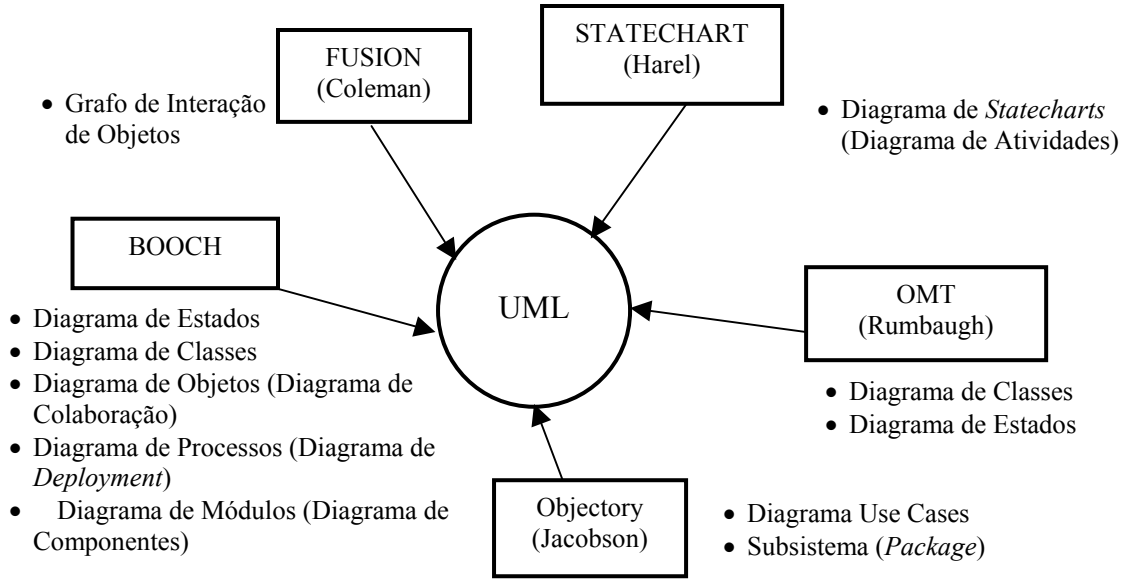
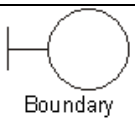


Figura 1 - A linguagem UML *apud* (PRADO, 2001)

Tabela 1 - Representações para sistemas *Web* (RATIONAL, 2003)

Representação	Nome	Descrição
	*HttpServlet*	Faz representação de um *Servlet* que é executado por uma página.
	*Entity* EJB	Faz a representação de um *Entity Bean* na implementação
	*Client Page*	Faz a representação de uma página cliente.
	*Form Page*	Faz a representação de uma página de formulário, contendo informações que são enviadas para um *Servlet* ou um JSP.
	*Boundary*	Representa a interface com outro sistema ou com alguma entidade externa

Neste trabalho, são utilizadas as técnicas para representação dos requisitos do sistema legado como diagramas de caso de uso, seqüência, classe e componentes. As representações fornecidas na Tabela 1 são utilizadas para a confecção dos diagramas de projeto, que representam a integração entre o sistema legado empacotado com a nova interface do usuário.

2.3 - ENGENHARIA REVERSA E REENGENHARIA

Até meados dos anos 80, era possível observar que apenas o hardware evoluía gradativamente e o software era tratado como atividade secundária. Nessa época, foram desenvolvidos vários sistemas de forma não organizada e não documentada, acarretando graves problemas com sua manutenção. Entretanto, alguns desses estão operando até hoje, mas considerando os recursos tecnológicos atuais, é desejável migrá-los para linguagens e bancos de dados que ofereçam mais recursos que os encontrados nesses sistemas **(CHIKOFSKY & CROSS, 1990)**.

A tarefa de migrar sistemas legados para tecnologias atuais pode ser realizada pelo processo de reengenharia, que pode ocorrer com ou sem mudança no paradigma de desenvolvimento. Esse processo é composto pela etapa de engenharia reversa, que atua na reconstituição dos modelos de análise do sistema, auxiliando na identificação dos componentes do sistema candidatos ao empacotamento e pela etapa de engenharia avante, que corresponde à implementação do sistema baseada nos modelos de análise obtidos na etapa de engenharia reversa. A condução da engenharia reversa e da reengenharia pode ser apoiada por métodos e famílias de padrões, que auxiliam na recuperação do projeto de sistema procedimentais tornando-o orientados a objetos.

Um exemplo de método para a condução da engenharia reversa e reengenharia é o Fusion/RE, que teve origem no Método Fusion **(COLEMAN et al, 1994)** *apud* **(PENTEADO, 1996)**, cujo objetivo é criar a visão orientada a objetos partindo de um sistema legado procedimental. Sua aplicação independe da documentação do sistema, necessitando apenas de seu código fonte. Esse método foi desenvolvido inicialmente com quatro passos, que são voltados para a Engenharia Reversa **(PENTEADO, 1996)** e, posteriormente, foram acrescentados mais dois passos para a segmentação do sistema **(PENTEADO et al, 1998)**. A seguir são apresentados, de forma resumida, os passos desse método:

1. **Revitalização da arquitetura do software legado:** consiste na recuperação da documentação disponível do software. O resultado é uma lista dos procedimentos

com a sua descrição e a hierarquia de chamadas (Chama/Chamado);

2. **Recuperação do Modelo de Análise do Sistema Atual (MASA):** consiste na criação de um pseudo-modelo orientado a objetos do software legado tomando como referência a base de dados e o código fonte. Esse modelo é uma abstração do modelo procedimental, sendo que as estruturas de dados, seus relacionamentos e cardinalidades são utilizados para a confecção do modelo de pseudo-classes e os procedimentos servem como referência para elaboração dos métodos;
3. **Criação do Modelo de Análise do Sistema (MAS):** corresponde ao refinamento do modelo anterior, abstraindo-se as pseudo-classes para classes e os procedimentos para métodos;
4. **Mapeamento do MAS para o MASA:** corresponde ao mapeamento das classes, dos atributos e procedimentos do MAS para os elementos correspondentes do MASA. Esse passo pode contribuir para o reuso de software;
5. **Elaboração do Projeto Avante:** consiste na elaboração dos modelos da fase de projeto para realização da Engenharia Avante, a partir dos modelos de análise elaborados no MAS;
6. **Segmentação do Sistema:** consiste na transformação dos procedimentos com anomalias em métodos nas respectivas classes do sistema, preservando a linguagem de programação original.

A aplicação do método Fusion/RE pode acarretar num aumento no número de métodos se comparado ao número de procedimentos do software legado, pois esses são transformados em partes menores aumentando a possibilidade de reuso. A segmentação fornece uma maneira simples para a transformação de uma linguagem procedimental para uma orientada a objetos.

Uma outra alternativa para a condução do processo de reengenharia de sistemas legados procedimentais para orientado a objetos é a Família de Padrões para REengenharia Orientada a Objetos FaPRE/OO **(RECCHIA, 2002)**, que tem como base os conceitos do método Fusion/RE e da linguagem de padrões de engenharia reversa no contexto orientado a objetos de Demeyer **(DEMEYER et al, 2002)**. Essa família de padrões é composta por um conjunto de padrões que encontram-se organizados em *clusters* como mostra a Figura 2 e são descritos a seguir:

- ***Cluster* 1:** Modelar os dados do legado: consiste em criar a visão orientada a objetos dos dados, tendo como ponto de partida o modelo de dados e o código fonte do sistema;

- ***Cluster* 2:** Modelar a funcionalidade do sistema: agrupa os padrões que auxiliam na recuperação da funcionalidade do sistema, criando os modelos que representam as regras de negócio, com enfoque de orientação a objetos;
- ***Cluster* 3:** Modelar sistema orientado a objetos: permite a criação da visão orientada a objetos do sistema;

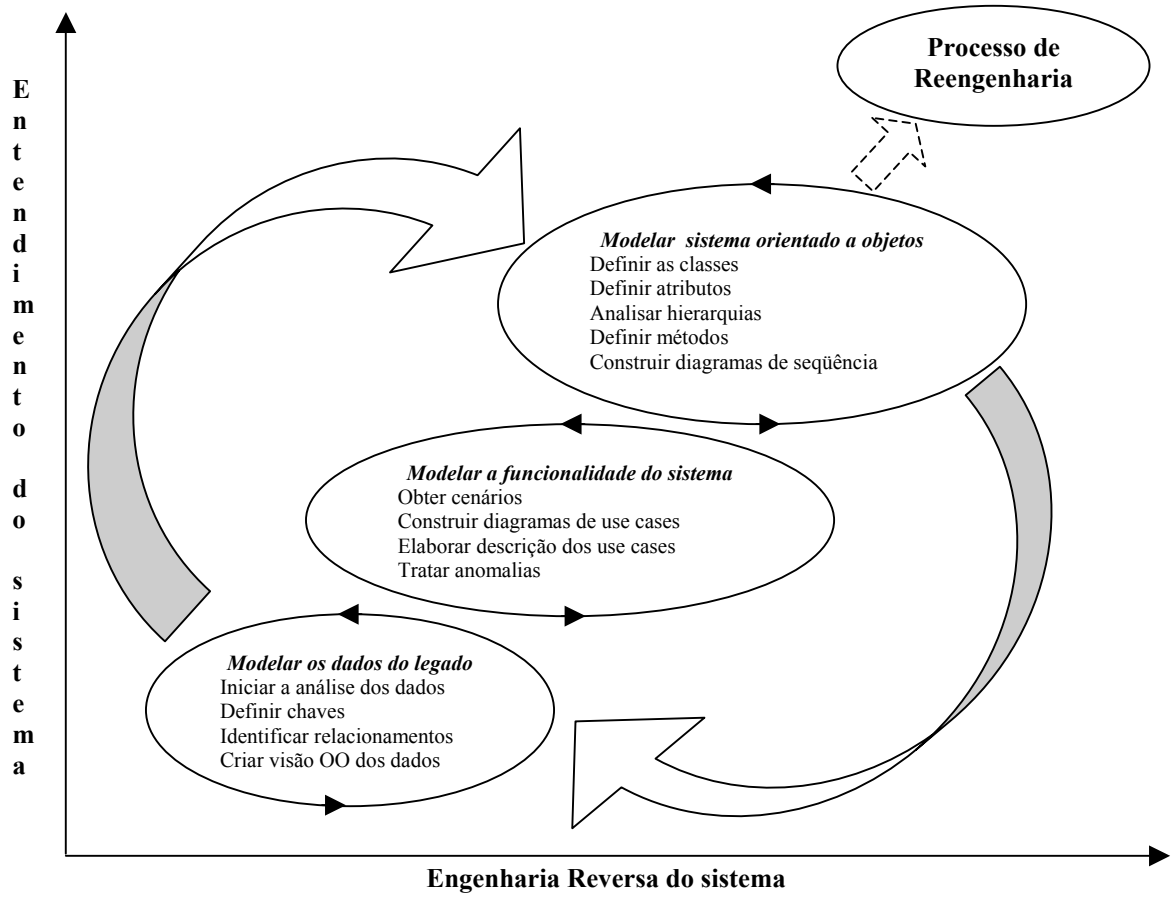


Figura 2 - Família de padrões (RECCHIA, 2002)

2.4 - A IHC - INTERAÇÃO HUMANO-COMPUTADOR

A disseminação de sistemas computacionais com interfaces acessíveis está diretamente relacionada à exploração dos recursos tecnológicos aliados às necessidades humanas. Isso deve-se ao fato de profissionais de diferentes áreas estarem contribuindo com a evolução das interfaces, podendo ser citados os psicólogos, os sociólogos, os educadores, os *designers* de software, entre outros **(BARANAUSKAS & ROCHA, 2000)**.

Segundo **BARANAUSKAS & ROCHA (2000)** "Interface é o lugar onde o contato de duas entidades ocorre e há a caracterização das propriedades físicas das partes que interagem

na comunicação". Para tratar desse contexto, tem-se uma subárea da computação denominada IHC, Interação Humano-Computador, cujo objeto de estudo é a comunicação entre o sistema computacional e o ser humano.

A IHC tem por objetivo minimizar as fronteiras entre o usuário e o sistema computacional, tornando essa comunicação mais harmoniosa. Para isso, são necessários estudos que envolvam não somente o uso de recursos, mas como adequá-los aos usuários para que se sintam mais confortáveis. Por exemplo, o uso de metáforas, que são representações dadas às funcionalidades de um software, como a figura de uma tesoura para mostrar a função recortar. No entanto, essas metáforas devem ser sugestivas, pois quando mal empregadas, além de causar incertezas, podem gerar erros por parte do usuário **(BARANAUSKAS & ROCHA, 2000)**.

Pode-se dizer que a principal inovação nos sistemas computacionais veio juntamente com os sistemas manipulados através de janelas, interfaces gráficas, chamadas atualmente de interfaces WIMP, *Windows, Icons, Mouse, Pull-down menu*. Essa inovação impulsionou estudos quanto à utilização das interfaces gráficas, pois apesar da interação com o usuário ser mais satisfatória, quando mal projetadas, podem causar graves problemas com relação à usabilidade do sistema. Esses problemas podem ser reduzidos com a aplicação de métodos, que atuam tanto no processo de desenvolvimento quanto no de avaliação do produto final. Dentre os métodos encontrados na literatura e estudados, interessou-se pela avaliação heurística **(NIELSEN, 1994)** e pelo método GOMS **(JOHN et al, 1994) (JOHN et al, 1994a) (JOHN et al, 1996b) (KIERAS, 1999)**, pois são de fácil aplicação e não exigem investimentos com recursos computacionais adicionais como hardware ou software.

A Avaliação Heurística é um método utilizado para encontrar problemas de uma interface. Basicamente, envolve um conjunto de avaliadores que examinam e julgam uma interface segundo os critérios de usabilidade, dez heurísticas de **NIELSEN (1994)**, as quais são descritas a seguir:

1. **Diálogo Simples e Natural:** a interface deve conter expressões e conceitos conhecidos pelo usuário, evitando o mau entendimento por parte dele e auxiliando na execução das tarefas. Essa heurística faz tratamento para um bom projeto gráfico e de cores, agrupando as informações para o usuário e sugerindo a limitação de cores em torno de cinco a sete cores;
2. **Falar a linguagem do usuário:** para execução de qualquer tarefa devem existir comandos claros e visíveis, evitando empregar palavras que não façam sentido para o usuário como termos técnicos. Deve-se projetar um caminho lógico para

execução das tarefas, facilitando a memorização. Finalmente, deve-se ter cuidado ao empregar metáforas em interfaces, pois essas devem representar adequadamente uma funcionalidade;

3. **Minimizar a carga de memória do usuário:** o usuário deve aprender o sistema para poder utilizá-lo. Ao ficar um determinado tempo sem interagir com ele, deve ser capaz de lembrar os comandos e sua disposição na interface, sem que haja necessidade de consultar manuais do sistema;
4. **Consistência:** a interface deve apresentar as informações de modo que o usuário possa aprender, disponibilizando atalhos para execução das tarefas como menus *pop-up*, teclas de atalhos, entre outros. A interface deve ser preparada para atender usuários leigos a experientes;
5. ***Feedback:*** o sistema deve fornecer ao usuário informações sobre a execução das tarefas, além de um tempo de resposta satisfatoriamente rápido para sua execução, fazendo com que o usuário não se sinta perdido achando que fez algo errado. Além disso, deve ser tolerante a falhas, informando através de mensagens quando o sistema pára de funcionar e o que ocasionou isso;
6. **Saídas (respostas) marcadas com clareza:** deve ser dada ao usuário a opção de direcionamento das tarefas através de botões para cancelar, voltar, ou navegar, permitindo a transição por outras partes do sistema;
7. **Atalhos:** como o projeto de interface deve atender os mais variados tipos de usuários, deve-se então oferecer teclas de atalho, menus suspensos, comandos que agilizam a execução das tarefas em relação ao modo normal de execução. Nos sistemas voltados para *Web*, a sugestão é o uso de *hyperlinks*, que permitem acesso rápido à execução das tarefas;
8. **Boas mensagens de erros:** essa tarefa é de extrema importância e muito difícil de ser erradicada. Deve-se projetar mensagens de erros que relatem o que o usuário fez de errado e como corrigir o erro. Isso deve ser feito de forma que ele não se sinta agredido ou inferiorizado, fazendo com que aprenda com o erro;
9. **Prevenção a erros:** uma interface deve guiar o usuário na execução das tarefas, evitando a entrada de informação indesejada. Dessa forma, evita que o sistema responda de forma inadequada aos dados de entrada;
10. **Ajuda e documentação:** deve apoiar o uso de interface, auxiliando na execução das tarefas, fornecendo de forma clara e objetiva as informações sobre estado do sistema;

Essas heurísticas são a base para o processo de desenvolvimento e avaliação de interfaces. No entanto, existem alguns questionamentos quanto à aplicabilidade da Avaliação Heurística no estudo da interface do usuário, devido a falta de uma comprovação empírica, sendo direcionada apenas pelo sentido, pela intuição e pela experiência de cada um. Por esses motivos, esse método produz melhores resultados quando aplicado por um grupo de avaliadores e não de forma individual, devido à diversidade e experiência de cada um. O resultado da aplicação desse método pode ser representado em relatórios técnicos que relatam os problemas de usabilidade e as sugestões para correção **(NIELSEN, 1994)**. Neste trabalho, essas heurísticas são utilizadas sem a aplicação formal do método Avaliação Heurística.

Outro método de avaliação de usabilidade é o GOMS (*Goals, Operators, Methods and Selection rules*), que fornece representação do conhecimento para execução das tarefas em função das Metas (G-*Goals*), dos Operadores (O-*Operators*), dos Métodos (M-*Methods*) e das Regras de Seleção (S-*Selection rules*) **(JOHN et al, 1994a) (JOHN et al, 1994b) (JOHN et al, 1996a) (KIERAS, 1999)**.

As metas representam o quê o usuário pretende realizar no sistema, sendo que essas podem ser subdivididas em submetas, formando uma hierarquia. Os operadores são as ações executadas a serviço de uma meta, podendo ser classificados como perceptual, cognitivo, motor, ou uma combinação desses. Os métodos são formados por seqüências de operadores e metas ou submetas. Caso exista uma hierarquia de metas deve haver uma equivalente para os métodos. Finalmente, as regras de seleção fornecem diretrizes para escolha do melhor método para execução de uma meta.

O método GOMS é composto por um conjunto de variações: *Keystroke-Level Model* (KLM), Card, Moran e Newell (CMN-GOMS), *Natural* GOMS *Language* (NGOMSL) e a *Cognitive-Perceptual-Motor* GOMS (CPM-GOMS,). Cada uma dessas variações possui limitações e níveis de abrangência.

A variação KLM é a abordagem mais simples, pois para estimar o tempo de execução para uma tarefa, apresenta-se uma lista de seis operadores com tempos pré-definidos, sendo totalizados no final da execução. São eles:

- **K :** para pressionar uma tecla ou um botão;
- **P :** para apontar para o destino ou em uma tela;
- **H :** para manipular o teclado ou outro instrumento;
- **D :** para desenhar uma linha ou segmento de reta;
- **M :** para preparação mental para execução de uma ação;
- **R :** para representar o tempo de resposta que o usuário espera do sistema;

A variação CMN-GOMS é baseada na hierarquia de metas. Os métodos são organizados de maneira informal como um programa de computador, podendo ser incluídos sub-métodos e condições. Dessa forma, é possível prever a seqüência de operadores e o tempo de execução das tarefas.

A variação NGOMSL está estruturada segundo a notação da linguagem natural para construção do modelo de interação, que é representado na forma de um programa de computador, fornecendo a seqüência de operadores para execução das tarefas. A execução desse modelo é da forma *top-down*, em que se parte do nível mais alto, que são as metas, passando pelos métodos, até se chegar naqueles métodos que são compostos por operadores mais primitivos, K - *Keystroke*. A Figura 3 mostra o modelo NGOML para a meta mover texto **(JOHN et al, 1996a)**.

```
NGOMSL Statements Executions External                    Operator Times
     Method for goal: Move text                                      1
      Step 1. Accomplish goal: Cut text.                             1
      Step 2. Accomplish goal: Paste text.                           1
      Step 3. Return with goal accomplished.                         1

     Method for goal: Cut text                                       1
      Step 1. Accomplish goal: Highlight text.                       1
      Step 2. Retain that the command is CUT, and
      accomplish goal: Issue a command.                              1
      Step 3. Return with goal accomplished.                         1

     Method for goal: Paste text                                     1
      Step 1. Accomplish goal: Position cursor at
      insertion point.                                               1
      Step 2. Retain that the command is PASTE,
      and accomplish goal: Issue a command.                          1
      Step 3. Return with goal accomplished.                         1
```

Figura 3 - Modelagem NGOMSL para mover texto, (JOHN et al, 1996a)

A variação CPM-GOMS também é baseada no tempo de execução e na análise dos componentes de atividades, que classifica os operadores como perceptual, cognitivo e motor. A grande influência nesse modelo é a percepção visual, que permite a formalização do modelo mental das tarefas, fazendo a percepção cognitiva e motora, para posteriormente, fazer a execução das mesmas. A percepção humana das tarefas está diretamente relacionada ao conhecimento do usuário sobre as tarefas.

A aplicação do Método GOMS, em qualquer uma das variações, permite que seja obtido um conjunto de informações sobre a funcionalidade do sistema, o tempo de execução das tarefas, seu aprendizado e, o suporte para prevenção de erros.

Neste trabalho foi utilizada a variação NGOMSL para extração da usabilidade da

interface legada, compondo o modelo de interação GOMS. Esse modelo fornece o caminho a ser percorrido na interface para execução das tarefas. A aplicação desse método permite que sejam obtidos documentos que relatam os erros de funcionalidade e de usabilidade da interface do sistema. Esses documentos são obtidos através da observação do sistema em funcionamento, em que o engenheiro de software deve testar todas suas funcionalidades.

2.5 - PROCESSO DE MIGRAÇÃO DE INTERFACE

Em **MOORE (1998)** é apresentado o processo de migração de sistemas legados com interfaces orientadas a caracteres para sistemas com interfaces gráficas, por proporcionarem interação mais "amigável" com o usuário que as anteriores. Existem algumas dificuldades nessa migração, pois há preocupação em manter a funcionalidade original, para que o usuário não tenha que aprender tudo novamente. Além disso, existem os fatores tempo e custo, que podem ser reduzidos apenas com a migração da interface do usuário.

Devido aos fatores mencionados anteriormente, pesquisadores e profissionais estão interessados, cada vez mais, em automatizar o processo de reengenharia de sistemas. Nesse sentido, tem-se o processo MORPH, *Model Oriented Reengineering Process for Human-Computer Interface*, que é composto por três etapas, Detecção, Transformação e Geração, como mostra a Figura 4, sendo as duas primeiras automatizadas e a última manual.

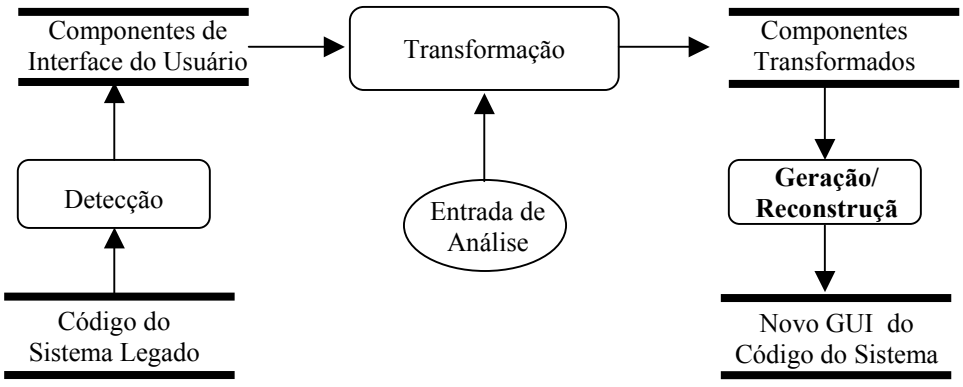


Figura 4 - A estrutura do Processo MORPH (MOORE, 1998)

A etapa de detecção corresponde à identificação dos componentes de interface do sistema legado, que são encaminhados para a etapa de transformação para serem transformados em componentes gráficos (*widgets*). O processo de geração corresponde à inserção dos componentes gráficos no mesmo ponto em que foram detectados.

O processo MORPH também pode ser aplicado na migração de sistemas legados para a *Web*, mapeando os componentes da interface legada para componentes que são utilizados na construção de interfaces voltadas para *Web* **(MOORE & MOSHKINA, 2000)**.

Diante desses dois tipos de migração, local e *Web*, pode-se notar que o processo de migração MOPRH enfatiza a mudança da interface do usuário mas preservando o ambiente nativo do sistema. Esse tipo de mudança pode ser caracterizado como reengenharia de interface do usuário.

Para realização deste trabalho, algumas das diretrizes do processo MORPH são reutilizadas e outras adaptadas, pois a reengenharia da interface do usuário realizada neste trabalho está voltada para sistemas desenvolvidos no ambiente *Delphi*, cujas interfaces do usuário já são gráficas.

2.6 - REUSO E COMPONENTES DE SOFTWARE

O desenvolvimento de software é, ao longo do tempo, uma tarefa árdua, em que os engenheiros de software estão interessados no reuso. Sua caracterização não está presente somente no desenvolvimento de um novo sistema, mas também num processo de reengenharia. Ao iniciar o processo de engenharia reversa para obter a lógica do sistema podem ser identificadas funcionalidades redundantes, que são isoladas em blocos de código com interfaces de acesso bem definidas, componentes de software. Em seguida, esses componentes são reutilizados no processo de Engenharia Avante para montagem do sistema. No entanto, um dos grandes desafios do reuso de software é produzir componentes que possam ser reutilizados em vários domínios **(PRIETO & ARANGO, 1991)** apud **(WERNER & BRAGA, 2000)**.

Para **WERNER & BRAGA (2000)** o Desenvolvimento de Software Baseado em Componentes (DSBC) tem como objetivo reduzir o problema em partes menores, diminuindo a complexidade da funcionalidade. Atualmente, existe um conjunto de métodos que apóiam o DSBC, dentre eles, destaca-se o de **JACOBSON et al (1997)**, que se baseia no paradigma da orientação a objetos e em técnicas da Linguagem UML e utiliza "Casos de Uso" como ponto de partida para identificação dos componentes reutilizáveis. Outro método que se destaca é o *Catalysis* **(D'SOUZA & WILL, 1998)**, que une os recursos da orientação a objetos e da computação distribuída.

Segundo **AOYAMA (1998)**, o DSBC é um novo paradigma no desenvolvimento de

software, cuja meta é compor componentes de software "*plug & play*" em um *framework*, que corresponde a conjunto de componentes que podem ser utilizados em vários sistemas, mudando radicalmente a forma de desenvolvimento. Essa forma assemelha-se ao desenvolvimento do hardware e oferece vários benefícios às empresas, que passariam ter um processo de desenvolvimento organizado e, conseqüentemente, com maior velocidade e aumento na produtividade.

Os componentes de software podem ser vistos como parte das linguagens de programação. Eles trabalham com um alto nível de abstração, baseado em suas interfaces de comunicação (*plug-in)*, o que facilita o desenvolvimento de software. A Figura 5 apresenta a arquitetura de componentes, expondo uma situação de reuso, em que diferentes aplicações, (1 e 3) ou (1 e 2), fazem uso dos mesmos componentes, caracterizando um aumento em produtividade **(JACOBSON et al, 1997, 1998)**.

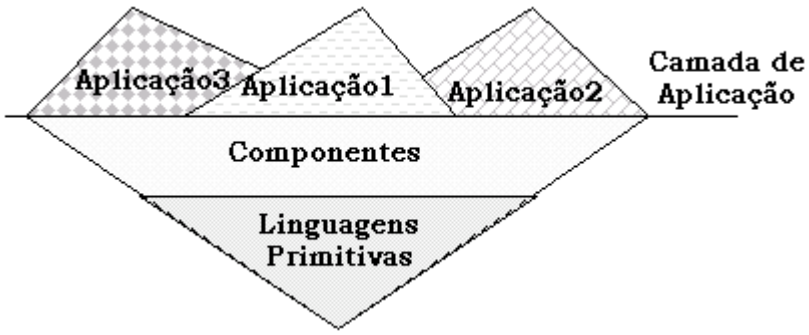


Figura 5 - Estrutura de Componentes (JACOBSON et al, 1997)

O grande interesse do desenvolvimento de componentes de software é o reuso, para isso é necessário que eles sejam independentes da aplicação para a qual foram projetados e que tenham um grau de qualidade elevado. Para que isso ocorra, um componente deve ter interfaces bem projetadas, que facilitam sua integração; deve ser bem testado, que garante sua eficiência; e deve ser bem documentado, que garante sua utilização **(JACOBSON et al, 1997) (PREE, 1997)**.

WERNER & BRAGA (2000) comentam que o projeto de componentes não é uma tarefa trivial do ponto de vista de projeto, implementação e validação. No entanto, citam dois tipos de componentes de software:

- **Componente caixa-branca:** que é voltado para o reuso, sendo necessário conhecer o domínio da aplicação para desenvolvê-lo. Através de um componente caixa-branca pode-se originar um componente caixa-preta, em um nível de

abstração mais baixo, pois algumas adaptações de contexto foram efetuadas no componente caixa-branca para gerá-lo;

- **Componente caixa-preta:** esse componente é mais fácil de ser utilizado do que o de caixa-branca. Esses componentes são unidades bem testadas que fornecem especificações de entradas e de saídas bem definidas em suas interfaces (*plug-in*) e que não necessitem de qualquer modificação/adaptação para utilizá-lo.

Para que haja componentes bons e reutilizáveis, Johnson e Foote apud **(JACOBSON et al, 1997)** sugerem alguns critérios para o projeto de componentes, tais como:

- Reduzir o número de parâmetros, pois poucos argumentos deixam as operações mais coesas, implicando em operações mais primitivas;
- Proibir o acesso direto às instâncias das variáveis, para isso, operações especiais devem ser usadas, como forma de analisar os dados que são passados para os componentes, liberando a implementação da interface;
- Cuidar para que seus nomes sejam consistentes, as operações, em diferentes componentes, devem possuir nomes similares se executam a mesma tarefa.

Um sistema de componentes é composto por uma biblioteca, que é também conhecida como repositório. Esse sistema envolve a classificação, a recuperação e a inserção dos componentes nessa biblioteca. Seu objetivo é facilitar o acesso e o reuso de componentes no desenvolvimento de sistemas **(JACOBSON et al, 1997) (PRESSMAN, 2000)**.

Os conceitos e métodos de componentes apresentados nessa seção servem como base para a realização deste trabalho, auxiliando na eliminação de redundâncias do código legado, na redução de complexidade das funcionalidades detectadas, na identificação e no projeto de componentes de software, fazendo com que os componentes identificados possam ser reutilizados na reconstrução do novo sistema.

2.7 - RECURSOS DE *MIDDLEWARE*

Atualmente, a Internet oferece muitos protocolos de comunicação, sendo o HTTP, *Hypertext Transfer Protocol*, sua base, que utiliza URLs, *Uniform Resource Locators* ou *Universal Resource Locators*, para a localização dos dados na WWW, *World Wide Web*, além da integração entre o cliente e o servidor **(DEITEL & DEITEL, 2001)**.

Os *servlets* oferecem uma maneira fácil para integração entre cliente e servidor na WWW, atuando do lado servidor, deixando o cliente livre de qualquer processamento, caracterizando uma aplicação cliente-magro (*thin-client*) **(AYERS et al, 1999)**. Esse recurso é uma evolução do CGI, *Common Gateway Interface*, quanto ao desempenho, pois a cada requisição do cliente uma nova sessão é aberta, existindo apenas um *servlet* para atender todas as requisições **(SUN, 2003)**.

O ciclo de vida de um *servlet* tem inicio com o seu carregamento na memória pelo servidor, permanecendo nela para execução das requisições do cliente e descarregado pelo servidor, quando necessário.

As características gerais de um *servlet* são: persistência, segurança, independência de plataforma, acesso ao banco de dados através do JDBC, *Java Database Connectivity*, que possibilita o gerenciamento de várias requisições de usuários por meio de sessões, permite maior interação com o cliente através da inserção e obtenção de *cookies*.

Apesar das características apresentadas, existem algumas críticas a respeito do uso de *servlets,* devido à mistura de código entre as linguagens Java e HTML, *Hypertext Markup Language*, deixando a programação desorganizada. No entanto, existe uma forma mais sofisticada de programação, que separa as páginas de respostas dos *servlets* de sua lógica. A Figura 6 mostra e execução do *servlet* "`Servlet.java`", que tem como retorno a requisição do formulário "`Formulario.html`" a página "`PaginaResposta.html`". Essa página não se encontra inserida no código do *servlet*, mas no pacote "`PaginaResposta`" por ele importado.

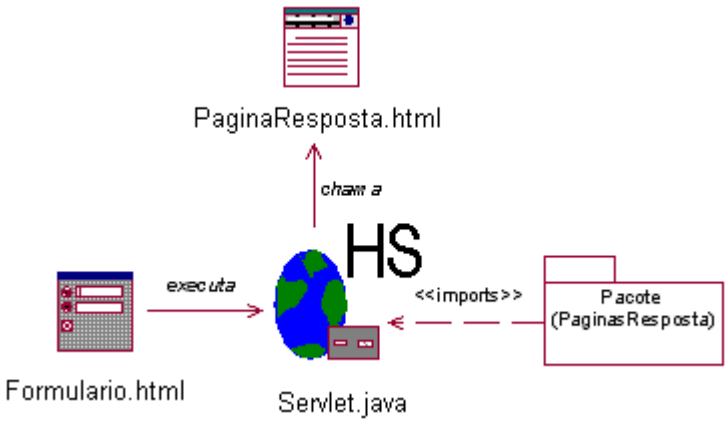


Figura 6 - *Servlet* segmentado

Outro recurso de *middleware* para Internet é JSP, *JavaServer Pages*, que é baseado na tecnologia de *Servlet*, compartilhando os recursos da linguagem de programação Java. É uma excelente referência para desenvolvimento de aplicações para *Web* devido à velocidade de

desenvolvimento em relação aos *Servlets*, pois é compilada pelo servidor em tempo de execução **(AYERS et al, 1999)**.

O ciclo de vida de JSP pode ser resumido da seguinte forma: ao receber uma requisição do cliente, o servidor compila automaticamente o arquivo JSP através da "JSP *Engine*", gerando um *servlet* residente na memória, que é encarregado de atender as requisições do cliente. O servidor descarrega esse *servlet* da memória quando necessário **(AYERS et al, 1999)**. O arquivo JSP é re-compilado automaticamente pelo servidor quando houver alguma alteração no seu código fonte.

Os *Servlets* e JSP são recursos direcionados ao desenvolvimento de sistema para *Web*, no entanto, por fazer parte da linguagem Java, compartilham recursos disponíveis para implementação, como componentes EJB, *Enterprise JavaBeans*, comunicação com dados via JDBC, *Java DataBase Connection*, entre outros.

Como recurso utilizado para distribuição tem-se a Especificação CORBA, *Common Object Request Broker Architecture*, **(OMG, 2002)** que é um padrão para objetos em sistemas distribuídos, gerenciado por um grupo de desenvolvedores conhecidos como OMG, *Object Management Group*. Esse grupo tem por objetivo estabelecer regras e padrões que possibilitem o desenvolvimento de aplicações distribuídas com a reusabilidade e interoperabilidade de objetos **(SIEGEL, 1996, 2000)**.

A especificação CORBA provê que os objetos enviem pedidos, *requests*, ou recebam respostas para/de outro objeto num sistema distribuído, de forma transparente como definido no ORB, *Object Request Broker*, do OMG. Para que isso ocorra, fica a cargo do ORB fornecer uma estrutura que possibilite os objetos se comunicarem independentemente do local da rede onde esteja o objeto requisitado, do protocolo da rede, da linguagem de programação que foi utilizada para implementação do objeto, da arquitetura do hardware e do Sistema Operacional **(OMG, 2002)**. A comunicação procede de forma que o objeto cliente seja atendido em seus pedidos sem saber quais objetos o serviram, ficando o ORB encarregado de localizar o objeto para execução do pedido, mediante sua interface de comunicação. Esse é o princípio básico para a comunicação do ORB para com os objetos, garantindo a portabilidade, a reusabilidade e a interoperabilidade destes em um ambiente heterogêneo **(OMG, 2002) (SIEGEL, 1996, 2000)**.

A Figura 7 mostra todos os componentes da arquitetura CORBA, cujas interfaces ORB são transparentes para os clientes. Em uma invocação de operação de forma dinâmica, usa-se a *Dynamic Invocation Interface* (DII). Caso se queira fazer uma invocação não dinâmica usa-se a *Interface Definition Language* (IDL) *Stubs*, que trabalha com dados

específicos dos objetos em tempo de compilação. Para cada tipo de objeto haverá uma interface a ser visualizada, dependendo dos requisitos passados no pedido **(OMG, 2002) (SIEGEL, 1996, 2000)**.

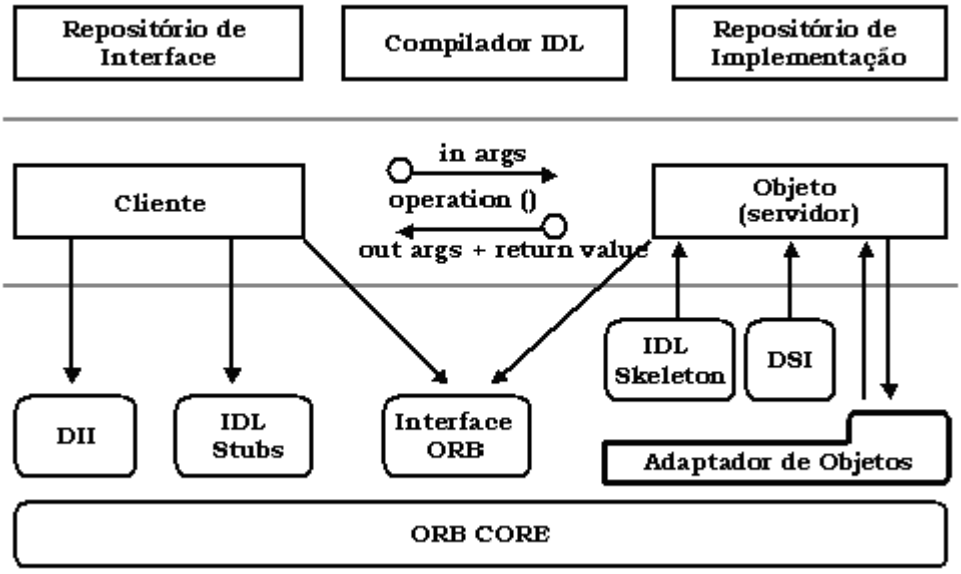


Figura 7 - A Estrutura das Interfaces do ORB (OMG,2002)

Na especificação CORBA um sistema distribuído pode ser posto em funcionamento com mais de um ORB, com isso o cliente terá referências a mais de um objeto e, portanto, cabe ao ORB distingui-las e encaminhá-las adequadamente para que o serviço requisitado seja atendido.

Os recursos apresentados nesta seção são utilizados, neste trabalho, para integrar o sistema legado empacotado e a nova interface do usuário. Esses recursos atuam como uma camada de *middleware*, recebendo e enviando dados entre eles.

2.8 - RECURSOS DE SOFTWARE PARA IMPLEMENTAÇÃO DO EMPACOTAMENTO

A implementação do empacotamento (*wrapping*) está centrada na construção de uma "camada de software" nos componentes de software identificados no sistema legado em *Delphi*. Essa camada tem por objetivo viabilizar a comunicação entre o sistema legado empacotado e a nova interface do usuário, sendo essa comunicação realizada através de um recurso de *middleware* desenvolvido na linguagem Java.

Um dos recursos estudados para implementação do empacotamento é o JNI, *Java Native Interface* **(SUN, 2003)**, que permite o empacotamento de sistemas nas linguagens C/C++, COBOL, *Object Pascal*, entre outras. Esse recurso disponibiliza o código empacotado para a linguagem Java, permitindo a utilização de recursos de *middleware* como *Servlets*, JSP e algumas implementações CORBA como a ORBacus **(IONA, 2003)**. A Figura 8 apresenta a arquitetura JNI.

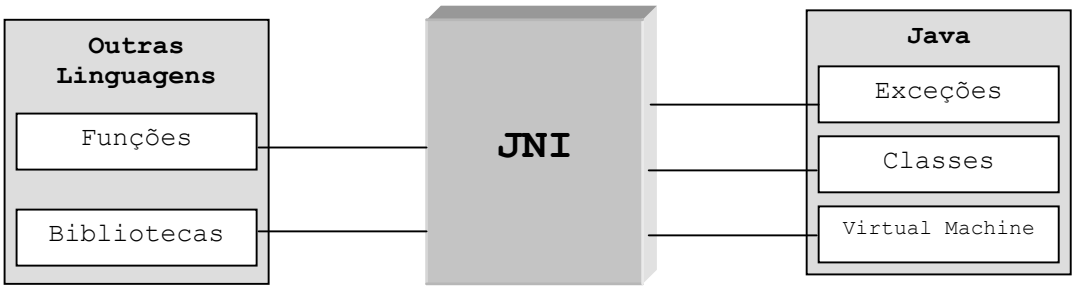


Figura 8 - Arquitetura JNI, (Sun, 2003)

A arquitetura exposta na Figura 8 mostra a integração da linguagem Java com outras linguagens como C/C++, COBOL e *Delphi*, sendo necessária uma "camada JNI" para interpretar e converter os tipos de dados entre as linguagens. A necessidade de conversão é justificada pela divergência dos tipos de dados, que podem ser de tamanho e de tratamento diferenciado.

A Figura 9 mostra a estrutura física do empacotamento de uma aplicação legada desenvolvida em *Delphi*. Basicamente, todo empacotamento é composto pelos arquivos "`Jni.pas`" e "`Jni_md.inc`", que contêm as funções conversoras de tipos de dados entre Java e *Delphi*. A nova aplicação está representada pelo arquivo "`Main.java`", que tem acesso à aplicação legada através do arquivo "`HelloWorld.java`". Esse último contém as chamadas das funcionalidades que estão presentes no arquivo "`HelloWorld.dpr`", formando um "canal de comunicação" entre o sistema legado e a nova aplicação. O sistema legado está representado pelo arquivo "`Legado.pas`".

O ambiente *Delphi* possui elevada quantidade de recursos, como a integração com o JNI, que viabiliza o empacotamento de sistemas desenvolvidos nesse ambiente. A seguir são abordados os aspectos mais relevantes do ambiente de desenvolvimento *Delphi*.

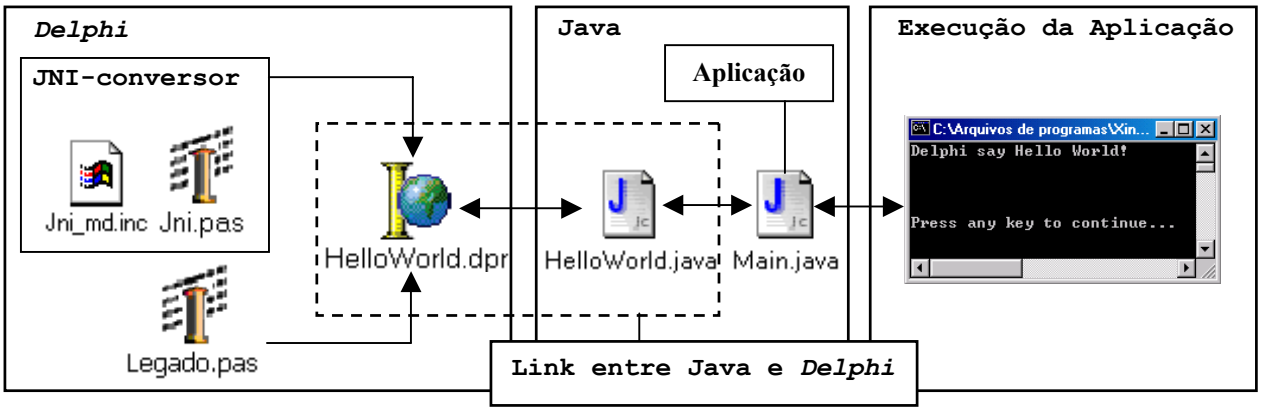


Figura 9 - Estrutura física do empacotamento

Uma das principais características desse ambiente de desenvolvimento é a organização. Basicamente, um sistema *Delphi* é composto por um arquivo "`.dpr`", que representa o projeto do sistema, por arquivos "`.pas`", que contém a programação da funcionalidade, e por arquivos "`.dfm`", que contém as especificações dos componentes utilizados para a construção do sistema. A Tabela 2 apresenta a extensão dos principais arquivos que compõem um sistema, suas definições e suas funcionalidades.

O ambiente *Delphi* possibilita o desenvolvimento de sistemas segundo três abordagens **(LEMOS, 2002) (BORLAND, 2003)**:

- **Visual:** permite a construção de sistemas de maneira rápida e fácil com o auxilio da paleta de componentes.
- **Orientada a Componentes:** pode ser aplicada sob dois pontos de vista: a)desenvolvimento com componentes, utilizando os componentes disponíveis no *Delphi*; b)desenvolvimento de componentes, criando e reutilizando os próprios componentes;
- **Orientada a Objetos:** consiste no agrupamento dos atributos e funcionalidades em classes, oferecendo melhor organização ao desenvolvimento.

Outro recurso importante do *Delphi* é a DLL, *Dynamic Link Library*, que são componentes portáveis a qualquer aplicação Windows. Uma DLL pode conter código, dados, ou qualquer recurso que se deseja compartilhar, sendo esses disponibilizados através de suas interfaces. Sua criação pode ser justificada pelas seguintes razões **(TEIXEIRA & PACHECO, 2001)**:

- **Código Comum:** quando são encontrados trechos de código duplicados em vários pontos do sistema (reuso de código).

- **Atualização Transparente:** independente do restante do sistema;
- **Exportando funcionalidade**: pode compartilhar procedimentos, funções e métodos de objetos, apesar de não exportarem classes e objetos;
- **Estrutura Modular:** podem ser acessadas por uma aplicação e não alteradas por ela, são compiladas e para serem utilizadas devem ser carregadas na memória;
- **Reuso:** corresponde ao compartilhamento das DLLs para outras aplicações escritas em linguagens como *Visual Basic*, *Visual* C++, *Delphi*, entre outros;

Tabela 2 - Arquivos de uma aplicação e seu significado (Adaptado de (Cantú, 2001))

Extensão Arquivo	Definição	Função
".EXE"	Arquivo compilado executável (Gerado pela compilação)	Este é um arquivo executável da aplicação. Incorporando todos os arquivos ".DCU" gerados quando sua aplicação é compilada.
".DPR"	Arquivo do Projeto	Código fonte em Pascal do arquivo principal do projeto. Lista todos os formulários e *units* no projeto, e contém código de inicialização da aplicação.
".DCU"	Arquivo da *Unit* compilada (Gerado pela compilação)	A compilação cria um arquivo ".DCU" para cada ".PAS" no projeto, utilizado para a execução da aplicação caso o código fonte não esteja presente para ser compilado.
".PAS"	Código fonte da *Unit* (*Object Pascal*)	Um arquivo ".PAS" é gerado por cada formulário que o projeto contém. O projeto pode conter um ou mais arquivos ".PAS" associados com algum formulário ou não.
".DFM"	Arquivo gráfico do formulário	Arquivo binário que contém as propriedades do desenho de um formulário contido em um projeto e ainda, as propriedades dos componentes de banco de dados como instruções SQL, localização do banco, etc.
".RES"	Arquivo de Recursos do Compilador	Arquivo binário que contém o ícone, mensagens da aplicação e outros recursos usados pelo projeto.

Os recursos apresentados nessa seção são de grande relevância para a realização deste trabalho, sendo o primeiro responsável pela integração do sistema legado com a nova interface do usuário e o segundo oferece suporte para essa integração, além de oferecer um conjunto de recursos que apóiam o entendimento e a extração das funcionalidades do sistema legado para o empacotamento.

2.9 - EMPACOTAMENTO DE SOFTWARE

Os trabalhos relacionados ao empacotamento de software são apresentados nesta seção, sendo abordado a forma de extração da lógica de negócio de sistemas, as técnicas de empacotamento e os padrões de empacotamento.

Segundo **SNEED (2001b)** a extração da lógica de negócio do código legado pode ser realizada através de um processo composto por quatro etapas, representadas na Figura 10 pelos retângulos. Esse processo encontra-se automatizado na primeira e na terceira etapa, sendo as demais realizadas de forma manual.

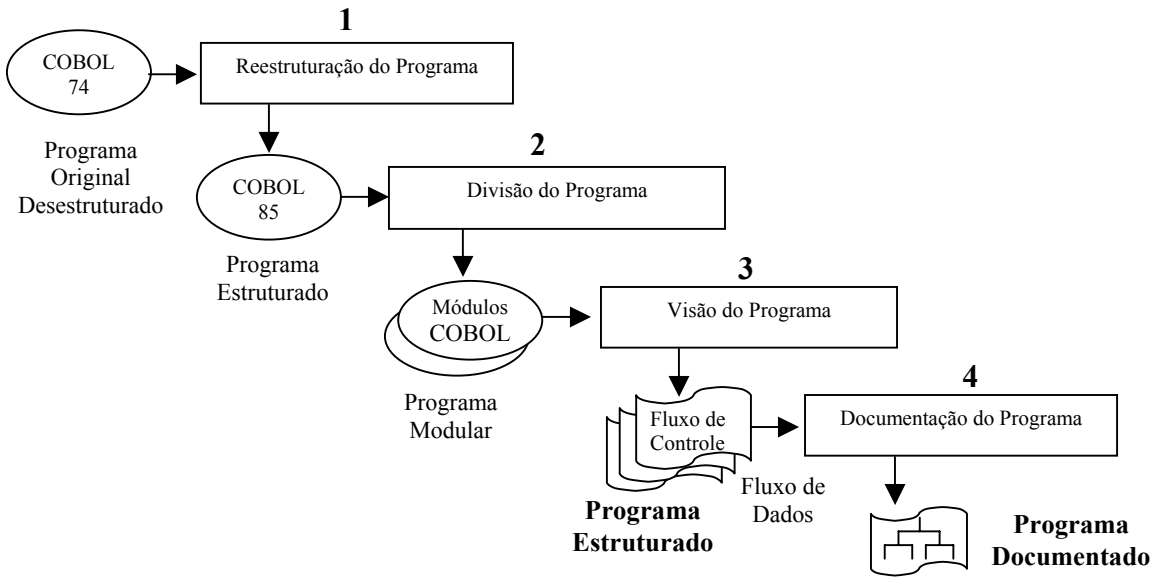


Figura 10 - Etapas para Extração da Lógica de Negócio (SNEED, 2001b)

Na Reestruturação do Programa, a ferramenta **"SoftRecom"** *apud* **(SNEED, 2001a)** prepara o código legado, reorganizando-o e produzindo o código estruturado do programa.

A Divisão do Programa faz uso da inteligência humana para identificação de pontos lógicos de entrada como funções, entrada de procedimentos e rótulos de entrada eliminando a redundância no código.

A Visão do Programa é apoiada pela ferramenta **"COBAnal"** *apud* **(SNEED, 2001a)**, cujo objetivo é documentar o código fonte gerando grafos de fluxo de controle.

A Documentação do Programa fornece o programa documentado que será armazenado em uma base de dados com todos documentos obtidos nas etapas anteriores como grafos, módulos e documentação do programa.

O processo de extração da lógica do sistema apresentado na Figura 10 fornece um conjunto de diretrizes que podem ser aplicadas em sistemas legados desenvolvidos no ambiente *Delphi*, sendo utilizadas as idéias centrais das etapas um e dois, extraindo delas "o que deve ser feito", sendo o "como isso é feito" é particular ao ambiente de desenvolvimento de cada sistema. A terceira etapa também é utilizada, porém com modificações, substituindo os grafos de fluxo de controle por diagramas de caso de uso e de seqüência **(BOOCH, 2000)**.

2.9.2 - TÉCNICAS DE EMPACOTAMENTO

O empacotamento do sistema é realizado após o entendimento de sua lógica. Nesta seção é apresentado um conjunto de técnicas de empacotamento para sistemas legados desenvolvidos na linguagem COBOL, cujas interfaces do usuário são orientadas a caracteres. Essas técnicas estão direcionadas a disponibilização do sistema legado para três categorias de sistemas, tais como: a *Web*, um ambiente cliente-servidor ou, sistemas locais. Isso caracteriza existência de uma técnica para tipo de categoria em que o sistema é disponibilizado. Essas técnicas podem ser a aplicadas ao empacotamento de sistemas desenvolvidos no ambiente *Delphi*.

SNEED (2001a) propõe o empacotamento de sistemas legados para uso em sistemas cliente-servidor através de CORBA-IDL. Esse empacotamento organiza o sistema em três camadas: a de interface do usuário, a de interface CORBA, responsável pelo acesso a camada que contém a aplicação legada, a qual representa a terceira camada, como mostra a Figura 11.

Segundo **SNEED (2001a)** existem três alternativas para realizar o empacotamento de sistemas legado para uso em sistema cliente-servidor:

1. **Redesenvolvimento**: consiste em reescrever a aplicação em uma nova linguagem, em um novo ambiente, baseada na funcionalidade legada;
2. **Reengenharia**: compreende na reprogramação das funções de acesso ao Banco de Dados, deixando os módulos básicos da aplicação inalterados. O programa legado deve passar por um processo de remodelagem e reestruturação, antes de se iniciar

o processo de migração para que as redundâncias sejam eliminadas;

3. **Encapsulamento**: corresponde ao desenvolvimento de uma "camada de software" que atua como interpretadora entre o sistema legado e a nova interface do usuário. Essa técnica preserva o sistema em seu ambiente nativo.

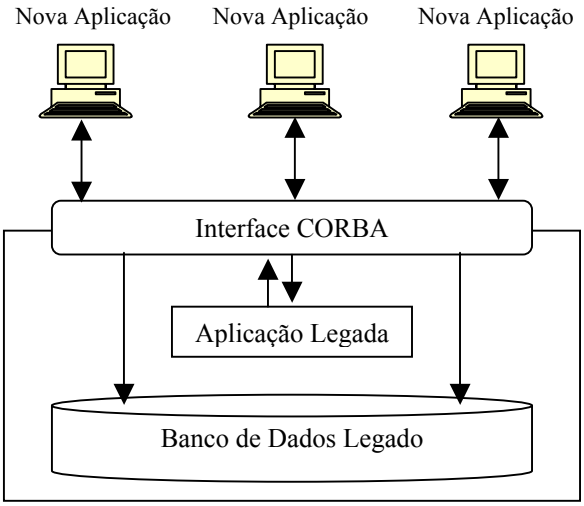


Figura 11 - Organização do Empacotamento utilizando CORBA (SNEED, 2001a)

Dentre as alternativas apresentadas, são utilizados neste trabalho os conceitos e as diretrizes que contribuem para manutenção do sistema legado em seu ambiente nativo. No entanto, a proposta de implementação utilizando CORBA-IDL não é utilizada, pois nem todos os ORBs oferecem o mapeamento IDL para a linguagem do código legado, ficando restrito a produtos comerciais que exigem altos investimentos.

Segundo **BRODIE E STONEBRAKER** *apud* **(SNEED, 1997)**, pode-se conectar sistemas legados com novas aplicações em três níveis de comunicação: de interface do usuário, de programa e de banco de dados, como mostra a Figura 12.

O nível de interface do usuário corresponde à migração da interface legada orientada a caracteres para uma nova interface gráfica, que passa a acessar o código legado. No nível de programa, parte da lógica da aplicação e as interfaces do usuário são reescritas, sendo as invocações ao sistema legado realizadas através da camada *wrapper*. No nível de banco de dados, tanto a interface do usuário quanto a lógica do sistema são migradas e, o acesso aos dados antigos é feito via *gateway*.

Os níveis de comunicação apresentados mostram as diferentes formas de disponibilização do código legado, sendo utilizados neste trabalho os conceitos dos níveis de interface e de programas, pois o objetivo principal é empacotar o sistema legado mudando apenas a interface do usuário para *Web*. Como os sistemas alvo neste trabalho foram

desenvolvidos no ambiente *Delphi* tanto de forma procedimental quanto com características da orientação a objetos, algumas funcionalidades podem não permitir a realização do empacotamento, sendo necessário reimplementá-las conforme proposto no nível de programa.

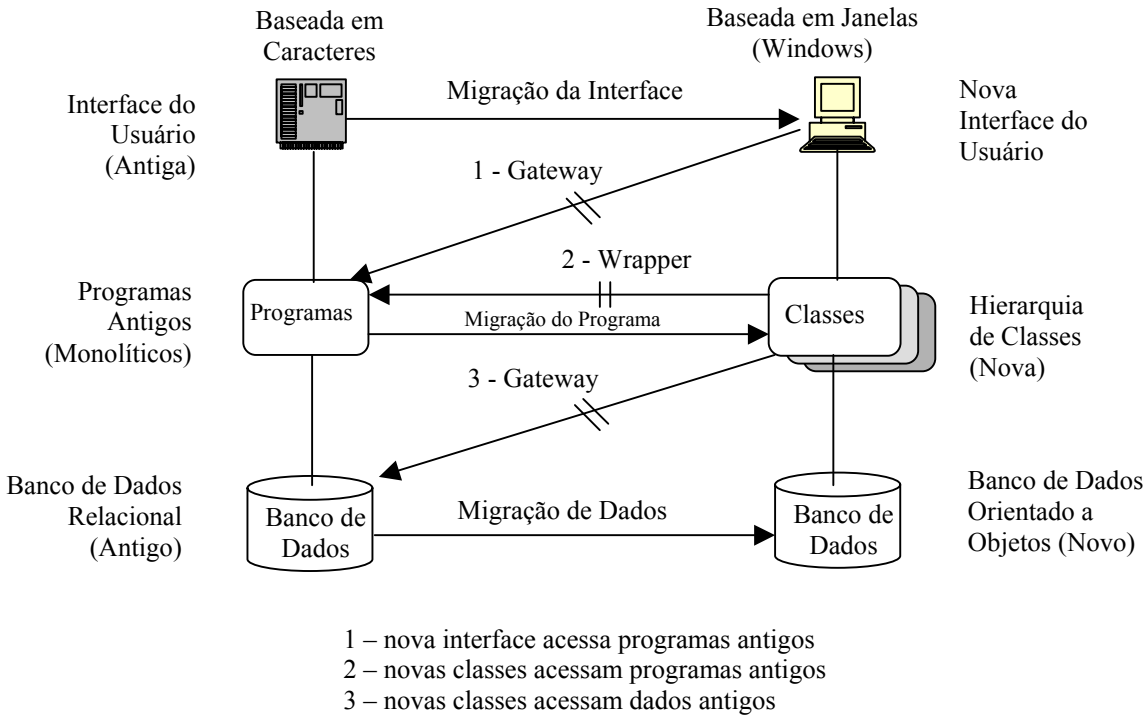


Figura 12 - Estratégias de Migração (SNEED, 1997)

Segundo **SNEED (2001a)** o empacotamento de sistemas legados para *Web* pode ser realizado através de interface XML, *eXtensible Markup Language*. Para otimizar esse processo de empacotamento **SNEED (2002)** utiliza duas ferramentas: "SoftWrap" e "XMLWrap". A primeira apresenta os nomes de todos os arquivos do sistema em uma lista, permitindo selecionar quais serão empacotados, preparando-os para aceitar a comunicação através do XML. A segunda tem por objetivo extrair as estruturas de dados, os nomes dos atributos, seus tipos e tamanhos, armazenando-os em uma tabela para que possa ser realizada sua conversão, pois pode haver divergências com os da nova interface do usuário. Essas ferramentas apóiam o processo de empacotamento, automatizando parte do processo e, atuando na preparação do sistema legado para integrá-lo com a nova interface **(SNEED, 2002)**.

Dessa proposta de empacotamento são extraídos conceitos referentes à preparação do código legado como a necessidade de conversão de tipos e de estruturas de dados entre as linguagens de programação do código legado e da nova interface do usuário. Destaca-se

também, a importância da automatização das tarefas, pois agiliza o processo de reengenharia.

MOWBRAY & ZAVANI (1995) introduzem o empacotamento por objetos, *object wrapping*, fazendo com que o código legado empacotado atue como objeto. Para isso, propõe cinco níveis de empacotamento:

- **Camadas *(Layering)*:** é um nível básico de empacotamento. Oferece mapeamento de uma aplicação para outra através da API, *Application Programming Interface*, podendo ser empregado o uso de IDL para disponibilizar as funcionalidades do sistema legado.
- **Migração de Dados *(Data Migration)*:** envolve a troca de modelo de dados. Um dos desafios refere-se a mudança do modelo relacional para o modelo orientado a objetos, devido ao esforço de conversão.
- **Reengenharia de Aplicações *(Reengineering Applications)*:** representa as mudanças que podem ser feitas em um sistema, quanto a funcionalidade, re-implementação, entre outras. Para isso, tem-se a análise de domínio, que oferece como contribuição, o entendimento do sistema para obtenção de seu modelo de análise;
- ***Middleware*:** é um termo utilizado para integração de software. Atua como um interpretador entre uma aplicação legada e a nova interface do usuário;
- **Encapsulamento *(Encapsulation)*:** essa técnica é também conhecida como caixa preta, pois o encapsulamento do sistema disponibiliza apenas interfaces de acesso do código legado.

Dentre os cinco níveis de empacotamento apresentados, foram utilizados os conceitos dos três últimos, pois os erros de funcionalidade encontrados num sistema podem ser corrigidos através do processo de reengenharia. A integração entre o sistema legado e a nova interface é feita pelo *Middleware*. Finalmente, a disponibilização das funcionalidades do sistema legado para a nova interface do usuário é feita através do Encapsulamento.

2.9.3 - PADRÕES PARA EMPACOTAMENTO DE SISTEMAS LEGADOS

Os padrões de empacotamento auxiliam na identificação do que deve ser empacotado e como disponibilizá-lo para a interface do usuário. A seguir são apresentados os quatro padrões de empacotamento propostos por **ASMAN (2000)**.

O padrão "Empacotar ou Não-empacotar" *(To Wrap or no To Wrap)* tem como propósito oferecer acesso ao código legado e não um modo de repará-lo. Seu objetivo é decompor o software legado em componentes, formando um *framework*. Isso pode caracterizar o reuso de software, pois esses componentes podem ser utilizados na construção da aplicação atual ou em outras.

O padrão "Consideração para Empacotar" *(Respect for Wrappers)* atua de forma análoga ao anterior. Seu objetivo também é a detecção de componentes no legado para incorporá-los à nova aplicação. Para isso, esse padrão tem como referência outro padrão, "*Managing Change to Reusable Software apud* **ASMAN (2000)**", que atua na eliminação de redundância de funcionalidade, no compartilhamento da aplicação, no desenvolvimento de componentes conforme os requisitos do usuário e na composição da nova aplicação reutilizando os componentes identificados.

O padrão "Empacotamento Simples ou Empacotamento Múltiplo" *(Single Wrappers and Multiple Wrappers)* é auxiliado pelos padrões apresentados anteriormente, pois esses determinam o que deve ser empacotado no sistema legado, através da decomposição do sistema em componentes. Esse padrão tem como objetivo fornecer a organização para esses componentes, através do empacotamento gradativo, em que as funcionalidades são adicionadas uma a uma. O empacotamento desse padrão pode ser classificado em duas formas:

- **Simples** (*Single Wrapper*)**:** nessa forma, o código legado é organizado em um pacote, que contém todos os serviços. Seu funcionamento procede-se da seguinte maneira: quando um objeto invoca um serviço da aplicação legada, uma mensagem é enviada para o *wrapper*, o qual chama a API legada e, quando o serviço exigido pelo objeto requisitante estiver pronto, uma mensagem é enviada para confirmar a implementação do serviço, como mostra a Figura 13.

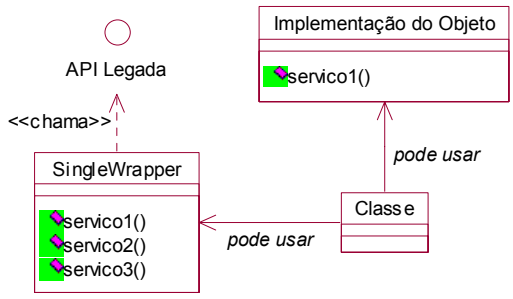


Figura 13 - Empacotamento Simples (ASMAN, 2000)

- **Múltiplo** (*Multiple Wrapper*)**:** nessa forma, os serviços do código legado são organizados de forma individual, em que cada um forma um pacote. A chamada a API legada é feita por cada um deles. Seu funcionamento é análogo ao funcionamento do empacotamento simples. A Figura 14 apresenta a forma de empacotamento Múltiplo.

C_Wrapper1 = Componente Wrapper1

C_Wrapper2 = Componente Wrapper2

C_Wrapper3 = Componente Wrapper3

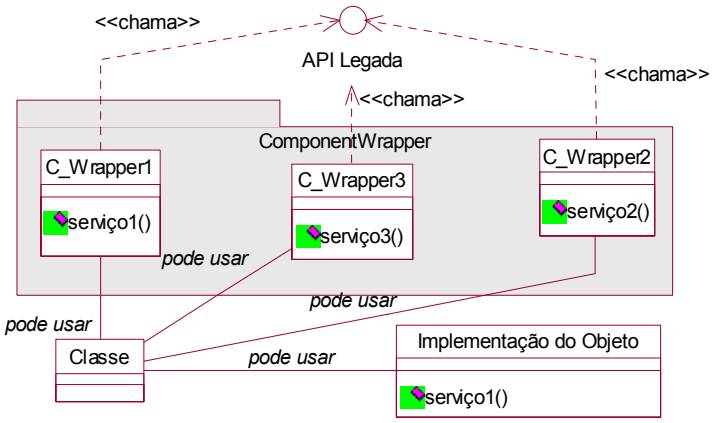


Figura 14 - Empacotamento Múltiplo (ASMAN, 2000)

O padrão "Empacotar, Não-Adaptar" (*Wrapping, Not Adapting*) permite o empacotamento de sistema sem característica da orientação a objetos, criando os objetos *wrappers*, que devem se comunicar com os demais objetos do sistema. Para a construção dos objetos *wrappers* é recomendado à utilização dos padrões de projeto *Adapter* e *Decorator* **(GRAND, 1998)**, pois permitem que os objetos trabalhem com os *Legacy Wrappers*, mas não como *Legacy Wrappers*. Esse padrão apóia a disponibilização dos sistemas legados e não uma forma para corrigi-los.

As contribuições desses padrões neste trabalho estão relacionadas à identificação das funcionalidades do sistema legado e ao agrupamento dessas em componentes de software, aumentando a possibilidade de reuso na reconstrução do sistema.

2.10 - CONSIDERAÇÕES FINAIS

Este capítulo apresentou um conjunto de temas que contribuíram para a condução e a realização deste trabalho. A Linguagem UML combinada aos métodos e família de padrões de Engenharia Reversa e Reengenharia contribui para a recuperação da lógica de um sistema procedimental tornando-o orientado a objetos. Além disso, a linguagem UML oferece suporte

à representação da integração entre o sistema legado empacotado e a nova interface do usuário.

A IHC oferece um conjunto de diretrizes que envolvem um estudo da interface legada, a avaliação dessa em relação a sua usabilidade e, ainda, a aplicação de critérios para melhoria de sua usabilidade, contribuindo significativamente com a migração de interfaces legadas para *Web*. Assim, os engenheiros de software e de interface podem verificar em quais pontos da interface podem ser melhorados quando a migração realizada é para *Web*.

Outro aspecto relevante é o reuso e componentes de software, que fornecem um conjunto de diretrizes para decomposição do sistema legado em componentes e para a reutilização desses na construção do novo sistema.

Os recursos de software contribuem com a identificação dos componentes de software no sistema legado e com a implementação do empacotamento desses componentes, que são disponibilizados para a nova interface do usuário através dos recursos de integração.

As técnicas e os padrões de empacotamento de software oferecem diretrizes que atuam desde a obtenção da lógica dos sistemas legados até seu o empacotamento, disponibilizando suas funcionalidades para a nova interface do usuário. Essas técnicas são direcionadas a um tipo de migração de sistema, que pode ser local, distribuída ou para *Web*. Isso implica na utilização de vários recursos para empacotar o sistema legado, como por exemplo, a utilização de CORBA-IDL para integrar sistemas legados numa arquitetura cliente-servidor ou XML para disponibilizar sistemas legados para *Web*, entre outros. No entanto, para a realização deste trabalho algumas dessas diretrizes são reutilizadas e outras são adaptadas, pois os sistemas que passam pelo processo de reengenharia de interface foram desenvolvidos no ambiente *Delphi*, com ou sem características da orientação a objetos.

O próximo capítulo apresenta o processo de reengenharia de interface de sistemas legados em *Delphi* direcionando o alvo de migração para *Web*.

CAPITULO 3

O PROCESSO DE REENGENHARIA DE INTERFACE

3.1 - CONSIDERAÇÕES INICIAIS

Este capítulo apresenta o Processo de Reengenharia de Interface (PRI) para sistemas legados desenvolvidos em *Delphi* com ou sem características de orientação a objetos para *Web*. Ele é apoiado pelas técnicas de empacotamento de software (*wrapping*) pelo processo de reengenharia de interface MORPH e pelos critérios de usabilidade.

As técnicas de empacotamento auxiliam na obtenção da lógica do sistema, na decomposição do sistema em pacotes, componentes de software, e na implementação do empacotamento **(SNEED, 1995, 1996a, 1996b, 1996c, 1997, 2000, 2001a, 2001b, 2002) (STROULIA et al, 1999, 2000)**.

O processo MORPH oferece um conjunto de diretrizes para a migração de sistemas legados com interfaces orientadas a caracteres para gráficas, que podem ser para *Web* ou para sistemas locais **(MOORE, 1994a, 1994b, 1996, 1998) (MOORE & MOSHKINA, 2000)**.

Os critérios de usabilidade atuam na migração da interface legada, no desenvolvimento da nova interface e na avaliação dessa como produto final **(NIELSEN,**

1994, 2000) (NIELSEN & TAHIR, 2000).

O processo de Reengenharia de Interface proposto neste trabalho visa apoiar a migração de sistemas legados de maneira geral, realizando o empacotamento de suas funcionalidades e acoplando uma nova interface do usuário para *Web*. O empacotamento do sistema atende a diversos tipos de migração como para sistemas locais, sistemas distribuídos ou para *Web*. Para esses dois últimos é necessária a inserção de uma camada de *middleware*, sendo CORBA para sistemas distribuídos e *Servlets* ou JSP para *Web*. O desenvolvimento da nova interface do usuário é baseado em critérios de usabilidade tanto no desenvolvimento quanto na avaliação dessa como produto final.

Os sistemas que passam por esse processo foram desenvolvidos segundo o paradigma procedimental ou orientado a objeto com interfaces gráficas. Neste trabalho, o processo foi instanciado para sistemas desenvolvidos no ambiente *Delphi*, em que suas funcionalidades foram empacotadas e uma nova interface usuário para *Web* foi desenvolvida.

Este capítulo apresenta na seção 3.2 a estrutura geral do Processo de Reengenharia de Interface, sendo seu detalhamento apresentado nas seções 3.2.1 a 3.2.7 e, na seção 3.3 são apresentadas as considerações finais.

3.2 - O PROCESSO DE REENGENHARIA DE INTERFACE

O Processo de Reengenharia de Interface é composto por sete etapas, que compreende desde o estudo da interface e da lógica do sistema até a implementação do novo sistema, que consiste na integração entre o sistema legado empacotado e a nova interface do usuário, como mostra a Figura 15.

O processo tem início com o código fonte e o executável do sistema legado. Ao executar o sistema legado é realizado um estudo de sua interface e de sua lógica para que ele possa ser organizado. Em seguida, são feitas a identificação dos contextos e a reorganização das suas funcionalidades para cada contexto identificado.

Com a etapa de identificação dos contextos finalizada, o engenheiro de software e de interface possui um conjunto de documentos que fornecem informações sobre a lógica do sistema legado e a usabilidade da interface do usuário, permitindo que sejam realizadas as etapas de composição dos pacotes, que corresponde ao empacotamento do sistema e de confecção da nova interface do usuário.

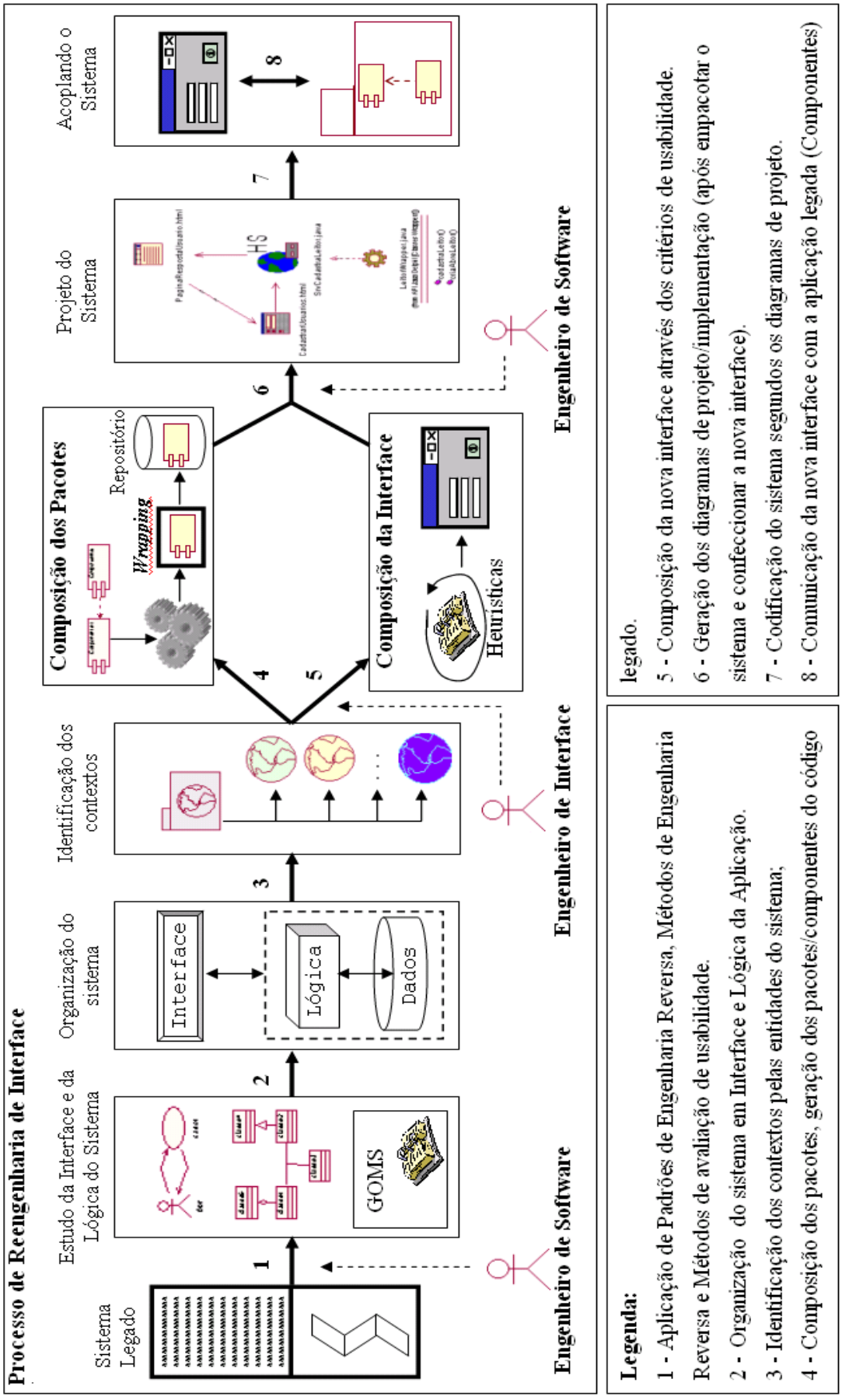


Figura 15 - Processo de Reengenharia de Interface (PRI)

A composição dos pacotes consiste na organização das funcionalidades do sistema em pacotes, componentes de software, os quais são envolvidos por uma "camada de software", que viabiliza a comunicação entre a funcionalidade legada empacotada e a nova interface do usuário.

A confecção da nova interface do usuário corresponde ao mapeamento da interface legada para a nova interface *Web*. Esse mapeamento é realizado com base nos critérios de usabilidade tanto para o entendimento da interface legada quanto para a construção da nova interface *Web*, que é realizada de maneira gradativa.

Com a definição do empacotamento do sistema e com a confecção da nova interface do usuário, cabe ao engenheiro de software projetar o novo sistema, fazendo a integração de ambos, para que possa ser realizada a sua implementação, finalizando o processo de Reengenharia de Interface. As etapas que compõem esse processo são detalhadas nas seções 3.2.1 a 3.2.7.

3.2.1 - Estudo da Interface e da Lógica do Sistema

O estudo da interface é realizado através da execução do sistema, apoiado pelo método GOMS, sendo utilizada neste trabalho sua variação NGOMSL **(JOHN et al, 1994a) (JOHN et al, 1994b) (JOHN et al, 1996) (KIERAS, 1999)** e com base nas heurísticas de **NIELSEN (1994)**, que foram utilizadas sem a aplicação formal do método Avaliação Heurística.

A obtenção da lógica do sistema **(SNEED, 2000)** é feita com base na análise do código fonte e com o conhecimento que o engenheiro de software adquire com o estudo de sua interface. A representação das funcionalidades do sistema legado é feita utilizando um conjunto de técnicas da UML **(BOOCH, 2000)** como diagramas de casos de uso e diagramas de seqüência.

A execução do sistema legado para aplicação do método GOMS produz os seguintes documentos: o modelo de interação GOMS, a lista de erros de usabilidade da interface, a lista de erros de funcionalidade do sistema e a lista de casos de uso. Esses documentos são confeccionados paralelamente ao modelo de interação GOMS, pois o sistema está sendo executado em todas as suas funcionalidades.

O modelo de interação GOMS fornece a representação teórica das funcionalidades do sistema gerando a seqüência de passos para a execução de uma tarefa, conforme apresentado no capítulo 2, seção 2.4.

Os erros de usabilidade de uma interface são identificados tomando por base, informalmente, os critérios de usabilidade de **NIELSEN (1994)** e organizados em uma lista como mostra a Figura 16. O número das heurísticas refere-se às apresentadas no capítulo 2, seção 2.4.

Lista de erros de interface		
Projeto: nome do sistema		**Data de Criação:** data
Número do erro	**Heurística**	**Descrição do erro encontrado**
1	Número	Descrição para o erro 1.
2	Número	Descrição para o erro 2.
. . .	. . . . .	. . . . .
N	Número	Descrição para o erro N.

Figura 16 - Lista de erros de interface do sistema (parcial)

Os erros referentes à funcionalidade do sistema são identificados com base na aplicação do método GOMS e complementado com a análise do código fonte do sistema legado. Os erros identificados são organizados em uma lista como mostra a Figura 17.

Lista de erros de funcionalidade	
Projeto: nome do sistema	**Data de Criação:** data
Número do erro	**Descrição do erro encontrado**
1	Descrição para o erro 1.
2	Descrição para o erro 2.
. . .	. . . . .
N	Descrição para o erro N.

Figura 17 - Lista parcial de erros de funcionalidade

Os casos de uso do sistema são gerados a partir da execução do sistema e da análise do código fonte, pois cada funcionalidade detectada torna-se candidata a um caso de uso. A Figura 18 mostra a lista de casos de uso.

O engenheiro de software, de posse desses documentos, possui um conjunto de informações referentes à usabilidade da interface e à lógica do sistema. Essa etapa prevê a modelagem dos diagramas de casos de uso com base na Lista de caso uso, Figura 18, que deve ser complementada com os respectivos diagramas de seqüência para evidenciar a troca de mensagens no sistema.

Lista de casos de uso		
Projeto: nome do sistema	**Data de Criação:** data	
Número	**Funcionalidade**	**Caso de Uso**
1	Funcionalidade do caso de uso 1.	Nome do caso de uso 1.
2	Funcionalidade do caso de uso 2.	Nome do caso de uso 2.
. . .	. . . . .	. . . . .
N	Funcionalidade do caso de uso N.	Nome do caso de uso N.

Figura 18 - Lista de casos de uso

A próxima etapa trata da organização do sistema, que consiste na preparação do código legado, identificando as funcionalidades candidatas ou não ao empacotamento e na organização dos componentes da interface legada para serem mapeados na etapa de confecção da nova interface do usuário.

3.2.2 - ORGANIZAÇÃO DO SISTEMA

A organização do sistema trata da separação da interface do usuário da sua lógica. É necessário que o engenheiro de software tenha conhecimento do ambiente de desenvolvimento *Delphi*, pois ele auxilia na identificação dos componentes do sistema e na análise do código fonte.

A análise visual dos componentes do sistema legado no ambiente de desenvolvimento *Delphi* resulta num conjunto de informações que deve ser organizado em uma lista como mostra a Figura 19. Essa lista contém o número do componente em ordem de ocorrência de detecção, seu nome no sistema e seu tipo, sua classificação (I - Interface, BD - Banco de dados, L - Ligação) e as informações neles contidas. As informações presentes nessa lista são reutilizadas nas etapas futuras do processo, tais como: para a de extração do código candidato ou não ao empacotamento e para o mapeamento dos componentes da interface legada para a construção da nova interface do usuário.

Em seguida, o engenheiro de software deve fazer a análise do código fonte do sistema legado, que corresponde a dois arquivos: os "`.dfm`", que contêm informações sobre a especificação dos componentes utilizados no sistema e os "`.pas`", que contêm a lógica do sistema. Dessa análise duas listas devem ser organizadas, a de procedimentos candidatos ao empacotamento e a de procedimentos candidatos à implementação. A segunda, representa as funcionalidades que não podem ser empacotadas, que correspondem, principalmente, às

funcionalidades de inserção e consulta no banco de dados, pois são desenvolvidas através da conexão dos componentes de interface com os de base de dados.

Lista de componentes			
Projeto: nome do sistema		**Data de Criação:** data	
Número	**Nome : Tipo**	**Classificação**	**Informações contidas**
1	Nome1 : Tipo1	(I | BD | L)	Informação 1.
2	Nome2 : Tipo2	(I | BD | L)	Informação 2.
. . .	. . . .	. . . .	. . . . .
N	NomeN : TipoN	(I | BD | L)	Informação N.

Figura 19 - Lista de componentes

A Lista de procedimentos candidatos ao empacotamento é elaborada com as informações mostradas na Figura 20, para cada procedimento analisado. A Figura 21 apresenta a lista de procedimento candidatos à implementação.

Lista de procedimentos candidatos ao empacotamento			
Projeto: nome do sistema		**Data de Criação:** data	
Número	**Procedimento**	**Descrição**	**Código Bruto**
1	Procedimento 1	Descrição 1	Código fonte 1
2	Procedimento 2	Descrição 2	Código fonte 2
. . .	.....	.....	.....
N	Procedimento N	Descrição N	Código fonte N

Figura 20 - Lista de procedimentos candidatos ao empacotamento

Lista de procedimentos candidatos à implementação		
Projeto: nome do sistema		**Data de Criação:** data
Número	**Procedimento**	**Descrição**
1	Procedimento A	Descrição 1.
2	Procedimento B	Descrição 2.
. . .	.....	.....
N	Procedimento Z	Descrição N.

Figura 21 - Lista de procedimentos candidatos à implementação

Para os sistemas implementados com características de orientação a objetos, é

necessária apenas a confecção da Lista de componentes, pois as suas funcionalidades já se encontram organizadas em classes.

A próxima etapa corresponde à identificação dos contextos do sistema, em que as suas funcionalidades são organizadas dentro do contexto que atuam, iniciando a criação da visão orientada a objetos do sistema.

3.2.3 - IDENTIFICAÇÃO DOS CONTEXTOS DO SISTEMA

Essa etapa é fundamental nos sistemas procedimentais, pois suas funcionalidades podem estar inseridas em contextos distintos e por não haver qualquer comprometimento com a organização do código do sistema. Em sistemas orientados a objetos essa etapa pode ser eliminada, pois o código fonte já se encontra organizado em classes.

O engenheiro de software utiliza as listas de procedimentos candidatos ao empacotamento e à implementação para identificar em que contexto cada um dos componentes utilizados na construção do sistema está associado. Dessa forma, uma lista contendo os contextos identificados e seus respectivos métodos como mostra a Figura 22 é elaborada.

Lista de contextos e métodos			
Projeto: nome do sistema		**Data de Criação:** data	
Número	**Contexto**	**Procedimento/ Função**	**Descrição**
1	Contexto 1	Assinatura do Método 1	Descrição 1.
2	Contexto 2	Assinatura do Método 2	Descrição 2.
. . .	. . . . .	. . . . .	. . . . .
N	Contexto N	Assinatura do Método N	Descrição N.

Figura 22 - Lista de contextos e métodos

Ao término dessa etapa o engenheiro de interface e de software possui um conjunto de documentos que fornece informações referentes à interface do usuário e à lógica do sistema, possibilitando o início das etapas referentes à composição dos pacotes e à confecção da nova interface do usuário. Essas etapas são independentes, podendo ser desenvolvidas em paralelo, aumentando a produtividade no processo de reengenharia.

3.2.4 - COMPOSIÇÃO DOS PACOTES

A etapa de composição dos pacotes é ilustrada na Figura 23. Após a identificação dos contextos, confecciona-se os modelos de dados e de análise do sistema, para a representação do empacotamento em diferentes níveis: de implementação e lógico. A partir desses modelos inicia-se a especificação do empacotamento, que consiste da preparação do código legado, extraindo dele as funcionalidades possíveis ou não de serem empacotadas. Em seguida, é confeccionado o modelo de componentes, que permite ao engenheiro de software definir quais as funcionalidades devem ser disponibilizadas para a nova interface do usuário. Finalmente, faz-se a implementação de cada componente, que consiste em envolvê-los por uma "camada de software", para viabilizar a comunicação entre a funcionalidade legada empacotada e a nova interface do usuário.

3.2.4.1 - IDENTIFICAÇÃO DO MODELO DE DADOS

Esta etapa corresponde à obtenção do modelo de dados, pois uma das características do empacotamento de sistemas (*wrapping*) é a preservação de sua lógica **(SNEED, 2001a) (SNEED, 2001b)**. Esse modelo fornece a lógica do empacotamento do sistema, que é obtido com apoio da Família de Padrões de Engenharia Reversa e de REengenharia Orientada a Objetos, FaPRE/OO **(RECCHIA, 2002)**.

Os sistemas desenvolvidos no ambiente *Delphi* possuem, basicamente, os seguintes tipos de armazenamento de dados:

- **Arquivos binários:** são estruturas de dados declaradas dentro do código fonte, sendo gerenciadas pela lógica do sistema;
- **Banco de dados:** são dados externos ao ambiente de desenvolvimento, podendo estar sob cuidados de um Sistema Gerenciador de Banco de Dados (SGBD) ou de armazenamento em tabelas;
- **Classes:** são estruturas declaradas dentro do código fonte, caracterizando a orientação a objetos, no entanto, seu armazenamento pode ser feito em banco de dados relacionais **(LEMOS, 2002)**.

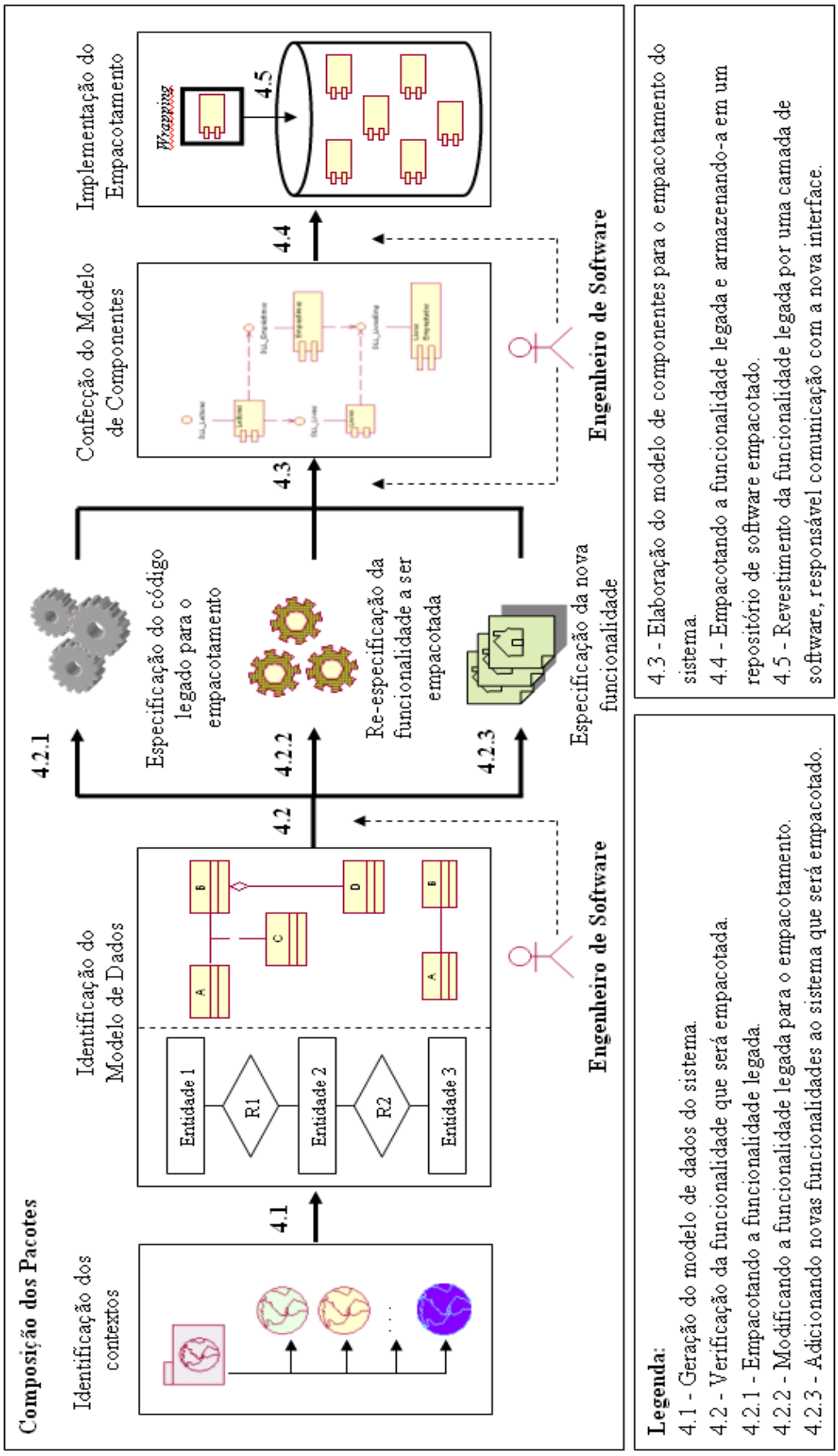


Figura 23 - PRI - Composição dos Pacotes

Os dois últimos são os mais comumente encontrados e os que são abordados nesta seção. A partir da análise do código fonte, localiza-se os componentes que fazem conexão com o banco de dados. Esses componentes possuem uma propriedade que mostra a localização física do banco de dados através de um diretório ou de um "*alias*", que corresponde ao nome dado ao banco de dados no sistema operacional. Com isso, inicia-se a construção do MER, Modelo Entidade Relacionamento, apoiado pela família de padrões FaPRE/OO **(RECCHIA, 2002)**. A identificação das tabelas ou entidades do MER é apoiada pelo ambiente de desenvolvimento do banco de dados utilizado no sistema legado, sendo realizada de forma particular para cada um deles. Em seguida, para cada uma das entidades/tabelas presentes no banco de dados são extraídas as seguintes informações: nomes das entidades/tabelas, seus atributos e tipos, além das chaves primárias e estrangeiras, organizando-as em uma Lista de tabelas, como mostra a Figura 24.

Lista de tabelas					
Projeto: nome do sistema			**Data de Criação:** data		
Tipo de Dados: Tabelas					
Número	**Tabela**	**Atributos**	**Tipo**	**Chave Primária**	**Chave Estrangeira**
1	Entidade/ Tabela	Atributo 1	T1	*	*
		Atributo 2	T2		
		Atributo 3	T3		
		. . . . .	. . .		
		Atributo N	TN		

Figura 24 - Lista de Tabelas

A seguir, o engenheiro de software pode iniciar a modelagem dos dados, criando o MER. Esse modelo é fundamental para os sistemas procedimentais, pois representa o empacotamento no nível de implementação. Nos sistemas orientados a objetos **(LEMOS, 2002)**, o MER representa apenas a lógica dos dados, pois empacotamento é representado pelo diagrama de classe do sistema tanto no nível de implementação quanto no lógico.

3.2.4.2 - ESPECIFICAÇÃO DO EMPACOTAMENTO

O entendimento do código legado é o ponto central para que seja realizada a sua reorganização de acordo com cada contexto identificado na etapa anterior, seção 3.2.3. Essa

etapa pode ser eliminada para os sistemas orientados a objetos **(LEMOS, 2002)** devido à organização do código em classes. No entanto, é fundamental nos sistemas procedimentais considerando a desorganização de código existente, pois algumas funcionalidades atuam em diversos contextos do sistema.

Para realizar esta etapa o engenheiro de software baseia-se nos seguintes documentos: a lista de contextos e métodos, seção 3.2.3, e a lista de tabelas, seção 3.2.4.1. Com base nesses documentos as pseudo-classes do sistema são criadas através da "associação" das entidades do MER com os contextos identificados, que fornecem, respectivamente, os atributos e métodos às pseudo-classes. Nessa etapa pode ocorrer a mudança nos nomes dos procedimentos como forma de adequação ao novo contexto de orientação a objetos. De modo análogo são criadas todas as pseudo-classes do sistema.

O próximo passo é a criação da primeira visão orientada a objeto do sistema, representada pelo MASA, Modelo de Análise do Sistema Atual, com base no MER obtido na seção 3.2.4.1, transferindo as entidades para as pseudo-classes. Em seguida, o engenheiro de software faz o refinamento do MASA gerando o MAS, Modelo de Análise do Sistema, que representa a visão orientada a objetos do sistema. A construção desses modelos é realizada com base na família de padrões FaPRE/OO **(RECCHIA, 2002)**. Esses modelos são essenciais para sistemas procedimentais, pois representam, respectivamente, a implementação e a lógica do empacotamento.

Para os sistemas orientados a objetos **(LEMOS, 2002)** o empacotamento ocorre de maneira natural, não havendo a necessidade da reorganização de código, apenas a obtenção do modelo de classe para representar a implementação e a lógica do empacotamento. Esse modelo é confeccionado através da análise do código fonte, arquivos "`.pas`", apoiada pela família de padrões FaPRE/OO **(RECCHIA, 2002).**

Após a confecção dos modelos de análise o engenheiro de software possui um conjunto de documentos que fornecem informações referentes à implementação e a lógica do empacotamento. Essas informações permitem que sejam iniciadas as etapas de especificação do código legado para o empacotamento, que pode ocorrer de três formas: extraindo o código legado candidato ao empacotamento e inserindo-o como métodos na pseudo-classe/classe que o detém; re-especificando a funcionalidade a ser empacotada, que consiste na correção dos erros funcionais identificados na etapa 3.2.1; e especificando a nova funcionalidade, que consiste na implementação daquelas funcionalidades que não são possíveis de serem empacotadas.

3.2.4.2.1 - ESPECIFICAÇÃO DO CÓDIGO LEGADO PARA O EMPACOTAMENTO

Esta etapa consiste em extrair do código legado as funcionalidades que são possíveis de serem empacotadas, agrupando-as como métodos nas novas pseudo-classes/classes do sistema. Para isso, o engenheiro de software baseia-se nos diagramas de classe, MASA e MAS, gerados na etapa anterior, seção 3.2.4.2; na lista de procedimentos candidatos ao empacotamento obtida na etapa de segmentação do sistema, seção 3.2.3; e nos digramas de caso de uso e de seqüência obtidos na etapa de estudo da lógica do sistema, seção 3.2.1.

A extração do código legado tem como referência os diagramas de seqüência obtidos na etapa de estudo da lógica do sistema, seção 3.2.1, pois esses apresentam todos os passos para execução de uma funcionalidade do sistema. Assim, o código legado referente a esses passos é extraído e inserido como método na pseudo-classe/classe do sistema.

Essa etapa pode ser eliminada nos sistemas desenvolvidos com características da orientação a objetos **(LEMOS, 2002)**, pois o código fonte já se encontra organizado como métodos nas classes.

3.2.4.2.2 - RE-ESPECIFICAÇÃO DA FUNCIONALIDADE A SER EMPACOTADA

Corresponde à correção dos erros de funcionalidade do sistema. Para isso, o engenheiro de software utiliza a lista de erros de funcionalidade detectados na etapa de estudo da interface e da lógica do sistema, seção 3.2.1.

Os erros tratados nesta etapa são aqueles, em que não é necessária a re-implementação de código para repará-lo. Esses erros resumem-se, por exemplo, a campos de consulta cuja resposta não é única, cuja solução é a troca do campo de consulta. Essas correções devem ser realizadas no código fonte legado para depois inseri-lo como método na pseudo-classe.

Existem erros que devem ser tratados na interface do usuário. Por exemplo, aqueles relacionados diretamente com a validação de dados podem ser tratados na interface, inibindo a entrada de dados inválidos, evitando que a informação percorra o caminho do cliente ao servidor para ser validada, minimizando problemas com a velocidade do sistema. Uma lista de erros com essa finalidade como mostra a Figura 25 é elaborada.

Lista de erros que serão tratados na interface	
Projeto: nome do sistema	**Data de Criação:** data
Número do erro	**Erro Encontrado**
1	Descrição 1.
2	Descrição 2.
. . .	. . . . .
N	Descrição N.

Figura 25 - Lista de erros que serão tratados na interface

3.2.4.2.3 - ESPECIFICAÇÃO DA NOVA FUNCIONALIDADE

Esta etapa trata da implementação das funcionalidades que não são possíveis de serem empacotadas. Para isso, o engenheiro de software baseia-se na lista de procedimentos candidatos à implementação gerada na etapa de segmentação do sistema, seção 3.2.2. Esses procedimentos correspondem às funcionalidades que estão associadas aos componentes *Delphi* e que tratam, principalmente, da inserção e da consulta no banco de dados, pois a construção dessas funcionalidades envolve, basicamente, a conexão de componentes de interface com componentes de ligação, os quais estão associados a componentes de banco de dados, que contém uma tabela do banco de dados associada a eles. Dessa forma, é estabelecida uma conexão entre a interface do usuário e o banco de dados, que atua como um canal de comunicação bidirecional, que corresponde à inserção e a consulta no banco de dados. Esta etapa não é necessária para os sistemas legados desenvolvidos com características da orientação a objetos **(LEMOS, 2002)**, pois essas funcionalidades já se encontram implementadas.

Ao término das etapas de especificação do empacotamento, algumas funcionalidades sofreram alterações e outras foram criadas, contribuindo para a atualização dos modelos de análise, MASA e MAS, obtidos anteriormente. Além disso, os nomes dos métodos podem ter sido atualizados como forma de adequá-los ao contexto da orientação a objetos. Dessa forma, cabe ao engenheiro de software atualizar esses modelos para que possa ser criada a visão de componentes de software do sistema. Ao atualizar esses modelos pode-se notar um aumento do número de métodos de cada pseudo-classe/classe, isso pode ocorrer devido ao reuso de código, pois o empacotamento de software usando componentes elimina a redundância de código, fazendo com que as funcionalidades atuem em contextos definidos.

A atualização dos modelos MASA e MAS são fundamentais para a implementação do empacotamento, pois o primeiro, MASA, representa o nível de implementação do

empacotamento para os sistemas procedimentais, em que cada pseudo-classe é transformada em um componente de software, enquanto que o segundo, MAS, representa o nível lógico do empacotamento. Nos sistemas orientados a objetos o modelo de classe, MAS, representa os níveis de implementação do empacotamento e cada classe é transformada em um componente.

3.2.4.3 - CONFECÇÃO DO MODELO DE COMPONENTES

O engenheiro de software deve fazer a associação de cada pseudo-classe do MASA, para os sistemas procedimentais, ou de cada classe do MAS, para os sistemas orientados a objetos, com um componentes de software de mesmo nome. Isso é realizado para que as funcionalidades dessas pseudo-classes/classes sejam disponibilizadas nas interfaces dos componentes, permitindo acesso a nova interface do usuário.

Em seguida, o engenheiro de software confecciona o modelo de componentes do sistema baseado nos diagramas de classe, MASA e MAS, obtidos na etapa anterior, seção 3.2.4.2, e de seqüência obtidos na seção 3.2.1. Após a confecção do diagrama de componentes, elabora-se uma lista de funcionalidades com todos os componentes contendo: seu nome e as funcionalidades que devem ser disponibilizadas em suas interfaces, como mostra a Figura 26.

Lista de funcionalidade dos componentes	
Projeto: nome do sistema	**Data de Criação:** data
Nome do Componente	**Funcionalidades presentes na interface**
Componente 1	Funcionalidade 1. Funcionalidade 2. Funcionalidade 3. . . . . Funcionalidade N

Figura 26 - Lista de funcionalidades implementadas no empacotamento

O engenheiro de software pode iniciar a etapa de implementação do empacotamento, que corresponde ao desenvolvimento das interfaces de cada componente, considerando a lista de funcionalidade dos componentes criada nesta etapa.

3.2.4.4 - IMPLEMENTAÇÃO DO EMPACOTAMENTO

A implementação do empacotamento consiste em desenvolver uma "camada de software" em cada pseudo-classe/classe do sistema, que representa a interface de cada componente de software do sistema. O objetivo dessa camada é receber os dados da nova interface do usuário, convertê-los em dados legíveis **(SUN, 2003)** à funcionalidade legada e encaminhá-los ao código legado para serem processados. O caminho inverso também é considerado. A Figura 27 mostra a estrutura física do empacotamento, mostrando a comunicação desde o *middleware* até o sistema legado. Nessa estrutura destaca-se a camada de software, que contém dois arquivos: "`AcessoMiddleware.java`" e "`AcessoLegado.dpr`". O primeiro atua do lado da interface do usuário, permitindo que o recurso de *middleware* tenha acesso ao sistema legado. O segundo corresponde à interface dos componentes de software, viabilizando a comunicação entre a interface do usuário e o código legado.

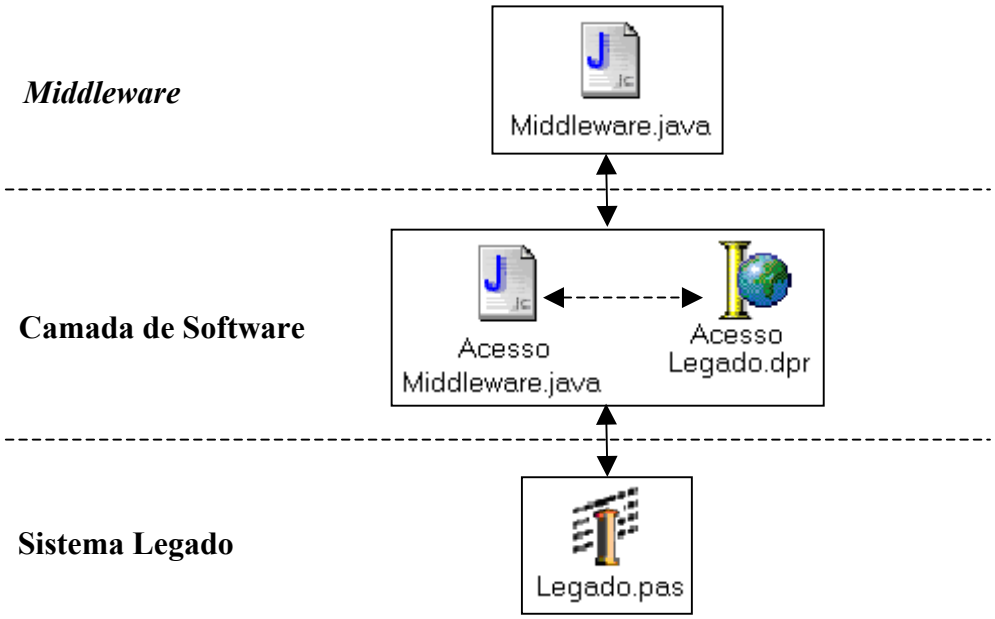


Figura 27 - Organização física do empacotamento

A implementação dos componentes do sistema baseada no modelo e na lista de funcionalidades de componentes obtidos na etapa anterior, seção 3.2.4.3, é feita encerrando a etapa de composição dos pacotes. Assim, o sistema encontra-se organizado em componentes de software com interfaces de acesso bem definidas, oferecendo a nova interface do usuário acesso ao sistema legado.

A próxima seção trata dos aspectos referentes à migração da interface legada do

usuário para *Web* e, apresenta, também, um conjunto de diretrizes para realizar essa migração.

3.2.5 - CONFECÇÃO DA NOVA INTERFACE DO USUÁRIO

Essa etapa está centrada na migração de uma interface local desenvolvida no ambiente *Delphi* para *Web*, como mostra a Figura 28. Após a identificação dos contextos do sistema, confecciona-se o modelo navegacional, que fornece o caminho a ser percorrido na interface para a execução das tarefas. Em seguida, realiza-se o mapeamento dos componentes de interface, isto é, a tradução dos componentes da interface legada para os novos componente *Web*, que são utilizados para a confecção da nova interface do usuário. O processo para a confecção da nova interface do usuário é evolutivo, utilizando-se os critérios de usabilidade para o desenvolvimento.

3.2.5.1 - CONFECÇÃO DO MODELO NAVEGACIONAL

Os caminhos a serem percorridos na interface para a execução das tarefas compõem o modelo navegacional, que proporciona a usabilidade dos sistemas *Web*.

Antes de iniciar a construção do modelo navegacional é necessário caracterizar a estrutura física de sistemas voltados para *Web*, Figura 29. Basicamente, esses modelos são constituídos de uma página principal que contém um conjunto de *links* para as demais páginas, chamadas de secundárias. Essas páginas são compostas por elementos gráficos utilizados na confecção de seus *layouts* que auxiliam na execução das tarefas, melhorando sua usabilidade.

- **Página principal:** corresponde a referência inicial para o sistema, nela estão contidas todas as referências para as demais informações do sistema. Contém as informações referentes ao negócio da empresa, deixando claro seus objetivos, ou seja, a finalidade do *site*;
- **Páginas secundárias:** são partes integrantes do sistema, contendo informações especificas de um determinado contexto;
- **Elementos para composição de *layout*:** corresponde a um conjunto de figuras, textos, componentes gráficos, entre outros, que são responsáveis pela ilustração da interface tornando-a mais agradável ao usuário.

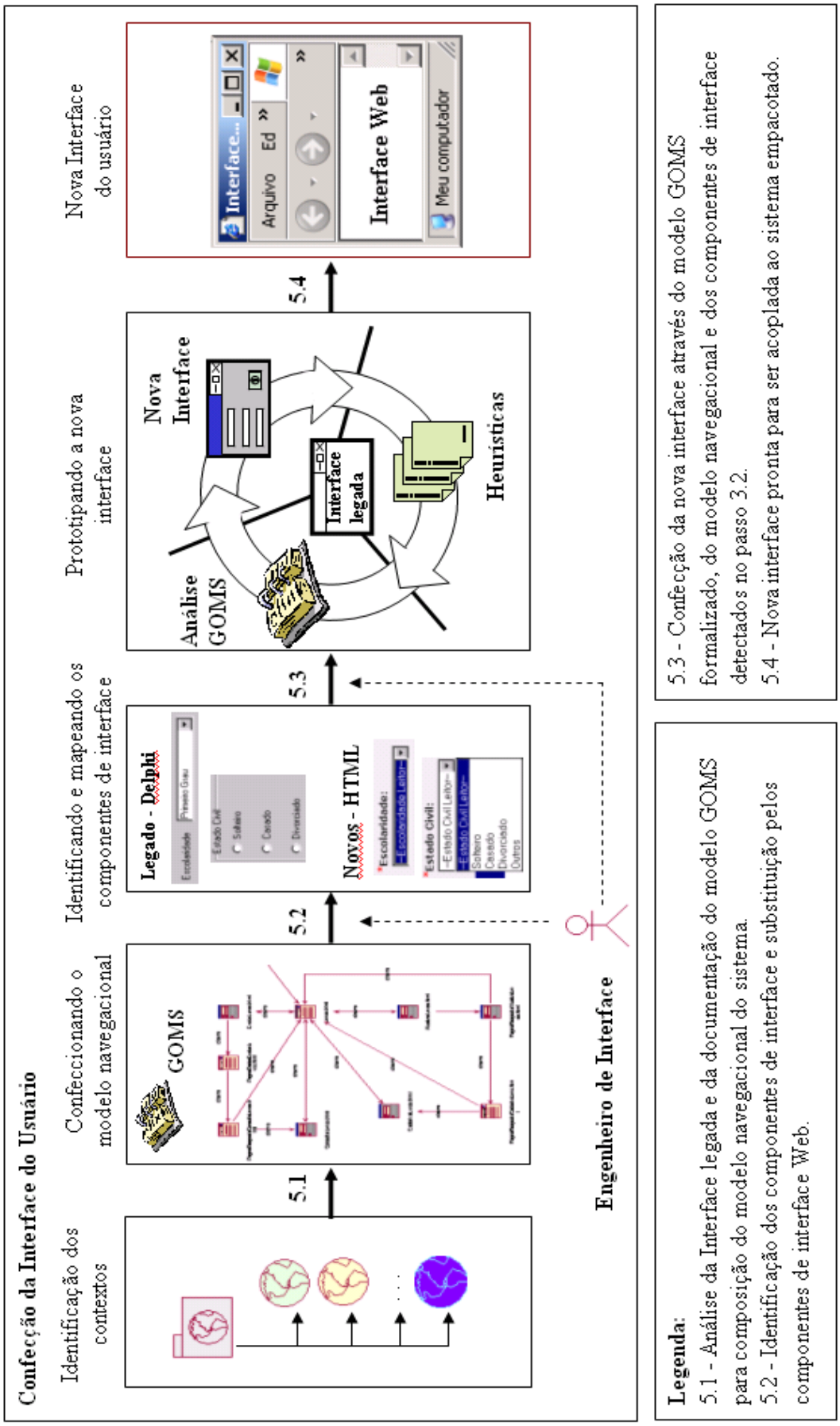


Figura 28 - PRI - Confecção da Nova Interface do Usuário

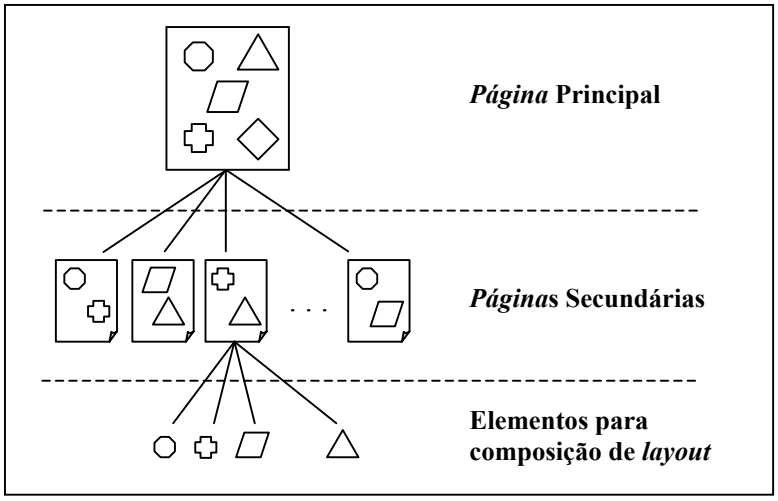


Figura 29 - Estrutura física de um sistema *Web*

A construção do modelo navegacional é baseada na estrutura física apresentada e nos documentos obtidos nas etapas anteriores do processo como: o modelo de interação GOMS e a lista de caso de uso obtidos na etapa de estudo da interface e da lógica do sistema, seção 3.2.1, e a lista de contextos e métodos obtida da etapa de organização do sistema, seção 3.5.2. O modelo navegacional tem início com a definição da página principal do sistema, que contém um conjunto de *links* para cada um dos contextos do sistema, que representam as páginas secundárias.

O detalhamento do modelo navegacional para cada contexto do sistema é então realizado, pois cada um deles é composto por um conjunto de funcionalidades conforme apresentado na lista de contextos e métodos, seção 3.2.3. Para cada funcionalidade do contexto do sistema deve ser criada uma página, que é referenciada pela página que representa o contexto do sistema. Para cada funcionalidade do sistema deve existir uma página de resposta, que são extraídas das mensagens de repostas dos diagramas de caso de uso. A navegabilidade entre as páginas de cada contexto é extraída do modelo de interação GOMS.

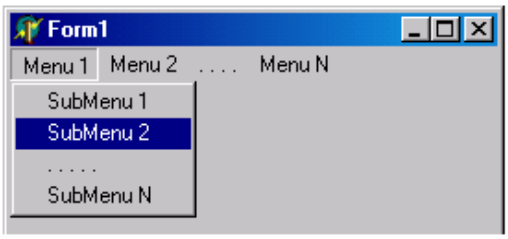
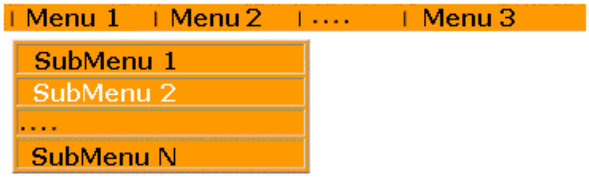
3.2.5.2 - Identificando e Mapeando os Componentes de Interface

Esta etapa trata do mapeamento dos componentes de interface *Delphi* para componentes de interface *Web*, utilizando a lista de componentes de interface elaborada na seção 3.2.2. Como os sistemas desenvolvidos no ambiente *Delphi* possuem interfaces gráficas WIMP (*Windows, Icon, Menu, Pull-down menu*), o mapeamento pode ocorrer das seguintes formas: a) Equivalente: representa a substituição natural do componente que existe tanto no

código legado *Delphi* quanto no novo em HTML e b)Funcional: quando não é possível fazer o mapeamento equivalente ou quando se necessita adequar o *layout* da nova interface, substituindo o componente da interface legada por outro de funcionalidade equivalente. A partir dessas formas de mapeamentos são apresentados os componentes de interface mais encontrados em interfaces *Delphi* e seu mapeamento para a linguagem HTML.

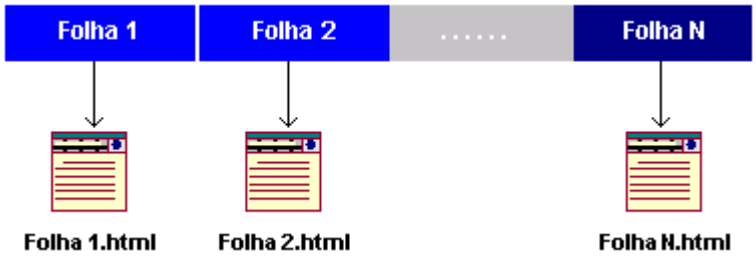
O componente "Menu" permite o acesso às funcionalidades do sistema em qualquer estado que se esteja. Esse componente não é encontrado diretamente na linguagem HTML. Pode ser obtido através da combinação dessa linguagem com *JavaScript*, fazendo com que o mapeamento seja realizado de forma equivalente. O mapeamento funcional pode ser realizado através de *links*, com uma "barra" superior contendo os itens do menu superior, que representam os contextos do sistema e, uma coluna à esquerda representando os itens desse menu, sub-menus, que são as funcionalidades dos contextos. O Quadro 1 mostra o mapeamento equivalente e funcional para o componente "Menu".

Quadro 1 - Componente Menu

<table>
<tr><th colspan="2">Mapeamento do Componente</th></tr>
<tr><td rowspan="6">Menu</td><td>Legado</td></tr>
<tr><td> 
</td></tr>
<tr><td>Mapeamento Equivalente</td></tr>
<tr><td> 
</td></tr>
<tr><td>Mapeamento Funcional</td></tr>
<tr><td> 
</td></tr>
</table>

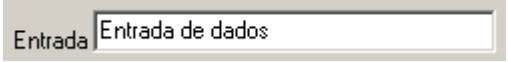
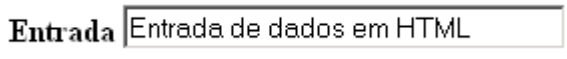
O Quadro 2 mostra o mapeamento para o componente "Página de Controle", que abriga um conjunto de formulários cujo acesso ocorre através de suas abas superiores, "`Folha1, Folha 2 ou Folha N`". Esse componente não existe diretamente na linguagem HTML, sendo necessário fazer seu mapeamento funcional, que consiste, basicamente, na criação de uma página central contendo *links* para as páginas secundárias, que representam os formulários que podem ser acessados, simulando a funcionalidade do componente.

Quadro 2 - Componente Página de Controle

Mapeamento do Componente	
Página de Controle	**Legado**
	
	Mapeamento Equivalente
	Não é possível
	Mapeamento Funcional
	

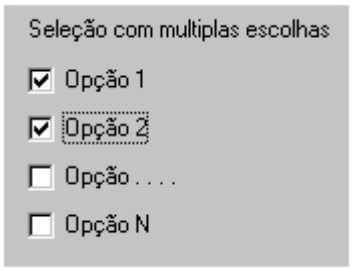
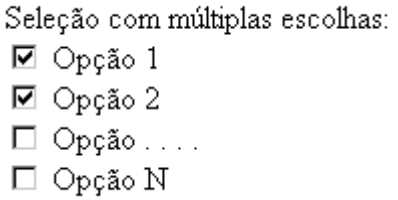
O componente "Entrada de Dados" é responsável pela entrada de dados da interface para o sistema, sendo que o seu mapeamento ocorre de maneira equivalente como mostra o Quadro 3.

Quadro 3 - Componente de entrada de dados

Mapeamento do Componente	
Entrada de Dados	**Legado**
	
	Mapeamento Equivalente
	
	Mapeamento Funcional
	Não é necessário

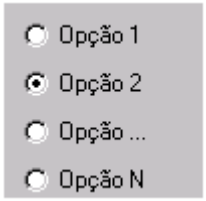
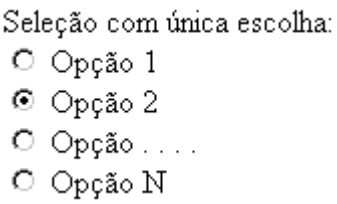
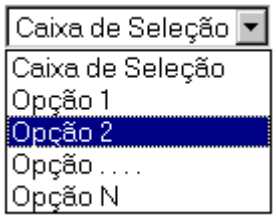
O componente "Múltipla Escolha" também é responsável pela entrada de dados no sistema, oferecendo ao usuário um conjunto de opções, permitindo que sejam selecionadas algumas delas. O mapeamento ocorre de forma equivalente como mostra o Quadro 4.

Quadro 4 - Componente de múltipla escolha

Mapeamento do Componente "Múltipla escolha"		
Legado	**Mapeamento Equivalente**	**Mapeamento Funcional**
		Não é necessário

O componente "Única Escolha" também é responsável pela entrada de dados no sistema, oferecendo ao usuário um conjunto de opções, porém permitindo a seleção de apenas uma. O mapeamento para esse componente pode ser realizado de maneira equivalente ou funcional como mostra o Quadro 5. Para a forma funcional, faz-se a substituição pelo componente "Caixa de Seleção", que também permite a escolha de uma opção, mas com *layout* mais enxuto que o anterior. Recomenda-se o mapeamento funcional quando o número de opções para escolha for elevado, ou seja, que acarrete problemas na apresentação do *layout* da interface do usuário, expondo as informações fora do campo visual da tela.

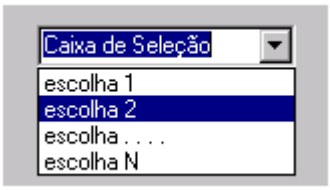
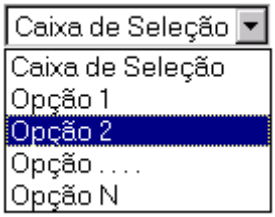
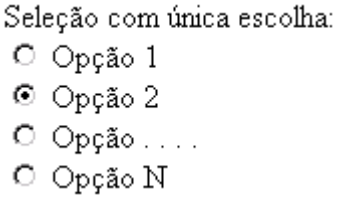
Quadro 5 - Componente de única escolha

Mapeamento do Componente "Única escolha"		
Legado	**Mapeamento Equivalente**	**Mapeamento Funcional**
		

O componente "Caixa de Seleção" pode ser mapeado de maneira equivalente e funcional como mostra o Quadro 6. A funcional ocorre quando a funcionalidade desse componente é análoga ao de "Única escolha". Sua utilização é recomendada quando o número

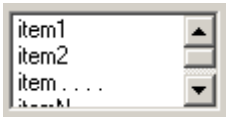
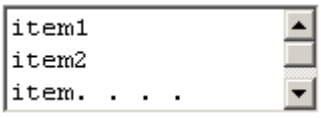
de opções para escolha não interfere na composição do *layout* da interface, oferecendo melhor representatividade das informações, deixando-as mais explicitas.

Quadro 6 - Componente de caixa de seleção

Mapeamento do Componente "Caixa de Seleção"		
Legado	**Mapeamento Equivalente**	**Mapeamento Funcional**
		

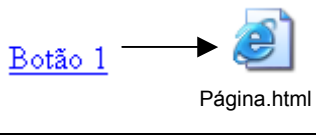
O componente "Lista de Itens" permite a entrada de dados da interface para o sistema através de uma grande quantidade de informação. O mapeamento desse componente é realizado de forma equivalente como mostra o Quadro 7.

Quadro 7 - Componente lista de itens

Mapeamento do Componente "Lista de Itens"		
Legado	**Mapeamento Equivalente**	**Mapeamento Funcional**
		Não é necessário

O Componente "Botão de Ação" permite a execução das funcionalidades do sistema ou a chamada para outras interfaces. Seu mapeamento pode ocorrer de forma equivalente, com a substituição imediata por um botão para execução de uma funcionalidade do sistema, ou de forma funcional, utilizando *hyperlinks* para fazer chamadas para outras interfaces, que representam as páginas, como mostra o Quadro 8.

Quadro 8 - Componente botão de ação

Mapeamento do Componente "Botão de Ação"		
Legado	**Mapeamento Equivalente**	**Mapeamento Funcional**
		

O componente "Apresentação de Relatório" é responsável pela exibição de um conjunto de informações de uma entidade/objeto. O mapeamento desse componente é realizado de forma funcional, organizando as informações do sistema legado em uma tabela, que são exibidas na página, como mostra o Quadro 9.

Quadro 9 - Componente de apresentação de relatório

<table>
<tr><th colspan="2">Mapeamento do Componente</th></tr>
<tr><td rowspan="6">Apresentação de relatório</td><td>Legado</td></tr>
<tr><td>Listagem da entidade 1

Código XX Nome nome

Endereço endereço 1
Telefone telefone 1
CPF cpf 1
Data de Nascimento data nascimento 1
Estado Civil estado civil 1
Escolaridade escolaridade 1</td></tr>
<tr><td>Mapeamento Equivalente</td></tr>
<tr><td>Não é possível</td></tr>
<tr><td>Mapeamento Funcional</td></tr>
<tr><td><table><tr><td>Código</td><td>código 1</td></tr><tr><td>Nome</td><td>nome 1</td></tr><tr><td>Endereço</td><td>endereço 1</td></tr><tr><td>Telefone</td><td>telefone 1</td></tr><tr><td>CPF</td><td>cpf 1</td></tr><tr><td>DataNascimento</td><td>data de nascimento 1</td></tr><tr><td>Estado Civil</td><td>estado civil 1</td></tr><tr><td>Escolaridade</td><td>escolaridade 1</td></tr></table></td></tr>
</table>

O componente "Navegação" permite a manipulação de registros no banco de dados por meio das operações de inserção, atualização e exclusão e, ainda, a de navegação por pelos registros, acessando o primeiro, o anterior, o próximo e o último. Esse componente pode ser mapeado apenas de forma funcional por meio de um conjunto de botões, em que cada um deles executa uma das funcionalidades mencionadas, como mostra o Quadro 10.

Além dos componentes apresentados nesta seção, existem outros que podem ser encontrados tanto na interface legada quanto na nova interface *Web*, mas não requerem mapeamento, como por exemplo, os textos, figuras, gráficos, entre outros. No entanto, é

recomendado um estudo para migrá-los para uma interface *Web*, para que não haja comprometimento com a adequação do novo *layout*.

Quadro 10 - Componente de navegação

<table>
<tr><td colspan="2">Mapeamento do Componente</td></tr>
<tr><td rowspan="6">Navegação</td><td>Legado</td></tr>
<tr><td>  </td></tr>
<tr><td>Mapeamento Equivalente</td></tr>
<tr><td>Não é possível</td></tr>
<tr><td>Mapeamento Funcional</td></tr>
<tr><td> 
</td></tr>
</table>

Esta seção apresentou o mapeamento dos componentes mais utilizados em interfaces *Delphi* para *Web*, fornecendo um conjunto de diretrizes que auxiliam na substituição e escolha de cada para a confecção da nova interface do usuário. A próxima seção apresenta as etapas referentes à construção de interfaces voltadas para *Web*, abrangendo desde o desenvolvimento até a avaliação da mesma como produto final.

3.2.5.3 - Diretrizes para o Desenvolvimento da Nova Interface

As diretrizes que apóiam a migração de uma interface legada desenvolvida no ambiente *Delphi* para *Web* envolve a seguinte documentação: a) o modelo de interação GOMS, com a seqüência de passos para execução das tarefas; b) o modelo navegacional, que fornece o caminho a ser percorrido na interface para execução das tarefas; e c) o mapeamento dos componentes de interface, apresenta as formas de mapeamento que apóiam a migração dos componentes de interfaces *Delphi* para *Web*. Além disso, toma-se por base, informalmente, os critérios de usabilidade para o desenvolvimento de interfaces *Web* propostos pelos autores **(NIELSEN, 1994, 2000) (NIELSEN & TAHIR, 2000)**.

A Figura 30 mostra a organização das diretrizes para o desenvolvimento da interface *Web* sendo que as elipses pontilhadas contêm os documentos já obtidos nas etapas anteriores

do processo.

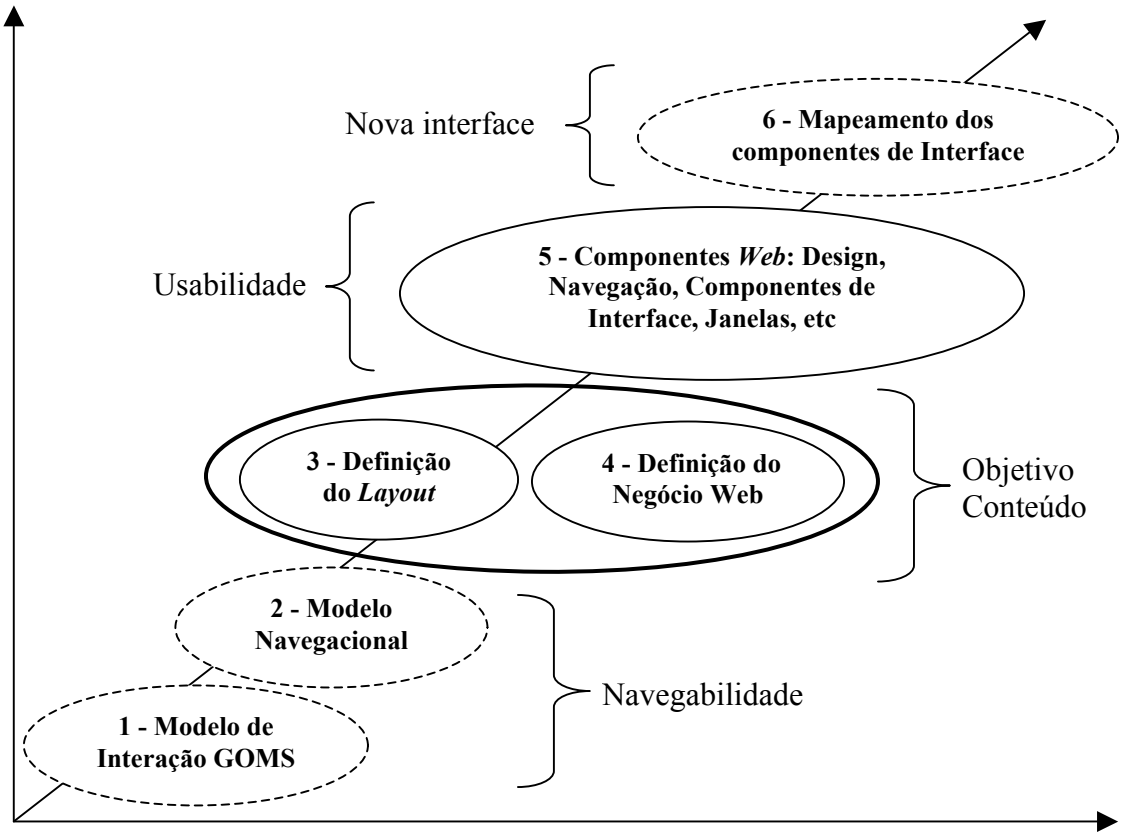


Figura 30 - Organização das etapas de desenvolvimento da interface *Web*

A construção das interfaces *Web* é iniciada com o Modelo de Interação GOMS, que fornece ao engenheiro de software uma representação teórica da interface legada, bem como, uma seqüência de passos para a execução das tarefas, sua especificação e sua organização estrutural. Essas informações combinadas com o Modelo Navegacional formam um conjunto preliminar de interfaces com nova navegabilidade para execução das tarefas.

Devido à relação existente entre a Definição do *Layout* com a do Negócio *Web* e vice-versa elas atuam em paralelo. A Definição do *Layout* apresenta uma breve descrição dos *layouts* para interfaces *Web*. Segundo **BERNARD & LARSEN (2003a)** deve haver equilíbrio entre o espaço e o conteúdo em uma página, ou seja, as informações devem ser inseridas em uma janela, que representa o campo visual do usuário. Os *layouts* mais comumente encontrados nas páginas são:

- **Justificado:** é considerado o mais flexível, pois o conteúdo da página se ajusta ao tamanho da janela, deixando as informações expostas de forma horizontal ou vertical conforme a largura da tela;
- **Centralizado:** apresenta o conteúdo das informações de forma centralizada na

tela, nesse caso a área disponível para inserção da informação é fixa em 640 *pixels* na horizontal, ficando livre a vertical;

- **Alinhado à esquerda:** apresenta as informações alinhadas à esquerda em uma área de 770 *pixels*, havendo necessidade de navegação horizontal dependendo da resolução gráfica do usuário, ou seja, a informação deixa de ser exibida na área da janela;

Estudos realizados sobre a escolha do *layout* apresentam o seguinte resultado de aceitação: a) justificado: 65% dos usuários, b) centralizado: 30% dos usuários e c) esquerda: 5% dos usuários. Esses resultados comprovam que a distribuição das informações em uma interface *Web* deve ser enquadrada ao campo visual do usuário **(BERNARD & LARSEN, 2003a)**.

A Definição do negócio *Web* apresenta um conjunto de diretrizes que auxiliam quanto à exposição dos objetivos de um *site*, transmissão de suas informações e criação de seu conteúdo. Essas diretrizes estão organizadas conforme seu contexto de atuação no desenvolvimento **(NIELSEN & TAHIR, 2000) (NIELSEN, 2000)**:

Contexto: Informando o objetivo do *site*.

Objetivo: permitir ao usuário identificar os serviços ou os produtos oferecidos pelo *site*.

Diretrizes:

- Exibir o nome da empresa ou logotipo em tamanho razoável e em local de destaque. Recomenda-se, normalmente, o canto esquerdo superior da página;
- Incluir um *slogan* resumindo explicitamente o quê o *site* ou a empresa faz, como uma marca registrada;
- Mostrar a importância do *site*, enfatizando as diferenças com os concorrentes;
- Definir uma página do *site* como sendo a principal, ou seja, o ponto de referência para as demais;

Contexto: Criação do conteúdo.

Objetivo: manter o interesse do usuário pelo *site*. Muitas vezes tem-se um espaço pequeno para um número elevado de informações.

Diretrizes:

- Usar nomes de seções centrados no usuário conforme a importância dessas para ele e não para a empresa;
- Evitar o conteúdo redundante, pois causa desinteresse no usuário, que está sempre

a procura de novidades;

- Evitar o uso de frases eruditas ou dialetos de *marketing*, pois fazem com que os usuários tenham que descobrir seu significado;
- Recomenda-se o uso de letras iniciais maiúsculas e outros padrões de estilo com consistência na edição do conteúdo;
- Não rotular uma área nitidamente definida da página, pois se essa for suficientemente auto-explicativa, causa redundância e desperdício de informação;
- Utilizar espaços não-separáveis somente nas palavras que precisam permanecer juntas para serem vista e entendidas;
- Utilizar o discurso imperativo em tarefas obrigatórias, como por exemplo, o preenchimento de um campo, "Insira o nome da cidade";
- Explicar o significado de abreviações imediatamente após a primeira ocorrência, para que usuário não fique sem saber seu significado ou que tenha que procurá-lo;
- Utilizar raramente todas as letras de uma palavra em maiúscula, pois isso causa problemas com legibilidade;
- Evite o uso de espaços em branco ou pontos para dar ênfase;
- Utilizar alto contraste entre fundo da página e o texto nela presente;

Contexto: Revelar o conteúdo por meio de exemplos.

Objetivo: auxiliar na transmissão da funcionalidade do *site*, apresentando os serviços oferecidos e seu conteúdo. Isso é muito importante, pois a informação visual é facilmente absorvida pelo usuário do que leitura de um texto.

Diretrizes:

- Usar exemplos ao em vez de descrevê-lo, pois uma imagem pode conter mais informação do que palavras, além de ser mais agradável aos usuários.
- Para cada exemplo, deve-se ter um *link* que o apresente de forma detalhada;
- Diferenciar os *links* que detalham o exemplo dos que trazem informações gerais sobre ele.

Os Componentes *Web* referem-se às diretrizes para: a construção de *links*, a navegabilidade do *site*, a composição do *layout*, a inserção de elementos gráficos e animações, entre outras. As diretrizes por contexto de atuação sugeridas por **(NIELSEN & TAHIR, 2000) (NIELSEN, 2000)** são:

Contexto: *Links*.

Objetivo: permitir a navegação pelas páginas de um *site*, ou pelas informações contidas nelas.

Diretrizes:

- Diferenciar os *links* do texto da página, tornando-os de fácil visualização;
- Evitar o uso de expressões genéricas, como "Clique aqui", para o nome de um *link*, usar nomes significativos a ele;
- Não utilizar *links* genéricos, como "Mais...", no final de uma lista de itens, deve-se informar aos usuários o que eles podem ter acesso ao clicar nele;
- Utilizar *links* coloridos para indicar os que foram visitados e os que ainda não. Recomenda-se o azul para os não visitados e uma outra cor que indique que o *link* foi acessado;
- Não utilizar a palavra "*Link*" para indicar sua presença em uma página, sua existência deve ser marcada pelo sublinhado e pela cor azul;
- Os *links* devem ser auto-explicativos, devem informar aos usuários o que irá acontecer ao acessá-lo, seja para direcioná-lo outra página, para ter acesso a um arquivo de vídeo, e-mail, etc;
- Os *links* contêm um conjunto de informações que podem ser apresentados em quatro formas básicas. Em (a) são apresentados os *links* distribuídos pelo texto da página, cujo acesso ocorre através da leitura seqüencial do texto ou pela busca rápida de forma visual. Em (b) os *links* estão organizados abaixo do texto, cujo acesso ocorre através da busca até o final do documento. Em (c) os *links* organizados em uma coluna ao lado esquerdo superior do texto, cujo acesso ocorre através do acesso a esse local. Finalmente, em (d) os *links* são organizados ao lado esquerdo e apresentados conforme seu surgimento no texto. Foi realizado um experimento utilizando cada uma das formas apresentadas, com o objetivo de mostrar qual delas permite a busca pela informação de maneira rápida e objetiva. O resultado desse experimento mostra que 50% dos usuários preferem os *links* misturados com o texto, forma (a); 34,5% preferem os *links* de forma correspondente ao texto, mas não inserido nele, forma (d); 14,5% preferem alinhados no topo a esquerda, forma (c). Esses resultados mostram que as duas primeiras formas são mais indicadas na construção de páginas **(BERNARD et al, 2003b)**.

Contexto: Navegação.

Objetivo: deve permitir o acesso às informações pelo usuário de forma intuitiva, sem que ele

precise clicar em algum item para saber seu significado.

Diretrizes:

- Alocar a área de navegação em local de destaque, preferencialmente ao lado do corpo principal da página;
- Agrupar os itens semelhantes na área de navegação;
- Não disponibilizar diversas áreas de navegação para o mesmo tipo de *links*;
- Evitar a redundância de *links*, ou seja, ao clicá-lo permaneça no mesmo lugar;
- Não usar termos técnicos para a seção de navegação, pois dificultam entendimento do usuário;
- A navegabilidade deve evitar questionamentos por parte do usuário como "Onde estou?", "Aonde posso ir?", entre outros, pois fazem com que abandonem o *site*, devido às incertezas geradas.

Contexto: Pesquisa.

Objetivo: trata-se de um dos elementos mais importante na construção de páginas. Por meio das pesquisas os usuários podem localizar informações de forma rápida, agilizando as execuções das tarefas, sendo recomendada em *site* de grande porte.

Diretrizes:

- Disponibilizar uma caixa de entrada para consulta na página ao invés de fornecer um *link* que leve uma área/página de pesquisa;
- Apenas use um botão de pesquisa "Busca/Pesquisa" ao lado da caixa de entrada;
- A pesquisa deve estar centrada no *site* inteiro ou deixar claro para o usuário que ela será feita na *Web*.

Contexto: Ferramentas e atalhos para tarefas:

Objetivo: fornecer uma maneira de agilizar a execução de tarefas e os recursos do *site*, atendendo rapidamente as necessidades do usuário.

Diretrizes:

- Oferecer acesso direto às tarefas de alta prioridade na página;
- Não ofereça funções na página como as já são oferecidas pelo navegador;

Contexto: Gráfico e animação

Objetivo: ilustrar o conteúdo de uma página com recursos visuais, pois ajudam na usabilidade da interface.

Diretrizes:

- A utilização de gráficos deve real e não para fins decorativos;
- Os gráficos e fotos devem ser rotulados se não representarem adequadamente o contexto;
- Editar fotos e diagramas adequadamente, segundo o tamanho de exibição;
- Evite o uso de gráfico como marca d'água, ou seja, imagem como plano de fundo e texto sobreposto;
- Não utilize animação em elementos críticos da página, como logotipo, *slogan*, ou título principal;
- Disponibilize para os usuários uma versão animada do *site*, mas não a defina como predefinida;
- Limite o tamanho dos arquivos das figuras, pois arquivos como "BMP" são grandes e retardam o carregamento do *site*, recomenda-se o uso de arquivos "GIF", "JPEG", entre outros.
- Recomenda-se a utilização moderada desses recursos, pois quando aplicados demasiadamente podem causar retardos em relação ao tempo de *download*, que é fator crítico em sistemas voltados para *Web*.

Contexto: *Design* gráfico.

Objetivo: prender a atenção do usuário referente aos elementos mais importantes da página.

Diretrizes:

- Limitar os estilos de fonte e outros atributos de formatação de texto, como tamanhos, cores, etc, pois um texto com *design* muito pesado pode desviar a atenção do usuário quanto seu significado;
- Utilizar contraste entre o texto e o plano de fundo, garantindo a legibilidade do que está escrito;
- O conteúdo principal de uma página deve estar enquadrado a uma tela com tamanho predominante a 800x600;
- Utilização do *layout* justificado para que a página possa se ajustar a várias resoluções de tela;

Contexto: Componentes da interface com o usuário.

Objetivos: representam os meus, lista de seleção, caixas de texto, entre outros, que são utilizados na interação com o usuário para execução das tarefas.

Diretrizes:

- Nunca utilize componentes de interface como parte da tela em que as pessoas não deverão clicar;
- Evite o uso de caixa de texto no topo da página para funcionalidades do *site*, pois esse local é destinado aos recursos de pesquisa;

Contexto: Janelas *pop-up* e páginas intermediárias.

Objetivo: não são necessárias em todo *site*, pois impedem que os usuários acessem o conteúdo imediatamente. Podem ser úteis no primeiro acesso, mas no terceiro ou trigésimo podem ser indesejados, podendo confundir alguns usuários.

Diretrizes:

- Ao digitar o endereço do *site*, o usuário deve ser encaminhado a sua página principal;
- Evite o uso de janelas *pop-up* para a execução das tarefas;

Contexto: Comunicação dos problemas técnicos e tratamento de emergências.

Objetivo: informar ao usuário o motivo da paralisação do serviço.

Diretrizes:

- Informar aos usuários de forma clara e objetiva na página caso o *site* ou parte dele fique paralisado;
- Ter um plano de emergência, para lidar com conteúdo crítico do *site*, no caso de uma paralisação.

Contexto: Recarregamento e atualização da página.

Objetivo: minimizar o impacto da transição de conteúdo da página, pois a experiência do usuário deve ser preservada.

Diretrizes:

- Não atualizar automaticamente a página para adicionar atualizações para os usuários;
- As atualizações devem ser aplicadas somente ao conteúdo modificado, por exemplo, como atualização de notícias.

Essas diretrizes contribuem com a construção da nova interface *Web*, fornecendo um guia para o atendimento de requisitos que aumentam sua usabilidade. A seguir são abordados

os aspectos referentes ao desenvolvimento da nova interface do usuário.

3.2.5.4 - A CONSTRUÇÃO DA NOVA INTERFACE

O desenvolvimento de interfaces para *Web* deve ser apoiado pelas diretrizes apresentadas na seção anterior, que auxiliam na melhoria de sua usabilidade. Além disso, esta etapa sugere que a construção da nova interface do usuário deve ser realizada de maneira gradativa, cujo desenvolvimento e avaliação é realizada com base nas diretrizes apresentadas e nos critérios de usabilidade de **NIELSEN (1994)**. Sugere-se, como modelo de ciclo de vida, o de prototipação evolutivo, como mostra a Figura 31. Esse modelo tem início com o modelo de interação GOMS, que fornece ao engenheiro de software sua especificação e sua organização estrutural. Em seguida, elabora-se o primeiro protótipo da interface, com base nos documentos dos grupos 3, 4, 5 e 6 da Figura 30, que posteriormente é refinado com a aplicação das heurísticas de **NIELSEN (1994)**, que são aplicadas de maneira informal ao método de Avaliação Heurística.

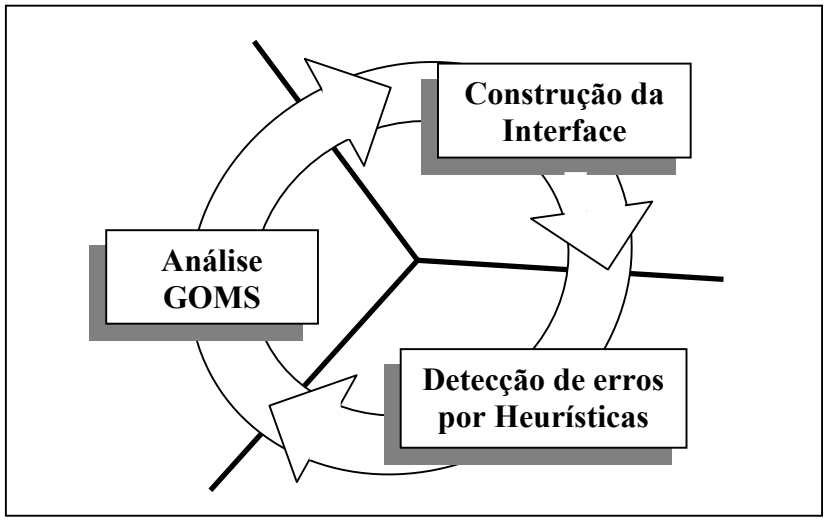


Figura 31 - Ciclo de vida para o desenvolvimento de interface *Web*

A construção da nova interface do usuário está organizada em seis etapas não seqüenciais, que abrangem a desde a definição da marca da empresa até a construção das páginas que fazem parte do *site*. Essas etapas estão organizadas sem uma seqüência explicitada para sua aplicação. O desenvolvimento de páginas pode ter inicio com (a)Definição do logo do *site*, que representa a marca da empresa ou do *site*. Em seguida,

pode-se (b)Definir seu título, cujo objetivo é apresentar a sua finalidade. Essas etapas podem ser apoiadas pela etapa (c)Definição das Cores, pois as utilizadas na definição do logo e do título são re-utilizadas no restante do desenvolvimento do *site*. O logo, o título e as cores utilizadas no desenvolvimento desses artefatos representam a primeira visão da interface *Web*, ficando para o engenheiro de interface a criatividade no desenvolvimento.

Com a definição da primeira visão do *site*, inicia-se a confecção da parte estrutural, que corresponde a sua (d) Navegabilidade e seu (e) *Layout*. A construção do *layout* da interface está relacionada com a apresentação de todo conteúdo de um *site*, sendo utilizados os componentes da interface legada que foram mapeados para *Web*, além da aplicação de critérios de usabilidade que auxiliam na exposição desses componentes pelo *layout* da página.

A segunda visão do *site* corresponde à criação das páginas que representam os contextos e suas funcionalidades. Para isso, o engenheiro de interface tem como base o modelo navegacional para extrair a navegabilidade entre as funcionalidades. A estrutura sugerida corresponde à criação de um menu superior contendo cada um dos contextos identificados e, dentro de cada contexto, suas funcionalidades são organizadas em menu à esquerda. Essa organização mostra que as funcionalidades de um contexto podem ser acessadas somente quanto se estiver dentro dele, caracterizando a nova organização do sistema.

Após a definição do *layout* do *site*, segunda visão do *site*, o engenheiro de interface pode iniciar a (f) Construção das páginas, que representam as funcionalidades dos contextos. Para isso, o engenheiro de interface é apoiado pelas diretrizes dos grupos 5 e 6, que fazem recomendações quanto ao *design* e a inserção de elementos gráficos na interface *Web*. O desenvolvimento de seu *layout* é feito com base nos componentes da interface legada que foram migrados para *Web*. Com isso, gerando a terceira visão do *site* e finalizando a etapa de confecção da nova interface do usuário.

Com a migração da interface do usuário para *Web* e com o empacotamento do sistema legado é iniciada a etapa de integração de ambos como mostra próxima seção.

3.2.6 - PROJETO DO SISTEMA

Essa etapa corresponde à integração entre o código legado empacotado e a nova interface do usuário. Para isso, o engenheiro de software deve escolher um recurso de *middleware* para a integração do sistema empacotado e da nova interface do usuário como

mostra a Figura 32. A arquitetura de software apresentada mostra a nova organização do sistema em camadas: interface, *middleware* e empacotamento. Nessa última, estão as funcionalidades do sistema legado estão decompostas em componentes de software, que auxiliam em uma tarefa de manutenção, aumentando o grau de certeza e segurança do engenheiro de software em qual funcionalidade está atuando.

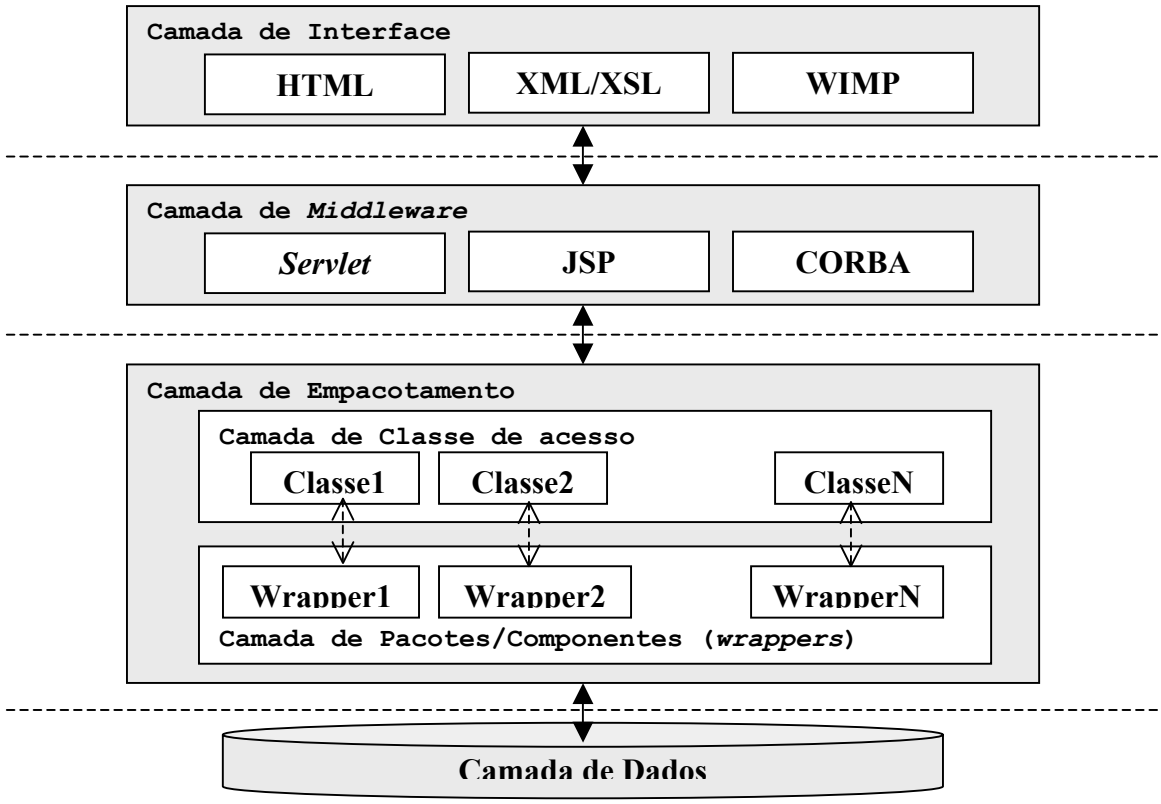


Figura 32 - Arquitetura de software do sistema

Após a definição do recurso de *middleware* para implementação do sistema, cabe ao engenheiro de software modelar a integração entre o sistema empacotado, a nova interface do usuário e o recurso de *middleware* responsável pela comunicação de ambos. Para modelar essa integração, os diagramas de caso de uso e de seqüência obtidos na seção 3.2.1 são utilizados, pois esses fornecem as seqüências de passos para execução das funcionalidades extraídas do sistema legado, que devem preservadas com o empacotamento do sistema. Os modelos desenvolvidos nesta etapa são confeccionados na ferramenta *Rational Rose* **(RATIONAL, 2003).** Esse tipo de diagrama é confeccionado para todas as funcionalidades do sistema.

Os diagramas de projeto mostram o caminho completo para sua execução das funcionalidades do sistema. A partir da página principal do *sit*e, um contexto do sistema é

chamado, que por sua vez chama uma página que representa uma das funcionalidades requisitadas desse contexto. Essa página contém um *servlet* responsável pela aquisição dos dados da interface do usuário até o encaminhamento deles para camada de acesso ao empacotamento, que recebe, trata os dados e os envia para o sistema legado para serem processados.

3.2.7 - ACOPLAMENTO DO SISTEMA

Com a definição da modelagem do sistema, o engenheiro de software inicia a última etapa do processo de reengenharia de interface, que corresponde à implementação do recurso de *middleware* responsável pela integração entre o sistema empacotado e a nova interface do usuário. Para isso, o engenheiro de software utiliza os diagramas de projeto obtidos na etapa anterior.

3.3 - CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentado o processo para reengenharia de interface, que é composto por um conjunto de diretrizes, que auxilia os engenheiros de software na condução da reengenharia de interface de um sistema. Esse processo está centrado em sistema legados desenvolvidos em *Delphi*, tanto no paradigma procedimental quanto no orientado a objetos, ambos com interfaces gráficas.

A proposta de empacotamento do processo de reengenharia de interface é decompor o código legado em pacotes, componentes de software, que podem ser disponibilizados tanto para uma aplicação distribuída, quanto para *Web* ou, ainda, para um sistema local. Dessa forma, esse processo mostra-se mais abrangente que as soluções para empacotamento encontradas na literatura, pois essas estão direcionadas a um domínio particular.

No que se refere ao desenvolvimento da interface do usuário, o processo de reengenharia de interface oferece um conjunto de diretrizes que apóia desde o estudo da interface legada até a construção da nova interface *Web* com base nos critérios de usabilidade, em que atuam tanto no desenvolvimento quanto na avaliação do produto final.

Outra característica inerente do processo de reengenharia de interface está relacionada aos sistemas alvo que se pode atender, pois esses sistemas foram desenvolvidos no ambiente

Delphi e possuem interface gráfica, geralmente com usabilidade baixa. Isso caracteriza que a modernidade do recurso empregado no desenvolvimento de um sistema não está diretamente relacionada à satisfação do usuário quanto ao seu uso.

CAPITULO 4

ESTUDO DE CASO

4.1 - CONSIDERAÇÕES INICIAIS

Este capítulo apresenta um estudo de caso em que são aplicadas as diretrizes do processo de reengenharia de interface apresentado no capítulo 3. Para instanciar esse processo escolheu-se um sistema legado procedimental desenvolvido no ambiente *Delphi* sem características orientadas a objetos, cujas funcionalidades são empacotadas, formando os componentes de software e, uma nova interface do usuário para *Web* é desenvolvida, que é acoplada a eles, formando um novo sistema. A organização deste capítulo é a seguinte: na seção 4.2 apresenta-se a descrição do sistema utilizado como estudo de caso; na seção 4.3 é apresentada a aplicação das diretrizes do processo de reengenharia de interface para esse sistema e na seção 4.4 as considerações finais.

4.2 - DESCRIÇÃO DO SISTEMA

O sistema utilizado como estudo de caso, com código fonte obtido pela Internet, para aplicação das diretrizes do processo de reengenharia de interface é o de "CONTROLE DE ESTOQUE DE UMA PAPELARIA". Esse sistema foi desenvolvido no ambiente *Delphi* segundo o paradigma procedimental e sem o comprometimento com a organização de código. Seu objetivo é a manutenção do estoque de produtos de uma papelaria ou de qualquer outro

segmento de empresas que necessitem desse tipo de controle. Os contextos do sistema e suas principais funcionalidades são:

- **Clientes:** são cadastrados, atualizados, excluídos e consultados. A impressão de relatórios contendo todos os dados de um determinado cliente pode ser solicitada;
- **Fornecedores:** o gerenciamento dos fornecedores de produtos da papelaria é realizado pela inclusão, atualização, exclusão e consulta, além da listagem completa desses;
- **Funcionários:** são responsáveis pela realização de pedidos a fornecedores e da venda de produtos para clientes. O gerenciamento de funcionários da papelaria é realizado por meio da inclusão, atualização, exclusão e consulta;
- **Pedido e Item Pedido:** cada pedido é composto por um ou vários itens pedidos, que correspondem aos produtos. O gerenciamento é realizado com a inclusão ou a exclusão de pedidos e itens pedidos a um fornecedor;
- **Venda e Item:** cada venda é composta de um ou mais itens de um produto. O gerenciamento de venda para os clientes é realizado com a inclusão e exclusão de vendas e seus itens, além da impressão do relatório de vendas;
- **Produtos:** são incluídos, atualizados, excluídos e consultados, além da impressão de relatórios contendo todos os dados de um produto;

Na próxima seção é apresentada a aplicação das diretrizes do processo de reengenharia de interface para o sistema do estudo de caso.

4.3 - PROCESSO DE REENGENHARIA DE INTERFACE

Para a realização do processo de reengenharia de interface há necessidade do sistema em funcionamento e de seu código fonte, que corresponde à lógica do sistema. Como mostrado no capítulo 3, este processo é composto por várias etapas, sendo cada uma delas tratadas nas seções 4.3.1 a 4.3.7.

4.3.1 - ESTUDO DA INTERFACE E DA LÓGICA DO SISTEMA

As diretrizes do método GOMS são aplicadas quando se executa o sistema para que se

obtenha os seguintes documentos: o modelo de interação GOMS, a lista de erros de usabilidade da interface, a lista de erros de funcionalidade do sistema e a lista parcial dos casos de uso do sistema.

Para exemplificar a extração do modelo de interação GOMS será utilizada a tarefa "`cadastrar clientes`". A Figura 33 mostra a tela inicial do sistema legado. A opção "`Cadastrar`" do menu superior oferece acesso um conjunto de opções, dentre elas, "`Clientes`", Figura 34, que ativa o formulário "`Cadastro de Clientes`" composto por um conjunto de botões, como mostra a Figura 35. Para cadastrar um novo cliente deve-se clicar no botão "`Novo Cliente`", apresentando o formulário com campos de entrada para um cliente, como mostra a Figura 36. Ao terminar de preencher esses campos deve-se clicar no botão "`Salvar`", para a gravação dos dados do cliente no banco de dados, finalizando a tarefa.

Figura 33 - Tela inicial do sistema

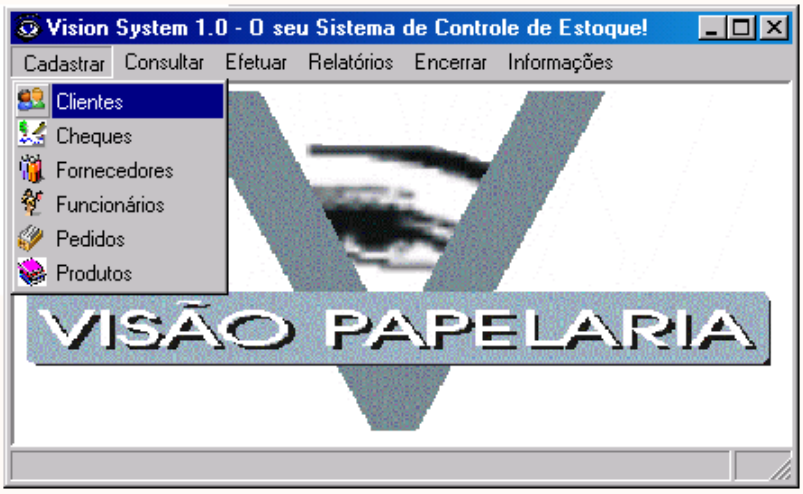


Figura 34 - Acesso ao menu cadastrar

Figura 35 - Formulário para cadastro de clientes

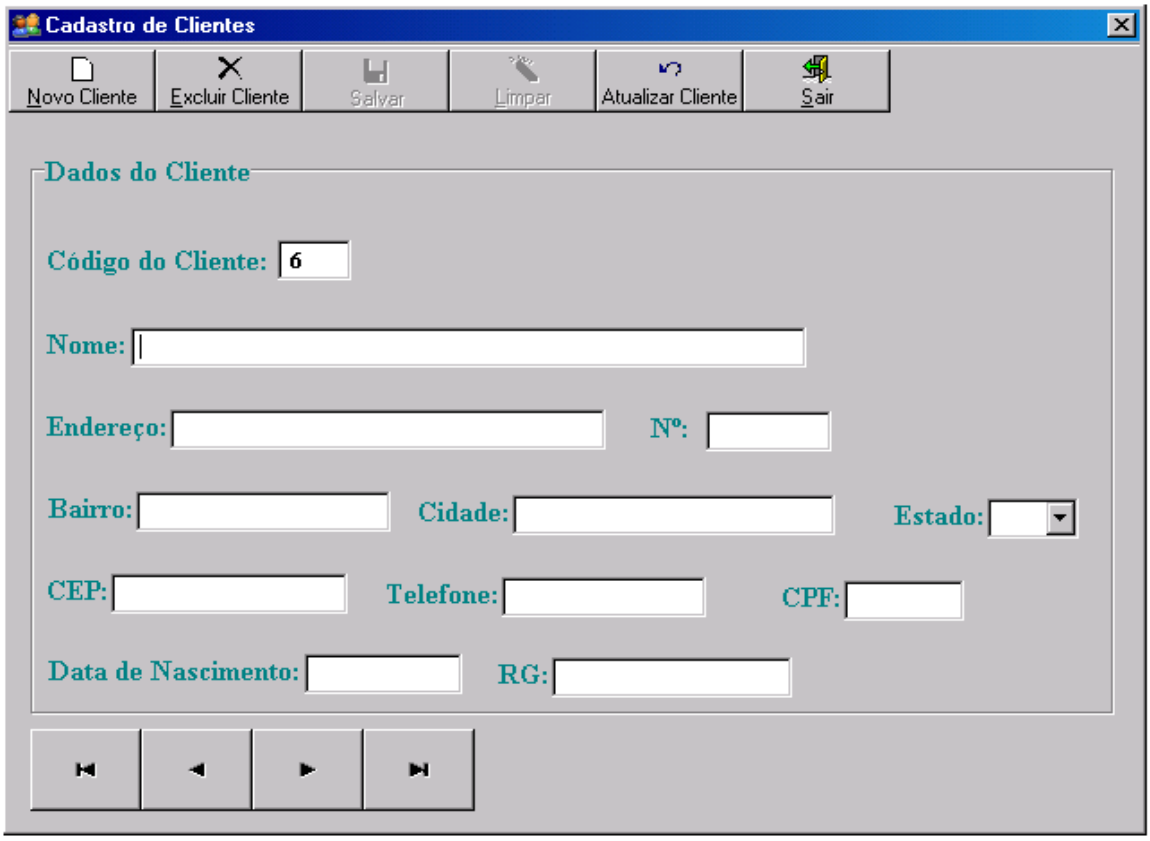


Figura 36 - Tela de cadastro de cliente

O acompanhamento da execução da tarefa "`cadastro de cliente`" nas interfaces apresentadas na Figuras 33, 34, 35 e 36 permite a geração do modelo de interação GOMS, como mostra a Figura 37. Esse modelo representa o caminho a ser percorrido na interface para o cadastro de um cliente.

Durante a execução do sistema, para se obter o modelo de interação, podem ser detectados erros de usabilidade da interface, pois todas as suas funcionalidades estão sendo percorridas na interface. A Figura 38 apresenta a lista (parcial) de erros de usabilidade encontrados quando da execução do cadastro de clientes. Essa lista é a utilizada para a melhoria da usabilidade na etapa de confecção da nova interface do usuário. De modo análogo são identificados os erros de usabilidade para todas as interfaces que compõe o sistema.

Método para a meta : Cadastrar Cliente
Passo 1: Executa a meta : Selecionar Menu;
Passo 2: Executa a meta : Selecionar Item Menu;
Passo 3: Executa a meta : Selecionar Botão;
Passo 4: Executa a meta : Preencher Campos;
Passo 5: Executa a meta : Selecionar Botão;
Passo 6: Retorna com a atividade realizada;

Método para a meta: Selecionar Menu
Passo 1: Focalizar o menu desejado.
Passo 2: Mover o mouse sobre o menu desejado.
Passo 3: Clicar nesse item.
Passo 4: Retorna com a atividade realizada.

Método para a meta: Selecionar item de Menu
Passo 1: Focalizar o item desejado de menu.
Passo 2: Mover o mouse até o item de menu desejado.
Passo 3: Clicar no item focalizado.
Passo 4: Retorna com a tarefa realizada.

Método para a meta: Selecionar Botão
Passo 1: Focalizar o botão desejado.
Passo 2: Mover o mouse até esse botão.
Passo 3: Clicar nesse botão.
Passo 4: Retorna com a tarefa realizada.

Regra de Seleção para Preencher Campos
Se o campo for de entrada de texto então utilize a meta: **Preencher Campos de Texto.**
Se o campo for uma caixa de seleção então utilize a meta: **Preencher Caixa de Seleção.**

Método para a meta: Preencher Campos de Texto
Passo 1: Preencher os campos do formulário com o teclado;
Passo 2: Executa a meta : **Movimentar campos;**
Passo 3: Retorna com a meta realizada;

Método para a meta: Preencher Caixa de Seleção
Passo 1: Movimentar o mouse até esse campo.
Passo 2: Clique nesse item.
Passo 3: Focalizar a opção desejada.
Passo 4: Clicar nessa opção.
Passo 5: Retorna com a meta realizada.

Regras de Seleção para Movimentar Campos
Se for para navegar entre campos de entrada então utilize a meta: **Movimentar com teclas de atalho;**
Se for para selecionar outro item de preenchimento então utilize a meta: **Movimentar com mouse;**

Método para a meta: Movimentar com teclas de atalho
Passo 1: Pressionar a tecla (tab) para movimentar para o próximo campo;
Passo 2: Retorna com a atividade realizada;

Método para a meta: Movimentar com mouse
Passo 1: Mover o mouse sobre o item de preenchimento desejado;
Passo 2: Clicar nesse item;
Passo 3: Perceber o item em foco;
Passo 4: Retorna com a atividade realizada;

Figura 37 - Modelo de interação GOMS para cadastro de usuário

Lista de erros de interface		
Projeto: Sistema de Livraria		**Data de Criação:** 12/04/2003
Número do erro	**Heurística**	**Descrição do erro encontrado**
1	1, 2	Os nomes dos menus na tela principal não são sugestivos ao contexto do usuário.
2	2	O nome do formulário "cadastro de cliente" não retrata a funcionalidade, pois possui um conjunto de operações como exclusão, alteração, entre outras.
3	3, 4	A interface do usuário exige carga cognitiva elevada para execução de um cadastro de cliente
4	6	O sistema não possui saídas imediatas para outras partes, fazendo com que o usuário encerre a operação vigente, para que possa mudar para uma outra;
5	9	Os campos, Cep, RG e CPF, permitem a entrada de dados não numéricos, não existe prevenção a erros.
6	8	O sistema não possui nenhuma mensagem de retorno ao usuário para a realização da operação de cadastro.

Figura 38 - Lista de erros de usabilidade para cadastro de clientes (parcial)

Outro documento que pode ser obtido com a extração do modelo de interação GOMS é a lista de erros de funcionalidade do sistema. Essa lista não é apresentada nesta etapa, pois o sistema em estudo não apresentou erros funcionais em sua execução.

Com a definição do estudo da usabilidade da interface, inicia-se a extração da lógica do sistema, que também é apoiada pela execução do sistema e complementada com a análise de seu código fonte. Para isso é confeccionada uma lista contendo os possíveis casos de uso, como mostra a Figura 39, que apresenta, parcialmente, as funcionalidades referentes ao gerenciamento de clientes.

Lista de casos de uso		
Projeto: Sistema de Livraria	**Data de Criação:** 12/04/2003	
Número	**Funcionalidade**	**Caso de Uso**
1	Cadastro de clientes	CadastrarCliente
2	Alteração dos dados do cliente	AlterarCliente
3	Consulta de um cliente	ConsultarCliente
4	Excluir um cliente	ExcluirCliente
5	Relatório de clientes	RelatorioCliente

Figura 39 - Lista de caso de uso para "Clientes" (parcial)

A Figura 40 apresenta o diagrama de caso de uso para as funcionalidades que fazem o

gerenciamento de clientes.

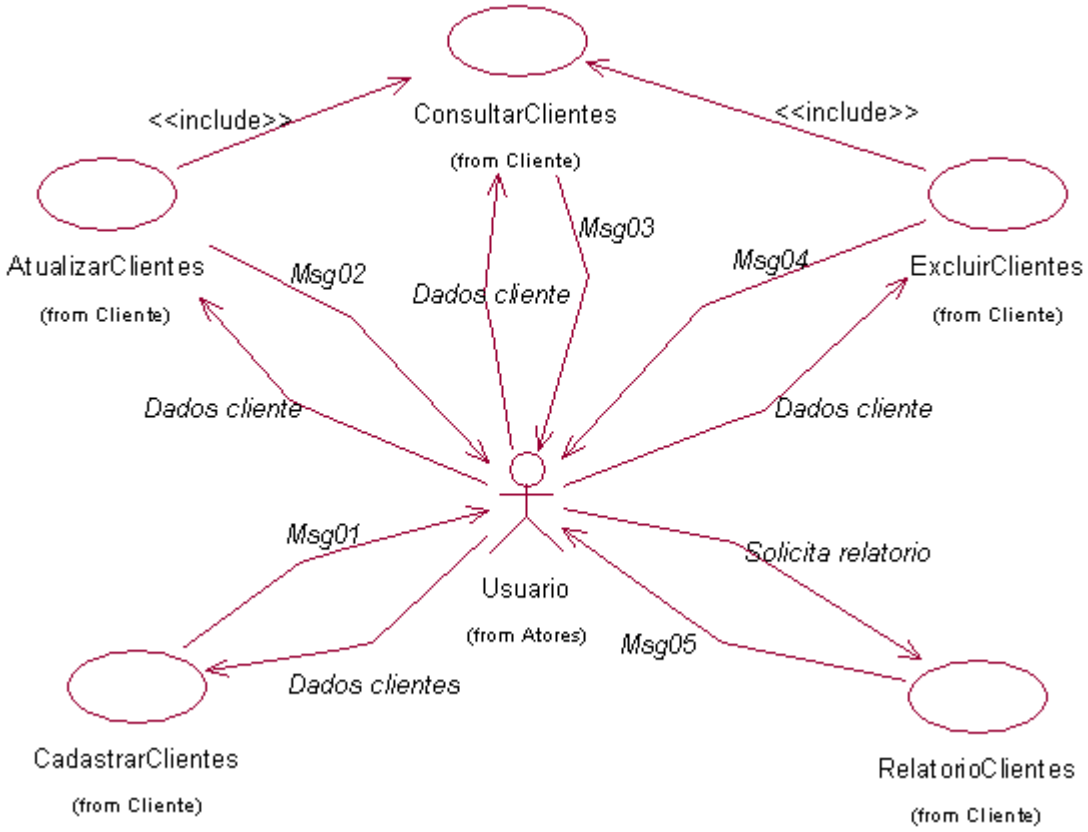


Figura 40 - Diagrama de caso de uso completo para clientes

A Figura 41 mostra o diagrama de seqüência para a funcionalidade "`cadastro de cliente`". Esses diagramas devem ser confeccionados para todos os casos de uso identificados.

4.3.2 - ORGANIZAÇÃO DO SISTEMA

Esta seção apresenta à segmentação do sistema, que corresponde na separação da interface do usuário da lógica da aplicação.

A Figura 42 mostra o formulário "`Cadastro de Clientes`", que utiliza um conjunto de componentes de interface *Delphi*, tais como: os botões de ação "`Novo Cliente`" e "`Excluir Cliente`"; os campos de entrada para "`Nome`" e "`Endereço`", entre outros. Esses componentes são utilizados no mapeamento da nova interface do usuário e também na extração da lógica do sistema. Por exemplo, os botões de ação têm uma ou mais

funcionalidades do sistema, que podem ser obtidas através de um duplo clique. O código *Delphi* correspondente ao botão "`Novo Cliente`" é exibido na Figura 43.

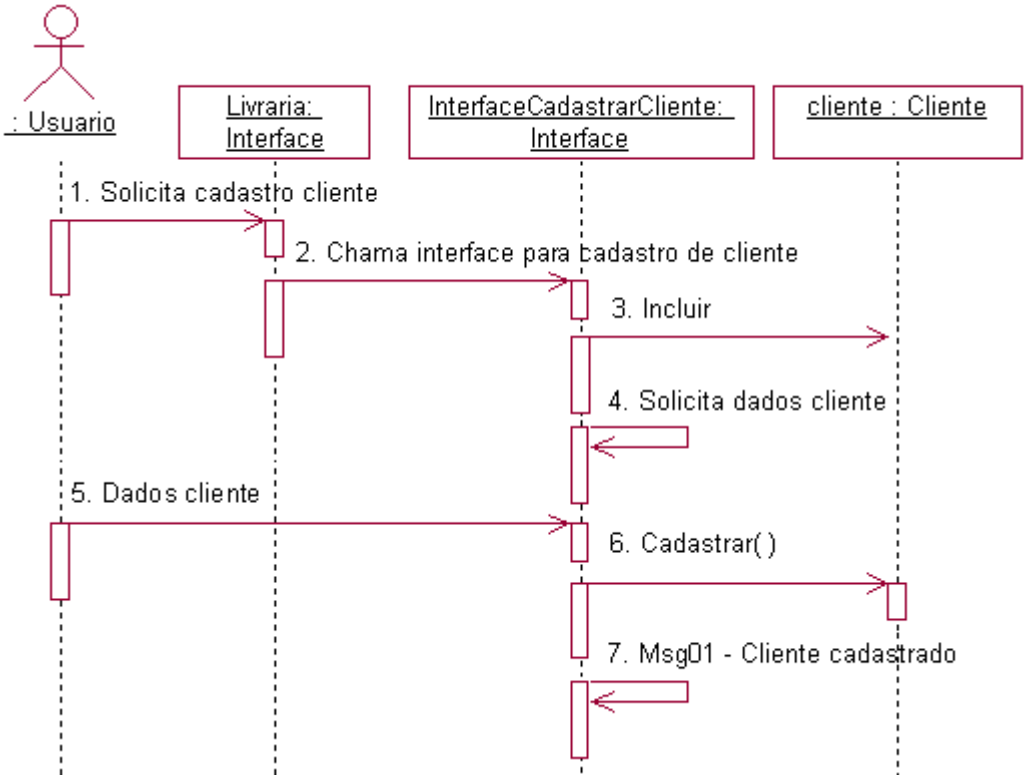


Figura 41 - Diagrama de seqüência para cadastro de cliente

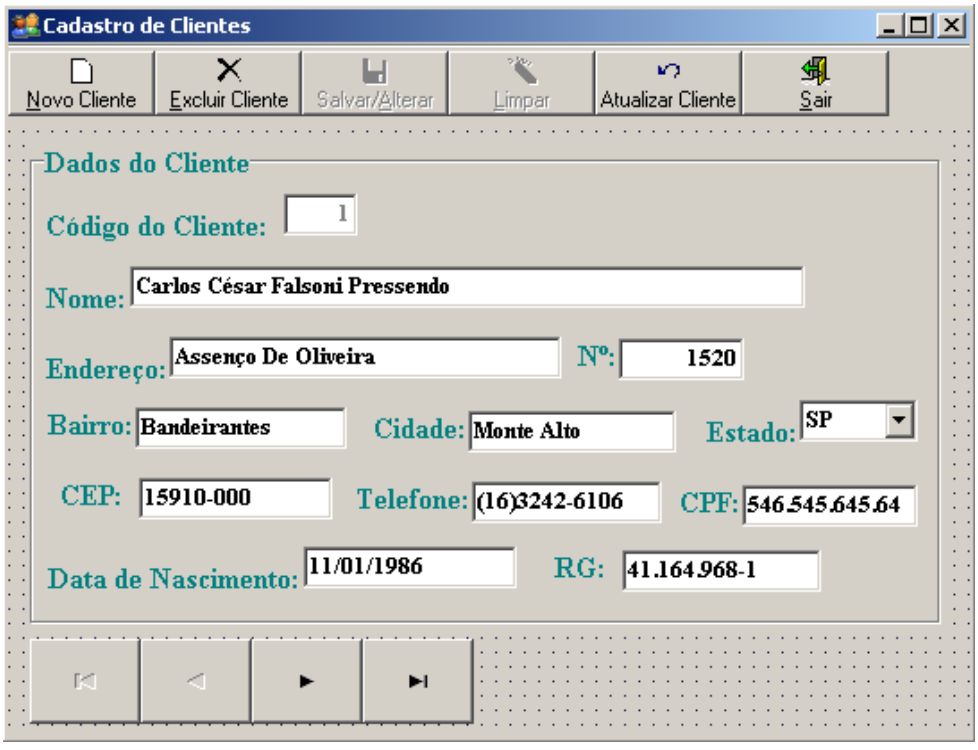


Figura 42 - Formulário para cadastro de clientes

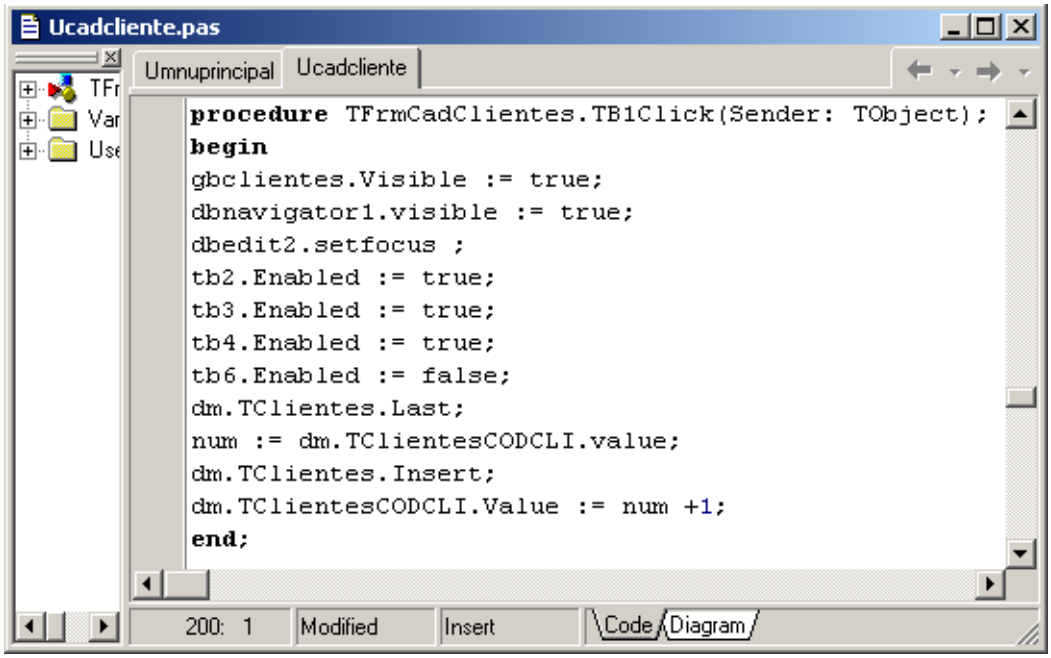


Figura 43 - Código fonte associando ao botão "Novo Cliente"

A lista (parcial) de componentes de interface para o formulário "`Cadastro de Clientes`" é exibida na Figura 44. Pode-se notar nessa figura a ausência de componentes que fazem a ligação do formulário com o banco de dados como "`DataSources`" e "`Tables`". Esses componentes estão organizados em um "`DataModule`" e são apresentados mais adiante com a análise do código fonte.

Lista de componentes			
Projeto: Sistema de Livraria		**Data de Criação:** 10/04/2003	
Número	**Nome: Tipo**	**Classificação**	**Informações contidas**
1	`DbNome:DBEdit`	Interface	Campo de entrada da interface para o banco de dados.
2	`LabNome: Label`	Interface	Exibe texto no formulário, não possui qualquer acesso ao banco de dados.
3	`DbEstado: DBComboBox`	Interface	Campo de entrada da interface para o banco de dados.
4	`TB1:TToolButton`	Interface	Execução da lógica do sistema contém toda programação funcional do sistema.

Figura 44 - Lista de componentes de interface (parcial)

A partir do código da Figura 45, marcado por (a) e (b), que refere-se, respectivamente, a inserção e a exclusão de clientes, extraem-se as funcionalidades candidatas ao empacotamento como mostra a Figura 47. As informações pertinentes aos componentes de banco de dados também encontram-se organizados nessa Figura, sendo seu código fonte apresentado na Figura 46, através de um arquivo "`.dfm`".

```
procedure TFrmCadClientes.TB1Click(Sender: TObject);
begin
      gbclientes.Visible := true;
      dbnavigator1.visible := true;
      dbedit2.setfocus ;
      tb2.Enabled := true;
      tb3.Enabled := true;
      tb4.Enabled := true;
      tb6.Enabled := false;
      dm.TClientes.Last;
      num := dm.TClientesCODCLI.value;
      dm.TClientes.Insert;
      dm.TClientesCODCLI.Value := num +1;
end;
procedure TFrmCadClientes.TB2Click(Sender: TObject);
begin
      if messagedlg ('Excluir Registro?', mtconfirmation,
                     [mbyes, mbNo],0) = mrNO
            then abort;
      dm.TClientes.Delete;
end;
```

(a)

(b)

Figura 45 - Código fonte do arquivo ".pas"

```
object TClientes: TTable
    Active = True
    DatabaseName = 'Vision'
    TableName = 'CLIENTES'
    Left = 15
    Top = 20
    object TClientesCODCLI: TIntegerField
      FieldName = 'CODCLI'
      Required = True
    end
    ......
    object TClientesCODCLI: TIntegerField
      FieldName = 'CODCLI'
      Required = True
    end
......
end
```

Figura 46 - Código fonte do arquivo ".dfm"

Lista de procedimentos candidatos ao empacotamento				
Projeto: Sistema de Livraria			**Data de Criação:** 10/04/2003	
Número	**Contexto**	**Procedimento**	**Descrição**	**Código Bruto**
1	Clientes	`TB2Click`	Esse procedimento faz a exclusão de um cliente.	`Procedure TFrmCadClientes.TB2Click (Sender: TObject); begin if messagedlg ('Excluir Registro?', mtconfirmation, [mbyes, mbNo],0) = mrNO then abort; dm.TClientes.Delete; end;`
2	Clientes	Informações de Banco de Dados	Esse trecho oferece informações da tabela clientes	`object TClientes: TTable Active = True DatabaseName = 'Vision' TableName = 'CLIENTES'`
3	Clientes	`TB1Click()`	Esse procedimento prepara a tabela cliente para o modo de inserção, posicionando a tabela par o último registro e adicionando ao novo cliente que será inserido.	`Procedure TFrmCadClientes.TB1Click (Sender: TObject); begin gbclientes.Visible := true; dbnavigator1.visible := true; dbedit2.setfocus ; tb2.Enabled := true; tb3.Enabled := true; tb4.Enabled := true; tb6.Enabled := false; dm.TClientes.Last; num := dm.TClientesCODCLI.value; dm.TClientes.Insert; dm.TClientesCODCLI.Value := num +1; end;`

Figura 47 - Lista de procedimentos candidatos ao empacotamento (parcial)

A Figura 48 mostra os procedimentos candidatos à implementação, que tratam dos campos de entrada da interface para o banco de dados e vice-versa. Essas funcionalidades são criadas devido à impossibilidade de empacotamento de componentes *Delphi* associados a eventos manipulados na interface do usuário.

A seção seguinte apresenta a identificação dos contextos do sistema, que consiste na organização do código legado por cada um deles.

Lista de procedimentos candidatos à implementação			
Projeto: Sistema de Livraria		**Data de Criação:** 10/04/2003	
Número	**Contexto**	**Procedimento**	**Descrição**
1	Clientes	SetaDadosCliente	Esse procedimento faz atribuição dos campos da nova interface do usuário para cada campo da tabela Cliente
2	Clientes	RetornaDadosCliente	Esse procedimento é encarregado de retornar uma tupla ou várias tuplas da entidade Cliente para a interface do usuário.

Figura 48 - Lista de procedimentos candidatos à implementação (parcial)

4.3.3 - IDENTIFICAÇÃO DOS CONTEXTOS DO SISTEMA

A identificação dos contextos do sistema e o agrupamento das funcionalidades em cada um deles, preparam para a visão orientada a objetos, com cada um dos contextos identificados sendo transformados em uma pseudo-classes na etapa de empacotamento. A Figura 49 mostra os menus existentes com as funcionalidades de clientes em vários contextos do sistema.

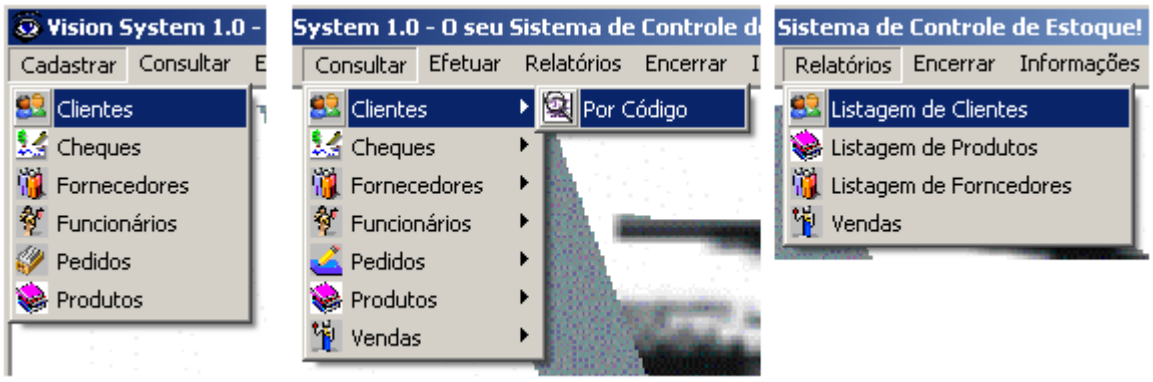


Figura 49 - Funcionalidades de clientes

Para reorganizar as funcionalidades de "`Cliente`" é confeccionada uma lista de contextos e métodos como mostra a Figura 50, considerando todos os contextos identificados no sistema.

Ao término dessa etapa tem-se um conjunto de documentos, com as informações sobre a usabilidade da interface e a organização das funcionalidades do código legado em contextos,

permitindo que sejam iniciadas etapas de composição dos pacotes e de confecção da nova interface do usuário.

Lista de contextos e métodos			
Projeto: Sistema de Livraria		**Data de Criação:** 10/04/2003	
Número	**Contexto**	**Procedimento/ Função**	**Descrição**
1	Clientes	**procedure** TFrmCadClientes. TB1Click(Sender: TObject);	Esse procedimento prepara para a inserção de um novo cliente no banco de dados.
2	Clientes	**Procedure** TFrmCadClientes. TB2Click(Sender: TObject);	Esse procedimento faz a exclusão de um cliente.
3	Clientes	**Procedure** TFrmCadClientes. TB4Click(Sender: TObject);	Esse procedimento permite a inserção do cliente no banco de dados.
4	Clientes	**Procedure** TFrmCadClientes. TB6Click(Sender: TObject);	Esse procedimento é utilizado para chamar o formulário que faz a atualização dos dados de um cliente.
5	Clientes	**procedure** TFrmConsultaClienteC. BtConsultarClick(Sender: TObject);	Esse procedimento faz a consulta de um cliente pelo seu código.
6	Clientes	**procedure** TFrmPrincipal. Listagensde ClientesClick(Sender: TObject);	Esse procedimento exibe a listagem de clientes

Figura 50 - Lista de contextos e métodos para clientes

4.3.4 - COMPOSIÇÃO DOS PACOTES

O empacotamento do sistema legado consiste no isolamento das funcionalidades legadas em pseudo-classes, que são envolvidas por uma "camada de software", cujo objetivo é fornecer acesso à nova interface do usuário

A obtenção do modelo de dados é apoiada pelo ambiente do banco de dados, que permite a extração das entidades do MER, de seus atributos e tipos e de suas chaves primárias e estrangeiras. Neste caso o banco de dados *Interbase* é utilizado e exibido na Figura 51 para

a entidade "`Clientes`". A Figura 52 mostra a lista parcial de tabelas utilizadas no estudo de caso.

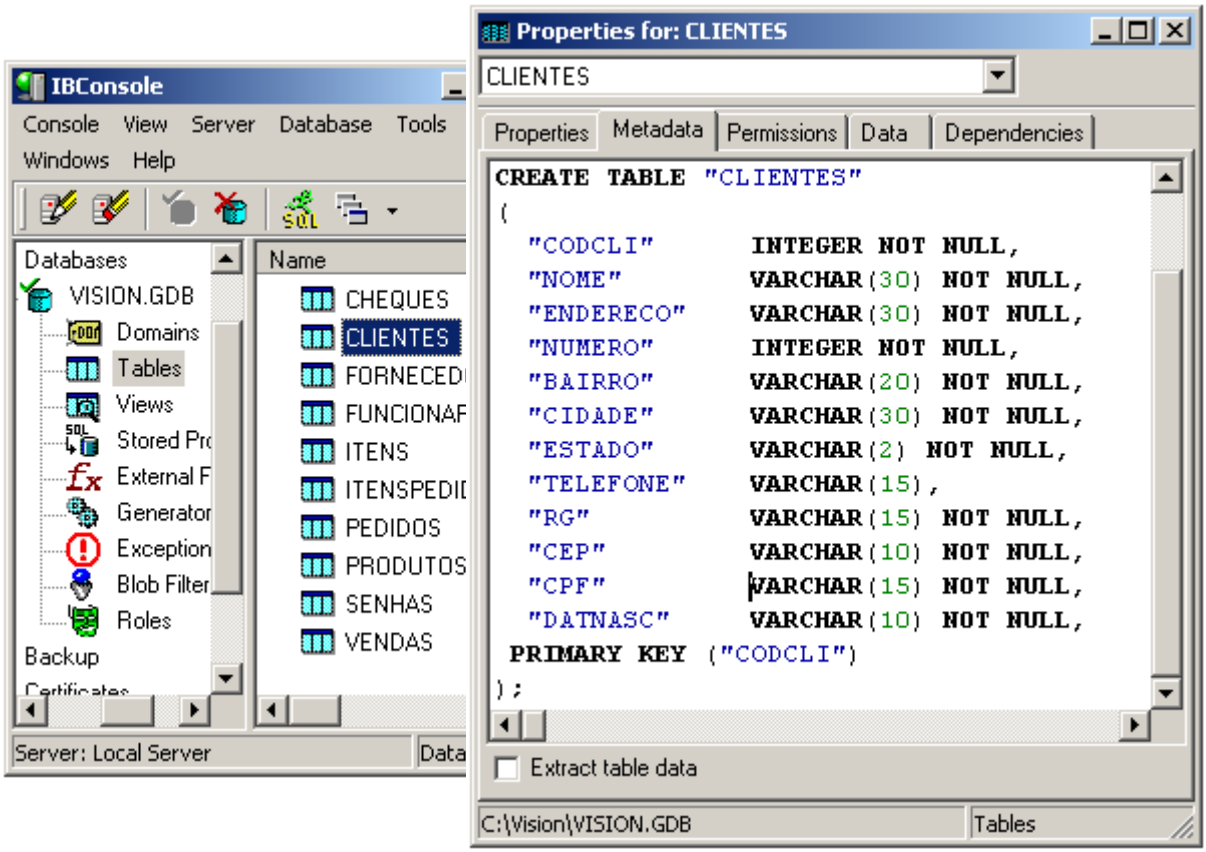


Figura 51 - Ambiente do InterBase

Com esses dados o MER é construído. A Figura 53 exibe o MER parcial do sistema que está sendo utilizado como estudo de caso, que trata da realização de um pedido que está sendo realizado a um fornecedor.

Lista de tabelas				
Projeto: Sistema de Livraria		**Data de Criação:** 10/04/2003		
Tipo de Dados: Tabelas				
Tabela	**Atributos**	**Tipo**	**Chave Primária**	**Chave Estrangeira**
CLIENTES	CODCLI	INTEGER	*	
	NOME	VARCHAR(30)		
	ENDERECO	VARCHAR(30)		
	NUMERO	INTEGER		
	BAIRRO	VARCHAR(20)		
	CIDADE	VARCHAR(30)		
	ESTADO	VARCHAR(2)		
	TELEFONE	VARCHAR(15)		
	RG	VARCHAR(15)		
	CEP	VARCHAR(10)		
	CPF	VARCHAR(15)		
	DATNASC	VARCHAR(10)		
FORNECEDORES	CODFORNECEDOR	INTEGER	*	
	NOME	VARCHAR(30)		
	TELEFONE	VARCHAR(15)		
	CIDADE	VARCHAR(30)		
	RESPONSAVEL	VARCHAR(30)		
	CNPJ	VARCHAR(15)		
ITENS	CODVENDA	INTEGER	*	
	CODPRODUTO	INTEGER		*
	QTD	INTEGER		
	PRECOPROD	DOUBLE		
	TOTALITENS	DOUBLE		
PEDIDOS	CODPEDIDO	INTEGER	*	
	VALOR	DOUBLE		
	CODFORNECEDOR	INTEGER		*
	DATA	VARCHAR		
	FPAG	VARCHAR		
PRODUTOS	CODPRODUTO	INTEGER	*	
	QTD	INTEGER		
	DESCRICAO	VARCHAR(30)		
	PRECOC	DOUBLE		
	PRECOV	DOUBLE		
	CODFORNECEDOR	INTEGER		
	LUCRO	DOUBLE		
	QTDMIN	INTEGER		

Figura 52 - Lista de entidade e atributos (parcial)

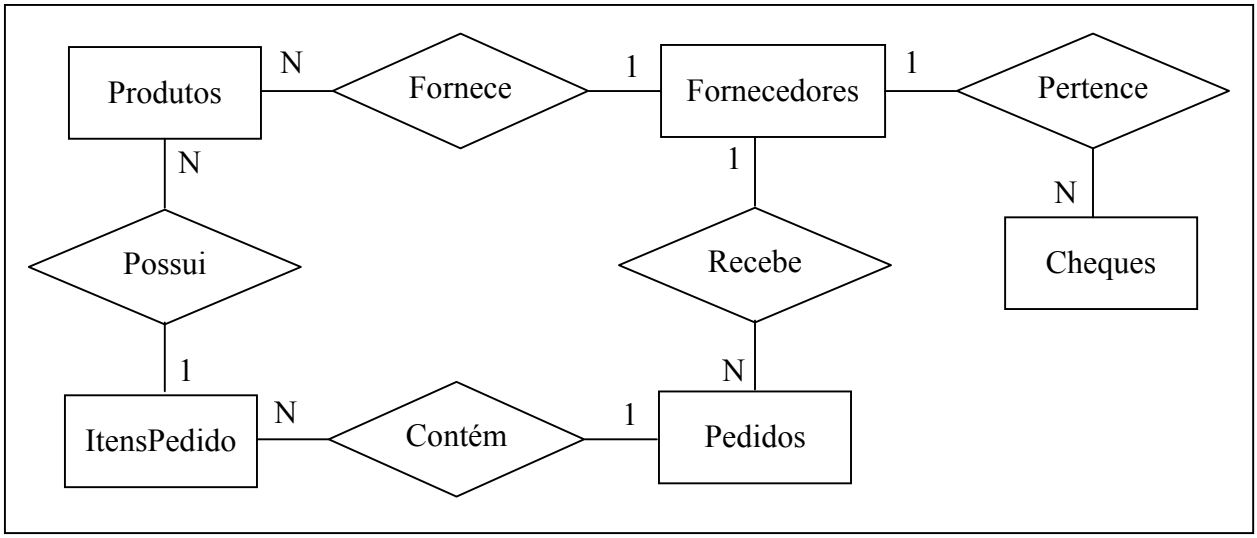
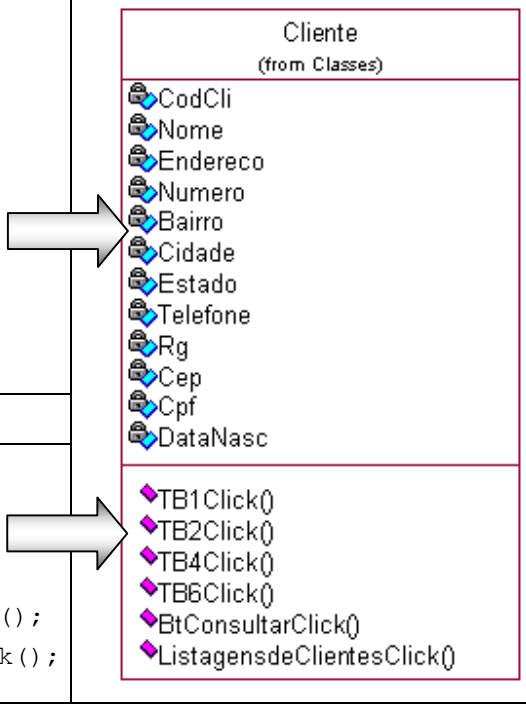


Figura 53 - Modelo Entidade Relacionamento para o contexto de pedido (parcial)

Após a modelagem do MER inicia-se o processo de formação das pseudo-classes do sistema, ou seja, a visão orientada a objetos. Essas pseudo-classes são geradas através da "fusão" de cada contexto identificado na etapa anterior com cada entidade do MER. A Figura 54 mostra a formação da pseudo-classe "`Cliente`" através da união dos atributos da entidade "`Clientes`" como as funcionalidades do contexto de mesmo nome.

Fusão da Entidade "`Clientes`" com o contexto "`Clientes`"	
Atributos da entidade "`Clientes`"	**Pseudo-classe "`Cliente`"**
CODCLI, NOME, ENDERECO, NUMERO, BAIRRO, CIDADE, ESTADO, TELEFONE, RG, CEP, CPF, DATNASC	
Funcionalidades do contexto "`Cliente`"	

`TFrmCadClientes.TB1Click();`
`TFrmCadClientes.TB2Click();`
`TFrmCadClientes.TB4Click();`
`TFrmCadClientes.TB6Click();`
`TFrmConsultaClienteC.BtConsultarClick();`
`TFrmPrincipal.ListagensdeClientesClick();`	

Figura 54 - Formação da pseudo-classe "Cliente"

Dessa forma, é criada a primeira visão orientada a objetos do sistema, construindo-se o MASA, Modelo de Análise do Sistema Atual. Esse modelo é muito importante para os sistemas procedimentais, pois representa o nível de implementação do empacotamento. A Figura 55 apresenta o MASA, equivalente ao MER apresentado na Figura 53, mostrando apenas os nomes das classes, pois seus atributos e métodos são detalhados na etapa de implementação do empacotamento.

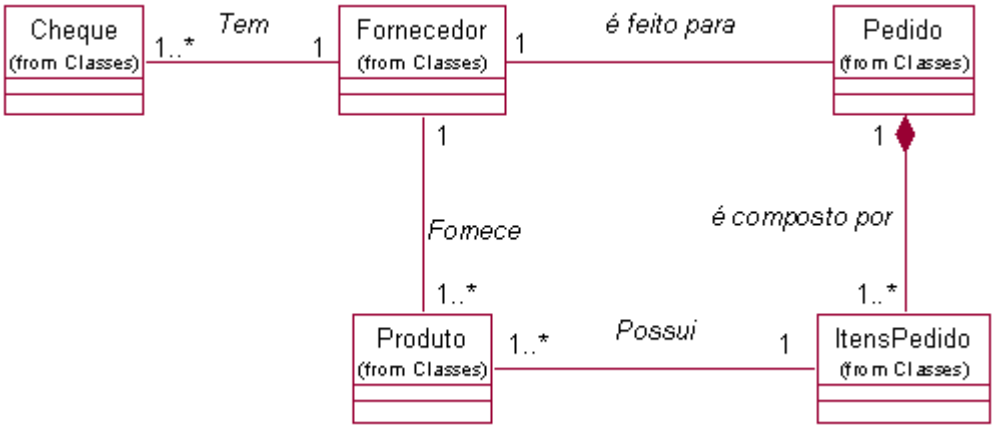


Figura 55 - MASA, Modelo de Análise do Sistema Atual (parcial)

A próxima etapa corresponde a criação do MAS, Modelo de Análise do Sistema, Figura 56, que representa o refinamento do MASA, sendo obtido com base nos padrões da família de padrões FaPRE/OO **(RECCHIA, 2002)**. Esse modelo corresponde a visão orientada a objetos do sistema, representando a lógica do empacotamento.

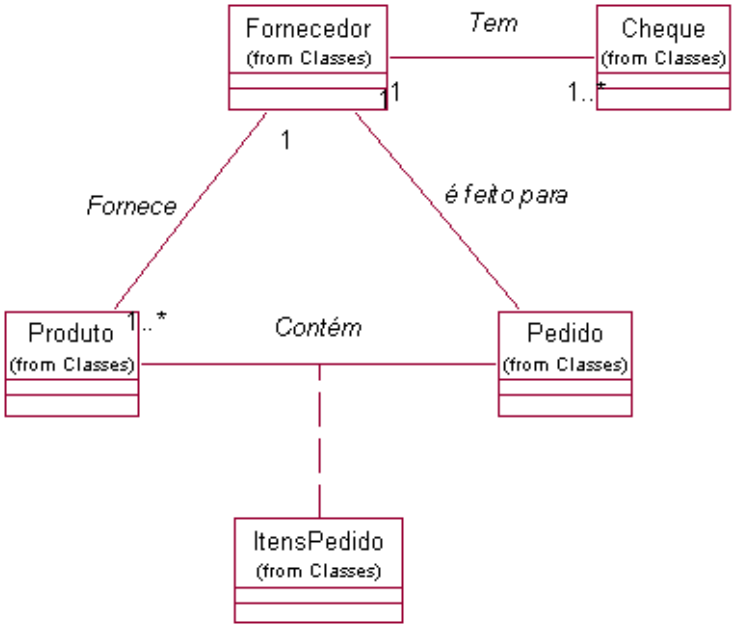


Figura 56 - Modelo de Análise do sistema

A seguir inicia-se a etapa de especificação do empacotamento, que corresponde à reorganização do código legado, agrupando as funcionalidades conforme os modelos de análise obtidos. Os diagramas de caso de uso e de seqüência obtidos do sistema legado na etapa de estudo da interface e da lógica do sistema, que apresentam as funcionalidades envolvidas para execução de uma tarefa são usados. A Figura 57 apresenta o diagrama de caso de uso para cadastro de clientes e a Figura 58 o correspondente diagrama de seqüência.

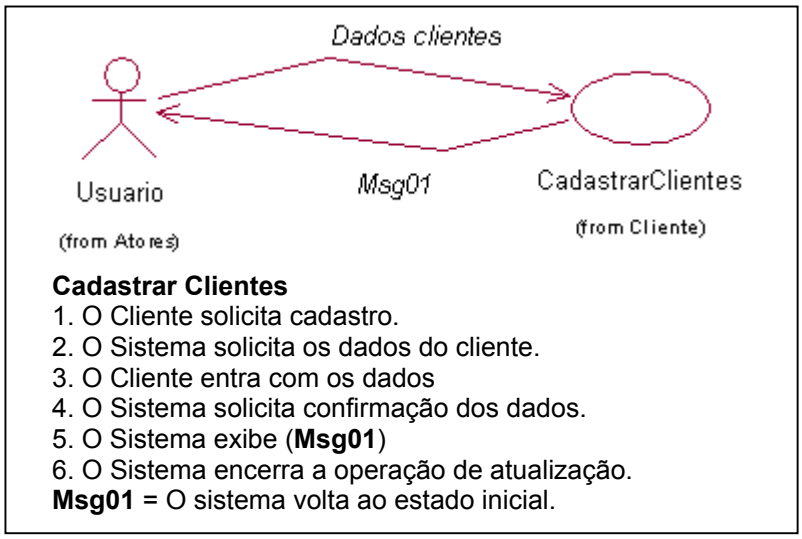


Figura 57 - Caso de uso para cadastro de cliente

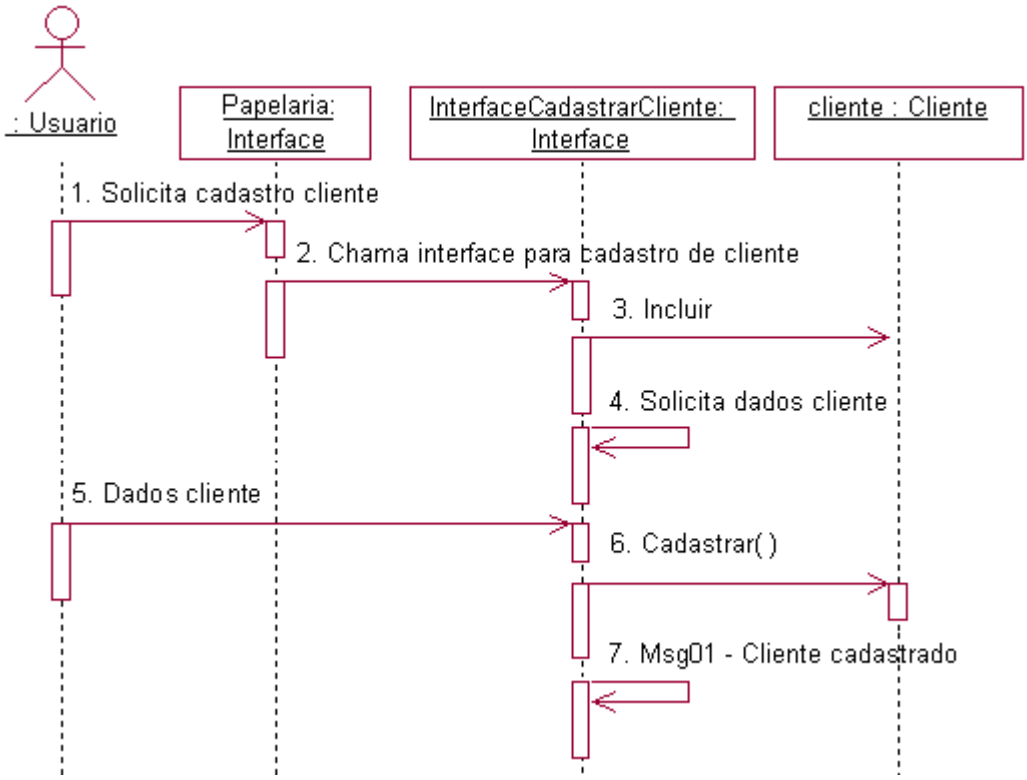


Figura 58 - Diagrama de seqüência para atualização de cliente

Considerando o código legado e os "pseudos-métodos" existentes, a Figura 59 mostra a reorganização desse código. Como já exibido na Figura 47, a reorganização ocorre pela construção de um novo método de acordo com a identificação da funcionalidade desejada. Assim, para o cadastro de clientes, dois métodos são criados: `NovoCliente` e `CadastrarCliente`.

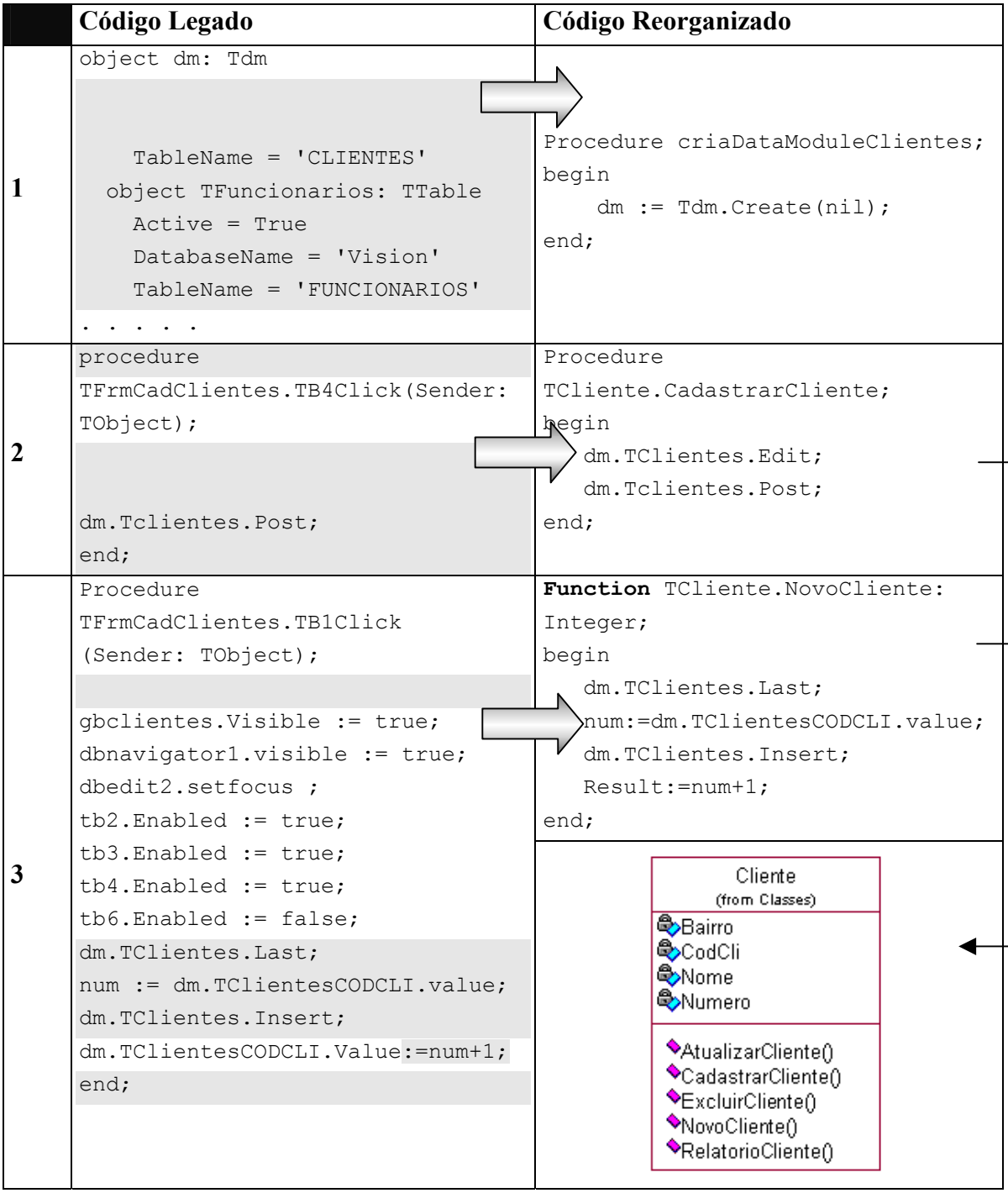


Figura 59 - Reorganização do código legado

A próxima etapa toma por base a lista de procedimentos candidatos à implementação, obtida na etapa de segmentação do sistema, para a criação desses métodos nas respectivas pseudo-classes, por não ser viável seu empacotamento. A Figura 60 apresenta a criação dos métodos que fazem a inserção e a consulta de um cliente no banco de dados, sendo nomeados de "`SetaDadosCliente`" e "`RetornaDadosCliente`", respectivamente.

```
procedure TCliente.SetaDadosClientes (Cod, Nome, Ender, Num, Bair,
                        Cid, Est, Tel, Rg, Cep, Cpf, Data: String);
begin
    with dm.TClientes do
        begin
            FieldbyName('CODCLI').AsInteger:=Cod;
            FieldbyName('NOME').AsString:=Nome;
            FieldbyName('ENDERECO').AsString:=Ender;
            FieldbyName('NUMERO').AsInteger:= StrToInt (Num);
            FieldbyName('BAIRRO').AsString:=Bair;
            FieldbyName('CIDADE').AsString:=Cid;
            FieldbyName('ESTADO').AsString:=Est;
            FieldbyName('TELEFONE').AsString:=Tel;
            FieldbyName('RG').AsString:=Rg;
            FieldbyName('CEP').AsString:=Cep;
            FieldbyName('CPF').AsString:=Cpf;
            FieldbyName('DATNASC').AsString:=Data;
        end;
end;
-------------------------------------------------------------------------
function TCliente.RetornaDadosCliente (Codigo : String; var Dados :
vetor):Boolean;
begin
    with dm.Qconsultaclientec do
        begin
           Dados[1] := IntToStr (FieldbyName('CODCLI').AsInteger);
           Dados[2] := FieldbyName('NOME').AsString;
           Dados[3] := FieldbyName('ENDERECO').AsString;
           Dados[4] := IntToStr (FieldbyName('NUMERO').AsInteger);
           Dados[5] := FieldbyName('BAIRRO').AsString;
           Dados[6] := FieldbyName('CIDADE').AsString;
           Dados[7] := FieldbyName('ESTADO').AsString;
           Dados[8] := FieldbyName('TELEFONE').AsString;
           Dados[9] := FieldbyName('RG').AsString;
           Dados[10] := FieldbyName('CEP').AsString;
           Dados[11] := FieldbyName('CPF').AsString;
           Dados[12] := FieldbyName('DATNASC').AsString;
        end;
end;
```

Figura 60 - Criação dos métodos de inserção e consulta de clientes

Com a reorganização do código concluída é iniciada a etapa de implementação do empacotamento, em que cada pseudo-classe identificada é revestida por uma "camada de software", para viabilizar a comunicação entre o sistema legado e a nova interface do usuário. Para isso, deve-se fazer a associação de cada pseudo-classe do sistema com um componente como mostra a Figura 61, sendo realizada de forma análoga para todas as pseudo-classes do sistema.

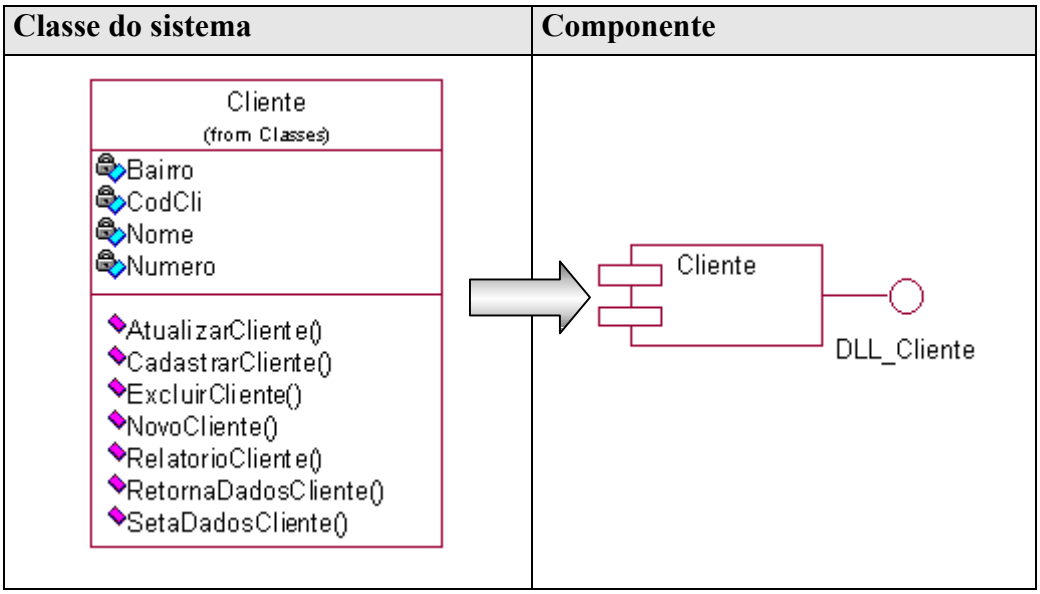


Figura 61 - Mapeamento de componentes

O diagrama de componentes identifica as funcionalidades que devem estar disponíveis para que a nova interface do usuário tenha acesso. A Figura 62 mostra o diagrama de componentes para o contexto de pedido conforme apresentado nos diagramas MAS e MAS nas Figuras 55 e 56. Em seguida, desenvolve-se uma "camada de software" em cada pseudo-classe do MASA, cujo objetivo é viabilizar a comunicação entre o sistema legado e a nova interface do usuário, recebendo os dados, convertendo-os em dados legíveis ao código legado e encaminhado para ele onde são processados. Essa camada também é responsável pelo caminho inverso, além de representar a interface dos componentes contendo as assinaturas dos métodos das pseudo-classes, que são as funcionalidades do código legado disponibilizadas para a nova interface do usuário.

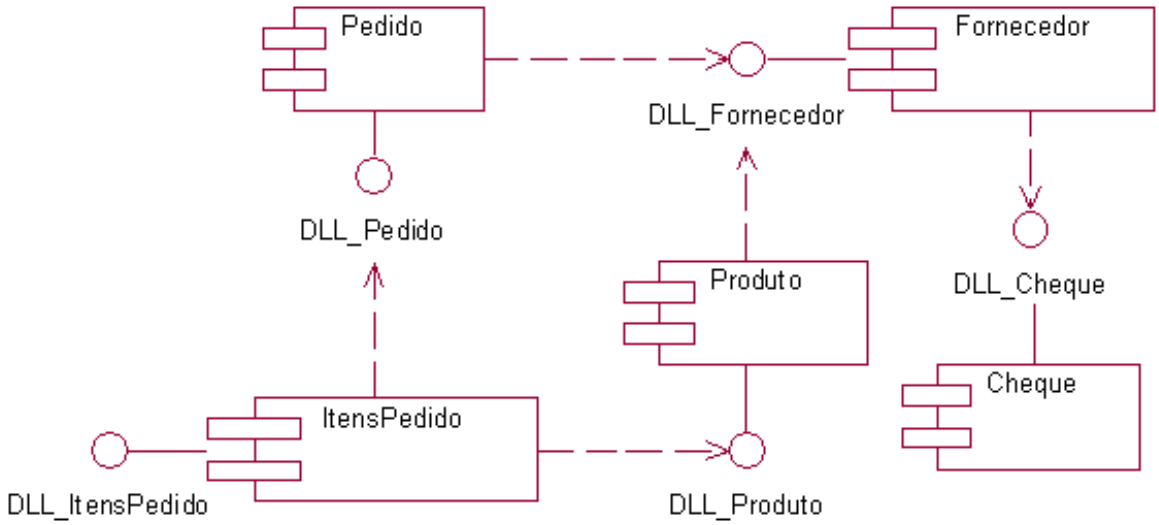


Figura 62 - Diagrama de componente para o contexto de pedido

A Figura 63 apresenta o empacotamento para a pseudo-classe "`Cliente`", envolvida por uma "camada de software", que representa a interface "`DLL_Cliente`" do componente "`Cliente`" gerado. A Figura 64 mostra a implementação dessa camada, apresentando a funcionalidade "`cadastrar clientes`", na parte marcada por (a) e as funcionalidades que são disponibilizadas na interface "`DLL_Cliente`" do componente "`Cliente`" na parte marcada por (b).

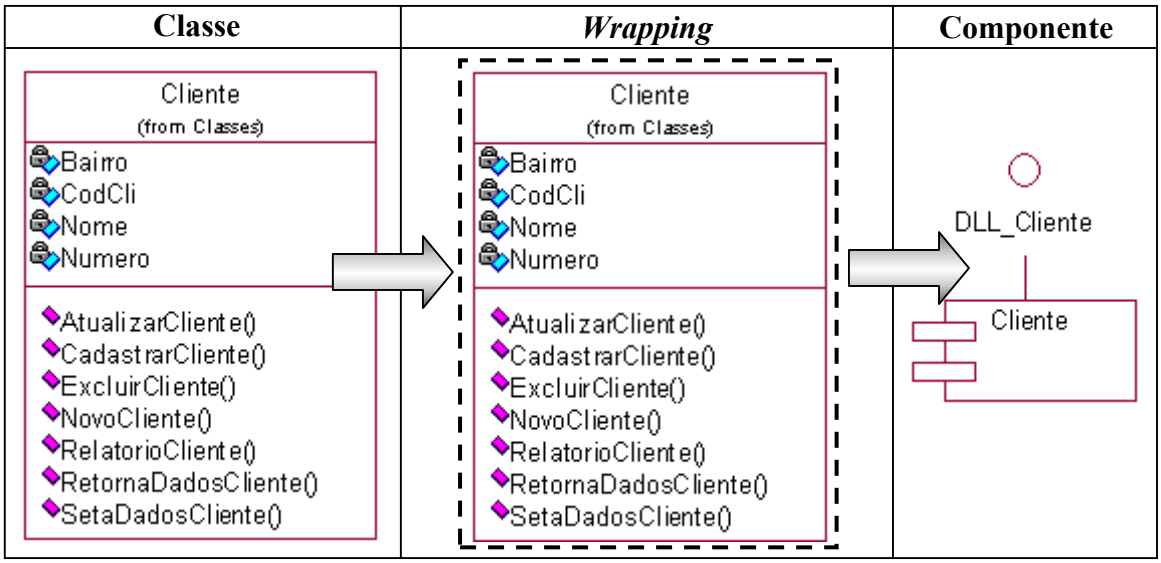


Figura 63 - Empacotamento das classes do sistema

Pela Figura 64 pode-se observar os métodos disponibilizados para a interface do

usuário são compostos por um prefixo "`Java_ClienteWrapperBean_`", que é utilizado para estabelecer o *link* de comunicação com a implementação do empacotamento que atua do lado da interface, como mostra a Figura 65. Essa implementação oferece acesso ao recurso de *middleware*, permitindo a comunicação entre o sistema legado empacotado com a nova interface do usuário. Esse processo de implementação é análogo para todas as pseudo-classes do sistema. Dessa forma, pode-se dizer que a etapa de composição dos pacotes está encerrada, permitindo que seja iniciada a confecção da nova interface do usuário, como mostra a próxima seção.

```
procedure Java_ClienteWrapperBean_cadastrarCliente
PEnv: PJNIEnv; Obj: JObject; nome, ender, num, bair,
cid, est, tel, rg, cep, cpf, data : JString)stdcall;
{$IFDEF WIN32}stdcall;{$ENDIF}{$IFDEF LINUX}cdecl;{$ENDIF}
var JVM : TJNIEnv;
NCliente : TCliente;
snome,sender,snum,sbair,scid,sest,stel,srg,scep,scpf,
sdata:String;
begin
NCliente := TCliente.Create;
// Cria uma instância do Java Ambiente.
JVM := TJNIEnv.Create(PEnv);
// Converte uma String Java para uma String Delphi
snome        := JVM.JStringToString(nome);
sender       := JVM.JStringToString(ender);
snum         := JVM.JStringToString(num);
sbair        := JVM.JStringToString(bair);
scid         := JVM.JStringToString(cid);
. . . . .
//Chama o método que faz a atribuição aos campos da tabela
NCliente.SetaDadosClientes(NCliente.NovoCliente, snome,
sender, snum, sbair, scid, sest, stel, srg, scep, scpf,
sdata);
//Chama o método para incluir novo objeto
NCliente.CadastrarCliente;
//Libera o objeto criado
NCliente.Free;
JVM.Free;
end;
. . . . .
{Declaração dos métodos que serão exportados}
exports
Java_ClienteWrapperBean_cadastrarCliente,
Java_ClienteWrapperBean_consultarCliente,
Java_ClienteWrapperBean_excluirCliente,
Java_ClienteWrapperBean_atualizarCliente,
Java_ClienteWrapperBean_relatorioCliente;
. . . . .
begin
{corpo da DLL}
end.
```

(a)

(b)

Figura 64 - Implementação da camada de software "DLL_Cliente"

```
public class ClienteWrapperBean implements java.io.Serializable
{
   public static native String cadastrarCliente(String nome, String
     ender, String num, String bair, String cid, String est, String
     tel, String rg, String cep, String cpf, String data );

   native public java.lang.String[] consultarCliente(String codigo);

   public static native String excluirCliente(String codigo);

   public static native String atualizarCliente(String codigo, String
     nome, String ender, String num, String bair, String cid, String
     est,String tel, String rg, String cep, String cpf, String data);

   native public java.lang.String[] relatorioCliente();

  . . . .

  static
    {
      System.loadLibrary("DLL_Cliente");
    }
}
```

Figura 65 - Classe de empacotamento

4.3.5 - CONFECÇÃO DA NOVA INTERFACE DO USUÁRIO

A nova interface do usuário consiste na migração da interface legada para *Web*. Essa etapa é composta pela confecção do modelo navegacional, pelo mapeamento dos componentes da interface legada para novos e pela confecção da nova interface com base nas diretrizes do processo de reengenharia de interface apresentado no capitulo 3.

A confecção do modelo navegacional é apoiada pela lista de contextos e métodos obtida na etapa 4.3.3. Esse modelo tem inicio com a definição da página principal, que representa o contexto inicial do sistema e contém um *link* para cada um dos contextos identificado. Cada contexto é composto por um conjunto de funcionalidades e está representado por uma página secundária. A Figura 66 apresenta o modelo navegacional inicial do sistema "CONTROLE DE ESTOQUE DE UMA PAPELARIA".

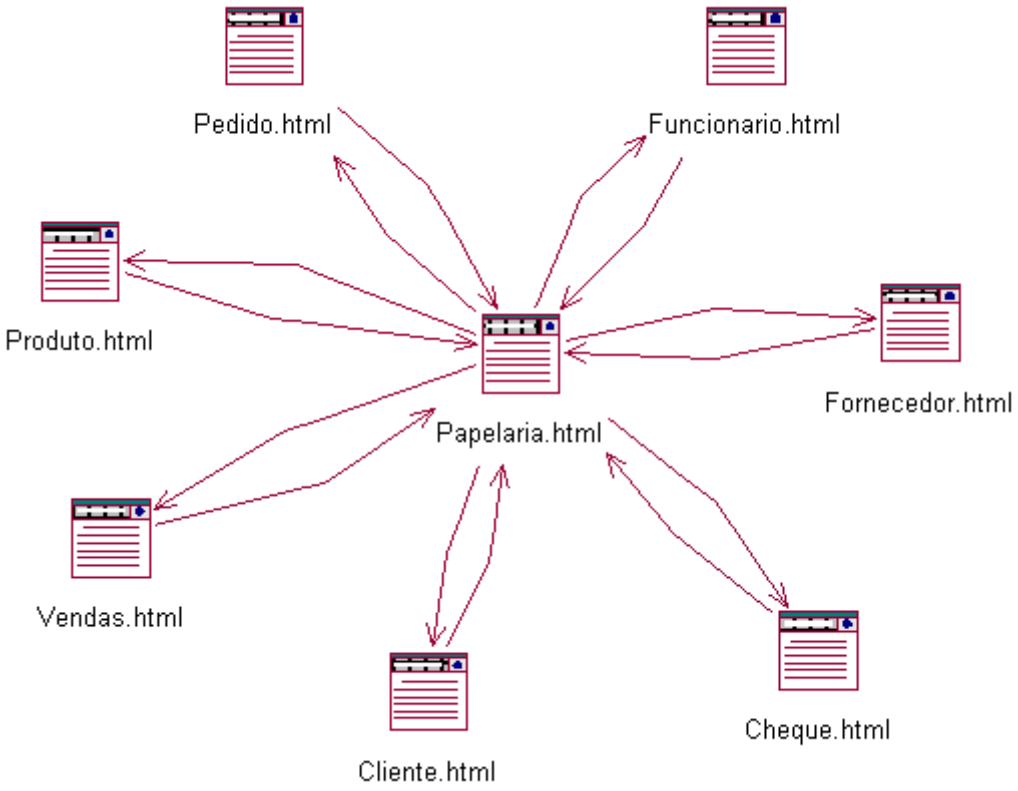


Figura 66 - Modelo Navegacional Inicial

A Figura 67 apresenta o modelo navegacional para o contexto "`Cliente`", que está representado pela página "`Cliente.html`". Essa página abriga um conjunto de *links* para as outras que representam as funcionalidades desse contexto, como por exemplo, "`CadastrarCliente.html`", apresentada em uma página de resposta, como por exemplo, "`RespostaCadastrarAtualizarCliente.html`". Essas páginas são extraídas das mensagens de repostas dos diagramas de caso de uso, identificados na seção 4.3.1.

A próxima etapa corresponde ao mapeamento dos componentes da interface legada para os da nova interface do usuário para *Web*. Essa etapa tem como base as diretrizes do processo de reengenharia de interface apresentadas na seção 3.2.5.2.

Para exemplificar o mapeamento realizado neste estudo de caso, considera-se a funcionalidade "`CadastrarClientes`", desde a tela inicial do sistema legado até o formulário encarregado de sua execução. A Figura 68 mostra o acesso ao menu principal, selecionando a opção "`Cadastrar → Clientes`", que ativa o formulário de "`Cadastro de Clientes`".

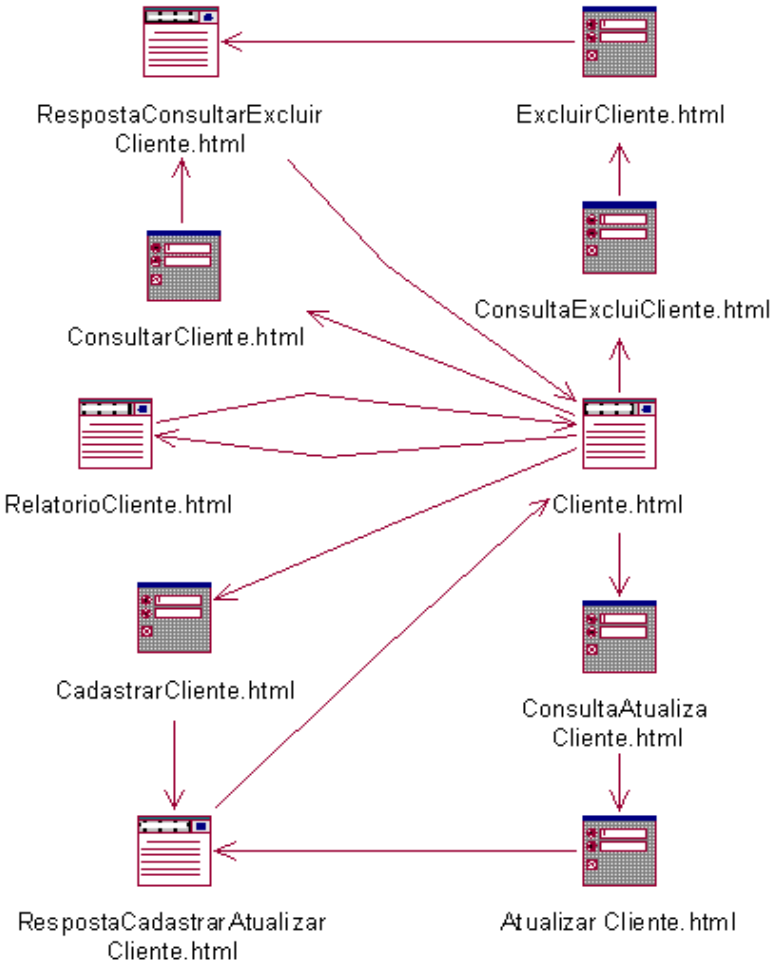


Figura 67 - Modelo Navegacional para o contexto "Clientes"

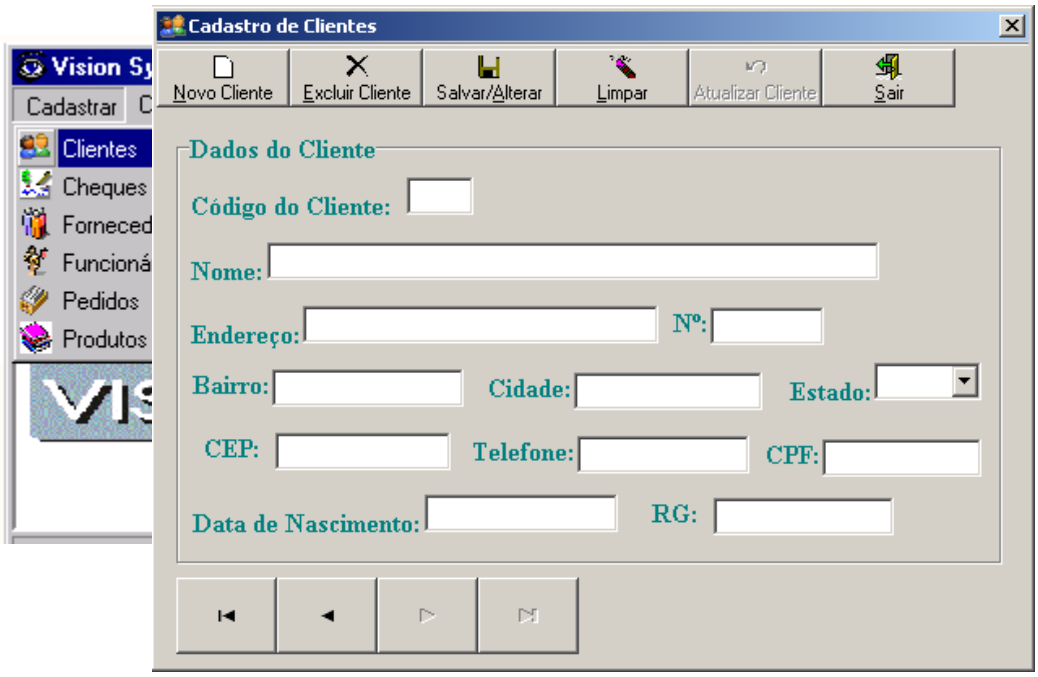


Figura 68 - Interface para cadastro de cliente

Os componentes de interface envolvidos para a execução da funcionalidade apresentada na Figura 68 são:

- **Menu:** oferece acesso às funcionalidades do sistema, como por exemplo, o menu "`Cadastrar`" que é composto por um conjunto de opções, dentre elas "`Clientes`";
- **Botões:** permitem a execução das funcionalidades do sistema, como por exemplo, o botão "`Novo Cliente`", que habilita a interface para preenchimento dos dados de um cliente;
- **Campos de entrada:** permite a entrada de um campo da entidade cliente, como por exemplo, seu nome;
- **Caixa de Seleção:** fornece um conjunto de opções ao usuário, porém permitindo a seleção de apenas uma.

Conforme apresentado nas diretrizes do processo de reengenharia de interface, esses componentes podem ser mapeados de forma equivalente ou funcional dependendo do *layout* da nova interface do usuário (*Web*).

Para o componente "Menu" foi realizado o mapeamento funcional, devido à nova organização do sistema, em que as funcionalidades estão agrupadas por contextos. A Figura 69 apresenta o mapeamento para esse componente, apresentando o legado e o novo para o menu "`Clientes`" e suas funcionalidades.

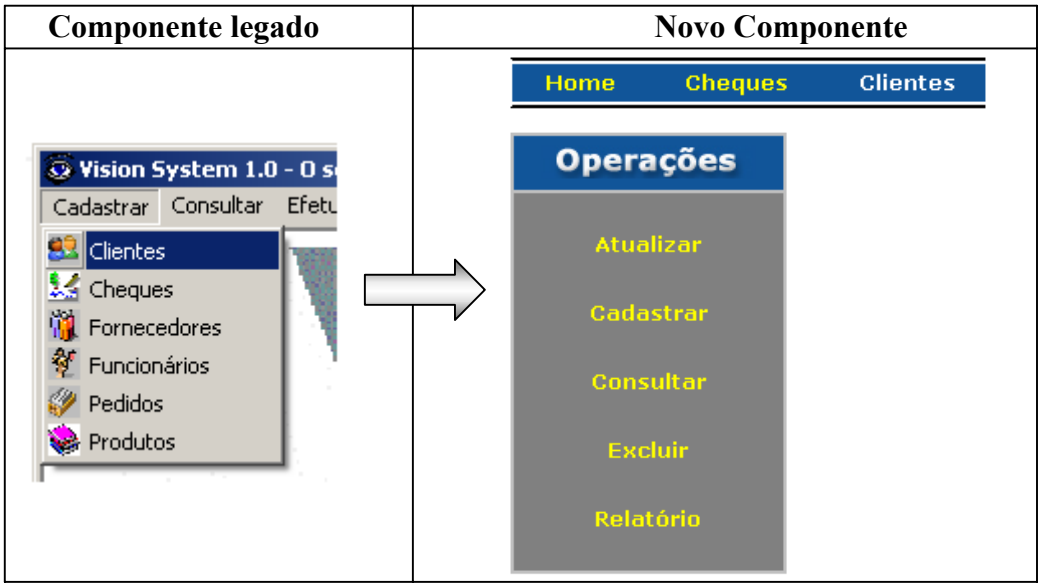


Figura 69 - Mapeamento funcional para o componente "Menu"

O demais componentes foram mapeados de forma equivalente, pois não houve a

necessidade de adequação em relação *layout* da interface.

Após a realização do mapeamento dos componentes de interface do usuário inicia-se o processo de construção da nova interface baseado nas diretrizes do processo de reengenharia de interface.

O processo tem inicio com a definição do logo do *site*, seu desenvolvimento está relacionado ao projeto de cores, as quais são utilizadas para construção do restante do *site*. A Figura 70 apresenta o logo para o *site* "Papelaria", que foi reutilizado do sistema legado, sofrendo apenas alterações quanto à aparência e as cores utilizadas.

Figura 70 - Logo do *site*

O próximo passo é a definição do título da página, informando a sua finalidade, para que o usuário se certifique do tipo de *site* que está visitando, Figura 71.

Figura 71 - Título do *site*

Com a definição do logo e do título do *site*, inicia-se a construção da primeira visão do *site*, apresentado a inserção do logo e de seu titulo, como mostra a Figura 72.

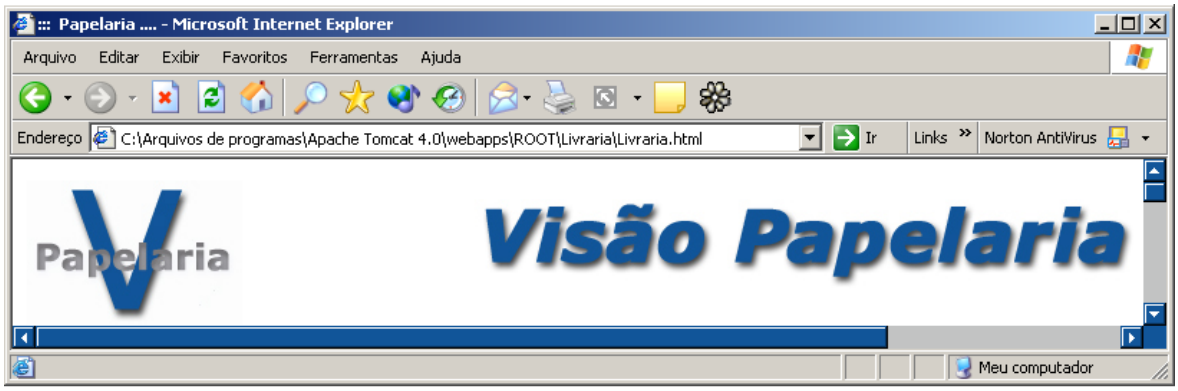


Figura 72 - Primeira visão do *site*

A Figura 73 mostra a página inicial do *site*, apresentando no menu superior todos os contextos do sistema que o usuário tem acesso. Logo abaixo do menu principal é oferecida, ao usuário, uma visão geral do *site*, informando os produtos oferecidos; os fornecedores desses produtos; os livros oferecidos pelo *site* e, uma seção "Bem vindo ao nosso *site*".

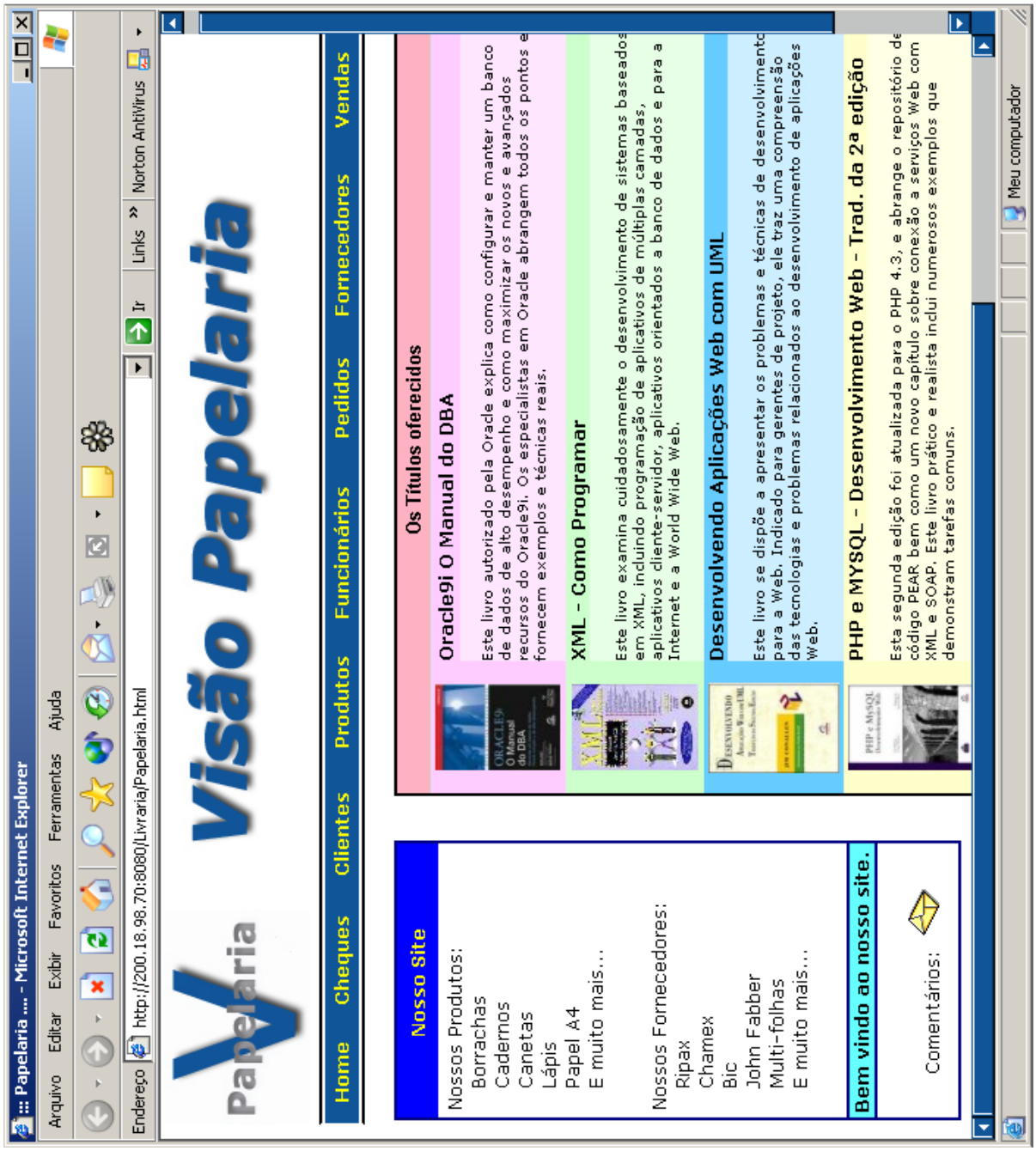


Figura 73 - Contexto inicial do *site*

A Figura 74 mostra a interface que trata das funcionalidades do contexto "`Clientes`". Ao clicar na opção "`Cadastrar`", Figura 74, é exibida a interface que faz o cadastro de um

cliente, que está representada pela página "`CadastrarCliente.html`", como mostra a Figura 75. Essa página é gerada através do mapeamento dos componentes da interface legada e da reorganização desses para a composição do novo *layout*. Essa interface difere da legada quanto a não aceitação de dados inválidos, como por exemplo, a inibição de entrada de caracteres em campos numéricos como "`CPF`" e o preenchimento de todos os campos de entrada, evitando que dados sejam omitidos do sistema. Ao se clicar no botão "`Cadastrar`" a página da Figura 76 é exibida.

A operação de consulta de um cliente está representada pela página "`ConsultarCliente.html`" como mostra a Figura 77. A execução dessa funcionalidade resume-se na solicitação do código do cliente, em que é verificada sua existência e, exibindo o resultado na página de resposta, representada na Figura 78.

A operação de exclusão tem funcionamento análogo ao apresentado para a consulta de cliente. Essa operação está representada pela página "`ExcluirCliente.html`", que verifica a existência do cliente pelo seu código, como mostra a Figura 79, apresentando como resposta a essa funcionalidade a página mostrada na Figura 80, que apresenta os dados do cliente que foi excluído, além de uma mensagem de confirmação da operação.

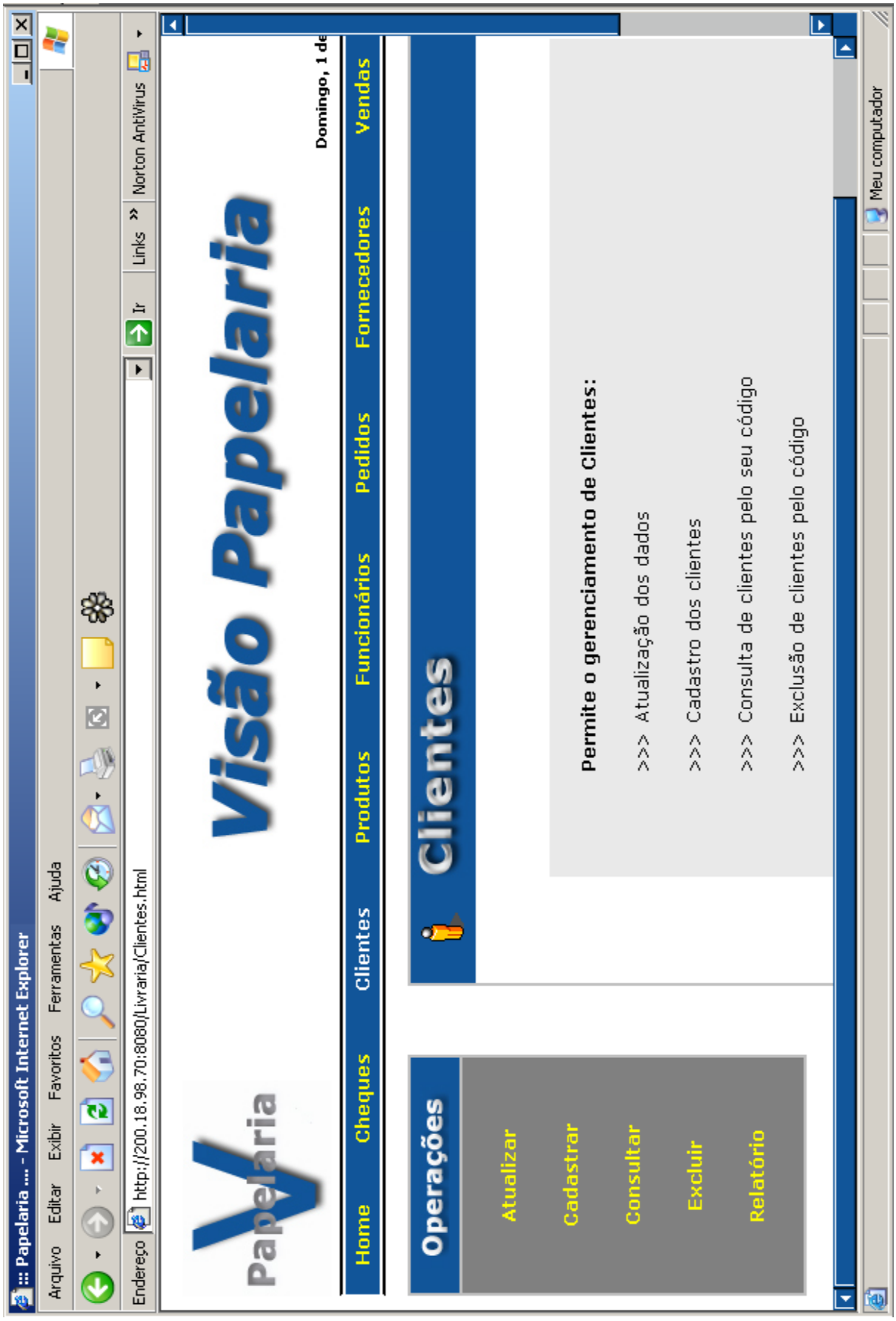


Figura 74 - Página para o contexto de clientes

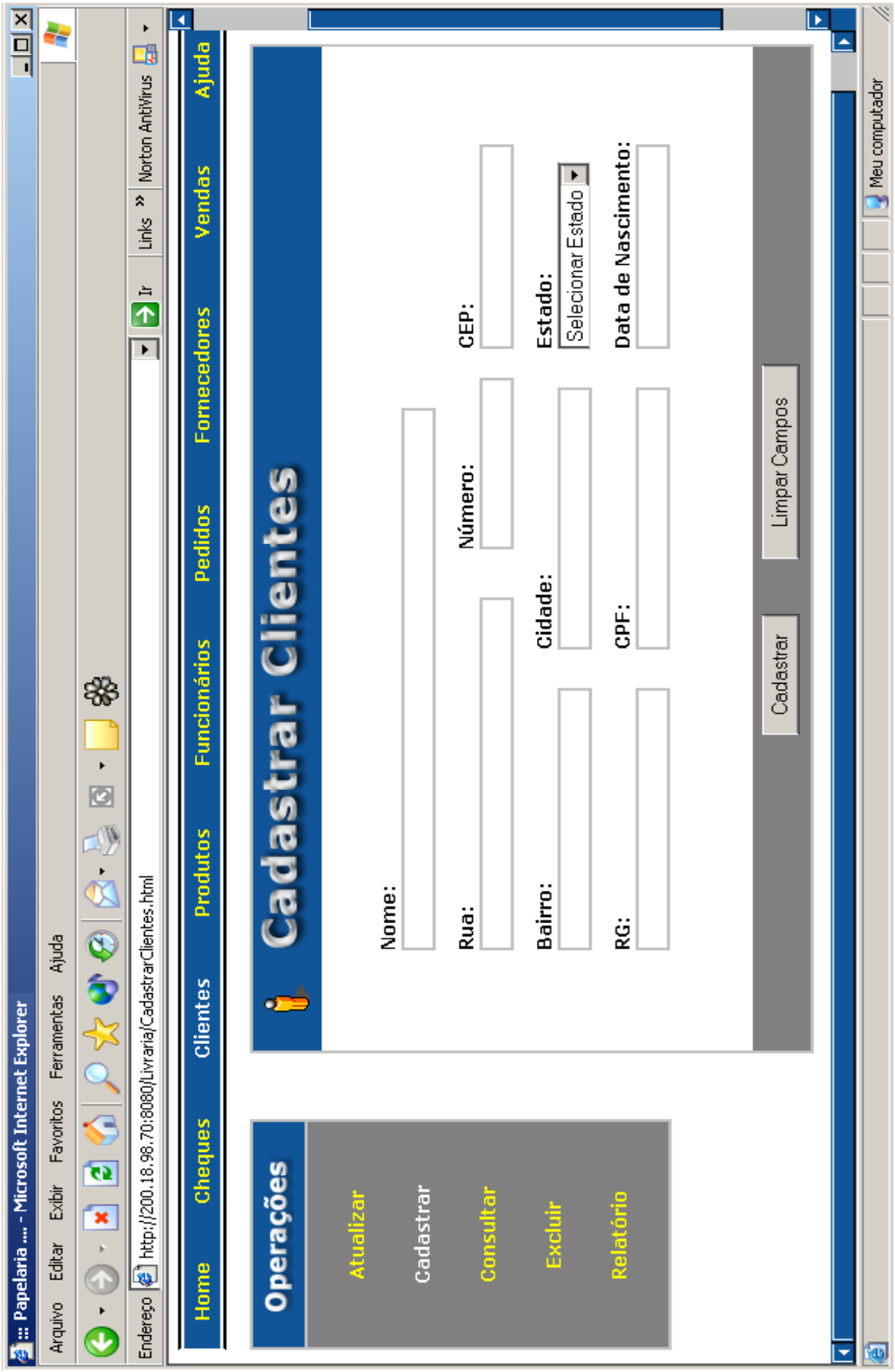


Figura 75 - Página para o cadastro de clientes

Figura 76 - Página de resposta ao cadastro de clientes

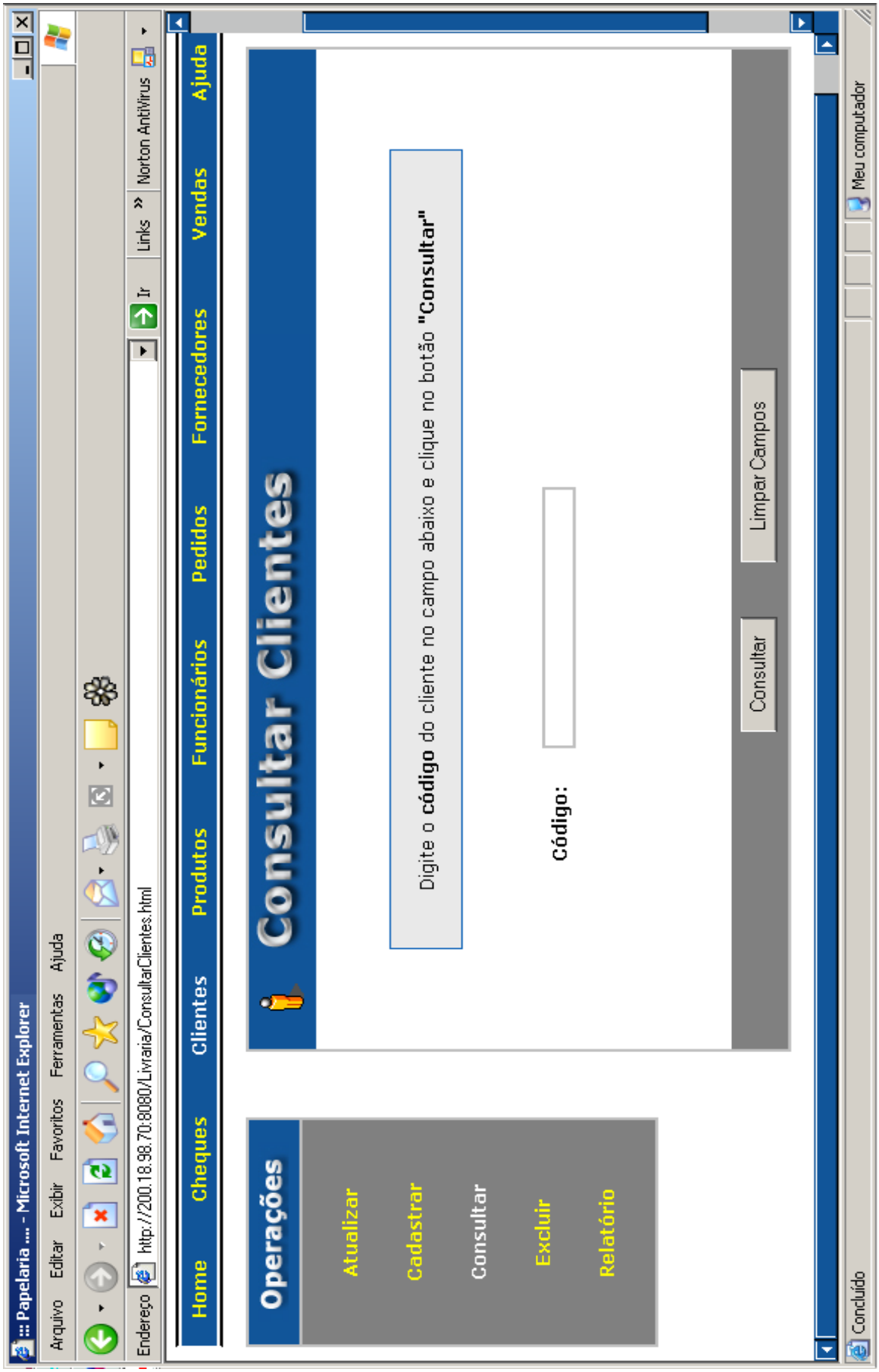


Figura 77 - Página para a consulta de clientes

Figura 78 - Página de resposta a consulta de um cliente

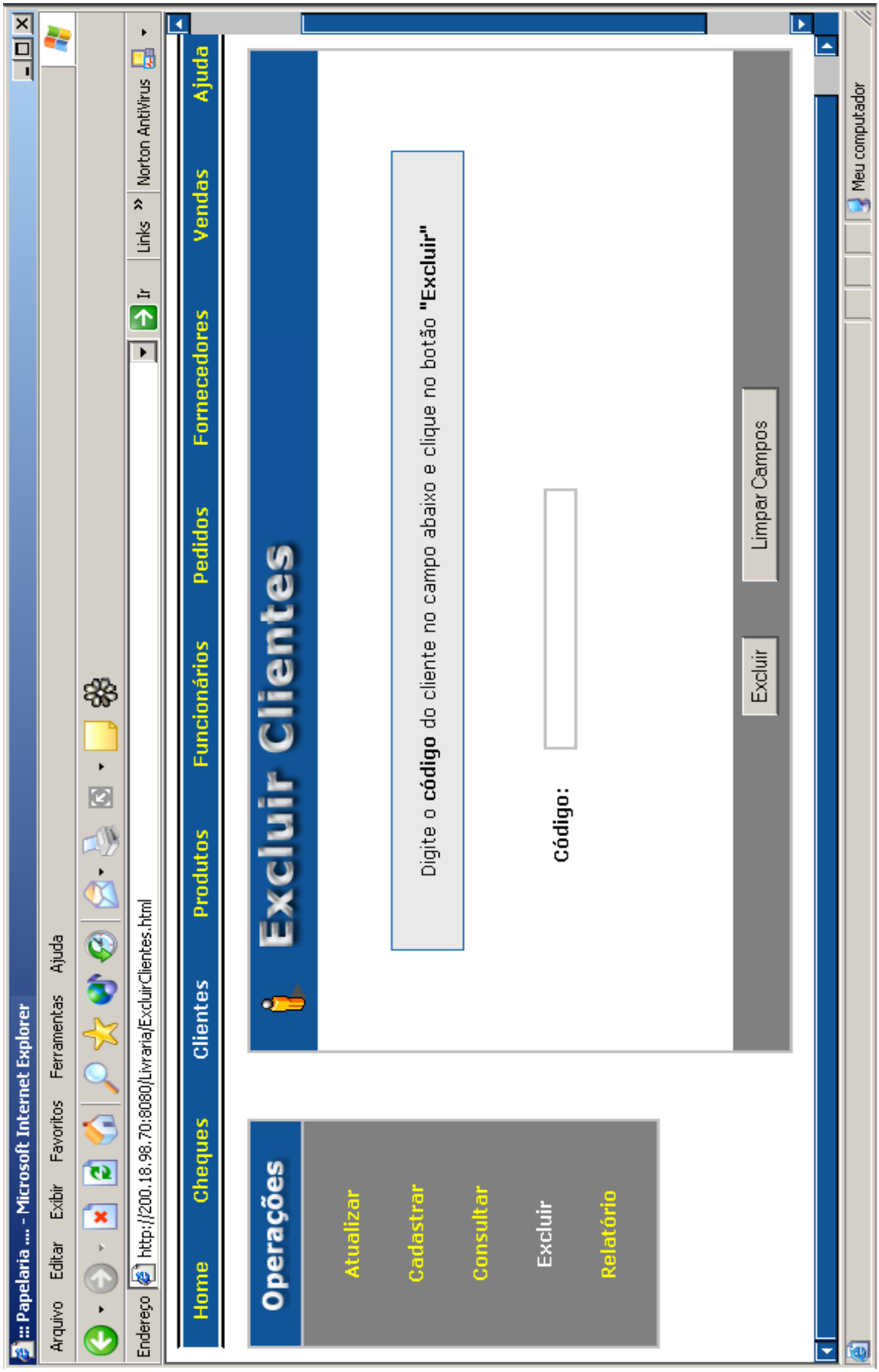


Figura 79 - Página para exclusão de um cliente

Figura 80 - Página de reposta a exclusão de um cliente

A confecção das interfaces do usuário está relacionada a uma das principais características do processo de reengenharia de interface, o reuso. Pode ser observado com maior freqüência nas páginas de resposta, como as de funcionalidades de consulta e de exclusão de um cliente, que apresentam as mesmas informações de retorno, apenas divergindo na mensagem de confirmação como "Cliente encontrado com sucesso" e "Cliente excluído com sucesso". A implementação dessas interfaces é baseada em componentes de software, em que a interface é empacotada em um método de classe e as informações de retorno bem como as mensagens de confirmação são enviadas como parâmetros, permitindo que elas atuam em funcionalidades diferentes. Essa etapa marca o encerramento da confecção da interface do usuário, permitindo que seja iniciada a etapa de projeto do sistema como mostra a próxima seção.

4.3.6 - PROJETO DO SISTEMA

Esta etapa corresponde à integração entre o sistema legado empacotado com a nova interface do usuário, sendo necessário a definição do recurso de *middleware* responsável por essa integração. Nesse sistema foram utilizados *servlets*. Em seguida, é feita a modelagem do sistema para cada funcionalidade como mostram as Figuras 81 e 82, para "`cadastrar cliente`" e "`excluir cliente`". Esses diagramas mostram o caminho completo para execução dessas funcionalidades, partindo desde o contexto inicial do sistema representado pela página "`Papelaria.html`", passando pelas páginas que contêm as funcionalidades dos contextos como "`CadastrarCliente.html`" e "`ExcluirCliente.html`". Cada uma dessas é composta por um *servlet*, que encaminha os dados da interface do usuário para o código legado através da camada de empacotamento representada pela interface "`DLL_Cliente`". Essa interface recebe os dados do *servlet*, converte-os em dados legíveis ao código legado e os encaminha para serem executados na classe "`Cliente`". Isso é possível devido ao arquivo "`ClienteWrapperBean.java`" conter as assinaturas dos métodos dessa classe, estabelecendo um canal de comunicação entre o código legado e a interface do usuário. Esses digramas são confeccionados para todas as funcionalidades do sistema.

Com a definição da modelagem do sistema é iniciada a etapa de acoplamento do sistema, que será apresentada na próxima seção.

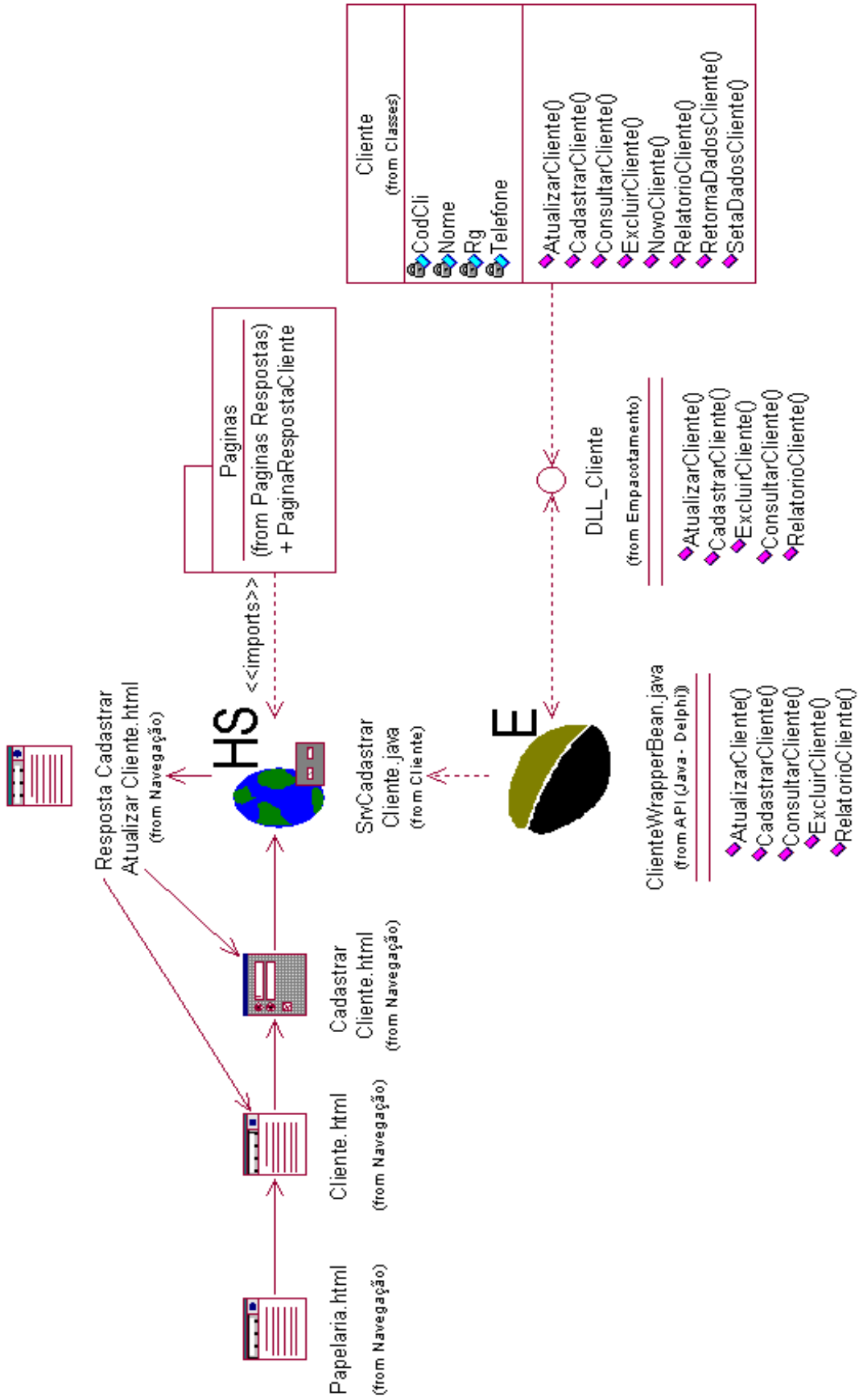


Figura 81 - Diagrama de projeto para o cadastro de cliente

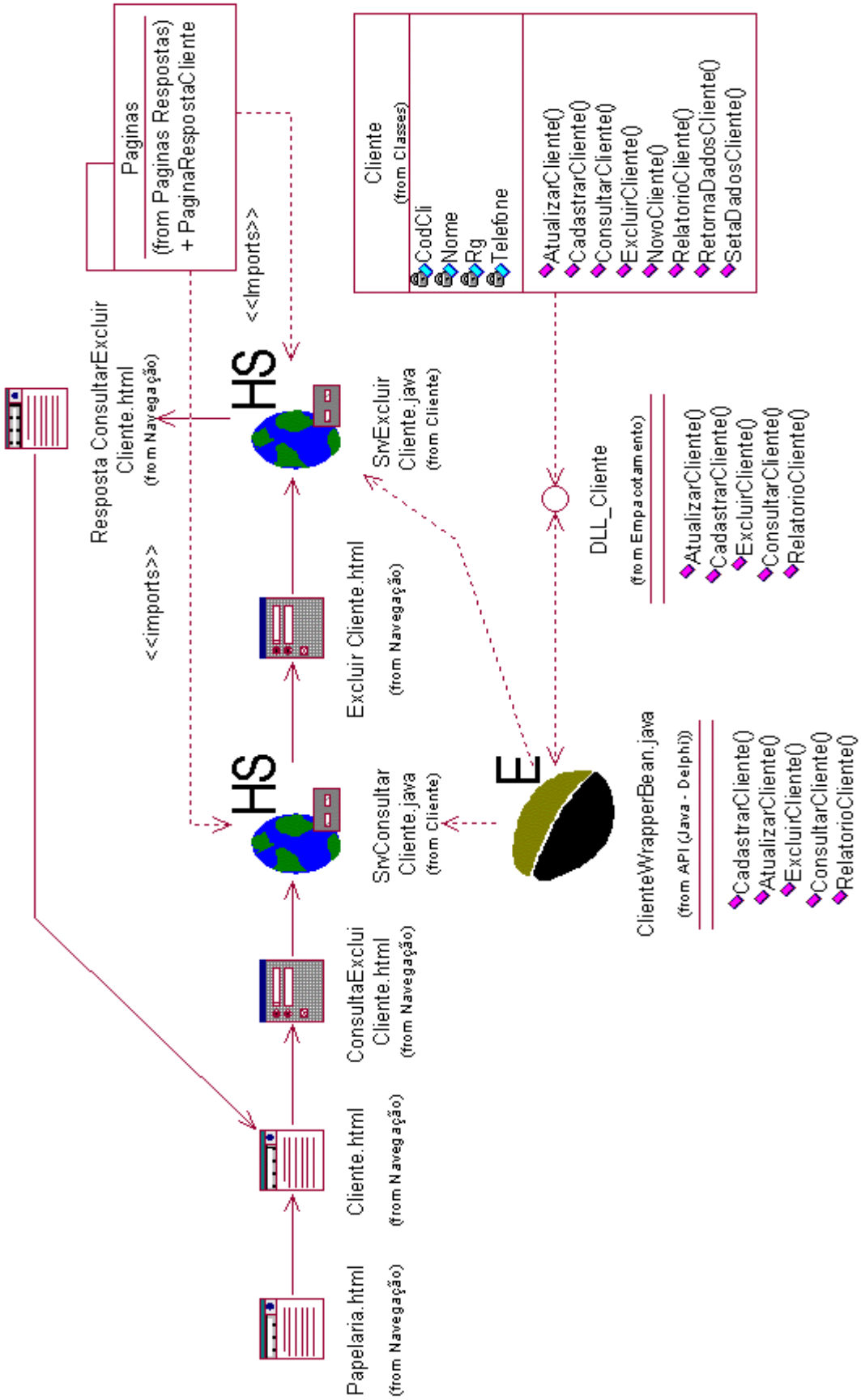


Figura 82 - Diagrama de projeto para excluir cliente

4.3.7 - ACOPLAMENTO DO SISTEMA

Para exemplificar a implementação desse sistema considere a funcionalidade "`CadastrarCliente`", cujo trecho de código da página "`CadastrarCliente.html`" exibido na Figura 83 destaca, em negrito, a chamada ao *servlet* utilizado para obter os dados da interface e encaminhar ao código legado. Na Figura 84 apresenta-se o trecho de código do *servlet* "`SrvCadastraCliente.java`", destacando, em negrito, o código que obtém os dados da interface e os encaminha para o código legado através do método "`cadastrarCliente`" presente no arquivo "`ClienteWrapperBean.java`" destacado em negrito, como mostra a Figura 85. Finalmente, na Figura 86 apresenta-se o trecho de código da interface "`DLL_Cliente`", que recebe os dados do *servlet*, converte-os em dados legíveis ao código legado e os encaminha para serem executados na classe "`Cliente`" através do método "`Cadastrar Cliente`" em negrito.

```
<html>
<head>
<title>::: Papelaria.....</title>
</head>
<body>

. . . . .
<FORM NAME="cadastraCliente"
ACTION="http://200.18.98.70:8080/servlet/SrvCadastraUsuario" METHOD="GET"
onsubmit="return Valida(this)">

. . . . .
<td width="80%" height="25" bgcolor="#FFFFFF" colspan="8"><input
type="text" name="nome" size="62" style="border: 2 solid #C0C0C0"
onkeypress="return validaSomenteString(this, event)"></td>

. . . . .
</FORM>
</body>
</html>
```

Figura 83 - Arquivo HTML "CadastrarCliente.html "

```
//variáveis globais
    ClienteWrapperBean cliente = new ClienteWrapperBean();
    PaginaRespostaCliente pagResp = new PaginaRespostaCliente();
        public void doGet(HttpServletRequest request, HttpServletResponse
               response) throws ServletException,IOException {
      StringBuffer pr = new StringBuffer();
      String snome            = request.getParameter("nome");
      String scep             = request.getParameter("cep");
      String stelefone        = request.getParameter("telefone");
      String scpf             = request.getParameter("cpf");
      . . . .
      String Frase = "Cliente cadastrado com sucesso";
      cliente.cadastrarCliente(snome, sender, snum, sbair, scid, sest,
                  stel, srg, scep, scpf, sdata );
     ....
      pagResp.paginaRespostaCadastroUsuario(pr, Frase, snome, sender, snum,
      sbair, scid, sest, stel, srg, scep, scpf, sdata );
      . . . .
}
```

Figura 84 - Arquivo *Servlet* "SrvCadastraCliente.java"

```
public class ClienteWrapperBean implements java.io.Serializable
{
  . . . .
  public static native String cadastrarCliente(String nome, String
            ender, String num, String bair, String cid, String est,
            String tel, String rg, String cep, String cpf, String
            data );
  native public java.lang.String[] consultarCliente(String codigo);
    public static native String excluirCliente(String codigo);
    public static native String atualizarCliente(String codigo, String
            nome, String ender, String num, String bair, String cid,
            String est, String tel, String rg, String cep, String
            cpf, String data );
  native public java.lang.String[] relatorioCliente();
  static {
    System.loadLibrary("DLL_Cliente");
  }
 }
```

Figura 85 - Arquivo Java "ClienteWrapperBean.java"

```
procedure Java_ClienteWrapperBean_cadastrarCliente(PEnv: PJNIEnv; Obj:
JObject; nome, ender, num, bair, cid, est, tel, rg, cep, cpf, data :
JString) stdcall;{$IFDEF WIN32} stdcall; {$ENDIF} {$IFDEF LINUX} cdecl;
{$ENDIF}
var
 JVM : TJNIEnv;
 NCliente : TCliente;
 snome, sender, snum, sbair, . . . , srg, scep, scpf, sdata: String;
begin
    //Cria novo objeto cliente.
    NCliente := TCliente.Create;
    // Converte uma String Java para uma String Delphi
    snome        := JVM.JStringToString(nome);
    sender       := JVM.JStringToString(ender);
    snum         := JVM.JStringToString(num);
    sbair        := JVM.JStringToString(bair);
    scid         := JVM.JStringToString(cid);
    sest         := JVM.JStringToString(est);
    . . . .
    //Chama o método para incluir novo objeto
    NCliente.SetaDadosClientes(NCliente.NovoCliente, snome, sender,
            snum, sbair, scid, sest, stel, srg, scep, scpf, sdata);
    NCliente.CadastrarCliente;
    . . . .
end;
```

Figura 86 - Arquivo Delphi "DLL_Cliente.dpr"

Essa etapa é realizada para cada digrama de projeto obtido na etapa anterior, sendo implementados apenas os *servlets*, pois o restante do sistema, legado empacotado e a nova interface *Web*, encontra-se pronto, fornecendo somente as chamadas para sua utilização. Dessa forma, finaliza-se o processo de reengenharia de interface.

4.4 - CONSIDERAÇÕES FINAIS

Este capítulo apresentou a aplicação das diretrizes do processo de reengenharia de interface em um sistema procedimental desenvolvido no ambiente *Delphi*, disponibilizando-o para a *Web*.

Com a aplicação do processo de reengenharia de interface, obteve-se um sistema reorganizado em camadas: a de interface do usuário, representada pelas páginas em HTML; a de integração, representada pelos *servlets*; a de comunicação entre a nova interface e o sistema legado e a de empacotamento, representada pelos componentes de software extraídos do sistema legado. Para essa última, destaca-se a nova organização das funcionalidades do sistema, que foram reorganizadas em métodos de pseudo-classes/classes, reduzindo o seu tamanho e a sua complexidade, proporcionando maior manutenibilidade.

A aplicação das diretrizes do processo de reengenharia de interface é viável para o empacotamento de software, devido a reutilização do ambiente nativo do sistema, por meio da reconstrução do novo sistema com base nos componentes de software identificados e empacotados. Além da preservação do conhecimento das pessoas com ele familiarizadas.

Observou-se ainda, a situação de reuso no desenvolvimento de interface, principalmente, nas páginas que tratam das respostas às funcionalidades do sistema. Essas foram inseridas como métodos de classes, recebendo e enviando parâmetros de/para os *servlets* que as referenciam, atuando em funcionalidades distintas.

CAPÍTULO 5

CONCLUSÕES E TRABALHOS FUTUROS

5.1 - CONSIDERAÇÕES INICIAIS

Na busca de atender à evolução de sistemas com a utilização de recursos computacionais mais recentes, técnicas de empacotamento de software e processos de reengenharia de interface foram estudados e avaliados, neste trabalho. Diretrizes foram desenvolvidas para que sistemas legados com interfaces gráficas de baixa usabilidade sejam transformados para *Web*. Os sistemas desenvolvidos no ambiente *Delphi*, com ou sem características orientadas a objetos são o foco deste trabalho.

Para realização deste trabalho foram utilizados três sistemas legados desenvolvidos no ambiente *Delphi*, sendo um com características da orientação a objetos e dois com orientação a procedimentos; diretrizes de algumas técnicas de empacotamento existentes na literatura para se elaborar as diretrizes do processo de reengenharia de interface.

A seção 5.2 apresenta os resultados obtidos e a seção 5.3 sugestões para trabalhos futuros.

5.2 - RESULTADOS OBTIDOS

Um dos resultados obtidos com a realização deste trabalho é um processo prospectivo de reengenharia de interface para sistemas legados desenvolvidos no ambiente *Delphi*, com ou sem características da orientação a objetos, para sistemas *Web* preservando o ambiente original de desenvolvimento, porém, com nova interface para a *Web*. Dessa forma, um conjunto de diretrizes que permite aos engenheiros de software realizar o empacotamento do sistema legado, migrar e confeccionar a nova interface do usuário apoiado pelas diretrizes que auxiliam na melhoria de sua usabilidade foi elaborado.

A aplicação do processo de reengenharia de interface mostra a viabilidade de se aplicar o empacotamento de software preservando o ambiente nativo de cada um deles, o conhecimento das pessoas familiarizadas com o sistema e das regras de negócios das empresas que os utilizam. Pode-se notar, também, a nova organização do sistema em camadas: de interface, de *middleware* e de empacotamento, sendo essa última caracterizada pela segmentação das funcionalidades do sistema em componentes de software, os quais passam a atuar em contexto definido. Pode-se inferir que com essa organização há aumento de manutenibilidade do sistema, pois a parte funcional encontra-se isolada da interface do usuário e, ainda, a organização das funcionalidades do sistema em componentes permite que os mantenedores atuem de forma segura, com qual componente devem alterar.

A técnica de empacotamento empregada neste processo de reengenharia de interface mostra ser mais abrangente que as demais encontradas na literatura **(MOORE, 1998) (SNEED, 1996b, 1997, 1999b, 2002)**, pois permite que um sistema legado seja disponibilizado para diferentes categorias de sistemas, como para *Web*, para um sistema cliente-servidor, ou para sistema local. Isso é possível devido ao recurso empregado no empacotamento do sistema ser uniforme para essas categorias.

Com a aplicação do empacotamento observa-se aumento no número de métodos das classes, porém, há também maior reuso desses na implementação do sistema. Isso ocorre devido à combinação dos conceitos das técnicas de empacotamento de software com os de desenvolvimento baseado em componentes. Esses conceitos estão centrados na redução da complexidade das funcionalidades que são empacotadas, aumentando a sua reusabilidade.

O desenvolvimento da interface do usuário no processo de reengenharia de interface é realizado com base nos métodos de avaliação de usabilidade, que proporcionam melhorias na usabilidade do produto final e, conseqüentemente, aumentam a satisfação do usuário. Nesse contexto, pode-se também observar o reuso de interface do usuário, isto é, uma página pode

ser utilizada como resposta para distintas funcionalidades do sistema.

5.3 - TRABALHOS FUTUROS

Os trabalhos futuros sugeridos visam aprimorar ou expandir o processo de reengenharia aqui proposto.

- Instanciar o processo de reengenharia de interface para sistemas desenvolvidos em linguagens de programação como C, C++ e COBOL, com ou sem interfaces orientadas a caracteres, para avaliar a aplicabilidade das diretrizes propostas neste processo;

- Desenvolvimento de um apoio computacional que auxilie na identificação de alguns documentos e na automatização de algumas etapas do processo, tais como: a identificação da lista de componentes da interface legada e seus possíveis mapeamentos para componentes *Web*; a automatização do empacotamento, que se resume na reorganização do código legado e no desenvolvimento da camada de software responsável pela comunicação com a interface do usuário;

- Realizar um estudo comparativo entre a aplicação do processo de reengenharia de interface com um processo de reengenharia convencional, com o intuito de quantificar o tempo gasto em cada um e qualificar o resultado do produto final;

- Aplicação de um conjunto de testes para garantir que a funcionalidade do sistema legado foi preservada após o empacotamento;

- Realizar um estudo comparativo entre a usabilidade do sistema legado com a usabilidade do novo sistema, quando o alvo da migração for para sistemas locais;

- Utilizar outros métodos de avaliação de usabilidade de interface no processo de reengenharia de interface para comparar o resultado do produto final em relação ao método utilizado neste trabalho, além de avaliar a adaptação desses métodos às diretrizes desse processo.

REFERÊNCIAS BIBLIOGRÁFICAS

AOYAMA, M. **New age of software development:** how component-based software engineering changes the way of software development?. Proceedings of International Workshop on Component-Based Software Engineering, Kyoto, Japan, April 1998.

ARGOUML. **Site oficial da ArgoUML**. Disponível em: <http://www.tigirs.org>. Acesso em: 15 fev. 2003.

ASMAN, P. **Legacy wrapping.** Disponível em: <http://jerry.cs.uiuc.edu/~plop/plop2k/proceedings/Asman/Asman.pdf>. Acesso em: 15 fev. 2003.

AYERS, D. et al. **Professional Java server programming**, Wrox Press, 1999.

BARANAUSKAS, M. C. C.; ROCHA, H. V. **Design e Avaliação de Interfaces Humano-Computador**, 1ª ed. São Paulo, IME-USP, 2000. 242p.

BERNARD, M.; LARSEN, L. **What is the best layout for multiple-column web pages?** Disponível em: <http://wsupsy.psy.twsu.edu/surl/usabilitynews/3S/layout.htm>. Acesso em: 10 jan. 2003a

BERNARD, M.; HULL, S.; DRAKE, D. **Where should you put the links?** A comparison of four locations. Disponível em: <http://wsupsy.psy.twsu.edu/surl/usabilitynews/3S/layout.htm>. Acesso em: 10 jan. 2003b;

BOOCH, G. et al **UML, Guia do usuário**. Rio de Janeiro : Editora Campos. 2000;

BORLAND. **Site oficial da Borland Brasil**. Disponível em: <http://www.borland.com.br>. Acesso em: 14 jan. 2003;

CANTÚ, M. **Mastering Delphi 6**. Ed. Sybex, 2001;

Chikofsky, E. J.; Cross, J. H. **Reverse engineering and design recovery -** A Taxonomy, IEEE Software. v.7, n.1, p. 13-17, 1990;

COLEMAN, D. et al. **Object-Oriented Development** - The Fusion Method. Prentice Hall. 1994;

D'SOUZA, D. F.; WILLS, A. C. **Objects, components, and frameworks with UML :** the catalysis approach. Addison-Wesley Object Technology Series. 1998;

DEITEL, H. M.; DEITEL, P. J. **Java, como programar**. tradução Edson FurnanKiewicz. 3.ed. Porto Alegre. Bookman. 2001;

DEMEYER, S. et al. **A pattern language for reverse engineering**. Disponível em: <http://www.iam.unibe.ch/~famoos/ patterns/>. Acesso em: 10 jan. 2003.

GRAND, M. **Patterns in Java**. vol. 1. John Wiley & Sons, Inc 177. p. 205-243. 1998;

IONA **Site oficial da Iona**. Disponível em: <http://www.iona.com>. Acesso em: 13 fev. 2003;

JACOBSON, I. et al. **Object-oriented software engineering -** a use case driven approach. Addison - Wesley. ACM Press. p. 289-312. 1997;

JACOBSON, I. et al. **Software reuse.** archictecture, process and organization for business success. Addison Wesley. 1998;

JOHN, BONNIE E. et al. **The GOMS family of analysis techniques:** tools for design and evaluation. Carnegie Mellon University. August. 1994;

JOHN, BONNIE E. et al. **Using GOMS for user interface design and evaluation:** which technique?. Carnegie Mellon University. ACM Transactions on Computer-Human Interaction, Vol. 3. No. 4. Pages 287-319. December 1996a;

JOHN, BONNIE E. et al. **The GOMS family of user interface analysis techniques:** comparison and contrast. Carnegie Mellon University. ACM Transactions on Computer-Human Interaction. Vol. 3. No. 4. Pages 320-351. December 1996b;

KIERAS, D. **A guide to GOMS model usability evaluation using GOMSL and CLEAN3.** University of Michigan. Disponível em: <http://citeseer.nj.nec.com/david99guide.html>. Acesso em: 12 dez. 1999;

LEMOS, G. S. **PRE/OO - Um processo de reengenharia orientada a objetos com ênfase na garantia de qualidade**. Dissertação de Mestrado. PPGCC-UfsCar. São Carlos-SP. 2002;

MOORE M. M. **Knowledge - based User Interface Migration**. Proceedings of the International Conference on Software Maintenance. August. p. 72. 1994;

MOORE, M. M. **Rule-based detection for reverse engineering user interface**, Proceedings of WCRE - Working Conference on Reverse Engineering. p. 42. 1996;

MOORE, M. M. **User interface migration**. Ph.D. Dissertation. College of Computing. Georgia Institute of Technology. Atlanta Georgia 30332-0280. 1998;

MOORE, M. M.; MOSHKINA, L. **Migrating legacy user interfaces to the internet:** shifting dialogue initiative. Proceedings on 7th Working Conference on Reverse Engineering. p. 52. 2000;

MOWBRAY, T.; ZAHAVI, R. **The Essential CORBA**. New York : John Wiley & Sons. p. 231. 1995;

NIELSEN, J.; TAHIR, M. **Homepage :** 50 Websites desconstruídos. Rio de Janeiro : Editora Campus. 2000;

NIELSEN, J. **Usability engineering**. Morgan Kaufmann Publishers. páginas: 115-163. 1994;

NIELSEN, J. **Projetando websites** - designing web usability. Rio de Janeiro : Editora Campus. 2000;

OMG. **Site oficial do object management group**. Disponível em: <http://www.omg.org>. Acesso em: 15 dez. 2002;

PENTEADO, R. A. D. et al. **Reengineering of legacy systems based on transformations using the object oriented paradigm**, V Working Conference on Reverse Engineering - IEEE. Honolulu. Hawai. Páginas 144-153. 1998;

PENTEADO, R. A. D. **Um método para engenharia reversa orientada a objetos**. Tese de Doutorado. Instituto de Física de São Carlos. Universidade de São Paulo - São Carlos/SP, 1996;

PRADO, A. F. **Desenvolvimento de software orientado a objetos.** notas de aula. Universidade Federal de São Carlos - São Carlos-SP, Departamento de Computação. 2001;

PREE, W. **Component-based software development** - a new paradigm in software engineering?. Software Engineering Group. University of Constance. 1997;

PRESSMAN, R S. **Software engineering:** a practitioner's approach, 5ª Edição, Junho de 2000;

RATIONAL. **Site oficial da Rational Rose**. Disponível em: <http://www.rational.com>. Acesso em fev. 2003;

RECCHIA, E. L. **Engenharia reversa e reengenharia baseada em padrões**. Dissertação de Mestrado. PPGCC-UFsCar. São Carlos-SP. 2002;

SIEGEL, J. **CORBA fundamentals and programming**. London : Ed. Wiley. 1996;

SIEGEL, J. **CORBA fundamentals and programming**. Second Edition. London : Ed. Wiley. 2000;

SNEED, H. M. **Planning the reengineering of legacy systems**. IEEE Computer Society. p.24 January. 1995;

SNEED, H. **Encapsulation legacy software for use in client/server systems,** in Proc. of 3rd WCRE - Working Conference on Reverse Engineering. IEEE Computer Society. Monterey. p.104. 1996b;

SNEED, H. **Object-oriented COBOL recycling.** Proceedings of 3rd WCRE - Working Conference on Reverse Engineering. IEEE Computer Society. Monterey. p.169b. 1996;

SNEED, H. M. **Program interface reengineering for wrapping.** 4th WCRE - Working Conference on Reverse Engineering. IEEE Computer Society. Amsterdan. p.206. 1997;

SNEED, H. M. **Generation of stateless components from procedural programs for reuse in a distributed system**. Proceedings of 4th European CSMR - Conference Software Maintenance Reengineering, IEEE Press. Zürich. p. 183-188. March 2000;

SNEED, H. M. **Wrapping legacy COBOL programs behind an XML-interface.** 8th WCRE - Working Conference on Reverse Engineering. IEEE Computer Society. p. 189. 2001a;

SNEED, H. M. **Extracting business logic from existing COBOL programs as a basis for redevelopment**" 9th International Workshop on Program Comprehension (IWPC'01). p. 167. 2001b;

SNEED, H. M. **Using XML to integrate existing software systems into web**. 26 th Annual International Computer Software and Applications Conference (COMPSAC'02). 2002;

STROULIA, E. et al. **Reverse engineering legacy interfaces:** an interaction-driven approach**,** in 6th WCRE - Working Conference on Reverse Engineering. IEEE Computer Society. Atlanta, Geórgia. p. 292. 1999;

STROULIA, E. et al. **Constructing XML-spreaking wrappers for web applications:** towards an interoperating web, in 7th WCRE - Working Conference on Reverse Engineering.

IEEE Computer Society. 2000;

SUN. **Site oficial da Sun Microsystems**. Disponível em: <http://java.sun.com/products/jdk/1.1/docs/ guide/jni/index.html>. Acesso em: 12 jan. 2003;

TEIXEIRA, S.; PACHECO, X. **Borland Delphi 6 develorper's guide**. Disponível em: <http://www. borland.com>. Acesso em: 20 dez. 2001;

TOGETHER **Site oficial da Borland**. Disponível em: <http://www.borland.com.br>. Acesso em: 15 abr. 2003;

WERNER, C. M. L.; BRAGA, R. M. M. **Desenvolvimento baseado em componentes.** SBES - Simpósio Brasileiro de Engenharia de Software. Universidade Federal do Rio de Janeiro. 2000;