UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

# Estratégia de Migração de Aplicações Legadas Visuais (tipo WIMP) para o Ambiente Web

por

CLÁUDIO ROBERTO DE LIMA MARTINS

Dissertação submetida à avaliação como
requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Profa. Dra. Ana Maria de Alencar Price
Orientadora

Porto Alegre, abril de 2003.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Martins, Cláudio Roberto de Lima

Estratégia de Migração de Aplicações Legadas Visuais (tipo WIMP) para o Ambiente Web / por Cláudio Roberto de Lima Martins. – Porto Alegre: PPGC da UFRGS, 2003.

104f.: il.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2003. Orientadora: Price, Ana Maria de Alencar.

1. Migração de sistemas legados. 2. Processo de reengenharia. 3. Reengenharia de *software*. 4. Engenharia reversa. 5. Metamodelos. 6. Aplicações legadas cliente/servidor. 7. Aplicações web. I. Price, Ana Maria de Alencar. II. Título.

*Dedico este trabalho aos meus pais, “Dona” Selma e “Seu” Lopes.*
*A minha querida Lia (in memoriam).*

# Agradecimentos

Quero, em primeiro lugar, agradecer ao Ser Superior, independente da denominação religiosa adotada, que nos norteia e fortalece.

Em seguida, aos meus pais que pacientemente construíram e se dedicam a uma família maravilhosa. Aos meus irmãos Mauro e João. Em especial à minha irmã Simone, dedico o título (de mestre) a ela.

À Professora Ana Price pela orientação dedicada nestes últimos dois anos. Obrigado pela troca de experiências e na liberdade concedida na escolha do tema da dissertação.

Ao Professor Marcelo Pimenta, um amigo acima de tudo, pela participação na elaboração dos artigos (todos publicados) e na motivação positiva nas horas difíceis das incertezas.

Aos amigos Alécio e Bruno que me deram o suporte logístico em Porto Alegre. Valeu a convivência sob o mesmo teto e pelas conversas saudáveis que tivemos.

Agradecimentos às instituições que propiciaram a minha participação no mestrado. A UFPA e CAPES pela iniciativa neste projeto. A UFRGS, em especial à Professora Carla pela organização e dedicação no andamento do curso. E, sobretudo, ao CEFET/PA, instituição que apostou e aposta na capacitação do seu corpo docente. Essas instituições provam que a educação pública e de excelência pode ser uma realidade em nosso país.

Às lembranças pessoais, quero dedicar aos amigos da Cinbesa e do CEFET/PA, que são muitos. Pelas conversas sempre interessantes e a pela boa convivência durante estes últimos anos.

Obrigado a todos, anonimamente lembrados (para não cometer injustiças), que participaram direta ou indiretamente da evolução de minha vida pessoal e profissional.

# Sumário

# Lista de Abreviaturas

| | |
|---|---|
| CASE | Computer-Aided Software Engineering |
| DER | Diagrama Entidade-Relacionamento |
| DLL | Dynamic Link Libraries |
| DP | Decision Point |
| HTML | Hypertext Markup Language |
| IU | Interface de Usuário |
| MCS | Modelo Conceitual do Sistema |
| MORPH | Model Oriented Reengineering Process for Human-Computer Interface |
| OMG | Object Management Group |
| OO | Orientação a Objetos |
| RAD | Rapid Application Development |
| SQL | Structured Query Language |
| UC/Re | Use Case para Reengenharia |
| UML | Unified Modeling Language |
| VB | Visual Basic |
| W3C | World Wide Web Consortium |
| WAE | Web Application Extension |
| WIMP | Window, Icon, Menu, Pointing device (ou Pull-down menu) |
| WWW | World Wide Web |
| XMI | XML Metadata Interchange |
| XML | Extended Markup Language |

# Lista de Figuras

# Lista de Tabelas

# Resumo

O sucesso da Internet como plataforma de distribuição de sistemas de informação encoraja organizações a disponibilizar serviços presentes em seus sistemas legados nesse ambiente. Uma parte desses sistemas foi desenvolvida na fase inicial do desenvolvimento das aplicações cliente/servidor para banco de dados, usando ambientes visuais com interfaces gráficas tipo WIMP, implementadas sob o paradigma procedimental/estruturado, baseado em objetos e eventos. Como conseqüência, produziu-se sistemas legados difíceis de manter, evoluir e adaptar a novas tecnologias e arquiteturas, pois os projetos desenvolvidos não seguiam, na maioria das vezes, os bons preceitos e práticas modernas defendidas na Engenharia de Software.

O objetivo deste trabalho é propor uma metodologia para migrar sistemas legados com as características citadas acima para a plataforma Web. O processo de migração proposto destaca duas estratégias: a elaboração de modelos de classes conceituais da aplicação e o tratamento dado à interface do usuário, para serem utilizados na reconstrução de uma nova aplicação. O processo é baseado em técnicas e métodos de engenharia reversa, que visa obter abstrações por meio de análise estática e dinâmica da aplicação. Na análise dinâmica, destaca-se o mecanismo para recuperar aspectos dos requisitos funcionais do sistema legado e representá-los na ferramenta denominada UC/Re (*Use Case* para Reengenharia). Todos os artefatos gerados durante o processo podem ser armazenados em um repositório, representando os metamodelos construídos na metodologia. Para delimitar e exemplificar o processo, escolheu-se como domínio de linguagem de programação do *software* legado, o ambiente Delphi (sob a linguagem Object Pascal).

É proposto também um ambiente CASE, no qual é descrito o funcionamento de um protótipo que automatiza grande parte das funcionalidades discutidas nas etapas do processo. Algumas ferramentas desenvolvidas por terceiros são empregadas na redocumentação do sistema legado e na elaboração dos modelos UML do novo sistema. Um estudo de caso, apresentando uma funcionalidade específica de um sistema desenvolvido em Delphi, no paradigma procedimental, é usado para demonstrar o protótipo e serve de exemplo para a validação do processo. Como resultado do processo usando o protótipo, obtém-se o modelo de classes conceituais da nova aplicação no formato XMI (formato padrão para exportação de modelos UML), e gabaritos de páginas em HTML, representando os componentes visuais da interface original na plataforma Web.

**Palavras-chave:** Migração de sistemas legados; processo de reengenharia; reengenharia de *software*; engenharia reversa; metamodelos; aplicações legadas cliente/servidor; aplicações web.

**TITLE**: "A MIGRATION STRATEGY OF LEGACY VISUAL APPLICATIONS (WIMP) TO WEB ENVIRONMENT"

# Abstract

The success of the Internet as a distribution platform of information systems is challenging organizations in deployment their services (legacy systems) on that environment. A subclass of those information systems had been implemented at the initial development phase of the client/server database applications: the ones using visual environment with WIMP graphic interfaces and being developed under the procedural paradigm, based on event and object programming. The result made legacy systems hard to maintain, to evolve and to get profit from advanced technology and new architectures because these projects couldn't be adapted to the modern practices advocated by Software Engineering disciplines.

The goal of this work is to propose a methodology for migrating legacy systems comprising the features mentioned above to the Web environment. The migration process highlights two strategies: the building of conceptual models of the application under analysis and the treatment given to user interfaces in the rebuilding of a new application. The process is based on techniques and methods of reverse engineering, getting abstractions by means of static and dynamic analysis of the application. On dynamic analysis, it is worthwhile to mention a mechanism for recovering functional requirements of the legacy system and a way of representing them by the tool called UC/Re (Use Cases for Reengineering). All the components generated during the process will be stored in a repository, representing the metamodels built through the methodology. For defining and exemplifying the process, it has been chosen the Delphi environment (using Object Pascal language) as a programming language domain of the legacy software.

Also, a CASE environment has been proposed, through which the operation of a prototype that automates great part of the functionalities argued in the stages of the process is described. Some tools developed by third part are used in the new documentation of the legacy system and on the elaboration of the UML models of the new system. A case study presenting a specific functionality of a system developed in Delphi, under the procedural paradigm, is used to demonstrate the prototype and as a validation instance of the process. At the end of the prototype processing we get a conceptual class model of the new application in the XMI format (standard format for exporting UML models), and templates of HTML pages, representing the visual components of the original interface at the Web environment.

**Keywords:** Legacy system migration; reengineering process; software reengineering; reverse engineering; metamodels; client/server legacy applications; web applications.

# 1 Introdução

Hoje, a revolução que a Internet trouxe e os serviços advindos desta tecnologia, propicia aos desenvolvedores de *software* um novo campo de atuação. A Web ou WWW (World Wide Web), por exemplo, que inicialmente servia como meio de distribuição de simples páginas de conteúdo estático, transformou-se em uma nova plataforma de distribuição de aplicações e acesso a conteúdo dinâmico, usando novos paradigmas advindos da arquitetura cliente-servidor e sistemas distribuídos.

Uma aplicação baseada na Web é um sistema de *software* executado em um ambiente Web, seja na própria rede mundial (Internet/*World Wide Web*) ou nas variantes das redes mais fechadas (*intranets* ou *extranets*) (CONALLEM, 2000). Com a infraestrutura que a Internet fornece, muitas organizações buscam desenvolver e migrar seus sistemas de informação para este ambiente. Siglas e nomenclaturas são criadas para disseminar este fenômeno, conhecidas por *e-commerce*, *e-government*, *e-banking*, *e-learning*, e tantas outras "modas" cunhadas pela inicial "e-". Alguns destes sistemas estão confinados em grandes *mainframes*, ou desenvolvidos em arquiteturas pouco flexíveis, como a cliente-servidor em duas camadas, em que a lógica de negócio e a interface de apresentação estão monoliticamente construídas nos programas clientes. Esses sistemas são conhecidos como *sistemas legados*, não apenas pelas características arquitetônicas, mas pela dificuldade de mantê-los.

Este trabalho visa apresentar uma metodologia de migração de sistemas de informação cliente-servidor para o ambiente Web. O enfoque maior é dado à construção dos modelos de projeto derivados do sistema origem, que servirão de base para a implementação em um paradigma orientado a objetos, sobre uma nova plataforma, que *a priori* é a Web.

## 1.1 Motivação

Com o sucesso da Internet muitas organizações estão buscando atualizar seus sistemas de informação legados. Uma categoria desses sistemas foi desenvolvida na década de 90, quando se iniciou o processo conhecido por "*downsizing*", que defendia a substituição completa dos sistemas no ambiente de *mainframe* (centralizados), para sistemas em um modelo descentralizado utilizando a arquitetura cliente/servidor. Esta decisão acarretou não somente a troca de ambiente operacional, mas uma nova forma de desenvolver os *software*s aplicativos.

A maturidade do paradigma orientado a objetos (doravante abreviado OO) induziu a adoção de linguagens implementadas em ambientes visuais conhecidos como RAD (*Rapid Application Development*), que ofereciam além das facilidades de um ambiente de desenvolvimento integrado[*], recursos OO em forma de componentes pré-definidos. Essas ferramentas RAD, pelas facilidades que ofereciam, induziam a desenvolver o *software* de forma *ad hoc*, usando técnicas e métodos úteis para a prototipagem e projeto de pequenos sistemas. Além disso, a geração que iniciou o desenvolvimento de aplicações nesse paradigma, aqui chamadas de aplicações WIMP[**],

[*] Os IDEs (*Integrated Development Environment*) como Visual Basic (VB), da Microsoft, PowerBuilder, da Sybase, Visual Object (VO), da Computer Associates, e Delphi, da Borland. De todos, somente Delphi e VB prevaleceram no mercado.

[**] WIMP é um acrônimo para "***W****indow*, ***I****con*, ***M****enu*, ***P****ointing device*", ambiente de interface gráfica para usuário presente em computadores pessoais baseados em sistemas executados em Windows e Macintosh, por exemplo.

continuava fortemente ligada à cultura procedimental e estruturada da geração passada e culminou com uma subutilização do potencial da orientação a objetos. Assim, os recursos OO usados eram aqueles que o ambiente de programação disponibilizava de pronto: os componentes visuais e os objetos de acesso a dados. O restante era codificado procedimentalmente em métodos para tratamento de eventos, sejam de interfaces gráficas (em sua maioria) sejam de tratamento de exceções (erros de execução e manipulação dos objetos de dados), sem aproveitar adequadamente os mecanismos poderosos da programação OO como herança, polimorfismo, encapsulamento, delegação, troca de mensagens, etc. Como conseqüência desta forma "híbrida" de programar (código procedimental em um ambiente OO), produziu-se sistemas difíceis de manter, evoluir e adaptar a novas tecnologias e arquiteturas, devido, sobretudo, às seguintes características:

a) **manutenção e reuso prejudicados**: conseqüência da mistura da lógica de negócio com a lógica visual de interface no mesmo código fonte, com replicação freqüente de código (problema conhecido como "clonagem" de código);
b) **não uso de técnicas de ocultação de informação**: os programas acessam diretamente os dados armazenados nas tabelas relacionais, através de propriedades e métodos dos componentes de dados, ou por meio de código SQL embutido na aplicação;
c) **incentivo a prototipagem baseada em objetos**: o uso indiscriminado e facilitado dos componentes fornecidos pelo ambientes/linguagens visuais, desencorajava a programação OO. Algumas vezes, a própria linguagem não permitia a possibilidade de criação de classes, como no caso do VB;
d) **fraca decomposição funcional:** módulos e unidades funcionais centradas na interface de usuário, em janelas gráficas na forma de formulários, quadros e caixas de diálogo, em vez de classes, componentes e pacotes. A pobre divisão modular traz dificuldades para isolar componentes da interface de usuário e de domínio, devido ao forte acoplamento funcional e de dados;
e) **desempenho inferior no acesso aos dados**: devido à arquitetura em duas camadas de *software* em cliente e servidor (aplicação e banco de dados, respectivamente), normalmente é necessário implantar, no lado cliente, um *middleware* de acesso aos dados, com desempenho nem sempre satisfatório;
f) **dificuldades na distribuição da aplicação**: qualquer modificação de código fonte ou de versão em algum componente, acarreta a reinstalação física do aplicativo em diversas máquinas clientes, com uso de arquivos do tipo executáveis (DLL e EXE, por exemplo). Este modelo é conhecido como "*fat client*".

Entretanto, com a consolidação da filosofia de padrões abertos e sistemas distribuídos confirmados pela revolução que a Internet possibilitou, o modelo cliente/servidor descrito foi e está sendo revisto, não apenas do ponto de vista da arquitetura, mas sobretudo do ponto de vista do custo total de propriedade (TCO – *Total Cost Ownership*), que se apresenta bastante elevado. Portanto, a mudança do paradigma de desenvolvimento e de construção de *software* foi uma conseqüência natural em face da evolução tecnológica ocorrida nestes últimos anos.

## 1.2 Objetivos

O presente trabalho propõe uma metodologia para migrar sistemas com as características anteriormente citadas, para um paradigma OO, em uma arquitetura em três camadas, mais especificamente, sistemas ambientados na Web. O processo de reengenharia usado é baseado em modelos de alto nível de abstração, em vez de conversão de código em baixo nível de abstração. O mecanismo de migração através de modelos e metaclasses é a solução para a transliteração de código, impraticável quando não há similaridade de arquitetura e de linguagens de codificação correspondentes nos sistemas (GALL *et al*., 1995), (TEREKHOV; VERHOEF, 2000). No caso de uma aplicação na plataforma Web, diversos tipos de linguagens e tecnologias estão presentes, aumentando a complexidade da reengenharia direta. Observa-se, na Figura 1.1, o atalho encontrado para a solução do problema de mapeamento da migração entre as aplicações, que permite:

a) total reuso dos dados, tanto do esquema lógico quanto do conceitual; e,
b) a evolução na construção dos modelos de projeto por meio de metamodelos, partindo dos componentes de implementação do sistema origem, passando por mapeamentos de modelos de mais alto nível, até alcançar o nível de implementação em uma arquitetura de *software* multi-camadas no sistema alvo.

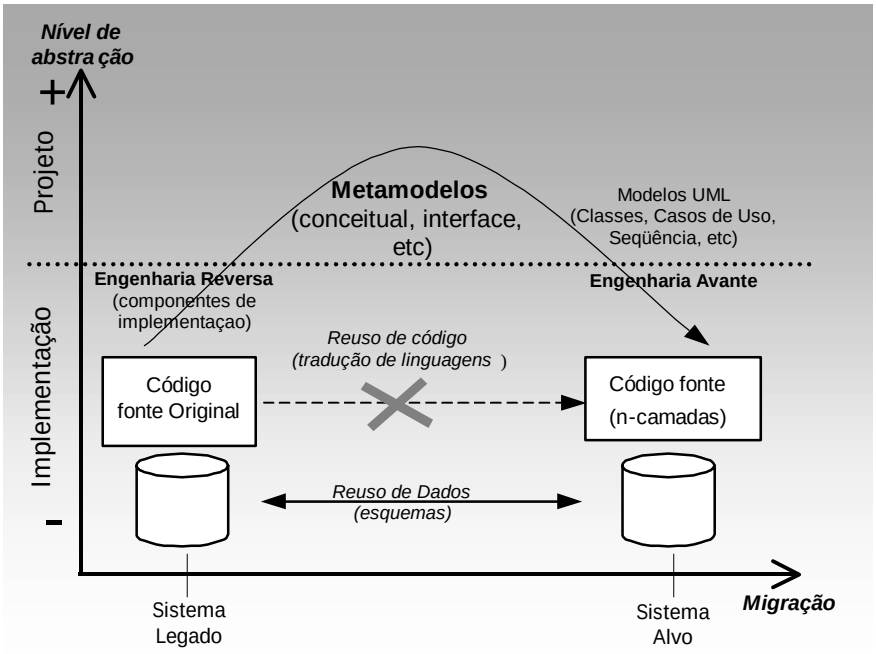



FIGURA 1.1 – Relação de reengenharia entre processo de migração e nível de abstração.

No decorrer do processo, metamodelos são construídos a fim de auxiliar a construção de modelos de projeto para o novo sistema. Os metamodelos são modelos de representação para os componentes obtidos por engenharia reversa do sistema origem e são armazenados em um repositório. A partir desse repositório será possível construir alguns modelos para o projeto de migração como, por exemplo, o de interface de apresentação do usuário, o de casos de uso para descrever as funcionalidades e o Modelo Conceitual do Sistema (MCS), com as classes de negócio envolvidas. Outro benefício com o uso de metamodelos e de repositório é a possibilidade de gerar os

modelos em um formato padrão, como o XMI (XML METADATA INTERCHANGE, 2003), que é aceito por ferramentas CASE de modelagem UML (UNIFIED MODELING LANGUAGE, 2003) que seguem esse padrão.

Apesar da abordagem proposta não prover um processo completo de migração[*], ela auxilia nas etapas mais importantes de projeto do novo sistema, ao permitir a construção de modelos de domínio da aplicação e o projeto da interface, necessários para uma etapa de engenharia avante. Vários passos são omitidos para não prejudicar os objetivos da dissertação. Entre esses passos, cita-se aspectos ligados ao estudo de viabilidade do projeto, planejamento, administração do fluxo de trabalho entre pessoas e equipes, (re)documentação e testes. A ênfase do trabalho se concentra na elaboração dos modelos de alto nível do sistema legado e na capacidade de estruturar, de forma simples e eficaz, os artefatos produzidos durante as etapas do processo. Ao final desse processo, o (re)projetista terá condições de desenvolver o sistema em uma nova plataforma, seguindo os bons preceitos do paradigma OO.

## 1.3 Organização do Texto

A estrutura da dissertação está organizada da seguinte forma:

**Capítulo 2**: apresenta os fundamentos, conceitos e terminologias utilizadas no decorrer do texto, além de apresentar uma visão geral tanto dos componentes presentes em uma aplicação legada, quanto daqueles presentes no ambiente Web, alvo do processo de migração.

**Capítulo 3**: discute alguns trabalhos relacionados à engenharia reversa e reengenharia, além de pesquisas que tratam formas de representação de *software*.

**Capítulo 4**: apresenta a proposta da metodologia onde se descreve o processo para a concepção dos metamodelos e discute-se as diretrizes para construção de modelos de projeto de migração.

**Capítulo 5**: é proposto um ambiente de apoio ao processo de reengenharia, com a definição de um protótipo que busca automatizar alguns passos da metodologia. Para demonstrar o ambiente e a ferramenta proposta, um estudo de caso é apresentado, no qual o processo é aplicado em um caso de uso específico de uma aplicação desenvolvida em Delphi.

Comentários finais, contribuições e trabalhos futuros são apresentados no **Capítulo 6**.

[*] Um processo completo de reengenharia e de migração trata de aspectos mais amplos relacionados à organização que utiliza o software a ser reconstruído. Diz respeito, também, ao planejamento e gerência do projeto de migração, enfim a todo suporte coordenado de atividades paralelas à ação de reimplementação do sistema.

# 2 Visão Geral e Fundamentos

Neste capítulo são apresentados os fundamentos e principais conceitos e a terminologia a serem utilizados ao longo da dissertação. Na seção 2.1, discute-se o tema reengenharia de *software* sob vários aspectos como engenharia reversa e uma classificação quanto aos tipos de reengenharia encontrados na área de *software*. Na seção 2.2, apresenta-se o tipo de aplicação legada que é tratado na proposta, com as considerações das características de arquitetura lógica. Na seção 2.3, o ambiente Delphi é apresentado uma vez que a proposta trata desse domínio de linguagem no processo de reengenharia, além de exemplos de código serem citados nesse ambiente de desenvolvimento. Na seção 2.4, vê-se os principais componentes presentes na aplicação alvo da migração, discutindo-se o ambiente e o desenvolvimento para a Web. As considerações finais estão descritas na Seção 2.5.

## 2.1 Reengenharia

A reengenharia de *software* examina sistemas de informação e aplicações com o objetivo de reestruturá-los ou reconstruí-los, de modo que apresentem mais qualidade (PRESSMAN, 2001). Combina vários processos e técnicas incluindo engenharia reversa e engenharia avante, resultando na criação de um novo sistema. Normalmente, a reengenharia de *software* engloba atividades que buscam melhorar a compreensão e a qualidade do *software* como um todo, aumentando a manutenibilidade, reusabilidade e evolução de um sistema.

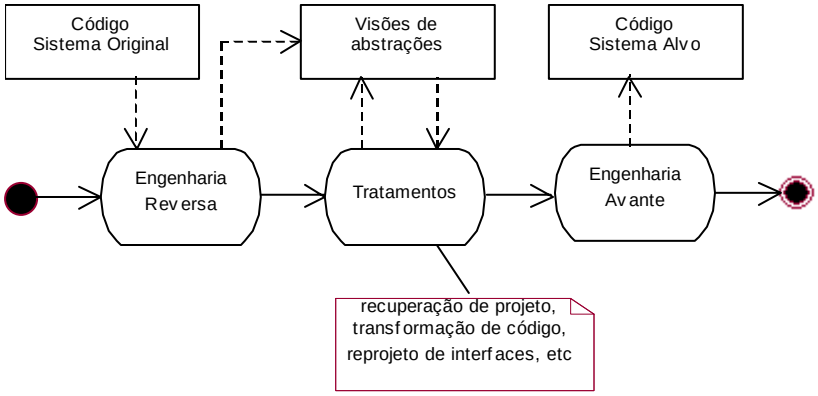



FIGURA 2.1 – Um processo genérico de reengenharia.

Um processo genérico de reengenharia de *software* pode ser visto no diagrama de atividades da Figura 2.1. A engenharia reversa cria visões de abstrações maior do sistema, com técnicas adequadas para tratar essas abstrações, seguida por atividades de engenharia avante para gerar o sistema na nova forma. Segundo Chikofsky e Cross (1990), a presença de um passo de engenharia reversa distingue reengenharia de reestruturação; reestruturação consiste em transformar um artefato de uma forma para outra, no mesmo nível relativo de abstração. Os questionamentos e dúvidas acerca das definições, como reengenharia e reestruturação, fizeram com que os autores organizassem alguns termos para a disciplina de reengenharia de sistemas, padronizando as terminologias utilizadas até então. São eles:

a) **Engenharia reversa** (*Reverse Engineering*) é o processo de analisar o sistema existente, identificando seus componentes representando-os em um nível mais alto de abstração. Possui duas categorias: redocumentação e recuperação de projeto.

b) **Redocumentação** (*Redocumentation*), também conhecida como *visualização de código*, é a criação ou revisão de uma representação da abstração semântica do sistema.
c) **Recuperação do projeto** (*Design recovery*), também chamada de *entendimento de programa*, é a adição do domínio de conhecimento e informações externas para identificar no sistema abstrações de alto nível, além daquelas obtidas diretamente pelo exame do sistema.
d) **Reestruturação** (*Restructuring*) é a transformação de uma forma de representação para outra no mesmo nível de abstração, preservando o comportamento externo do sistema (funcionalidade e semântica).
e) **Engenharia avante** *(Forward Engineering)*, ou também conhecido por engenharia direta, progressiva ou de produção, refere-se ao tradicional processo de desenvolvimento de *software*. É um processo que requer a tradução de abstrações, modelo lógico e projeto para uma implementação física (codificação). Quando o processo é automatizado é chamado de *geração*.
f) **Reengenharia** (*Reengineering*), também conhecida como *renovação* ou *recuperação*, é a análise e alteração do sistema para reconstruí-lo em uma nova forma e a implementação subseqüente dessa nova forma. Envolve duas grandes etapas: engenharia reversa e a engenharia avante, e pode incluir reestruturação, reprojeto e a reimplementação do *software*.

Os autores também relatam a importância da engenharia reversa combinada com o uso de ferramentas de automatização, para aumentar a compreensão, manutenção e recuperação de documentação na reconstrução de um novo sistema. Para essas ferramentas terem o nível de excelência esperado, seis objetivos devem ser atingidos: (1) lidar com a complexidade relativa ao volume de informação recuperada, classificando essas informações em graus de relevância; (2) fornecer alternativas para a criação de visões de representação em forma de diagramas; (3) recuperar informações perdidas, decorrentes da inconsistência da documentação que não refletem as modificações introduzidas na manutenção do sistema; (4) detectar efeitos colaterais provocados pela modificação do código; (5) sintetizar as visões e modelos em níveis de maior abstração; (6) facilitar o reuso pela capacidade de identificar possíveis componentes do sistema candidatos ao reuso.

Segundo Pressman (2001), a reengenharia de *software* abrange uma série de atividades que inclui análise de inventário, reestruturação de documentos, engenharia reversa, reestruturação de programas e de dados, e engenharia avante. A *análise de inventário* permite que a organização avalie cada aplicação sistematicamente, com o objetivo de determinar quais são candidatas à reengenharia. A *reestruturação de documentos* cria um arcabouço de documentação necessário para o suporte de longo prazo de uma aplicação. *Engenharia reversa* é o processo de análise de um programa, num esforço de extrair informação de projeto de dados, arquitetural e procedimental. *Reestruturação de programas* e de *dados* realiza a atualização do código e dos dados em uma nova arquitetura ou estrutura, seguindo princípios mais atuais, porém conservando a mesma funcionalidade. *Engenharia avante* reconstrói um programa usando práticas modernas de engenharia de *software* e informação adquirida durante a engenharia reversa. A Figura 2.2 apresenta o modelo cíclico para o processo de reengenharia discutido por Pressman (2001), em que cada uma das atividades pode ser revisada. Para qualquer ciclo particular o processo pode terminar após qualquer uma destas atividades.

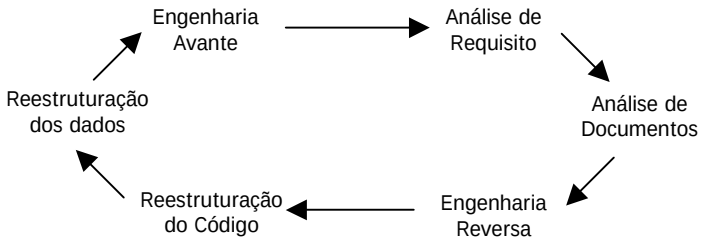



FIGURA 2.2 – Modelo do processo de reengenharia de *software* proposto por Pressman (2001).

Apesar de encontrarmos na literatura diversas definições sobre termos ligados à reengenharia, nesta dissertação os termos *migração* e *reengenharia* terão o mesmo significado. Migração é um termo utilizado para enfatizar a atividade de mudança da plataforma tecnológica de hardware e *software* do sistema legado, porém preservando os requisitos funcionais do sistema; reengenharia enfatiza o processo de *redesenvolvimento*[*] do sistema, englobando algum tipo de reestruturação, engenharia reversa e avante, reutilização e migração.

Outras técnicas ligadas à reengenharia, especificamente no campo da reestruturação de código, diz respeito à *refatoração* e *segmentação*.

A *refatoração* (*refactoring*) é a técnica que busca melhorar a compreensão e a manipulação do código fonte, porém conservando a funcionalidade da aplicação. As mudanças ocorrem na estrutura interna do programa e são realizadas em pequenos passos. No contexto do paradigma OO, a refatoração ocorre, por exemplo, com renomeação de métodos, transferência de um atributo de uma classe para outra, unificação de dois métodos semelhantes em uma superclasse, e muitos outros exemplos que buscam seguir boas práticas de projeto OO (FOWLER *et al.*, 1999).

A *segmentação* é uma técnica de reengenharia para reestruturar o código de sistemas legados, implementados em um paradigma procedimental[**], adicionando ao sistema características orientadas a objetos, porém preservando as funcionalidades. A documentação obtida com a engenharia reversa é utilizada para auxiliar o engenheiro de *software* a reorganizar as linhas de código para adição dessas características (PENTEADO *et al.*, 1999). É uma técnica que pode ser utilizada como passo inicial em um processo de reengenharia para mudança de paradigma. Consiste na identificação de procedimentos anômalos, isto é, aqueles com baixa coesão e que manipulam mais de uma estrutura/objetos de dados. Em seguida, estes procedimentos são segmentados em blocos de procedimentos mais coesos, que darão origem, em um processo de reengenharia OO, em futuros métodos das classes candidatas.

O *encapsulamento*, ou empacotamento (originalmente, "*wrapping*"), é mais uma técnica de reengenharia que em vez de reestruturar o sistema, propõe ocultar o velho código dentro de uma nova camada. Por exemplo, um código em uma linguagem procedural poderia ser encapsulado dentro de uma camada de objetos. Para o exterior, o código é orientado a objetos, mas os atributos e métodos dos objetos só delegam o trabalho às variáveis e rotinas do código procedural. Normalmente, essa técnica é usada

[*] O redesenvolvimento prevê, também, a incorporação de novas funcionalidades ao sistema que está sendo migrado.

[**] Embora o termo "procedimental" (com o significado de "*baseado em procedimentos*") esteja corretamente empregado, ao longo do texto será empregado e substituído pelo vocábulo "*procedural*", permissividade encontrada comumente em artigos da área.

como uma etapa intermediária em um processo incremental de migração, em que a coexistência de dois sistemas (o legado e o "novo", pelo menos na interface de usuário), é possível. No passo seguinte, tanto o sistema "empacotado" quanto o legado são descontinuados e substituídos por um sistema que segue tecnologias e paradigmas mais atuais.

Na Tabela 2.1, observa-se um resumo dos processos de engenharia reversa e reengenharia, destacando técnicas, ferramentas e formas de soluções para cada técnica empregada.

TABELA 2.1 – Resumo das características dos processos de Engenharia Reversa e Reengenharia.

| **Processo** | **Técnicas** | **Ferramentas/soluções** |
|---|---|---|
| Engenharia Reversa | Redocumentação | ▪ formatadores de código (*pretty-printer*);<br>▪ geradores de diagramas;<br>▪ geradores de listagem de referência-cruzada. |
| | Recuperação de projeto | ▪ *software* de métricas;<br>▪ ferramentas de visualização (*browsers)*;<br>▪ analisadores estáticos (*parsing*);<br>▪ analisadores dinâmicos (*trace* e *debugging*). |
| Reengenharia | Reestruturação de código | ▪ conversores de código (desestruturado para estruturado);<br>▪ tradutores de código fonte de uma linguagem em outra. |
| | Reestruturação de dados | ▪ integração e centralização de banco de dados;<br>▪ unificação e normalização de inconsistências;<br>▪ atualização de modelo de dados. |
| | Refatoração (*refactoring*) | ▪ *renomeação* e mudança de métodos, classes, atributos, etc. |
| | Segmentação | ▪ máquinas de transformação *intra-domínio*[*];<br>▪ identificação de padrões sintáticos no código fonte.<br>▪ reestruturação de código por meio da manipulação de variáveis (*slicing*) e *remodularização*. |
| | Enpacotamento (*wrapping*) | ▪ Objetos distribuídos (CORBA, DCOM, RMI, etc). |

## 2.2 Aplicações Legadas do Tipo WIMP

Uma aplicação legada do tipo WIMP é um sistema de informação implementado em um ambiente de interface gráfica de usuário (GUI), instalado de forma *stand-alone* ou cliente-servidor acessando uma base de dados relacional. De um modo geral, sistemas legados possuem documentação precária ou inexistente e passaram, ao longo dos anos, por manutenções realizadas por diferentes pessoas, sem seguirem boas práticas de engenharia de *software*. No contexto histórico, tais aplicações surgiram quando da criação dos ambientes visuais de desenvolvimento rápido conhecidos como RAD (ver Seção 1.1), que incentivavam o desenvolvimento de

[*] Uma *máquina transformadora intra-domínio* é uma ferramenta que contém regras de transformações que mapeiam a sintaxe e a semântica dos comandos de uma linguagem de programação para um código fonte segmentado, implementado na mesma linguagem, mas organizado segundo os princípios da orientação a objetos. Um exemplo é o sistema transformacional da ferramenta Draco-PUC (LEITE *et al.*, 1994).

aplicações WIMP no paradigma baseado em objetos (os disponibilizados pelos ambientes) e orientados a eventos.

Na Figura 2.3 é proposta uma estrutura simplificada para esse tipo de aplicação, na qual se observam os principais componentes, relevantes ao tema da dissertação. O estereótipo "implClass", semelhante ao estereótipo "*implementation class*" (UNIFIED MODELING LANGUAGE, 2003), é uma classe que fornece uma implementação física (componente), incluindo um conjunto de atributos, associações com outras classes e métodos para as operações. A seguir, são descritas as funcionalidades de cada componente da estrutura.

**Aplicação** – compreende todos os elementos físicos do sistema com pelo menos uma unidade modular. Normalmente, um projeto de aplicação é armazenado em uma estrutura de diretório contendo os arquivos de códigos fonte e executável.

**Módulo** – representa uma unidade física de código, como um arquivo fonte, que pode referenciar outros módulos (por meio de chamadas do tipo *uses*, *include* ou *import*, por exemplo), além de conter rotinas, *containers* visuais e objetos de dados DataSet. Em Delphi, por exemplo, um módulo é identificado por uma *unit*, que pode conter estruturas do tipo *class*, *functions* ou *procedures*; em VB, temos o conceito de *module*, que pode ser do tipo *class* ou geral (para os procedimentos do tipo *sub* e *function*); em Java, temos as unidades modulares definidas em *class*.

**DataSet** – é todo componente de manipulação de uma tabela, através de métodos apropriados para incluir, excluir e alterar registros de dados, acesso a atributos e outros recursos. Como é um objeto, deve ser declarado como uma variável objeto possuindo algumas operações e propriedades. A relação de dependência a um banco de dados é realizada por um mecanismo de acesso, conhecido também por *middleware*. Eventos que ocorram no âmbito da aplicação devem ser tratados por manipuladores, codificados na própria aplicação cliente. Um exemplo de evento desse tipo acontece quando da mudança de estado do registro de dados corrente, de inativo para o modo de edição, que pode ser tratado por meio de um procedimento.

**Contêiner Visual** – componente de interface visual contendo outros componentes visuais (controles visuais), inclusive outros contêineres visuais. Em um ambiente WIMP são conhecidos por *widgets containers*, como formulários, *panels*, *frames* ou caixa de diálogo, por exemplo. Por ser um objeto, deve ser declarado como uma variável ou componente objeto possuindo algumas operações, propriedades e eventos.

**Controle Visual** – é um componente de apresentação visual de dados, que está acoplado direta ou indiretamente a um DataSet. Um acoplamento direto ocorre quando o controle tem acesso direto a um campo do DataSet; o indireto se faz quando um controle não tem vínculo com DataSet, porém o controle serve de intermediário para a apresentação e processamento de um campo do DataSet, operações realizadas por meio de atribuições entre o controle e o campo, e vice-versa. Em um ambiente WIMP, os controles visuais são conhecidos como *widgets* de controle (menus, botões de comando, *text-field*, *text-label* ou rótulos, etc.), isto é, objetos visuais com propriedades, operações e eventos associados. Geralmente, nos eventos disparados a partir destes controles é que se processa a lógica de negócio. Por exemplo, um controle *text-field* (associado a um campo/atributo de um DataSet) pode ter um evento chamado "*on_change*". Quando o valor do *text-field* muda, o evento "*on_change*" executa o código relativo a este evento, provavelmente, chamando um procedimento definido no módulo funcional.

**Mecanismo de acesso** – interface de acesso e conexão da aplicação com o banco de dados. Os ambientes de desenvolvimento normalmente fornecem bibliotecas de acesso a banco de dados como ODBC, ADO (Microsoft) e BDE (da Borland), que

devem ser implantados (instalados) junto com a aplicação cliente, para acesso em rede local a um servidor de banco de dados. Nessas bibliotecas, o programa cliente estabelece uma conexão assim que é carregado e envia comandos através desta conexão, normalmente, na linguagem SQL.

**Procedimento** – unidade funcional de código para executar uma rotina, uma função ou implementar um método de um objeto. Um procedimento pode invocar outro ou a si mesmo (procedimento recursivo).

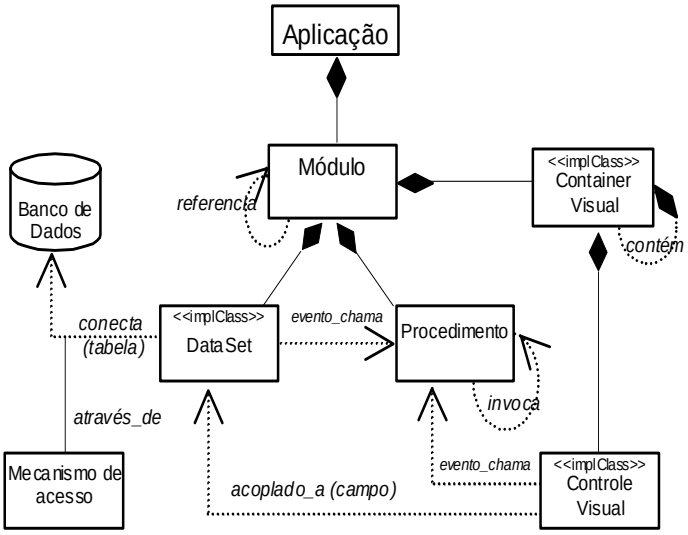



FIGURA 2.3 – Estrutura básica de uma aplicação WIMP cliente/servidor.

Na Tabela 2.2, vê-se a correspondência destes componentes para algumas linguagens e ambientes visuais disponíveis no mercado.

TABELA 2.2 – Correspondência dos componentes da arquitetura com linguagens/ambientes de programação visual.

| **Componente** | **Linguagens/Ambientes Visuais** | | |
|---|---|---|---|
| | **Object Pascal/Delphi** | **Visual Basic/Visual Studio** | **Java** |
| Módulo | *unit* | *module* | *class* |
| DataSet | TDataSet (TTable, TSql e outros components) | DataSet (RecordSet, Connection) | objetos JDBC |
| Container Visual | TForm, TPanel,etc. (VCL) | *Form* | *containers* em AWT e Swing (ex: Frame) |
| Controle Visual | TDataControls, TDBEdit e outros comp. (VCL) | *Controls* (Label, Text, etc.) | controles definidos em AWT/Swing |
| Mecanismo de acesso | BDE, ADO, etc. (TDatabase) | ODBC, ADO | JDBC |
| Procedimento | *function, procedure* | *function, sub* | método |

## 2.3 O Ambiente Delphi

O ambiente Delphi é discutido nesta seção, uma vez que a proposta trata desse domínio de linguagem no processo de reengenharia. Além disso, exemplos, com trechos de código na linguagem nativa do Delphi, *Object Pascal*, são referenciados ao longo da dissertação. Também, como motivação na escolha desse ambiente, o fato que Delphi está entre os ambientes RAD mais utilizados para o desenvolvimento de sistemas de

informação no Brasil, como comprovado em publicações em revistas como Informática Exame (PROGRAME, 1996) e Byte Brasil (SILVA; SANTOS, 1997).

Delphi surgiu em um momento em que outros ambientes de programação[*] eram lançados com o propósito de simplificar o desenvolvimento de *software* visual para o sistema operacional *Windows*. A linguagem base do ambiente é oriunda do *Object Pascal*, permitindo ao programador o desenvolvimento de *software* tanto no paradigma *procedural*/estruturado quanto no orientada/baseado em objetos. Outra característica importante é que, como qualquer linguagem que gera *software* "visual" (interface gráfica de usuário), é orientada a eventos.

Entre as várias características que fazem do ambiente Delphi um dos mais utilizados, podemos citar (CORNELL; STRAIN, 1995):

a) Aplicativos implementados em Delphi têm praticamente a mesma velocidade daqueles desenvolvidos em C ou C++;
b) Com o Delphi podem ser construídas DLLs (*Dynamic Link Libraries*);
c) Podem ser construídos objetos reutilizáveis seguindo os paradigmas da orientação a objetos.

Os programas construídos em Delphi são divididos em módulos de código fonte, chamados de *unit*. Cada programa inicia com um cabeçalho de especificação do nome do programa, seguido por uma cláusula opcional *uses*, depois por um estrutura de bloco de declarações e comandos. A cláusula *uses* lista as *units* que serão ligadas e dependentes ao programa fonte, semelhante a uma declaração *include*, em C, ou *import*, em Java. Na Tabela 2.3, observam-se os componentes estruturais (arquivos) presentes em uma aplicação Delphi.

TABELA 2.3 – Componentes presentes em uma aplicação Delphi

| **Tipo do arquivo** | **Definição do componente** | **Características de projeto** |
|---|---|---|
| ".pas" | contém o código fonte implementado com a linguagem *Object Pascal*. Cada *unit* Delphi está associada a um arquivo desse tipo. | modulo de código fonte;<br>permite a criação de biblioteca de compartilhamento entre programas (entre outras *units*);<br>permite distribuição de componentes entre desenvolvedores, sem necessitar do código fonte, apenas o objeto (.DCU). |
| ".dpr" | "**D**elphi **PR**oject", arquivo principal do projeto, com a lista de *units* usadas e a inicialização da aplicação. | representa o programa principal ("main") da aplicação. Uma aplicação de projeto (dpr) referencia as *units* que farão parte do *software* aplicativo gerado no ambiente. |
| ".dfm" | "**D**elphi **F**or**M**", contém as descrições dos controles do Form (interface do aplicativo) com suas respectivas propriedades. | arquivo contendo as informações do formulário (objeto *TForm*) vinculado a uma *unit*. |
| ".dcu" | "**D**elphi **C**ompile **U**nit", correspondente a uma *unit* compilada. | arquivo resultante da compilação da *unit* fonte (.PAS). |
| ".dof" | "Delphi Options File", arquivo com as opções do projeto. | As opções podem ser diretivas, diretórios e opções de compilação. |
| ".res" | "RESources", recursos do projeto. | São exemplos: ícones e *bitmaps* utilizados na aplicação gerada. |

[*] Como exemplo temos o Visual Basic, da Microsoft, Visual Objects, da Computer Associates e outros que ofereciam funcionalidades comuns e um ambiente integrado de desenvolvimento (editor, compilador, componentes e outras facilidades reunidas no mesmo conjunto de ferramentas).

No contexto deste trabalho, a recuperação dos componentes de uma aplicação implementada em Delphi, considera uma arquitetura em duas camadas: uma cliente e outra servidora. O lado cliente é a própria aplicação, com a interface de usuário, as regras do negócio e o mecanismo de acesso aos dados. O lado servidor está o gerenciador do banco de dados, responsável pelo armazenamento dos dados e de parte das regras, em forma de *stored-procedures* e *triggers*.

Do ponto de vista do ambiente Delphi, a partir da versão 5, o acesso e a manipulação dos dados se realiza através dos mecanismos chamados de BDE, ADO, Interbase, e Midas. Na grande maioria os sistemas implementados utilizam o BDE como o mecanismo de acesso a banco de dados relacionais (locais ou em máquinas remotas).

Os mecanismos citados abstraem a forma como os dados serão armazenados fisicamente, isto é, tanto podem estar em tabelas locais como em um gerenciador de banco de dados; as formas de manipulação dos atributos e dos métodos são os mesmos. Na aplicação, estes dados são controlados pela escolha de um componente *Dataset*.

A aplicação pode conter mais de um *Dataset*. Cada *Dataset* representa uma tabela lógica.

Ao usar o BDE para acesso aos dados, os seguintes componentes *Dataset* poderão ser utilizados:

a) **Table** (TTable): correspondem às tabelas armazenadas em um banco de dados remoto ou em arquivos locais.
b) **Query** (TQuery): *Queries* fornecem um mecanismo de manipulação dos dados através da SQL.
c) **Stored procedures** (TStoredProc): é um componente que reúne um conjunto de instruções SQL declaradas e armazenadas em um servidor de banco de dados.
d) **Nested datasets** (TNestedTable): representam os registros no banco de dados Oracle8, em forma de conjuntos (tabelas) aninhados.

Para exibir e atualizar os dados, através da interface gráfica com o usuário, alguns componentes visuais foram disponibilizados. São chamados de *Data Controls*, como DBText, DBGrid, DBEdit, DBNavigator, DBListbox, DBCombobox, entre outros. Estes controles de dados deverão estar inseridos em um *container*, em um formulário (TForm), por exemplo. Estão agrupados em: (a) controles que representam um campo (atributo) simples, por exemplo o DBText e o DBEdit; (b) controles que representam um conjunto de registros, tais como DBGrids e controles de grades; e, (c) controle de navegação, como o TDBNavigator, uma ferramenta visual que permite ao usuário navegar no *dataset*, isto é, permite mover o ponteiro dos registros do *dataset* em várias direções.

A ferramenta de desenvolvimento Delphi, desde sua primeira versão já se mostrou preocupada em distribuir a aplicação nas três camadas lógicas (apresentação, lógica, acesso a dados). Para implementar essa filosofia de trabalho, Delphi estabelece três categorias de componentes: componentes visuais, componentes de acesso à base de dados e componentes de ligação (ver os componentes visuais no ambiente Delphi na Figura 2.4).

Essa forma de trabalho, organizada em camadas distintas, permite também uma maior reutilização de código e portanto um aumento de produtividade na construção de aplicações através dos recursos do Delphi de criação de componentes e *templates* de telas. A seguir uma definição detalhada dos componentes de acesso a dados.

a) **Componentes visuais** - Componentes responsáveis pela interface com o usuário. Correspondem à camada de Apresentação discutida anteriormente.

Esses componentes estão localizados na página **Data Controls** da paleta de componentes do Delphi.

b) **Componentes de Acesso à base de dados** - Componentes responsáveis em criar toda a estrutura necessária para acessar e manipular o banco de dados. São os componentes encarregados em interfacear os serviços de gerenciamento e manipulação fornecidos pela base de dados. Esses componentes também possuem propriedades e eventos destinados a implementação da lógica de negócio na aplicação. Esses componentes estão localizados na página **Data Access** da paleta de componentes.

c) **Componente de ligação** - Componente responsável pela interface entre as duas camadas acima. Sua principal característica é tornar os componentes visuais independentes dos componentes de acesso, ou seja, a camada de Apresentação da Lógica do Negócio. Esse componente é o *TDataSource* que fica localizado também na página **Data Access** da paleta de componentes.

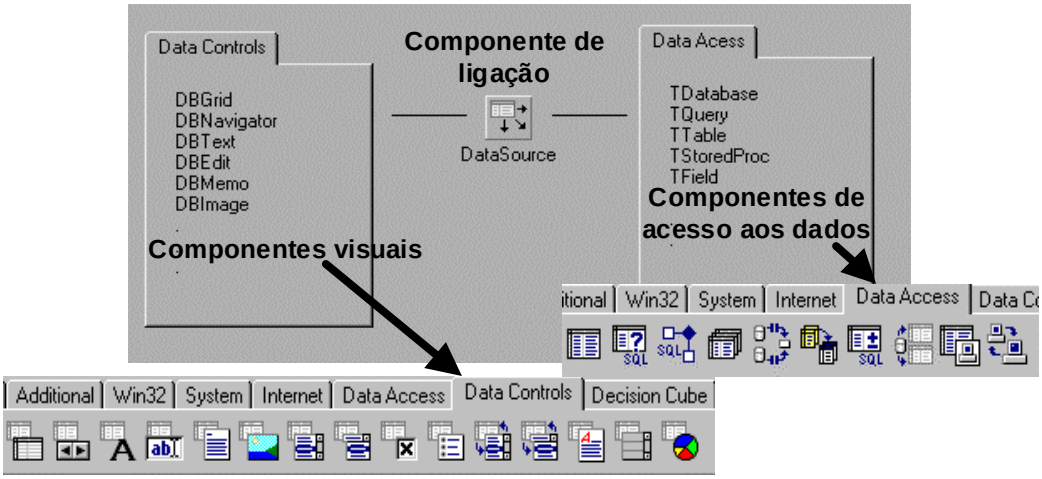



FIGURA 2.4 – Categorias de componentes do Delphi para acesso a base de dados.

## 2.4 Aplicações Web

Pode-se definir uma aplicação Web como uma aplicação de *software* que utiliza a Web como ambiente de execução. Aplicações Web envolvem *Websites* ou sistemas Web (CONALLEM, 2000). *Website* é a forma original de sistema hipermídia distribuído, criado por Tim Bernes-Lee, com o propósito de permitir pesquisa e acesso direto a documentos e informações publicadas em vários computadores que formam a Internet. Documentos, arquivos armazenados em um computador servidor, são acessados e visualizados através de um *software* chamado *browser*, instalado no computador cliente, utilizando-se da infraestrutura da Internet, através do protocolo HTTP (*Hyper Text Transfer Protocol*). A Figura 2.5 mostra a arquitetura básica de um *Website* (um projeto centrado em documento), onde um servidor Web recebe uma solicitação de um *browser*, localiza o documento em um sistema de arquivos local, e envia o documento de volta ao *browser*. É um sistema Web, sistema hipermídia onde recursos do sistema são interligados entre si. As ligações (*links*) são a forma de navegar entre os recursos do sistema. Estes recursos podem ser documentos textuais, podendo o sistema também distribuir áudio, vídeo e outros tipos de dados.

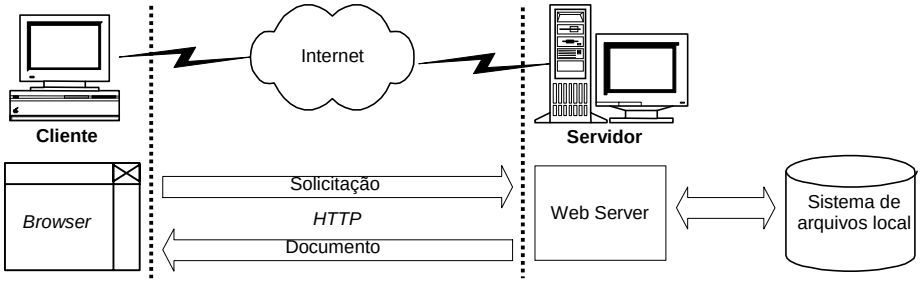



FIGURA 2.5 – Arquitetura básica de um Sistema Web ou *Website* (CONALLEM, 2000)

Uma aplicação Web amplia o conceito de *Website*, ao adicionar funcionalidade ao sistema. Em outras palavras, aplicação Web é um sistema Web que permite aos usuários executarem lógica de negócio com um *browser* Web. Uma aplicação Web deve ser entendida como uma forma de uso de *software* acessando dados persistentes através do serviço WWW (World-Wide Web), permitindo a construção dinâmica de páginas para manipular estes dados. Diferente de *Websites* estáticos, onde o conteúdo é um arquivo ou documento pré-formatado, através de um *designer* de páginas (usando um editor HTML, por exemplo), aplicativos Web devem construir "*on the fly"* o conteúdo em função da interação do usuário com as páginas, via *browser*. A arquitetura básica é mostrada na Figura 2.6.

### 2.4.1 Arquitetura de Funcionamento

A arquitetura de um *Website* dinâmico, apresentada na Figura 2.6, simplifica as peculiaridades encontradas na implementação do mecanismo de controle e gerência de execução (*App Engine*) de um aplicativo Web. A própria definição de uma aplicação para a Web implica que, no mínimo, há três componentes importantes na arquitetura de funcionamento: o *browser* cliente, o servidor Web e o servidor de aplicação. Ao servidor de aplicação é possível adicionar mais um elemento: o servidor de banco de dados, como repositório de dados da aplicação.

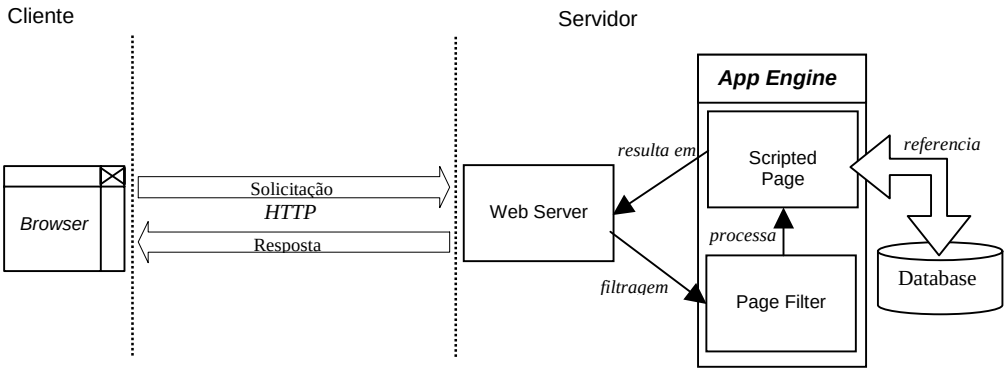



FIGURA 2.6 – Arquitetura de um *Website* dinâmico

É bastante diversificada a solução disponível para a distribuição das camadas de *software* de um sistema Web dinâmico. Uma solução de arquitetura representa um esquema de organização estrutural das camadas, fornecendo um conjunto de subsistemas distribuídos, interconectados e cumprindo responsabilidades bem definidas. Os três padrões de arquitetura mais comuns, segundo Conallem (2000), são:

a) Cliente-magro (*Thin Web Client*): usada mais freqüentemente em aplicações baseadas na Internet, na qual tem-se pouco controle sobre a configuração do cliente. O *browser* é o *software* cliente, seguindo os

padrões abertos recomendados pelo consórcio W3C para o HTML, em uma versão bastante enxuta. Toda a lógica do negócio é executada no servidor.

b) Cliente-gordo (*Thick Web Client*): é a arquitetura onde grande parte da lógica do negócio é executada na máquina cliente. Tipicamente, o cliente usa HTML dinâmico, applets Java, controles ActiveX, e outros *software*s adicionados ao *browser* para executar a lógica do negócio.

c) *Web Delivery*: a diferença em relação às duas anteriores é o suporte de comunicação entre cliente e servidor. Enquanto as outras arquiteturas utilizam basicamente o protocolo HTTP, em *Web Delivery* outros protocolos são usados, tais como IIOP (Internet Inter-ORB Protocol da OMG, para o CORBA), DCOM (Distributed COM, da Microsoft) e RMI/JRMP (Remote Method Invocation/Java Remote Method Protocol, para o ambiente Java) que são utilizados em sistemas de objetos distribuídos, onde o cliente se comunica diretamente com objetos localizados não apenas no servidor Web, mas distribuídos em outras máquinas. Esses protocolos permitem sobrepor o comportamento "sem-conexão", presente no protocolo HTTP. O *browser* atua como um despachante e um dispositivo contêiner para um sistema de objetos distribuídos.

A escolha por uma arquitetura vai depender de uma análise do público alvo, da confiabilidade da rede em função da abrangência (Internet ou intranet), e da disponibilidade e facilidade de uso das ferramentas de apoio e de construção da arquitetura. Entretanto, não há uma rigidez na decisão por apenas uma solução. Por exemplo, em um sistema de comércio eletrônico pode-se combinar uma arquitetura de cliente-magro para o subsistema de vendas a clientes na Internet, mas usar a arquitetura de cliente-gordo ou de *Web Delivery* no subsistema de gerência e estoque, dependendo da configuração do acesso dos usuários ao sistema.

### 2.4.2 Desenvolvimento para Web

O desenvolvimento de aplicações Web, novas ou construídas a partir de sistemas de informação já existentes (legados), segue uma forma não sistemática e *ad hoc*, semelhante às técnicas usadas nas décadas de 1960 e 1970. Pressman (2000), por exemplo, discute os motivos que levam a não adoção dos princípios da engenharia de *software* pelos desenvolvedores *web*. Entre as várias razões abordadas, o autor cita algumas, tais como:

a) **Tempo curto no desenvolvimento**: enquanto que aplicações tradicionais o desenvolvimento leva meses, as aplicações Web são desenvolvidas em semanas e até em dias, dificultando o planejamento e o amadurecimento dos requisitos do produto.

b) **Rápida evolução**: aplicações Web tornam-se obsoletas em um curto período de tempo. Para os sistemas tradicionais, como os sistemas legados, a vida útil é de vários anos, chegando a décadas. Uma aplicação Web pode ser considerada legada depois de seis meses. A velocidade da evolução desencoraja os desenvolvedores a seguirem um processo de desenvolvimento de *software* e a capturarem com mais cuidado os requisitos do sistema.

c) **Conjunto diferente de preocupações**: o mapeamento de aplicações Web para os tradicionais sistemas orientados a objeto e procedurais não é tão evidente quando se desenvolve com as tecnologias disponíveis para a web. Ferramentas atuais facilitam a construção de versões, encurtando o ciclo de

vida do desenvolvimento, mas não incorporam abstrações para suporte ao reuso (GELLERSEN; GAEDKE, 1999).

d) **Expectativas do usuário**: os usuários de aplicações Web são mais tolerantes a erros, tornando as tarefas de teste e verificação negligenciadas ou pouco rigorosas, como as seguidas no desenvolvimento tradicional.
e) **Origens da Web**: herança de uma prática que considera a Web como uma plataforma de compartilhamento de documentos, isto é, um problema de autoria de conteúdo de páginas e não um problema de engenharia de *software*.

A Engenharia de Software, com seus conceitos e métodos, pode ser aplicada ao desenvolvimento na Web, porém com certas adaptações e a inclusão de outras disciplinas que preencham as lacunas inexistentes no desenvolvimento convencional. A essa nova área do conhecimento Ginige e Murugesan (2001) denominaram de **Engenharia para a Web** (*Web Engineering*, **WebE**), que estabelece o uso de sólidos princípios científicos de engenharia, gestão, abordagens disciplinadas e sistemáticas para o bem sucedido desenvolvimento, implantação e manutenção de sistemas e aplicações de alta qualidade, baseados na Web.

## 2.5 Considerações Finais

Reengenharia de *software* é uma área que traz desafios a quem se propõe realiza-la, considerando vários assuntos e preocupações envolvidos no processo. Dentro do escopo que este trabalho se apresenta, várias outras áreas e temas devem estar bem definidos e esclarecidos.

A engenharia reversa, como primeira etapa do processo de reengenharia, contribui para o aumento da manutenibilidade de sistemas legados, pelo fato de se gerar toda a sua documentação, muitas vezes inexistente ou desatualizada. Com a recuperação da documentação de sistemas legados, em um nível de abstração maior que aquela presente na implementação, pode-se obter ganhos significativos na compreensão do *software* e posterior suporte na etapa de manutenção. Em seguida, com tratamentos e técnicas adequadas, aplicando-se o processo de engenharia avante é possível prover projeto e implementação em plataformas, ambientes e/ou paradigmas mais modernos.

A definição clara do tipo do sistema legado e o ambiente alvo, que é o objeto do processo de migração são discutidos em detalhes neste capitulo. É apresentada e discutida uma estrutura com os principais componentes presentes nesse modelo de aplicação.

O ambiente Delphi, escolhido como domínio do ambiente e linguagem de programação do sistema origem, é discutido em um nível de detalhe bastante aceitável para o entendimento de alguns conceitos pertinentes ao processo de reengenharia. Na pequena resenha apresentada do ambiente, dá-se atenção a elementos que são relevantes à metodologia proposta, que são: os componentes estruturais do *software* produzido em Delphi e os que tratam do acesso e manipulação dos dados.

Para finalizar o capítulo sobre fundamentos, o ambiente alvo, no caso a Web, é mostrado em sua arquitetura básica de funcionamento e discutida as dificuldades e características presentes no desenvolvimento de aplicações nesse tipo de ambiente.

# 3 Trabalhos Relacionados

Nestes últimos anos, muito esforço foi direcionado para pesquisas e trabalhos ligados à reengenharia de *software* e para ferramentas que automatizem o processo de engenharia reversa de sistemas legados. Alguns eventos históricos contribuíram para esse esforço, tal como a passagem dos *mainframes* para a arquitetura cliente/servidor, o problema do ano 2000 – conhecido como "*bug do milênio*", o uso de aplicações em ambientes gráficos, a popularização dos microcomputadores e, recentemente, a Internet. Entretanto, o cenário que sempre impulsionou a reengenharia nas organizações foi a busca por redução dos custos de manter sistemas de *software*, ou aumentar os lucros com a modernização no redesenvolvimento de novos sistemas. Os trabalhos de pesquisa refletem isso; grande parte se preocupa com a migração de sistemas de informação (aplicação e dados) para plataformas tecnológicas mais modernas, de sistemas implementados em linguagens comercias sobre o paradigma estruturado (Cobol, Fortran, Clipper, e C, por exemplo) para o paradigma OO, e da reestruturação de interfaces de usuário do modo caractere para o modo gráfico. Poucos trabalhos tratam especificamente do mesmo domínio de aplicação que é tratado nesta dissertação. Contudo, generalizando todas as idéias e abordagens, percebe-se pontos em comum que podem ser revistos e adaptados para novos domínios de problema.

Destacam-se, neste capítulo, alguns trabalhos acadêmicos que são relevantes e contribuem para a formação da proposta metodológica apresentada nesta dissertação. Os trabalhos relacionados são apresentados sob dois enfoques. O primeiro, Seção 3.1, trata, de uma forma geral, de métodos e técnicas de engenharia reversa e reengenharia, sem considerar aspectos do ambiente alvo. No segundo, Seção 3.2, considera aspectos de interface de usuário para o ambiente Web. Na Seção 3.3 é feita uma análise comparativa de alguns trabalhos e a presente metodologia proposta, com relação a características comuns.

## 3.1 Engenharia Reversa e Reengenharia

Apesar de vários relatos de sucesso sobre engenharia reversa em sistemas legados, alguns autores, tais como Chikofsky e Cross (1990), Canfora, Cimitile e Carlini (1992), Quilici (1995), Jarzabek e Wang (1998), criticam o uso de ferramentas, pois a maioria não consegue atingir o potencial e os objetivos da engenharia reversa citados em Chikofsky e Cross (1990). Canfora, Cimitile e Carlini (1992) identificaram um número de problemas que inibem a adoção dessas ferramentas, tais como:

a) não identificam o nível correto de detalhes recuperados do projeto e falham em estratégias apropriadas de apresentação dos modelos;
b) recuperam somente uma pequena parcela da informação do sistema legado, necessário para a compreensão do projeto; e
c) não são flexíveis o bastante, pois não facilitam a personalização do ambiente operacional.

Os autores também acrescentam que tais ferramentas normalmente recuperam visões do projeto, tal como grafos de estrutura, que são essenciais para a manutenção do *software*, mas insuficientes para a etapa de engenharia avante.

Quilici (1995) defende o uso combinado de uma ferramenta para extrair parcialmente as especificações do sistema e, posteriormente, utilizar a intervenção humana para extrair as restantes, que são impossíveis por meios automáticos. Uma base de conhecimento contendo todas as informações extraídas, tanto por meios automáticos quanto manuais, é apresentada ao usuário como um grafo. Os nós, que representam

elementos de projeto, são ligados aos vários trechos de código fonte que implementam esses elementos.

Técnicas e ferramentas estudam formas de representar o *software* em uma visão genérica de estruturas de código, produzindo metamodelos simples de diferentes tipos de informação. Exemplo de ferramenta que fornece ao usuário uma interface flexível de múltiplas visualizações da arquitetura é o sistema Rigi (WONG *et al.*, 1995), que se concentra na recuperação da estrutura da arquitetura de grandes sistemas. Outras ferramentas são focadas em linguagens específicas seguindo técnicas como detecção de *clichés*, que comparam estilos e padrões de arquitetura de código, útil no estudo de como representar e usar o conhecimento e experiência de programação (RICH; WILLS, 1990).

A revisão bibliográfica descrita neste capitulo enfoca trabalhos que, além de fornecerem um arcabouço de técnicas e ferramentas de recuperação de projeto, descrevem métodos que permitem a mudança de paradigma de construção do *software*, isto é, a reestruturação do código procedural em um modelo orientado a objetos, conforme destacados nas subseções a seguir.

### 3.1.1 Abordagens de Reengenharia em Sistemas Legados Tradicionais

Em sistemas de informação tradicionais, codificados no paradigma procedural, os objetos são implementados em estruturas de dados por variáveis de memória. Em programas seguindo boas práticas de projeto estruturado, cada uma das operações sobre os objetos deveria ser realizada como um procedimento único, com forte coesão e baixo acoplamento. Por negligência de projeto e de outros fatores, vários sistemas legados seguem uma codificação confusa, onde normalmente as operações sobre os objetos de dados misturam-se com outras funcionalidades do programa. Esse tipo de prática dificulta a identificação dos objetos em uma etapa de reestruturação para um código OO. Técnicas como *slicing* (WEISER, 1984), que permite dividir (fatiar) o código em pequenos blocos em função do uso das variáveis, pode ser um passo inicial para facilitar a reestruturação OO.

Liu e Wilde (1990) propuseram dois métodos para identificar objetos no código. O primeiro método começa identificando as estruturas de dados globais que são prováveis candidatos a representar os atributos de um objeto. Em seguida, agrupam-se procedimentos que manipulam dados globais e analisam-se os relacionamentos entre os diversos procedimentos com a finalidade de encontrar as prováveis operações (métodos) dos objetos. O segundo método inicia identificando as dependências entre os tipos de dados locais em um programa e os tipos de dados identificados no primeiro método que compartilham dos mesmos dados em comum, formando os objetos candidatos.

Canfora, Cimitile e Munro (1994), buscando o reuso, classificam possíveis objetos e abstrações em quatro grupos: *objetos abstratos* (estruturas de dados + operações), *tipos de dados abstratos* (tipos de dados + operações + instanciação), *estruturas de dados genéricos* (objetos abstratos parametrizados por tipos para generalização) e *tipos de dados abstratos genéricos*. Então, os autores empregam critérios heurísticos para identificar os candidatos em cada grupo.

Gall e Klösch (1995) descrevem outra abordagem para encontrar objetos. Iniciam com a engenharia reversa para gerar uma estrutura de grafos e diagramas de fluxo de dados. Então, identificam os objetos em torno das entidades persistentes (dados em arquivos e tabelas) e das estruturas e variáveis de memória, porém relevantes sob o ponto de vista do domínio da aplicação.

Sneed e Nyary (1995) aplicam também a idéia do uso conjugado de ferramentas de extração com uso de práticas (heurísticas) semiautomatizadas. As

ferramentas de apoio à engenharia reversa são utilizadas em diferentes componentes de *software* como interface de usuário, arquivos e banco de dados, programas, etc., para identificar objetos em sistemas legados. Dependendo do tipo de *software* legado, dos objetivos do projeto, e dos tipos de objetos que é de interesse para a reengenharia, deve-se aplicar heurísticas adequadas para a recuperação do projeto OO.

Como estudo de caso de reengenharia, consideram programas escritos em C para um paradigma OO, em C++. A identificação de classes candidatas no código de C é realizado de forma semiautomática, envolvendo a aplicação de heurísticas e análise manual dos programas por um perito humano. Um perito do domínio fica ciente dos conceitos do domínio da aplicação que são candidatos a classes, enquanto o perito na técnica de programação OO julga quais conceitos do projeto são "bons" candidatos a classes. As classes geradas automaticamente pelo processo automático devem ser apresentadas aos peritos humanos para a aceitação e o refinamento. No caso de sistemas legados implementados em COBOL, em um ambiente de *mainframe*, as estruturas como interface de usuário, banco de dados, e regras de negócio devem ser alteradas radicalmente, devido às mudanças para um ambiente cliente/servidor. A análise dos programas permite identificar quatro diferentes tipos de candidatos a objetos: os arquivos, as visões de banco de dados, as estruturas de dados e os parâmetros de ligação. Um objeto pode ser qualquer um desses tipos, e seus atributos os membros de cada conjunto de dados, isto é, os campos subordinados. Para a extração das operações, deve-se analisar as referências cruzadas entre os procedimentos. Após a extração das operações, deve-se identificar as conexões entre os objetos. Para isso, analisa-se a operação de um objeto e obtém-se variáveis que estão fora do escopo da operação. Essas variáveis são parâmetros utilizados pelas operações do objeto, mas que não são atributos desse objeto, e sim de outros.

Portanto, os trabalhos de pesquisa citados nesta seção servem de referência básica para métodos mais recentes. Os métodos mais recentes tratam não somente de engenharia reversa em sistemas legados tradicionais, mas de sistemas implementados em modelos de projeto mais modernos, porém com falhas ou deficiências que afetam a qualidade do *software* produzido. A seguir, alguns desses métodos são apresentados.

### 3.1.2 Método Fusion/RE e PRE/OO

Fusion/RE (PENTEADO, 1996), baseado no método Fusion (COLEMAN *et al.*, 1994), se apresenta como um método de engenharia reversa para construção do modelo de análise OO de sistemas legados que foram implementados em outro paradigma diferente do paradigma OO. Já foi aplicado a sistemas implementados em Clipper (CAGNIN *et al*., 1999), C (PENTEADO *et al*., 1998) e COBOL (CAMARGO; PENTEADO, 2001), onde foi usado no processo de reengenharia orientada a objetos para a Web. Nesse método, a engenharia reversa é feita em quatro passos, resumidos a seguir.

1. **Revitalizar a Arquitetura do Sistema.** Consiste na revitalização da arquitetura do sistema com base na documentação existente ou no código fonte, caso não exista. Ao final desse passo, gera-se uma lista, por meio de referência cruzada, de todos os procedimentos, com sua descrição e hierarquia de chamadas ("chama/chamado por").

2. **Recuperar o Modelo de Análise do Sistema Atual (MASA)**. Elabora-se um modelo de análise do sistema atual considerando apenas as estruturas de dados existentes no código fonte. Um pseudo Modelo de Objetos do sistema é gerado, identificando-se as pretendentes a classes e seus relacionamentos, bem como seus possíveis atributos e procedimentos associados.

Neste passo, um esquema de classificação de anomalias de código é utilizado, pois alguns procedimentos manipulam ou não mais de um objeto. A classificação é a seguinte: **O** para procedimento **o**bservador, **C** para construtor e **I** para procedimento de implementação. Um procedimento é observador de uma estrutura de dados (classe candidata), quando apenas lê essa estrutura. É construtor, quando a altera. É possível (e comum) encontrar procedimentos que possuem um comportamento híbrido, observando e alterando mais de uma estrutura de dados. Por exemplo, um procedimento é classificado como (O+C+), pois é observador e construtor de duas ou mais estruturas de dados.

Após a criação do modelo de objetos pode-se, opcionalmente, elaborar os cenários. As entradas e saídas do sistema necessitam de agentes para manipula-las, constituindo os cenários do funcionamento do sistema. Um cenário é uma seqüência de eventos fluindo entre agentes e o sistema com uma finalidade específica. Os cenários auxiliam na elaboração do Modelo de Ciclo de Vida e do Modelo de Operações.

A seguir, define-se o Modelo de Ciclo de Vida do sistema, que retrata o seu comportamento geral, em uma seqüência cronológica em que as operações são realizadas. Estas servem como base para a construção do Modelo de Operações do Sistema, no qual cada operação é descrita de forma minuciosa, de acordo com um gabarito preestabelecido.

3. **Criar o Modelo de Análise do Sistema (MAS)**. Obtém-se um modelo orientado a objetos abstraído do MASA sem redundâncias ou inconsistências, enfatizando o domínio da aplicação e não a implementação. Os modelos de Objetos, de Ciclo de Vida e de Operações são abstraídos daqueles construídos anteriormente, com alguns cuidados adicionais, como por exemplo à mudança de nomes para outros mais significativos, generalização e especialização de classes. Os procedimentos sem anomalias geram métodos diretamente, enquanto os com anomalias devem ser desmembrados em diversos métodos, de modo que cada método interaja com apenas uma classe.

4. **Mapear os Modelos (MAS para o MASA)**. Descrevem-se as relações de equivalência entre o Modelo de Análise do Sistema (MAS) e o Modelo de Análise do Sistema Atual (MASA). São mapeados as classes, atributos e métodos do MAS para os elementos correspondentes do MASA, anotando-se possíveis inclusões e exclusões. Esse passo é importante na futura manutenção e reuso do sistema, permitindo a *rastreabilidade* entre um velho e o novo sistema a ser implementado.

No final da aplicação do método Fusion/RE, pode-se adotar quatro alternativas: uso da documentação produzida para facilitar a manutenção, melhoria da qualidade do sistema atual, desenvolvimento do sistema por reengenharia e conservação da implementação atual, com aperfeiçoamentos futuros usando a orientação a objetos (BRAGA, 1998). O esquema dos quatro passos e atividades do método pode ser visto na Figura 3.1.

Em uma revisão do método Fusion/RE, Penteado *et al*. (1998) propuseram mais dois passos correspondentes à segmentação, a seguir descritos nos passos 5 e 6.

5. **Elaboração do Projeto Avante**. Visa a elaboração dos modelos da fase de projeto para que a engenharia avante seja feita mudando-se o paradigma de procedimental para orientado a objetos, mantendo-se ou não a linguagem procedimental em uso. Os diagramas gerados nesse passo são quatro: o Grafo de Interação de Objetos, Grafos de Visibilidade, Diagrama de Descrições de Classes e os Grafos de Herança. O Grafo de Interação de Objetos mostra quais objetos participam da execução de uma operação e a seqüência de troca de mensagens entre eles, sendo construído com base no Modelo de Operações. Os Grafos de Visibilidade estabelecem a forma de comunicação

entre as classes, especificando a interface para troca de mensagens do sistema. O Diagrama de Descrições de Classes complementam o Modelo de Objetos do sistema, especificando quais os atributos de cada classe e sua interface externa. Os Grafos de Herança definem as especializações existentes entre as classes complementando as Descrições de Classes.

6. **Segmentação do Sistema**. Os procedimentos com anomalias são transformados em métodos. O código é examinado, dividido em métodos e são inseridos comentários mostrando a correspondência entre eles. Esse passo é elaborado com base nos diagramas do projeto avante do *software*.

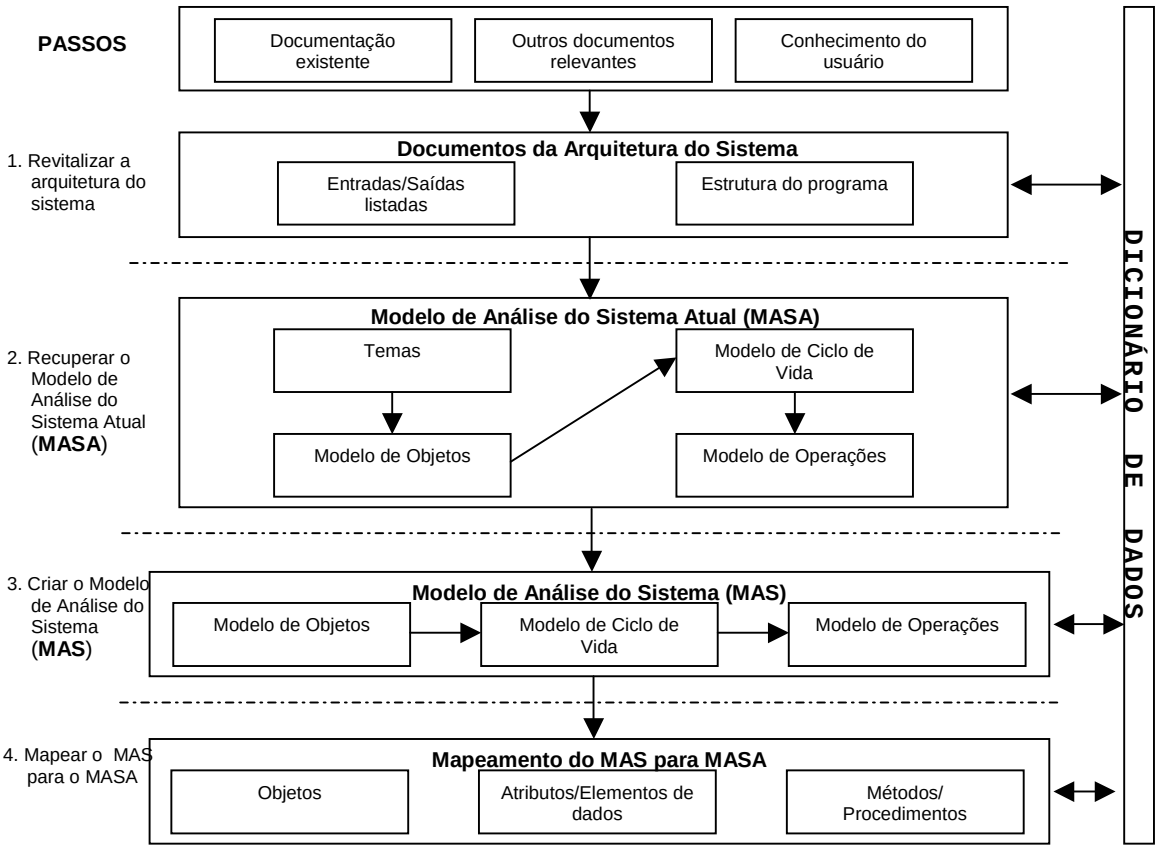



FIGURA 3.1 – Esquema original do método Fusion/RE (PENTEADO, 1996).

O Processo de Reengenharia Orientada a Objetos (PRE/OO) (LEMOS, 2002) é um processo de reengenharia que surge como evolução do método Fusion/RE (PENTEADO, 1996). Utiliza a UML (OMG, 1999) para representar os modelos orientados a objetos. Modifica a forma de condução do processo de ciclo de vida, de seqüencial linear para evolutivo. Além disso, adiciona métodos de qualidade, quando aplica KPAs (*Key Process Area*) do Nível 2 de maturidade do SW-CMM (*Capability Maturity Model for Software*) (PAULK *et al.*, 1995).

O processo proposto em PRE/OO, por meio de padrões, visa auxiliar o engenheiro de *software* na plena realização da reengenharia orientada a objetos. Para isso, a Família de Padrões para Reengenharia Orientada a Objetos (FaPRE/OO) proposta por RECCHIA (2002) é adaptada para sistemas implementados em Delphi sem a utilização do paradigma orientado a objetos, que serão convertidos para Delphi orientado a objetos. Os padrões seguem o o formato proposto na Linguagem de Padrões de Engenharia Reversa e Reengenharia (DEMEYER et al., 1999) e também utilizada por RECCHIA (2002) em seu trabalho intitulado “Família de Padrões para

Reengenharia Orientada a Objetos" (FaPRE/OO). PRE/OO é organizado em sete *clusters* de padrões divididos em dois grupos: (i) padrões para a realização da reengenharia orientada a objetos, e (ii) padrões para a melhoria da qualidade da reengenharia.

De um modo geral, com algumas modificações, PRE/OO pode ser aplicado como um processo genérico de reengenharia de sistemas de *software* legados, apesar da especificação inicial ter sido feita em um domínio para *software* implementado em Delphi. A seu favor está o método organizado em padrões para realizar o processo de reengenharia.

### 3.1.3 Metamodelos e Refatoração em Aplicações OO

O projeto FAMOOS Esprit (*A Framework based Approach for Mastering Object-Oriented Systems*) (DUCASSE; DEMEYER, 1999) é um exemplo de abordagem de reengenharia de *software* para reestruturar código legado OO para métodos e linguagens de programação OO mais atuais. FAMOOS compreende uma metodologia e um conjunto de ferramentas para detectar falhas de projeto em sistemas OO, cujo objetivo é tornar tais sistemas mais eficientes e flexíveis. Entre as diversas preocupações no projeto FAMOOS, destaca-se FAMIX (*FAMOOS Information Exchange Model*), um modelo de representação de *software* OO independente da linguagem de programação OO. A finalidade do FAMIX é o compartilhamento e troca de informações entre várias ferramentas que utilizam metamodelos de representação OO. Alguns autores (JARZABEK e WANG, 1998) (KOSCHKE *et al.*, 1998), (TICHELAAR *et al*, 2000a) destacam o uso de metamodelos genéricos, mantidos em repositórios, pois fornecem um mecanismo mais eficiente para compartilhamento de informação entre diversos tipos de ferramentas CASE. Além disso, facilita o armazenamento, consultas e extração de qualquer informação relacionada à representação do *software*. Atualmente, FAMIX utiliza os formatos CDIF (*Common Data/Design Interchange Format*) (CDIF, 2003) e, recentemente, XMI (TICHELAAR *et al*., 2000b). XMI (XML METADATA INTERCHANGE, 2003) é um novo padrão definido pela OMG para intercâmbio de metamodelos UML entre ferramentas CASE de modelagem OO.

## 3.2 Migração para Web: Reengenharia de Interface de Usuário

Embora muitos trabalhos sejam realizados atualmente na migração de código de sistemas legados para novas plataformas, surpreendentemente poucas pesquisas têm tentado mostrar a viabilidade da automatização (total ou parcial) da reengenharia de seus componentes de interação. Um raro exemplo de trabalho nesta linha é o projeto MORPH (*Model Oriented Reengineering Process for Human-Computer Interface*) (MOORE, 1998), que propõe migrar interfaces de sistema legado (orientada a caractere) para interfaces no ambiente Web (MOORE; MOSHKINA, 2000).

MORPH é uma técnica que inicialmente foi desenvolvida para realizar a reengenharia de interfaces baseadas em caractere para interfaces gráficas. Em seguida, os autores estenderam a proposta para realizar a reengenharia de interface para Web. Três passos são utilizados no processo de reengenharia, que são Detecção, Transformação e Geração, detalhados a seguir.

### 3.2.1 Modelo do Processo MORPH

O processo de reengenharia de interface proposto em MORPH é resumidamente visto na Figura 3.2. A seguir são definidas as três etapas do processo.

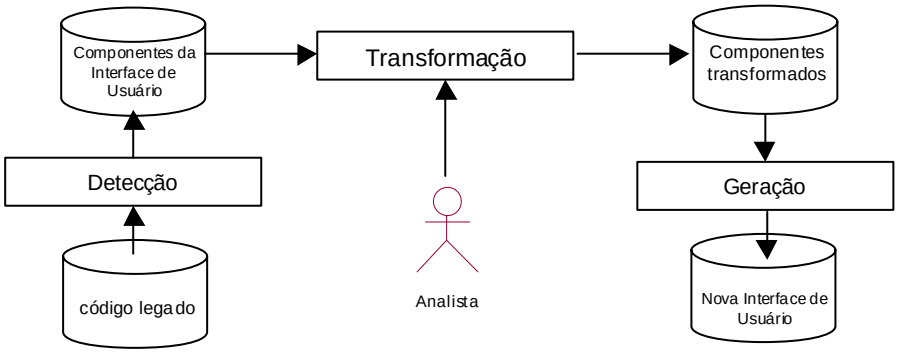



FIGURA 3.2 – Processo MORPH.

#### 3.2.1.1 Detecção

Por engenharia reversa, uma análise estática é realizada no código da aplicação legada, com o propósito de extrair componentes importantes ligados à interface de usuário. MORPH estabelece duas formas de análise estática: análise de fluxo e comparação de padrões, definidos adiante.

**Análise de fluxo** – MORPH usa duas formas. A primeira é a análise do fluxo de controle, que é a codificação da estrutura de fluxo de controle de um programa. Usando os resultados desta análise, MORPH realiza uma análise específica no fluxo de dados, a fim de localizar pontos de entrada e saída do programa. Em seguida, é observado o resto do fluxo de controle, em função dos efeitos que os valores são atribuídos às estruturas de dados.

**Comparação de padrões** – Completada a análise do fluxo do programa, sobre os resultados são aplicadas regras de comparação de padrões sintáticos, para identificar ocorrências de componentes de interface do usuário. As regras são aplicadas sobre os resultados da análise de fluxo e não sobre o código legado, que torna a técnica de padrões independente da linguagem legada.

Uma vez encerrado o passo de Detecção, MORPH armazena as informações obtidas, em forma de abstrações de interface de usuário, em uma base de conhecimento, utilizando uma linguagem de representação.

#### 3.2.1.2 Transformação

A informação armazenada na base de conhecimento é usada para uma classificação do tipo de componente da interface e para determinar qual a abstração representará a funcionalidade equivalente em uma biblioteca de componentes visuais. A etapa de Transformação permite também que um perito (analista) refine e melhore os resultados da interface de usuário, em situações em que MORPH decide ou preenche informações erroneamente, por não aplicar corretamente regras adequadas na etapa de Detecção. O resultado da Transformação serve para a etapa seguinte gerar uma nova interface gráfica do usuário.

#### 3.2.1.3 Geração

É a criação e inserção dos componentes visuais (*widget*) implementados em uma interface de usuário, em um ambiente gráfico equivalente em funcionalidades ao sistema legado. O passo de Geração usa as informações da etapa de Detecção para

identificar onde será colocado o novo componente gráfico, e informações de Transformação para determinar qual componente gráfico deve ser utilizado.

## 3.3 Considerações Finais

Este capítulo apresentou trabalhos relevantes que contribuem para a realização desta dissertação. Para auxiliar a comparação entre diversas abordagens, agrupou-se em quatro grupos, buscando similaridades gerais entre os trabalhos. Desta forma, a Tabela 3.1 apresenta os grupos, fornecendo um resumo dos métodos e as contribuições.

TABELA 3.1 – Resumo dos trabalhos relacionados.

| Grupo | Abordagem geral | Contribuições na dissertação |
|---|---|---|
| Abordagens de reengenharia em sistemas legados tradicionais | Fornecem a base e validação das técnicas de engenharia reversa em *software* legado, além de discutirem formas de reengenharia para mudança de paradigma de desenvolvimento. | Heurísticas e técnicas de engenharia reversa para identificação de classes/objetos. |
| Método Fusion/RE e PRE/OO | Evolui de um método de engenharia reversa para um processo completo de reengenharia em sistemas legado implementados em diversas linguagens atuais. | Com algumas adaptações, alguns passos são incorporados no processo proposto. |
| Metamodelos e refatoração em aplicações OO | Reestruturação de código legado OO, com uso de repositórios em uma linguagem padrão (XMI). | Uso de repositório e heurísticas para melhoria no projeto OO. |
| Modelo do Processo MORPH | Define um processo genérico para reengenharia de interface de usuário no modo caractere para um ambiente gráfico. | Abordagem mista: o uso de regras para mapeamento entre componentes de ambientes de interface distintas, e a participação de um perito humano para decidir e resolver as ambigüidades que falham no processo automático. |

No trabalho aqui proposto, são utilizados os fundamentos básicos de cada enfoque relacionado anteriormente. O processo de reengenharia do código legado se preocupa com a recuperação de componentes de alto nível de abstração e na reestruturação do código legado para uma abordagem OO. A representação do esquema de implementação (visão de alto nível dos componentes) é realizada usando metamodelos, que são armazenados em um repositório. A partir desse repositório é possível gerar modelos de projeto em um formato padrão, como XMI (XML METADATA INTERCHANGE, 2003). Quanto à etapa de tratamento de interface de usuário, o processo de reengenharia é direcionado para sistemas implementados com interfaces do tipo WIMP. Como os ambientes de implementação (do sistema original e do ambiente Web) não são totalmente compatíveis, o método emprega duas estratégias. A primeira utiliza regras de conversão para os componentes visuais que possuam correspondentes diretos em ambas implementações. A segunda, defende o uso de *design patterns* (de projeto de interface Web) para reestruturar as funcionalidades de interface do sistema legado, difíceis de implementar no formato HTML (MARTINS *et al.*, 2002c). No próximo capítulo, o processo de reengenharia é apresentado em mais detalhes.

# 4 O Processo de Reengenharia

Este capítulo apresenta o núcleo da metodologia de migração, em que um processo descreve um conjunto de passos organizados para atingir os objetivos de um processo de reengenharia. O processo proposto engloba duas atividades principais: o conhecimento do sistema origem (engenharia reversa) e a elaboração de modelos de projeto (reprojeto), para posterior implementação do novo sistema (engenharia avante). Uma abordagem de reconstrução total de um sistema de *software* legado é referenciada na literatura da área como abordagem *Big Bang*, também conhecida como *Cold Turkey* (BRODIE; STONEBRAKER, 1995), pois a construção do novo sistema "parte do zero", usando uma nova plataforma de hardware e modernas arquiteturas de *software*, ferramentas de desenvolvimento e banco de dados (BISBAL et al.,1999).

De um modo geral, um processo de migração (com redesenvolvimento) é constituído de quatro atividades distintas, como visto na Figura 4.1, que são discutidas logo a seguir.

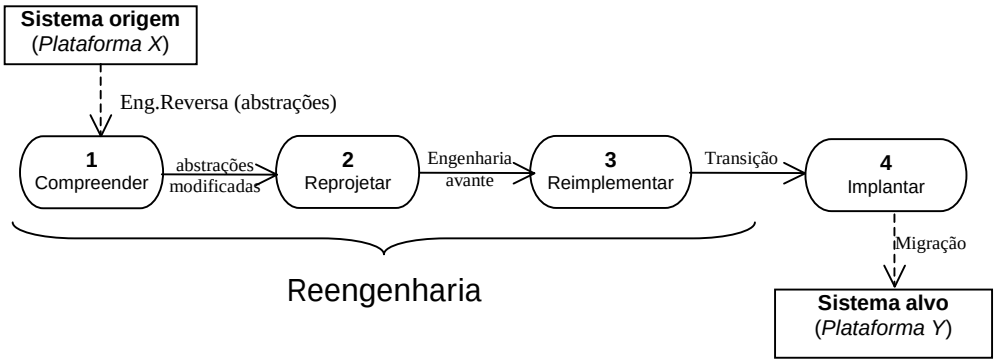



FIGURA 4.1 – Atividades presentes em um processo de migração e reengenharia.

**Compreender**. O objetivo desta atividade é conhecer a arquitetura do sistema (captura de abstrações) e identificar as partes que devem ser reprojetadas. Técnicas e ferramentas de engenharia reversa devem ser utilizadas nesta atividade.

**Reprojetar**. Compreende todo o esforço em modificar a abstração, capturada na atividade anterior, para uma melhor eficiência, seguindo conceitos mais modernos (OO, por exemplo) e buscando a reutilização.

**Reimplementar**. Usando modelos e artefatos do novo projeto, o sistema deve ser implementado em uma nova arquitetura, se possível usando ferramentas e técnicas de engenharia avante para reduzir o tempo de implementação.

**Implantar**. É a etapa crítica do processo, quando ocorre a mudança total ou parcial do velho sistema para o novo. Portanto, uma abordagem cuidadosa deve contemplar aspectos organizacionais (processo de negócio), assim como culturais (pessoal técnico e usuários) e tecnológicos (arquitetura de *software* e hardware). Deve-se articular planos de teste e validações, suficientes para garantir a equivalência entre o sistema original e o sistema destino, acrescentado de novos requisitos de qualidade. Preliminarmente, um plano de transição deve ser elaborado para incluir o mapeamento dos dados e dos processos, para garantir a equivalência das funcionalidades, bem como estratégias para substituição do sistema velho, com a possibilidade de conversão de arquivos e de banco de dados e substituição de hardware (BISBAL *et al*.,1999).

Apesar do processo proposto não ser um processo completo de migração, ele ajuda o (re)desenvolvedor na construção dos modelos de requisitos funcionais e no projeto de interfaces, necessários para a etapa seguinte de engenharia de produção

(avante). Esta última etapa, a engenharia avante, não faz parte da proposta, pois várias ferramentas existentes e métodos propostos atualmente podem auxiliar o engenheiro de *software* a realizar essa tarefa. Outra questão importante é que, embora a metodologia trate de um tipo de domínio para o sistema origem e enfatize uma plataforma específica para o sistema alvo, no caso a Web, nada impede que, a partir dos modelos gerados, as estratégias sejam adaptadas para a reestruturação do sistema legado no próprio ambiente ou prevendo a convivência de ambos os sistemas em plataformas diferentes, com a devida reutilização de componentes comuns como dados e regras de negócio, por exemplo.

A visão geral da proposta e as etapas do processo são descritas em detalhes nas próximas seções.

## 4.1 Visão geral da Proposta

A proposta visa abordar as duas primeiras atividades: *compreensão* e *reprojeto* do sistema legado (veja o fluxo da Figura 4.1). Na primeira atividade é necessário conhecer a arquitetura do sistema e identificar as partes que devem ser reprojetadas. Pela quantidade de dados envolvidos, deve-se utilizar técnicas automáticas ou semi-automáticas, para o levantamento da documentação e dos componentes do *software* a partir dos códigos-fonte e do esquema de banco de dados. Outra técnica menos automática, porém necessária para a compreensão do sistema como um todo, é realizar entrevistas com os especialistas do sistema, recuperando as especificações de requisitos vigentes, manuais e modelos de projeto (BRAGA, 1998) (PENTEADO, 1996).

Um mecanismo de suporte à tarefa de compreensão é proposto para recuperar aspectos dinâmicos do sistema. No caso, é proposta a aplicação de casos de uso (*use cases*, em UML), adaptados para a realidade da reengenharia de *software*.

Em seguida, com base nas informações coletadas e nos casos de uso, deve-se produzir modelos de análise e de projeto de todas as classes e componentes de interface de usuário, que passarão pelo processo de engenharia avante. Isso enfatiza o que as classes/componentes fazem e não como se encontram implementadas atualmente.

Em resumo, o processo aqui proposto é definido em quatro etapas:

1. Recuperar abstrações
2. Construir casos de uso para Reengenharia
3. Gerar Modelo Conceitual do Sistema (MCS)
4. Mapear interfaces WIMP para Web

Uma visão geral do processo de reengenharia pode ser vista na Figura 4.2, em forma de diagrama de atividades, mostrando o fluxo das etapas principais da proposta. Antes de iniciar a primeira etapa, uma preparação da configuração do ambiente do sistema legado deve ser feita, como por exemplo, providenciar cópias do código fonte e dos dados do sistema em uma área de trabalho de transição. Esse procedimento garantirá a continuidade da utilização, em paralelo, da aplicação legada pelos usuários, até que se conclua a migração como um todo. Artefatos produzidos em cada etapa do processo comporão estruturas em um metamodelo de representação, mapeadas em um repositório. O uso de um repositório facilita a pesquisa, extração e a geração de modelos para o projeto de migração. O processo proposto nesta dissertação é um aprimoramento daquele apresentado pelo autor em (MARTINS *et al.*, 2002a) e (MARTINS *et al.*, 2002b). Nas próximas seções, as quatro etapas são descritas com mais detalhes.

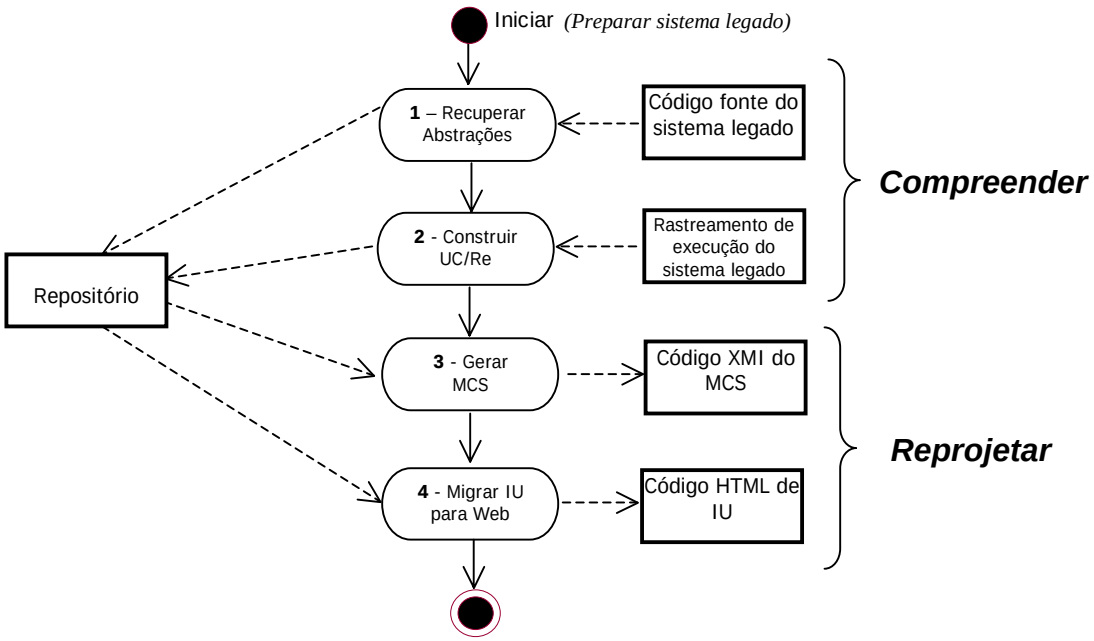



FIGURA 4.2 – Fluxo de atividades principais da metodologia proposta.

## 4.2 Primeira Etapa – Recuperar Abstrações

Nesta etapa são apresentadas as diretrizes necessárias para a engenharia reversa de um sistema de informação WIMP implementado no paradigma baseado em eventos, em linguagens visuais orientas a objetos. Para realizar esta etapa é necessário o uso de ferramentas de apoio à extração de fatos, tipo *parsers* e analisadores de código, definidas para a gramática da linguagem visual[*] do sistema origem. Entretanto, grande parte das ferramentas de engenharia reversa captura informações em um nível de detalhamento que não satisfaz às necessidades da reengenharia (JARZABEK; WANG, 1998), o que provavelmente ressalta a necessidade da intervenção humana no processo. É importante destacar que para grandes sistemas, modelos e documentos que retratem o entendimento de aspectos estruturais são mais importantes do que o conhecimento de um componente isolado qualquer (WONG *et al*., 1995). Definir quais abstrações são relevantes ao processo, delimita não apenas a quantidade de dados a serem recuperadas, como facilita a administração do repositório da base de conhecimento.

As abstrações que interessam ao processo de reengenharia são: módulos, objetos de dados, procedimentos e objetos de interface de usuário. Estes objetos formarão a base de conhecimento para a construção dos futuros modelos de projeto da reengenharia. O fluxo de atividades da etapa "Recuperar Abstrações" é apresentado na Figura 4.3. Observa-se que a atividade "Reestruturar código de procedimentos" é adicionada à etapa como um passo para melhoria do código legado.

Os passos e regras para identificação das abstrações são detalhados nas subseções que seguem.

[*] No contexto do trabalho, o termo *"linguagem visual"* se aplica ao domínio das linguagens dos ambientes de desenvolvimento integrados, citados no Capítulo 1, como por exemplo Delphi, Visual Basic, Visual Café e outros que sigam esse modelo.

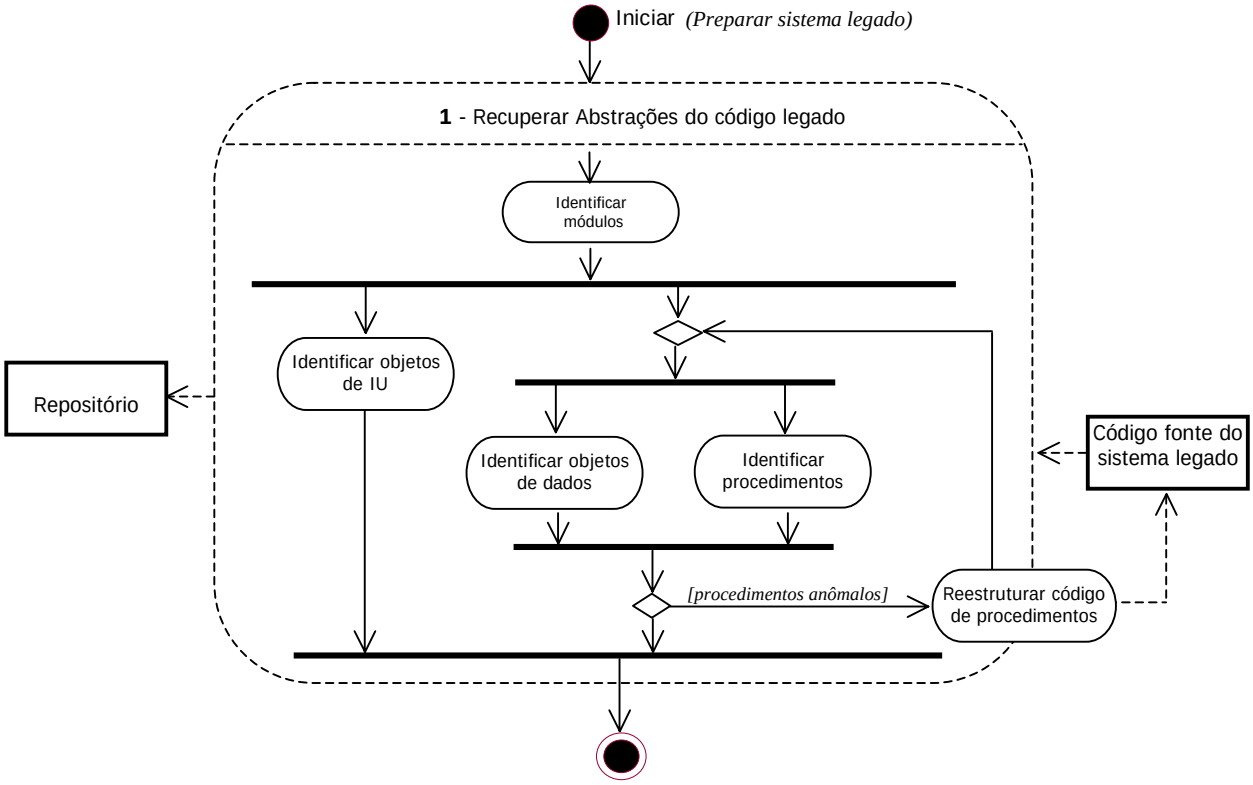



FIGURA 4.3 – Fluxo de atividades da etapa “Recuperar Abstrações”.

### 4.2.1 Módulos

No âmbito da proposta, considera-se um módulo como uma unidade de decomposição funcional de um sistema, fisicamente representado por um mecanismo definido no ambiente de desenvolvimento do sistema legado, tal como um arquivo “.pas", em Delphi, ou um “.java”/ “.class”, em Java. No tradicional enfoque do projeto estruturado, os módulos são organizados, de um modo geral, em programas e rotinas que se inter-relacionam através de chamadas, trocando informações através de dados em forma de variáveis parametrizadas nas chamadas, ou compartilhadas em uma área de uso global.

Em UML, o conceito de pacotes (e sua representação em forma de diagrama) pode ser útil na compreensão e na visão de dependência entre os módulos. O conjunto de todos os módulos forma subsistemas (módulos/pacotes maiores), que por conseguinte formam o sistema como um todo, descrevendo um mapa lógico e funcional do sistema. A relação de dependência entre os módulos, na semântica de uma linguagem visual, ocorrerá em forma de uma instrução do tipo *includes* ou *uses*, citando-se Java e Delphi, respectivamente.

Outras medidas como a utilização de métricas pode auxiliar o projetista da migração a perceber características quantitativas e de complexidade dos módulos, úteis na etapa de reestruturação e segmentação do código fonte, a serem vistas na Subseção 4.2.4. Métricas simples, como quantidade de linhas de código de comentários e comandos, procedimentos, objetos, variáveis, e outros parâmetros são informações que podem ser estruturadas na metaclasse Módulo, como listados na Figura 4.4.

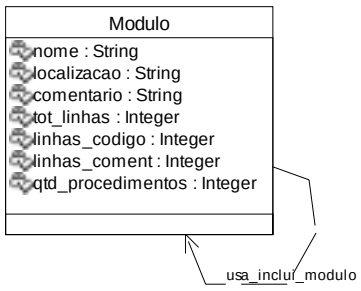



FIGURA 4.4 – Estrutura para informações dos módulos.

Em síntese, as regras para identificação de módulos aplicadas no domínio de Delphi, são as seguintes:

**R1** – O módulo principal está representado em um arquivo de projeto (“.dpr”). Neste arquivo estará registrada a localização física de todos os módulos do sistema.

**R2** – Um módulo é representado por um arquivo *unit* (“.pas”).

**R3** – Uma dependência entre um módulo e outro se faz pela declaração “**uses $u_1$, $u_2$, $u_i$,.., $u_n$**", onde $u_i$ representa um módulo a ser *incluído* no módulo que declara o uso.

### 4.2.2 Objetos de Dados

São os objetos fornecidos pelo ambiente de programação que fazem acesso aos dados persistentes a partir de classes do tipo *DataSets*, herdando atributos e métodos para acesso e manipulação a campos (colunas), registros e conexão a tabelas de um banco de dados.

Se for opção realizar a engenharia reversa do banco de dados, o primeiro passo é localizar qual instância do banco de dados é utilizada pelo sistema. Linguagens visuais fornecem objetos apropriados para conectar a aplicação a um banco de dados. Em VB (versão 6), usando o mecanismo chamado ADO, temos o objeto *ADOConnection*. Em Delphi, usando o mecanismo BDE, um objeto da classe *TDatabase* normalmente é que armazena as informações sobre o acesso ao banco. Na Figura 4.5, observam-se dois códigos nas linguagens citadas, com exemplos de objetos de conexão a um banco de dados Oracle, destacando-se os parâmetros do usuário e senha, “SISVEN” e “XYZPWD”, respectivamente.

As informações obtidas da conexão com o banco de dados serão úteis para conhecer as particularidades das tabelas de dados e objetos armazenados no banco de dados. A metaclasse Database (Figura 4.6) ficará responsável pelo registro dessas informações no metamodelo.

**a) Visual Basic (usando ADO)**

```
Option Explicit
Public conn As New ADODB.Connection

Public Sub OpenConn()
 conn.CursorLocation = adUseClient
 conn.ConnectionString= "dsn=MizMoORCL;" & _
                        "uid= SISVEN; pwd= XYZPWD"
 conn.Open
End Sub
```

**b) Delphi 5 (usando BDE)**

```
procedure OpenConn;
var Database1: TDatabase;
begin
 with Database1 begin
 .Params.Strings := 'SERVER NAME=ORCL.world'
      + 'USER NAME=SISVEN PASSWORD=XYZPWD';
 end;
end;
```

FIGURA 4.5 – Informações sobre acesso à instância de um banco de dados, definidas em duas linguagens de programação.

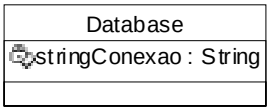



FIGURA 4.6 – Metaclasse Database.

Através de engenharia reversa de banco de dados, por consulta às visões do dicionário de dados ou usando uma ferramenta CASE, é possível extrair a estrutura das tabelas, além dos relacionamentos e objetos associados, como índices, stored-*procedures* e *triggers*.

Complementando o metamodelo relativo ao banco de dados adiciona-se as metaclasses Tabela e Campo, contendo as informações sobre as tabelas, os campos destas tabelas e os relacionamentos entre as tabelas, através de determinados campos, conforme pode ser visto na Figura 4.7.

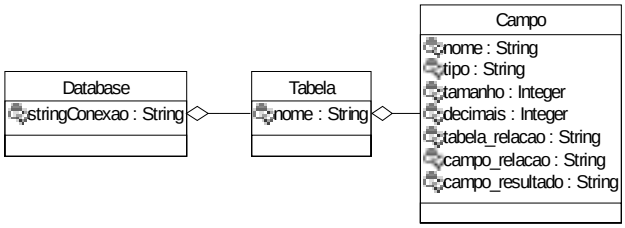



FIGURA 4.7 – Metaclasses sobre o banco de dados de migração.

Em seguida, são identificadas as formas de manipulação das tabelas, descritas nos módulos do sistema. Um módulo pode conter procedimentos que acessam uma ou mais tabelas através dos objetos DataSet. Um DataSet fornece informações sobre qual tabela de dados é acessada do banco de dados, por meio de uma propriedade do objeto ou de um método para execução de comandos SQL.

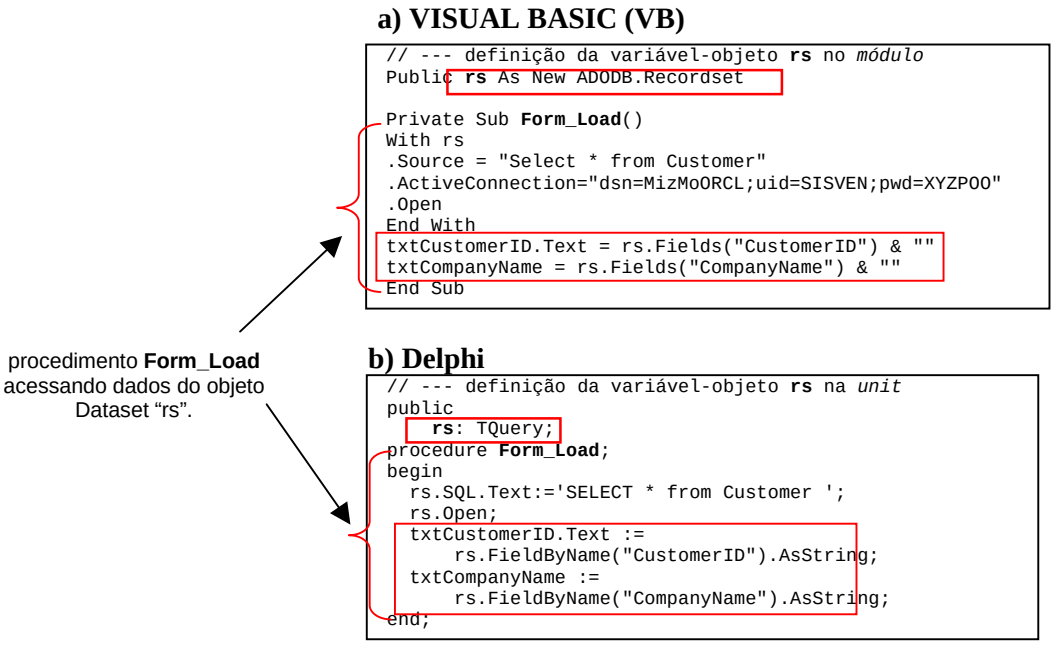



FIGURA 4.8 – Exemplos de objetos DataSet em VB e Delphi.

Nos exemplos da Figura 4.8, observam-se trechos de código fonte representando um tipo de objeto DataSet definido para a variável "rs", instanciada a

partir de *ADODB.RecordSet* e *TQuery*, classes correspondentes às linguagens VB e Delphi, respectivamente. Em ambos os exemplos, o procedimento é o mesmo: abre-se o objeto *rs*, informando-se o comando SQL para recuperar todas as tuplas (registros) da tabela Customer, com todas as colunas (campos); depois, preenche-se dois controles do formulário (txtCustomerID e txtCompanyName) com os valores dos campos correspondentes da tabela. O mesmo exemplo serve para ilustrar o estilo de programação discutido no Capítulo 1, que deve ser evitado, isto é, a mistura no código da aplicação do mecanismo de acesso (SQL) e conexão aos dados com objetos de interface de usuário (controles visuais de formulário), além de infringir o conceito de ocultação da informação defendido em programação OO.

Desta forma, a estrutura das metaclasses fica completa com a inclusão da classe DataSet, conforme visto na Figura 4.9. A seguir, são relacionadas as regras para identificar os objetos Datasets em Delphi.

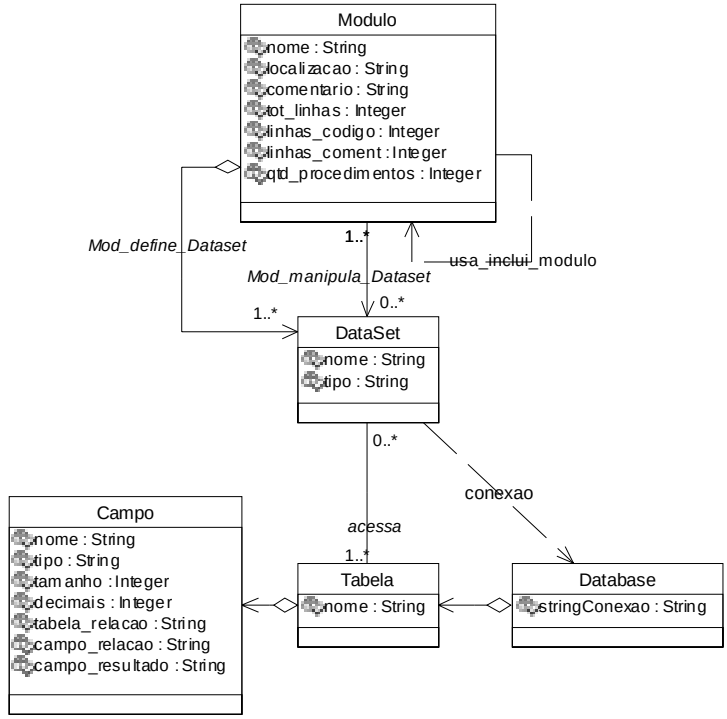



FIGURA 4.9 – Estrutura de metaclasses com a classe DataSet.

**R1** – Construir uma tabela de símbolos com as variáveis do tipo/classe TDataset. Um TDataSet é a classe base para todas as subclasses descendentes para manipulação de dados, e são classificadas em três tipos: *Table*, *Query* e *Stored-procedure*. A Tabela 4.1 mostra os tipos e os descendentes de Datasets presentes no ambiente Delphi.

**R2** – Para cada objeto Dataset, identificar qual tabela do banco de dados está sendo usada. Em objetos do tipo Query é possível que a instrução SQL a ser executada aponte para mais de uma tabela. Essa informação pode estar registrada tanto na definição do objeto Dataset (caso esteja definido no formulário como um atributo não visual), como inserida no código fonte por meio de um método apropriado.

TABELA 4.1 – Tipos de componentes Dataset em Delphi.

| Tipo de Dataset | Descrição | Classes em Delphi (descendentes da classe TDataset) |
|---|---|---|
| **Table** | Representa uma simples tabela de um banco de dados, incluindo todas as linhas (registros) e colunas (campos). | TTable, TADOTable, TSQLTable, e TIBTable |
| **Query** | Representa um comando SQL. Uma Query retorna registros como resultado da execução de um comando SQL (normalmente uma declaração SELECT), ou pode executar um comando que apenas atualiza um ou mais registros (por exemplo, uma declaração UPDATE para modificar valores na tabela). | TQuery, TADOQuery, TSQLQuery, e TIBQuery. |
| **Stored procedure** | Representa um procedimento armazenado no servidor de banco de dados. | TStoredProc, TADOStoredProc, TSQLStoredProc, e TIBStoredProc. |

### 4.2.3 Procedimentos

O conceito de procedimento compreende qualquer método[*] implementado em um módulo. A regra para identificação de um procedimento em Delphi é a seguinte:

**R1** – Um procedimento inicia com a declaração *procedure* ou *function*, seguida de um cabeçalho com o nome do procedimento, declaração dos argumentos (opcional) e tipo do valor de retorno, se for uma *function*. No caso de um procedimento pertencer a uma classe (comportando-se como método), o nome da classe deve preceder o cabeçalho por um sinal de ponto (“.”), e a assinatura do método deve estar definida na classe.

O cenário mais comum é encontrar, em aplicações legadas implementadas em Delphi[**], procedimentos como métodos de uma classe TForm (formulário), que trata um determinado evento do formulário ou de um componente pertencente a ele.

Assim como nos módulos, métricas são incorporadas à metaclasse Procedimento. Para determinar os valores das métricas, utiliza-se uma versão simplificada do método McCabe (1976), conhecido como “complexidade ciclomática”. O método é um dos mais utilizados em avaliação de medida de complexidade, aplicados em processos de qualidade de *software* (WATSON e MCCABE, 1996). As informações são as mesmas presentes na metaclasse Módulo, exceção do atributo “Pontos de Decisão” (**DP** – *Decision Point*). DP é definido por um valor inteiro positivo, que serve como medida para aferir um certo grau de complexidade do procedimento. A finalidade prática deste valor é que se pode quebrar um procedimento em pedaços menores, assim que o valor de DP ultrapassar um valor limite convencionado para a complexidade, por exemplo, um valor acima de 10. Outra forma de uso, é classificar o procedimento em graus de risco quanto à capacidade de o código receber modificações, dependendo de faixas de complexidade, conforme visto na Tabela 4.2. A seguir, discute-se a regra e o algoritmo para calcular a métrica DP em um código Delphi.

**R2** – Recupera-se o bloco do código fonte que implementa o corpo do procedimento. Para calcular DP de um procedimento, o algoritmo abaixo é executado:

[*] O termo método é empregado, no contexto da abstração Procedimento, com o mesmo sentido de rotina/função, não apenas para implementar uma operação de classe, mas também para cumprir uma tarefa definida no módulo.

[**] O perfil destas aplicações é comentado no Capítulo 1.

1. Iniciar para cada procedimento, atribuindo-se 1 a DP.
2. Adicionar 1 a DP, para cada uma das palavras reservadas: “IF”, “WHILE”, “REPEAT”, “FOR”, “AND”, “OR”, “XOR”, “GOTO”.
   Obs.: Para uma expressão composta de operadores “AND”, “OR” e “XOR” adiciona-se 1 para cada expressão.
3. Adicionar 1 para cada rótulo de uma declaração “CASE”. Quando o “CASE” tem um “ELSE”, adiciona-se mais 1.

TABELA 4.2 – Valores da Complexidade Ciclomática, segundo Watson e McCabe (1996).

| **Faixa de Complexidade (DP)** | **Risco** |
|---|---|
| 1-10 | procedimento simples, baixo risco. |
| 11-20 | procedimento complexo, risco moderado. |
| 21-50 | procedimento complexo, alto risco. |
| acima de 50 | procedimento inviável, altíssimo risco. |

Destaca-se que a apresentação de uma lista com procedimentos e seus valores de métricas de complexidade, adicionados a outras métricas, pode ser um suporte inicial na tarefa de identificar código com anomalias, candidatos à reestruturação. Assunto que é discutido, a seguir.

### 4.2.4 Reestruturar Código de Procedimentos Anômalos

Esta atividade é opcional nesta etapa, pois possivelmente nem todos os módulos e procedimentos participarão do processo de migração. A antecipação do passo de reestruturação de código é indicada no caso de todo o sistema legado ser reconstruído, servindo não apenas para um processo de migração, sobretudo para a melhoria de qualidade do próprio sistema origem. Contudo, esse compromisso nem sempre é assumido, pois os custos envolvidos são bastante elevados.

Na prática, esse passo é recorrente no processo, pois ocorre em outras etapas, como:

a) Na etapa de construção de casos de uso para reengenharia onde é feita a seleção de todos os módulos e cenários que comporão a nova aplicação. A reestruturação de código é invocada quando o engenheiro de *software* perceber que o número de procedimentos envolvidos no cenário é insuficiente para o tamanho e compreensão de um caso de uso;
b) Na etapa de construção do MCS, um dos passos é a identificação e depuração de métodos candidatos, oriundos dos procedimentos identificados na etapa de recuperação de abstrações.

É objetivo também identificar o tipo de relacionamento entre procedimentos (com métodos candidatos) e objetos DataSet. Os procedimentos escolhidos são os relevantes para a compreensão das regras de negócio, isto é, os que tratam com os dados de negócio: variáveis e métodos que direta ou indiretamente estejam vinculados a objetos DataSet. Exclui-se todos os procedimentos que não pertencem a esta regra, como aqueles que implementam funcionalidades de interface visual, por exemplo.

Satisfeitas as condições de relevância, um procedimento deve ser quebrado em pedaços menores de bloco de código que possam realizar uma única tarefa com forte coesão e baixo acoplamento. Esses blocos de código formarão os métodos que modificam o estado de um objeto DataSet. É esperado que um procedimento apresente

anomalias do tipo: acessa mais de um DataSet, modifica mais de um campo (atributo) ou lê vários atributos. Neste caso, isola-se o bloco para cada método encontrado que manipule um único DataSet, classificando o tipo de manipulação realizada sobre este objeto, conforme os valores definidos, a seguir:

- "**M**", quando o módulo é modificador de um DataSet, isto é, quando o código apresenta operações de atualização aos dados (inclusão, alteração e exclusão) de um DataSet; e

- "**O**", quando o módulo apenas recupera (observa) valores de um DataSet.

Assim como no método Fusion/RE (PENTEADO, 1996), um procedimento pode atuar sobre mais de um objeto Dataset. Neste caso, usa-se o operador "+" para denotar o tipo de manipulação sobre múltiplos objetos de dados, podendo se combinar as operações, tais como "+M", "+O", "+O/M" e "+M/O", para representar a manipulação de mais de uma tabela, observando mais de uma tabela, observando mais de uma e manipulando uma tabela, e manipulando mais de e observando uma tabela, respectivamente. Na situação "+M/+O", o procedimento tanto manipula quanto observa mais de uma tabela.

Um exemplo de procedimento com anomalias pode ser visto no código da Figura 4.10, que apresenta um procedimento com a seguinte assinatura: **Calcula_Saldo (CliID: integer, Valor: float)**. A finalidade do procedimento é atualizar o saldo de um cliente com os dados do valor (a debitar do saldo) e o código do cliente passados como parâmetros no procedimento. Após uma análise, pode-se afirmar que três blocos de código podem ser isolados formando os métodos M1, M2 e M3. O método M1 é do tipo "O" (observador do objeto rs), enquanto M2 e M3 são do tipo "M" (modificadores do estado de rs). O critério de nomeação dos métodos deve respeitar os objetivos do que cada método se propõe a executar, buscando sempre realizar uma única tarefa (regra da alta coesão) (COAD; YOURDON, 1991) (PRESSMAN, 2001).

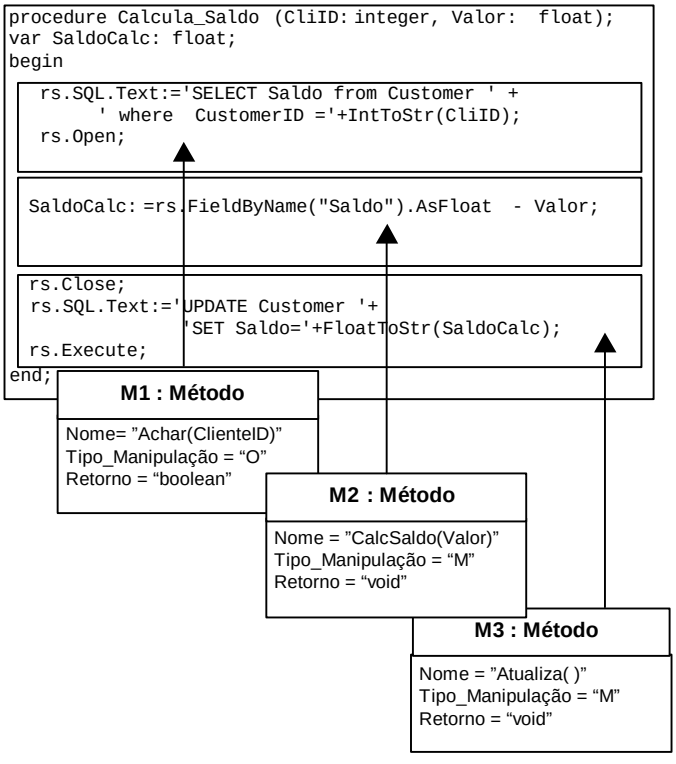



FIGURA 4.10 – Exemplo de procedimento anômalo.

Atualiza-se o metamodelo de metaclasses conceituais com duas estruturas: Procedimento e Método. O metamodelo (parcial) é visto na Figura 4.11. Em seguida, as regras para identificação de anomalias em procedimentos em Delphi são apresentadas.

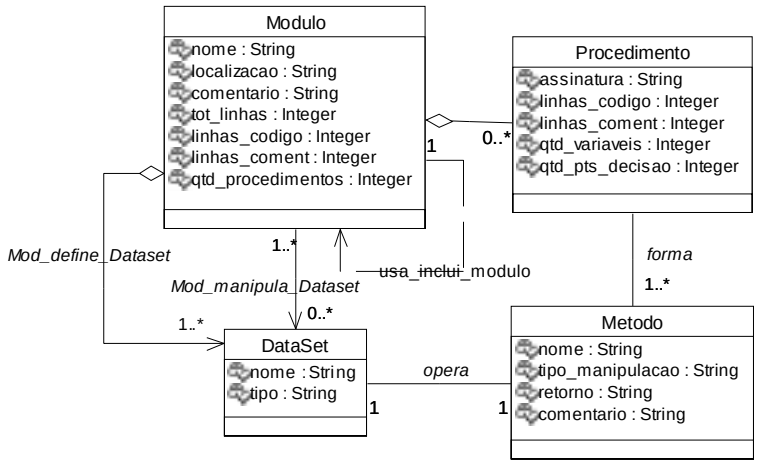



FIGURA 4.11 – Modelo com as estruturas Procedimento e Método.

#### 4.2.4.1 Regras para identificação de procedimentos anômalos em Delphi.

**R1** – Para cada módulo, listar os procedimentos que manipulam mais de um objeto Dataset. A forma de manipulação é sinalizada pela presença de uma chamada de método ou propriedade de um objeto Dataset, como é exemplificado na Tabela 4.3, que mostra alguns métodos para a classe TDataset que representa a classe base para todas as subclasses do tipo Dataset.

TABELA 4.3 – Uma classificação para alguns métodos e propriedades de objetos da classe TDataset em Delphi.

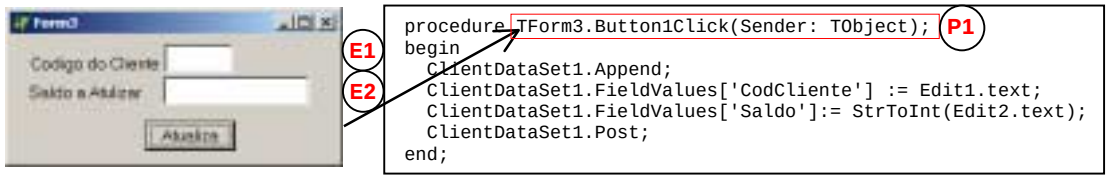
| Métodos (p/f), Propriedades (a) | Descrição resumida | Tipo de Manipulação |
|---|---|---|
| (p)Append, (p)Insert | Prepara a inclusão de um novo registro, em branco (vazio) no fim da tabela. | M |
| (p)Cancel | Cancela e libera um registro que foi preparado para inclusão ou alteração. | M |
| (p)Post | Grava (fisicamente) um registro no banco de dados, que foi preparado para inclusão e alteração. | M |
| (p)Edit | Prepara um registro para edição. | M |
| (f)Locate(...) | Localiza um registro a partir de uma chave de pesquisa. | O |
| (f)FieldByName(...), (a)FieldValues[...] | Recupera (O) ou modifica (M), no caso de uma atribuição) o valor de um campo de dado, baseado no nome do campo informado. | O ou M |

OBS: Na primeira coluna (Método e Propriedades), “p”, “f”, e “a”, significam método “procedimento”, método “função” e “atributo/propriedade”, respectivamente. Na última coluna (Tipo de Manipulação), “M” é método/propriedade que “modifica” e “O” é um método/propriedade que observa um objeto Dataset.

**R2** – Para quebrar um procedimento em blocos de procedimentos (futuros métodos candidatos), a regra geral é aplicar o conceito de *coesão funcional*: um método deve desempenhar uma, e somente uma, função identificável (PRESSMAN. 2001). Na prática, o código legado é permeado de idiossincrasias próprias de cada programador que realizou a manutenção ou implementação do código, fato que impossibilita a identificação de padrões de código pelas variantes encontradas. Por

exemplo, pode-se encontrar uma implementação que utiliza componentes visuais acoplados aos campos de um Dataset; outra que utiliza variáveis ou componentes visuais desligados dos campos, atualizando-se os valores dos campos e dos componentes visuais através de atribuições. Estes dois exemplos são vistos na Figura 4.12, em que um formulário atualiza dois campos de um Dataset.

**(a) Exemplo de código usando componentes visuais Edit1 e Edit2 (E1 e E2) desconectados aos campos.**




**(b) Exemplo de código usando componentes visuais Edit1 e Edit2 (E1 e E2) conectados aos campos.**

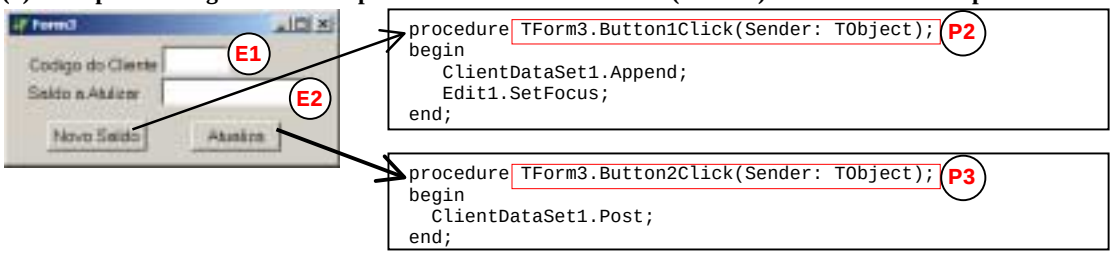



FIGURA 4.12 – Dois exemplos de códigos de procedimentos para a mesma funcionalidade.

Os exemplos da Figura 4.12 ilustram a dificuldade de realizar a atividade de reestruturação. Apesar do exemplo (b) mostrar dois procedimentos (P2 e P3) com coesão aparentemente mais forte (o desejável) e o procedimento P1, no exemplo (a), possuir um código com quatro métodos Dataset diferentes (coesão mais baixa), o exemplo (b) esconde um ardil perigoso: os controles de edição (E1 e E2) estão ligados implicitamente aos campos de dados, algo que o procedimento P1 faz de forma mais clara, através dos controles visuais desconectados dos campos, facilitando a leitura e a reestruturação do código. A recomendação é que seja feita também, se possível, a reestruturação da interface de usuário, utilizando controles visuais de edição "desligados" dos campos/atributos de dados. Neste caso, os procedimentos dos exemplos (a) e (b) ficariam da forma vista a seguir (Figura 4.13).

P1
```
procedure TForm3.Button1Click(Sender: TObject);
begin
  AdicionaNovoCliente;              // ClientDataSet1.Append;
  SetCodCliente(Edit1.text);        // ClientDataSet1.FieldValues['CodCliente'] := Edit1.text;
  SetSaldo(Edit2.text);             // ClientDataSet1.FieldValues['Saldo']:= StrToInt(Edit2.text);
  GravaCliente;                     // ClientDataSet1.Post;
end;
```

P2
```
procedure TForm3.Button1Click(Sender: TObject);
begin
   AdicionaNovoCliente;      // ClientDataSet1.Append;
   Edit1.SetFocus;
end;
```

P3
```
procedure TForm3.Button2Click(Sender: TObject);
begin
  SetCodCliente(Edit1.text);        // ClientDataSet1.FieldValues['CodCliente'] := Edit1.text;
  SetSaldo(Edit2.text);             // ClientDataSet1.FieldValues['Saldo']:= StrToInt(Edit2.text);
  GravaCliente;                     // ClientDataSet1.Post;
end;
```

FIGURA 4.13 – Reestruturação de código para os exemplos da Figura 4.12.

### 4.2.5 Objetos de Interface de Usuário

Uma interface de usuário (homem-computador) compreende todos os comportamentos do usuário e do sistema computacional que são observáveis externamente (CHI, 1985). Há uma linguagem de entrada, uma de saída para refletir os resultados e um diálogo coordenando a interação entre ambos.

A engenharia reversa de interface tipo WIMP inclui três atividades principais:

a) identificação dos objetos gráficos de telas e as dependências com os atributos das classes de negócio;

b) identificação das informações ou mensagens (do computador ao usuário e do usuário ao computador) envolvidas na execução de cada tarefa; e

c) identificação da seqüência de mensagens necessárias para cumprir a tarefa.

Para realizar essas atividades, é necessário estudar o funcionamento da interface existente, isto é, deve-se observar como o sistema interage com o usuário. Existem problemas como reconhecer uma tela única aparecendo várias vezes com informações diferentes, ou identificar as informações que permitem passar de uma tela para a tela seguinte. Para isto, há a necessidade de analisar não só a forma de interação do sistema (análise dinâmica), mas também o código fonte (análise estática).

Basicamente, os objetos que se busca identificar na interface do usuário são aqueles definidos a partir de uma biblioteca de componentes visuais do ambiente de desenvolvimento do sistema legado. De um modo geral, os ambientes de desenvolvimento visual dispõem em suas bibliotecas de componentes[*], classes que representam janelas, *widgets* para entrada de dados e saída de informação. Exemplos de componentes para entrada de dados são botões de comando, caixa de texto e caixas de combinação; no caso de componentes de saída têm-se caixas de diálogo e completos componentes para montagem de relatórios, apresentados em forma de *preview* ou gerados para dispositivos de impressoras.

Portanto, encontram-se bibliotecas em ambientes como Delphi e Java, por exemplo, VCL (*Visual Componente Library*) e AWT (*Abstract Window Toolkit*), respectivamente, que servem de base para muitas outras bibliotecas disponibilizadas por terceiros. No enfoque desta dissertação, que considera o ambiente Delphi como domínio, a ênfase será naqueles componentes mais comumente usados na biblioteca VCL para projeto de interface de usuário através de formulários. Nas subseções que seguem, são apresentados o vocabulário típico e uma taxonomia para os componentes visuais, baseados na tese de Moore (1998), em seu projeto chamado MORPH. Em seguida, regras para extração dos objetos de interface de usuário são discutidas e exemplificadas no domínio Delphi.

#### 4.2.5.1 Conceitos e Classificação

Para não suscitar dúvidas quanto aos termos e conceitos utilizados, é necessário tornar claro alguns vocábulos envolvidos no processo de reengenharia de interface. Nesta subseção são introduzidos os mais importantes termos. Conceitos e definições mais completas podem ser encontrados em Foley *et al*. (1990), Dix et al. (1993) e Hix & Hartson (1993).

---

[*] Também conhecidos por *toolkits*, são bibliotecas de rotinas usadas por programadores para a implementação de detalhes de baixo nível. Em geral, quando se emprega um *toolkit* na implementação de uma interface gráfica, o controle do diálogo reside no componente visual. A implementação resultante é um conjunto de *widgets*. *Widget* é um objeto de interação com apresentação e comportamento bem definidos como botões, menus e outros. A interface desenvolvida com o uso de um *toolkit* geralmente interage com a aplicação através de *callbacks*, que são manipuladores de eventos do *widget* codificados na aplicação.

*4.2.5.1.1 Diálogo*

É a estrutura de conversação entre o usuário e o sistema de computador. Inclui a seqüência das interações entre o usuário e o sistema, e a estrutura sintática das entradas e saídas (DIX *et al*., 1993). O termo diálogo representa o protocolo que coordena a comunicação existente entre o ser humano e o sistema de computação, isto é, a troca de símbolos e ações entre o usuário e o computador.

*4.2.5.1.2 Interação*

É a tarefa, por parte do usuário, da entrada de uma unidade de informação para a aplicação (FOLEY *et al.*, 1990). Dependendo do ambiente da aplicação, várias técnicas de interação[*] podem ser empregadas. Em um ambiente tipo WIMP, por exemplo, o usuário pode selecionar em uma lista de combinação, uma opção para o sistema através de um clique acionado por um *mouse*. Em um ambiente orientado a texto, o usuário pode fornecer a opção selecionada teclando em um caractere pelo teclado.

Moore (1998) define uma classificação para os tipos de interação, organizados em quatro conceitos:

a) **Seleção** – o usuário escolhe um elemento de um conjunto de alternativas. Exemplos em ambiente do tipo WIMP (a partir deste ponto, todos os demais exemplos seguirão este tipo de ambiente) são os menus em cascata, lista de seleção e botões de combinação.
b) **Entrada Textual** – o usuário transcreve uma seqüência de caracteres como dado de entrada. Exemplo: caixa de texto (linha simples e tamanho limitado) e memorandos (múltiplas linhas).
c) **Entrada Quantificada** – o usuário especifica um valor numérico que satisfaça um critério de valores dentro de um intervalo. Normalmente, os ambientes dispõem de *widgets* com máscara e formatos numéricos.
d) **Entrada Posicional** – o usuário especifica uma posição (ponto) espacial nas coordenadas de um plano bidimensional (x, y) ou tridimensional (x, y, z) da tela. Esta técnica de interação é possível com uso de dispositivos apontadores (*mouse*) ou simulando-se através de uma combinação de teclas de atalho.

Além desses quatro conceitos, é importante identificar o domínio do **tipo de dado da interação,** isto é, o tipo do valor do dado inserido na unidade de entrada durante uma tarefa de interação. Dependendo da forma de implementação da tarefa de interação o tipo pode ser *string*, caractere ou numérico.

*4.2.5.1.3 Apresentação*

Também denominada *saída*, representa elementos que são percebidos fisicamente pelos usuários (sentidos da visão e audição) quando usam a aplicação (DIX *et al*., 1993). Exemplos de técnicas de apresentação incluem rótulos de texto, caixas de mensagens e barras de progressão.

Pela taxonomia definida por Moore (1998), dois tipos de apresentação podem ser encontrados:

**Saída Textual** – saída típica com apresentação de uma seqüência de texto ao usuário. Exemplos: rótulos de texto, caixas de diálogo.

[*] A técnica de interação representa a forma como a interação é realizada.

**Saída Quantificada** – a saída é disposta ao usuário em forma de valor numérico (inteiro, em ponto decimal, percentual, etc.). Exemplo: valores numéricos exibidos em caixa de texto e em barras de progressão no estilo *gauge*.

*4.2.5.1.4 Hierarquia de abstração de componentes visuais, segundo MORPH*

Em MORPH, Moore (1998) descreve uma estrutura hierárquica de classes para organizar os componentes de diálogo, seguindo a classificação apresentada anteriormente.

A estrutura é organizada da seguinte forma: o nó raiz representa o componente de diálogo, base para todos os outros nós descendentes. O nível abaixo do nó raiz representa o conjunto de componentes para as tarefas de interação e apresentação. No nível seguinte, os nós representam os tipos de técnicas de interação e apresentação. Uma visão parcial da hierarquia pode ser vista no diagrama apresentado na Figura 4.14.

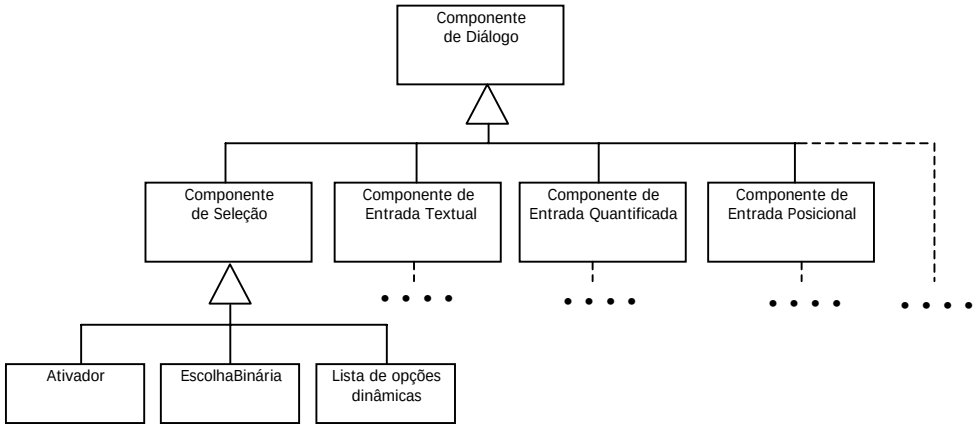



FIGURA 4.14 – Estrutura hierárquica de abstração em MORPH (Moore, 1998).

*4.2.5.1.5 Estrutura de abstração proposta na metodologia*

Nesta proposta do trabalho, a estrutura é personalizada para o contexto das linguagens visuais. Como a unidade de apresentação é um formulário, a estrutura parte deste componente para montar a hierarquia de (meta) classes de componentes de diálogo. Acrescentam-se alguns componentes de apresentação importantes para identificar objetos visuais no código legado: componentes de imagem e de agrupamento (contêiner) de outros componentes, representados por ApImagem e ApGrupo, respectivamente. A Figura 4.15(a) exibe a estrutura completa de metaclasses para os tipos de componentes de diálogo mais comuns, e ao lado (b) uma versão simplificada do mesmo modelo, usando o estereótipo <<ComponenteDlg>>, para ocultar os detalhes da estrutura hierárquica completa do modelo. A definição detalhada do metamodelo, incluindo relacionamentos de componentes de diálogo com os objetos de dados, é apresentada no Anexo 1.

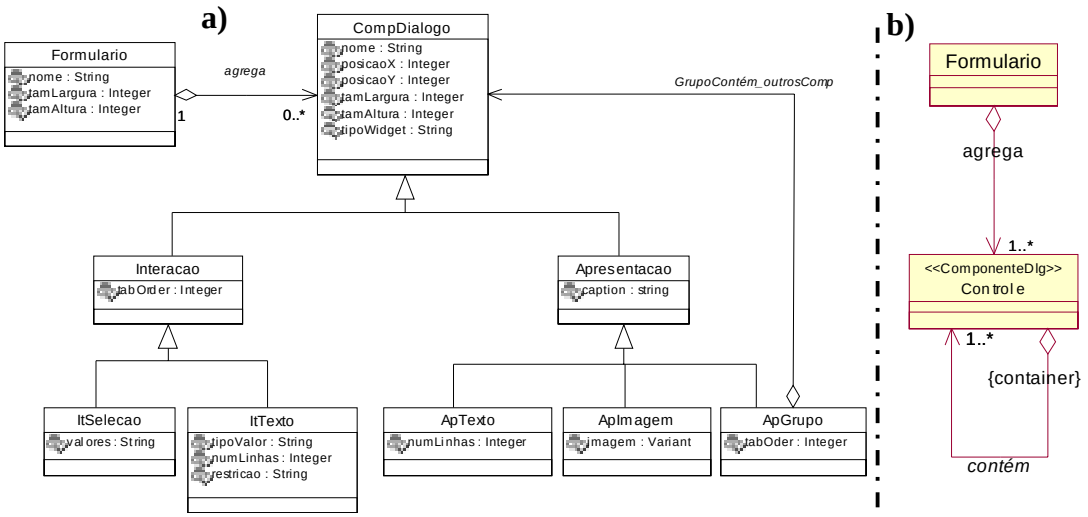



FIGURA 4.15 – Estrutura com metaclasses para representação dos componentes visuais (a) e sua versão simplificada (b).

### 4.2.5.2 Regras para a identificação de objetos de interface de usuário

A técnica empregada para a identificação dos objetos visuais é a análise estática do código, onde estão definidos os componentes de diálogo*. Normalmente, linguagens visuais registram esse código em um bloco separado do código de instruções do programa da aplicação. Na Figura 4.17, vê-se duas definições de formulário nos ambientes VB e Delphi, para uma interface apresentada na Figura 4.16.

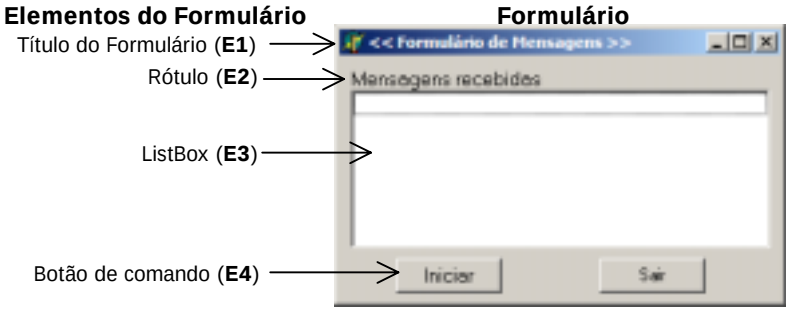



FIGURA 4.16 – Visão de um formulário com alguns elementos (componentes de diálogo) identificados.

Na visão do formulário (Figura 4.16), alguns elementos são destacados (códigos E1, E2, E3 e E4) para facilitar a identificação no código de definição dos objetos, vistos na Figura 4.17. Pela definição da sintaxe do código, cada objeto possui um nome único e o tipo da classe do componente visual, seguido das propriedades específicas para a classe do objeto no ambiente de desenvolvimento.

As regras para identificação de objetos visuais no ambiente Delphi, seguindo a classificação apresentada na estrutura de metamodelos da Figura 4.15, é a seguinte:

**R1** – Para cada módulo, localizar o arquivo com a definição dos componentes visuais do formulário. Um arquivo do tipo “dfm” armazena estes componentes e possui o mesmo nome do módulo, com a seguinte estrutura de nome: ”*<NomeDoMódulo>*.*<DFM>*”.

* A análise dinâmica da interface será realizada na etapa seguinte, a de construção dos casos de uso para reengenharia.

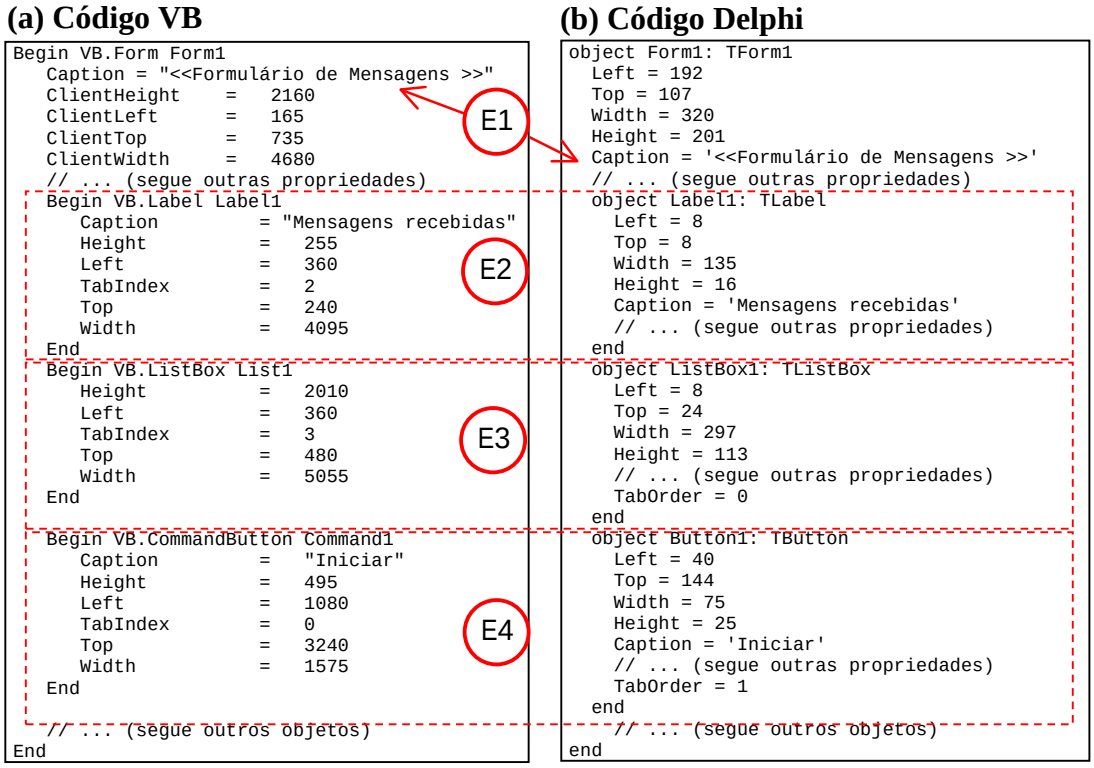



FIGURA 4.17 – Definição de componentes de diálogo de um formulário em duas linguagens visuais.

**R2** – Para recuperar os objetos visuais e as informações relevantes da definição do formulário, segue um mapeamento para obter propriedades importantes de cada objeto, como os exemplos apresentados na Tabela 4.4 (o Anexo 2 apresenta uma lista mais completa do mapeamento). Entretanto, algumas questões devem ser observadas:

a) Para os componentes que possuem conexão (acoplamento) com objetos de dados, acrescente-se as propriedades *Datasource* e *Datafield* às informações relevantes a serem recuperadas.

b) Para objetos de interação em que a propriedade "*ReadOnly = True*", o componente terá comportamento de diálogo de apresentação (saída de informação).

TABELA 4.4 – Exemplos de mapeamento de classes de objetos visuais em Delphi.

| **Classe do Objeto** | **Classificação do componente de diálogo** | **Propriedades relevantes do objeto** |
|---|---|---|
| TForm | Contêiner raiz | *Caption*, tamanho (*Width, Height*) |
| TEdit | componente de interação textual | *TabOrder*, *Text*, posição (*Left, Top*) e tamanho (*Width, Height*) |
| TDBText | componente de interação textual | *DataSource*, *DataField*, *TabOrder*, posição (*Left, Top*) e tamanho (*Width, Height*) |
| TComboBox, TDBComboBox | componente de interação de seleção | Posição (Left, Top) e Tamanho (Width, Height), TabOrder, Items.Strings |
| TLabel, | componente de apresentação textual | *Caption*, Posição (Left, Top) e Tamanho (Width, Height) relativos ao formulário (medidos em *pixels*). |

## 4.3 Segunda Etapa – Construir Casos de Uso para Reengenharia

O objetivo desta etapa é modelar as funcionalidades e o comportamento do *software* legado, fundamental para a compreensão do sistema legado. Utilizar-se-á o mecanismo conhecido como caso de uso (*use case*), definido inicialmente por Jacobson (1992, 1995) e incorporado em UML (UNIFIED MODELING LANGUAGE, 2003).

A definição formal de um caso de uso, segundo UML, é "*um conjunto de seqüências de ações que um sistema desempenha para produzir um resultado observável de valor a um ator específico*". Para Jacobson *et al*. (1992), um caso de uso é um documento narrativo que descreve como um ator (agente externo) interage com o sistema, através de uma seqüência de eventos, para completar um processo.

Recuperar, de forma automática um caso de uso é um desafio para a engenharia reversa. Os autores da UML demonstram essa dificuldade:

> *"...Aplicar automaticamente a engenharia reversa a um diagrama de caso de uso está bem longe do estado da arte, simplesmente por haver uma perda de informações quando se passa da especificação do comportamento de um elemento para o modo como ele é implementado. Entretanto, é possível estudar um sistema existente e discernir seu comportamento pretendido, que você pode então colocar sob a forma de um diagrama de caso de uso."* (BOOCH, JACOBSON, e RUMBAUGH, 1997, p. 238)

Quando a documentação do sistema é precária, a alternativa é realizar uma inspeção, com ajuda de um especialista, das funcionalidades que se pretende colocar na forma de um diagrama de casos de uso.

Booch, Jacobson e Rumbaugh (1997) apresentam também diretrizes gerais para fazer a engenharia reversa de um diagrama de casos de uso, sob dois aspectos: casos de uso baseados em atores e os baseados em eventos. As diretrizes são:

a) Identificar cada ator a partir da interação com o sistema.
b) Para cada ator, considerar a maneira como esse ator interage com o sistema, como altera o estado do sistema, ou responde a algum evento. No caso de um *software* com interface do tipo WIMP, essas interações podem ocorrer por meio de botões, menus, caixas de texto ou grades (*grids*). O responsável pela ativação do evento deve ser considerado como o ator do caso de uso.
c) Traçar o fluxo de eventos do sistema executável relativo a cada ator. Iniciar com os fluxos normais e somente depois considerar os caminhos alternativos.
d) Agrupar os fluxos relacionados, declarando um caso de uso correspondente. Considerar a modelagem de variantes, usando relacionamentos do tipo estendido (*extends*) e considerar a modelagem de fluxo comum pela aplicação de relacionamentos de inclusão (*includes*).
e) Representar esses atores e casos de uso em um diagrama de caso de uso e estabelecer seus relacionamentos.

Na abordagem proposta, a criação de casos de uso (*use cases*) para reengenharia (UC/Re é o termo empregado no texto) é um importante mecanismo não apenas para capturar os requisitos do sistema legado, mas como ferramenta para identificar aspectos dinâmicos da aplicação, mostrando a interação entre as abstrações em forma de objetos instanciados em um cenário de execução.

O objetivo da ferramenta UC/Re é estender o conceito de caso de uso, definido em UML, para a realidade da reengenharia de *software*. UC/Re provê um arcabouço de mais alto nível de abstração para compreensão dos componentes que farão parte do

processo de migração do sistema legado. Além disso, o uso de UC/Re permitirá aos projetistas um mecanismo para acompanhamento e validação dos requisitos funcionais e operacionais do novo sistema.

Para mapear um UC/Re a partir da execução de um cenário da aplicação legado, parte-se das diretrizes de engenharia reversa apresentadas anteriormente pelos autores de UML, e seguindo outras diretrizes e técnicas. Jacobson *et al*. (1992), por exemplo, descreve-se os casos de uso como grafos de transição de estados, no qual cada passo realizado do caso de uso é mapeado em um estado e cada um dos eventos estimulados/gerados no diálogo entre ator/sistema é mapeado em uma transição de estado no grafo.

Outra técnica que pode auxiliar é a realização de um caso de uso na forma de diagrama de seqüência ou como descrição narrativa do cenário. Cenário é uma descrição formal de um fluxo de eventos que ocorre durante a instância de um caso de uso, que na prática corresponde a uma representação textual de um diagrama de seqüência. Na Figura 4.18, é apresentado um exemplo de cenário para o caso de uso "Incluir Cliente", em que um vendedor (ator) atende um cliente a ser cadastrado em um sistema tipicamente implementado em uma arquitetura cliente/servidor "*fat-client*".

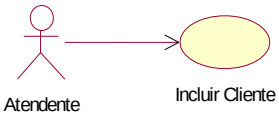
Caso de uso: **Incluir Cliente**

**a)** Descrição do Cenário

– Fluxo normal de eventos

Ator: **Vendedor/Atendente**

1 - Atendente abre janela de cadastro de cliente
2 - Atendente clica no botão "Incluir"
3 - Atendente informa os dados do cliente: código, nome, endereço, etc.
5 - Atendente clica no botão "Salvar"
6 - Atendente clica no botão "Sair"




**b)** Diagrama de Seqüência

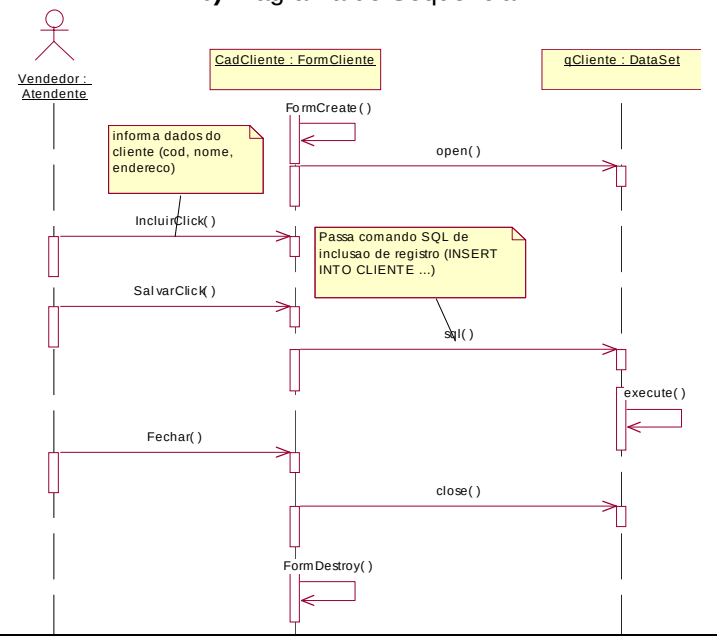



FIGURA 4.18 – Exemplo de caso de uso apresentado em um Cenário (a) e Diagrama de Seqüência (b).

Para recuperar as mensagens trocadas entre ator e sistema, bem como os objetos envolvidos no cenário, como é mostrado no exemplo da Figura 4.18(b), as mensagens no Diagrama de Seqüência poderiam ser obtidas através de um rastreamento de execução, que fica registrado em um arquivo de "*log*". Essa estratégia pode ser realizada por meio de funções especializadas que são inseridas no próprio código, por exemplo. Outra forma de fazê-lo é executando no modo de depuração (*debugging*), recurso geralmente disponibilizado por ferramentas do ambiente de programação. Esse procedimento é uma das atividades envolvidas na etapa de construção dos UC/Re.

Ao todo, quatro passos principais são necessários para essa etapa, que são:

1. Instrumentar o código legado,
2. Executar os cenários para UC/Re,
3. Analisar o rastreio de execução,
4. Registrar os dados do UC/Re

Assim como nas etapas anteriores, a atividade “Reestruturar Código...” é novamente invocada para dar suporte à tarefa de melhoria do código legado. A Figura 4.19 mostra o fluxo de atividades da etapa “Construir UC/Re”. Em seguida, os passos desta etapa são discutidos em mais detalhes.

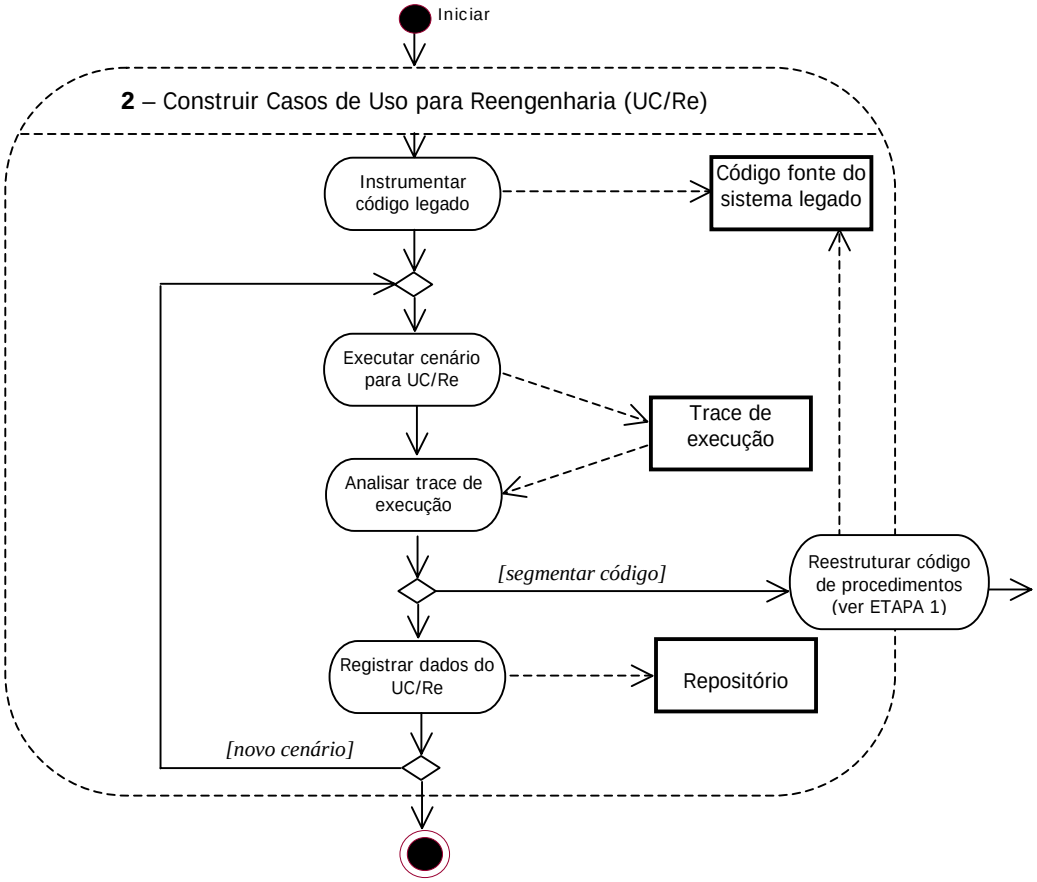



FIGURA 4.19 – Fluxo das atividades envolvidas na etapa “Construir UC/Re”.

### 4.3.1 Instrumentar o Código Legado

O objetivo deste passo é fornecer um mecanismo de rastreio dinâmico (*trace*) de execução do código. Como Delphi gera uma versão compilada do código executável, a alternativa é introduzir comandos de depuração no código legado. As informações recuperadas dizem respeito a chamadas de procedimentos e edição de controles visuais de interação que são ativados na execução de um cenário. A seguir, é apresentado e discutido o algoritmo para realizar esta tarefa.

#### 4.3.1.1 Algoritmo para Instrumentar código legado Delphi.

1) Cria-se um módulo (uma *unit* Delphi) chamado "DebugUnit.pas". Este módulo conterá uma classe de depuração, com o objetivo de coletar as informações da análise dinâmica da aplicação legada. Uma análise dinâmica corresponde a um cenário de execução para um caso de uso escolhido para a reengenharia. A classe de depuração possui a seguinte estrutura de atributos e métodos:

```
type Debug = class of
      Lista:  TStringList;
      procedure mensagem(msg: string);    // registra o procedimento chamado e controle ativado
      procedure gravaTrace;  // grava o log de trace em um arquivo.
end;
```

2) Acrescenta-se DebugUnit ao projeto do sistema. Pelo ambiente Delphi, isso pode ser feito através do comando do menu principal (*Project* / *Add to project.*.).
3) Define-se uma variável objeto da classe Debug, pública para o sistema. A definição da variável, a partir do programa principal do sistema, torna a visibilidade possível a outras *units* do sistema.
4) Para cada *unit* do sistema, executar os procedimentos a seguir.
   4a) Introduzir a chamada ao módulo "DebugUnit.pas". Acrescenta-se uma chamada na declaração *uses unit1, unit2, ...., DebugUnit*.
   4b) Inserir após a primeira declaração *begin* do corpo do procedimento, a chamada ao método *mensagem*, da classe Debug. Antes, deve-se recuperar o cabeçalho do procedimento que servirá de parâmetro de *mensagem*, desta forma:

```
procedure|function <cabeçalho_do_procedimento>;
....
begin
Debug.Mensagem ('CALL => '+ <procedure|function cabeçalho_do_procedimento>);
        ..... (segue)

end; // fim
```

   4c) Inserir o procedimento para registrar o evento de entrada em um controle visual. O procedimento possui a seguinte definição:

   - Na classe *Tform* da *unit*:

```
type
  Tform<NomeForm...> = class(TForm)
  // .....
        procedure enterControl(Sender: TObject);
    private
      .....
  end;
```

   - Na seção *implementation*, introduzir a implementação do método **enterControl:**

```
implementation
  //  . . . .
        procedure Tform<NomeForm...>.enterControl(Sender: TObject);
        begin
            Debug.Mensagem( 'ENTRY =>  '+ Self.ActiveControl.Name);
        end;
  // . . . .
end. // fim da unit
```

O método **Self.ActiveControl.Name** recupera o nome do controle que está de posse do foco no formulário.
- Em seguida, atualiza-se a chamada ao manipulador do evento *onEnter* de cada controle visual do formulário (somente para os controle que não possuam *onEnter*). Para isso, abre-se o arquivo de definição do formulário (algo como "nomeUnit.DFM") associado à *unit* em questão. Atribui-se, na propriedade *onEnter*, o valor "*enterControl*", para cada objeto visual de interação (controles com a propriedade *TabOrder*) do formulário, como mostra o exemplo abaixo:

```
object nomeForm: Tform_NomeForm
  // . . . .
  object fldSerie: TDBEdit
        //  . . . . (propriedades do objeto)
           TabOrder = 1            // propriedade TabOrder indica que o controle é um controle de interação
           OnEnter = enterControl      // acrescentar a propriedade OnEnter com esta sintaxe.
  end
 //. . . . (segue outros objetos)
end  // Fim da definição dos controles do formulário.
```

### 4.3.2 Executar os Cenários para UC/Re

Após a instrumentalização concluída, o sistema legado deve ser compilado com as modificações introduzidas.

Um planejamento deve ser feito para agrupar os casos de uso que farão parte do processo de migração. A execução de um cenário deve ser acompanhada de um usuário especialista no domínio da aplicação legada, a fim de resolver dúvidas inerentes ao próprio sistema.

Para ilustrar a demonstração deste passo e dos próximos, um exemplo de cenário baseado em um caso de uso de um sistema de controle de estoque é apresentado a seguir. O sistema será discutido em mais detalhes no capítulo referente ao estudo de caso.

O Sistema de Gestão de Estoque (SGE) implementa, em um de seus módulos, a tarefa de cadastrar e atualizar o estoque a partir de compras registradas em uma nota fiscal. O caso de uso então escolhido é "Efetuar Compras", que realiza o cadastramento da nota fiscal (NF) de compra de produtos e a atualização do estoque, a partir da entrada dos produtos comprados. O usuário (ator principal) responsável por esta tarefa é o encarregado do almoxarifado da empresa, o almoxarife. A interface do usuário é mostrada na Figura 4.20, capturada com algumas informações preenchidas nos campos de edição do formulário; alguns agrupamentos de informações e outros detalhes importantes estão destacados na figura.

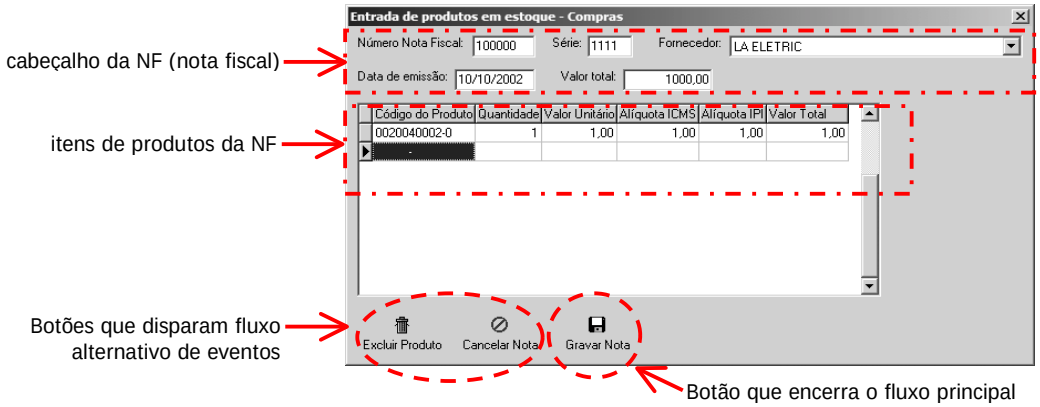



FIGURA 4.20 – Formulário de entrada de dados da nota fiscal do sistema exemplo.

Um cenário básico para a tarefa do caso de uso “Efetuar Compras”, seria:

1 – Almoxarife abre o formulário “Entrada de Produtos em Estoque - Compras”, a partir do menu principal de opções do SGE.

2 – Almoxarife informa os dados do cabeçalho da nota fiscal: número da nota fiscal, série, fornecedor, data da emissão e o valor total da nota. Para a informação do fornecedor, o sistema exibe em uma caixa de combinação todos os nomes dos fornecedores cadastrados; portanto, é necessário que o fornecedor esteja previamente cadastrado no sistema (*pré-condição*).

3 – Em uma grade de itens, são informados os produtos adquiridos na compra, com os seguintes dados: código do produto, quantidade, valor unitário, as alíquotas de ICMS e IPI. É pré-condição que o ator conheça o código de um produto previamente cadastrado no sistema. Caso o almoxarife desconheça o código do produto, o sistema disponibiliza consulta a uma lista de produtos.

**Fluxo alternativo***: o almoxarife pode excluir um item de produto lançado na grade, bastando marcar um produto na grade e clicando no botão “Excluir Produto”, localizado abaixo da grade.*

4 – O sistema calcula automaticamente o valor total da compra daquele produto informado.

5 – Após o último item de produto lançado, o almoxarife clica no botão “Gravar Nota”.

**Fluxo alternativo:** *O sistema abre uma janela de diálogo e solicita a confirmação para efetuar a atualização do estoque e o cadastro da nota fiscal, operação internamente realizada pela rotina “Processar Nota”.*

**Fluxo alternativo (ponto de extensão)***: o almoxarife pode cancelar toda a operação, clicando no botão “Cancelar Nota”.*

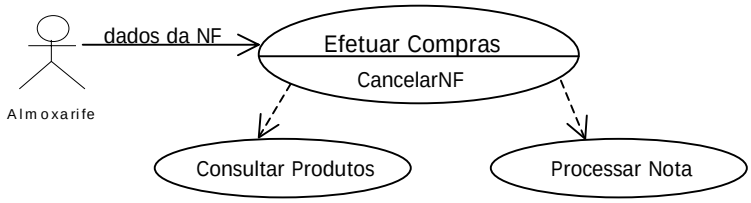



FIGURA 4.21 – Diagrama de casos de uso para o cenário exemplo.

A Figura 4.21 mostra o diagrama do caso de uso “Efetuar Compras”, incluindo os casos de uso “Consultar Produtos” e “Processar Nota”, além do ponto de extensão “CancelarNF” (Cancelar Nota Fiscal). Esses casos de uso adicionais foram identificados a partir dos fluxos alternativos do cenário principal. No encerramento do cenário, o mecanismo de rastreio introduzido no código produzirá um histórico (*log*) com os eventos ocorridos. A seguir, discute-se este documento.

### 4.3.3 Analisar o Rastreio de Execução

O *log* produzido conterá uma listagem de chamadas a procedimentos e eventos que ocorreram na execução do cenário do UC/Re. A lista deve ser analisada pelo engenheiro de *software*, verificando o nível de granularidade adequada para a compreensão dos eventos/transições de estado do caso de uso. Se o nível de detalhe é insuficiente para o entendimento do cenário, isso sinaliza que os procedimentos envolvidos têm de ser reestruturados em código ou em significância, isto é, segmentá-los no caso de serem procedimentos monolíticos e renomeá-los, caso possuam identificadores difíceis de ler.

Pela execução do caso de uso anterior, acompanhada do mecanismo de rastreio, um histórico possível para o cenário seria aquele visto na Figura 4.22. Para facilitar a identificação do tipo de evento, introduz-se em cada ocorrência um marca como EVENT, CALL e ENTRY representando um evento, chamada de procedimento e entrada de dados pelo usuário, respectivamente. O significado das marcas é detalhado no Capítulo 5, sobre a proposta da ferramenta de apoio ao processo de reengenharia.

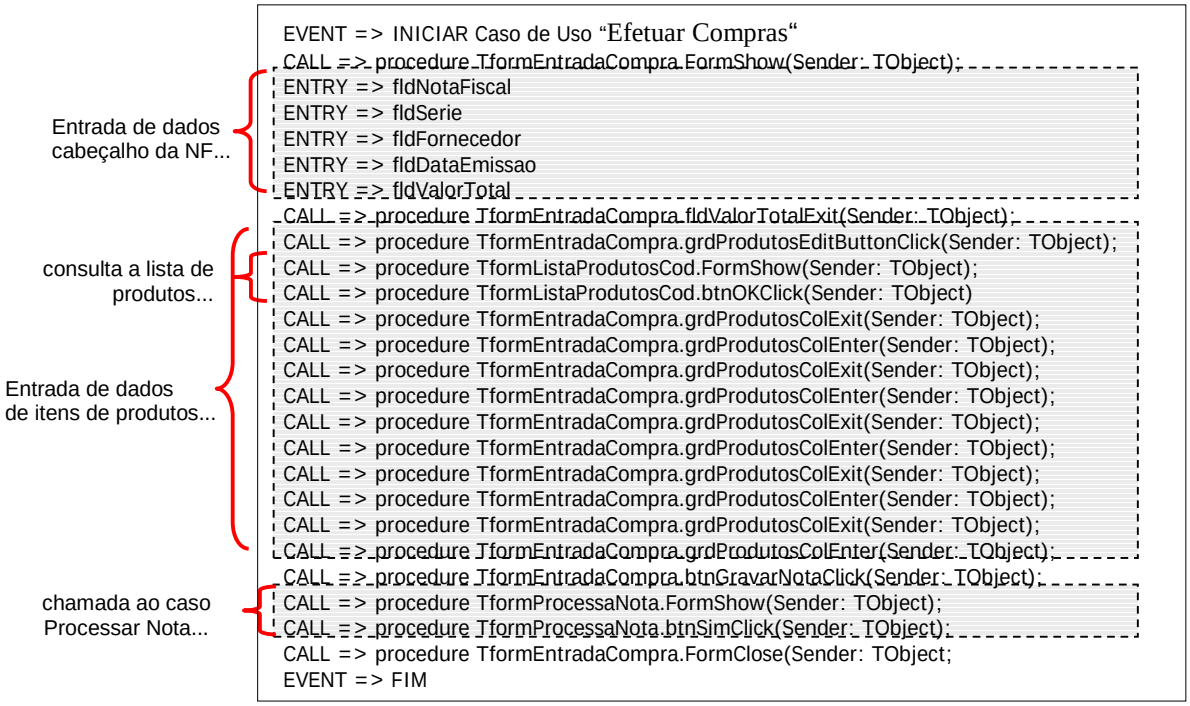



FIGURA 4.22 – *Log* com o histórico de eventos disparados para o caso de uso "Efetuar Compras".

Na análise do histórico gerado pelo *trace*, percebe-se as correlações da descrição do cenário básico com os eventos listados no *log*. Para compreender melhor as informações do *log*, tanto os campos de edição (controles visuais) de entrada de dados, quanto as chamadas a procedimentos, que foram recuperados no primeiro passo da etapa "Recuperar Abstrações", podem trazer outros significados em relação a outras abstrações, como é mostrado na Tabela 4.5. O registro no repositório das abstrações recuperadas permite realizar uma referência cruzada dos campos e tabelas com os controles visuais e os procedimentos ativados. Outra aplicação do *log* é construir o diagrama de transição de estado, como pode ser visto na Figura 4.23.

TABELA 4.5 – Abstrações e manipulações de objetos de dados do caso de uso "Efetuar Compras"

| **Abstração** | **DataSet** | **Manipulação** |
|---|---|---|
| procedure TformEntradaCompra.FormShow | dmBaseDados.tblNotaFiscal<br>dmBaseDados.tblProdutos<br>dmBaseDados.tblEntrada<br>dmBaseDados.tblFornecedores | +O/M |
| fldNotaFiscal | dmBaseDados.tblNotaFiscal (NumeroNotaFiscal) | M |
| fldSerie | dmBaseDados.tblNotaFiscal (Serie) | M |
| ( . . .) | ( . . .) | ( . . .) |
| TformProcessaNota.FormShow | dmBaseDados.tblProdutos<br>dmBaseDados.tblHistorico<br>dmBaseDados.tblNotaFiscalItens<br>dmBaseDados.tblEntrada.First | +M/O |

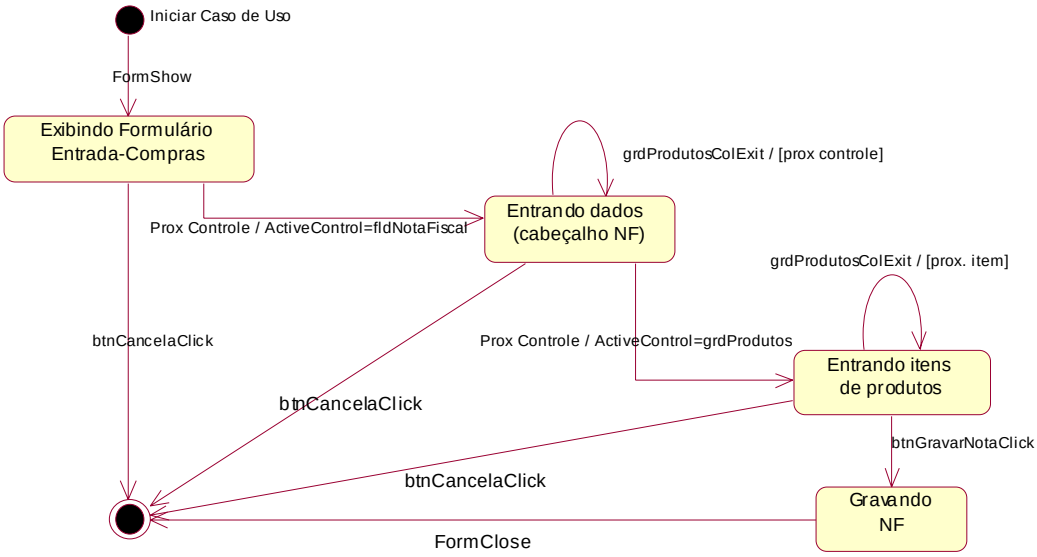



FIGURA 4.23 – Diagrama de Transição de Estados para o caso de uso “Efetuar Compras”.

Outra maneira de compreender o funcionamento interno dos UC/Re é pela engenharia reversa dos seus componentes principais, tais como as classes de formulário com atributos e métodos (procedimentos), além das relações de dependência com outros componentes. Apesar desses componentes terem sido identificados na etapa anterior (etapa “Recuperar Abstrações”) e consultadas a partir de um repositório, a visualização por meio de diagramas no padrão UML, recuperados por ferramenta de visualização, fornecerá detalhes maiores. Nas Figuras 4.24 e 4.25, vê-se os diagramas de classe e de seqüência pertencentes ao UC/Re “Efetuar Compras”. A Figura 4.24 é um exemplo do uso combinado de uma ferramenta de visualização de abstrações* citada anteriormente; nota-se a correlação de dependência do módulo principal (*EntraCompras*) do UC/Re com outros módulos do sistema e a representação do formulário em forma de um diagrama de classe UML.

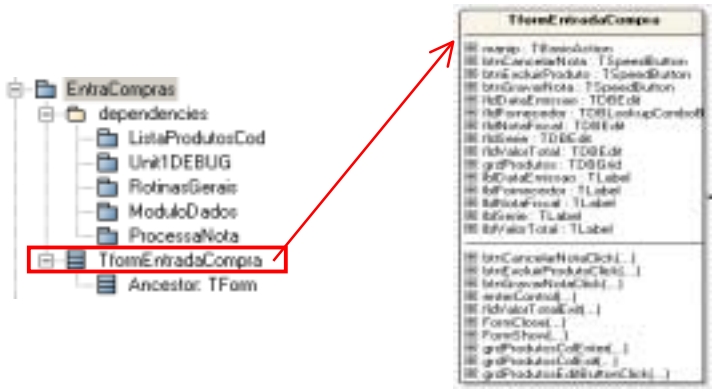



FIGURA 4.24 – Diagrama de Pacotes e de Classes para o caso de uso “Efetuar Compras”.

* A ferramenta utilizada para recuperar a visão de pacotes (módulos) e classes, é ESS-Model, que será discutida no Capitulo 5.

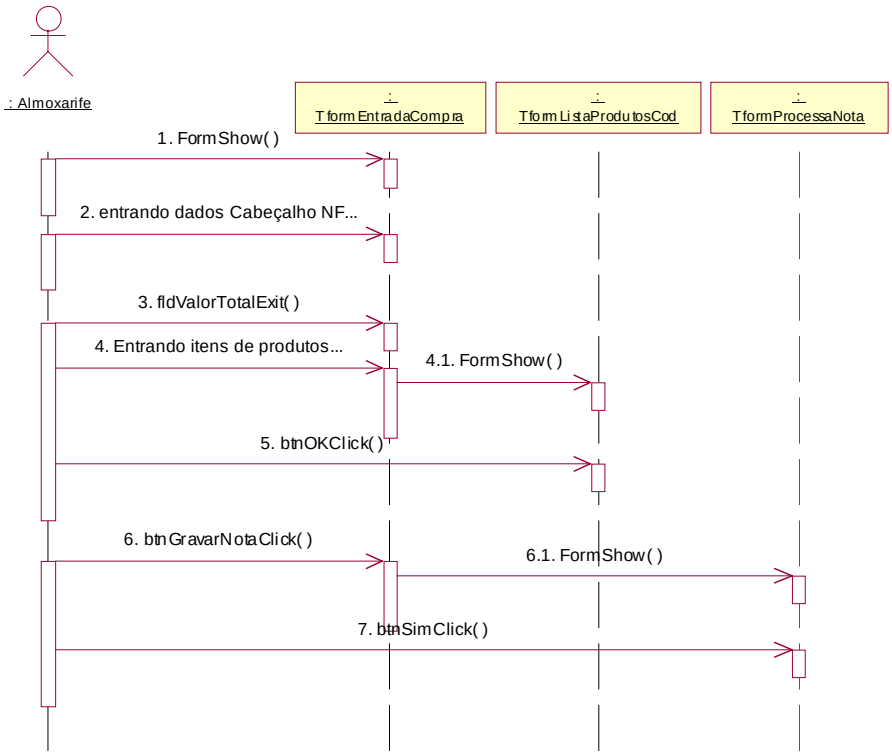



FIGURA 4.25 – Diagrama de Seqüência para o caso de uso “Efetuar Compras”.

Na Figura 4.25, em que um diagrama de seqüência mostra a troca de mensagens entre os objetos do UC/Re, é importante destacar a forma como a arquitetura do domínio do sistema legado é reproduzida. Na arquitetura definida nesse tipo de domínio, os formulários (janelas) cumprem o papel de objeto de controle e apresentação para um caso de uso específico. Um objeto formulário contém uma máquina de transição de estados que controla o comportamento dinâmico do objeto de controle, em função de seu estado corrente e dos eventos que provocam as mudanças de estado. A dificuldade maior nessa arquitetura, é rastrear os eventos automáticos provocados pelos objetos, herdados de classes do ambiente e não codificados pelos programadores. Por exemplo, os campos de edição de entrada de dados (colunas) da grade de produtos são embutidos no contêiner, não podendo ser recuperados um a um de forma direta, no histórico gerado pelo *trace* de execução. Para ilustrar este fato, no *log* da Figura 4.22, os eventos disparados e registrados como ”CALL procedures” *TformEntradaCompra.grdProdutosColExit(...)* e *TformEntradaCompra.grdProdutosColEnter(...)* , são recuperados em resposta a eventos do próprio contêiner e não de uma coluna específica da grade, dificultando a identificação de qual controle visual de entrada de dados está sendo executado. A solução é mapear visualmente qual dos controles (colunas) da grade de produtos está sendo solicitado, e registrá-lo manualmente no *log*, em diretivas como, por exemplo, “*EVENT -> Sistema pede Código do produto*”, “*EVENT -> Sistema pede ...*”, e outros controles de entrada que ocorrerem.

No âmbito da proposta, os diagramas de seqüência, de estados e de UC/Re serão úteis na implementação, como guia para o projeto e validação do sistema. E, uma vez (re)construídas as classes conceituais de negócio e as interfaces de usuário, que serão discutidas nas próximas etapas, será necessário rever os UC/Re em função das novas abstrações geradas.

### 4.3.4 Registrar os Dados do UC/Re

Embora informações básicas como o nome, descrição e atores que participam do caso de uso sejam necessariamente empregados, a UML não define um formato rígido. Para Larman (1995), o formato pode ser adaptado conforme as necessidades da documentação para facilitar a clareza da documentação. O autor também descreve casos de uso em dois níveis de detalhe: o essencial e o real. Em casos de uso essenciais apenas o diálogo trocado entre ator e sistema são descritos em nível alto de compreensão. Nos reais as funcionalidades são detalhadas em nível de objetos intrínsecos ao sistema, descrevendo-se mensagens trocadas entre ator, sistema e os objetos do sistema.

Em geral, um caso de uso deve incluir as seguintes informações:

a) **Identificação**: Definição única e representativa do caso de uso.

b) **Comentários** breves sobre os objetivos do caso de uso;

c) **Ator**: o usuário que utiliza o sistema. Inclui tanto humanos quanto outros sistemas computacionais.

d) **Requisitos**: contrato que define fatos que o caso de uso deve permitir ao ator efetuar. Preferencialmente, usa-se um termo que lembre a ação sobre o objeto que participa diretamente do caso de uso, tais como <registrar pagamento>, <incluir cliente>, <liberar crédito>, etc.;

e) **Restrições**: regras sobre o que pode e não pode ser feito. Inclui pré-condições e pós-condições que devem ser verdadeiras antes e depois da realização de um caso de uso. *Invariantes* também devem ser respeitadas, como afirma que um fornecedor deve possuir um número de CGC único para identificação;

f) **Cenários**: descrição seqüencial dos passos efetuados em uma instância do caso de uso. Normalmente possui um cenário principal e outros para relatar um fluxo alternativo ou excepcional de eventos;

Além das informações anteriormente citadas, no contexto de reengenharia outros detalhes são adicionados, como:

g) **Localização** – Módulo (*unit*, em Delphi) que inicia o caso de uso.

h) **Pontos de extensão** - Algum ponto de extensão ou exceção que estende o comportamento em um determinado momento no caso de uso. No diagrama de casos de uso é representado pelo estereótipo “*extend*”.

i) **Pontos de inclusão** - outros casos de usos usados no caso de uso principal. No diagrama de casos de uso é representado pelo estereótipo “*include*”. Possivelmente uma declaração “*uses*” no código legado sinaliza módulos onde algumas (ou todas) funcionalidades são mapeadas para casos de uso de inclusão.

j) **Objetos** – Objetos de dados (*Datasets*) e de projeto relevantes que participam do cenário.

k) **Interface de Usuário** (IU) – Imagem da janela, capturada no momento da execução do cenário principal.

Essas informações reunidas e mapeadas em estruturas no metamodelo, podem ser vistas na Figura 4.26.

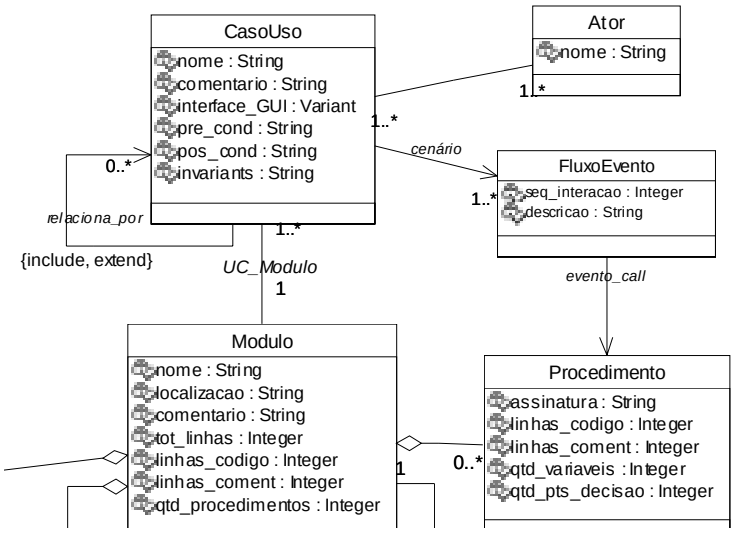



FIGURA 4.26 – Representação no metamodelo das estrututuras de UC/Re.

## 4.4 Terceira Etapa – Gerar Modelo Conceitual do Sistema (MCS)

Após a recuperação das principais abstrações e dos aspectos dinâmicos do sistema legado, a etapa seguinte é a elaboração do Modelo Conceitual do Sistema (MCS). Para formar a base da estratégia de construção do MCS, algumas diretrizes e heurísticas definidas em alguns métodos e trabalhos de pesquisa relacionados ao tema são usados, como em Gall et al.(1995), Penteado (1996), Fowley et al.(1999), Ducasse & Demeyer (1999) e outros.

A estratégia para a identificação de conceitos (classes com seus atributos e métodos) proposta neste trabalho é baseada na relação entre variáveis objetos de dados (DataSet's) e procedimentos (métodos candidatos) que participam dos UC/Re, previamente elaborados na etapa anterior. Para realizar esta estratégia, alguns passos são necessários:

1. definir o perfil da nova aplicação;
2. identificar as classes candidatas;
3. identificar os métodos;
4. construir o Modelo Conceitual do Sistema (MCS).

A Figura 4.27 mostra o diagrama de atividades da terceira etapa, acrescida de uma atividade auxiliar já referenciada na primeira e na segunda etapas: a reestruturação do código fonte dos procedimentos no sistema legado. Uma descrição detalhada de cada passo é feita, a seguir, com a definição de novas estruturas do metamodelo a ser mantido no repositório de migração.

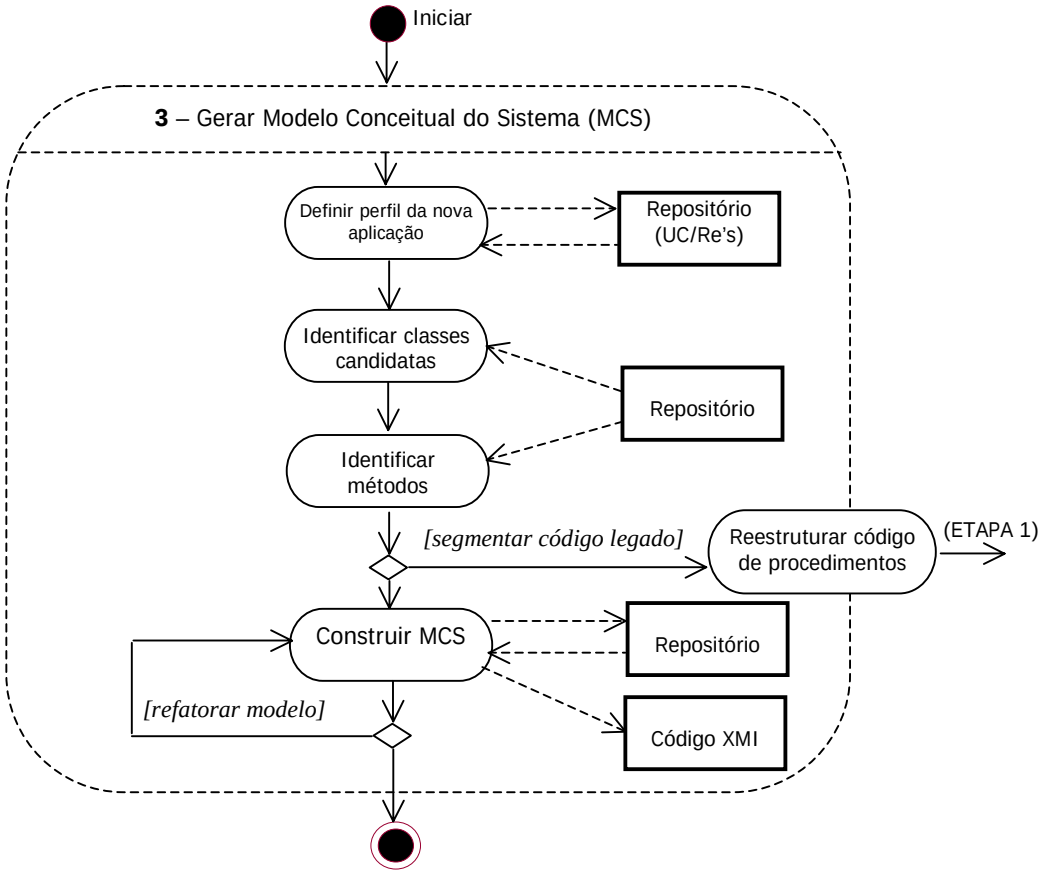



FIGURA 4.27 – Fluxo das atividades envolvidas na etapa “Gerar MCS”.

### 4.4.1 Definição do Perfil da Nova Aplicação

O objetivo é definir o escopo das funcionalidades do novo sistema. Uma parte delas foi recuperada em forma de UC/Re, e nas abstrações associadas como módulos, operações, menus e janelas que o usuário/ator interage. No final da etapa “Gerar MCS”, novas funcionalidades podem ser adicionadas à lista de casos de uso, evidenciando novos conceitos para o novo sistema.

O roteiro descrito a seguir, demonstra como definir o perfil.

**Definir o escopo do sistema a ser migrado**. Inicialmente é obtido dos UC/Re selecionados na etapa anterior. Outros requisitos e casos de uso podem ser definidos de forma parcial ao projeto de migração; na conclusão da etapa da construção do modelo conceitual, devem ser atualizadas as referências a novas classes de domínio.

**Definir graus de prioridade aos UC/Re**. Preferencialmente os UC/Re que participam do fluxo principal do sistema devem ser priorizados, juntamente com aqueles que mantêm com estes relações de dependência de tipo estendido e de inclusão. Também devem ser considerados os UC/Re que definem as restrições de pré-condição de um caso de uso prioritário.

### 4.4.2 Identificação das Classes Candidatas

Este passo considera as abstrações que representam os objetos Dataset, Tabela e Campo, no metamodelo, que foram capturados nas etapas anteriores. Um diagrama entidade-relacionamento (DER), originado a partir do banco de dados do sistema legado é uma ferramenta adicional que pode auxiliar na visualização dos objetos de dados.

Parte-se do princípio que o DER está consistente e que o banco de dados está normalizado.

A versão preliminar do MCS segue regras bem simples para identificação das supostas classes, seus atributos e relacionamentos, que são as seguintes:

**R1** - Todas as tabelas do DER, identificadas a partir dos objetos Dataset dos UC/Re, são convertidas em classes candidatas;

**R2** - Para cada tabela, cada campo se transformará em atributo da classe candidata, observando-se o mapeando de domínio de valores do atributo, entre os modelos. Todos os atributos terão visibilidade privada.

**R3** - As associações entre classes são derivadas dos relacionamentos entre as tabelas, isto é, ligações entre tabelas por meio de chave primária e por meio de chave estrangeira.

Outras regras de mapeamento do modelo relacional para um modelo OO podem ser encontradas na literatura. Trabalhos como os de Fong & Huang (1997) tratam do mapeamento de modelos de banco de dados relacionais para modelos orientados a objetos. Em Gall *et al.* (1995), a transformação de um diagrama entidade-relacionamento (DER) para um modelo estático de classes e objetos é realizada pelas similaridades entre as duas representações: objetos e seus atributos são diretamente derivados das entidades do DER, e as associações "generalizaçao-especialização" e "todo-parte", entre as classes, são mapeadas pelos relacionamentos *is-a* e *part-of* entre as entidades, respectivamente.

Tome-se como exemplo o UC/Re "Efetuar Compras". Recupera-se, a partir do repositório, os objetos Dataset dos módulos que participam do caso de uso (ver Tabela 4.6). Em seguida, elabora-se o DER das entidades (tabelas) envolvidas, como na Figura 4.28.

TABELA 4.6 – Objetos de dados do caso de uso "Efetuar Compras"

| Módulo Chamador | DataSet | Tabela | Manipulação |
|---|---|---|---|
| EntradaCompra.pas | dmBaseDados.tblNotaFiscal | NotaFiscal | M |
| | dmBaseDados.tblProdutos | Produtos | O |
| | dmBaseDados.tblEntrada | Entrada | M |
| | dmBaseDados.tblFornecedores | Fornecedores | O |
| ListaProdutosCod.pas | dmBaseDados.tblProdutos | Produtos | O |
| ProcessaNota.pas | dmBaseDados.tblProdutos | Produtos | M |
| | dmBaseDados.tblHistorico | Histórico | M |
| | dmBaseDados.tblNotaFiscalItens | NotaFiscalItens | M |
| | dmBaseDados.tblEntrada.First | Entrada | O |

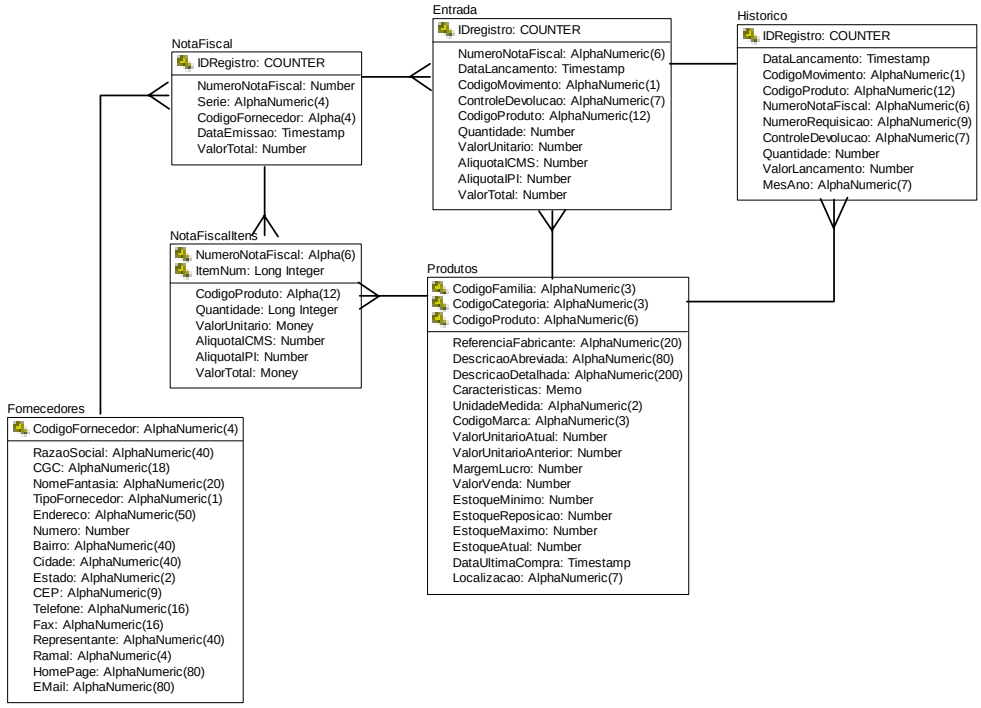



FIGURA 4.28 –Diagrama Entidade-Relacionamento das entidades do caso de uso “Efetuar Compras”.

Aplicando-se algumas heurísticas citadas anteriormente, um modelo de classes inicial teria a feição vista na Figura 4.29. A Tabela 4.6 mostra a transformação das tabelas em classes, notando-se as mudanças nos nomes das classes. No caso da tabela “Entrada”, muda-se para um nome de classe mais adequado ao contexto, pois se trata de uma classe de projeto, típica classe de controle de transação, transiente no processo, não fazendo parte do modelo de classes de negócio. Todos os campos das tabelas são mapeados em atributos, com os tipos convertidos em correspondentes domínios OO, considerando as regras de mapeamento de tipos apresentado no Anexo 3.

TABELA 4.6 – Mapeamento das tabelas em classes candidatas no MCS.

| **Tabela** | **Nome da Classe Candidata** | **Tipo de Classe** |
|---|---|---|
| Entrada | Transação_Entrada | Controle |
| Fornecedores | Fornecedor | Negócio |
| Histórico | Histórico | Negócio |
| NotaFiscal | NotaFiscal | Negócio |
| NotaFiscalItens | IItens_NotaFiscal | Negócio |
| Produtos | Produto | Negócio |

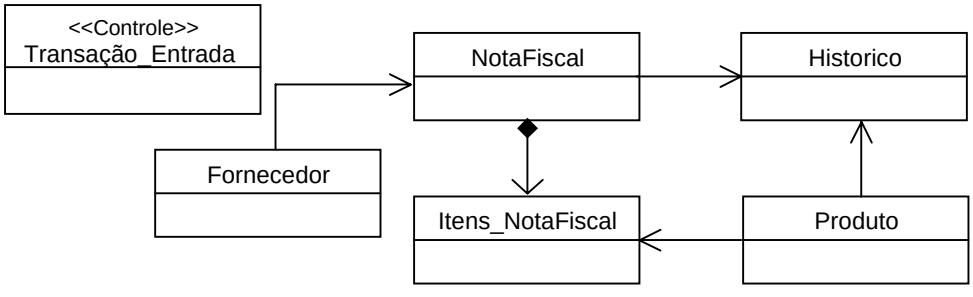



FIGURA 4.29 – Modelo de classes de negócio e de controle para o caso de uso “Efetuar Compras”.

### 4.4.3 Identificação dos Métodos

Os métodos advêm do objeto Método no metamodelo, conseqüência do processo de segmentação do código dos procedimentos. Se o procedimento não sofrer segmentação, apenas um método será definido que é o próprio procedimento. Cada atributo da classe terá os método "*set*" e "*get*", representando as operações de leitura e escrita ao atributo.

A Tabela 4.7 mostra procedimentos que participam do UC/Re "Efetuar Compras". Alguns são convertidos diretamente em métodos candidatos; outros desconsiderados por se tratar de procedimentos de implementação, e um deles (procedure btnSimClick...) é sinalizado com a observação de que deve sofrer um processo de reestruturação de código (segmentação), pois manipula várias pseudoclasses.

TABELA 4.7 – Métodos candidatos do caso de uso "Efetuar Compras".

| Procedimento | Objetivo | Manip. | PseudoClasse | Métodos Candid |
|---|---|---|---|---|
| **Módulo:** EntradaCompra | | | | |
| procedure FormShow | Exibe o form; abre as tabelas | Impl. | - | - |
| procedure FormClose | Fecha o form; fecha as tabelas | Impl. | - | - |
| procedure btnExcluirProdutoClick | Exclui um item de produto da NotaFiscal | M | Transação_Entrada | Exclui |
| procedure btnCancelarNotaClick | Cancela toda a transação | M | Transação_Entrada | Cancela |
| procedure btnGravarNotaClick | Chama o caso de uso "Processa Nota" | Impl. | - | - |
| procedure grdProdutosColExit | Controla a edição das colunas da grade de itens de produtos | Impl. | | |
| procedure grdProdutosColEnter | Controla a edição das colunas da grade de itens de produtos | Impl. | | |
| procedure fldValorTotalExit | Atualiza data de lançamento e código de movimento da NF | M | Transação_Entrada | setDataLancamento(Date)<br>setCodigoMovimento(string) |
| procedure grdProdutosEditButtonClick | Chama UC "Pesquisa produto" | Impl. | | |
| **Módulo:** ListaProdutosCod | | | | |
| procedure FormShow | Exibe o form. com lista de produtos em ordem alfabética de descrição. | Impl. | - | - |
| procedure btnOKClick | Retorna o código do produto selecionado. | O | Produto | getCodigoChave: string |
| **Módulo:** ProcessaNota | | | | |
| procedure FormShow | Exibe o form de confirmação de processamento da NF | Impl. | - | - |
| procedure btnSimClick | Lê a transação atualizando estoque de produtos, gravando NF e histórico de lançamento. | +M/O | Transacao_Entrada<br>Produto<br>NotaFiscal<br>ItensNotaFiscal<br>Historico | **OBS**: Segmentar código legado. |

### 4.4.4 Construção do Modelo Conceitual do Sistema

Após a versão preliminar do MCS, uma análise deve ser realizada buscando encontrar distorções de projeto OO. Diretrizes e heurísticas para qualidade de projeto OO são importantes para alcançar o refinamento desejado, algumas delas baseadas em práticas já amadurecidas de análise e projeto OO, como em Coad & Yourdon (1991), Penteado (1996), Pressman (2001) e outros. Refatoração é outra importante contribuição

para a melhoria do projeto OO, destacando-se trabalhos como o catálogo de padrões de refatoração de Fowley et al.(1999), e Ducasse & Demeyer (1999). A seguir, algumas dessas diretrizes são apresentadas.

**a) Remover atributos de ligação de relacionamentos**

Alguns atributos de classe representam chaves estrangeiras para associações entre as classes, que surgem dos relacionamentos entre as tabelas. Em um modelo OO estes atributos não são necessários quando o objetivo é fazer o relacionamento entre tabelas.

**b) Transformar classes em relacionamentos**

Algumas tabelas do modelo ER (entidade-relacionamento) são transformadas em relacionamentos no MCS. Como regra, tabelas (classes) que não possuam atributos (apenas atributos de ligação/chave), ou que possuam apenas um atributo fraco, podem ser transformadas em relacionamentos. Considera-se como atributo fraco àquele que não identifica unicamente o objeto da classe, podendo ser repetido em diversos objetos. Tais classes são tipicamente tabelas do modelo ER com duas chaves estrangeiras e nenhum ou apenas um atributo. Outra característica marcante dessas classes é que cada uma das chaves estrangeiras faz a ligação da classe com outra classe por intermédio de um relacionamento com cardinalidade “muitos para um”. No caso de haver um atributo na classe que está sendo eliminada, o relacionamento deverá carregá-lo.

**c) Identificar agregações e especializações de classe**

Agregações podem ser criadas sempre que uma classe englobar outras. Especializações (relações de herança) podem ser criadas para isolar em uma superclasse os atributos e métodos comuns a diversas subclasses.

**d) Converter para nomes mais significativos**

Os nomes dos atributos, métodos e classes do MCS devem ser modificados para nomes mais significativos, tornando os conceitos mais claros possíveis, de acordo com o domínio de aplicação.

**e) Refinar os métodos**

Redefinir e desmembrar os métodos quando o método realizar mais de uma funcionalidade na mesma classe. Mesmo que na fase de identificação de blocos nos procedimentos tenha-se a preocupação com este critério, nada impede que ainda existam métodos com baixa coesão.

**f) Refinar os métodos: eliminar/unir métodos duplicados**

É possível que vários métodos estejam duplicados em mais de um módulo e procedimento. Deve ser observado se os nomes e os objetivos são os mesmos, assim como parâmetros de entrada e valor de retorno do método. Se ocorrer de métodos terem a mesma funcionalidade e nomes diferentes, opta-se por um nome mais genérico que caracterize a função que o método executa.

**g) Desconsiderar classes de controle**

Classes de controle são típicas classes de projeto. Servem de suporte ao processamento e, normalmente, são transientes e redundantes, isto é, não armazenam dados e duplicam atributos de outras classes de negócio.

Assim como nas etapas anteriores, uma estrutura é definida para registrar as abstrações. O metamodelo visto na Figura 4.30, permite mapear as classes, atributos, métodos e os tipos de relações entre as classes do MCS. Este metamodelo é uma versão simplificada daquele definido em UML (UNIFIED MODELING LANGUAGE, 2003).

A preocupação com a rastreabilidade de retorno às abstrações que originaram os novos conceitos é garantida na estrutura do metamodelo com os relacionamentos de associação "*baseado em*".

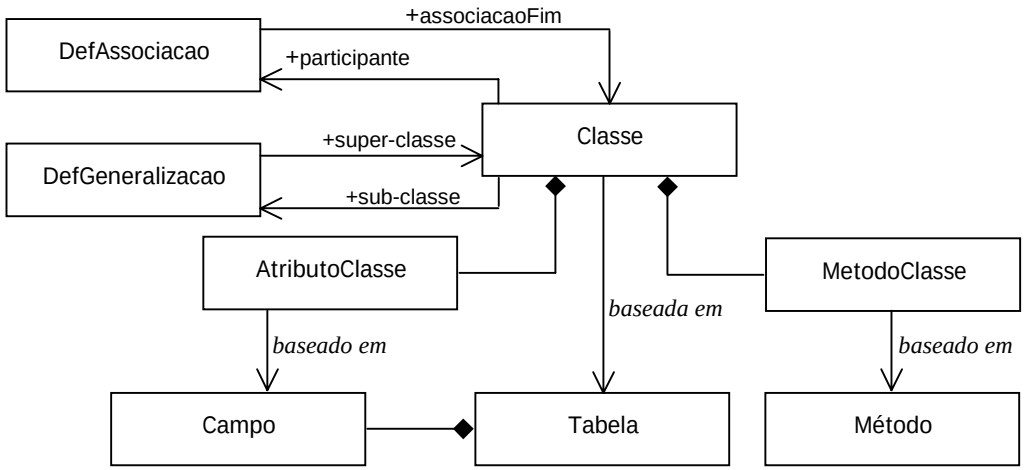



FIGURA 4.30 – Metamodelo para representar o MCS.

## 4.5 Quarta Etapa – Mapear Interfaces WIMP para Web

Esta etapa não pode ser negligenciada no processo, pois uma das características mais importante em aplicações Web é a questão da usabilidade. Usabilidade, no contexto da Web, é a medida pela qual se avalia o potencial de realizar uma tarefa na WWW, considerando fatores como facilidade de uso, consistência visual, e um processo claro e objetivo para cumprir uma tarefa.

Parece intuitivo acreditar que exista uma correspondência quase direta e equivalente entre o conceito de uma janela WIMP e uma página web. De fato, ao se comparar a tela inicial de um sistema aplicativo tradicional com a *homepage* de um *website* de serviços, pode-se afirmar que sim. Entretanto, a importância dada à aparência visual e à usabilidade de uma *homepage* é bem diferente àquela dada a uma tela inicial do sistema tradicional. A *homepage* é a página mais importante entre todas de um *website* (NIELSEN; TAHIR, 2001), fator que destaca as preocupações citadas anteriormente. Na opinião de Nielsen e Tahir (2001), sobre *homepages* mal projetadas, "a melhor maneira de espantar os usuários de um *site* é atrasá-los usando páginas de introdução inúteis e criando obstáculos para a navegação."

Nesta seção, são apresentadas as diretrizes para realizar o mapeamento das interfaces de usuário do sistema origem (do tipo WIMP) para o ambiente Web. A seguir, são enunciadas as decisões que definem as estratégias adotas, em função da inviabilidade de conversão total de componentes presentes em ambos os ambientes (MARTINS *et al.*, 2002c).

### 4.5.1 Estratégias de Mapeamento

Os objetos visuais, presentes em janelas gráficas, devem ser mapeados para as correspondentes formas apresentadas na plataforma Web. Se for decisão do projeto de migração adotar a arquitetura cliente-magro (CONALLEM, 2000), todos os recursos adotados seguirão aqueles definidos no padrão HTML. Dependendo da estratégia a ser

adotada, alguns controles visuais usados no ambiente WIMP podem ser mapeados diretamente ou não para HTML, como mostra a Figura 4.31.

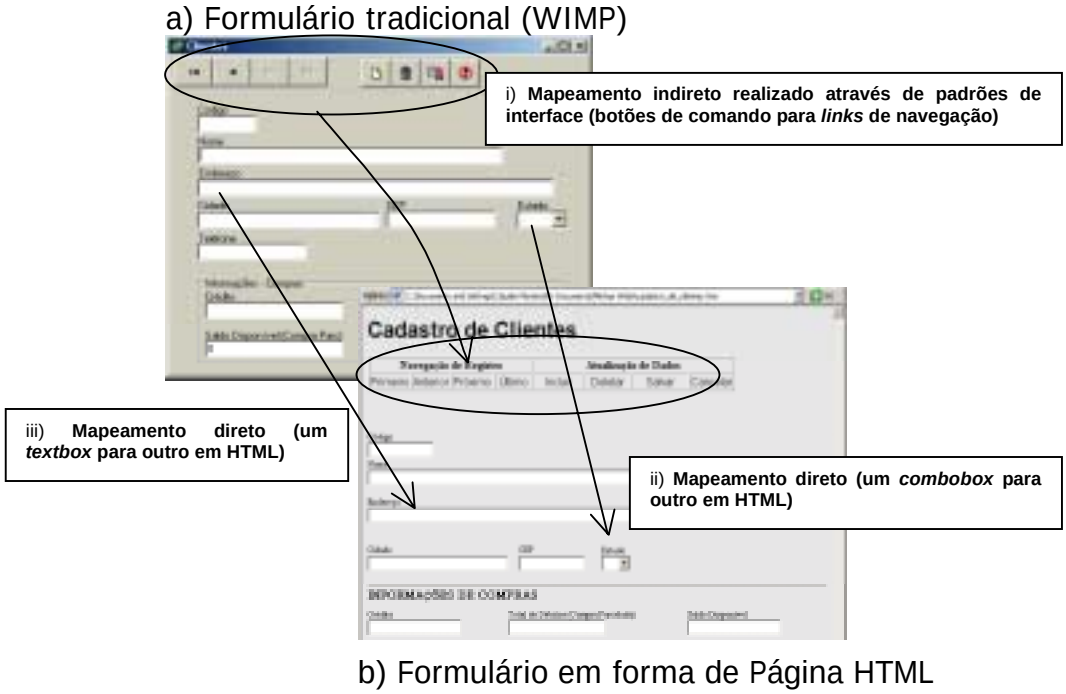



FIGURA 4.31 – Exemplo de mapeamento de interfaces de usuário entre os ambientes do tipo WIMP (a) e Web (b).

Percebe-se em alguns controles de interface tipo WIMP a ausência de objetos correspondentes ao modelo HTML. Neste caso, a adoção de padrões de projetos (Design Patterns) de interface (GARRIDO et al., 1997), (ROSSI et al., 1997) é uma abordagem que pode ser utilizada. Por exemplo, na Figura 4.31 (i), para os botões de comando com imagens que denotam operações sobre os dados, como salvar, excluir, atualizar e navegar entre os registros da tabela de Clientes, utilizou-se em HTML uma solução de elos (links) sobre textos mais explicativos (“Incluir”, “Deletar”, “Salvar”, etc) para implementar a mesma funcionalidade. Essa foi uma estratégia adotada levando-se em conta a usabilidade de interface (tornar mais claro ao usuário a função do objeto visual), apesar do controle botão de comando possuir um correspondente direto em HTML.

No âmbito da abordagem da engenharia reversa, os componentes WIMP, presentes no sistema origem, são aqueles extraídos da definição de contêineres, como os formulários e frames, recuperados como abstrações na etapa inicial do processo de reengenharia (ver Seção 4.2). Os componentes relevantes a serem armazenados no repositório são os controles visuais associados aos dados (campos) dos DataSets. Estas associações, ou vínculos de acoplamento de dados, são fundamentais para o mapeamento dos atributos das classes conceituais do sistema.

#### 4.5.1.1 Mapeamento Direto

É realizado quando os componentes em ambos os sistemas possuem correspondentes diretos. A tradução de código da definição dos controles visuais WIMP para formulários HTML pode ser de forma automática, como pode ser visto na Figura

4.32. No exemplo da mesma Figura 4.32, o código legado representando a definição de um formulário com controles WIMP (em linguagem Object-Pascal/Delphi) é traduzido diretamente em código HTML.

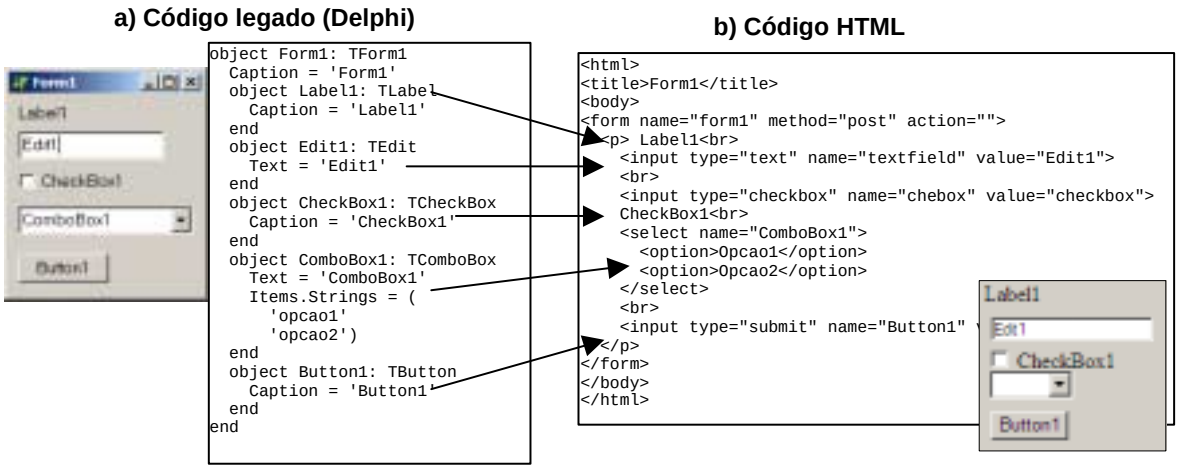



FIGURA 4.32 – Mapeamento direto

### 4.5.1.2 Mapeamento Indireto

Neste caso, percebe-se em alguns controles de interface tipo WIMP a ausência de objetos correspondentes no modelo HTML. Adotar "Padrões de Projetos" (*Design Patterns)* de interface (GARRIDO, ROSSI e SCHWABE 1997) (ROSSI, SCHWABE e GARRIDO. 1997) é uma abordagem que pode ser utilizada como forma de mapear os componentes visuais. Por exemplo, considere uma janela gráfica contendo botões de comando com imagens que denotam operações sobre os dados de um formulário, como salvar, excluir, atualizar e navegar entre diversos registros de um tabela de Clientes. No formato HTML utilizar-se a solução de elos (*links*) sobre textos explicativos (como "Incluir", "Deletar", "Salvar", etc) para implementar a mesma funcionalidade. Essa é uma estratégia adotada levando-se em conta a usabilidade de interface (tornar mais claro ao usuário a função do objeto visual), apesar do controle botão de comando possuir um correspondente direto em HTML.

Um exemplo mais evidente de mapeamento indireto pode ser demonstrado quando da presença de um componente Tab (vários frames sobrepostos, como "folhas de páginas" de um fichário) dentro de um container formulário. Desta forma, pode-se mapear para um padrão de projeto do tipo "Multi-Página", em que cada página HTML representa uma "folha" do fichário, e links posicionados acima da página fariam o papel das abas de cada folha. Este exemplo pode ser visto na Figura 4.33.

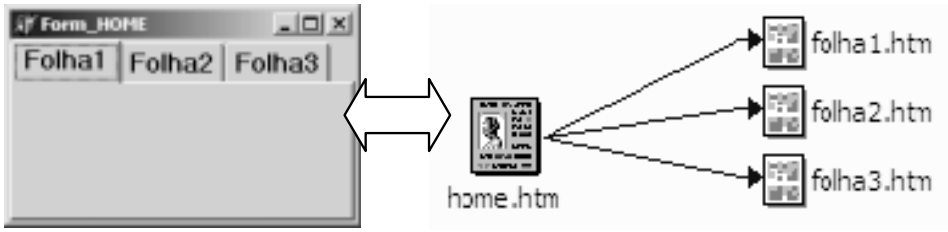



FIGURA 4.33 – Mapeamento indireto de um controle WIMP em HTML, por meio de um Padrão de Projeto de interface de usuário ("Multi-Página").

### 4.5.2 Atividades Envolvidas

O desdobramento da etapa de mapeamento de interfaces é feito em quatro passos:

1. Selecionar UC/Re e objetos de interface de usuário
2. Gerar uma página HTML
3. Selecionar um padrão de interface
4. Ajustar o código HTML

A Figura 4.34 mostra o fluxo de atividades desta etapa. Em seguida, as subseções discutem os passos em mais detalhes.

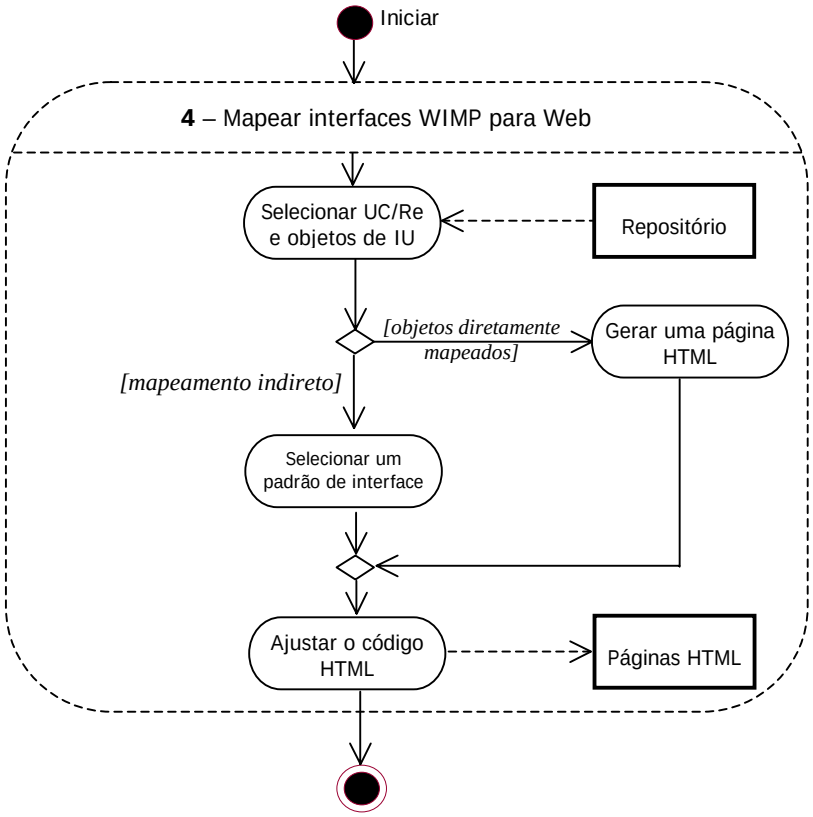



FIGURA 4.34 – Fluxo de atividades da etapa “Mapear interfaces WIMP para Web”.

#### 4.5.2.1 Selecionar UC/Re e objetos de interface de usuário

A partir do repositório, recupera-se os objetos visuais pertencentes ao UC/Re selecionado. Em seguida, deve-se identificar quais objetos podem ser mapeados diretamente ou não. Os objetos que não puderem ser diretamente traduzidos em código HTML devem ser separados para o passo seguinte. A Tabela 4.8 mostra o exemplo para o UC/Re “Efetuar Compras”.

TABELA 4.8 – Objetos visuais do UC/Re “Efetuar Compras”.

| **Formulário:** TformEntradaCompra | | | |
|---|---|---|---|
| **Objeto Visual** | **Tipo Widget Delphi** | **Classe do Componente** | **Mapeamento** |
| grdProdutos | TDBGrid | ApGrupo | Indireto |
| lblNotaFiscal | TLabel | ApTexto | Direto |
| lblSerie | TLabel | ApTexto | Direto |
| lblFornecedor | TLabel | ApTexto | Direto |
| lblDataEmissao | TLabel | ApTexto | Direto |
| lblValorTotal | TLabel | ApTexto | Direto |
| fldFornecedor | TDBLookupComboBox | ItSelecao | Direto |
| btnExcluirProduto | TSpeedButton | ItSelecao | Direto |
| btnCancelarNota | TSpeedButton | ItSelecao | Direto |
| btnGravarNota | TSpeedButton | ItSelecao | Direto |
| fldSerie | TDBEdit | ItTexto | Direto |
| fldDataEmissao | TDBEdit | ItTexto | Direto |
| fldValorTotal | TDBEdit | ItTexto | Direto |
| fldNotaFiscal | TDBEdit | ItTexto | Direto |

#### 4.5.2.2 Gerar uma página HTML

Após a identificação dos objetos visuais que podem ser mapeados diretamente em código HTML, seguindo regras de tradução, o processo deve gerar automaticamente a respectiva página HTML, correspondente ao formulário do UC/Re. A seguir, algumas dessas regras de tradução são mostradas na Tabela 4.9.

TABELA 4.9 – Exemplo de regras de tradução para objetos visuais em HTML.

| Objeto Visual | Propriedades | HTML correspondente |
|---|---|---|
| TForm | *Caption* | <TITLE> *valor do caption* </TITLE> |
| TEdit | *Text* | <INPUT> … |
| TMemo, TDBMemo | *Lines.Strings (*em *TMemo)* | *texto com os valores de <Lines.Strings>* |
| TImage, TDBImage1 | Tamanho (Width, Height), Picture.Data (em TImage) | <img src="*nomeImage.gif*" ….> |

Para posicionar cada objeto HTML na página, faz-se a correlação entre as posições originais do controle de diálogo de apresentação (saída) no formulário (propriedades *Left*, *Top* do objeto visual, medidos em *pixels*) e o tamanho do formulário (verificar as propriedades *Width*, *Height* do objeto TForm). Para os controles de diálogo de interação (entrada de dados), a propriedade *TabOrder* do controle ajudará na ordem em que estes controles são ordenados na página.

#### 4.5.2.3 Selecionar um padrão de interface

Para aqueles objetos cujo mapeamento é realizado de forma indireta, o projetista deve escolher um padrão de interface que satisfaça a equivalência de funcionalidades da interface original. Um catálogo de padrões de projeto de interface é indicado para ajudar na melhor opção. A Tabela 4.10 demonstra alguns exemplos de mapeamento indireto através de padrões de projeto de interface.

TABELA 4.10 – Exemplos de padrões para mapeamento indireto de objetos visuais.

| Objeto Visual | Padrão de Interface | Comentário |
|---|---|---|
| TpageControl | *Frames* e *hiperlinks* | Cada *Frame* corresponde a uma aba da página de controle. No caso de hiperlinks, cada *link* pode simular a chamada a uma página. |
| TmainMenu / TmenuItem | *Hyperlinks* | Cada opção do menu (TMenuItem) pode compor um elo (*link*) para outro nó (página). |
| TGroupBox | *Páginas individuais* | Separar cada grupo em páginas individuais. |
| TDBGrid | *Carrinho de compras* | Simula a inserção de dados em uma lista de itens, usando a metáfora do "carrinho de compras", em que cálculos e tratamento de transação em um conjunto de dados é requerida. |

#### 4.5.2.4 Ajustar o código HTML

Este passo é realizado de forma manual. O projetista deverá montar as duas partes de código HTML construídas nos passos anteriores. O resultado será a página HTML (ou páginas, dependendo dos padrões utilizados), correspondente em funcionalidade a interface do WIMP do sistema legado. Observações devem ser inseridas no projeto de interface, para indicar as relações entre objetos visuais com o Modelo Conceitual do Sistema (MCS).

## 4.6 Considerações Finais

O processo apresentado neste capítulo busca auxiliar o engenheiro de *software* nas tarefas de compreensão e construção dos modelos conceituais para um novo sistema a ser projetado.

A divisão do processo em quatro etapas, simplifica a adoção do método. Cada etapa produz artefatos e abstrações em um nível de compreensão aceitável para a reutilização no processo. Essas abstrações são mapeadas em metamodelos de representação, úteis na construção de modelos e geração de código por engenharia direta. A Tabela 4.11 resume as etapas do processo, descrevendo de forma objetiva as características, os artefatos produzidos e o grau de automatização de cada etapa em questão.

No capítulo seguinte, é apresentada a proposta do ambiente protótipo, detalhando algumas ferramentas necessárias ao suporte da metodologia de reengenharia discutida na dissertação.

TABELA 4.11 – Resumo do processo de migração.

| Etapa | Descrição | Artefatos produzidos | Grau de automatização |
|---|---|---|---|
| **Recuperar abstrações** | Extrair abstrações relevantes para o processo, a partir do código fonte do sistema legado. | Módulos, procedimentos, e objetos de dados e de interface de usuário. | **Alto**. Ferramentas de engenharia reversa como analisadores de código. |
| **Construir casos de uso para Reengenharia** | Obter o comportamento dos processos a serem migrados. | Cenários e casos de uso para reengenharia (UC/Re). | **Baixo**. Exige a participação de usuário especialista no domínio da aplicação. Uma análise do *log* de execução deve ser feita pelo engenheiro de projeto, que tomará decisões sobre necessidade de reestruturar código legado. |
| **Gerar Modelo Conceitual do Sistema (MCS)** | Produzir um modelo de classes de negócio para o novo sistema. | Modelo de classes. | **Médio**. Uma parte é obtida a partir do metamodelo, resultado das etapas anteriores. Outra, utiliza-se heurísticas OO e refatoração para depurar modelo de classes candidatas e na segmentação e descoberta de métodos das classes. |
| **Mapear interfaces WIMP para Web** | Definir estratégias para confecção das novas interfaces de usuário no novo ambiente (Web). | Formulários em HTML. | **Médio**. Alguns componentes (WIMP) podem ser mapeados diretamente em código HTML. No caso da impossibilidade do mapeamento direto, o uso de padrões de projeto e regras de ouro para interface Web é recomendado. |

# 5 Aplicação do Processo de Reengenharia Apoiado por um Ambiente CASE

Aplicar as etapas do processo em um projeto de reengenharia de grandes e médios sistemas de informação se torna, muitas vezes, humanamente difícil sem o suporte automatizado (ou semiautomatizado) de um ferramental apropriado. A primeira etapa, por exemplo, manipula grande quantidade de dados obtidos do código fonte do sistema legado. A rigor, é plenamente possível adotar técnicas e ferramentas de engenharia reversa para atingir os objetivos na etapa de recuperação de abstrações. Outras etapas e passos, em graus menores de automatização, também podem ser suportados por uma ferramenta.

Este capítulo descreve o protótipo de um ambiente de apoio ao processo de reengenharia apresentado nesta dissertação. São definidos os principais componentes e apresentadas algumas ferramentas de terceiros, que podem auxiliar na etapa de descoberta de abstrações e redocumentação do sistema legado. Para demonstrar o funcionamento do ambiente, um estudo de caso é apresentado, no qual o processo é aplicado em um caso de uso específico de um sistema de informação, desenvolvido em Delphi.

Na Seção 5.1 é apresentada a visão geral do ambiente, a arquitetura dos componentes principais e as correspondentes funcionalidades para cada etapa do processo. Na Seção 5.2 é descrito o funcionamento do protótipo, demonstrado através de um estudo de caso para uma aplicação implementada em Delphi. As considerações finais são apresentadas na Seção 5.3.

## 5.1 Visão Geral do Ambiente

A princípio, o ambiente busca seguir o mesmo fluxo de atividades da metodologia apresentada no capítulo anterior. Alguns passos são passíveis de automatização completa, outros não. Para contemplar o acompanhamento de todo o processo, o protótipo mantém um histórico das etapas e dos passos realizados e a realizar, de forma que o *reprojetista*[*] responsável fica ciente do que está acontecendo.

Além dos quatros componentes representando as etapas principais do processo, temos a presença de um componente de visualização de documentação, outro para chamada a ferramentas externas (desenvolvidas por terceiros) e a ferramenta de instrumentação do código legado. Outras camadas de componentes dão suporte ao ambiente, como o controle de tarefas, a interface de usuário e o componente de mapeamento do metamodelo em estruturas de armazenamento.

Na versão inicial, o protótipo foi implementado em Delphi, pela facilidade que este ambiente tem em disponibilizar componentes prontos para tratamento de interface de usuário e de acesso a banco de dados. Além disso, Delphi possui uma classe específica para análise de código fonte *Object Pascal*, no caso a classe *TParser*, e ferramentas e componentes *open source*, fornecidos por terceiros, que são adotados no ambiente para a recuperação de abstrações e redocumentação do sistema legado. Para representar o repositório, utilizou-se como banco de dados o Access, versão 2000, manipulado no ambiente através de componentes ADO.

[*] No contexto do ambiente proposto, considera-se o *reprojetista* como o principal ator do processo de reengenharia. Embora os termos "reprojetista" e "redesenvolvedor" não existam no vocabulário, no contexto desta dissertação terão o mesmo significado que o empregado para denominar *engenheiro de software* no papel de projetista de aplicações.

A seguir, é apresentada a arquitetura em componentes e as principais funcionalidades implementadas no protótipo do ambiente proposto.

### 5.1.1 Arquitetura

A arquitetura do ambiente proposto (ver Figura 5.1) é composta de componentes de interface de usuário (incluindo um *browser* HTML), um controlador de tarefas, um componente de mapeamento do metamodelo para o repositório e as ferramentas desenvolvidas por terceiros.

O "Controle de Tarefas" gerencia o ambiente no que diz respeito ao controle da interface de usuário e da interface com os componentes responsáveis por executar as tarefas disponíveis no ambiente. Além disso, faz a chamada a rotinas externas ao ambiente, isto é, controla a execução de programas e ferramentas desenvolvidos por terceiros. Entre as ferramentas de terceiros, duas geram a documentação do sistema legado, nas formas diagramática (diagramas de classes) e textual (páginas HTML).

Os componentes que dizem respeito às tarefas principais, isto é, "Recuperar Abstrações", "Construir UC/Re", "Construir MCS" e "Mapear IU para Web", contemplam, total ou parcialmente, as etapas já discutidas no processo de reengenharia proposto.

O componente "Visualizador" é a implementação de um *browser* simples, que visualiza a documentação HTML gerada no ambiente e pelas ferramentas externas ao ambiente.

O componente "Instrumentador de Código" prepara o código legado com sinalizadores que são utilizados para gerar o histórico de execução de cenários.

Finalmente, o componente "Metamodelo" representa as estruturas de (meta) classes para representar o metamodelo discutido na metodologia proposta. Realiza, também, o mapeamento dessas estruturas em dados persistentes no banco de dados relacional (Access).

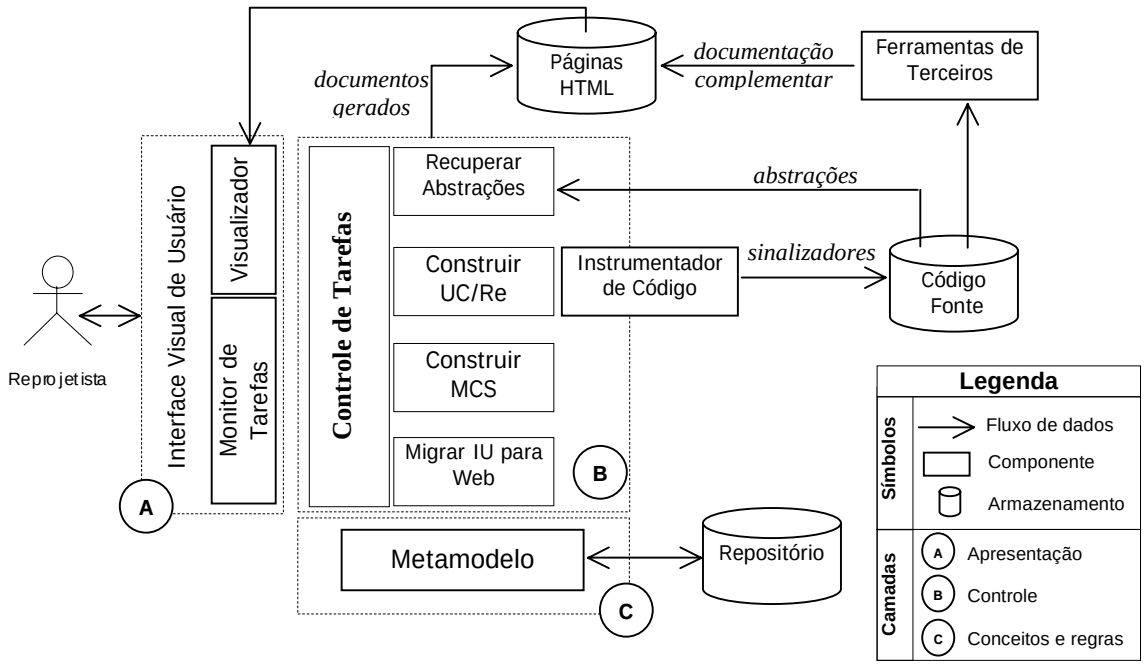



FIGURA 5.1 – Arquitetura do ambiente para realizar o processo de reengenharia.

A Tabela 5.1 apresenta um resumo de cada componente, destacando as principais funcionalidades, os níveis de automatização da tarefa e os produtos gerados na realização de cada tarefa.

TABELA 5.1 – Componentes disponíveis no ambiente proposto.

| Componente | Funcionalidades | Automatização | Produtos gerados |
|---|---|---|---|
| **Recuperar Abstrações** | Analisador sintático de código. | **Total** | Abstrações do código legado, que são armazenados no repositório. |
| **Registrar UC/Re** | Registra informações dos UC/Re.<br>Permite recuperar a documentação dos casos de uso, referenciando as abstrações que participam do cenário de execução. | **Parcial**. Parte da construção dos cenários é obtida através dos arquivos de rastreio (*log* de execução). | Dados dos UC/Re, incluindo cenário principal e imagem da principal interface de usuário.<br>Documentação com as referências cruzadas das abstrações. |
| **Construir MCS** | Gera as classes para o MCS a serem exportadas às ferramentas de modelagem UML. | **Parcial**. As classes candidatas são recuperadas automaticamente das abstrações do tipo *DataSet*.<br>A depuração das classes é feita manualmente. | Arquivo no formato XMI, com as definições das classes candidatas. |
| **Migrar IU para Web** | Gera código HTML dos objetos visuais dos UC/Re selecionados. | **Parcial**. Somente objetos mapeados de forma direta são traduzidos automaticamente em HTML. | Código parcial HTML das interfaces de usuário. |
| **Ferramentas de Terceiros** | Documentam o código legado. | **Total**. | Documentos no formato HTML com a documentação do sistema legado. |
| **Instrumentador de código** | Instrumentar código legado com mensagens (sinalizadores), que representam os eventos disparados na execução de cenários dos UC/Re. | **Total**. | Arquivo de rastreio (*trace*), com o histórico dos eventos disparados na execução dos cenários dos UC/Re do sistema legado.<br>Imagem da principal interface de usuário do UC/Re. |
| **Visualizador** | Visualiza documentos em formato HTML gerados pelo ambiente e por ferramentas externas. | **Total**. | Documentos visualizados. |

A seguir, é apresentado o funcionamento da implementação do protótipo para o ambiente proposto, utilizando-se um estudo de caso como demonstração.

## 5.2 Funcionamento do Ambiente, Demonstrado por um Estudo de Caso

A fim de avaliar as funcionalidades do ambiente proposto, um estudo de caso é apresentado nesta seção. A aplicação legada que serve de estudo de caso é típica para sistemas discutidos no Capítulo 1, isto é, regras de negócios misturadas com código de interface de usuário. A aplicação chama-se SGE, "Sistema de Gerência de Estoque", e foi desenvolvida para controlar o estoque de produtos eletrônicos de um comércio dessa área. Foi implementada em Delphi, usando tabelas armazenadas em Paradox, que são manipuladas no código por componentes de dados da classe *TTable* (sem uso de código SQL).

O primeiro passo que o reprojetista deve tomar é preparar uma cópia da aplicação legada para uma área específica para o processo de migração. Essa cópia inclui todos os programas (código fonte) e dados do sistema (tabelas do banco de dados).

A partir deste momento, a ferramenta inicia com um novo projeto de migração. Como apresentado na Figura 5.2, o reprojetista informa o nome do projeto de migração e o local onde se encontra a cópia do sistema legado. A ferramenta recupera o nome de todos os módulos definidos no sistema SGE.

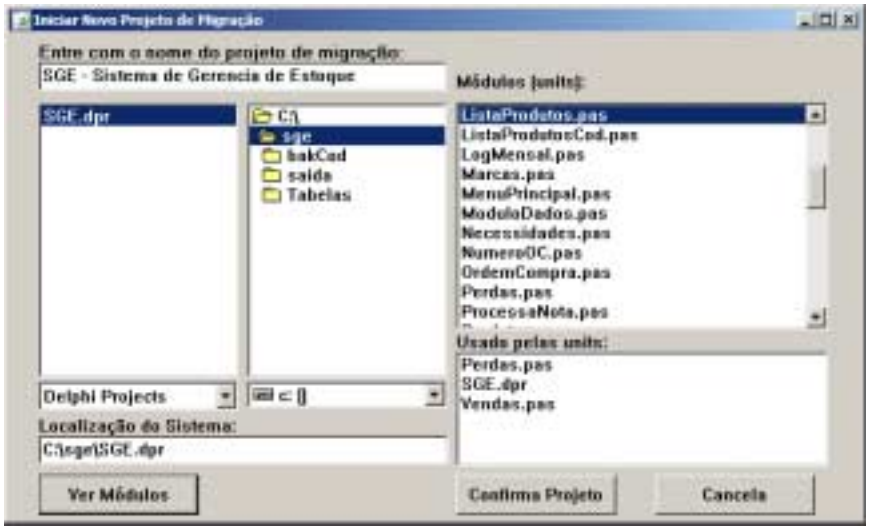



FIGURA 5.2 – Iniciando um novo projeto de migração.

Em seguida, a janela do monitor de tarefas (ver Figura 5.3) é apresentada, disponibilizando opções para as quatro atividades principais do processo e chamadas às ferramentas de apoio para documentação do sistema legado.

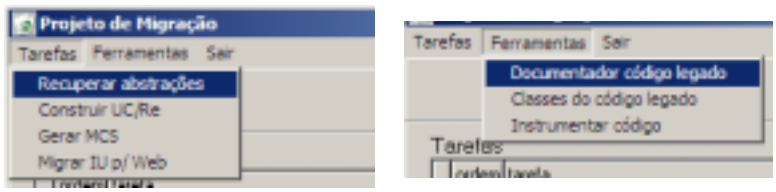



FIGURA 5.3 – Visões com as opções principais do Projeto de Migração.

O roteiro descrito a seguir, demonstra a aplicação das quatro tarefas principais do processo de reengenharia, facilitando a compreensão do funcionamento do ambiente proposto.

### 5.2.1 Tarefa 1: Recuperar Abstrações

Deve ser ativada na primeira iteração do processo, bem como nas vezes em que o código fonte (do sistema legado) sofrer reestruturação através de segmentação, por exemplo. Este componente implementa um analisador sintático de código para a linguagem *Object Pascal*. Como foi descrita na Seção 4.2, sobre a primeira etapa do processo, esta tarefa tem como objetivo recuperar todos os elementos necessários para a realização das etapas seguintes: abstrações em forma de módulos, assinaturas de procedimentos, objetos de dados e de interface de usuário, e referencias cruzadas entre estas abstrações. A Figura 5.4 mostra a janela com o *feedback* de mensagens, exibindo os módulos e procedimentos recuperados do sistema SGE. Ao encerrar esta atividade, o controle de tarefas atualiza o monitor, como pode ser visto na Figura 5.5 (todas as atividades são registradas no monitor de tarefas). Todas as abstrações identificadas e recuperadas são armazenadas no repositório. A atividade seguinte é a construção dos UC/Re.

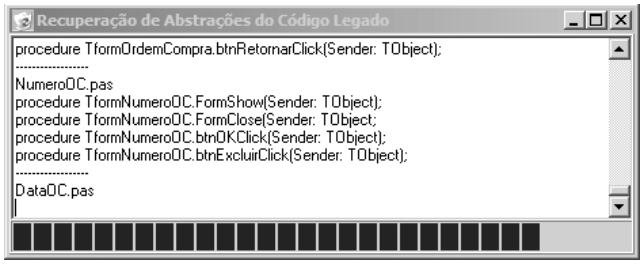
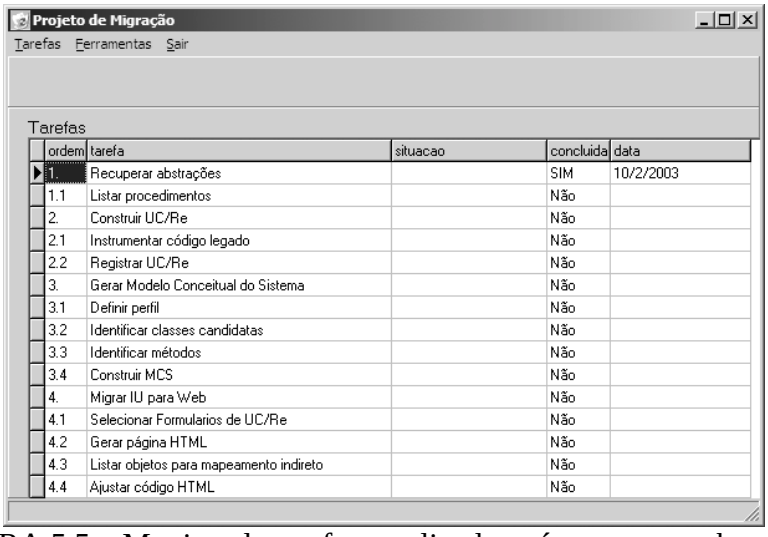



FIGURA 5.4 – Processamento da tarefa “Recuperar Abstrações**”.**




| ordem | tarefa | situacao | concluida | data |
|---|---|---|---|---|
| 1. | Recuperar abstrações | | SIM | 10/2/2003 |
| 1.1 | Listar procedimentos | | Não | |
| 2. | Construir UC/Re | | Não | |
| 2.1 | Instrumentar código legado | | Não | |
| 2.2 | Registrar UC/Re | | Não | |
| 3. | Gerar Modelo Conceitual do Sistema | | Não | |
| 3.1 | Definir perfil | | Não | |
| 3.2 | Identificar classes candidatas | | Não | |
| 3.3 | Identificar métodos | | Não | |
| 3.4 | Construir MCS | | Não | |
| 4. | Migrar IU para Web | | Não | |
| 4.1 | Selecionar Formularios de UC/Re | | Não | |
| 4.2 | Gerar página HTML | | Não | |
| 4.3 | Listar objetos para mapeamento indireto | | Não | |
| 4.4 | Ajustar código HTML | | Não | |

FIGURA 5.5 – Monitor de tarefas atualizado após recuperar abstrações.

Nessa versão inicial, implementou-se um analisador de código para objetos *dataset* simples, que manipulam diretamente as tabelas, através de componentes da classe *TTable*, sem intermediação de código SQL. Essa limitação não comprometeu o resultado, uma vez que o estudo de caso trata de um sistema implementado nesse modelo de acesso a dados.

### 5.2.2 Tarefa 2: Construir UC/Re

Antes de iniciar esta tarefa, a ferramenta de instrumentação do código legado (disponível na opção Ferramentas, no menu principal) deve ser invocada na primeira vez que a tarefa for executada*. A finalidade é preparar o código legado para o processo de rastreio de execução, útil na elaboração dos cenários dos casos de uso.

Em seguida à instrumentação, o sistema legado (SGE, no estudo de caso), já reconstruído (gerada a versão executável) com as alterações no código, deve ser executado nas funcionalidades alvo do processo de reengenharia. Na execução do sistema SGE, uma janela (Figura 5.6), representando o monitor de rastreio, fica no fundo da janela principal do sistema recuperando e registrando os eventos que ocorrem na interação do usuário com o sistema legado. A Figura 5.6 mostra um rastreio recuperado para a execução do UC/Re "Efetuar Compras". Percebe-se que é possível o reprojetista/usuário do sistema registrar eventos no histórico, para facilitar o entendimento e leitura dos eventos ocorridos, bastando clicar no botão "Registrar Ev". A ferramenta é capaz, também, de capturar a imagem da tela do sistema (botão "Captura Tela") e gravar o histórico em um arquivo de "*log*" (botão "Gravar Log").

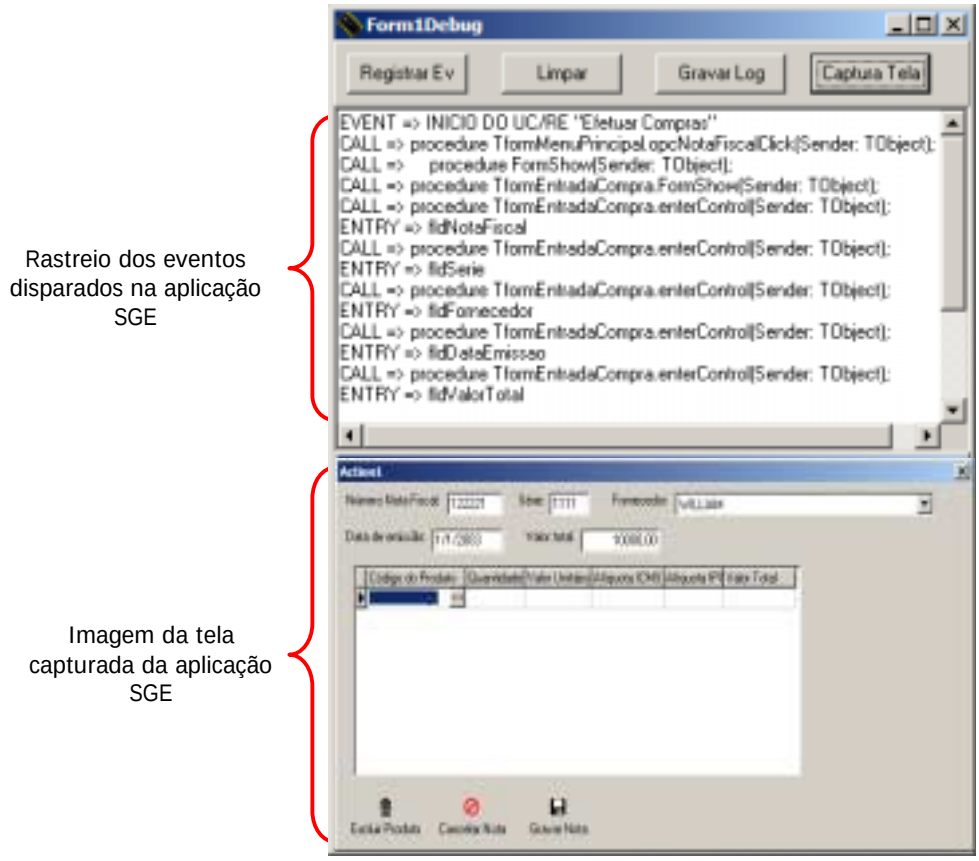



FIGURA 5.6 – Janela com o monitor de rastreio de execução do sistema legado.

A ferramenta de rastreio identifica três tipos de eventos indicados no histórico pelos símbolos: CALL, EVENT e ENTRY. CALL é a chamada a um procedimento no sistema legado; a assinatura do procedimento é recuperada automaticamente. EVENT é um sinalizador inserido pelo usuário/reprojetista para indicar um evento não recuperado automaticamente pela ferramenta; o objetivo é adicionar comentários para melhor

* Assim como na tarefa "Recuperar Abstrações", a instrumentação deve ser ativada toda vez que houver mudança no código fonte do sistema legado. O monitor de tarefas indica quando este procedimento é necessário.

compreensão do histórico. ENTRY é um sinalizador inserido automaticamente, representando a entrada de dados em um objeto visual de interface de usuário; o nome do objeto é obtido automaticamente pela ferramenta.

Após completar a execução de todos os prováveis cenários que irão compor o projeto de migração, o reprojetista deve registrá-los no ambiente da ferramenta. A Figura 5.7 exibe a janela para atualizar os dados de todos os UC/Re do sistema legado. Ao escolher um determinado UC/Re na lista, pode-se visualizar um relatório gerado pelo protótipo, com informações daquele caso de uso, bastando clicar no botão "Documentador" que se apresenta na barra de botões, acima da lista de UC/Re. Outros três botões ("Insere", "Altera" e "Deleta UC/Re") têm por objetivo chamar a janela de atualização de um UC/Re selecionado da lista, operação que será detalhada a seguir.

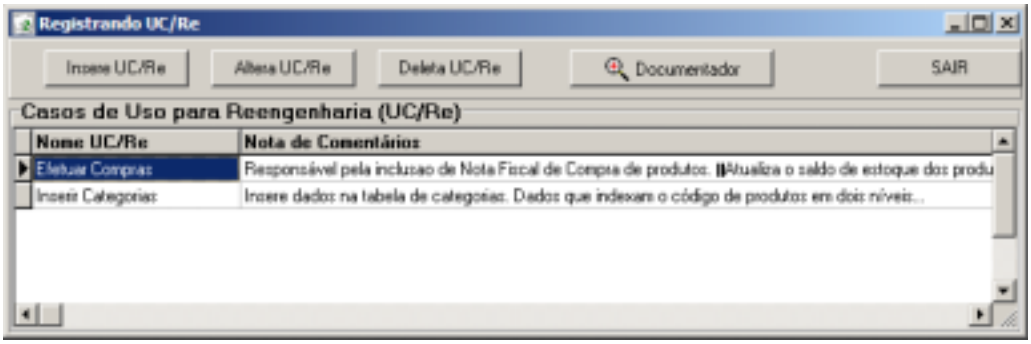



FIGURA 5.7 – Janela com todos os UC/Re e para registro de um novo (botão "Insere...").

Ao atualizar um UC/Re selecionado da lista, uma janela é apresentada com quatro abas, agrupando informações diversas sobre o caso de uso, que são: "UC/Re (Geral)", "Restrições", "Interface de Usuário" e "Cenário Principal". Estes agrupamentos podem ser vistos nas Figura 5.8 a 5.11 e detalhados a seguir.


FIGURA 5.8 – Grupo com informações gerais do UC/Re "Efetuar Compras".

No grupo de informações gerais do UC/Re (Figura 5.8), o reprojetista transcreve os seguintes dados: nome do caso de uso, único para cada UC/Re; notas de comentários gerais; lista dos atores principal e secundário; define os relacionamentos com outros UC/Re, na forma de inclusão ou extensão; e indica qual módulo inicia o UC/Re.

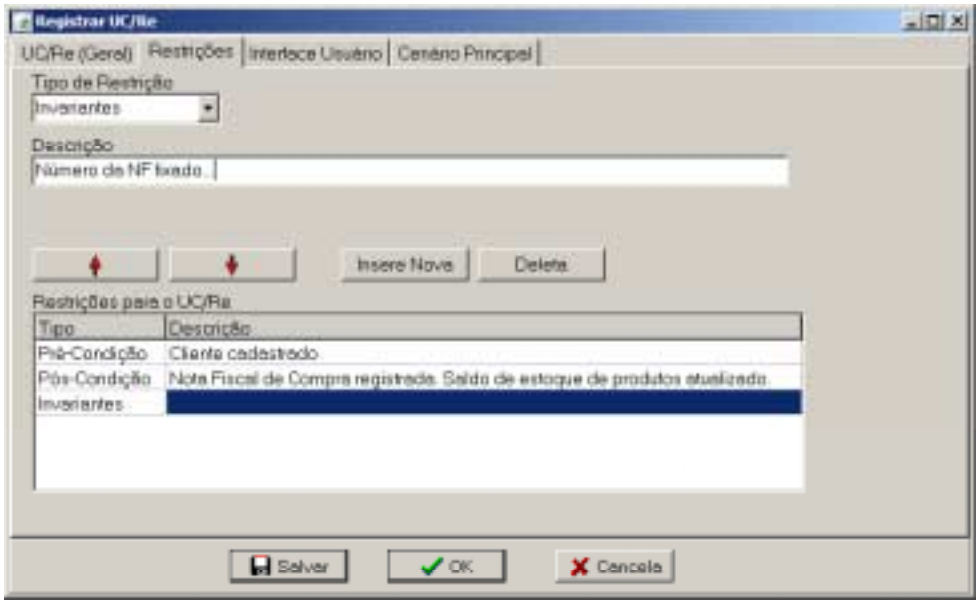



FIGURA 5.9 – Grupo com informações sobre restrições do UC/Re.

No agrupamento “Restrições” (Figura 5.9), são definidas as regras preliminares à realização do UC/Re (as *pré-condições*), as resultantes da realização (as *pós-condições*) e aquelas que são fixadas como constantes ao funcionamento do caso de uso (as *invariantes*).


FIGURA 5.10 – Grupo com a imagem da janela capturada para o UC/Re “Efetuar Compras”.

A imagem da tela principal do UC/RE, vista na área de visualização da Figura 5.10, é fruto do recurso de rastreio de execução da aplicação SGE. Para inserir a imagem, o reprojetista deve informar o arquivo que a armazena, clicando no botão “Carrega Imagem”.

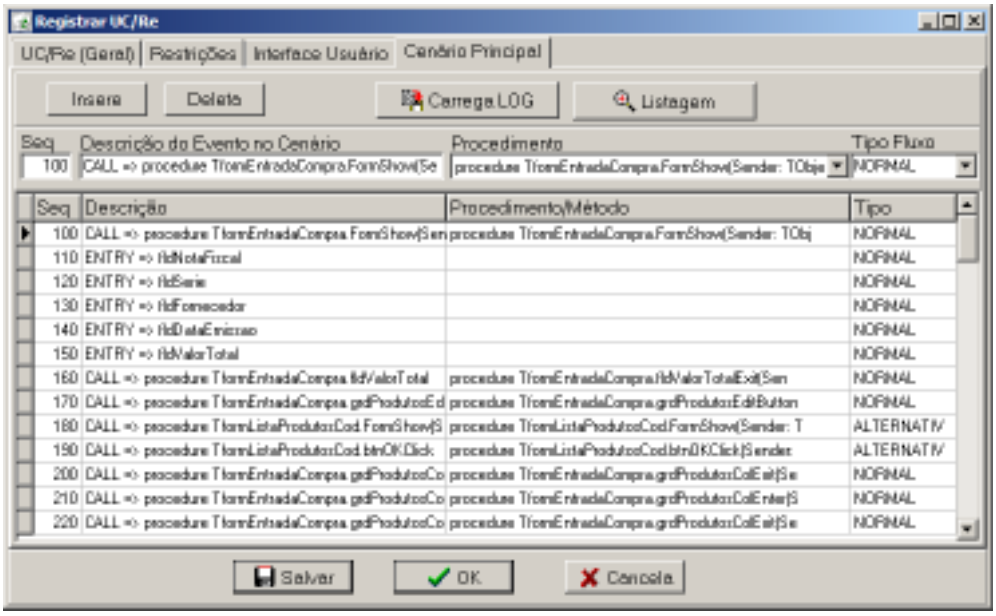



FIGURA 5.11 – Grupo com informações do cenário principal do UC/Re “Efetuar Compras”.

A Figura 5.11 mostra a possibilidade de construção automática do cenário principal, a partir das informações coletadas no rastreio de execução, feita anteriormente. Do arquivo de *log* gerado no rastreio de execução, o protótipo lança a descrição dos eventos, extrai o nome do procedimento chamado (identificado na marca CALL) e define que tipo de fluxo está em curso, normal ou alternativo, decisão que é tomada em função de qual procedimento foi chamado. A *priori*, um procedimento que não pertença ao módulo do fluxo *normal* do cenário principal é definido como de *fluxo alternativo*; no estudo de caso, o fluxo dos procedimentos do módulo “ListaProdutos”, indicados na Figura 5.11 pelas seqüências números 180 e 190, é considerado do tipo *alternativo*, no cenário estabelecido para o caso de uso “Efetuar Compras”.

A Figura 5.12 mostra a visão da documentação completa do UC/Re “Efetuar Compras”, com os módulos participantes obtidos a partir do cenário de execução (capturados a partir do arquivo de “Log”). Nota-se as referências a abstrações do tipo módulo, procedimentos, tabelas, métricas (pontos de decisão e linhas de código) e outras informações que são obtidas a partir do repositório.

Além da documentação gerada pelo protótipo, o ambiente propõe o uso de duas ferramentas externas para prover documentação auxiliar. As ferramentas realizam engenharia reversa do código legado para gerar documentação em um nível de detalhe bastante aceitável para as necessidades do projeto de migração. Na seção que segue, são apresentadas essas ferramentas.

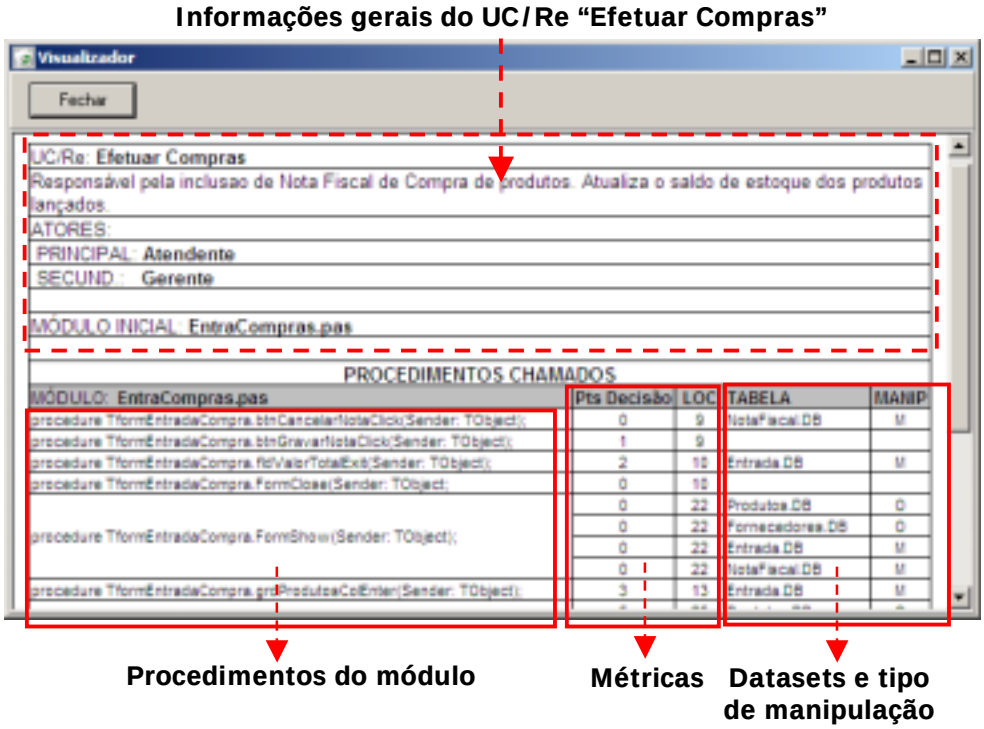



FIGURA 5.12 – Visão da documentação do UC/Re com os módulos participantes.

### 5.2.2.1 Ferramentas de terceiros para documentação auxiliar do sistema legado.

As ferramentas acopladas ao ambiente são da categoria "*open source*", que obedecem à política de licenças livres. São úteis para serem empregadas na fase de compreensão do sistema legado, pois geram documentação automática a partir do código fonte Delphi.

As ferramentas DIPasDoc e ESS-Model podem ser empregadas na etapa de compreensão da aplicação legada, onde detalhes, como referencia cruzada de elementos que não estão representados no repositório, for necessário. Um exemplo de uso dessas ferramentas é na captura de informações sobre identificadores (usando DIPasDoc) ou da visão (usando ESS-Model) da hierarquia de classes (se existirem), que foram declarados em um módulo específico.

A seguir são apresentadas as ferramentas DIPasDoc e ESS-Model, com figuras mostrando exemplos para o módulo "EntraCompras.pas".

**DIPasDoc**

DIPasDoc (JUNKER, 2003) gera documentação HTML a partir de comentários presentes no código fonte Pascal (Delphi), no mesmo estilo utilizado no documentador nativo do Java, o JavaDoc. É uma ferramenta licenciada para uso livre, observando-se os direitos previstos em *softwares GNU General Public License* (GPL).

A Figura 5.13 mostra as abstrações obtidas do código fonte. A ferramenta extrai elementos como units, hierarquia de classes definidas no código, todas as classes e interfaces de objetos, tipos de dados definidos pelo programador, declaração de variáveis, constantes, funções, procedimentos e identificadores, todos com referencia cruzadas. As informações das definições dos elementos são obtidas de comentários inseridos antes de cada declaração de um elemento, respeitando a declaração de comentários inseridos entre os símbolos "{" e "}".


FIGURA 5.13 – Visão parcial da documentação do módulo “EntraCompras”, gerada pela ferramenta DIPasDoc.

**ESS-Model**

Com ESS-Model (ESS-MODEL, 2003), pode-se recuperar a visão de diagrama de classes e de pacotes do código fonte em Delphi. Como mostra a Figura 5.14, é possível obter o diagrama dos pacotes (módulos/*units*) do módulo “EntraCompra.pas” do sistema SGE; para cada módulo selecionado na árvore de pacotes, é visualizado as classes pertencentes a este módulo.

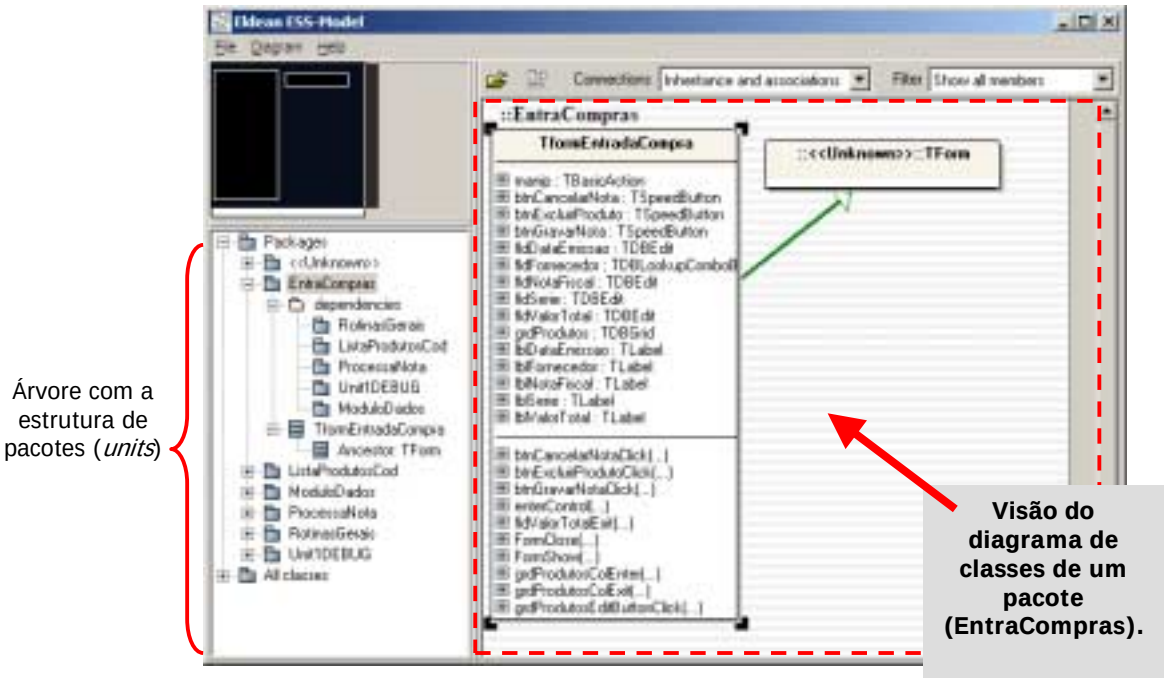



FIGURA 5.14 – Visão parcial dos diagramas de pacotes e classes, obtidos pela ferramenta ESS-Model.

### 5.2.3 Tarefa 3: Gerar MCS

De posse dos UC/Re, o reprojetista seleciona aqueles que participarão efetivamente do projeto de migração, definindo níveis de prioridade para cada um, conforme pode ser visto na Figura 5.15.

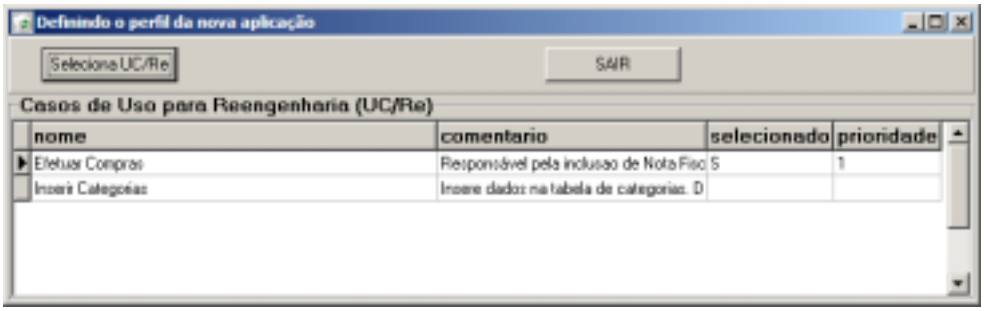



FIGURA 5.15 – Seleção de UC/Re para o projeto de migração.

Definidos os UC/Re, o passo seguinte é a identificação de possíveis classes candidatas ao Modelo Conceitual do Sistema, baseados nos objetos de dados referenciados nos UC/Re selecionados. A Figura 5.16 exibe a janela de classes candidatas, com seus possíveis atributos e métodos. Todos os dados disponíveis são obtidos automaticamente do repositório. Ao reprojetista cabe o papel de depurar os atributos, métodos e classes, seguindo heurísticas citadas na Seção 4.4. Na área identificada na Figura 5.16 para as classes candidatas, o reprojetista deve informar os nomes para as futuras classes do modelo. Na área de atributos é definido automaticamente o tipo do dado e tamanho do atributo a partir das referencias ao campo base e ao tipo de dado do campo, respectivamente; o reprojetista modifica ou não o nome do atributo, que recebe, a princípio, o mesmo nome do campo. Os métodos inicialmente são identificados pelas referências aos procedimentos que manipulam as tabelas e devem ser nomeados pelo projetista. Outros métodos devem ser identificados pela presença de anomalias presentes no código legado do procedimento. Duas decisões podem ser tomadas. A primeira é a reescrita do código legado usando técnicas de segmentação, tomando-se o cuidado de manter, ao final do processo de reestruturação, a mesma funcionalidade do sistema original. Em seguida, retoma-se a tarefa de recuperar abstrações e construção dos cenários principais dos UC/Re selecionados. A segunda opção é identificar os blocos de métodos, mas sem realizar a reestruturação no código legado. Independente da decisão assumida, ambas delegam à fase de construção final do MCS a localização de métodos duplicados, dos homônimos e outros tipos de depurações que devem ser tratadas com práticas de projeto OO, como a refatoração, por exemplo.

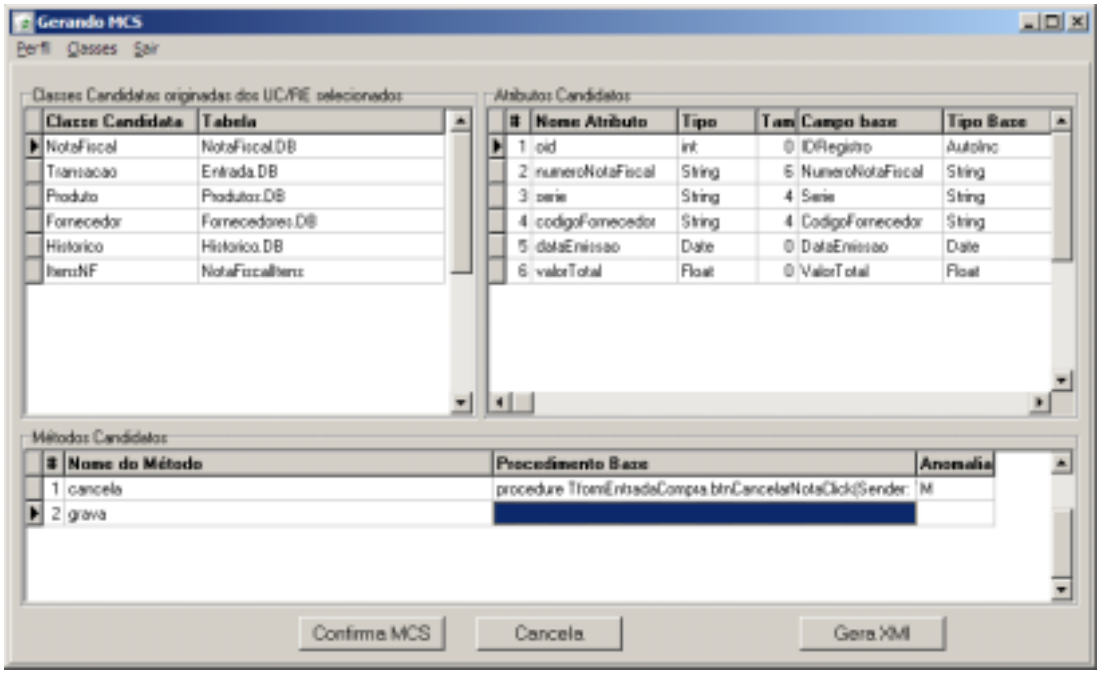



FIGURA 5.16 – Classes identificadas a partir dos UC/Re selecionados.

No exemplo mostrado na Figura 5.16, seis classes são identificadas e renomeadas para NotaFiscal, Transação, Produto, Fornecedor, Histórico, ItensNF. O protótipo assume que a visibilidade dos atributos é privada; para os métodos é pública. Após a depuração aplicada às classes, o reprojetista pode gerar a definição das classes para o formato XMI, para posterior importação por uma ferramenta CASE de modelagem UML, que siga este padrão de intercâmbio. No estudo de caso, utilizou-se a ferramenta de modelagem ArgoUML (ARGOUML, 2003), disponível para uso livre, seguindo a filosofia "*open source*". A Figura 5.17 mostra a ferramenta ArgoUML com as classes importadas a partir do arquivo XMI (ver trecho do código na Figura 5.18), gerado pela ambiente proposto.

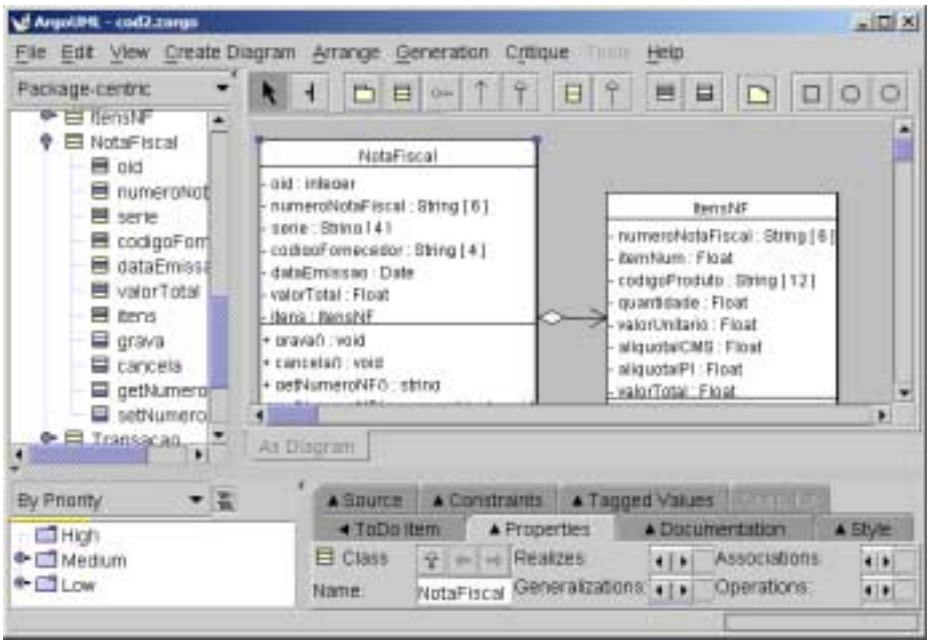



FIGURA 5.17 – Apresentação da interface da ferramenta ArgoUML, com as classes importadas, via formato XMI.

Para a geração do arquivo XMI, optou-se por uma solução intermediária. O protótipo gera as definições das classes em *Object Pascal*, em seguida submete as definições à ferramenta ESS-Model, que fornece um mecanismo de geração de código XMI a partir de código escrito em Delphi (Object Pascal) ou Java. O protótipo chama a ferramenta ArgoUML, e o reprojetista se encarrega de abrir o arquivo XMI gerado anteriormente. ArgoUML converte automaticamente as definições em XMI para um modelo de classes UML.

Do modelo importado pela ferramenta ArgoUML, o reprojetista poderá refinar e aplicar técnicas de refatoração (também discutidas na Seção 4.4), para melhorar o modelo de classes conceituais do sistema a ser migrado. Dependendo da ferramenta de modelagem usada, pode-se também gerar código, por engenharia de produção (ou avante), em uma linguagem alvo definida para o modelo de implementação do projeto de migração. No caso de ArgoUML, é possível gerar este código em Java, por exemplo.

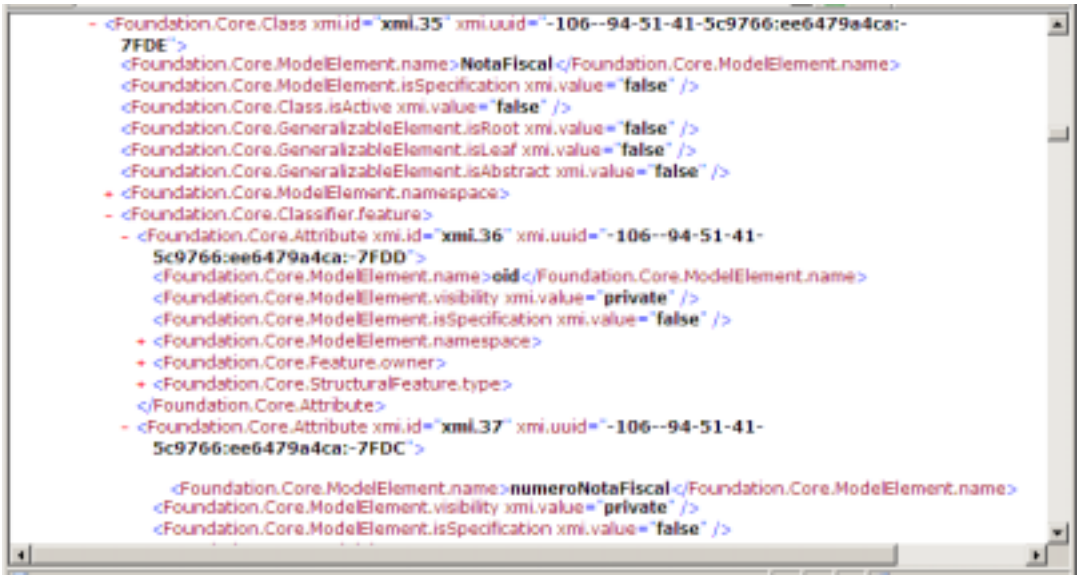



FIGURA 5.18 – Trecho do código XMI, gerado pelo ambiente.

### 5.2.4 Tarefa 4: Mapear IU para a Web

A última tarefa a ser executada é a geração do código HTML dos objetos visuais dos UC/Re selecionados. O ambiente se propõe a realizar a tradução automática dos elementos visuais que podem ser mapeados diretamente em código HTML. Os elementos que não satisfaçam essa condição são identificados e terão de ser confeccionados por um *Web Design*, em ferramenta adequada para edição de páginas Web. Para o exemplo apresentado no estudo de caso, o UC/Re "Efetuar Compras" apresenta um elemento que é mapeado de forma indireta, representando uma grade de itens de produtos, representado pelo objeto "grdProdutos", conforme visto na Figura 5.19. A solução apresentada na Figura 5.20, exibe parte do código HTML gerado automaticamente e outra adicionada ao código, seguindo padrões de projeto discutidos na Seção 4.5.

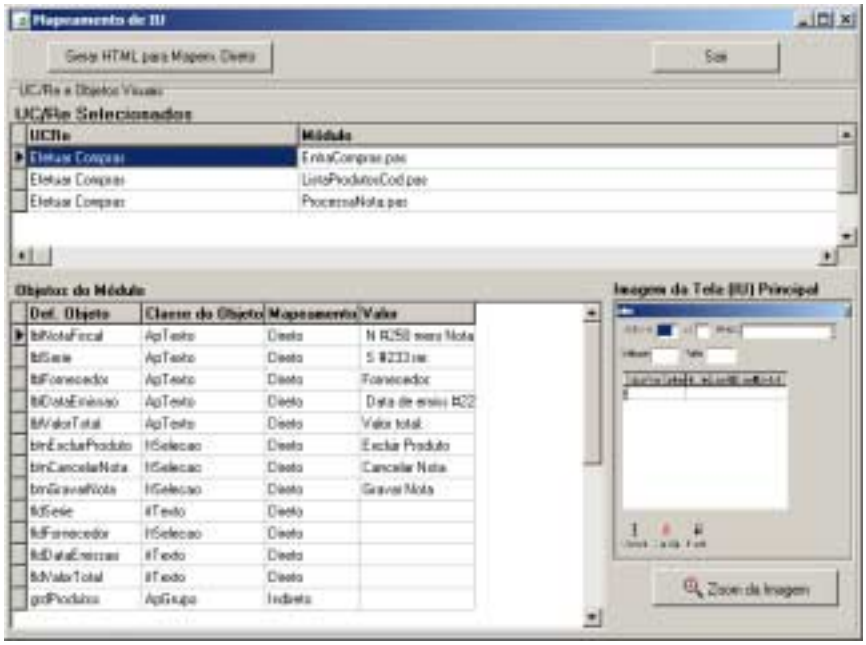



FIGURA 5.19 – Janela com os objetos visuais, obtidos dos UC/Re selecionados.

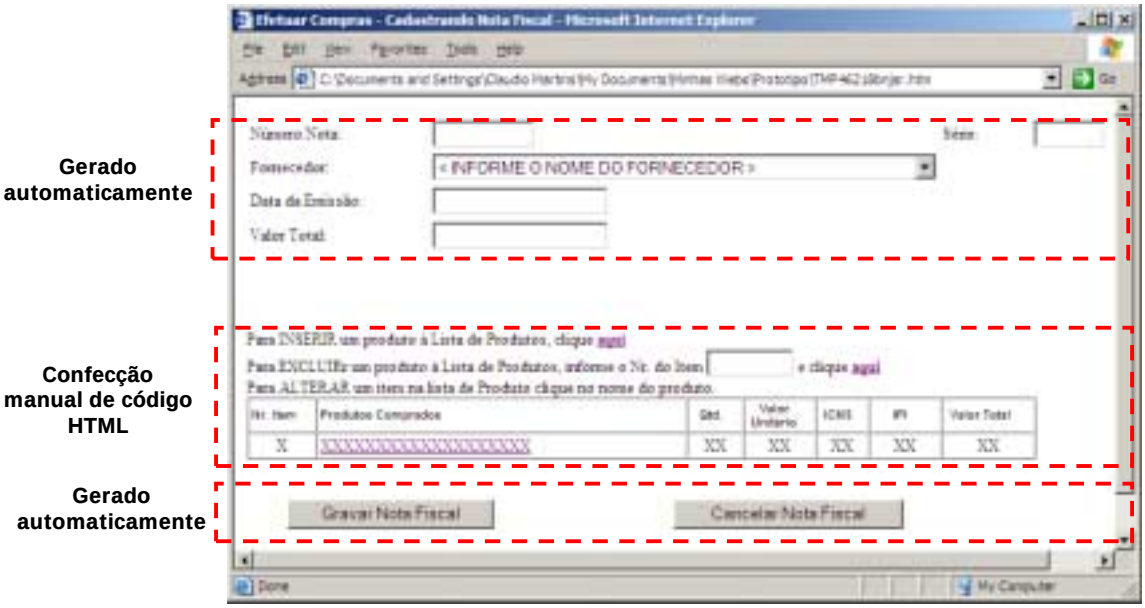



FIGURA 5.20 – Visão do gabarito de página HTML correspondente à interface principal do UC/Re “Efetuar Compras”.

## 5.3 Considerações Finais

Este capítulo apresentou o protótipo para um ambiente de apoio ao processo de reengenharia proposto nesta dissertação. As principais funcionalidades foram implementadas e validadas por um estudo de caso, que representa os exemplos citados no decorrer do texto da dissertação.

O sistema utilizado como estudo de caso é uma típica aplicação cliente/servidor, implementada em Delphi, acessando tabelas locais no formato Paradox para controlar estoque de produtos eletrônicos. Escolheu-se um caso de uso simples para demonstrar as funcionalidades do protótipo e, sobretudo, validar o método do processo de reengenharia proposto.

Apesar do sistema legado não manipular os dados com recursos e comandos em SQL, usando apenas acesso direto a tabelas Paradox, o protótipo atingiu os objetivos propostos para cada etapa discutida na metodologia. Portanto, com algumas adaptações no analisador sintático (no componente “Recuperar Abstrações”) é possível tratar e recuperar abstrações originadas em código SQL.

Os recursos de instrumentação de código e construção dos UC/Re demonstraram ser bastante úteis na compreensão do comportamento dinâmico da aplicação. A identificação de abstrações referenciadas em cada UC/Re construído, delimita a quantidade de informação e auxilia o reprojetista na tarefa de compreensão das partes que serão migradas para o novo ambiente, evitando a manipulação com abstrações desnecessárias e não pertencentes ao escopo do projeto de reengenharia.

A construção do modelo de classes conceituais do sistema (MCS) foi facilitada pela forma automática de se obter as informações sobre classes, atributos e métodos candidatos, a partir das relações entre tabelas de dados, campos e procedimentos implementados no código legado. A maior dificuldade encontrada na criação do MCS é a possibilidade de reestruturação do código legado em função das anomalias presentes em alguns procedimentos. Como regra geral, cada procedimento deveria gerar um método, entretanto muitos deles apresentam baixa coesão ou manipulam vários objetos *dataset*. Este fato obriga o reprojetista a intervir no código legado, segmentando o procedimento anômalo em blocos de procedimentos menores, forçando a realização de uma nova iteração na descoberta de abstrações e na construção dos UC/Re.

A tarefa de migração de interfaces provou ser perfeitamente possível quanto à geração automática de código HTML, com relação aos objetos visuais da interface do tipo WIMP que são diretamente mapeados em código de páginas Web. Os objetos mapeados indiretamente são identificados e devem ser implementados por um especialista em projeto de interfaces Web (um *Web Design*, por exemplo), seguindo regras e padrões de projeto para este tipo de interface.

# 6 Conclusões

Este trabalho apresentou uma proposta metodológica de reengenharia de *software* para uma classe de sistemas de informação, implementados em ambientes de linguagens visuais, sob o paradigma procedural/estruturado, baseado em objetos e eventos sob interfaces gráficas do tipo WIMP. O objetivo principal da proposta é preparar o modelo de classes conceituais e o projeto da interface do usuário para serem utilizados no reprojeto de uma nova aplicação, a partir de requisitos e abstrações obtidos da aplicação legada, através de técnicas e métodos de engenharia reversa. Para delimitar e exemplificar o processo, escolheu-se como domínio de linguagem de programação do *software* legado, o ambiente Delphi (sob a linguagem Object Pascal), e para plataforma da aplicação alvo, a Web.

É proposto também um ambiente CASE, no qual é descrito o funcionamento de um protótipo que automatiza grande parte das funcionalidades discutidas nas etapas do processo. Algumas ferramentas desenvolvidas por terceiros, disponíveis para uso livre, são empregadas na redocumentação do sistema legado e na elaboração dos modelos UML do novo sistema. Um estudo de caso, apresentando uma funcionalidade específica de um sistema desenvolvido em Delphi, no paradigma procedural, é usado para demonstrar o protótipo e serve de exemplo para a validação do processo. Como resultado do processo, obtém-se o modelo de classes conceituais da aplicação legada e gabaritos de páginas em HTML, representando os componentes visuais da interface do tipo WIMP do sistema legado na nova plataforma. A partir da elaboração desses artefatos, o reprojetista terá condições de desenvolver o sistema em uma nova plataforma, seguindo os bons preceitos do paradigma OO.

Antes da formulação da proposta, no Capítulo 2, são apresentados os principais conceitos, fundamentos e terminologias empregados no texto da dissertação. Discute-se temas ligados à engenharia reversa e reengenharia de sistemas de *software*. Em seguida, mostra-se a visão geral dos componentes principais presentes tanto na arquitetura dos sistemas legados especificados no enfoque escolhido (Delphi, em questão), quanto na arquitetura Web, alvo do processo de migração.

Para a formulação da proposta, várias abordagens foram utilizadas. No Capítulo 3, quatro assuntos são discutidos para reunir enfoques com características comuns. O primeiro grupo forma a base teórica para todos os métodos de engenharia reversa, ao abordar a recuperação de conceitos e de fatos em sistemas de *software* legados tradicionais, projetados no paradigma procedural. O segundo, apresenta o método Fusion/RE (PENTEADO, 1996) e a sua evolução, o PRE/OO (LEMOS, 2002); com adaptações no enfoque, esses métodos formam a base para algumas etapas do processo proposto. No grupo seguinte, são discutidos alguns projetos de pesquisa no campo da refatoração de projetos OO e de formatos de representação de abstrações, úteis no processo de refinamento do modelo de classes e na estruturação e armazenamento do metamodelo proposto na metodologia, respectivamente. O último grupo trata do projeto MORPH (MOORE, 1998), que apesar de considerar um processo genérico para reengenharia de interface de usuário no modo caractere para um ambiente gráfico, serve pelas idéias gerais sugeridas. Algumas dessas idéias são adotadas na etapa de migração de interfaces de usuário, tais como a formulação de regras para mapeamento entre componentes de ambientes de interfaces distintas, independente do idioma do código de implementação, e a defesa da participação de um perito humano para decidir e resolver as ambigüidades geradas no processo de geração automática da interface alvo.

No Capítulo 4, o processo de reengenharia é detalhado em quatro etapas e metamodelos são construídos ao longo do processo a fim de auxiliar a construção de modelos de projeto para o novo sistema. Os metamodelos são modelos de representação para os componentes obtidos por engenharia reversa do sistema origem e são armazenados em um repositório. A divisão em etapas visa simplificar a aplicação do processo, facilitar a compreensão da metodologia e transpor em tarefas que possam ser automatizas por ferramentas de *software*. Na etapa "Recuperar Abstração", realiza-se a engenharia reversa através da análise estática do código legado, que extrai abstrações em forma de módulos, procedimentos, objetos de dados e de interface visual de interação com o usuário; essas abstrações formam a base de conhecimento, armazenadas em um repositório. A etapa "Construir UC/Re" propõe recuperar os aspectos dinâmicos da aplicação legada, usando um mecanismo de rastreio de execução e representando os requisitos funcionais na ferramenta chamada "*use case para reengenharia*", UC/Re. A partir dos UC/Re, delimita-se o escopo do projeto de migração, isto é, define-se quais abstrações serão efetivamente utilizadas nas etapas seguintes. Além disso, UC/Re proporciona um ferramental para o acompanhamento do projeto e validação de requisitos, útil nas fases de implementação, teste e implantação da nova aplicação. Na etapa "Gerar MCS (Modelo Conceitual do Sistema)", produz-se o modelo de classes de negócio para o novo sistema, obtido, em parte, a partir das abstrações que representam os dados e procedimentos que atuam sobre esses dados, e em parte pela aplicação de práticas e heurísticas de projeto OO. A última etapa trata do mapeamento das interfaces legadas do tipo WIMP para o ambiente Web, utilizando-se duas estratégias: uma para traduzir componentes que possuem similaridades em ambos ambientes, outra para identificar componentes que não são mapeados diretamente em código de páginas HTML.

Para demonstrar a viabilidade da aplicação do processo de reengenharia, um protótipo de um ambiente de apoio é apresentado no Capítulo 5. Para demonstração do ambiente CASE, com o uso do protótipo e de ferramentas de terceiros, é utilizado como exemplo um caso de uso de um sistema de informação que segue o domínio discutido na abordagem, isto é, um sistema implementado em Delphi com as mesmas características mencionadas na introdução da dissertação.

Apesar do processo proposto abordar as fases iniciais de um projeto completo de reengenharia, isto é, as fases de compreensão e reprojeto (de forma parcial) do sistema legado, evidenciou-se a viabilidade de reconstrução de sistemas por meio de modelos de alto nível de abstração, evitando-se tradução e transformação direta de código, algo muito difícil de realizar em plataformas de natureza distintas, ou usando técnicas de empacotamento (*wrapping*), solução quase sempre transitória em um processo de migração. Adotando o processo proposto, o reprojetista terá condições de desenvolver o sistema em uma nova plataforma (Web, para este caso), seguindo os bons preceitos do paradigma OO.

## 6.1 Contribuições

Como principais contribuições deste trabalho, podem ser citadas:

- A elaboração de um processo de reengenharia a ser aplicado para as fases de compreensão e reprojeto de sistemas legados, com mudança do paradigma: de procedural para orientado a objetos.
- Uma abordagem dirigida a modelos de alto nível de abstração, em vez de técnicas de tradução ou transformação direta do *software* legado, difíceis de realizar em plataformas de naturezas diferentes.

- O uso da ferramenta UC/Re e do mecanismo de rastreio de execução, úteis para a análise dinâmica da aplicação legada, validação dos requisitos, acompanhamento dos casos de teste e verificação das equivalências de funcionalidades em ambos os sistemas (o legado e o novo).
- O tratamento de mapeamento dado à interface de usuário, do tipo WIMP para o ambiente Web, seguindo duas estratégias: uma com regras de tradução direta, outra utilizando padrões de projeto de interface visual, com participação de um perito em projeto de interface Web.
- A definição de um ambiente CASE para automatização das etapas propostas no processo por um protótipo, apoiado por ferramentas de terceiros.

## 6.2 Trabalhos Futuros

Em prosseguimento a este trabalho, propõe-se estender e melhorar pontos do processo e do ambiente propostos na dissertação.

- Como a tarefa de segmentação é feita de forma manual, deve-se estudar formas automatizadas (ou semiautomatizadas) para a reestruturação de código legado, seguindo técnicas e ferramentas como a de sistemas transformacionais. Um exemplo de sistema transformacional que vem sendo utilizado nesta área é o Draco-PUC (LEITE et al., 1994), como constatado em pesquisas sobre reengenharia em sistemas codificados na linguagem Clipper (ABRAHÃO; PRADO, 1999) (JESUS; FUKUDA; PRADO, 1999), em que foram combinados o método Fusion/RE e a máquina transformacional Draco-PUC.
- Ampliar o analisador sintático do protótipo (presente no componente para recuperar abstrações), para tratar os objetos de dados que manipulam código SQL.
- Incorporar no protótipo, a geração automática das definições das classes em XMI, sem intermediação de mecanismos externos. A versão atual utiliza ESS-Model para realizar essa operação.
- Pesquisar mecanismos e regras que possam automatizar a estratégia de mapeamento de interfaces do tipo WIMP, quando essa for realizada por meio indireto, isto é, através de gabaritos de páginas HTML escolhidos a partir de um catálogo de padrões de projeto de interface.
- Ampliar o processo até as fases de implementação e implantação do novo sistema, contextualizando as decisões de projeto para arquiteturas típicas da Web.
- Instanciar a metodologia e o protótipo do ambiente para outros domínios de linguagens, como o Visual Basic, aplicando o processo em estudos de caso de complexidade maior que o apresentado nesta dissertação.

## 6.3 Produção Científica

Como produção científica do autor, durante o período de desenvolvimento do mestrado na instituição, estão publicados os seguintes trabalhos, artigos e resumos:

1. MARTINS, Cláudio Roberto de Lima; PIMENTA, Marcelo. S.; PRICE, Ana Maria de A. **Reengenharia de Aplicações Cliente/Servidor para a Plataforma Web Usando Metamodelos**. In: Conferencia Latinoamericana de Informática, infoUYclei 2002, 28.; 2002, Montevideo, Uruguai. Program and Abstracts... Montevideo, Uruguai: CLEI, p. 50, 2002.

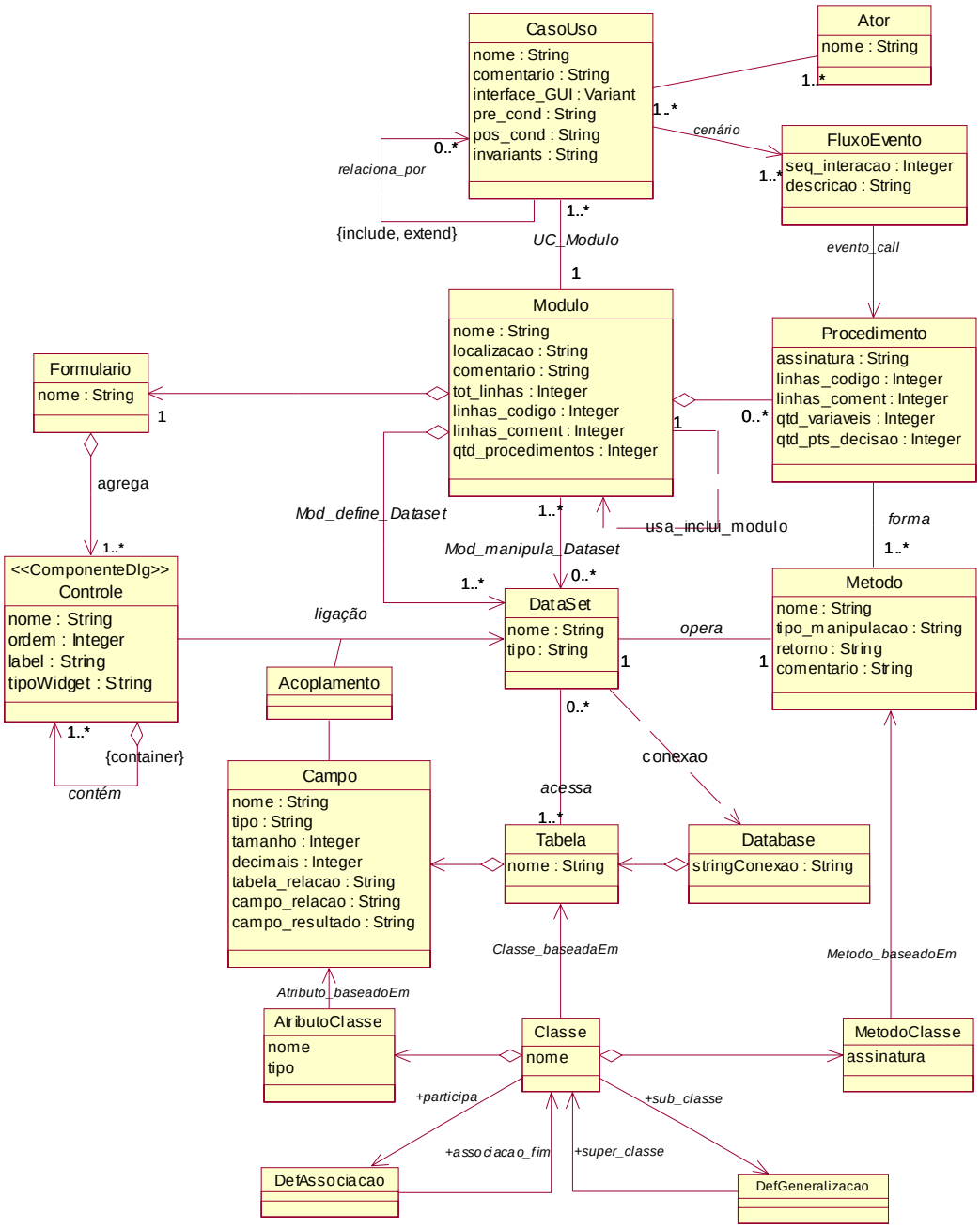
2. MARTINS, Cláudio Roberto de Lima; PIMENTA, Marcelo. S.; PRICE, Ana Maria de A. **Migração de Aplicações Cliente/Servidor para a Plataforma Web Usando Metamodelos**. In: Jornada Ibero-Americana de Engenharia de Software e Engenharia de Conhecimento, JIISIC, 2., Salvador, 2002. Anais... Salvador: UNIFACS, 2002. 1 CD-ROM, sessão técnica 7.

3. MARTINS, Cláudio Roberto de Lima; PIMENTA, Marcelo. S.; PRICE, Ana Maria de A. **Mapeamento de Interfaces WIMP para Interfaces Web**. In: Symposium on Human Factors in Computer Systems, IHC, 5., 2002, Fortaleza. Proceedings... Fortaleza: SBC, 2002. p. 381-383.

4. MARTINS, Cláudio Roberto de Lima; PRICE, Ana Maria de A.; PIMENTA, Marcelo. S. **Sistematização do Desenvolvimento de Aplicações Hipermídia na Web: Comparação de Metodologias**. In: Simpósio Brasileiro de Sistemas Multimídia e Hipermídia, SBMIDIA, 7., 2001, Florianópolis. Anais... Florianópolis: UFSC, 2001. p. 238-239.

5. MARTINS, Cláudio Roberto de Lima; PRICE, Ana Maria de A. **Métodos e Padrões de Projeto para Aplicações na Web**. Trabalho individual. Biblioteca do Instituto de Informática. UFRGS. 2001. 65 f.

# Anexo 1 Definição Completa do Metamodelo



# Anexo 2 Mapeamento de Componentes Visuais em Delphi

| Classe do Objeto | Classificação do componente de diálogo | Propriedades relevantes do objeto |
|---|---|---|
| TForm | Contêiner raiz | *Caption*, tamanho (*Width, Height*) |
| TEdit | componente de interação textual | *TabOrder*, *Text*, posição (*Left, Top*) e tamanho (*Width, Height*) |
| TDBText | componente de interação textual | *DataSource*, *DataField*, *TabOrder*, posição (*Left, Top*) e tamanho (*Width, Height*) |
| TMemo, TDBMemo | componente de interação textual (múltiplas linhas) | *Lines.Strings (*em *TMemo)*, *TabOrder*, Posição (*Left, Top*) e Tamanho (*Width, Height*) |
| TButton, TDBNavigator | componente de interação de seleção | Posição (Left, Top) e Tamanho (Width, Height), Caption, TabOrder |
| TCheckBox, TDBCheckBox | componente de interação de seleção | Posição (Left, Top) e Tamanho (Width, Height), Caption, TabOrder |
| TRadioButton | componente de interação de seleção | Posição (Left, Top) e Tamanho (Width, Height), Caption, TabOrder |
| TListBox, DBListBox | componente de interação de seleção | Posição (Left, Top) e Tamanho (Width, Height),Items.Strings, TabOrder |
| TComboBox, TDBComboBox | componente de interação de seleção | Posição (Left, Top) e Tamanho (Width, Height), TabOrder, Items.Strings |
| TRadioGroup, TDBRadioGroup | componente de interação de seleção | Posição (Left, Top) e Tamanho (Width, Height), Caption, Items.Strings, TabOrder |
| TMaskEdit | componente de interação de texto (qualificado/quantificado com restrição na máscara de edição) | Posição (Left, Top) e Tamanho (Width, Height), EditMask , MaxLength, TabOrder , Text |
| TTabControl | componente de interação de agrupamento (contêiner) | Posição (Left, Top) e Tamanho (Width, Height), TabOrder, Tabs.Strings |
| TPageControl | componente de interação de agrupamento (contêiner) | Posição (Left, Top) e Tamanho (Width, Height), *TabOrder* |
| TMainMenu | componente de interação de agrupamento (contêiner) | |
| TMenuItem | componente de interação de seleção | *Caption* |
| TLabel, | componente de apresentação textual | *Caption*, Posição (Left, Top) e Tamanho (Width, Height) relativos ao formulário (medidos em *pixels*). |
| TGroupBox | componente de apresentação de agrupamento (contêiner) | Posição (Left, Top) e Tamanho (Width, Height), Caption, TabOrder |
| TPanel | componente de apresentação de agrupamento (contêiner) | Posição (Left, Top) e Tamanho (Width, Height), Caption, TabOrder |
| TTabSheet | componente de interação de seleção | Caption |
| TGroup | componente de apresentação de agrupamento (contêiner) | Caption |
| TImage, TDBImage1 | componente de apresentação de imagem | Posição (Left, Top) e Tamanho (Width, Height), Picture.Data (em TImage) |
| TDBGrid | componente de apresentação de agrupamento (contêiner) | DataSource, (Columns/ item/FieldName)* |

# Anexo 3 Mapeamento de Tipos de Dados

Mapeamento de tipos de dados de campos (Paradox) para domínios OO.

| **Tipo Paradox** | **Correspondente OO** |
|---|---|
| Alpha | String |
| Number | Double |
| Money | Double |
| Short | Short |
| Long Integer | Integer |
| BCD | Variant |
| Date | Date |
| Time | Date |
| Timestamp | Date |
| Memo | String |
| Formatted Memo | String |
| Graphic | Variant |
| OLE | Variant |
| Logical | Boolean |
| Autoincrement | Integer |
| Binary | Variant |
| Bytes | Byte |

# Referências

ABRAHÃO, S. M.; PRADO, A. F. Web-Enabling Legacy Systems Through Software Transformations. In: IEEE INTERNATIONAL WORKSHOP ON ADVANCE ISSUES OF E-COMMERCE AND WEB-BASED INFORMATION SYSTEMS, 1999, Santa Clara, USA. Proceedings... [S.l.: s.n.], 1999. p. 149-152.

ARGOUML. **Tigris.org - Open Source Software Engineering**. Disponível em: <http://argouml.tigris.org/>. Acesso em: 20 mar. 2003.

BISBAL, J.et al. J. Legacy Information Systems: issues and directions. **IEEE Software**, Los Alamitos, v. 16, n. 5, p. 103-111, Sept./Oct. 1999.

BOOCH, G.; JACOBSON, I.; RUMBAUGH, R. **Unified Modeling Language**: guia do usuário. Rio de Janeiro: Campus, 1997.

BRAGA, R. **Padrões de Software a partir de Engenharia Reversa de Sistemas Legados**. 1998. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, USP, São Carlos.

BRODIE, M.; STONEBRAKER, M. **Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach**. San Francisco, California: Morgan Kaufmann, USA, 1995.

CAGNIN, M.; PENTEADO, R.; GERMANO, F.; MASIERO, P. Reengenharia com Uso de Padrões de Projeto. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 13., 1999, Florianópolis. **Anais...** Florianópolis: UFSC, 1999. p. 273-288.

CAMARGO, V. **Reengenharia Orientada a Objetos de Sistemas COBOL com a Utilização de Padrões de Projeto e Servlets**. 2001. Dissertação (Mestrado em Computação) – Departamento de Computação, UFSCar, São Carlos.

CANFORA, G.; CIMITILE, A.; CARLINI, U. A Logic-Based Approach to Reverse Engineering Tools Production. **IEEE Transactions on Software Engineering**, Los Alamitos, v. 18, n. 12, p. 1053-1064, Dec. 1992.

CANFORA, G.; CIMITILE, A.; TORTORELLA, M.; MUNRO, M. A Precise Method for Identifying Reusable Abstract Data Types in Code. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, ICSM, 1994, Victoria, Canada. **Proceedings…** [S.l.]: IEEE, 1994. p. 404-413.

CDIF TECHNICAL COMMITTEE. **CDIF framework for modeling and extensibility**. [S.l.]: Electronic Industries Association, 1994. (Technical Report EIA/IS-107). Disponível em: <http://www.eigroup.org/cdif/index.html>. Acesso em: 20 mar. 2003.

CHI, U. Formal Specification of User Interfaces: A Comparison and Evaluation of Four Axiomatic Approaches. **IEEE Transactions on Software Engineering**, Los Alamitos, v. 11, n. 8, p. 671-685, Aug. 1985.

CHIKOFSKY, E.; CROSS II, J. Reverse engineering and design recovery : a taxonomy. **IEEE Software**, Los Alamitos, v. 7, n. 1, p. 13-17, Jan./Feb.1990.

COAD, P.; YOURDON, E. **Object-Oriented Analysis**. 2nd ed. New Jersey: Prentice Hall, 1991.

COLEMAN, D. et al. **Object-Oriented Development – The Fusion Method**. New Jersey: Prentice Hall, 1994.

CONALLEM, J. **Building Web Applications with UML**. Massachusetts: Addison Wesley, 2000.

CORNELL, G.; STRAIN, T. **Delphi:** segredos e soluções. São Paulo: Makron Books, 1995.

DEMEYER, S.; DUCASSE, S.; NIERSTRASZ, O. A Pattern Language for Reverse Engineering. In: EUROPEAN CONFERENCE ON PATTERN LANGUAGES OF PROGRAMMING AND COMPUTING, 4., 1999, Bad Irsee, Germany. **Proceedings…** [S.l.]: Universitätsverlag Konstanz GmbH, 1999.

DIX, A. et al. **Human-Computer Interaction**. New Jersey: Prentice-Hall, 1993.

DUCASSE, S.; DEMEYER, S. **The FAMOOS Object-Oriented Reengineering Handbook**. University of Berne, 1999. Disponível em: <http://www.iam.unibe.ch/~famoos/handbook/>. Acesso em: 20 mar. 2003.

ESS-MODEL. **Endean ESS-Model**. Disponível em: <http://www.essmodel.com>. Acesso em: 20 mar. 2003.

FOLEY, J. et al. **Computer Graphics Principles and Practice**. 2nd ed. Massachusetts: Addison-Wesley, 1990.

FONG, J.; HUANG, S. **Information Systems Reengineering**. Singapura: Springer-Verlag, 1997.

FOWLER, M. et al. **Refactoring:** Improving the Design of Existing Code. Massachusetts: Addison Wesley, 1999.

GALL, H.; KLOSCH, R. Finding Objects in Procedural Programs: an Alternative Approach. In: WORKING CONFERENCE ON REVERSE ENGINEERING, 2., 1995, Toronto, Canada. **Proceedings…** [S.l.]: IEEE, 1995. p. 208-216.

GALL, H.; KLOSCH, R.; MITTERMEIR, R. Object-Oriented Re-Architecturing. In: EUROPEAN SOFTWARE ENGINEERING CONFERENCE, ESEC, 5., 1995, Sitges, Spain. **Proceedings…** [S.l.]: Springer-Verlag, 1995. p. 499-519.

GARRIDO, A.; ROSSI, G.; SCHWABE, D. Pattern Systems for Hypermedia. In: PATTERN LANGUAGE OF PROGRAMMING, PLOP, 1997, Monticello, USA. **Proceedings…** [S.l.: s.n.], 1997. Disponível em: <http://st-www.cs.uiuc.Edu/~plop/plop97/Workshops.html>. Acesso em: 20 mar. 2003.

GELLERSEN, H-W.; GAEDKE, M. Object-oriented Web Application Development. **IEEE Internet Computing**, Los Alamitos, v. 3, n.1, p. 60-68, Jan./Feb. 1999.

GINIGE, A.; MURUGESAN, S. Web Engineering: An Introduction. **IEEE Multimedia**, Los Alamitos, v.8, n.1, p. 14-18, Jan./Mar. 2001.

HIX, D.; HARTSON, H. **Developing User Interfaces - Ensuring Usability Through Product and Process**. Indianapolis: John Wiley and Sons, 1993.

JACOBSON, I. et al. **Object-Oriented Software Engineering - A Use Case Driven Approach**. Massachusetts: Addison-Wesley, 1992.

JACOBSON, I. et al. **The Object Advantage:** Business Process Reengineering with Object Technology. Massachusetts: Addison-Wesley, 1995.

JARZABEK, S.; WANG, G. Model-based Design of Reverse Engineering Tools. **Journal of Software Maintenance: Research and Practice**, Indianapolis, v.10, n. 5, p. 353-380, Sept./Oct. 1998.

JESUS, E.; FUKUDA, A.; PRADO, A. Reengenharia de Software para Plataformas Distribuídas Orientadas a Objetos. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 13., Florianópolis. **Anais...** Florianópolis: UFSC, 1999. p. 289-304.

JUNKER, R. **The Delphi Inspiration:** DIPasDoc. Disponível em: <http://www.zeitungsjunge.de/delphi/PasDoc>. Acesso em: 20 mar. 2003.

KOSCHKE, R.; GIRARD, J-F.; WÜRTHNER, M. An Intermediate Representation for Reverse Engineering Analyses. In: WORKING CONFERENCE ON REVERSE ENGINEERING, WCRE, 5., 1998, Honolulu, Hawaii, USA. **Proceedings…** [S.l.]: IEEE, 1998. p. 241-250.

LARMAN, C. **Applying UML and patterns**. New Jersey: Prentice-Hall, 1998.

LEITE, J.; FREITAS, F., SANTANNA, M. Draco-PUC Machine: a Technology Assembly for Domain Oriented Software Development. In: INTERNATIONAL CONFERENCE OF SOFTWARE REUSE, 3., 1994, Rio de Janeiro, Brasil. **Proceedings…** [S.l.]: IEEE, 1994. p. 94-100.

LEMOS, G. **PRE/OO – Um Processo de Reengenharia Orientada a Objetos com Ênfase na Garantia da Qualidade**. 2002. Dissertação (Mestrado em Ciência da Computação) – Centro de Ciências Exatas e de Tecnologia, UFSCar, São Carlos.

LIU, S.; WILDE, N. Identifying objects in a conventional procedural language: an example of data design recovery. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, ICSM, 1990, San Diego, USA. **Proceedings…** [S.l.]: IEEE, 1990. p. 266-271.

MARTINS, C.; PIMENTA, M.; PRICE, A. Migração de Aplicações Cliente/Servidor para a Plataforma Web Usando Metamodelos. In: JORNADA IBERO-AMERICANA DE ENGENHARIA DE SOFTWARE E ENGENHARIA DE CONHECIMENTO,

JIISIC, 2., Salvador, Brasil. **Proceedings...** Salvador: ICMC-USP, UNIFACS, 2002. 1 CD-ROM, sessão técnica 7, artigo 20.

MARTINS, C.; PIMENTA, M.; PRICE, A. Reengenharia de Aplicações Cliente/Servidor para a Plataforma Web Usando Metamodelos. In: CONFERENCIA LATINOAMERICANA DE INFORMÁTICA, INFOUYCLEI, 28., 2002, Montevideo, Uruguai. **Program and Abstracts...** Montevideo, Uruguai: CLEI, 2002. p. 50.

MARTINS, C.; PIMENTA, M.; PRICE, A. Mapeamento de Interfaces WIMP para Interfaces Web. In: SYMPOSIUM ON HUMAN FACTORS IN COMPUTER SYSTEMS, IHC, 5., 2002, Fortaleza, Brasil. **Proceedings…** Fortaleza: SBC, 2002. p. 381-383.

MCCABE, T. A Complexity Measure. **IEEE Transactions on Software Engineering**, New York, v.2, n. 4, p. 308-320, Dec. 1976.

MOORE, M.; MOSHKINA, L. Migrating Legacy Systems to the Internet: Shifting Dialogue Initiative. In: WORKING CONFERENCE ON REVERSE ENGINEERING, WCRE, 7., 2000, Brisbane, Australia. **Proceedings…** [S.l.]: IEEE, 2000. p. 52-57.

MOORE, M. **User Interface Reengineering**. 1998. Thesis (PhD in Computing) - College of Computing, Georgia Institute of Technology, Atlanta, GA, USA.

NIELSEN, J.; TAHIR, M. **Homepage Usability:** 50 Websites Deconstructed. Indianapolis: New Riders Publishing, 2001.

PAULK, M. et al. **The Capability Maturity Model:** Guidelines for Improving the Software Process. Reading, MA: Addison-Wesley, 1995.

PENTEADO, R. **Um Método para Engenharia Reversa Orientada a Objetos**. 1996. Tese (Doutorado em Física Computacional) – Instituto de Física de São Carlos, USP, São Carlos.

PENTEADO, R.; MASIERO, P.; PRADO, A.; BRAGA, R. Reengineering of Legacy Systems Based on Transformation Using the Object–Oriented Paradigm. In: WORKING CONFERENCE ON REVERSE ENGINEERING, WCRE, 5., 1998, Honolulu, Hawai, USA. **Proceedings…** [S.l.]: IEEE, 1998. p.144-153.

PENTEADO, R.; MASIERO, P. C., CAGNIN, M. I. An Experiment of Legacy Code Segmentation to Improve Maintainability. In: EUROPEAN CONFERENCE ON SOFTWARE MAINTENANCE REENGINEERING, CSMR, 3., 1999, Amsterdan. **Proceedings…** [S.l.]: IEEE, 1999. p 111-119.

PRESSMAN, R. (Manager) What a Tangled Web We Weave. **IEEE Software**, Los Alamitos, v.17, n.1, p. 18-21. Jan./Feb. 2000.

PRESSMAN, R. **Software Engineering:** a Practitioner's Approach. 5th ed. Auchland: McGraw-Hill, 2001.

PROGRAME com Força do Visual. **Informática Exame**, São Paulo, v. 11, n. 126, p.26-29, set. 1996.

QUILICI, A. Reverse Engineering of Legacy Systems: A Path Toward Success. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 1995, 17., Seattle, USA. **Proceedings…** New York: ACM, 1995. p. 333-336.

RECCHIA, E. **Engenharia Reversa e Reengenharia Baseadas em Padrões**. 2002. Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, UFSCar, São Carlos.

RICH, C.; WILLS, L. Recognizing a Program's Design: A Graph-Parsing Approach. **IEEE Software**, Los Alamitos, v.7, n.1, p. 82-89, Jan./Feb. 1990.

SILVA, C.; SANTOS, P. Delphi x Visual Basic. **Byte Brasil**, São Paulo, v.7, n.7, p. 24-26, jul. 1997.

ROSSI, G.; SCHWABE, D.; GARRIDO, A. Design Reuse in Hypermedia Applications Development. In: ACM CONFERENCE ON HYPERTEXT, Hypertext, 8., 1997, Southampton, UK. **Proceedings…** New York: ACM, 1997. p 57-66.

SNEED, M.; NYARY, E. Extracting object-oriented specifications from procedurally oriented programs. In: WORKING CONFERENCE ON REVERSE ENGINEERING, WCRE, 2., 1995, Toronto, Canada. **Proceedings…** [S.l.]: IEEE, 1995. p. 217–226.

TEREKHOV, A.; VERHOEF, C. The Realities of Language Conversions. **IEEE Software**, Los Alamitos, v.17, n.6, p. 111-124. Nov./Dec. 2000.

TICHELAAR, S. et al. A Meta-model for Language-Independent Refactoring. In: INTERNATIONAL SYMPOSIUM ON PRINCIPLES OF SOFTWARE EVOLUTION, ISPSE, 2000, Kanazawa, Japan. **Proceedings…** [S.l.]: IEEE, 2000.

TICHELAAR, S.; DUCASSE, S.; DEMEYER, S. FAMIX and XMI. In: WORKING CONFERENCE ON REVERSE, WCRE, 7., 2000, Brisbane, Australia. **Proceedings…** [S.l.]: IEEE, 2000. p. 296-298.

UNIFIED MODELING LANGUAGE (UML). **Catalog of OMG Modeling and Metadata Specifications**. Disponível em: <http://www.omg.org/technology/documents/modeling_spec_catalog.htm>. Acesso em: 20 mar. 2003.

WATSON, A.; MCCABE, T. **Structured Testing:** a Testing Methodology Using the Cyclomatic Complexity Metric. 1996. Disponível em: <http://hissa.ncsl.nist.gov/HHRFdata/Artifacts/ITLdoc/235/mccabe.html>. Acesso em 10 jun. 2002.

WEISER, M. Program slicing. **IEEE Transactions on Software Engineering**, Los Alamitos, v.10, n.4, p.353–357, Apr. 1984.

WONG, K. et al. Structural Redocumentation: a case study. **IEEE Software**, Los Alamitos, v.12, n.1, p. 46-54, Jan. 1995.

XML METADATA INTERCHANGE (XMI). **Catalog of OMG Modeling and Metadata Specifications**. Disponível em: <http://www.omg.org/technology/documents/modeling_spec_catalog.htm>. Acesso em: 20 mar. 2003.