UNIVERSIDADE FEDERAL DE SANTA CATARINA

Trabalho de Conclusão de Curso

# UM FRAMEWORK PARA DESENVOLVIMENTO DE APLICATIVOS CLIENTE/SERVIDOR

Joáber Biazus Cavichioli

Florianópolis, fevereiro de 2003.

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

# UM FRAMEWORK PARA DESENVOLVIMENTO DE APLICATIVOS CLIENTE/SERVIDOR

Joáber Biazus Cavichioli

Orientador: Olinto José Varela Furtado
Banca Examinadora: Patrícia Vilain
Juan Wilder Moore Espinoza

Florianópolis, 25 de fevereiro de 2003.

# Agradecimentos

Agradeço principalmente a meus pais, pelo investimento, carinho e amor a mim conferidos nestes anos de curso, sem vocês nada disso seria possível.

Ao meu orientador, por ter me aceito como seu orientando e por ter sido um excelente professor e um grande amigo de todos seus alunos.

A Universidade Federal de Santa Catarina, em particular ao Departamento de Informática e Estatística, pelo suporte oferecido para minha formação acadêmica.

Aos meus professores, pelo conhecimento a mim transmitido.

Aos meus colegas, que sem dúvida representaram os momentos mais felizes destes anos de curso e com certeza estarão sempre em meu coração e em minhas lembranças.

A Deus, por ter me criado e amado como grande pai que é.

# Sumário

**Lista de Figuras** ........ VI

**Resumo** ........ VII

**Abstract** ........ VIII

**Introdução** ........ 1
1.1 Motivação ........ 1
1.2 O Problema ........ 2
1.3 Objetivos Gerais ........ 3
1.4 Objetivos Específicos ........ 3
1.5 Organização do Texto ........ 3

**Frameworks Orientados a Objetos** ........ 5
2.1 Definição ........ 5
2.2 Classificação ........ 7
2.3 Qualidades de um Bom Framework ........ 9
2.4 Ciclo de Vida de Frameworks ........ 11
2.5 Indivíduos Envolvidos com o Desenvolvimento e o Uso de Frameworks ........ 15

**Padrões de Projeto** ........ 17
3.1 Introdução ........ 17
3.2 Definição ........ 18
3.3 Como padrões solucionam problemas de projeto ........ 19
3.3.1 Procurando objetos apropriados ........ 20
3.3.2 Determinando a granularidade dos objetos ........ 21
3.3.3 Especificando interfaces de objetos ........ 21
3.3.4 Especificando implementações de objetos ........ 23
3.3.4.1 Programando para uma interface, não para uma implementação ........ 24
3.3.4.2 Herança versus composição ........ 25
3.3.4.3 Delegação ........ 26
3.3.5 Relacionando estruturas de tempo de execução e compilação ........ 28
3.3.6 Projetando para mudanças ........ 29
Padrões Utilizados no Framework Desenvolvido ........ 31
4.1 Introdução ........ 31
4.2 Acoplamento Fraco ........ 32
4.3 Coesão Alta ........ 35
4.4 Template Method ........ 37
4.5 Fachada ........ 39

**Framework Desenvolvido** ........ 42
5.1 Introdução ........ 42
5.2 Ferramentas Utilizadas ........ 42
5.3 Padrões de Nomenclatura ........ 43

5.4 Módulos Estruturais ......................................................................................47
5.4.1 Módulo csComponentes ...........................................................................47
5.4.1.1 Componentes ...................................................................................47
5.4.1.2 Quadros de Diálogo .........................................................................50
5.4.2 Módulo csEntidades ................................................................................51
5.4.2.1 TCustomClientDataSet ......................................................................53
5.4.2.2 TcsCustomEntidade ..........................................................................53
5.4.2.3 TcsEntidade .....................................................................................55
5.4.2.4 TecsQuery .......................................................................................56
5.4.2.5 TcsAcessoDados...............................................................................57
5.4.2.6 TcsAcessoDadosIB ...........................................................................57
5.4.2.7 TsTrataMsgErro...............................................................................58
5.4.2.8 ucsMensagens..................................................................................59
5.4.3 Módulo csAplicação ................................................................................59
5.4.3.1 Estrutura hierárquica das janelas do módulo csAplicação .........................60
5.4.3.1.1 TfcsPrincipal................................................................................61
5.4.3.1.2 TfcsPrincSeguranca.......................................................................62
5.4.3.1.3 TfcsForm ....................................................................................62
5.4.3.1.4 TfcsEdicao ..................................................................................63
5.4.3.1.5 TfcsCadastro ...............................................................................63
5.4.3.1.6 TfcsCadastroGrid..........................................................................65
5.4.3.1.7 TfcsCadastroEdit...........................................................................65
5.4.3.1.8 TfcsCadastroGridEdit.....................................................................66
5.4.3.1.9 TfcsCadastroMestreGrid ................................................................67
5.4.3.2 Segurança .......................................................................................69
5.4.3.2.1 Cadastro de Janelas e seus Componentes ..........................................70
5.4.3.2.2 Configuração das Permissões ..........................................................71

**Conclusão ..............................................................................................................73**
Sugestões para Trabalhos Futuros .....................................................................73

**Bibliografia.............................................................................................................74**

**Anexo A – Artigo.....................................................................................................76**

**Anexo B – Código Fonte das Principais Classes .....................................................82**

# Lista de Figuras

FIGURA 2.1 - Aplicação desenvolvida totalmente .................................................................. 6
FIGURA 2.2 - Aplicação desenvolvida reutilizando classes de biblioteca ................................ 6
FIGURA 2.3 - Aplicação desenvolvida reutilizando um framework ........................................ 7
FIGURA 2.4 - Ciclo de vida de frameworks ........................................................................... 12
FIGURA 2.5 - Destaque da atuação do desenvolvedor no ciclo de vida de frameworks .......... 13
FIGURA 2.6 - Destaque da influência de aplicações existentes na geração e na alteração de um framework ........................................................................................................ 14
FIGURA 2.7 - Destaque influência de aplicações geradas a partir de um framework na alteração deste framework ........................................................................................ 15
FIGURA 2.8 - Indivíduos envolvidos no desenvolvimento tradicional de aplicações .............. 15
FIGURA 2.9 - Indivíduos envolvidos no desenvolvimento aplicações baseadas em frameworks ........................................................................................................ 16
FIGURA 3.1 - Exemplo de delegação ..................................................................................... 27
FIGURA 4.1 - Diagrama de classes parcial ............................................................................. 32
FIGURA 4.2 - Primeiro diagrama de colaboração ................................................................... 33
FIGURA 4.3 - Segundo diagrama de colaboração ................................................................... 33
FIGURA 4.4 - Exemplo template method ............................................................................... 38
FIGURA 4.5 - Exemplo da utilização do padrão Fachada ....................................................... 39
FIGURA 5.1 - Ilustração da utilização de botões contendo figuras ......................................... 48
FIGURA 5.2 - Hierarquia de classes dos componentes de botão ............................................ 49
FIGURA 5.3 - Quadro de diálogo de erro e confirmação de operação .................................... 51
FIGURA 5.4 - Hierarquia de classes do módulo csEntidades .................................................. 52
FIGURA 5.5 - Mensagem de erro tratada pela classe TcsTrataMsgErro ................................. 58
FIGURA 5.6 - Hierarquia de janelas do módulo csAplicação ................................................. 61
FIGURA 5.7 - Janela TfcsEdicao ............................................................................................ 63
FIGURA 5.8 - Janela TfcsCadastro ......................................................................................... 64
FIGURA 5.9 - Janela TfsCadastroGrid ................................................................................... 65
FIGURA 5.10 - Janela TfsCadastroEdit .................................................................................. 66
FIGURA 5.11 - Janela TfsCadastroGridEdit – pasta Tabela ................................................... 67
FIGURA 5.12 - Janela TfsCadastroGridEdit – pasta Formulário ............................................ 68
FIGURA 5.13 - Janela TfsCadastroMestreGrid ...................................................................... 68
FIGURA 5.14 - Classes que compõem a Segurança ............................................................... 70
FIGURA 5.15 - Janela de configuração de permissões ........................................................... 72

# Resumo

O presente trabalho descreve a abordagem de frameworks orientados a objetos no desenvolvimento de software bem como a importância da utilização de padrões de projetos no desenvolvimento de frameworks. Ele irá definir o que é um framework orientado a objetos, suas principais características, os benefícios que esta abordagem oferece, o que são padrões de projetos e os benefícios de sua utilização.

A finalidade de um framework orientado a objeto é reutilizar código e projeto. Para que isto seja possível, o framework deve ser tão flexível e extensível quanto possível para que possa dar suporte ao desenvolvimento de diferentes aplicações bem como evoluir à medida que as aplicações desenvolvidas sob o framework evoluem. Neste contexto a utilização de padrões de projetos torna-se de suma importância. Um framework projetado através do uso de padrões de projeto tem muito maior probabilidade de atingir altos níveis de reusabilidade de projeto e código, comparado com um que não usa padrões de projeto. Os padrões ajudam a tornar a arquitetura do framework adequada a muitas aplicações diferentes, sem necessidade de reformulação.

O framework implementado no presente trabalho é um framework orientado a objetos para o desenvolvimento de aplicativos cliente/servidor. Ele é um framework de baixa complexidade e alta extensibilidade, o que faz com que sua curva de aprendizado seja pequena e permite ao desenvolvedor agregar facilmente novas funcionalidades ao framework.

O presente trabalho irá descrever como o framework implementado foi estruturado, suas principais classes e relacionamentos e a descrição dos padrões de projeto utilizados em sua implementação.

**Palavras-chave:** frameworks orientados a objetos, padrões de projeto, reutilização.

# Abstract

This work presents the object oriented approach of frameworks for software development, and also the importance of using design pattners on framework development. It will define what is a object oriented framework, its major features, the benefits this approach leads to, what are design patterns and the benefits of their utilization.

The finality of an object oriented framework is code and project reuse. To make it possible, the framework must be as flexible and extensible as it cans be, so it cans both give support to the development of different applications and evolves while applications developed under the framework evolve. In this context, the utilization of design patterns is turned in a very important matter. One framework projected trough design patterns has much more probability of reaching high levels of code and project reuse, comparing with another one wich do not use design patterns. Design patterns help to turn the framework architecture suitable for many different applications without reformulation.

The framework implemented on this work is an object oriented framework to client/server applications development. It is a framework with low complexity and high extensibility, what makes its learning curve little and permits the developer to aggregate new functionality to the framework.

This work will describe how the implemented framework was structured, its major classes and relations, and the description of design patterns applied in the implementation.

**Palavras-chave:** object-oriented frameworks, design patterns, software reuse.

# Capítulo 1

# Introdução

## 1.1 Motivação

Diferentemente de épocas ulteriores, onde os desenvolvedores de artefatos de software[1] dedicavam seus esforços quase que exclusivamente para a etapa de implementação, sem dedicar o devido tempo à análise, documentação, projeto estrutural e testes dos mesmos, acarretando com isto uma gama de problemas no produto final, dificultando e postergando a etapa de manutenção do mesmo, atualmente, a Engenharia de Software nos oferece diferentes abordagens que buscam melhorar a qualidade dos artefatos de software bem como diminuir o tempo e esforços necessários para produzí-los.

Segundo Martin [MAJ 94], uma das preocupações mais urgentes na indústria da informática, hoje, é a necessidade de se criar softwares e sistemas corporativos de modo muito mais veloz e a um baixo custo.

É neste escopo que o desenvolvimento de Frameworks se torna cada vez mais comum e necessário. Framework é o esqueleto-base sobre o qual uma aplicação é construída, constituído de uma estrutura de classes com implementações incompletas, que, estendidas, permitem produzir diferentes artefatos de software. A grande vantagem desta abordagem é a reutilização de código e projeto, que tem por intuito diminuir o tempo e o esforço no desenvolvimento de softwares. Esta abordagem tem como grande característica estabelecer padrões estruturais, de interface e código e controlar o fluxo da aplicação, liberando o desenvolvedor de software desta preocupação.

Um dos principais objetivos de um Framework é a reutilização de artefatos de software, em contraposição ao desenvolvimento de todas as partes do sistema, fator que leva a uma melhoria na qualidade e um aumento na produtividade do desenvolvimento do software, isto baseado na perspectiva de que reutilizando artefatos de software já desenvolvidos e depurados, haverá uma redução no tempo

[1] A expressão *artefato de software* aqui se refere de forma genérica, não necessariamente a código, podendo se referir à aplicação, framework ou componente.

de desenvolvimento, testes e possibilidade de introdução de erros na produção de novos artefatos.

O presente trabalho tem por objetivo implementar um framework para desenvolvimento de aplicativos cliente/servidor (client/server) de duas camadas. Aplicativos cliente/servidor (client/server) de duas camadas são aplicativos que possuem a camada de apresentação (janelas do sistema) e das regras de negócio (não todas, elas podem também estar contidas também no banco de dados) em uma camada e os dados, armazenados em um banco de dados em outra camada. Desta forma a segunda camada (banco de dados) pode ser compartilhada por diferentes computadores (pertencentes a primeira camada).

Seu desenvolvimento foi motivado pela necessidade de um ambiente flexível, de fácil aprendizado e manipulação, que ajudasse a desenvolver softwares cliente/servidor de forma rápida, padronizada e consistente.

Seu principal atrativo em relação aos frameworks no mercado é sua simplicidade. Ele não é um framework complexo, que implementa todos os detalhes de um aplicativo client/server, muitas vezes amarrando o usuário a uma implementação própria e possuindo uma curva de aprendizado maior. Sem dúvida, frameworks desta natureza são muito bons, pois livram o desenvolvedor de software que o está utilizando praticamente de toda implementação e controle de janelas e estruturação de projeto, deixando o mesmo livre para dedicar seus esforços na solução do problema que se está atacando. O framework desenvolvido, ao contrário, tem uma curva de aprendizado bastante pequena. Esta simplicidade foi implementada para que o mesmo pudesse seguir as características a que se propõe, oferecer um ambiente flexível, de fácil aprendizado e manipulação, que ajude o desenvolvedor de software a desenvolver seus sistemas cliente/servidor de forma rápida, padronizada e consistente.

## 1.2 O Problema

Desenvolver um framework não é uma tarefa simples, bem como aprender a utilizá-lo. Seu desenvolvimento é dificultado por dever ser produzido de forma flexível o suficiente para que possa abranger diferentes tipos de aplicações pertencentes ao

seu domínio. A dificuldade de utilização está relacionada ao esforço requerido para apreender a desenvolver aplicações a partir do framework.

Existe uma carência de metodologias para desenvolvimento de frameworks. Elas estão mais voltadas para produzir código do que uma implementação de alto nível, bem como ao apoio à descrição de seu funcionamento. A documentação mais comum para descrever o funcionamento de um framework acaba sendo uma descrição textual, análise de código e exemplos de aplicações desenvolvidas a partir dele.

## 1.3 Objetivos Gerais

O presente trabalho tem por objetivo implementar um framework que dê suporte ao desenvolvimento de aplicativos cliente/servidor utilizando-se de padrões de projeto como guia para implementação de sua estrutura e do relacionamento entre suas classes. Também objetiva oferecer padronização para nomenclatura e código, o que contribui em muito para que o usuário do framework possa mais rapidamente compreendê-lo além de poder ser utilizado para padronizar futuros softwares construídos sobre ele.

## 1.4 Objetivos Específicos

Estudar a utilização de padrões de projetos para o desenvolvimento de frameworks.

## 1.5 Organização do Texto

O capítulo 2 fará uma introdução à abordagem de frameworks orientados a objetos. Serão discutidas suas principais características, finalidades e benefícios.

O capítulo 3 abordará padrões de projetos. Neste capítulo será definido o que são, para que servem, como solucionam problemas de projetos e quais os benefícios que trazem quando aplicados.

Baseado nos capítulos 2 e 3, o capítulo 4 aborda os padrões de projetos utilizados no projeto do framework desenvolvido no presente trabalho.

O capítulo 5 será dedicado ao projeto do framework desenvolvido. Será explicado em detalhes como sua estrutura foi montada, suas principais classes e janelas e o suporte que oferece ao desenvolvimento de novos aplicativos.

Finalmente, no capítulo 6 é feita a conclusão sobre o trabalho e algumas sugestões para trabalhos futuros.

# Capítulo 2

# Frameworks Orientados a Objetos

## 2.1 Definição

"Framework é um esqueleto de implementação de uma aplicação ou um subsistema de aplicação, em um domínio de problema particular. É composto de classes abstratas e concretas e provê um modelo de interação ou colaboração entre as instâncias de classes definidas pelo framework" [WIA 91].

"Um framework é utilizado através de configuração ou conexão de classes concretas e derivação de novas classes concretas a partir das classes abstratas do framework" [WIR 90],

"Um framework é um conjunto de classes cooperantes que constroem um projeto reutilizável para uma classe específica de software" [GAM 02 referenciando DEU 89, JF 88].

"Um framework é um conjunto de classes interrelacionadas com o objetivo de facilitar o desenvolvimento de um determinado domínio de aplicação. É composto de classes concretas e abstratas, estas últimas possuindo implementações incompletas que devem ser estendidas para compor as classes completas da aplicação final. A motivação para o desenvolvimento de frameworks é a reutilização de código e projeto, com o intuito de aumentar a produtividade no desenvolvimento de softwares" [SRP 00].

O framework dita a arquitetura de uma aplicação. Ele irá definir a estrutura geral, sua divisão em classes e objetos e em conseqüência as responsabilidades-chave das classes de objetos, como estas colaboram, e o fluxo de controle. Um framework predefine esses parâmetros de projeto, de maneira que o projetista/implementador de aplicativos possa se concentrar nos aspectos específicos da sua aplicação [GAM 02].

Um framework captura as decisões de projeto que são comuns ao seu domínio de aplicação. Assim, frameworks enfatizam a **reutilização de projeto** em relação à reutilização de código, embora um framework, geralmente, inclua subclasses concretas que o desenvolvedor pode utilizar imediatamente [GAM 02].

Para um aumento da produtividade, a granularidade do artefato de software a ser reutilizado é um fator importante. Um framework possui um grau mais elevado de granularidade do que a reutilização de bibliotecas de funções (reutilização de rotinas). Quando uma classe é reutilizada, todas suas rotinas (métodos) e atributos são reutilizados, portanto, reutilizar uma classe tende a ser mais eficiente que uma rotina isolada. A reutilização de classes da abordagem de framework se situa ainda num patamar de granularidade superior a reutilização de classes de uma biblioteca, pois neste caso são utilizados artefatos de software isolados, cabendo ao desenvolvedor estabelecer sua interligação, no caso do framework, é procedida a reutilização de um conjunto de classes interrelacionadas, interrelacionamento estabelecido pelo projeto do framework. Por reutilizar código e projeto, frameworks contribuem mais significativamente para o aumento da produtividade no desenvolvimento de software [SRP 00]. As figuras abaixo ilustram esta diferença (a área sombreada representa as classes reutilizadas).

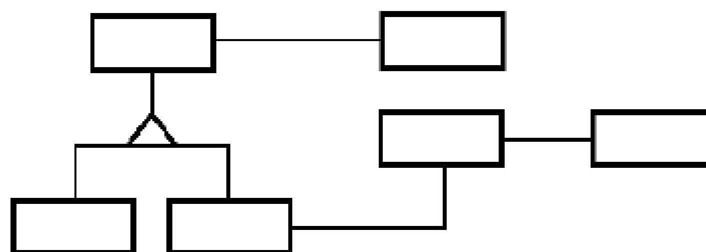

Figura 2.1 - Aplicação desenvolvida totalmente

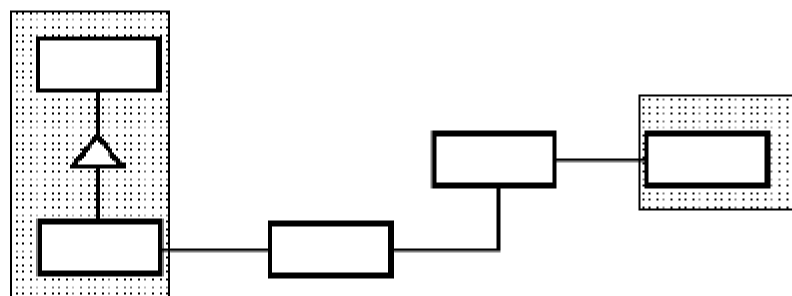

Figura 2.2 - Aplicação desenvolvida reutilizando classes de biblioteca

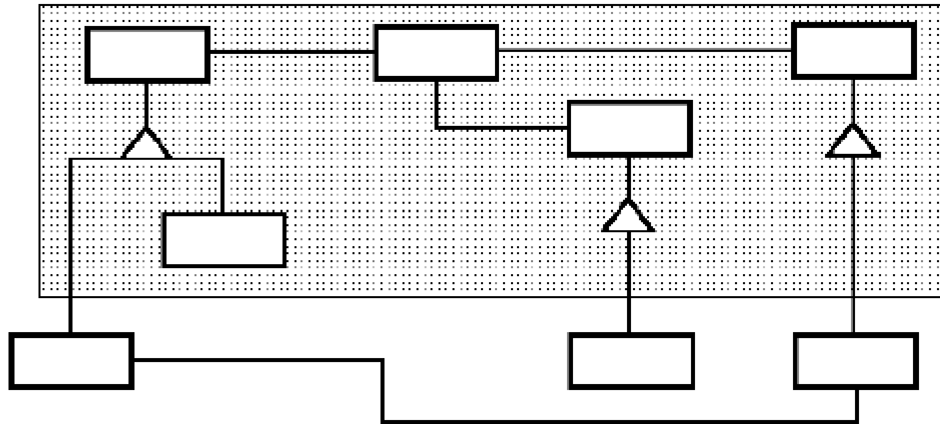

Figura 2.3 - Aplicação desenvolvida reutilizando um framework

A reutilização neste nível leva a uma inversão de controle entre a aplicação e o software sobre a qual ela está baseada. Quando se utiliza uma biblioteca de funções, se escreve o corpo principal da aplicação e se chama o código que se quer reutilizar. Quando se usa um framework, se reutiliza o corpo principal e escreve-se o código que este chama. O desenvolvedor terá de escrever operações com nomes e convenções de chamadas já especificadas; porém isto reduz as decisões de projeto que o mesmo tem que tomar [GAM 02].

Como resultado o desenvolvedor pode não só construir aplicações mais rapidamente, como também as aplicações têm estruturas similares. Elas são mais fáceis de manter e parecem mais consistentes para seus usuários. Por outro lado, o desenvolvedor perde alguma liberdade criativa, uma vez que muitas decisões de projetos já foram tomadas por ele.

## 2.2 Classificação

Segundo Jonhson [JOH 92], de acordo com a forma de utilização, os frameworks são classificados em três tipos: caixa-branca, caixa-preta e caixa-cinza.

### Frameworks Caixa Branca (White-Box)

Baseiam-se nos mecanismos de herança e ligação dinâmica (*dynamic binding*) presentes em orientação a objetos. Os recursos existentes em um framework caixa

branca são reutilizados e estendidos a partir da herança das classes do framework e sobrecarga (*overriding*) de métodos pré-definidos. O termo caixa branca se refere à visibilidade: com herança, os detalhes internos das classes ancestrais são freqüentemente visíveis para as subclasses.

Esta forma de implementação é descrita pelo padrão de projeto *Template Method*. Neste padrão, um método é redefinido em uma subclasse, que irá modificar o comportamento do método herdado. Este método redefinido é chamado de *template method.*

Frameworks caixa branca são mais fáceis de implementar, porém mais difíceis de usar, pois a geração de subclasses a partir da herança das classes do framework exige um conhecimento de sua estrutura e dos métodos a serem implementados para que as subclasses criadas funcionem corretamente.

**Frameworks Caixa Preta (Black-Box)**

São baseados em componentes de software. A extensão da arquitetura é feita a partir de interfaces definidas para componentes. Os recursos existentes são reutilizados e estendidos por meio da definição de um componente adequado a uma interface específica e sua integração ao framework. O termo caixa preta, assim como o caixa branca, refere-se à visibilidade: os detalhes internos dos componentes não são visíveis.

Esta forma de implementação é descrita pelo padrão de projeto *Strategy*. Este padrão define uma família de algoritmos, encapsula cada uma delas e as torna intercambiáveis.

Frameworks caixa preta são mais fáceis de usar, pois não há herança envolvida, apenas objetos, e estes sendo mais concretos que classes, são mais fáceis de serem compreendidos, porém, frameworks caixa preta são menos flexíveis que os frameworks caixa branca, pois o aplicativo é construído a partir de um conjunto de componentes interligados e não pela derivação de classes, que permite ao desenvolvedor redefinir as operações e até a funcionalidade da classe herdada.

**Frameworks Caixa-Cinza**

É combinação dos dois casos, buscando aproveitar a flexibilidade do framework caixa-branca e a facilidade de compreensão e uso do framework caixa-preta. Na prática, a maioria dos frameworks desenvolvidos pertence a esta classificação.

## 2.3 Qualidades de um Bom Framework

Para que um framework ofereça um bom suporte ao domínio de aplicações a que se propõe, deve buscar implementar algumas características que irão contribuir em muito para o aumento da qualidade do framework, entre elas, podemos citar como principais a generalidade, alterabilidade e extensibilidade. Para que isto ocorra, o projeto do framework deve ser bem elaborado, buscando identificar que partes devem ser mantidas flexíveis para produzir um projeto bem estruturado. Desta forma, deve-se utilizar os princípios de um projeto orientado a objetos, como o uso da herança para reutilização de interface ao invés de reutilização de código e o uso do polimorfismo na definição das classes e métodos. Suas classes abstratas devem estar no topo da hierarquia de classes, pois a finalidade destas classes é definir as interfaces a serem herdadas pelas classes concretas das aplicações.

Pode-se com isto afirmar que o desenvolvimento de um framework é mais complexo que o desenvolvimento de uma aplicação específica. A seguir, uma breve explanação sobre as principais qualidades que um bom framework deve possuir.

**Generalidade**

Reflete a capacidade do framework em dar suporte a várias aplicações diferentes de um mesmo domínio, sendo flexível o suficiente para que as características de alterabilidade e extensibilidade possam ser aplicadas.

**Alterabilidade**

Reflete a capacidade do framework de alterar suas funcionalidades em função da necessidade de uma aplicação específica sem que estas alterações resultem em conseqüências imprevistas no conjunto de sua estrutura.

**Extensibilidade**

Reflete a capacidade do framework de ampliar sua funcionalidade sem conseqüências imprevistas no conjunto de sua estrutura. Ligada diretamente à manutenibilidade do framework, permite que sua estrutura evolua por toda sua vida útil, pois à medida em que vai sendo utilizado, novos recursos vão sendo agregados para que o mesmo se ajuste as novas aplicações a que dá suporte.

**Simplicidade**

A estrutura geral do framework deve ser de fácil compreensão de forma que o desenvolvedor possa aprendê-lo em pouco tempo. Obviamente todos os detalhes do projeto não poderão ser aprendidos em poucos dias, porém o desenvolvedor deve estar apto para entender seu funcionamento em pouco tempo, deixando o aprendizado de seus detalhes para o decorrer de sua utilização.

Esta simplicidade é alcançada pelo projeto de interfaces limpas e consistentes, pela utilização de padrões em seu projeto, código, interfaces e nomenclaturas. Todos os objetos pertencentes ao mesmo domínio devem herdar de uma interface comum, de modo que em se conhecendo os métodos e atributos desta interface, bem como seu design e funcionamento, o aprendizado de suas subclasses é facilitado.

**Clareza**

Os aspectos comportamentais do framework devem estar encapsulados. Não há necessidade do desenvolvedor saber todos os detalhes de como o framework faz alguma coisa para que ele possa utilizá-lo. A interface pública das classes do framework deve ser tão simples quanto possível. Visto que são nas interfaces públicas que o desenvolvedor estará trabalhando, é importante mantê-las não mais complexas que o necessário para que se possa alcançar a funcionalidade desejada.

O desenvolvedor não deveria ter que seqüencialmente chamar três ou quatro métodos no framework para acompanhar um certo processo, a menos que o processo seja utilizado muitas vezes na parte.

**Fronteiras**

Um framework tem responsabilidades claras e sucintas, e deve acatá-las e nada mais. Todas as funcionalidades exteriores a fronteira do framework devem ser tratadas pelo desenvolvedor. Quando um framework ultrapassa esta fronteira, ele se torna complexo e provavelmente o desenvolvedor ao tentar utiliza-lo precisará implementar código adicional junto ao framework para conseguir o comportamento desejado. Um framework não provê a funcionalidade da aplicação, ele provê o esqueleto sobre o qual a aplicação é construída. Sua funcionalidade é responsabilidade do desenvolvedor que o utiliza. Caso o framework deseje prover classes mais especializadas, estas devem ser fornecidas em bibliotecas de classes separadas como subclasses das classes do framework. Isto permitirá ao desenvolvedor escolher usar ou não estas classes, fazendo assim uma clara distinção entre o framework e o kit de ferramentas do desenvolvedor.

**Ganchos**

Um caminho para prover capacidade de expansão das funcionalidades do framework é oferecer ao desenvolvedor métodos pelos quais ele possa escrever código que afetará o comportamento do framework. Por exemplo, se o framework possui um formulário de cadastro qualquer, que possui um método Salvar() para gravar as alterações feitas pelo usuário, o framework poderia dispor de um método AntesDeSalvar() e DepoisDeSalvar(), vinculados ao método Salvar(). Estes métodos podem prover ao desenvolvedor locais para escrever código antes e depois das edições do usuário serem salvas. Se o método Salvar() respeitar o valor retornado pelo método AntesDeSalvar(), o desenvolvedor pode decidir, por exemplo, cancelar a operação se alguma condição não for respeitada, sem precisar implementar código adicional na classe do framework para prover esta funcionalidade.

## 2.4 Ciclo de Vida de Frameworks

O ciclo de vida de um framework difere do ciclo de vida de uma aplicação convencional porque um framework nunca é um artefato de software isolado, mas sua existência está sempre relacionada à existência de outros artefatos de software

originadores do framework, originados a partir dele ou que exerçam alguma influência na definição da estrutura de classes do framework [SRP 00]. Segundo Ricardo P. Silva, existem três fontes de informação que influenciam na definição da estrutura de um framework: artefatos de software existentes, artefatos de software produzidos a partir do framework e conhecimento do desenvolvedor do framework (ou da equipe de desenvolvimento). A figura 2.4 ilustra estas três fontes de influência. As setas representam o fluxo da informação que levam à produção da estrutura de classes do framework bem como de aplicações sob o framework.

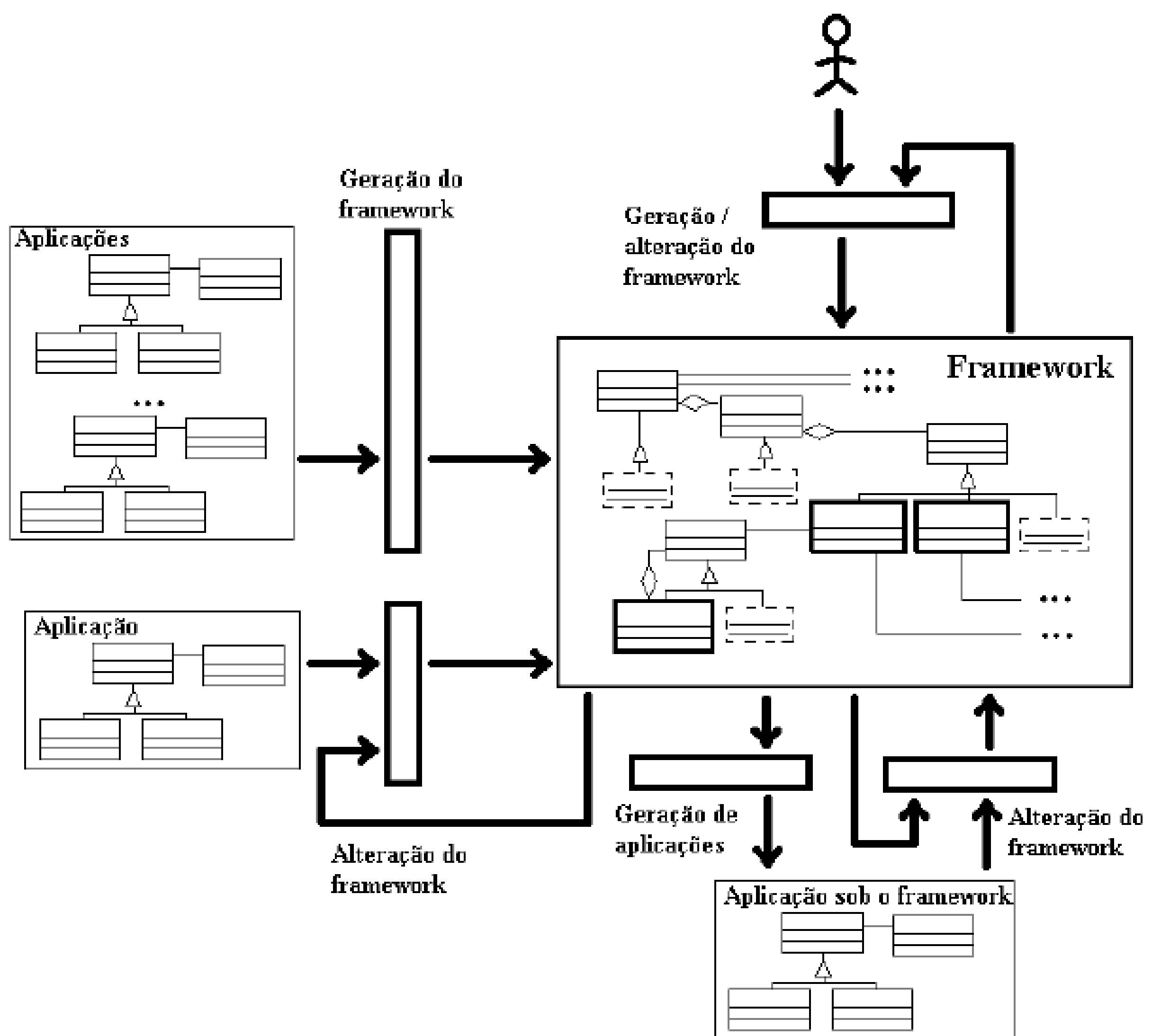


Figura 2.4 – Ciclo de vida de frameworks

**A atuação do desenvolvedor:** nenhuma abordagem de framework dispensa a figura do desenvolvedor. Ele é o responsável por decidir que classes comporão a

estrutura do framework, suas responsabilidades e a flexibilidade provida aos usuários do framework. O desenvolvedor atua não somente na construção do framework, mas também na sua manutenção. A figura 2.5 destaca a atuação do desenvolvedor no ciclo de vida de um framework.

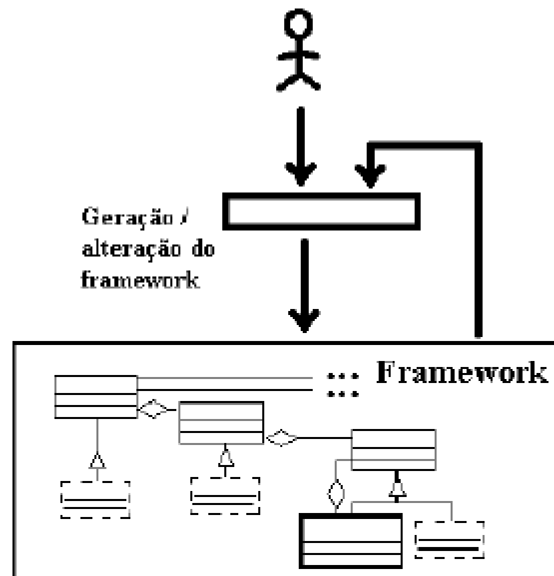


Figura 2.5 – Destaque da atuação do desenvolvedor no ciclo de vida de frameworks

**Aplicações do domínio tratado:** um framework constitui um modelo de um domínio de aplicações. Assim, pode ser desenvolvido a partir de um conjunto de aplicações do domínio, que atuam como fonte de informação deste domínio. Esta influência de aplicações do domínio pode ocorrer durante o processo de desenvolvimento do framework ou na fase de manutenção. Neste último caso, a alteração seria motivada pela obtenção de novos conhecimentos do domínio tratado, não considerados ou não disponíveis durante o desenvolvimento do framework.

A figura 2.6 ilustra um procedimento em que um conjunto de aplicações leva à produção de um framework e um procedimento em que uma aplicação leva à alteração na estrutura do framework. Veja que a figura do desenvolvedor não está presente na figura, o que faz pensar que sejam procedimentos automatizados, porém, fica subentendido que ele é o responsável pela avaliação e desenvolvimento de todo o processo.

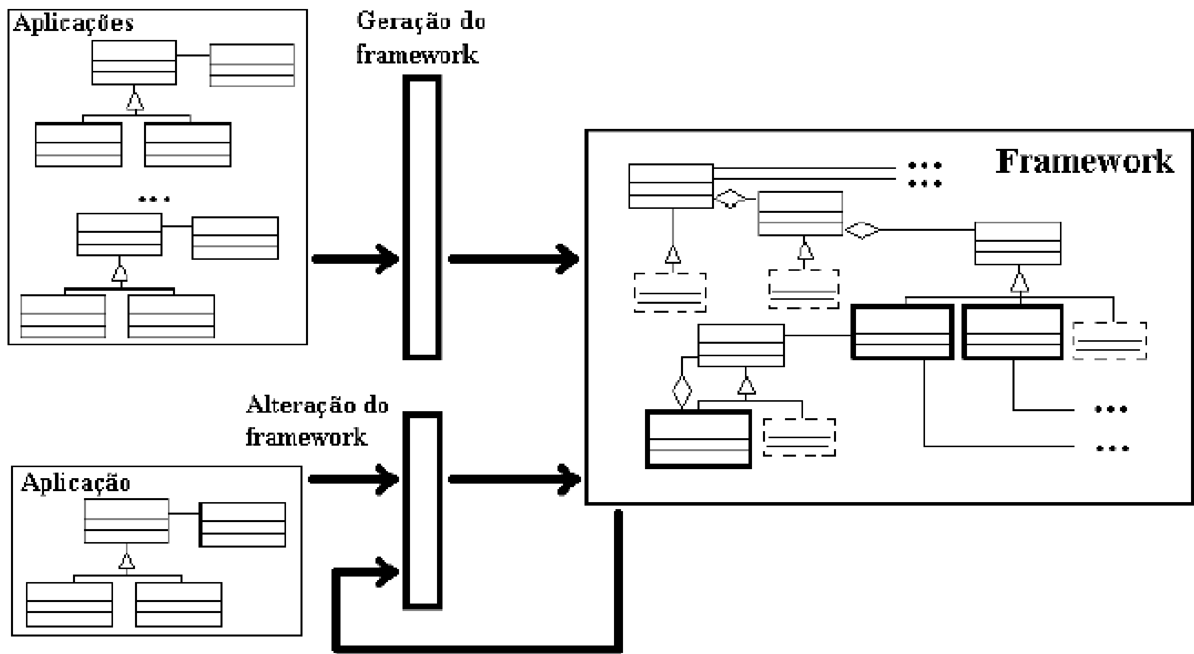


Figura 2.6 – Destaque da influência de aplicações existentes na geração e na alteração de um framework.

**Aplicações geradas sob o framework:** a finalidade básica de um framework é ser reutilizado na produção de diferentes aplicações, minimizando o tempo e o esforço requeridos para isto. A construção de um framework é sempre precedida por uma análise de domínio, em que são buscadas informações do domínio tratado. Como uma abstração de uma realidade tratada, é inevitável que o framework seja incapaz de conter todas as informações do domínio, o que ocasiona que aplicações construídas sob um framework levam a obtenção de novos conhecimentos sob o domínio tratado, indisponíveis durante a construção do framework. Estas novas informações podem levar a necessidade de se alterar o framework para que ele dê suporte a elas. A figura 2.7 ilustra a influência de aplicações geradas sob um framework na alteração de sua estrutura original. Conforme já mencionado, fica subentendido que o responsável pela análise e alteração do framework é o desenvolvedor.

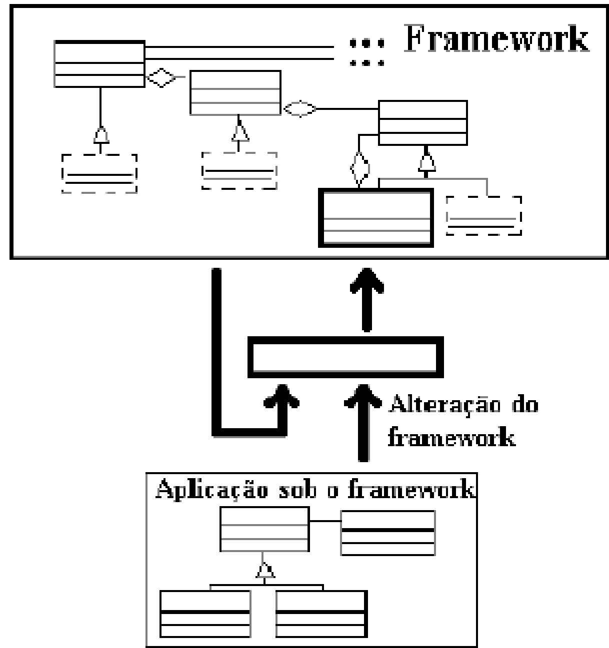


Figura 2.7 – Destaque influência de aplicações geradas a partir de um framework na alteração deste framework

## 2.5 Indivíduos Envolvidos com o Desenvolvimento e o Uso de Frameworks

O desenvolvimento tradicional de aplicações envolve dois tipos de indivíduos: desenvolvedor de aplicação e usuário de aplicação. Desenvolvedores devem levantar os requisitos de uma aplicação, desenvolvê-la (o que inclui a documentação que ensina a usar a aplicação, com manuais de usuário) e entregá-la aos usuários. Usuários interagem com uma aplicação através de sua interface [SRP 00]. A figura 2.8 apresenta os indivíduos envolvidos neste caso:

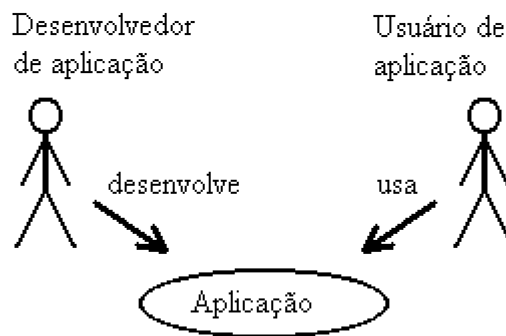


Figura 2.8 – Indivíduos envolvidos no desenvolvimento tradicional de aplicações

O desenvolvimento de frameworks introduz um outro indivíduo, além do desenvolvedor e usuário de aplicação: o desenvolvedor do framework. No contexto dos frameworks, o papel do usuário é o mesmo descrito acima. O papel do desenvolvedor de aplicações difere no caso anterior pela inserção do framework no processo de desenvolvimento de aplicações. Com isto, o desenvolvedor de aplicação é um usuário do framework, que deve estender e adaptar a estrutura deste framework para a produção de aplicações. Ele tem as mesmas funções do caso anterior: obter os requisitos da aplicação, desenvolvê-la usando o framework e desenvolver a documentação da aplicação. O novo papel criado no contexto dos frameworks, o desenvolvedor do framework, tem a responsabilidade de produzir frameworks e algum modo de ensinar como usá-los para produzir aplicações. A figura 2.9 apresenta os indivíduos envolvidos neste último caso.

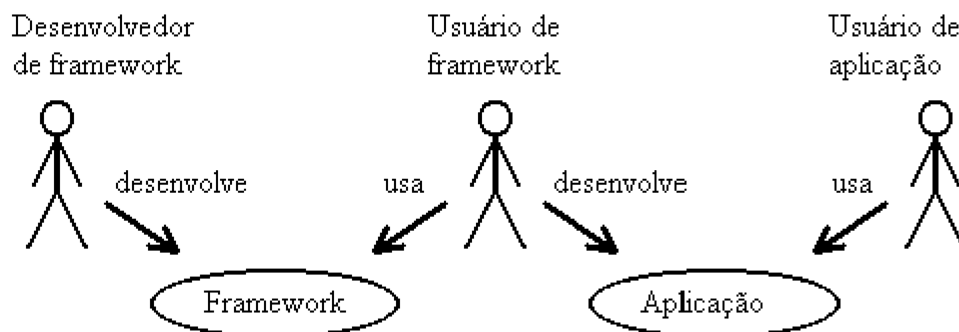


Figura 2.9 – Indivíduos envolvidos no desenvolvimento aplicações baseadas em frameworks

# Capítulo 3

# Padrões de Projeto

## 3.1 Introdução

Padrões (*Patterns*) constituem uma abordagem recente em termos de reutilização de projeto, no contexto do desenvolvimento de softwares orientados a objetos. A principal questão tratada é como proceder para registrar, para poder reutilizar em um desenvolvimento de software, a experiência de projeto adquirida em desenvolvimentos anteriores [SRP 00].

Os Padrões de Projeto foram originados a partir da observação de que diferentes partes do framework possuíam uma estrutura de classes semelhante. Isto demonstrou a existência de padrões de solução para problemas de projeto semelhantes, e que se repetiam à medida em que se procurava produzir uma estrutura flexível (para extensão ou adaptação) [SRP 00].

Padrões de Projeto descrevem soluções simples para problemas específicos no projeto de software orientado a objetos, capturando soluções que foram desenvolvidas e aperfeiçoadas ao longo do tempo. Ele reflete modelagens e recodificações, nunca relatadas, resultantes dos esforços dos desenvolvedores por maior reutilização e flexibilidade em seus sistemas de software. Padrões de Projeto capturam estas soluções em uma forma sucinta e facilmente aplicável. Uma vez que o tenha compreendido, você terá percepções que tornarão seus projetos mais flexíveis, modulares, reutilizáveis e compreensíveis [GAM 02].

Todas as arquiteturas orientadas a objetos bem-estruturadas estão cheias de padrões. De fato, uma das maneiras de medir a qualidade de um sistema orientado a objetos é avaliar se os desenvolvedores tomaram bastante cuidado com as colaborações comuns entre seus objetos. Focalizar-se em tais mecanismos durante o desenvolvimento de um sistema pode levar a uma arquitetura menor, mais simples e muito mais compreensível que aquelas produzidas quando estes padrões são ignorados [GAM 02].

Os padrões de projeto tornam mais fácil a reutilização de projetos e arquiteturas bem-sucedidas, pois descrevem técnicas já testadas e aprovadas. Eles ajudam a

escolher alternativas de projeto, evitando alternativas que poderiam comprometer a reutilização. Padrões de projeto podem melhorar a documentação e a manutenção do sistema, bem como fornecer uma especificação explícita de interações de classes e objetos e o seu objetivo subjacente. Em suma, ajudam um projetista a obter um projeto "certo" mais rápido [GAM 02].

A importância de padrões na criação de sistemas complexos foi há muito tempo reconhecida em outras disciplinas. Chistopher Alexander e seus colegas talvez foram os primeiros a propor a idéia de utilizar uma linguagem de padrões em projetos de edificações e cidades. Suas idéias, e as contribuições de outros, assentaram raízes na comunidade de software orientado a objetos.

Para fixar melhor a idéia, podemos fazer um relacionamento com uma novela ou peça de teatro. Os autores dos roteiros raramente projetam suas tramas do zero, ao invés disso, eles seguem padrões como "O herói tragicamente problemático" (Macbeth, Hamlet, etc) ou "A Novela Romântica" (uma centena de novelas e romances). Do mesmo modo, projetistas de software orientado a objetos seguem padrões, tal como "represente estados com objetos". Uma vez que se conhece o padrão, uma grande quantidade de decisões de projeto decorre automaticamente.

## 3.2 Definição

Chistopher Alexander afirma: "cada padrão descreve um problema no nosso ambiente e o núcleo da sua solução, de tal forma que você possa usar esta solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira". Embora Alexander estivesse falando sobre padrões em construções e cidades, o que diz pode ser utilizado em padrões de projeto orientado a objetos. Nossas soluções são expressas em termos de objetos e interfaces ao invés de paredes e portas, mas no cerne de ambos os tipos de padrões está a solução para um problema num contexto.

Segundo Gamma [GAM 02], um padrão de projeto tem quatro elementos essenciais:

1. **O Nome do padrão:** é uma referência que podemos usar para descrever um problema de projeto, suas soluções e conseqüências em uma ou duas palavras. Dar nome a um padrão é muito importante, pois aumenta o

vocabulário do projeto, permitindo a conversação com nossos colegas, em nossa documentação e até com nós mesmos, tornando mais fácil pensar sobre projetos e comunicá-los a outras pessoas.

2. **O Problema:** descreve quando aplicar o padrão. Ele explica o problema e o seu contexto, podendo descrever problemas de projeto específicos, estruturas de classes ou objeto sintomáticas de um projeto inflexível. Para facilitar o entendimento de sua aplicabilidade, o problema pode conter uma lista de condições que devem ser satisfeitas para que se faça sentido aplicar o padrão.
3. **Solução:** descreve os elementos que compõem o projeto, seus relacionamentos, suas responsabilidades e colaborações. A solução não descreve um projeto concreto ou uma implementação em particular, mas sim uma descrição abstrata de um problema de projeto e de como um arranjo geral de elementos (classes e objetos no nosso caso) resolve o problema.
4. **Conseqüências:** são os resultados e análises das vantagens e desvantagens da aplicação do padrão. Embora não sejam mencionadas na descrição das decisões de projeto, elas são críticas para a avaliação das alternativas de projeto e para a compreensão dos custos e benefícios da aplicação do padrão. As conseqüências de um padrão incluem o seu impacto sobre a flexibilidade, extensibilidade ou portabilidade de um sistema.

## 3.3 Como padrões solucionam problemas de projeto

Os padrões de projeto solucionam muitos dos problemas que os projetistas enfrentam no dia a dia, e de muitas maneiras diferentes. [GAM 02] apresenta vários destes problemas e como os padrões de projeto os solucionam. A seguir, será explicado como padrões de projetos solucionam alguns problemas de projeto bem como será dada uma introdução aos principais conceitos da orientação a objeto para que se possa entender melhor como os padrões de projetos atuam nas soluções dos problemas apontados.

### 3.3.1 Procurando objetos apropriados

Programas orientados a objetos são feitos de objetos. Um objeto empacota tanto os dados quanto os procedimentos que operam sobre esses dados. Os procedimentos são tipicamente chamados de métodos ou operações. Um objeto executa uma operação quando ele recebe uma solicitação (ou mensagem) de um cliente.

As solicitações (*requests*) são a única maneira de mudar os dados internos de um objeto. Por causa destas restrições, diz-se que o estado interno de um objeto está encapsulado; ele não pode ser acessado diretamente e sua representação é invisível do exterior do objeto.

A parte difícil sobre projeto orientado a objetos é a decomposição de um sistema em objetos. A tarefa é difícil porque muitos fatores entram em jogo: encapsulamento, granularidade, dependência, flexibilidade, desempenho, evolução, reutilização, e assim por diante. Todos influenciam a decomposição, freqüentemente de forma conflitantes.

As metodologias de projeto orientado a objetos favorecem muitas abordagens diferentes. Você pode escrever uma descrição de um problema, separar os substantivos e verbos e criar as classes e operações correspondentes. Ou você pode se concentrar sobre as colaborações e responsabilidades no seu sistema. Ou ainda, poderá modelar o mundo real e, na fase de projeto, traduzir os objetos encontrados durante a análise. Sempre haverá desacordo sobre qual é a melhor abordagem.

Muitos objetos num projeto provêm do modelo de análise. Porém, projetos orientados a objetos freqüentemente acabam tendo classes que não têm contrapartida no mundo real. Algumas dessas de baixo nível, como vetores. Outras estão em um nível muito mais alto. A modelagem estrita do mundo real condiz a um sistema que reflete as realidades atuais, mas não necessariamente as futuras. As abstrações que surgem durante um projeto são as chaves para tornar um projeto flexível.

Os padrões de projeto ajudam a identificar abstrações menos óbvias bem como os objetos que podem capturá-las. Por exemplo, objetos que representam um processo ou algoritmo não ocorrem na natureza, no entanto, eles são uma parte

crucial de projetos flexíveis. Estes objetos são raramente encontrados durante a análise, ou mesmo durante os estágios iniciais de um projeto; eles são descobertos mais tarde, durante o processo de tornar um projeto mais flexível e reutilizável.

### 3.3.2 Determinando a granularidade dos objetos

Os objetos podem variar tremendamente em tamanho e número. Podem representar qualquer coisa indo para baixo, até o nível do hardware, ou seguindo todo o caminha para cima, até chegarmos a aplicações inteiras. Como decidimos o que deve ser um objeto?

O padrão *Fachada*, por exemplo, descreve como representar subsistemas completos como objetos e o padrão *Flyweigth* descreve como suportar enormes quantidades de objetos nos níveis de granularidade mais finos. Outros padrões descrevem maneiras específicas de decompor um objeto em objetos menores.

### 3.3.3 Especificando interfaces de objetos

Cada operação declarada por um objeto especifica o nome da operação, os objetos que ela aceita como parâmetros e o valor retornado por ela. Isto é conhecido como *assinatura* da operação. O conjunto de todas as assinaturas definido pelas operações de um objeto é chamado de *interface do objeto*. A interface de um objeto caracteriza o conjunto completo de solicitações que podem ser enviadas para o mesmo. Qualquer solicitação que corresponde a uma assinatura na interface do objeto pode ser enviada para o mesmo.

Um *tipo* é um nome usado para denotar uma interface específica. Quando dizemos que um objeto tem um tipo “Janela”, significa que ele aceita todas as solicitações para as operações definidas na interface chamada “Janela”. Um objeto pode ter muitos tipos, assim como objetos muito diferentes podem compartilhar um mesmo tipo. Parte da interface de um objeto pode ser caracterizada por um tipo, e outras partes por outros tipos. Dois objetos do mesmo tipo necessitam compartilhar somente partes de suas interfaces. As interfaces podem conter outras interfaces como subconjuntos. Dizemos que um tipo é um *subtipo* de outro se sua interface

contém a interface do seu *supertipo*. Freqüentemente dizemos que um subtipo *herda* a interface do seu supertipo.

As interfaces são fundamentais em sistemas orientados a objetos. Os objetos são conhecidos somente através das suas interfaces. Não existe nenhuma maneira de saber algo sobre um objeto ou de pedir que faça algo sem intermédio de sua interface. A interface de um objeto nada diz sobre sua implementação – diferentes objetos estão livres para implementar as solicitações de diferentes maneiras. Isso significa que dois objetos que tenham implementações completamente diferentes podem ter interfaces idênticas.

Quando uma mensagem é enviada a um objeto, a operação específica que será executada depende de ambos – mensagem e objeto receptor. Diferentes objetos, que suportam solicitações idênticas, podem ter diferentes implementações das operações que atendem a estas solicitações. A associação em tempo de execução de uma solicitação a um objeto e a uma das suas operações é conhecida como ligação dinâmica (*dynamic binding*).

A ligação dinâmica significa que o envio de uma solicitação não prenderá você a uma particular implementação até o tempo de execução. Conseqüentemente, você poderá escrever programas que esperam um objeto com uma interface em particular, sabendo que qualquer objeto que tenha a interface correta aceitará a solicitação. Além do mais, a ligação dinâmica permite substituir objetos que tenham interfaces idênticas uns pelos outros. Esta capacidade de substituição é conhecida como *polimorfismo* e é um conceito-chave em sistemas orientados a objetos. Ela permite a um objeto-cliente criar poucas hipóteses sobre outros objetos, exceto que eles suportam uma interface específica. Polimorfismo simplifica as definições dos clientes, desacopla objetos entre si e permite a eles variarem seus inter-relaciona-mentos em tempo de execução.

Os padrões de projeto ajudam a definir interfaces pela identificação de seus elementos-chave e pelos tipos de dados que são enviados através de uma interface. Um padrão de projeto também pode lhe dizer o que não colocar na interface. Por exemplo, o padrão *Memento* descreve como encapsular e salvar o estado interno de um objeto de modo que o objeto possa ser restaurado àquele estado mais tarde.

Os padrões de projeto também especificam relacionamentos entre interfaces. Em particular, freqüentemente exigem que algumas classes tenham interfaces similares, ou colocam restrições sobre interfaces de algumas classes.

### 3.3.4 Especificando implementações de objetos

Uma implementação de um objeto é definida por sua *classe*. A classe especifica os dados internos do objeto e de sua representação e define as operações que o objeto pode executar.

Objetos são criados por *instanciação* de uma classe. Diz-se que o objeto é uma *instância* da classe. O processo de instanciar uma classe aloca memória para os dados internos do objeto (compostos de *variáveis de instância*) e associa as operações a estes dados. Muitas instâncias semelhantes de um objeto podem ser criadas pela instanciação de uma classe.

Novas classes podem ser definidas em termos das classes existentes usando-se *herança de classe*. Quando uma *subclasse* herda de uma *classe-mãe*, ela inclui as definições de todos os dados e operações que a classe-mãe define. Os objetos que são instâncias das classes conterão todos os dados definidos pela subclasse e suas classes-mãe. Eles serão capazes de executar todas as operações definidas por esta subclasse e seus “ancestrais”.

Uma *classe abstrata* é uma classe cuja finalidade principal é definir uma interface comum para suas subclasses. Uma classe abstrata postergará parte de, ou toda, sua implementação para operações definidas em subclasses, portanto, uma classe abstrata não pode ser instanciada. As operações que uma classe abstrata declara, mas não implementa são chamadas *operações abstratas*. As classes que não são abstratas são chamadas de *classes concretas*.

As subclasses podem refinar e redefinir comportamentos de suas classes ancestrais (mãe, avó, etc.). Mais especificamente, uma classe pode *redefinir* uma operação definida por sua classe-mãe. A redefinição dá as subclasses a oportunidade de tratar solicitações em lugar das suas classes ancestrais. A herança de classe permite definir classes simplesmente estendendo outras classes, tornando fácil definir famílias de objetos que têm funcionalidades relacionadas.

Aqui é importante compreender a diferença entre herança de classe e herança de interface. A herança de classe define a implementação de um objeto em termos da implementação de outro objeto. Resumidamente, é um mecanismo para compartilhamento de código e de representação. Diferentemente disso, a herança de interface descreve quando um objeto pode ser usado no lugar de outro.

#### 3.3.4.1 Programando para uma interface, não para uma implementação

A herança de classe é basicamente um mecanismo para estender a funcionalidade de uma aplicação pela reutilização da funcionalidade das classes ancestrais, permitindo definir rapidamente um novo tipo de objeto em termos de um existente. Ela permite obter novas implementações quase de graça, herdando a maior parte do que você necessita de classes existentes.

Contudo, a habilidade da herança para definir famílias de objetos com interfaces idênticas (usualmente por herança de uma classe abstrata) também é importante, pois quando a herança é usada apropriadamente, todas as classes derivadas de uma classe abstrata compartilharão sua interface. Isto implica que uma subclasse meramente acrescenta ou substitui operações da classe mãe, e não oculta operações dela. Todas as subclasses podem então responder a solicitações na interface desta classe abstrata, tornando-se, todas, subtipos da classe abstrata. Isto reduz grandemente as dependências de implementação entre subsistemas, favorecendo a reutilização.

Baseado no que foi dito acima, não declare variáveis como instâncias de classes concretas específicas. Em vez disso, prenda-se somente a uma interface definida por uma classe abstrata. Naturalmente, você tem que instanciar classes concretas, isto é, especificar uma particular implementação em algum lugar do sistema, e os padrões de projeto de criação permitem fazer exatamente isto, como por exemplo, os padrões Abstract Factoty, Builder e Singleton. Ao abstrair o processo de criação de objetos, estes padrões oferecem diferentes maneiras de associar uma interface com sua implementação de forma transparente no momento da instanciação. Os padrões de criação asseguram que seu sistema esteja escrito em termos de interfaces, não de implementações.

#### 3.3.4.2 Herança versus composição

As duas técnicas mais comuns para a reutilização de funcionalidades em sistemas orientados a objetos são a herança de classe e a composição de objetos. A reutilização por meio de subclasses é freqüentemente chamada de reutilização caixa branca (ou aberta), porque, com herança, os detalhes internos das classes ancestrais são freqüentemente visíveis para as subclasses.

A composição de objetos é uma alternativa à herança de classe. Aqui, a nova funcionalidade é obtida pela montagem e/ou composição de objetos, para obter funcionalidades mais complexas. A composição de objetos requer que os objetos que estão sendo compostos tenham interfaces bem definidas. Este tipo de reutilização é chamado de reutilização caixa preta, porque os detalhes internos dos objetos não são visíveis.

A herança e a composição têm, cada uma, vantagens e desvantagens. A herança de classe é definida estaticamente em tempo de compilação e é simples de se usar, uma vez que é suportada diretamente pela linguagem de programação. A herança de classe também torna mais fácil modificar a implementação que está sendo reutilizada. Porém, a herança de classe tem algumas desvantagens. Em primeiro lugar, você não pode mudar as implementações herdadas das classes ancestrais em tempo de execução, porque a herança é definida em tempo de compilação. Em segundo lugar, e geralmente isso é o pior, as classes ancestrais freqüentemente definem pelo menos parte da representação física das suas subclasses. Porque a herança expõe para uma subclasse os detalhes da implementação dos seus ancestrais, freqüentemente é dito que a "herança viola o encapsulamento" [GAM 02 citando SNY 86]. A implementação de uma subclasse, desta forma, torna-se tão amarrada à implementação da sua classe-mãe que qualquer mudança desta forçará uma mudança naquela.

As dependências de implementação podem causar problemas quando se está tentando reutilizar uma subclasse. Se algum aspecto da implementação herdada não for apropriado para novos domínios de problemas, a classe-mãe deve ser reescrita ou substituída por algo mais apropriado. Esta dependência limita a flexibilidade e, em última instância, a reusabilidade. Uma cura para isto é herdar somente de classes

abstratas, uma vez que elas normalmente fornecem pouca ou nenhuma implementação.

A composição de objetos é definida dinamicamente em tempo de execução pela obtenção de referências para outros objetos por um determinado objeto. A composição requer que os objetos respeitem as interfaces uns dos outros, o que por sua vez exige interfaces cuidadosamente projetadas, que não impeçam de você usar um objeto com muitos outros. Porém, existe um ganho. Como objetos são acessados exclusivamente através de suas interfaces, o encapsulamento não é violado. Qualquer objeto pode ser substituído por outro em tempo de execução, contanto que tenha o mesmo tipo. Além do mais, porque a implementação de um objeto será escrita em termos de interfaces de objetos, existem substancialmente menos dependências de implementação.

Favorecer a composição de objetos em relação à herança de classes, ajuda a manter cada classe encapsulada e focalizada em uma única tarefa. Suas classes e hierarquias de classes se manterão pequenas, com menor probabilidade de crescerem até se tornarem monstros intratáveis. Por outro lado, o projeto terá mais objetos (embora menos classes) e o comportamento do sistema dependerá de seus inter-relacionamentos ao invés de ser definido em uma classe.

Idealmente, você não deveria ter que criar novos componentes para obter reutilização. Deveria ser capaz de conseguir toda a funcionalidade de que necessita simplesmente montando componentes existentes através da composição de objetos. Mas este raramente é o caso, porque o conjunto de componentes disponíveis nunca é exatamente rico o bastante na prática. A reutilização por herança torna mais fácil criar novos componentes que podem ser obtidos pela composição de componentes existentes. Assim, a herança e a composição de objetos trabalham juntas.

A composição de objetos é aplicada repetidas vezes nos padrões de projeto.

#### 3.3.4.3 Delegação

A delegação é uma maneira de tornar a composição de objetos tão poderosa para fins de reutilização como a herança [GAM 02 refereciando LIE 86, JZ 91]. Na delegação, dois objetos são envolvidos no tratamento de uma solicitação: um objeto *receptor* delega operações para o seu *delegado*.

Para exemplificar, ao invés de fazer uma classe Janela uma subclasse de Retangulo (porque janelas são retangulares), a classe Janela deve reutilizar o comportamento de Retangulo, conservando uma variável de instância de Retangulo, e *delegando* o comportamento específico de Retangulo para ela. Em outras palavras, ao invés da classe Janela *ser* um Retangulo ela *teria* um Retangulo. Agora, Janela deve encaminhar as solicitações para a sua instância Retangulo explicitamente, ao passo que antes teria herdado essas operações.

O diagrama da figura 3.1 ilustra a classe Janela delegando sua operação área a uma instância de Retangulo.

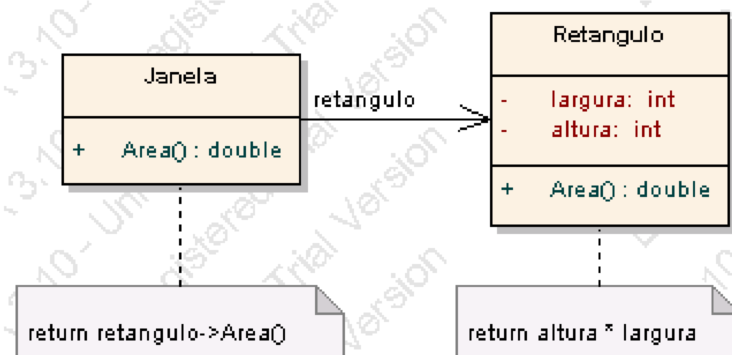


Figura 3.1 – Exemplo de delegação

A principal vantagem da delegação é que ela torna fácil compor comportamentos em tempo de execução e mudar a forma como são compostos. A classe Janela pode ter o formato circular em tempo de execução, simplesmente pela substituição da sua instância Retangulo por uma instância de Circulo, assumindo-se que Retangulo e Circulo tenham o mesmo tipo.

A delegação tem uma desvantagem que compartilha com outras técnicas que tornam o software mais flexível através da composição de objetos: o software dinâmico, altamente parametrizado, é mais difícil de compreender do que o software mais estático. A delegação é uma boa escolha de projeto somente quando ela simplifica mais do que complica e funciona melhor quando é usada baseada em padrões de projeto.

Diversos padrões de projeto usam delegação, como por exemplo, o padrão *Strategy*. Neste padrão um objeto delega uma solicitação específica para um objeto quer representa uma estratégia para executar a solicitação.

A delegação é um exemplo extremo da composição de objetos. Ela mostra se pode sempre substituir a herança pela composição de objetos como um mecanismo para a reutilização de código.

### 3.3.5 Relacionando estruturas de tempo de execução e compilação

Considere a distinção entre *agregação* e *associação* de objetos e como diferentemente elas se manifestam nos tempos de compilação e execução. A agregação implica que um objeto possui, ou é responsável por, outro objeto. Geralmente dizemos que um objeto *tem* ou *é parte de* outro objeto. Agregação implica que um objeto agregado e seu proprietário têm idênticos tempos de vida.

Associação implica que um objeto meramente *tem conhecimento* de outro objeto. Algumas vezes, é chamada de relacionamento ”usa”. Objetos que se conhecem podem solicitar operações uns dos outros, mas eles não são responsáveis um pelo outro. A associação é um relacionamento mais fraco do que a agregação e sugere um acoplamento muito menor entre objetos.

É fácil confundir agregação com associação, porque elas são freqüentemente implementadas da mesma forma. Em última instância, a associação e agregação são determinadas mais pela intenção do que por mecanismos explícitos da linguagem. A distinção pode ser difícil de ver na estrutura de tempo de compilação, porém, é significativa. Os relacionamentos de agregação tendem a ser em menor número e mais permanentes que as associações. Em contraste, as associações são feitas e refeitas mais freqüentemente, algumas vezes existindo somente para a duração de uma operação e são também mais dinâmicas, tornando-as mais difíceis de distinguir no código-fonte.

Com tal disparidade existente entre as estruturas de um programa em tempo de execução e tempo de compilação, é claro que o código não revelará tudo acerca de como o sistema funcionará. A estrutura de um sistema em tempo de execução deve ser imposta mais pelo projetista do que pela linguagem. Os relacionamentos entre objetos e seus tipos devem ser projetados com grande cuidado porque eles determinam quão boa ou ruim é a estrutura em tempo de execução.

Muitos padrões de projeto capturam explicitamente a distinção entre estruturas de tempo de compilação e execução. Por exemplo, o padrão *Observer* envolve

estruturas de tempo de execução que são freqüentemente difíceis de compreender, a menos que você conheça o padrão.

### 3.3.6 Projetando para mudanças

A chave para a maximização da reutilização está na antecipação de novos requisitos e mudanças nos requisitos existentes e em projetar sistemas de modo que eles possam evoluir de acordo.

Para projetar um sistema de maneira que seja robusto e em face de tais mudanças, você deve levar em conta como o sistema pode necessitar mudar ao longo de sua vida. Um projeto que não leva em consideração a possibilidade de mudanças está sujeito ao risco de uma grande reformulação no futuro, podendo envolver redefinições e reimplementações de classes. A reformulação afeta muitas partes de um sistema de software e, invariavelmente, mudanças não antecipadas são caras.

Os padrões de projeto ajudam a evitar isto ao garantirem que o sistema possa mudar segundo maneiras específicas. Cada padrão de projeto permite a algum aspecto da estrutura do sistema variar independentemente de outros aspectos, desta forma tornando um sistema mais robusto a um particular tipo de mudança.

Abaixo, serão apresentadas algumas causas comuns de reformulação de projeto:

1. *Criando um objeto pela especificação explícita de uma classe:* especificar um nome de uma classe quando você cria um objeto faz com que se comprometa com uma particular implementação, em vez de se comprometer com uma determinada interface. Este compromisso pode complicar futuras mudanças. Para evitá-los, crie objetos indiretamente.
2. *Dependências de operações específicas:* quando se especifica uma operação em particular, se compromete com uma maneira de atender uma solicitação. Evitando solicitações codificadas inflexivelmente, você torna mais fácil mudar a maneira como uma solicitação é atendida, tanto em tempo de compilação como em tempo de execução.

3. *Dependências de plataforma de hardware e software:* as interfaces externas do sistema operacional e as interfaces de programação de aplicações (APIs) são diferentes para diferentes plataformas de hardware e software. O software que depende de uma plataforma específica será mais difícil de portar para outras plataformas. Pode ser até mesmo difícil mantê-lo atualizado na sua plataforma nativa.
4. *Dependências de representações ou implementações de objetos:* clientes que sabem como um objeto é representado, armazenado, localizado ou implementado podem necessitar serem mudados quando o objeto muda. Ocultar essas informações dos clientes evita a propagação de mudanças em cadeia.
5. *Dependências algorítmicas:* os algoritmos são freqüentemente estendidos, otimizados e substituídos durante o desenvolvimento e reutilização. Os objetos que dependem de algoritmos terão que mudar quando o algoritmo mudar. Portanto os algoritmos que provavelmente mudarão deveriam ser isolados.
6. *Acoplamento forte:* classes que são fortemente acopladas são difíceis de reutilizar isoladamente, uma vez que dependem uma das outras. O acoplamento forte leva a sistemas monolíticos, nos quais não se pode mudar ou remove uma classe sem compreender e mudar muitas outras classes.
7. *Incapacidade para alterar classes de modo conveniente:* algumas vezes é necessário modificar uma classe que não pode ser convenientemente modificada. Talvez necessite do código-fonte e não disponha do mesmo (como é o caso de se trabalhar com bibliotecas comerciais de classes), ou talvez qualquer mudança possa requerer a modificação de muitas subclasses existentes. Padrões de projeto oferecem maneiras para modificações de classes em tais circunstâncias.

# Capítulo 4

# Padrões Utilizados no Framework Desenvolvido

## 4.1 Introdução

O projetista de um framework aposta que sua arquitetura funcionará para todas as aplicações do domínio tratado. Qualquer mudança substancial no projeto do framework reduziria seus benefícios consideravelmente, uma vez que a principal contribuição de um framework para uma aplicação é a arquitetura que ele define. Portanto, é imperativo projetar o framework de maneira que ele seja tão flexível e extensível quanto possível.

Devido às aplicações serem tão dependentes do framework para o seu projeto, elas são particularmente sensíveis a mudanças na interface do framework. À medida que um framework evolui, as aplicações têm que evoluir com ele. Isto torna o acoplamento fraco ainda mais importante; de outra maneira, mesmo pequenas mudanças no framework teriam grandes repercussões.

O que foi discutido é o ponto mais crítico no projeto de um framework. Um framework projetado através do uso de padrões de projeto tem muito maior probabilidade de atingir altos níveis de reusabilidade de projeto e código, comparado com um que não usa padrões de projeto. Frameworks maduros comumente incorporam vários padrões de projeto. Os padrões ajudam a tornar a arquitetura do framework adequada a muitas aplicações diferentes, sem necessidade de reformulação.

Um benefício adicional é obtido quando o framework é documentado com os padrões de projeto que ele usa. Pessoas que conhecem os padrões obtêm mais rapidamente uma compreensão do framework. Mesmo pessoas que não os conhecem podem se beneficiar da estrutura que eles emprestam à documentação do framework. A melhoria da documentação é importante para todos os tipos de software, mas ela é particularmente importante para frameworks. Os frameworks têm uma curva de aprendizado acentuada, que tem que ser percorrida antes que eles se tornem úteis. Embora os padrões de projeto não possam achatar a curva de

aprendizado completamente, podem torná-la mais suave ao fazer com que os elementos-chave do projeto do framework se tornem mais explícitos [GAM 02].

Os tópicos subseqüentes abordam os padrões de projetos utilizados no framework desenvolvido no presente trabalho. Os padrões utilizados foram os mais básicos e importantes. Eles praticamente ditaram a organização estrutural do projeto do framework desenvolvido.

## 4.2 Acoplamento Fraco (Low Coupling) [ASW 98]

Este padrão tem a finalidade de atribuir uma responsabilidade a uma classe de maneira que o acoplamento permaneça fraco. O acoplamento é uma medida de quão fortemente uma classe está conectada a outras classes, tem conhecimento das mesmas ou depende delas. Uma classe com acoplamento fraco (ou baixo) não é dependente de muitas outras classes (muitas outras é uma expressão que depende do contexto tratado).

Uma classe com acoplamento forte (ou alto) depende de muitas outras classes. Tais classes são indesejáveis, pois sofrem dos seguintes problemas:

- Mudanças em classes relacionadas forçam mudanças locais.
- Mais difíceis de compreender isoladamente.
- Mais difíceis de reutilizar, porque o seu uso requer a presença adicional das classes das quais ela depende.

Para exemplificar, considere o diagrama de classe parcial de uma aplicação de terminal ponto de venda ilustrado pela figura 4.1. Suponha que necessitamos criar uma instância de *Pagamento* e associá-la a *Venda*. Qual classe deveria ser responsável por isso?

Figura 4.1 – Diagrama de classes parcial

Uma vez que *Caixa* registra um *Pagamento* no domínio do mundo real, ele seria um candidato para criar o *Pagamento*. A instância *Caixa* poderia então enviar uma mensagem *acrescentarPagamento* para a classe *Venda*, passando um novo *Pagamento* como parâmetro. Um possível diagrama de colaboração parcial refletindo isso é ilustrado na figura 4.2.

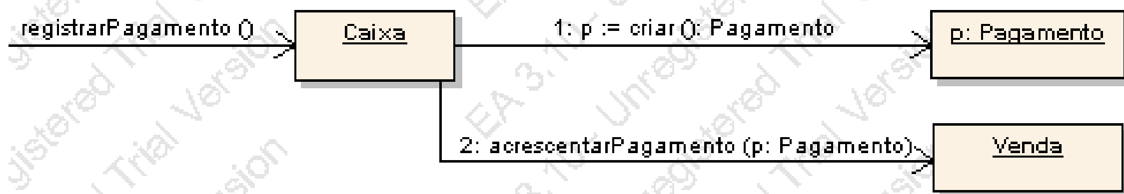


Figura 4.2 – Primeiro diagrama de colaboração

Esta atribuição de responsabilidade acopla a classe *Caixa* à classe *Pagamento.* Uma solução alternativa para criar *Pagamento* é associa-lo à *Venda*, como ilustrado na figura 4.3:

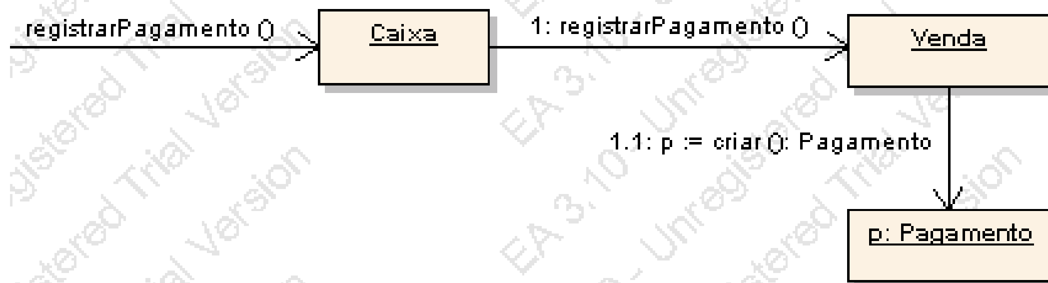


Figura 4.3 – Segundo diagrama de colaboração

Qual das soluções de projeto, baseada na atribuição de responsabilidades, resulta em Acoplamento Fraco? Em ambos os casos iremos supor que a *Venda* deve possuir conhecimento de um *Pagamento* relativo a ela. O primeiro projeto, no qual *Caixa* cria o *Pagamento* é acrescentado um acoplamento de *Caixa* a *Pagamento*, enquanto o segundo projeto, no qual a *Venda* é responsável pela criação do *Pagamento*, o acoplamento não é aumentado, pois quando a classe *Venda* quiser obter informações sobre o P*agamento*, não precisará perguntar a classe *Caixa*.

O Acoplamento Fraco é um princípio a ser levado em conta em todas as decisões de projeto, é um objetivo subjacente que se deve ter sempre em mente. Ele

é um padrão de *avaliação*, ou seja, um padrão que um projetista utiliza enquanto avalia todas as decisões de projeto.

O Acoplamento Fraco estimula a atribuição de uma responsabilidade de maneira que a sua localização não aumente o acoplamento até um nível que conduza aos resultados negativos que o acoplamento forte pode produzir. Ele apóia o projeto de classes que são mais independentes. Isso reduz o impacto de mudanças e promove maior capacidade de reutilização, o que aumenta as oportunidades de se obter uma produtividade mais alta. O Acoplamento Fraco não pode ser considerado isoladamente de outros padrões, como, por exemplo, o Coesão Alta, explicado no tópico seguinte, mas necessita ser incluído como um dentre vários princípios de projeto que influenciam uma escolha de atribuição de uma responsabilidade.

O acoplamento pode não ser tão importante se a reutilização não for um objetivo, como apoio à melhoria do reuso de componentes, fazendo com que eles sejam mais independentes. Todo o contexto dos objetivos de reutilização deve ser levado em consideração antes de se fazer um esforço para minimizar o acoplamento, pois às vezes, é gasto um tempo excessivo tentando obter-se componentes reutilizáveis para futuros projetos "mitológicos", muito embora não haja uma indicação clara de uma necessidade de reuso do componente. Isto não quer dizer que tentar obter o reuso seja um esforço desperdiçado, mas o esforça deverá ser moderado por considerações de custo-benefício.

Não há uma medida absoluta que determine quando o acoplamento é muito forte. O importante é que um desenvolvedor possa avaliar o grau atual de acoplamento e julgar se o seu aumento trará problemas. Em geral, as classes que são inerentemente muito genéricas por natureza e com alta probabilidade de reutilização, deveriam ter acoplamento especialmente fraco.

O caso extremo do Acoplamento Fraco é quando não existe ou existe muito pouco acoplamento entre as classes. Isso não é desejável, porque uma metáfora central da tecnologia de objetos é um sistema composto de objetos conectados que se comunicam através de mensagens. Se o Acoplamento Fraco é aplicado em excesso, leva a um projeto fraco, porque conduz a uns poucos objetos ativos, sem coesão, inchados e complexos, que efetuam todo o trabalho. Ao mesmo tempo,

existirão muitos objetos praticamente passivos, com acoplamento zero, que funcionam como simples repositório de dados. Algum grau moderado de acoplamento entre as classes é normal e necessário, de forma a criar um sistema orientado a objetos, no qual as tarefas são executadas por uma colaboração entre objetos conectados.

## 4.3 Coesão Alta (High Cohesion) [ASW 98]

Este padrão tem a finalidade de atribuir uma responsabilidade a uma classe de forma que a coesão permaneça alta. Em termos de projeto orientado a objetos, coesão (ou mais especificamente coesão funcional) é uma medida de quão fortemente relacionadas e focalizadas são as responsabilidades de uma classe. Uma classe com responsabilidades altamente relacionadas e que não executa um formidável volume de trabalho tem coesão alta.

Uma classe com coesão baixa faz muitas coisas não-relacionadas, ou executa demasiado trabalho. Tais classes são indesejáveis, pois sofrem dos seguintes problemas:

- Difíceis de compreender.
- Difíceis de reutilizar.
- Difíceis de manter.
- Delicadas, pois são constantemente afetadas pelas mudanças.

Classes com coesão baixa representam, geralmente, uma abstração de “grande granularidade” ou, então, assumiram responsabilidades que deveriam ter sido delegadas a outros objetos.

O mesmo exemplo utilizado no padrão Acoplamento Fraco pode ser utilizado aqui. Se utilizarmos o projeto da figura 4.2, estaremos atribuindo a responsabilidade de criar um *Pagamento* para *Caixa*. Neste exemplo isolado, isso pode até ser aceitável, porém se continuarmos a fazer a classe *Caixa* responsável por realizar algum trabalho ou a maior parte dele, o qual está relacionado com cada vez mais operações do sistema, ela se tornará progressivamente carregada com tarefas e perderá sua coesão.

Imagine que existissem cinqüenta operações do sistema, todas recebidas pela classe *Caixa*. Se ela fizesse o trabalho relacionado a cada uma, se tornaria um objeto “inchado”, sem coesão. Ao contrário, como ilustrado na figura 4.3, o segundo projeto delega a criação do *Pagamento* para a classe *Venda*, o que favorece uma coesão mais alta na classe *Caixa*, além de um acoplamento fraco.

Na prática, o nível de coesão sozinho não pode ser considerado isoladamente de outras responsabilidades e de outros princípios.

Assim como o Acoplamento Fraco, a Coesão Alta é um princípio a se ter em mente durante todas as decisões de projeto; é um objetivo subjacente a ser levado em conta continuamente; é um padrão de *avaliação* que um projetista utiliza enquanto julga todas as decisões de projeto.

Grady Booch descreve a coesão funcional alta como algo que existe quando os elementos de um componente (tal como uma classe) trabalham todos em conjunto, para fornecer algum comportamento bem-delimitado [LAG 00 referenciando BOO 94].

Abaixo, alguns cenários que ilustram graus variáveis de coesão funcional:

1. *Coesão Muito Baixa*: uma classe é a única responsável por muitas coisas em áreas funcionais muito diferentes.

2. *Coesão Baixa*: uma classe é a única com responsabilidades de uma tarefa complexa em uma área funcional.

3. *Coesão Alta*: uma classe tem responsabilidades moderadas em uma área funcional e colabora com outras classes para levar a termo as tarefas.

4. *Coesão Moderada*: uma classe tem peso leve e responsabilidades exclusivas em umas poucas áreas diferentes que estão logicamente relacionadas ao conceito da classe, mas não entre si.

Como regra prática, uma classe com coesão alta tem um número relativamente pequeno de métodos, com funcionalidades altamente relacionadas e não executa muito trabalho. Se a tarefa for grande, ela colabora com outros objetos para compartilhar o esforço.

Uma classe com coesão alta é vantajosa porque é relativamente fácil de manter, compreender e reutilizar. O alto grau de funcionalidades relacionadas, combinado com um pequeno número de operações, também simplifica a manutenção e as melhorias. A granularidade fina de funcionalidades altamente relacionadas também suporta um aumento do potencial de reutilização, porque uma classe altamente coesiva pode ser usada para uma finalidade muito específica.

## 4.4 Template Method [GAM 02]

Este padrão nos mostra como definir o esqueleto de um algoritmo em uma operação, postergando alguns passos para as subclasses, permitindo que as mesmas redefinam certos passos do algoritmo sem mudar a estrutura do mesmo.

O Framework desenvolvido no presente trabalho utiliza a classe *TcsEntidade* para implementar as regras de negócios que irão manipular os dados do banco de dados. Esta classe possui um método chamado *SetSelect* que configura o comando sql a ser inserido no componente de acesso aos dados. Quando a classe é criada, este método é chamado para gerar o sql padrão no componente de acesso aos dados. Para que este select possa ser gerado, o método precisa sabe o nome da tabela que está se trabalhando. Para isto, este método chama o método *GetNomeTabela*, um método virtual que deve ser obrigatoriamente substituído nas subclasses de TcsEntidade.

O método SetSelect é um *template method*. Um método template define um algoritmo em termos da operação abstrata que as subclasses redefinem para fornecer um comportamento concreto. O método GetNomeTabela é um método *hook*. Um método hook é um método gancho que fornece comportamento por falta que subclasses podem estender se necessário. Um método gancho geralmente não executa nada por falta. A figura 4.4 ilustra as classes descritas acima.

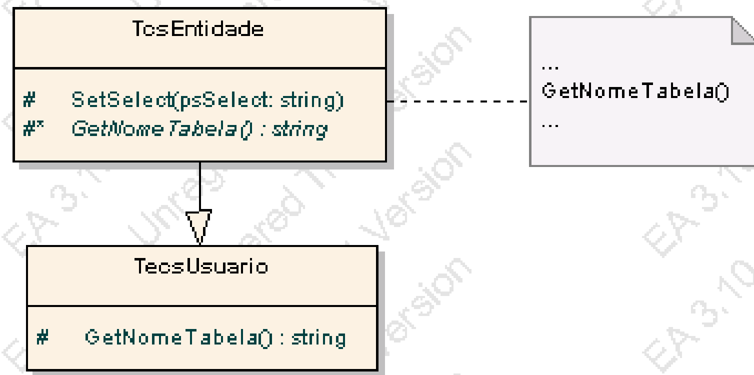


Figura 4.4 – Exemplo template method

O padrão Template Method pode ser usado:

- Para implementar as partes invariantes de um algoritmo uma só vez e deixar para as subclasses a implementação do comportamento que pode variar.
- Quando o comportamento comum entre subclasses deve ser fatorado e concentrado numa classe comum para evitar a duplicação de código. Primeiramente você identifica as diferenças no código existente e então separa as diferenças em novas operações. Por fim, você substitui o código que apresentava as diferenças por um método template que chama uma destas novas operações.
- Para controlar extensões de subclasses. Você pode definir um método template que chama métodos ganchos (hook) em pontos específicos, desta forma permitindo extensões somente nestes pontos.

Os métodos template são uma técnica fundamental para a reutilização de código. Eles são particularmente importantes em bibliotecas de classes porque são os meios para a fatoração dos comportamentos comuns nas bibliotecas de classes.

Os métodos template conduzem a uma estrutura de inversão de controle, algumas vezes chamada de “o princípio de Hollywood”, ou seja: “não nos chame, nós chamaremos você” [SWE 85]. Isto se refere a como uma classe-mãe chama as operações de uma subclasse, e não o contrário.

É importante para os métodos template especificarem quais métodos são ganchos (*podem* ser redefinidos) e quais são métodos abstratos (*devem* ser

redefinidos). Para reutilizar uma classe abstrata efetivamente, os codificadores de subclasses devem compreender quais as operações projetadas para redefinição.

## 4.5 Fachada (Facade) [GAM 02]

O padrão de projeto Fachada fornece uma interface unificada para um conjunto de interfaces em um subsistema. Fachada define uma interface de nível mais alto que torna o subsistema mais fácil de ser usado.

Estruturar um sistema em subsistemas ajuda a reduzir a complexidade. Um objetivo comum de todos os projetos é minimizar a comunicação e as dependências entre subsistemas. Uma maneira de atingir este objetivo é introduzir um objeto fachada, o qual fornece uma interface única e simplificada para os recursos e facilidades mais gerais de um subsistema. A figura 4.5 ilustra o relacionamento entre as classes de um sistema sem a utilização do padrão Fachada e com a utilização do padrão Fachada.

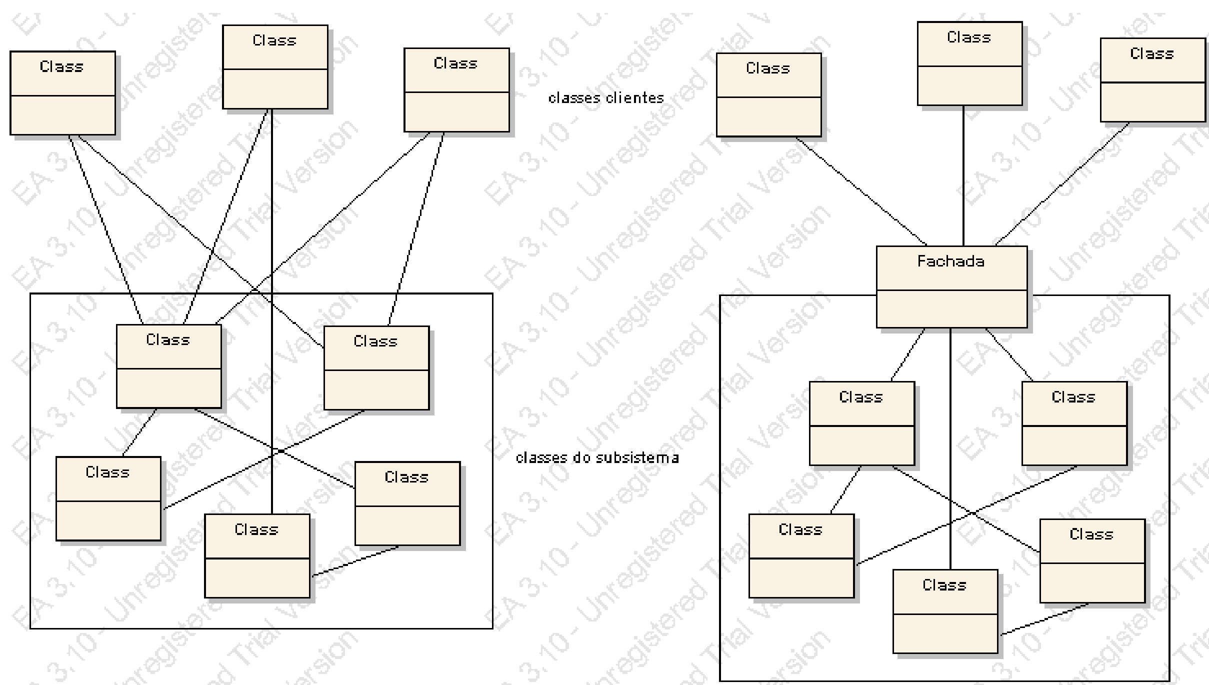


Figura 4.5 – Exemplo da utilização do padrão Fachada

Considere, por exemplo, um ambiente de programação que fornece acesso às aplicações para o seu subsistema compilador. Este subsistema contém classes, tais como Scanner, Parser, ProgramNode, BytecodeStream e ProgramNodeBuilder, que

implementam o compilador. Algumas aplicações especializadas podem necessitar acessar estas classes diretamente. Mas a maioria dos clientes de um compilador geralmente não se preocupa com detalhes tais como análise (*parsing*) e geração de código; eles apenas querem compilar seu código. Para eles, as interfaces poderosas, porém de baixo nível, do subsistema compilador somente complicam sua tarefa.

Para fornecer uma interface de nível mais alto que pode isolar os clientes destas classes, o subsistema compilador também inclui uma classe Compiler. A classe Compiler funciona como uma fachada, que oferece aos clientes uma interface única e simples para o subsistema compilador. Ela junta as classes que implementam a funcionalidade de um compilador, sem ocultá-las completamente. O compilador fachada torna a vida mais fácil para a maioria dos programadores, sem, entretanto, ocultar a funcionalidade de nível mais baixo dos poucos que a necessitam.

O padrão Fachada pode ser usado quando:

- Você desejar fornecer uma interface simples para um subsistema complexo. Os subsistemas se tornam mais complexos à medida que evoluem. A maioria dos padrões, quando aplicados, resultam em mais e menores classes. Isto torna o subsistema mais reutilizável e mais fácil de customizar, porém, ele também se torna mais difícil de usar para aqueles clientes que não necessitam customizá-lo. Uma fachada pode fornecer, por omissão, uma visão simples do sistema, que é boa o suficiente para a maioria dos clientes. Somente os clientes que necessitam maior customização necessitarão olhar além da fachada.
- Existirem muitas dependências entre os clientes e as classes de implementação de uma abstração. Ao introduzir uma fachada para desacoplar o subsistema dos clientes e de outros subsistemas, estar-se-á promovendo a independência e portabilidade dos subsistemas.
- Você desejar estruturar em camadas seus subsistemas. Use uma fachada para definir o ponto de entrada para cada nível de subsistemas. Se os subsistemas são independentes, então você pode simplificar as dependên-

cias entre eles fazendo com que se comuniquem uns com os outros exclusivamente através de suas fachadas.

O padrão fachada oferece os seguintes benefícios:

- Ele isola os clientes dos componentes do subsistema, desta forma reduzindo o número de objetos com que os clientes têm que lidar e tornando o subsistema mais fácil de usar.
- Ele promove um acoplamento fraco entre o subsistema e seus clientes. Freqüentemente, os componentes num subsistema são fortemente acoplados. O acoplamento fraco lhe permite variar os componentes do subsistema sem afetar os seus clientes.
- Ele não impede as aplicações de utilizarem as classes do subsistema caso necessitem fazê-lo. Assim, você pode escolher entre a facilidade de uso e a generalidade.

# Capítulo 5

# Framework Desenvolvido

## 5.1 Introdução

No decorrer do processo de desenvolvimento de softwares, uma grande parcela do tempo de desenvolvimento é gasta na estruturação do projeto do software, em como suas classes se relacionam e na implementação das interfaces que o usuário irá acessar. Como já dito em capítulos precedentes, a grande vantagem da utilização de frameworks é a reutilização de código e projeto, que tem por intuito diminuir o tempo e o esforço no desenvolvimento de softwares. Esta abordagem tem como grande característica estabelecer padrões estruturais, de interface e código e controlar o fluxo da aplicação, liberando o desenvolvedor de software desta preocupação e fazendo com que o mesmo gaste a maior parte do seu tempo na solução dos problemas relativos ao software em desenvolvimento.

O presente capítulo dará ênfase a descrição do suporte a desenvolvimento oferecido pelo framework, suas principais classes e relacionamentos e como o mesmo está estruturado. Este capítulo não pretende apresentar a especificação de todo o projeto do framework, pois seria demasiado extenso e cansativo, mas ressaltar as partes de sua estrutura que lhe conferem flexibilidade para suportar o desenvolvimento de novos softwares.

## 5.2 Ferramentas Utilizadas

Para implementação do framework, a ferramenta utilizada foi o ambiente de programação Borland Delphi 6. Esta ferramenta foi escolhida por ser excelente no desenvolvimento de aplicativos desktop, pois oferece uma fácil programação de interface e componentes. O framework desenvolvido no presente trabalho utiliza-se muito destas duas características.

O banco de dados utilizado para que o framework pudesse ser testado foi o Firebird. Ele é um banco descendente do Interbase 6.0. Tornou-se um banco independente a ele quando a Borland liberou os códigos fontes do Interbase 6.0.

Este foi o banco de dados escolhido por ser um banco freeware, exigir pouca memória do computador, ser um banco de fácil manuseio, possuir um bom ferramental de trabalho e muito material para consulta.

Para modelagem das tabelas do banco de dados (ER), utilizou-se a ferramenta Erwin 4.1. Esta é uma das mais famosas ferramentas de modelagem ER disponíveis no mercado. Foi a ferramenta escolhida por ser a ferramenta que domino e por possuir uma excelente qualidade visual para a modelagem dos dados. Esta ferramenta não só ilustra graficamente o relacionamento das tabelas do banco de dados como também gera o script de criação do banco de dados.

Para executar o script gerador do banco de dados, trabalhar sobre os dados, gerar e testar sqls, foi utilizada a ferramenta IBExpert 2.5. Esta foi a ferramenta escolhida por ser a melhor ferramenta de administração de banco de dados para Interbase freeware disponível no mercado.

Para desenhar os diagramas de classe apresentados no trabalho utilizou-se da ferramenta Enterprise Architect 3.10 Trial. Esta foi a ferramenta escolhida por ser uma ferramenta de fácil manuseio e oferecer todos os recursos que o trabalho precisava.

## 5.3 Padrões de Nomenclatura

Toda a nomenclatura de código, unidades de código fontes (units), classes, formulários, entidades do banco de dados (tabelas) e propriedades e eventos de componentes do framework seguem um padrão de nomenclatura. O código implementado, além do padrão de nomenclatura segue um padrão de identação. Utilizar um padrão de nomenclatura e identação no desenvolvimento de software é algo obrigatório. Todo desenvolvedor de software sabe o quanto é difícil dar manutenção em um código escrito por outro desenvolvedor, ainda mais se este código não seguir nenhum padrão de nomenclatura e identação. Imagine agora uma empresa que tenha 10 programadores trabalhando em um mesmo programa. Se cada programador implementa o código a sua maneira, o software vai parecer um tremendo remendo. Isto traz problemas de manutenção dentro da própria empresa. Se um programador tiver que dar manutenção no código de um colega seu, este terá

dificuldades para interpretar o código mesmo os dois trabalhando na mesma empresa. Imagine então se a manutenção for feita por um programador que não pertence à empresa e devido a isto provavelmente não terá tanto conhecimento do software como um programador desta.

Utilizar padrões de nomenclatura e identação, além de facilitar a manutenção do código, pois a nomenclatura e identação do código escrito pelos programadores da empresa serão a mesma, deixa o código muito mais bonito e organizado, facilitando não só aos programadores da empresa, mas a qualquer programador que tiver acesso ao código e aos padrões utilizados, interpretar muito mais facilmente o significado do código implementado.

A seguir, serão listados os padrões de nomenclaturas utilizados no framework em nomes units, classes, formulários, entidades e propriedades e eventos de componentes. Padrões para nomenclatura e identação de código também foram utilizados no framework, porém, não serão abordados no presente trabalho por se tratarem de padrões mais voltados ao código implementado, sendo importantes para o desenvolvedor que utiliza o framework na criação de novos softwares e não para o entendimento estrutural e comportamental do framework ao qual o presente capítulo se propõe a elucidar.

**Sigla CS:** esta é a sigla mais importante no padrão de nomenclatura utilizado pelo framework. CS aqui significa Client/Server. Esta sigla será utilizada em toda a nomenclatura de units, classes e entidades do framework. A finalidade desta sigla é identificar que uma unit, classe ou entidade pertence ao framework. Isto é útil para que o usuário do framework possa separar o que pertence ao framework e o que pertence ao software desenvolvido sobre o framework. Por exemplo, a classe de segurança do framework é nomeada como *TcsSeguranca*, ficando evidente para um usuário do framework que a classe pertence a ele. É aconselhável ao usuário do framework utilizar o mesmo padrão para a nomenclatura de seu software. Digamos, por exemplo, que o framework está sendo utilizado no desenvolvimento de um software de locadora, e esta locadora chama-se Olho Vivo. Seria aconselhável ao desenvolvedor do software utilizar, por exemplo, a sigla “ov” para identificar que uma

unit, classe ou entidade pertence ao software. Desta forma, fica evidente que uma classe de nome *TovEmprestimo* pertence ao software e não ao framework.

**Units:** todas os nomes de unidades de código fonte (units) do framework são precedidos da letra “u” (unit) seguida da sigla identificadora do framework “cs”. Assim, a unit responsável pela implementação da segurança no framework é nomeada com *ucsSeguranca*. Se olharmos para um arquivo de nome *ucsSeguranca.pas,* através do padrão de nomenclatura utilizado, sabemos que o arquivo trata-se de uma unit e pertence ao framework. Da mesma forma, utilizando o padrão de nomenclatura, um arquivo de nome *uovEmprestimo.pas* nos diz que é uma unit pertencente ao software da locadora Olho Vivo.

**Classes:** todos os nomes de classes do framework são precedidos da letra “T” seguida da sigla identificadora do framework “cs”. Assim, a classe responsável pela segurança no framework é nomeada com *TcsSeguranca*. Veja que aqui, além do padrão para a nomenclatura da classe, existe um padrão entre o nome da classe e o nome da unit. A classe principal de uma unit possui o mesmo nome da unit, apenas trocando-se a letra inicial “u” da unit pela letra “T”. A letra “T” não é só a letra padrão para identificação de classes, ela é utilizada em todos os Tipos (type) no framework. Esta é a nomenclatura utilizada pelo ambiente de desenvolvimento Delphi.

**Formulários:** todos os nomes de formulários no framework possuem o mesmo nome que suas units, trocando-se a letra “u” por “f”. Assim, se a unit de cadastro de usuários se chamar *ucsCadUsuario*, o nome de seu formulário será *fcsCadusuário*.

**Entidades:** entidades no presente trabalho estão relacionadas às tabelas do banco de dados. Elas possuem uma nomenclatura própria devido a sua importância no framework. As entidades são os elementos mais importantes em um software de característica cliente/servidor. A finalidade de uma nomenclatura específica para entidades é fornecer ao usuário do framework, baseado no nome da entidade no banco de dados, saber qual o nome da unit e da classe que representam esta entidade no framework. Exemplificarei para que se possa entender melhor este padrão de nomenclatura.

O framework possui uma entidade denominada *ecsUsuario*, cuja finalidade é armazenar os dados dos usuários cadastrados no framework. Nomes de entidades no framework iniciam com a letra “e” seguida da sigla identificadora do framework “cs” e sempre no singular. Com este padrão de nomenclatura, sabemos que *ecsUsuario* é uma entidade pertencente ao framework. Como já dito antes, é aconselhável que entidades que não pertençam ao framework utilizem este padrão de nomenclatura para identificar suas entidades. No caso da locadora, a entidade que armazenaria os empréstimos seria *eovEmprestimo*.

Para que o software possa acessar esta entidade do banco de dados, o usuário do framework deve implementar uma classe derivada *TcsEntidade* (esta classe será explicada mais adiante). Esta classe deve ser nomeada com o mesmo nome da entidade do banco de dados, precedido da letra “T”. Assim, ela irá chamar-se *TecsUsuario* e o nome da unit onde está contida será *ucsUsuario*. Seguindo este padrão de nomenclatura, basta o usuário do framework saber o nome da entidade no banco de dados para saber o nome da unit e da classe que o implementam no framework, bem como sabendo o nome da classe saberá o nome da entidade no banco de dados. Isto traz um ganho de produtividade enorme. Digamos que o usuário do framework esteja implementando um novo cadastro em seu software e deseje acessar os dados da entidade do banco de dados *ecsUsuario*. Ele não precisa perder tempo para descobrir que classe do framework implementa esta entidade, pois, seguindo o padrão de nomenclatura ele sabe que é a classe *TecsUsuario* contida na unit *ucsUsuario* a responsável pela implementação.

**Propriedades e Eventos de Componentes:** o nome das propriedades e eventos publicados nos componentes oferecidos pelo framework são precedidos pela sigla identificadora do framework “cs”. Propriedades e eventos publicados são as propriedades e eventos de um componente que aparecem no Object Inspector do Delphi.

## 5.4 Módulos Estruturais

O framework desenvolvido no presente trabalho está estruturado em três módulos, *csComponentes, csEntidades e csAplicação*. O módulo csComponentes é o local onde os componentes a serem utilizados pelo framework são implementados. Este módulo não está vinculado aos outros dois, isto é, não depende deles para funcionar. Ele é um módulo auxiliar, que poderia ser utilizado por um outro aplicativo qualquer. O módulo csEntidades é o local onde as entidades que implementam as regras de negócio e o acesso aos dados do banco estão localizadas. Este módulo utiliza algumas classes do módulo csComponentes, mas não de seus componentes. Também compartilha de classes que pertencem também ao módulo csAplicação, isto é necessário para que os dois módulos possam se comunicar. O módulo csAplicação é o módulo que contém as interfaces que compõem o aplicativo. É neste módulo que as janelas sobre as quais o usuário irá interagir com o sistema serão implementadas. Ele utiliza os componentes e classes do módulo csComponentes e das entidades do módulo csEntidades.

A seguir, será discutido qual a finalidade, que classes compõem e como se relacionam os três módulos citados acima.

### 5.4.1 Módulo csComponentes

O módulo csComponentes é o local onde os componentes e classes auxiliares do framework são implementados. Os componentes e classes presentes neste módulo não dependem dos demais módulos, podendo ser usado em separado em um outro aplicativo que não utilize o framework. Três componentes e uma unit deste módulo são utilizadas pelos outros dois módulos do framework. Os componentes são os botões *csBotaoCadastro*, *csBotaoInsSeta* e *csBotaoDelSeta* e a unit *ucsMsgDlg*.

#### 5.4.1.1 Componentes

O módulo csComponente possui uma classe especializada na criação de botões personalizados. O módulo csAplicação utiliza muito de botões em suas janelas. Estes botões possuem uma figura que representa a operação que se deseja

realizar, por exemplo, o botão fechar possui uma figura característica que torna evidente ao usuário a operação realizada por ele. Este é um botão que irá aparecer em todas as janelas do módulo csAplicação. Esta figura possui um tamanho de 3,02Kb. Imagine que em média uma janela utilize quatro botões com figuras de 3,02Kb e o aplicativo possua um total de 30 janelas. O tamanho do arquivo executável do aplicativo terá então 362,4Kb (3,02Kb x 4 x 30) a mais de tamanho devido apenas as figuras dos botões. A figura 5.1 apresenta a janela de Cadastro de Grupos de Usuários utilizada pelo framework. Observe na figura a utilização de cinco botões contendo figuras. Os botões do lado superior direito da janela são os botões csBotaoInsSeta e csBotaoDelSeta e os três botões do lado inferior esquerdo da janela são botões csBotaoCadastro.

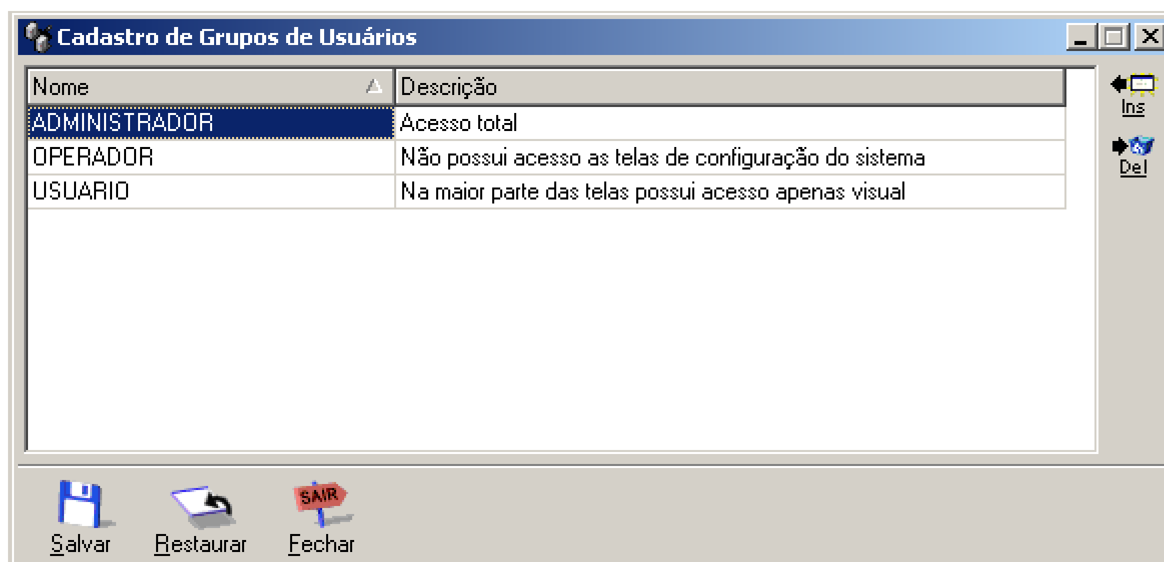


Figura 5.1 – Ilustração da utilização de botões contendo figuras

Para solucionar este problema bem como livrar o desenvolvedor de software a ter que ficar configurando cada vez que precisar inserir um botão em uma janela, a figura que ele irá exibir, seu caption, hint e dimensões, a classe *TcsCustomBotao* foi criada. Esta classe é uma classe abstrata, que oferece suporte para a criação de botões personalizados, bastando ao desenvolvedor criar uma nova classe filha e sobrescrever alguns poucos métodos para que se tenha uma classe concreta que implementa um botão personalizado. Nesta nova classe, o desenvolvedor irá especificar todas as configurações do botão, como suas dimensões, figuras que

pode exibir e para cada figura qual será seu hint e caption. Assim, quando este botão for jogado em um formulário, ele virá com estas configurações já definidas, livrando o desenvolvedor desta tarefa.

O problema do espaço ocupado pelas figuras dos botões foi resolvido utilizando-se de arquivos de recursos (.res). Um arquivo de recurso é um arquivo compilado, que em nosso caso, irá conter figuras. Para que um botão possa exibir figuras, deve-se criar um arquivo de recursos contendo todas as figuras que o botão pode exibir. Quando o usuário seleciona a figura que deseja exibir no botão, no caso do botão poder exibir mais de uma figura, como é o caso do botão csBotaoCadastro, a classe se encarrega de acessar o arquivo de recursos para obter a figura selecionada pelo usuário e atribuí-la ao botão. Desta forma, as figuras não irão aumentar o tamanho do arquivo executável do aplicativo, pois estarão contidas no arquivo de recursos. Para que isto funcione, o arquivo de recursos deve ser distribuído junto com o arquivo executável.

Gerar o arquivo de recursos para as figuras a serem utilizadas nos botões é extremamente fácil. Não irei aqui explicar como isto é feito, para isto, existem inúmeras bibliografias relacionadas ao ambiente Delphi que abordam este tópico.

A figura 5.2 ilustra a hierarquia de classes que compõem o conjunto de botões desenvolvidos no framework.

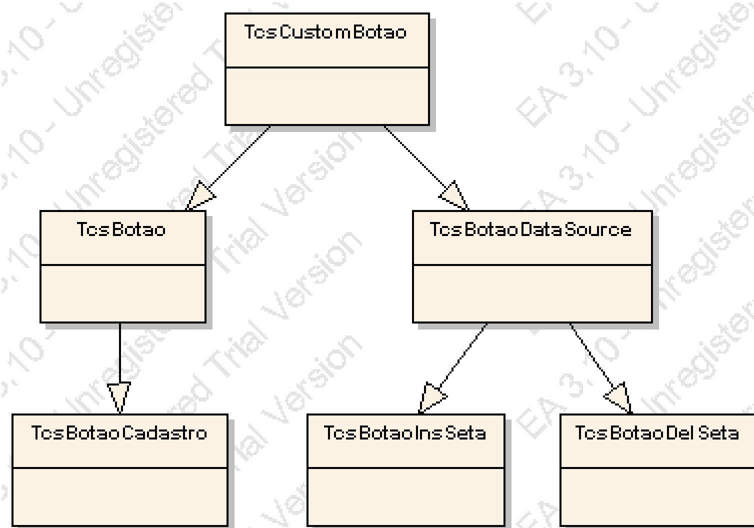


Figura 5.2 – Hierarquia de classes dos componentes de botão

Como já dito anteriormente, a classe *TcsCustomBotao* é uma classe abstrata que implementa todo o código responsável pela criação do botão personalizado. Assim, para que se possa criar um novo botão personalizado deve-se herdar desta classe. A classe *TcsBotao* não possui programação adicional em relação à classe *TcsCustomBotao*. Esta classe apenas torna pública algumas propriedades protegidas e o evento que trata o click do botão. Esta também é uma classe abstrata e deve ser utilizada quando se quiser criar botões que possam exibir mais de uma figura. A classe *TcsBotaoCadastro* é uma classe concreta que implementa o botão a ser utilizado em janelas de cadastro, possuindo 13 figuras que representam operações relacionadas a cadastro, como Salvar e Restaurar. Os botões do canto inferior esquerdo da figura 5.1 são botões da classe *TcsBotaoCadastro*. A classe *TcsBotaoDataSource* é uma classe abstrata, possuindo uma propriedade denominada *csDataSource*, onde o usuário deve especificar o componente *DataSource* a ser vinculado ao botão. No ambiente de desenvolvimento Delphi, o componente *DataSource* implementa o elo de ligação entre um conjunto de dados (TQuery por exemplo) e os componentes de visualização de dados (TDBGrid por exemplo). A classe oferece suporte à execução de operações sobre o conjunto de dados vinculado a propriedade *csDataSource* quando o usuário der um click no botão. A classe *TcsBotaoInsSeta* é uma classe concreta que executa a operação de inserção no conjunto de dados vinculado à propriedade *csDataSouce* do componente quando um click for dado sobre o botão. A classe *TcsBotaoDelSeta* é uma classe concreta que executa a operação de deleção no conjunto de dados vinculado a propriedade *csDataSouce* do componente quando um click for dado sobre o botão. A classe irá exibir um quadro de diálogo perguntando ao usuário se deseja excluir o registro antes de executar a operação de deleção sobre o conjunto de dados.

#### 5.4.1.2 Quadros de Diálogo

Além dos componentes de botão, o módulo csComponentes possui uma unit muito utilizada no framework, a unit *ucsMsgDlg*. Esta unit implementa os quadros de diálogo onde as mensagens do framework serão exibidas.

Quadros de diálogos são muito utilizados em qualquer aplicativo, é através deles que o software se comunica com o usuário que o está utilizando. Quando um

erro ocorre, um quadro de diálogo é exibido notificando o usuário sobre o erro ocorrido. Quando uma operação precisa ser confirmada, é através de um quadro de diálogo que o software interroga o usuário para confirmar a execução ou não da operação. O ambiente de desenvolvimento Delphi oferece quadros de diálogo onde o desenvolvedor pode digitar a mensagem que deseja transmitir ao usuário do software. Um dos mais interessantes são os quadros de diálogo exibidos pelo método *MessageDlg* da unit *Dialogs*. Este método permite selecionar que tipo de quadro de diálogo que se deseja exibir (mtWarning, mtError, mtInformation, mtConfirmation, mtCustom), porém existe um sério problema aqui, o quadro de diálogo está em Inglês e só é possível utilizar os tipos oferecidos pelo método. Para suplantar este problema, a unit *ucsMsgDlg* foi criada. Ela tem um funcionamento muito parecido com a unit *Dialogs* e utiliza as mesmas figuras que ela em seus quadros de diálogo, com o diferencial de seus quadros de diálogo estarem em Português e oferecendo a possibilidade de criação de novos tipos de quadros. A figura 5.3 ilustra dois dos cinco tipos de quadros de diálogo oferecidos pela unit *ucsMsgDl*.

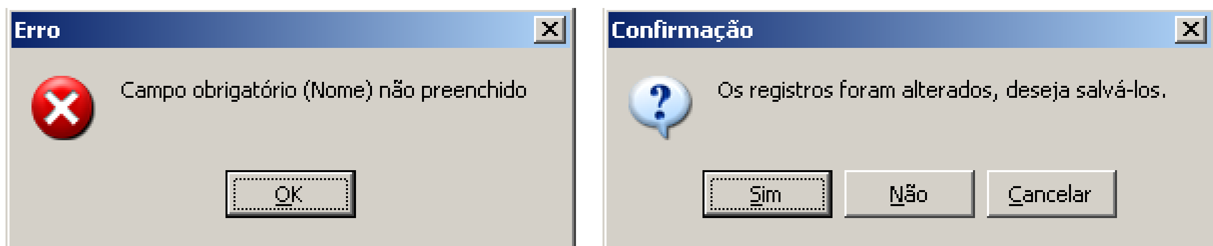


Figura 5.3 – Quadro de diálogo de erro e confirmação de operação

### 5.4.2 Módulo csEntidades

O módulo csEntidades é o local onde as entidades do framework são implementadas. Uma Entidade é geralmente uma classe que representa uma tabela do banco de dados, mas também pode ser uma classe que implementa regras de negócio que trabalham em várias tabelas ou até regras de negócio que não têm nenhum vínculo com o banco de dados.

A finalidade do módulo csEntidades é separar as regras de negócio da camada de apresentação. Uma entidade, após implementada e compilada, é registrada como

um componente no Delphi. Assim, para que o módulo csAplicação utilize esta entidade, basta acessar o componente que a representa na paleta de componentes do Delphi e jogá-lo sobre um formulário ou criá-lo dinamicamente via código. Todas as regras de negócio da entidade estarão “dentro do componente”. Com isto, as regras de negócio implementadas na entidade podem ser utilizadas em vários lugares diferentes, o que as tornam totalmente separadas da camada de apresentação favorecendo a reutilização e a modularização, visto que estão contidas em um único local.

Além das classes que dão suporte a implementação de entidades, o módulo csEntidades possui uma classe e uma unit que também devem ser destacadas, a classe *TcsTrataMsgErro*, responsável pelo tratamento das mensagens de erro geradas pelo aplicativo e a unit *ucsMensagens,* que tem o propósito de padronizar as mensagens exibidas nos quadros de diálogo ao usuário.

A figura 5.4 ilustra a hierarquia das classes que dão suporte a implementação de Entidades do módulo csEntidades. Os tópicos seguintes do capítulo irão explicar em detalhes cada uma das classes ilustradas na figura, a classe TcsTrataMsgErro e a unit ucsMensagens.

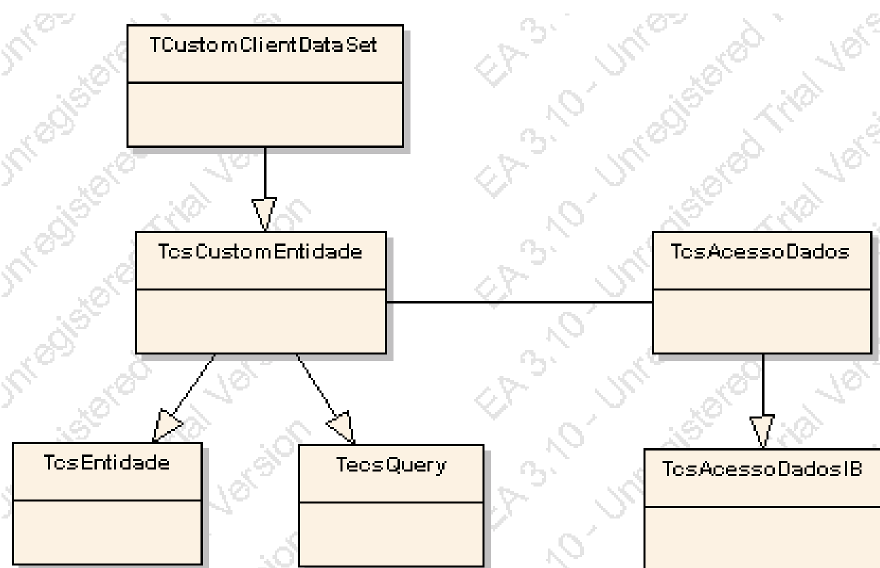


Figura 5.4 – Hierarquia de classes do módulo csEntidades

#### 5.4.2.1 TCustomClientDataSet

Esta é uma classe pertencente ao ambiente de desenvolvimento Delphi, sendo a classe-mãe de toda a hierarquia de classes que dão suporte a implementação de entidades no framework. Esta classe foi escolhida devido a precisar-se de uma classe que desse suporte a utilização de diferentes tipos de componentes de acesso a dados[2]. A principal característica desta classe é que ela trabalha com dados localizados em memória. Quando uma requisição de consulta é feita ao banco de dados, os dados retornados serão armazenados na memória do computador. Pode-se então trabalhar sobre eles sem que nenhuma comunicação seja estabelecida com o banco de dados. Somente quando se deseja salvar as alterações feitas é que a classe irá comunicar-se como o banco de dados informando sobre os dados alterados. É esta característica que faz com que a classe dê suporte a diferentes componentes de acesso a dados.

Para que a comunicação entre a classe TCustomClientDataSet e o componente de acesso a dados possa ser estabelecida, uma outra classe é utilizada para que este elo de comunicação possa ser estabelecido, a classe TDataSetProvider. Quando a classe TCustomClientDataSet deseja alguma requisição de dados, ela comunica à classe TDataSetProvider, que repassa o pedido ao componente de acesso a dados vinculado a ela, que por sua vez fará a comunicação com o banco de dados, retornando os dados requisitados a classe TDataSetProvider que os irá repassar à classe TCustomClientDataSet. Pode-se com isto notar que a classe TCustomClientDataSet torna-se uma classe genérica, pois a troca de dados é feita apenas com a classe TDataSetProvider e portanto não tem vínculo ao componente de acesso a dados que se está utilizando.

#### 5.4.2.2 TcsCustomEntidade

Esta é a classe mais importante de todo o framework. É uma classe abstrata que dá suporte a implementação de Entidades. Quando deseja-se criar uma nova entidade, deve-se criar uma subclasse da classe TcsCustomEntidade e sobrescrever alguns métodos virtuais para que a classe funcione corretamente.

[2] Aqui se refere aos componentes do ambiente de programação Delphi responsáveis pela implementação do protocolo de comunicação com o banco de dados, como, por exemplo, os componentes TQuery e TIBQuery.

A classe oferece vários métodos que podem ser sobrescritos para que se possa implementar tratamento adicional sobre dados e eventos pertencentes à Entidade. Dentre os métodos oferecidos, um deve ser obrigatoriamente sobrescrito, o método *GetNomeTabela*. Este método é o responsável por informar o nome da tabela do banco de dados que a entidade está implementando. A classe precisa saber o nome da tabela para que possa gerar os sqls de alteração de dados corretos e obter os campos que fazem parte da chave primária. Isto será melhor discutido quando a classe TcsAcessoDados for abordada.

Apesar do método GetNomeTabela dever ser obrigatoriamente sobrescrito nas subclasses, ele não deve obrigatoriamente retornar o nome de uma tabela existente no banco de dados. Como já dito na introdução do módulo csEntidades, nem sempre criamos uma Entidade para implementar o acesso a dados de uma tabela específica no banco de dados, podemos querer implementar selects de consulta a mais de uma tabela ou criar uma entidade que dê suporte a execução de sqls genéricos, que não estão vinculados a apenas uma tabela, como é o caso da classe TecsQuery, abordada mais a frente. O nome de uma tabela existente no banco de dados deve ser obrigatoriamente fornecido quando a entidade está implementando o acesso a esta tabela, com o intuito de poder alterar seus dados.

Dentre os métodos que podem ser sobrescritos nas subclasses para implementar tratamento adicional sobre dados e eventos da entidade, os métodos a seguir são os principais.

*RegistraSelects(poSelects: TStringList):* este método deve ser sobrescrito quando se deseja registrar diferentes selects que a classe pode executar. Após o componente que representa a entidade ser registrado no Delphi, o usuário pode selecioná-los pela propriedade csSelect.

*ConfiguraSelect(psSelect: string)*: este método deve ser sobrescrito para implementar os selects registrados no método *RegistraSelects*. Quando um select for selecionado pelo usuário, este método será chamado. O parâmetro *psSelect* informa o nome do select selecionado pelo usuário.

*AntesSalvar(var psMsgResposta: string)*: este método deve ser sobrescrito quando se deseja fazer uma crítica sobre os dados antes de serem gravados no

banco de dados, como por exemplo, se um campo obrigatório foi preenchido. Caso alguma validação seja violada, deve-se atribuir algum valor ao parâmetro *psMsgResposta,* informando sobre o problema. A classe irá então gerar uma exceção contendo a mensagem informada no parâmetro *psMsgResposta* e a operação será cancelada.

*AntesSalvarRegistro()*: este método deve ser sobrescrito quando se deseja tratar os dados antes de serem salvos no banco de dados. Este método será chamado para cada registro alterado pelo usuário. A entidade *TecsUsuario* sobrescreve este método para criptografar a senha informada pelo usuário antes dos dados serem salvos. Os parâmetros do método não foram descritos aqui por não serem relevantes ao entendimento de sua finalidade.

*ErroValidacao(psMsgErro: string; var psMsgResposta: string)*: este método deve ser sobrescrito quando se deseja tratar a mensagem de erro quando alguma operação inválida ocorrer. Quando um erro ocorre, na tentativa de salvar os dados, este método será chamado. O parâmetro *psMsgErro* contém a mensagem de erro informada pelo delphi, pode-se então interpretar a mensagem e substituí-la por uma mais amigável. Esta substituição é feita atribuindo-se algum valor ao parâmetro *psMsgResposta*.

#### 5.4.2.3 TcsEntidade

Classe abstrata que deve ser utilizada quando se quer implementar uma entidade que representa uma tabela no banco de dados. A implementação adicional desta classe em relação a TcsCustomEntidade está no relacionamento *Mestre/Detalhe* implementado pela classe e na publicação de algumas propriedades protegidas na classe-mãe.

A classe TcsEntidade possui uma propriedade denominada *csEntidadePai*, que tem o mesmo significado que uma entidade *Mestre* para uma entidade *Detalhe*. Quando se atribui uma entidade à propriedade *csEntidadePai*, esta passa a interagir com sua *entidade filha*. Quando se atribui uma *entidade pai* a uma *entidade filha*, a classe assegura-se que a entidade filha pertença a mesma *transação* da entidade pai. Transação em banco de dados é uma operação que quando iniciada grava *logs*

de todas as alterações feitas sobre os dados. Se a transação for confirmada, as alterações nos dados serão salvas, se for cancelada, os dados serão restaurados para os valores que continham quando a transação foi iniciada. Este é um mecanismo muito utilizado em banco de dados para assegurar que os dados sejam gravados apenas se todas as operações executadas forem bem sucedidas.

Existem componentes de acesso a dados, como é o caso do Interbase Express, que aceitam múltiplas transações. Cada componente de acesso a dados deve pertencer a uma transação. Quando se diz a uma entidade que possui uma entidade pai, a classe assegura que a entidade filha pertença a mesma transação da entidade pai. Desta forma, os dados de ambas as entidades só serão gravados se a operação de gravação for bem sucedida em ambas. O processo funciona mais ou menos assim: quando o método que salva os dados da entidade pai for executado, uma transação será iniciada. A entidade pai salvará primeiramente seus dados e depois os de suas filhas, um a um. Se um erro ocorrer na tentativa de salvar os dados de qualquer uma das entidades, pai ou filhas, a transação ficará pendente, aguardando a decisão do usuário em corrigir o problema e tentar novamente o salvamento dos dados ou cancelar a operação. Se uma nova tentativa de salvamento for executada e todas as entidades conseguirem gravar seus dados, a transação é confirmada. Se o usuário decidiu cancelar a operação, a transação é cancelada e os dados de todas entidades serão restaurados para os valores que continham quando a transação foi iniciada.

#### 5.4.2.4 TecsQuery

Classe concreta implementada com a finalidade de dar suporte a execução de sqls que são implementados dinamicamente. Ela possui uma propriedade pública denominada *SQL*, que possibilita a atribuição dinâmica do comando sql que se deseja executar. Quando um valor é atribuído a esta propriedade, ele é repassado ao componente de acesso a dados vinculados ao TDataSetProvider da classe. Esta classe pode ser usada, por exemplo, para executar um sql adicional em uma operação interna de uma entidade. Se esta classe não existisse, a única forma de executar selects seria criando-se uma entidade e registrando o select desejado.

#### 5.4.2.5 TcsAcessoDados

Classe abstrata agregada a entidade e que é responsável pela criação do componente de acesso a dados que será vinculado ao TDataSetProvider da entidade. É através da interface desta classe que a entidade interage com este componente. Como o componente de acesso a dados pode variar, é necessário que se tenha uma classe abstrata que implemente a interface de comunicação entre ele e a entidade. Por exemplo, quando se registra um comando sql na entidade, deve-se repassá-lo para o componente de acesso a dados para que o mesmo possa executá-lo. A classe oferece métodos virtuais que devem ser sobrescritos nas subclasses para um correto funcionamento.

#### 5.4.2.6 TcsAcessoDadosIB

Classe concreta que utiliza os componentes de acesso a dados Interbase Express para implementação do protocolo de acesso ao banco. No presente trabalho, foi implementado apenas uma classe concreta para acesso a dados, a TcsAcessoDadosIB. Caso se deseje utilizar outros tipos de componentes de acesso a dados, deve-se implementar uma subclasse da classe TcsAcessoDados que implemente o novo tipo de componente.

Como foi dito no item anterior, a classe TcsAcessoDados oferece métodos virtuais que devem ser sobrescritos pelas subclasses para um funcionamento correto, e entre estes métodos os mais importantes são os métodos *ObtemCamposTabela* e *AtualizaUpdateSql*.

O método *ObtemCamposTabela* é responsável por obter a chave primária e os campos da tabela implementada pela entidade. O método *AtualizaUpdateSql* é responsável pela geração dos comandos sql de inserção, atualização e exclusão, que serão utilizados quando o método que salva os dados da entidade for executado. Suponhamos que o select abaixo seja implementado na entidade:

```
Select  u.*, g.nmGrupoUsuario, g.deGrupoUsuaio
from ecsUsuario u, ecsGrupoUsuario g
where u.idGrupoUsuario = g.idGrupoUsuario
```

Se os dados exibidos pela entidade forem os trazidos pelo select acima, alterações em seus dados não poderão ser salvas, pois existem campos de duas tabelas diferentes. Não é possível gravar dados quando existem junções entre tabelas. Para resolver este problema, os comandos de inserção, atualização e exclusão são utilizados, baseados na tabela que se está utilizando, em sua chave primária e nos campos que a compõem, gerando assim, comandos que irão afetar somente a tabela desejada. No caso da classe em questão, a classe TIBUpdateSQL fornecida pelo Delphi é utilizada para que se possa salvar alterações com o sql acima. Os comandos selects calculados serão atribuídos a esta classe para que a operação possa ser realizada. Se os componentes de acesso a dados fossem da BDE, a classe TUpdateSql seria utilizada. Cada tipo de componente de acesso tem seu componente específico para suplantar este problema, por isto os comandos sqls de atualização são implementados na classe que implementa o componente de acesso.

#### 5.4.2.7 TsTrataMsgErro

Esta classe tem a finalidade de tratar as mensagens de erros geradas pelo aplicativo. Se alguma exceção for gerada e não for tratada pelo desenvolvedor, ela irá passar por esta classe, que irá tratar a mensagem de erro antes de exibi-la ao usuário, bem como utilizar a unit *ucsMsgDlg*, discutida no módulo csComponentes, para exibir a mensagem em seu quadro de diálogo de erro. Esta classe trabalha em conjunto com a unit *ucsMensagens*, utilizando mensagens já definidas nela para serem exibidas ao usuário.

A figura 5.5 ilustra dois quadros de diálogo de erro. O quadro de diálogo a esquerda exibe uma mensagem de erro gerada pelo aplicativo, sem qualquer tratamento, já o quadro de diálogo a direita exibe a mesma mensagem de erro tratada pela classe TcsTrataMsgErro.

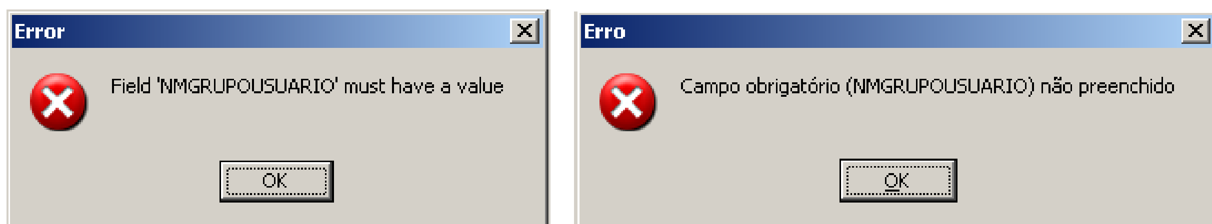


Figura 5.5 – Mensagem de erro tratada pela classe TcsTrataMsgErro

#### 5.4.2.8 ucsMensagens

Esta unit tem a finalidade de padronizar as mensagens exibidas ao usuário nos quadros de diálogo. Digamos que existam dois programadores trabalhando no desenvolvimento do mesmo software. No caso de um campo obrigatório não ser preenchido pelo usuário antes de salvar os dados, um programador poderia escrever a mensagem “Você não preencheu o campo X” e o outro poderia escrever “Digite um valor no campo X”. Agora imagine se existissem dez programadores trabalhando no mesmo software, inúmeras mensagens diferentes seriam exibidas para tratar o mesmo problema.

A unit ucsMensagens foi criada para resolver este problema. Nela são cadastradas mensagens a serem exibidas ao usuário, não só mensagens de erro, mas também mensagens de aviso, confirmação e informação. Assim, quando um desenvolvedor deseja exibir uma mensagem para informar ao usuário que um campo obrigatório não foi preenchido, ele irá utilizar a mensagem já definida pela unit. Desta forma, todas as mensagens relativas ao mesmo problema serão iguais. Isto facilita ao usuário interpretar a mensagens exibidas mais rapidamente, pois após ler uma vez uma mensagem de campo obrigatório não preenchido, ele irá interpretar esta mensagem quase que automaticamente na próxima vez que for exibida, pois está acostumado com o padrão. Caso a mensagem mudasse de acordo com o programador que a digitou, o usuário teria que ler com mais atenção o quadro de diálogo para ter certeza do que se trata, pois para o mesmo erro ele está vendo uma mensagem nova. O desenvolvedor também ganha com isto, pois não precisa ficar pensando em que mensagem escrever caso já exista uma mensagem cadastrada na unit, este raciocínio só é feito uma vez para cada tipo de mensagem a ser exibida.

### 5.4.3 Módulo csAplicação

É no módulo csAplicação que as janelas do aplicativo são construídas. Este módulo é dependente dos componentes do módulo csComponentes e das entidades do módulo csEntidades.

O módulo csAplicação fornece um esqueleto-base sobre o qual a aplicação será construída. O grande benefício deste módulo para o desenvolvedor de software

que está utilizando o framework é a reutilização de código e projeto, que tem por intuito diminuir o tempo e esforços dedicados no desenvolvimento de software. Este módulo estabelece padrões estruturais e de interface, liberando o desenvolvedor desta preocupação.

O módulo csAplicação fornece um conjunto de classes abstratas que o desenvolvedor pode herdar para implementar as janelas que comporão o aplicativo. Nos tópicos a seguir algumas destas janelas serão explicadas bem como a finalidade de cada uma. O framework não oferece somente classes abstratas de janelas, existem classes concretas, implementadas utilizando-se das classes abstratas fornecidas pelo framework, que estão prontas para serem utilizadas nos aplicativos, como é o caso das classes de janelas para cadastro de usuário e grupos de usuários. Aqui os benefícios que a reutilização de código e projeto trazem ao desenvolvimento de software ficarão mais evidentes.

Por fim, será discutido como a segurança foi implementada no módulo csAplicação. Caso o desenvolvedor deseje, o módulo csAplicação fornece suporte ao controle de usuários que acessam o aplicativo e mecanismos para que se possa configurar permissões de acesso às janelas do mesmo.

#### 5.4.3.1 Estrutura hierárquica das janelas do módulo csAplicação

A figura 5.6 ilustra a estrutura hierárquica das classes que compõem as janelas que o framework oferece ao desenvolvedor que o utiliza na criação das janelas que irão compor o software construído sobre ele. No topo da hierarquia vemos a classe *TForm*, esta classe é uma classe do ambiente de desenvolvimento Delphi e é a classe que implementa um formulário neste ambiente. Todas as janelas do framework terão esta classe como mãe.

A hierarquia de classes ilustrada na figura 5.6 tem duas divisões significativas. Elas estão representadas pelas classes *TfcsPrincipal* e *TfcsForm*. A classe TfcsPrincipal é a janela que deve ser herdada para se implementar a janela principal do aplicativo. Esta é a janela por onde as demais janelas serão chamadas pelo usuário. A classe TfcsForm representa as janelas que serão abertas pela janela principal. Todas as janelas de cadastros, consultas, relatórios ou outro tipo qualquer será uma subclasse da classe TfcsForm.

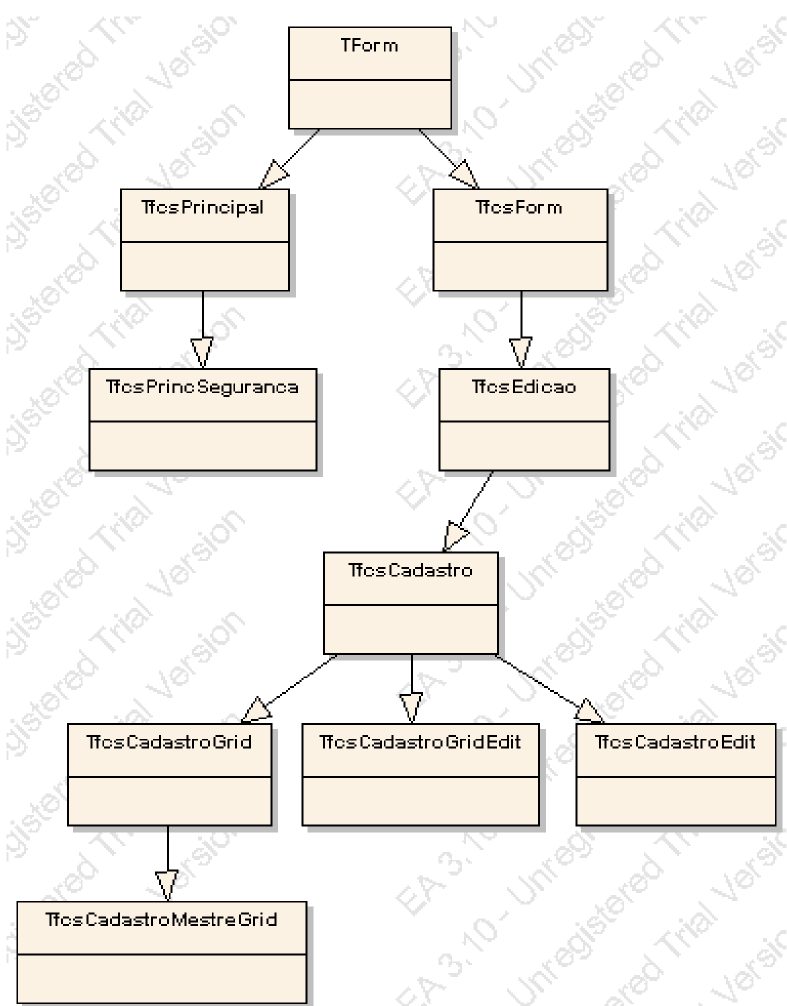

Figura 5.6 – Hierarquia de janelas do módulo csAplicação

Existe um forte relacionamento entre estas duas classes. A janela TfcsPrincipal tem controle sobre as janelas que são abertas a partir dela. Poderíamos dizer que estas seriam suas “janelas filhas”. Este será o nome utilizado no presente trabalho para referenciar as janelas abertas a partir da janela principal.

Os tópicos subseqüentes irão abordar cada uma das classes da hierarquia de janelas ilustradas pela figura 5.6.

#### 5.4.3.1.1 TfcsPrincipal

A janela principal do aplicativo que se está construindo sobre o framework deve ser herdada desta janela. Ela não possui nenhum componente em sua interface,

deixando a parte de seu designer a cargo do desenvolvedor. Esta classe implementa o controle das janelas, derivadas da classe TcsForm, que serão abertas por ela.

A classe TfcsPrincipal possui um método bastante utilizado pelo desenvolvedor do framework, o método *AbreForm(sNomeForm: string)*. Este é o método que deve ser chamado quando se deseja abrir uma janela filha no aplicativo. Para isto, basta informar o nome da janela filha no parâmetro *sNomeForm.* A classe irá verificar se a janela já está aberta, em caso afirmativo, ela será trazida à frente de todas as demais janelas filhas abertas. Em caso negativo, a classe criará uma instância da janela passada por parâmetro e irá abri-la.

#### 5.4.3.1.2 TfcsPrincSeguranca

A janela principal do aplicativo deve ser herdada desta janela quando se deseja que o aplicativo tenha um controle de segurança. Se a janela principal for herdada da classe TfcsPrincSeguranca, o aplicativo exibirá uma janela inicial, onde o usuário deve digitar um login e uma senha para que possa acessar o aplicativo, para isto, ele deve ser um usuário cadastrado no sistema.

Quando se utiliza o controle de segurança, o framework oferece ao desenvolvedor recursos para que ele possa configurar as permissões de acesso para cada janela filha do aplicativo, como por exemplo, se o usuário pode acessar determinada janela, ou se em uma janela de cadastro ele tem permissão para inserir, alterar ou excluir registros. O controle de segurança oferecido pelo framework será abordado em detalhes no tópico Segurança.

#### 5.4.3.1.3 TfcsForm

Esta é a classe mãe de todas as classes das janelas filhas do aplicativo. Ela possui um forte relacionamento com a classe TfcsPrincipal. Esta janela não possui componentes em sua interface, estando no topo da hierarquia das janelas filhas do aplicativo, deve deixar sua interface totalmente limpa para que outras janelas, herdadas a partir dela, possam definir suas interfaces particulares.

A parte principal de seu código está relacionada a segurança. É nesta janela que as configurações de permissão dos componentes visuais das janelas serão lidas e aplicadas. É fácil perceber porque esta parte da implementação foi posta aqui.

Estando no topo da hierarquia das janelas filhas, todas as janelas filhas do aplicativo terão controle de permissão em seus componentes visuais.

Configurações de permissão para componentes nas janelas filhas do framework são discutidas no tópico Segurança.

#### 5.4.3.1.4 TfcsEdicao

Esta janela não possui nenhuma implementação. Sua finalidade é estabelecer um padrão de interface para os botões principais que as janelas terão. Sua interface possui um componente csBotaoCadastro, cuja finalidade é fechar a janela quando clicado. Este componente está presente em todas as janelas do aplicativo. A figura 5.7 ilustra a janela TfcsEdicao.

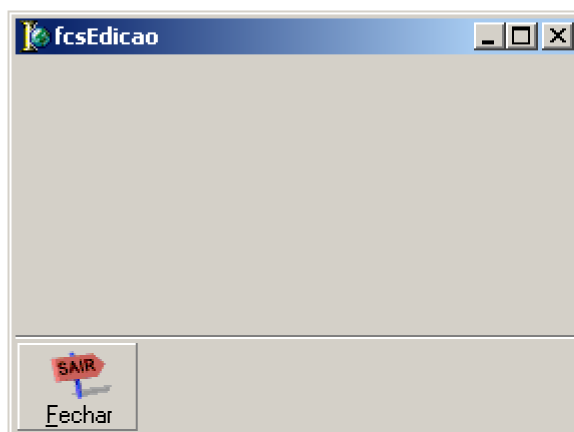


Figura 5.7 – Janela TfcsEdicao

#### 5.4.3.1.5 TfcsCadastro

Esta é a janela mais importante da hierarquia de janelas do framework. Em um aplicativo de característica cliente/servidor, janelas de cadastro são as janelas mais importantes. Esta classe implementa a maior parte do suporte a cadastro que as janelas oferecidas pelo framework precisam para dar suporte a esta característica. Todas as janelas cuja finalidade é cadastrar dados herdarão desta janela. A classe TfcsCadastro oferece vários métodos virtuais que podem ser sobrescritos nas subclasses para oferecer tratamento diferenciado na forma como a classe trabalha com os dados a serem cadastrados nela.

A figura 5.8 ilustra a janela TfcsCadastro. Ela possui três componentes a mais em sua interface em relação à janela TfcsEdicao, dois botões, cujas finalidades são salvar os dados alterados na janela e restaurar os dados alterados para o valor que possuíam no último salvamento e um componente *DataSource*, um componente da VCL[3] do Delphi, cuja finalidade é ser o elo de ligação entre um conjunto de dados (Entidade no nosso caso) e os componentes de visualização de dados (um grade de dados por exemplo).

Como dito anteriormente, todas as janelas de cadastro herdarão desta janela, por isto os botões *Salvar* e *Restaurar* foram postos aqui, pois todo cadastro terá estas operações. A entidade principal da janela (entidade pai de todas as outras), deve ser vinculada ao componente *DataSource* da janela. É desta forma que a classe sabe qual das entidades da janela é a principal.

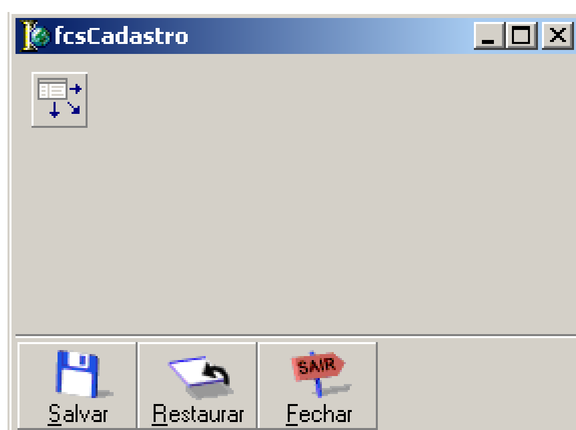


Figura 5.8 – Janela TfcsCadastro

Quando o botão salvar é pressionado, a classe chama o método responsável pelo salvamento dos dados da entidade principal. Se esta possui entidades filhas, chama o respectivo método de cada uma delas. É assim que funciona, por exemplo, a operação que salva os dados alterados na janela. Ele baseia-se na entidade principal.

Também é nesta janela que as permissões de inserção, alteração e exclusão atribuídas às janelas de cadastro serão lidas e aplicadas. Sendo operações relativas a cadastro, foram aqui implementadas devido a esta ser janela mãe de todas as

[3] VCL significa Visual Componente Library, é a biblioteca de componentes visuais do Delphi.

janelas de cadastro oferecidas pelo framework. Estas permissões serão discutidas em detalhes no tópico Segurança.

### 5.4.3.1.6 TfcsCadastroGrid

Este tipo de janela deverá ser utilizado para cadastros simples que possuírem pouca quantidade de registros. Todos os dados devem ser visualizados na janela sem ser necessário realizar rolagem horizontal dos mesmos. Caso isso não possa ser garantido deve-se utilizar os outros tipos de janelas. O cadastro (inserção, alteração e exclusão) dos dados é realizado na própria *Grid*.

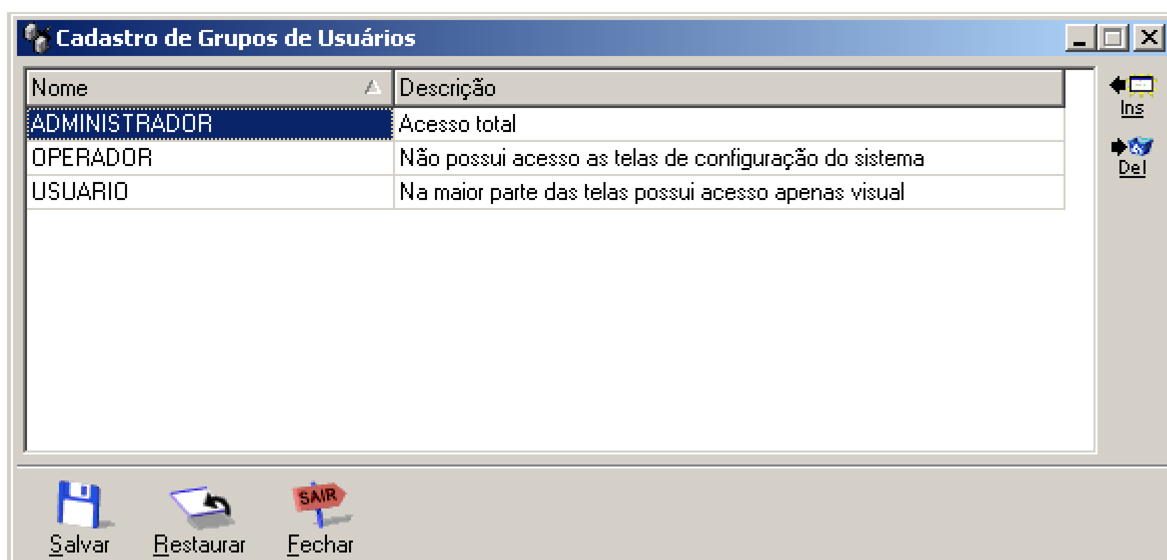


Figura 5.9 – Janela TfsCadastroGrid

A interface apresenta uma *Grid* onde os dados são visualizados e cadastrados. Se o usuário clicar na coluna da Grid, os dados serão ordenados por ela de forma ascendente, se um novo click for dado na mesma coluna, serão ordenados de forma descendente.

### 5.4.3.1.7 TfcsCadastroEdit

Este tipo de cadastro deve ser utilizado quando se deseja cadastrar uma informação por vez. Este tipo de cadastro é útil quando o volume de dados da entidade for muito grande ou quando o tempo de resposta dos selects for grande.

Para que o registro que se deseja trabalhar possa ser visualizado na janela, deve-se fornecer recursos para que o usuário possa encontrá-lo, como por exemplo, uma pesquisa onde o usuário, após preencher alguns campos para filtragem dos dados a serem pesquisados, selecionaria o registro de seu interesse e este seria exibido na janela para que o mesmo pudesse alterá-lo.

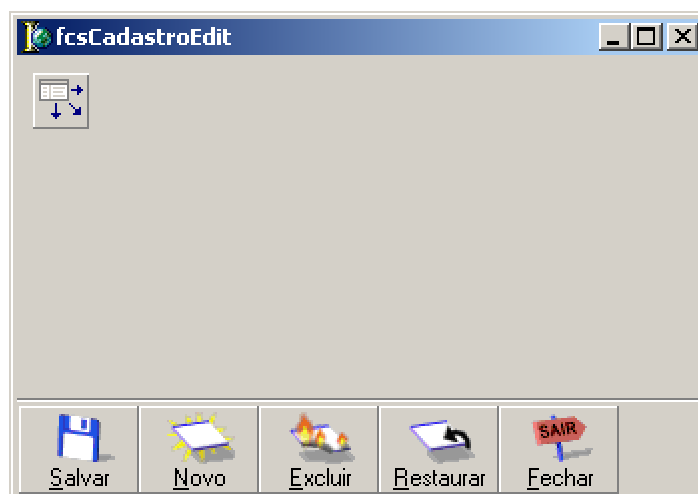


Figura 5.10 – Janela TfsCadastroEdit

**5.4.3.1.8 TfcsCadastroGridEdit**

Esta janela deve ser utilizada para os cadastros de tabelas onde o número de campos da tabela provoca o aparecimento da barra de scroll horizontal, não sendo possível utilizar a janela TfcsCadastroGrid. Também pode ser utilizada quando um determinado campo exige um componente ou tratamento mais complexo para ser cadastrado, não sendo possível fazê-lo na Grid.

Esta janela de cadastro é composta de duas pastas: uma contendo a Grid, com funcionamento idêntico a janela TcsCadastroGrid, e outra com todos os campos disponíveis no formato de formulário, como a classe TcsCadastroEdit. Fica a critério do desenvolvedor decidir quais campos serão apresentados na Grid. Geralmente todos os campos devem ser apresentados na pasta de formulário. Todos os campos que aparecem na Grid devem aparecer na pasta Formulário.

A figura 5.11 ilustra a pasta Tabela da janela TfcsCadastroGridEdit e a figura 5.12 ilustra a pasta Formulário da janela TfcsCadastroGridEdit.

#### 5.4.3.1.9 TfcsCadastroMestreGrid

Este cadastro é utilizado para o cadastramento de duas tabelas do banco de dados, com uma relação de um para muitos entre elas. Tanto a tabela pai quanto à tabela filha são apresentadas em grid.

A grid superior contém os dados da tabela pai e a inferior os da filha. A grid da tabela filha mostrará todas as informações relativas ao registro corrente apresentado na tabela pai. A movimentação do registro corrente da tabela pai faz com que a grid filha é atualizada com as informações correspondentes.

A figura 5.13 ilustra a janela TfcsCadastroMestreGrid.

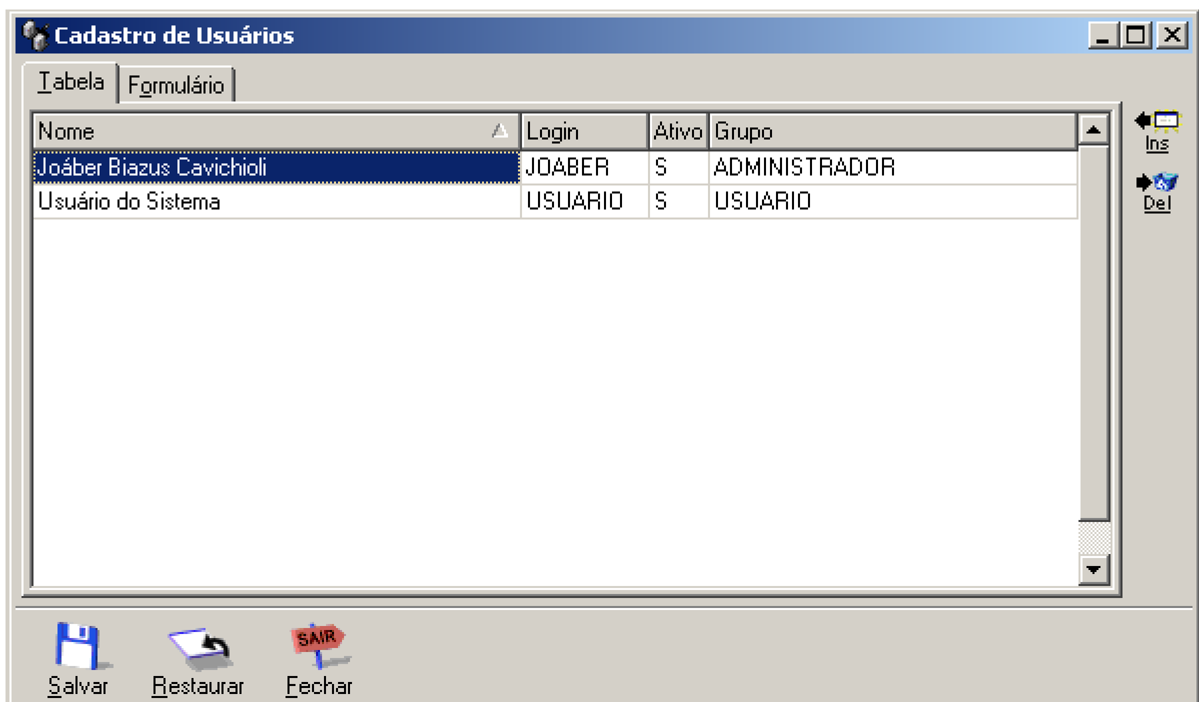


Figura 5.11 – Janela TfsCadastroGridEdit – pasta Tabela

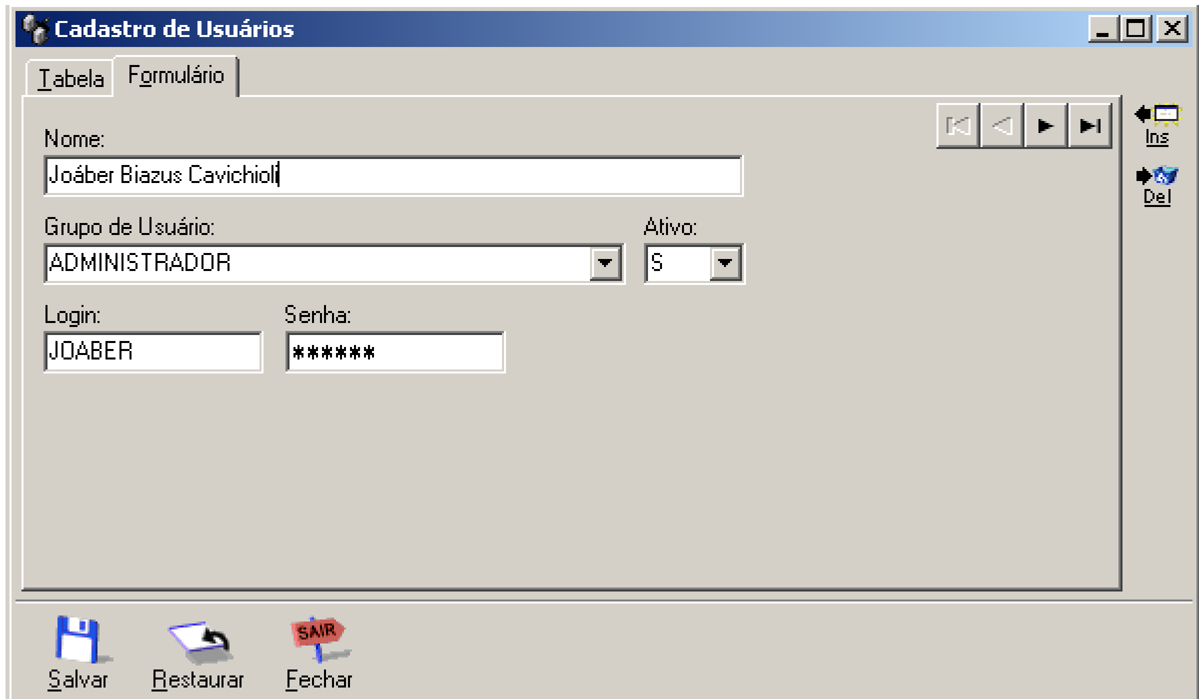


Figura 5.12 – Janela TfsCadastroGridEdit – pasta Formulário

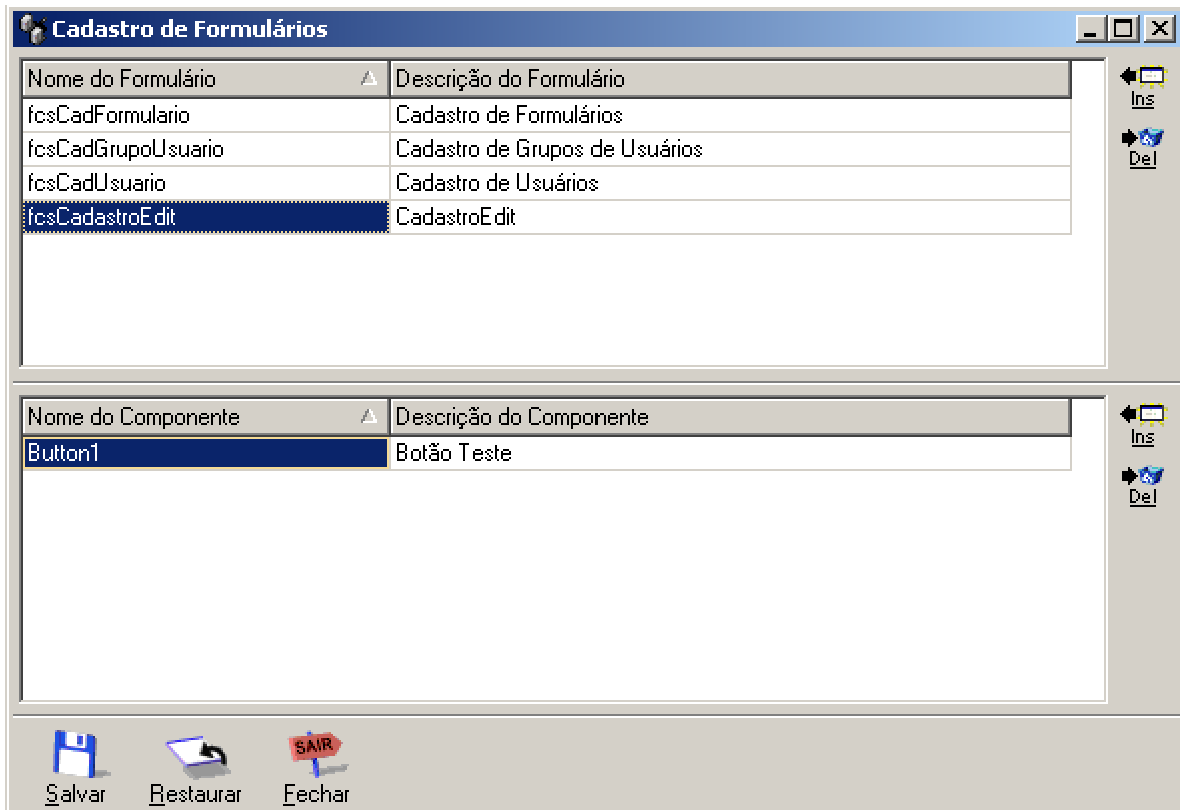


Figura 5.13 – Janela TfsCadastroMestreGrid

#### 5.4.3.2 Segurança

A segurança atua na restrição de ações passíveis de autorização, configuráveis por grupos de usuários. As autorizações determinam quais janelas o usuário pode acessar e quais operações de registro ele pode realizar (inserção, alteração e exclusão). Componentes pertencentes às janelas podem ser programados para sofrerem segurança também. Neste caso, deve haver intervenção do programador do aplicativo para habilitar esse tipo de segurança.

Existem dois elementos chaves na segurança do framework, o *Grupo de Usuários* e o *Usuário*. Todas as permissões configuradas no framework são personalizáveis para cada Grupo de Usuários. Para se ter acesso ao aplicativo implementado sobre o framework, o usuário deve estar cadastrado no mesmo, pois na inicialização do aplicativo será pedido um login e senha para que se possa ter acesso a ele.

Quando se cadastra um novo usuário, deve-se dizer a que Grupo de Usuários ele pertence. As permissões do usuário serão as permissões configuradas pelo Grupo de Usuário ao qual ele pertence. Grupos de Usuários podem ter muitos usuários, mas um usuário só pode pertencer a um Grupo de Usuário.

Quando um usuário consegue acesso ao aplicativo, a classe *TcsSeguranca* vai ao banco de dados e busca todas as permissões do Grupo de Usuário ao qual o usuário pertence. A partir disto, suas permissões serão verificadas em todas as telas que o mesmo tentará acessar.

A classe *TcsSeguranca* é a classe que controla toda a segurança do aplicativo. Esta classe foi implementada utilizando-se o padrão Fachada. Ela trabalha em conjunto com outras classes, oferecendo uma interface única que o aplicativo irá acessar para interagir com a segurança oferecida pelo framework.

A figura 5.14 ilustra as classes que compõem a segurança oferecida pelo framework.

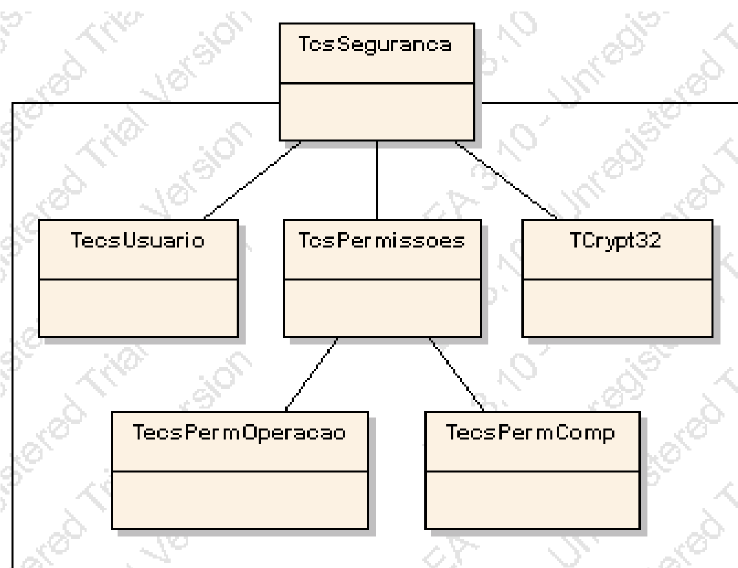


Figura 5.14 – Classes que compõem a Segurança

A classe *TecsUsuario* é uma entidade cuja finalidade é verificar se o login e senha digitados pelo usuário são válidos. Em caso afirmativo, ela informa para a TcsSeguranca os dados do usuário e do grupo ao qual ele pertence.

A classe *TcsPermissoes* é classe que informa a classe TcsSeguranca as permissões de visualização de telas, operações de registro e componentes do grupo de usuário desejado. Ela trabalha em conjunto com as classes *TecsPermOperacao* e *TecsPermComp*, duas entidades, cujas finalidades são buscar, respectivamente, as configurações de permissão para operações e componentes no banco de dados para serem utilizadas pela classe TcsPermissoes.

A classe *TCrypt32* é uma classe de criptografia. Ela oferece a classe TcsSeguranca métodos para criptografar e descriptografar textos. É utilizada no framework para criptografar a senha do usuário antes de ser armazenada no banco de dados.

#### 5.4.3.2.1 Cadastro de Janelas e seus Componentes

Para que se possa configurar permissões para as janelas do aplicativo desenvolvido sobre o framework, elas devem ser cadastradas no mesmo. Para isto, ele oferece uma janela de cadastro já implementada, onde o desenvolvedor deve cadastrá-las para que o usuário possa ter acesso à configuração de suas

permissões. O desenvolvedor não é obrigado a cadastrar todas as janelas, fica a seu critério escolher quais janelas podem ter suas permissões configuradas pelo usuário.

Nesta janela de cadastro, também é possível cadastrar, para cada janela, componentes[4] que se deseja aplicar permissões de acesso. Um componente pode possuir três tipos de permissões, *habilitado*, *desabilitado* e *invisível*. Se o componente for configurado como habilitado, o usuário terá acesso a ele para executar a operação a qual o componente se propõe. Se estiver como desabilitado, ele será visível ao usuário, porém, nenhuma operação pode ser executada nele. Se estiver configurado como invisível, o componente não será visível ao usuário. A figura 5.13 ilustra a tela de cadastro de janelas (formulários no delphi).

#### 5.4.3.2.2 Configuração das Permissões

O framework oferece uma janela já implementada onde o usuário pode configurar as permissões das janelas cadastradas pelo desenvolvedor. É através desta janela que o administrador do sistema pode configurar as permissões de seus usuários. A figura 5.15 ilustra a janela de Configuração de Permissões oferecida pelo framework.

No canto superior esquerdo da janela existe uma grid que mostra todos os grupos de usuários cadastrados. As configurações presentes no lado direito da janela são referentes ao grupo selecionado nesta grid. Abaixo da grid de grupos de usuários está a grid de formulários. Nelas todos os formulários cadastrados pelo desenvolvedor estão listados. Para se configurar uma determinada janela, deve-se selecionar o grupo de usuário desejado, a janela desejada e então alterar os dados contidos na parte direita da janela de configurações.

A parte superior direita da janela oferece opções para se configurar se a janela será visível e se as operações de inserção, edição e exclusão poderão ser executadas. Se o CheckBox *Visualizar* estiver desmarcado, nenhum dos usuários do grupo selecionado terá acesso a esta janela. Se o CheckBox *Inserir* estiver desmarcado, nenhum usuário do grupo selecionado poderá inserir novos registros na janela selecionada. Se o CheckBox *Editar* estiver desmarcado, a janela será somente

---

[4] Componentes aqui se referem aos objetos visuais contidos em uma janela, como botões e caixas de edição.

leitura, os usuários do grupo selecionado não poderão efetuar quaisquer alterações sobre seus dados. Finalmente, se o CheckBox *Excluir* estiver desmarcado, nenhum usuário do grupo selecionado poderá excluir registros na janela selecionada.

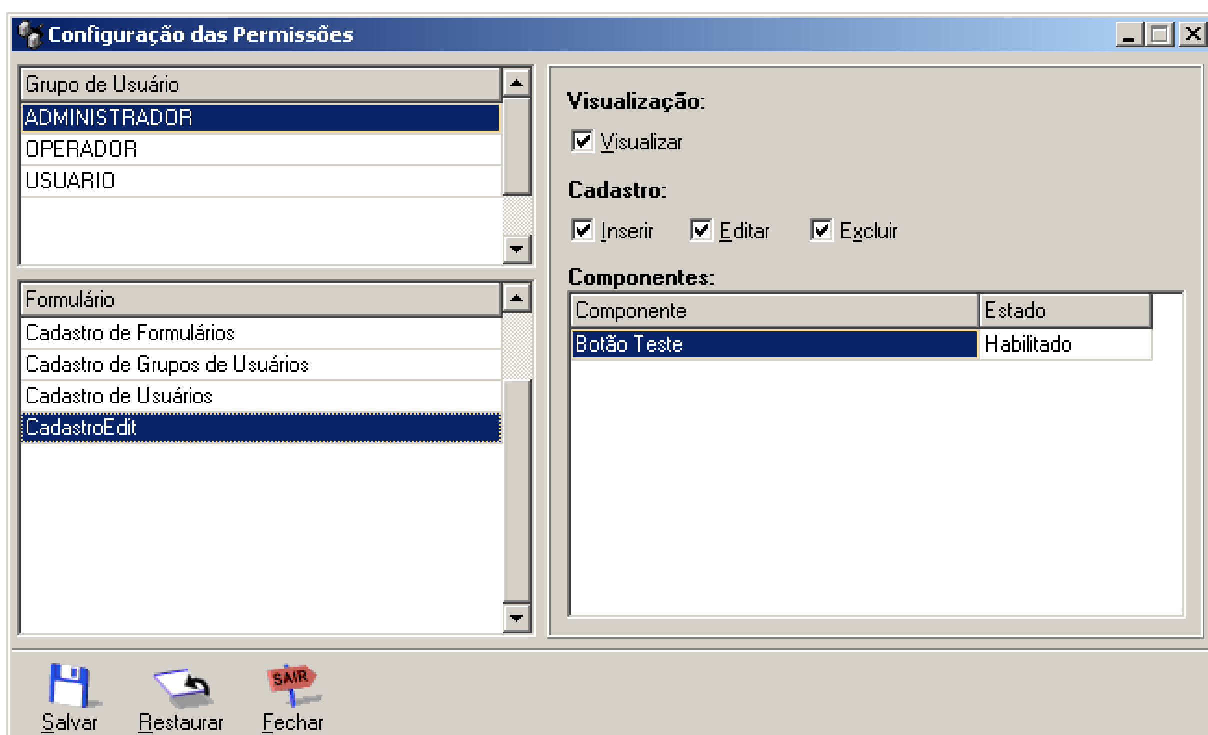


Figura 5.15 – Janela de configuração de permissões

No canto inferior direito está a grid de componentes. Nela estão listados todos os componentes da janela selecionada cadastrados pelo desenvolvedor. Na coluna *Estado*, deve-se informar se o componente estará habilitado, desabilitado ou invisível para os usuários do grupo selecionado.

# Capítulo 6

# Conclusão

O presente trabalho mostra a importância da reutilização no desenvolvimento de artefatos de software. Neste contexto, a abordagem de frameworks contribui significativamente neste processo. Frameworks é uma abordagem que está sendo cada vez mais utilizada, com o intuito de diminuir o tempo e esforços no desenvolvimento de artefatos de software, baseado em sua característica de reutilizar código e projeto.

Para se ter um bom projeto, este deve ser bem estruturado e planejado. Padrões de projeto ajudam o projetista no desenvolvimento de projetos melhor estruturados, oferecendo soluções que foram desenvolvidas e aperfeiçoadas ao longo do tempo, fazendo com que o projetista obtenha um projeto "certo" mais rápido.

O objetivo de criar um ambiente flexível, de fácil aprendizado e manipulação, que ajudasse a desenvolver softwares cliente/servidor de forma rápida, padronizada e consistente foi atingido com sucesso.

O framework desenvolvido possui uma grande flexibilidade e principalmente uma curva de aprendizado bastante pequena.

## Sugestões para Trabalhos Futuros

Muitas melhorias e novos recursos podem ser implementados no framework desenvolvido, entre as principais estão:

- Aprimorar o módulo csEntidades, oferecendo mais recursos, principalmente para o tratamento do relacionamento mestre/detalhe entre entidades.
- Criar novas classes para cadastro e principalmente classes para consulta.
- Implementar classes para geração de relatórios.

# Bibliografia


[ASW 98] AMBLER, Scott W. Análise e projeto orientado a objeto – Seu guia para desenvolver sistemas robustos com tecnologia de objetos. Tradução Oswaldo Zanelli. Rio de Janeiro: Infobook, 1998.

[BOO 94] BOOCH, G., 1994. Object-Oriented Analysis and Design. Redwood City, CA.: Benjamin/Cummings.

[CAM 00] CANTÚ, Marco. Dominando o Delphi 5 – a bíblia. Tradução João E.N. Tortello. São Paulo: Markon Books, 2000.

[DEU 89] DEUTSCH, L. Peter. Design reuse and frameworks in the SImalltalk-80 system. In Ted J. Biggerstaff and Alan T. Perlis, editors, Software Reusability, Volume II: Appplications and Experience, páginas 57-71. Addison-Wesley, Reading, MA, 1989.

[JF 88] JOHNSON, Ralph E., FOOTE, Brian. Designing reusable classes. Journal of Object-Oriented Programming, 1 (2):22-35, Junho/Julho 1988.

[JOH 92] JOHNSON, Ralph E. Documenting frameworks using patterns. SIGPLAN Notices, New York, v.27, n.10, Outubro 1992. Trabalho apresentado na OOPSLA, 1992.

[JZ 91] JOHNSON, Ralph E., ZWEIG Jonathan. Delegation in C++. Journal of Object-Oriented Programming, 4(11):22-35, Novembro 1991.

[GAM 02] GAMMA, Eric, HELM, Richard, JOHNSON, Ralph, VLISSIDES, John. Padrões de Projeto: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2002.

[GJO 00] GUIMARÃES, José O. Frameworks. Departamento de Computação UFSCar. São Carlos, SP, 2000.

[LAG 00] LARMAN, Craig. Utilizando UML e padrões: uma introdução à análise e ao projeto orientado a objetos. Tradução Luiz A. Meireles Salgado. Porto Alegre: Bookman, 2000.

[LIE 86] LIEBERMAN, Henry. Using prototypical objects to implement shared behavior in object-oriented systems. In Object-Oriented Programming Systems, Languages and Applications Conference Proceedings, páginas 214-223, Portland, OR, Novembro 1986.

[MAJ 94] MARTIN, James. Princípios de análise e projeto baseados em objetos. Tradução Cristina Bazán. Rio de Janeiro: Campus, 1994.

[KOR 98] KONOPKA, Ray. Desenvolvendo Componentes Personalizados em Delphi 3. Tradução Elisa M. Ferreira. São Paulo: Berkeley Brasil, 1998.

[SNY 86] SNYDER, Alan. Encapsulation and inheritance in object-oriented languages. In Object-Oriented Programming Systems, Languages and Applications Conference Proceedings, páginas 38-45, Portland, OR, Novembro 1986. ACM Press.

[SRP 00] SILVA, Ricardo P. Suporte ao desenvolvimento e uso de frameworks e componentes. Tese para a obtenção do grau de Doutor em Ciências da Computação. Universidade Federal do Rio Grande do Sul, UFRGS. Porto Alegre, 2000.

[SWE 85] SWEET, Richard E. The Mesa programming evironment. SIGPLAN Notices, 20(7):216-229, Julho 1985.

[WIA 91] WIRFS-BROCK, A. et al. Designing reusable designs: Experiences designing object-oriented frameworks. In: Object-Oriented Programming Systems, Languages and Applications Conference; European Conference on Object-Oriented Programming, 1991, Ottawa. Addendum to the procceddings… Ottawa: [s.n.], 1991.

[WIR 90] WIRFS-BROCK, R, JOHNSON, R. E. Surveying current research in object-oriented design. Communications of the ACM. V.33, n.9. sep. 1990.

# Anexo A – Artigo

## UM FRAMEWORK PARA DESENVOLVIMENTO DE APLICATIVOS CLIENTE/SERVIDOR

**Joáber Biazus Cavichioli**

Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
CEP 88040-900 – Florianópolis – SC
joaber@inf.ufsc.br

**RESUMO**

Este artigo apresenta uma visão geral da estrutura que compõe o framework para desenvolvimento de aplicativos cliente/servidor desenvolvido em meu trabalho de conclusão de curso. Será dada uma rápida introdução na abordagem de frameworks e padrões de projetos, justificando porque a utilização de padrões de projeto é tão importante no desenvolvimento de frameworks.

**ABSTRACT**

This article presents an overview of the structure which forms the client/server application development framework developed on my monograph. Both framework and design patterns overviews will be presented, justifying why the use of design pattners is so important on framework development.

## 1. Introdução

"Um framework é um conjunto de classes interrelacionadas com o objetivo de facilitar o desenvolvimento de um determinado domínio de aplicação. É composto de classes concretas e abstratas, estas últimas possuindo implementações incompletas que devem ser estendidas para compor as classes completas da aplicação final" [SRP 00].

Segundo Martin [MAJ 94], uma das preocupações mais urgentes na indústria da informática, hoje, é a necessidade de se criar softwares e sistemas corpora-tivos de modo muito mais veloz e a um baixo custo. É neste escopo que o desenvolvimento de frameworks se torna cada vez mais comum e necessário. A grande vantagem desta abordagem é a reutilização de código e projeto, que tem por intuito diminuir o tempo e o esforço no desenvolvimento de softwares. Esta abordagem tem como grande característica estabelecer padrões estruturais, de interface e código e

controlar o fluxo da aplicação, liberando o desenvolvedor de software desta preocupação e por conseqüência aumentando a produtividade no desenvolvimento de softwares.

Um dos principais objetivos de um framework é a reutilização de artefatos de software, em contraposição ao desenvolvimento de todas as partes do sistema, fator que leva a uma melhoria na qualidade e um aumento na produtividade do desenvolvimento do software, isto baseado na perspectiva de que reutilizando artefatos de software já desenvolvidos e depurados, haverá uma redução no tempo de desenvolvimento, testes e possibilidade de introdução de erros na produção de novos artefatos [SRP 00].

A reutilização neste nível leva a uma inversão de controle entre a aplicação e o software sobre a qual ela está baseada. Quando se utiliza uma biblioteca de funções, se escreve o corpo principal da aplicação e se chama o código que se quer reutilizar. Quando se usa um framework, se reutiliza o corpo principal e escreve-se o código que este chama. O desenvolvedor terá de escrever opera-ções com nomes e convenções de chamadas já especificadas; porém isto reduz as decisões de projeto que o mesmo tem que tomar [GAM 02]. A figura abaixo ilustra uma aplicação desenvolvida sob um framework, a área sombreada repre-senta as classes reutilizadas.

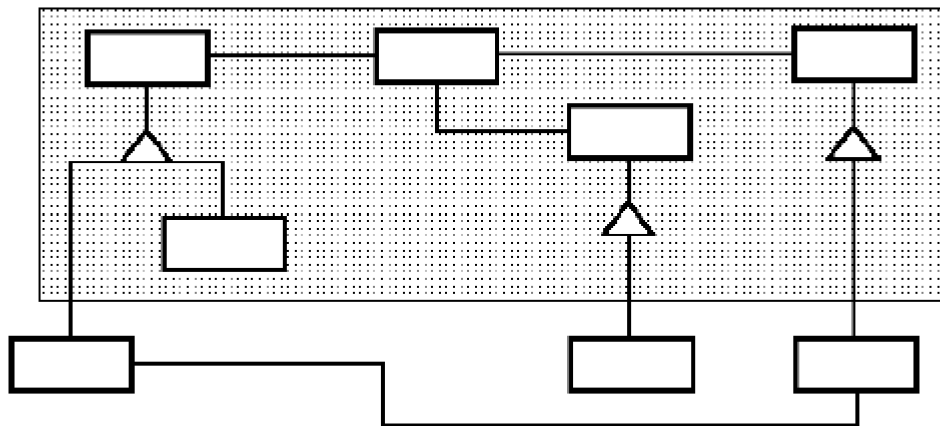

Figura 1 - Aplicação desenvolvida reutilizando um framework

Como resultado da reutilização da abordagem de frameworks, o desenvolve-dor pode não só construir aplicações mais rapidamente, como também as aplicações têm estruturas similares. Elas são mais fáceis de manter e parecem mais consistentes para seus usuários. Por outro lado, o desenvolvedor perde alguma liberdade criativa, uma vez que muitas decisões de projetos já foram tomadas por ele.

---

[1] A expressão *artefato de software* aqui se refere de forma genérica, não necessariamente a código, podendo se referir à aplicação, framework ou componente.

## 2. Padrões de Projeto

"Padrões de Projeto descrevem soluções simples para problemas específicos no projeto de software orientado a objetos, capturando soluções que foram desenvolvidas e aperfeiçoadas ao longo do tempo. Ele reflete modelagens e recodificações, nunca relatadas, resultantes dos esforços dos desenvolvedores por maior reutilização e flexibilidade em seus sistemas de software" [GAM 02].

Padrões de projetos descrevem soluções de projetos de software orientados a objetos de uma forma sucinta, clara e facilmente aplicável. Eles tornam mais fácil a reutilização de projetos e arquiteturas bem-sucedidas, pois descrevem técnicas já testadas e aprovadas. Eles ajudam a escolher alternativas de projeto, evitando alternativas que poderiam comprometer a reutilização.

## 3. Padrões de Projeto no Desenvolvimento de Frameworks

Um framework deve ser projetado de forma que sua arquitetura funcione para todas as aplicações do domínio tratado. As aplicações construídas sobre o framework são dependentes dele para seu projeto. Qualquer mudança substancial no projeto do framework reduz consideravelmente os benefícios que sua aborda-gem se propõe, uma vez que sua principal contribuição para uma aplicação é a arquitetura que ele define. À medida que um framework evolui, as aplicações devem evoluir com ele, e vice-versa. Portanto, é imperativo projetar o framework de maneira que ele seja tão flexível e extensível quanto possível [GAM 02].

O que foi discutido acima é o ponto mais crítico no projeto de um framework. Um framework projetado através do uso de padrões de projeto tem muito maior probabilidade de atingir altos níveis de reusabilidade de projeto e código, comparado com um que não usa padrões de projeto. Frameworks maduros comumente incorporam vários padrões de projeto. Os padrões ajudam a tornar a arquitetura do framework adequada a muitas aplicações diferentes, sem necessidade de reformulação [GAM 02].

Um benefício adicional também é ganho quando um framework é documen-tado com os padrões de projeto que ele usa. Pessoas que conhecem os padrões obtêm mais rapidamente uma compreensão do framework, isto é muito importante em frameworks, pois eles possuem uma curva de aprendizado acentuada, que deve ser percorrida antes que possa ser utilizado [GAM 02].

## 4. O Framework para Aplicativos Cliente/Servidor

Seu desenvolvimento foi motivado pela necessidade de um ambiente flexível, de fácil aprendizado e manipulação, que ajudasse a desenvolver softwares cliente/servidor de forma rápida, padronizada e consistente.

Seu principal atrativo em relação aos frameworks no mercado é sua simplicidade. Ele não é um framework complexo, que implementa todos os controles e detalhes que um aplicativo cliente/servidor possui. Não quer dizer que frameworks desta natureza não sejam bons, o problema é que frameworks assim possuem uma

curva de aprendizado bastante grande e muitas vezes amarram o desenvol-vedor que o utiliza a uma implementação própria. O framework desenvolvido, ao contrário, tem uma curva de aprendizado bastante pequena e pela sua simplicida-de é bastante flexível e extensível.

O framework desenvolvido está dividido em três módulos. Esta divisão possui duas finalidades: aumentar o desacoplamento das diferentes partes do framework, para que possam ser reutilizados em softwares que não são desenvolvidos sob o framework e separar claramente as diferentes camadas do framework, que são a camada de apresentação (telas do aplicativo), a camada de negócio (regras de negócio) e a camada de componentes (componentes e classes auxiliares).

Esses três módulos são: csComponentes, csEntidades e csAplicação.

## 4.1 Módulo csComponentes

O módulo csComponentes é o local onde os componentes e classes auxiliares do framework são implementados. Este módulo não depende dos demais módulos do framework, podendo ser usado em separado em um outro aplicativo que não utilize o framework.

Todos os componentes e classes que não dependam dos demais módulos devem ser implementados aqui, como por exemplo, componentes de botões e editores personalizados, classes para tratamento de strings e números.

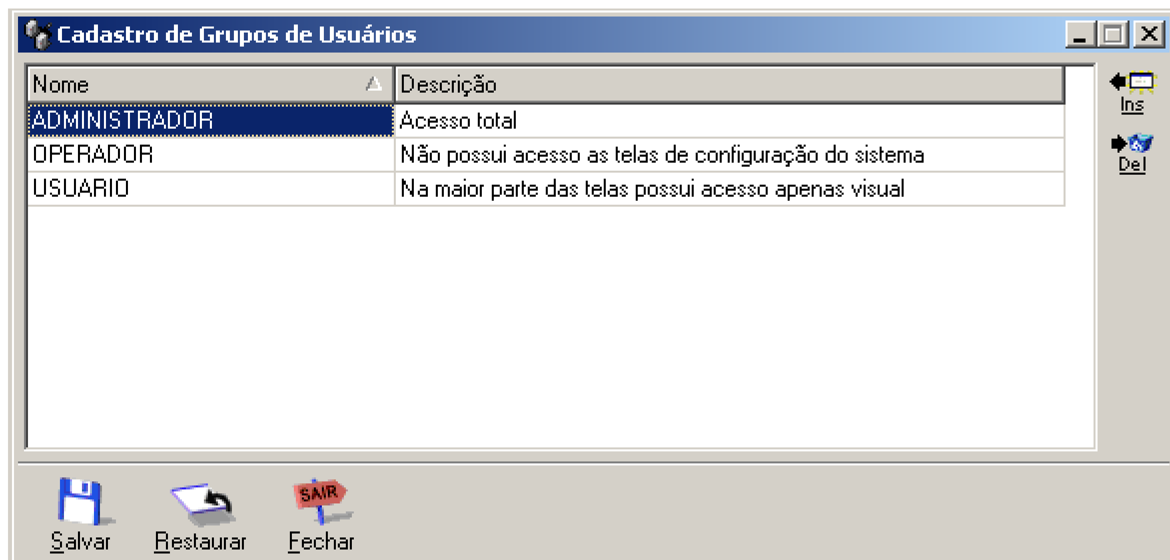


Figura 2 – Ilustração da utilização dos botões do módulo csComponentes

O framework desenvolvido utiliza muito de botões com figuras que represen-tam a operação a ser realizada quando o usuário clicar sobre ele. Foi então desenvolvido um conjunto de classes que possibilitam ao usuário do framework criar botões personalizados facilmente. Ele também oferece classes concretas que implementam botões personalizados que são utilizados nas telas do framework. Isto facilita muito o trabalho do desenvolvedor, pois não precisa ficar configurando as propriedades de um botão toda vez que o mesmo for posto em um formulário, além

de reduzir o tamanho do executável, pois as figuras presentes nos botões são lidas de um arquivo de recurso, sendo portanto compartilhadas por todos os botões, não incrementando o tamanho do arquivo executável quando um novo botão for adicionado.

A figura 2 ilustra três botões oferecidos pelo framework. Os três botões do canto inferior esquerdo da janela na verdade são botões da mesma classe, mudando apenas a figura selecionada pelo usuário. Os dois botões do canto superior direito da janela são os botões de “Inserção” e “Deleção” de registros, que possuem figura fixa e suas respectivas operações de Inserção e Deleção já implementadas.

### 4.2 Módulo csEntidades

O módulo csEntidades é o local onde as regras de negócio são implementadas. Ele foi criado para separar as regras de negócio da camada de apresentação. Um a Entidade no framework desenvolvido é uma classe, que será registrada como um componente na paleta de componentes do ambiente de desenvolvimento Delphi (ambiente utilizado para desenvolvimento do framework). Na maioria dos casos ela representa uma tabela do bando de dados do aplicativo cliente/servidor implementado sobre o framework, mas também pode incluir regras de negócios não vinculadas a apenas uma tabela do banco de dados. Assim, para que uma janela utilize esta entidade, basta selecioná-la na paleta de componentes do Delphi e jogá-la sobre a janela ou criá-la dinamicamente via código. Todas as regras de negócio da entidade estarão “dentro do componente”. Com isto, as regras de negócio implementadas na entidade podem ser utilizadas em vários lugares diferentes, o que as tornam totalmente separadas da camada de apresen-tação, favorecendo a reutilização e a modularização, visto que estão contidas em um único local.

### 4.3 Módulo csAplicação

O módulo csAplicação é o local onde as janelas do aplicativo desenvolvido sobre o framework são construídas. É um módulo totalmente dependente dos outros dois módulos, pois utiliza os componentes do módulo csComponentes em suas telas e das regras de negócio do módulo csEntidades.

O módulo csAplicação fornece um conjunto de classes abstratas que o desenvolvedor pode herdar para implementar as classes das janelas que comporão o aplicativo desenvolvido sob o framework. O framework não oferece somente classes abstratas de janelas no módulo csAplicação, existem classes concretas, implementadas utilizando-se das classes abstratas fornecidas pelo framework, que estão prontas para serem utilizadas nos aplicativos. Aqui, os benefícios que a reutilização de código e projeto trazem ao desenvolvimento de software são mais evidentes.

Este módulo também implementa a segurança do aplicativo. Segurança aqui se refere às permissões de acesso que o usuário do aplicativo podem ter, como por exemplo, se ele pode acessar determinada janela ou executar determinada operação. O framework oferece condições para que o desenvolvedor do software possa configurar quais janelas deseja que controle de permissões sejam aplicados e para cada janela que componentes visuais serão passíveis de controle de permissões de acesso. O usuário do aplicativo implementado sob o framework pode

então configurar estas permissões projetadas pelo desenvolvedor do aplicativo, de forma que o usuário que adminstra o sistema possa especificar quais usuários poderão ou não acessar determinada tela e seus componentes ou executar determinadas operações.

## 5. Conclusão

Este artigo mostra a importância da reutilização no desenvolvimento de artefatos de software. Neste contexto, a abordagem de frameworks contribui significativamente neste processo. Frameworks é uma abordagem que está sendo cada vez mais utilizada, com o intuito de diminuir o tempo e esforços no desenvolvimento de artefatos de software, baseado em sua característica de reutilizar código e projeto.

Para se ter um bom projeto, este deve ser bem estruturado e planejado. Padrões de projeto ajudam o projetista no desenvolvimento de projetos melhor estruturados, oferecendo soluções que foram desenvolvidas e aperfeiçoadas ao longo do tempo, fazendo com que o projetista obtenha um projeto “certo” mais rápido.

Foi dada uma visão geral do framework desenvolvido, que apesar de ser simples, sua utilização contribui em muito para o desenvolvimento de aplicativos melhor estruturados, organizados e em menor tempo.

## 6. Bibliografia


[GAM 02] GAMMA, Eric, HELM, Richard, JOHNSON, Ralph, VLISSIDES, John. Padrões de Projeto: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2002.

[MAJ 94] MARTIN, James. Princípios de análise e projeto baseados em objetos. Tradução Cristina Bazán. Rio de Janeiro: Campus, 1994.

[SRP 00] SILVA, Ricardo P. Suporte ao desenvolvimento e uso de frameworks e componentes. Tese para a obtenção do grau de Doutor em Ciências da Computação. Universidade Federal do Rio Grande do Sul, UFRGS. Porto Alegre, 2000.

# Anexo B – Código Fonte das Principais Classes

## Módulo Componentes

### unit ucsBotao;

```
unit ucsBotao;

interface

uses
  Windows, Messages, SysUtils, Classes, Controls, Buttons;

type
  TcsDimenssoes = class(TPersistent)
  private
    FComTexto: Boolean;
    FAlturaSemTexto: Word;
    FAlturaComTexto: Word;
    FLarguraSemTexto: Word;
    FLarguraComTexto: Word;
    FOnAtributosAlterados: TNotifyEvent;
    procedure AvisaAlteracao();
    procedure SetComTexto(const pbValor: Boolean);
    procedure SetAlturaComTexto(const pnValor: Word);
    procedure SetLarguraComTexto(const pnValor: Word);
    procedure SetAlturaSemTexto(const pnValor: Word);
    procedure SetLarguraSemTexto(const pnValor: Word);
  public
    {Indica um evento executado ao ser realizada alguma alteração nos atributos}
    property OnAtributosAlterados: TNotifyEvent read FOnAtributosAlterados
      write FOnAtributosAlterados;
  published
    { Indica se os botões de um componente TcsBotoes terão rótulo ou não}
    property csComTexto: Boolean read FComTexto write SetComTexto default True;
    { Indica a altura em pixels dos botões de um componente
      TcsBotoes quando a propriedade FComTexto é True}
    property csAlturaComTexto: Word read FAlturaComTexto
      write SetAlturaComTexto;
    { Indica a altura em pixels dos botões de um componente
      TcsBotoes quando a propriedade FComTexto é False. }
    property csAlturaSemTexto: Word read FAlturaSemTexto
      write SetAlturaSemTexto;
    { Indica a largura em pixels dos botões de um componente
      TcsBotoes quando a propriedade FComTexto é True. }
    property csLarguraComTexto: Word read FLarguraComTexto
      write SetLarguraComTexto;
    { Indica a largura em pixels dos botões de um componente
      TcsBotoes quando a propriedade FComTexto é False. }
    property csLarguraSemTexto: Word read FLarguraSemTexto
      write SetLarguraSemTexto;
  end;
```

```
TcsCustomBotao = class(TWinControl)
private
  { Private declarations }
  FSpeedButton: TSpeedButton;
  FDimenssoes: TcsDimenssoes;
  FFiguras: TStringList;
  FCaptions: TStringList;
  FHints: TStringList;
  FCaption: string;
  FCsEnabled: Boolean;
  FFigura: string;
  FFlat: Boolean;
  FHint: string;
  FNumGlyphs: Word;
  FShowHint: Boolean;
  FSpacing: Integer;
  FOnClick: TNotifyEvent;
  FOnMouseDown: TMouseEvent;
  FOnMouseMove: TMouseMoveEvent;
  FOnMouseUp: TMouseEvent;
  procedure AtributosAlterados(Sender: TObject);
  procedure SetCaption(psValor: string);
  procedure SetCsEnabled(pbValor: Boolean);
  procedure SetFigura(psValor: string);
  procedure SetFlat(pbValor: Boolean);
  procedure SetHint(psValor: string);
  procedure RaiseMetodoNaoImplementado(psNomeMetodo: string);
  function GetFigura(Index: Word): string;
  function GetFigurasCount(): Word;
  // Eventos
  procedure ExecutaOnMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
  procedure ExecutaMouseMove(Sender: TObject; Shift: TShiftState;
    X, Y: Integer);
  procedure ExecutaOnMouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
protected
  procedure Loaded(); override;
  procedure ExecutaOnClick(Sender: TObject); virtual;
  { Estes três métodos Sets foram incluindos em protected para que a classe
    herdade possa alterar a configuração padrão destas três propriedades. }
  procedure SetNumGlyphs(pnValor: Word);
  procedure SetShowHint(pbValor: Boolean);
  procedure SetSpacing(pnValor: Integer);
  { Sobreescrever. }
  procedure DimensoesDefaultBotoes(var pbComTexto: Boolean;
    var pnAlturaComTexto, pnLarguraComTexto, pnAlturaSemTexto,
    pnLarguraSemTexto: Word); virtual; abstract;
  procedure RegistraRecursosImagens(poFiguras: TStringList); virtual; abstract;
  procedure RegistraCaptions(poCaptions: TStringList); virtual; abstract;
  procedure RegistraHints(poHints: TStringList); virtual; abstract;
public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy(); override;
  property Figura[Index: Word]: string read GetFigura;
  property FigurasCount: Word read GetFigurasCount;
  // Propriedades customisáveis
  property csCaption: string read FCaption write SetCaption;
```

```
    property csDimenssoes: TcsDimenssoes read FDimenssoes write FDimenssoes;
    property csFigura: string read FFigura write SetFigura;
    property csNumGlyphs: Word read FNumGlyphs write SetNumGlyphs;
    property csSpacing: Integer read FSpacing write SetSpacing;
    // Eventos customisáveis
    property csOnClick: TNotifyEvent read FOnClick write FOnClick;
  published
    // Propriedades
    property csEnabled: Boolean read FCsEnabled write SetCsEnabled;
    property csFlat: Boolean read FFlat write SetFlat default True;
    property csHint: string read FHint write SetHint;
    property csShowHint: Boolean read FShowHint write SetShowHint;
    // Eventos
    property csOnMouseDown: TMouseEvent read FOnMouseDown write FOnMouseDown;
    property csOnMouseMove: TMouseMoveEvent read FOnMouseMove write FOnMouseMove;
    property csOnMouseUp: TMouseEvent read FOnMouseUp write FOnMouseUp;
    // Delphi
    property Anchors;
    property Visible;
  end;

  TcsBotao = class(TcsCustomBotao)
  published
    property csCaption;
    property csDimenssoes;
    property csFigura;
    property csNumGlyphs;
    property csSpacing;
    // Eventos
    property csOnClick;
  end;

  EMetodoNaoImplementado = class(Exception);
  EFigurasNaoAtribuidas = class(Exception);
  EImagemNaoExiste = class(Exception);
  ENumGlyphsIncorreto = class(Exception);
  ENumCaptionsIncorreto = class(Exception);

procedure Register;

implementation

uses
  ucsBotaoEditor, DesignIntf;

{ TcsBotao }

procedure Register;
begin
  RegisterPropertyEditor(TypeInfo(string), TcsBotao, 'csFigura',
    TFigurasProperty);
end;

constructor TcsCustomBotao.Create(AOwner: TComponent);
var
  nComTexto: Boolean;
  nAlturaComTexto, nLarguraComTexto, nAlturaSemTexto, nLarguraSemTexto: Word;
begin
```

```
inherited Create(AOwner);

try
  DimensoesDefaultBotoes(nComTexto, nAlturaComTexto, nLarguraComTexto,
    nAlturaSemTexto, nLarguraSemTexto);
except
  RaiseMetodoNaoImplementado('DimensoesDefaultBotoes');
end;

FDimenssoes := TcsDimenssoes.Create();
FDimenssoes.FComTexto := nComTexto;
FDimenssoes.FAlturaComTexto := nAlturaComTexto;
FDimenssoes.FAlturaSemTexto := nAlturaSemTexto;
FDimenssoes.FLarguraComTexto := nLarguraComTexto;
FDimenssoes.FLarguraSemTexto := nLarguraSemTexto;
FDimenssoes.OnAtributosAlterados := AtributosAlterados;

FSpeedButton := TSpeedButton.Create(Self);
FSpeedButton.Name := Name + 'Botao';
FSpeedButton.Transparent := True;
FSpeedButton.Layout := blGlyphTop;
FSpeedButton.OnClick := ExecutaOnClick;
FSpeedButton.OnMouseDown := ExecutaOnMouseDown;
FSpeedButton.OnMouseMove := ExecutaMouseMove;
FSpeedButton.OnMouseUp := ExecutaOnMouseUp;
InsertControl(FSpeedButton);

FFiguras := TStringList.Create();
try
  RegistraRecursosImagens(FFiguras);
except
  RaiseMetodoNaoImplementado('RegistraRecursosImagens');
end;

if FFiguras.Count = 0 then
  raise EFigurasNaoAtribuidas.Create('O método (RegistraRecursosImagens) não ' +
    'retornou nenhuma figura de recurso. ');

FCaptions := TStringList.Create();
try
  RegistraCaptions(FCaptions);
except
  RaiseMetodoNaoImplementado('RegistraCaptions');
end;

if FCaptions.Count <>  FFiguras.Count then
  raise ENumCaptionsIncorreto.Create('O número de captions registrados no ' +
    'método (RegistraCaptions) é diferente ' + #13 + 'do número de ' +
    'figuras registradas no método (RegistraRecursosImagens). ');


FHints := TStringList.Create();
try
  RegistraHints(FHints);
except
  RaiseMetodoNaoImplementado('RegistraHints');
end;
```

```
  if FHints.Count <>  FFiguras.Count then
    raise ENumCaptionsIncorreto.Create('O número de hints registrados no ' +
      'método (RegistraHints) é diferente ' + #13 + 'do número de ' +
      'figuras registradas no método (RegistraRecursosImagens). ');

  SetNumGlyphs(2);
  SetFigura(FFiguras.Strings[0]);
  SetFlat(True);
  SetHint(FHints.Strings[0]);
  SetShowHint(True);
  SetSpacing(-2);
  SetCsEnabled(True);
  if csDesigning in ComponentState then
    AtributosAlterados(Self);
end;

destructor TcsCustomBotao.Destroy();
begin
  FSpeedButton.Free();
  FDimenssoes.Free();
  FFiguras.Free();
  FCaptions.Free();
  FHints.Free();
  inherited;
end;

procedure TcsCustomBotao.Loaded();
begin
  inherited Loaded();
  AtributosAlterados(Self);
end;

procedure TcsCustomBotao.AtributosAlterados(Sender: TObject);
var
   nAltura, nLargura: Integer;
begin
   if FDimenssoes.FComTexto then begin
     FSpeedButton.Height := FDimenssoes.csAlturaComTexto;
     FSpeedButton.Width := FDimenssoes.csLarguraComTexto;
     nAltura := FDimenssoes.csAlturaComTexto;
     nLargura := FDimenssoes.csLarguraComTexto;
     FSpeedButton.Caption := FCaption;
   end
   else begin
     FSpeedButton.Height := FDimenssoes.csAlturaSemTexto;
     FSpeedButton.Width := FDimenssoes.csLarguraSemTexto;
     nAltura := FDimenssoes.csAlturaSemTexto;
     nLargura := FDimenssoes.csLarguraSemTexto;
     FSpeedButton.Caption := '';
   end;

  Constraints.MaxHeight := nAltura;
  Constraints.MinHeight := nAltura;
  Height := nAltura;
  Constraints.MaxWidth := nLargura;
  Constraints.MinWidth := nLargura;
  Width := nLargura;
end;
```

```
procedure TcsCustomBotao.SetFigura(psValor: string);
var
  nIndex: Integer;
begin
  if FFigura <> psValor then
  begin
    nIndex := FFiguras.IndexOf(psValor);
    if nIndex = -1 then
      Exit;

    FFigura := psValor;
    try
      FSpeedButton.Glyph.LoadFromResourceName(HInstance, FFigura);
      SetCaption(FCaptions.Strings[nIndex]);
      SetHint(FHints.Strings[nIndex]);
    except
      raise EImagemNaoExiste.Create('A imagem (' + FFigura + ') não existe ' +
        'no arquivo de recuros.');
    end;
  end;
end;

procedure TcsCustomBotao.SetCaption(psValor: string);
begin
  if FCaption <> psValor then
  begin
    FCaption := psValor;
    if FDimenssoes.csComTexto then
      FSpeedButton.Caption := FCaption;
  end;
end;

procedure TcsCustomBotao.SetCsEnabled(pbValor: Boolean);
begin
  if FCsEnabled <> pbValor then
  begin
    FCsEnabled := pbValor;
    Enabled := FCsEnabled;
    FSpeedButton.Enabled := FCsEnabled;
  end;
end;

procedure TcsCustomBotao.SetNumGlyphs(pnValor: Word);
begin
  if FNumGlyphs <> pnValor then
  begin
    if (pnValor = 0) or (pnValor > 4) then
      raise ENumGlyphsIncorreto.Create('O valor tem que estar entre 1 e 4');

    FNumGlyphs := pnValor;
    FSpeedButton.NumGlyphs := FNumGlyphs;
  end;
end;

procedure TcsCustomBotao.SetFlat(pbValor: Boolean);
begin
  if FFlat <> pbValor then
  begin
```

```
    FFlat := pbValor;
    FSpeedButton.Flat := FFlat;
  end;
end;

procedure TcsCustomBotao.SetHint(psValor: string);
begin
  if FHint <> psValor then
  begin
    FHint := psValor;
    FSpeedButton.Hint := FHint;
  end;
end;

procedure TcsCustomBotao.SetShowHint(pbValor: Boolean);
begin
  if FShowHint <> pbValor then
  begin
    FShowHint := pbValor;
    FSpeedButton.ShowHint := FShowHint;
  end;
end;

procedure TcsCustomBotao.SetSpacing(pnValor: Integer);
begin
  if FSpacing <> pnValor then
  begin
    FSpacing := pnValor;
    FSpeedButton.Spacing := FSpacing;
  end;
end;

procedure TcsCustomBotao.RaiseMetodoNaoImplementado(psNomeMetodo: string);
begin
  raise EMetodoNaoImplementado.Create('O método (' + psNomeMetodo + ') não ' +
    'foi implementado pela classse (' + ClassName + ').');
end;

function TcsCustomBotao.GetFigura(Index: Word): string;
begin
  Result := FFiguras.Strings[Index];
end;

function TcsCustomBotao.GetFigurasCount(): Word;
begin
  Result := FFiguras.Count;
end;

procedure TcsCustomBotao.ExecutaOnClick(Sender: TObject);
begin
  if Assigned(FOnClick) then
    FOnClick(Self);
end;

procedure TcsCustomBotao.ExecutaOnMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if Assigned(FOnMouseDown) then
```

```
    FOnMouseDown(Self, Button, Shift, X, Y);
end;

procedure TcsCustomBotao.ExecutaMouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  if Assigned(FOnMouseMove) then
    FOnMouseMove(Self, Shift, X, Y);
end;

procedure TcsCustomBotao.ExecutaOnMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Assigned(FOnMouseUp) then
    FOnMouseUp(Self, Button, Shift, X, Y);
end;

{ TcsAtributos }

procedure TcsDimenssoes.AvisaAlteracao();
begin
  if Assigned(FOnAtributosAlterados) then
    FOnAtributosAlterados(Self);
end;

procedure TcsDimenssoes.SetAlturaComTexto(const pnValor: Word);
begin
  if FAlturaComTexto <> pnValor then begin
    FAlturaComTexto := pnValor;
    AvisaAlteracao();
  end;
end;

procedure TcsDimenssoes.SetAlturaSemTexto(const pnValor: Word);
begin
  if FAlturaSemTexto <> pnValor then begin
    FAlturaSemTexto := pnValor;
    AvisaAlteracao();
  end;
end;

procedure TcsDimenssoes.SetComTexto(const pbValor: Boolean);
begin
  if FComTexto <> pbValor then begin
    FComTexto := pbValor;
    AvisaAlteracao();
  end;
end;

procedure TcsDimenssoes.SetLarguraComTexto(const pnValor: Word);
begin
  if FLarguraComTexto <> pnValor then begin
    FLarguraComTexto := pnValor;
    AvisaAlteracao();
  end;
end;
```

```
procedure TcsDimenssoes.SetLarguraSemTexto(const pnValor: Word);
begin
  if FLarguraSemTexto <> pnValor then begin
    FLarguraSemTexto := pnValor;
    AvisaAlteracao();
  end;
end;

end.
```

## unit ucsBotaoCadastro;

```
unit ucsBotaoCadastro;

interface

uses
  ucsBotao, Classes;

type
  TcsBotaoCadastro = class(TcsBotao)
  private
    { Private declarations }
  protected
    { Protected declarations }
    procedure DimensoesDefaultBotoes(var pbComTexto: Boolean;
      var pnAlturaComTexto, pnLarguraComTexto, pnAlturaSemTexto,
      pnLarguraSemTexto: Word); override;
    procedure RegistraRecursosImagens(poFiguras: TStringList); override;
    procedure RegistraCaptions(poCaptions: TStringList); override;
    procedure RegistraHints(poHints: TStringList); override;
  end;

implementation

{$R csBotoesCadastro.RES}

procedure TcsBotaoCadastro.DimensoesDefaultBotoes(var pbComTexto: Boolean;
  var pnAlturaComTexto, pnLarguraComTexto, pnAlturaSemTexto,
  pnLarguraSemTexto: Word);
begin
  pbComTexto := True;
  pnAlturaComTexto := 44;
  pnLarguraComTexto := 60;
  pnAlturaSemTexto := 36;
  pnLarguraSemTexto := 40;
end;

procedure TcsBotaoCadastro.RegistraRecursosImagens(poFiguras: TStringList);
begin
  poFiguras.Add('Salvar');
  poFiguras.Add('Consultar');
  poFiguras.Add('Interromper');
  poFiguras.Add('Selecionar');
  poFiguras.Add('Novo');
```

```
  poFiguras.Add('Editar');
  poFiguras.Add('Excluir');
  poFiguras.Add('Limpar');
  poFiguras.Add('Restaurar');
  poFiguras.Add('Visualizar');
  poFiguras.Add('Imprimir');
  poFiguras.Add('Configurar');
  poFiguras.Add('Fechar');
end;

procedure TcsBotaoCadastro.RegistraCaptions(poCaptions: TStringList);
begin
  poCaptions.Add('&Salvar');
  poCaptions.Add('Consul&tar');
  poCaptions.Add('Inte&rromper');
  poCaptions.Add('Sele&cionar');
  poCaptions.Add('&Novo');
  poCaptions.Add('E&ditar');
  poCaptions.Add('&Excluir');
  poCaptions.Add('&Limpar');
  poCaptions.Add('&Restaurar');
  poCaptions.Add('&Visualizar');
  poCaptions.Add('&Imprimir');
  poCaptions.Add('Confi&gurar');
  poCaptions.Add('&Fechar');
end;

procedure TcsBotaoCadastro.RegistraHints(poHints: TStringList);
begin
  poHints.Add('Salva as informações da janela');
  poHints.Add('Consulta os dados segundo os parâmetros informados');
  poHints.Add('Interrompe a consulta');
  poHints.Add('Seleciona o registro corrente');
  poHints.Add('Insere novo registro');
  poHints.Add('Edita o registro corrente');
  poHints.Add('Exclui o registro corrente');
  poHints.Add('Limpa as informações da janela');
  poHints.Add('Restaura a janela com as informações do último salvamento');
  poHints.Add('Visualiza o relatório');
  poHints.Add('Imprime o relatório');
  poHints.Add('Configura as opções da impressora');
  poHints.Add('Fecha a janela');
end;

end.
```

## unit ucsBotaoDataSource;

```
unit ucsBotaoDataSource;

interface

uses
  ucsBotao, Classes, DB;
```

```
type
  TDataSetNotifyConfirmEvent = procedure(DataSet: TDataSet;
    var pbCancelar: Boolean) of object;

  TcsBotaoDataSource = class(TcsCustomBotao)
  protected
    FDataSource: TDataSource;
    FOnAntesExecutar: TDataSetNotifyConfirmEvent;
    FOnDepoisExecutar: TDataSetNotifyEvent;
    procedure ExecutaOnClick(Sender: TObject); override;
    procedure ExecutaOperacao(); virtual; abstract;
    procedure DimensoesDefaultBotoes(var pbComTexto: Boolean;
      var pnAlturaComTexto, pnLarguraComTexto, pnAlturaSemTexto,
      pnLarguraSemTexto: Word); override;
    procedure Notification(AComponent: TComponent;
      Operation: TOperation); override;
  public
    constructor Create(AOwner: TComponent); override;
    procedure Click(); override;
  published
    property csDataSource: TDataSource read FDataSource write FDataSource;
  end;

implementation

constructor TcsBotaoDataSource.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  SetSpacing(3);
end;

procedure TcsBotaoDataSource.DimensoesDefaultBotoes(var pbComTexto: Boolean;
  var pnAlturaComTexto, pnLarguraComTexto, pnAlturaSemTexto,
  pnLarguraSemTexto: Word);
begin
  pbComTexto := False;
  pnAlturaComTexto := 44;
  pnAlturaSemTexto := 30;
  pnLarguraComTexto := 60;
  pnLarguraSemTexto := 31;
end;

procedure TcsBotaoDataSource.ExecutaOnClick(Sender: TObject);
var
  bCancelar: Boolean;
begin
  if FDataSource <> nil then
  begin
    bCancelar := False;

    if Assigned(FOnAntesExecutar) then
      FOnAntesExecutar(FDataSource.DataSet, bCancelar);

    if bCancelar = False then
    begin
      ExecutaOperacao();

      if Assigned(FOnDepoisExecutar) then
```

```
      FOnDepoisExecutar(FDataSource.DataSet);
    end;
  end;
end;

procedure TcsBotaoDataSource.Click();
begin
  ExecutaOnClick(Self);
end;

procedure TcsBotaoDataSource.Notification(AComponent: TComponent;
  Operation: TOperation);
begin
  inherited Notification(AComponent, Operation);
  if (Operation = opRemove) and (AComponent = FDataSource) then
    FDataSource := nil;
end;

end.
```

## unit ucsBotaoSeta;

```
unit ucsBotaoSeta;

interface

uses
  ucsBotaoDataSource, DB, Classes;

type
  TcsBotaoInsSeta = class(TcsBotaoDataSource)
  protected
    procedure ExecutaOperacao(); override;
    procedure RegistraRecursosImagens(poFiguras: TStringList); override;
    procedure RegistraCaptions(poCaptions: TStringList); override;
    procedure RegistraHints(poHints: TStringList); override;
  published
    property csOnAntesInserir: TDataSetNotifyConfirmEvent read FOnAntesExecutar
      write FOnAntesExecutar;
    property csOnDepoisInserir: TDataSetNotifyEvent read FOnDepoisExecutar
      write FOnDepoisExecutar;
  end;

  TcsBotaoDelSeta = class(TcsBotaoDataSource)
  protected
    procedure ExecutaOperacao(); override;
    procedure RegistraRecursosImagens(poFiguras: TStringList); override;
    procedure RegistraCaptions(poCaptions: TStringList); override;
    procedure RegistraHints(poHints: TStringList); override;
  published
    property csOnAntesExcluir: TDataSetNotifyConfirmEvent read FOnAntesExecutar
      write FOnAntesExecutar;
    property csOnDepoisExcluir: TDataSetNotifyEvent read FOnDepoisExecutar
      write FOnDepoisExecutar;
  end;
```

```
implementation

{$R csBotoesCadastroSeta.RES}

uses
  ucsMsgDlg;

{ TcsBotaoInsSeta }

procedure TcsBotaoInsSeta.ExecutaOperacao();
begin
  if FDataSource.DataSet <> nil then
    FDataSource.DataSet.Insert();
end;

procedure TcsBotaoInsSeta.RegistraCaptions(poCaptions: TStringList);
begin
  poCaptions.Add('InsSeta');
end;

procedure TcsBotaoInsSeta.RegistraHints(poHints: TStringList);
begin
  poHints.Add('Insere novo registro');
end;

procedure TcsBotaoInsSeta.RegistraRecursosImagens(poFiguras: TStringList);
begin
  poFiguras.Add('InsSeta');
end;

{ TcsBotaoDelSeta }

procedure TcsBotaoDelSeta.ExecutaOperacao();
begin
  if (FDataSource.DataSet <> nil) and (not FDataSource.DataSet.IsEmpty)
  and MsgConfirmacao('Deseja excluir o registro.') then
    FDataSource.DataSet.Delete();
end;

procedure TcsBotaoDelSeta.RegistraCaptions(poCaptions: TStringList);
begin
  poCaptions.Add('DelSeta');
end;

procedure TcsBotaoDelSeta.RegistraHints(poHints: TStringList);
begin
  poHints.Add('Excluir o registro corrente');
end;

procedure TcsBotaoDelSeta.RegistraRecursosImagens(poFiguras: TStringList);
begin
  poFiguras.Add('DelSeta');
end;

end.
```

## unit ucsMsgDlg;

```
unit ucsMsgDlg;

interface

type
  TBotaoPrecionado = (bpOK, bpSim, bpNao, bpCancelar);

function MsgConfirmacao(sMsg: string): Boolean;
function MsgConfirmAlteracao(sMsg: string): TBotaoPrecionado;
procedure MsgInformacao(sMsg: string);
procedure MsgErro(sMsg: string);
procedure MsgAviso(sMsg: string);

implementation

uses
  SysUtils, Dialogs, Forms, Controls, StdCtrls;

type
  TMensagemDlg = (mdConfirmacao, mdConfAlteracao, mdInformacao,  mdErro, mdAviso);

function GeraMenssagem(sMsg: string; eMensagem: TMensagemDlg): TBotaoPrecionado;
const
  nLARGURA_BOTAO = 65;
  nDISTANCIA_BOTAO = 6;

type
  TNomeBotoes = (nbOK, nbSim, nbNao, nbCancelar);

var
  fDialogo: TForm;
  sTitulo: string;
  oBotao: TButton;
  cBotoes: set of TNomeBotoes;
  eBotao: TNomeBotoes;
  nLeft, nBotoes, nWidthBotoes: integer;
  eMsgDlgType: TMsgDlgType;
begin
  Result := bpOK;
  cBotoes := [nbOK];
  nBotoes := 1;
  eMsgDlgType := mtInformation;
  case eMensagem of
    mdInformacao :
      begin
        sTitulo := 'Informação';
        eMsgDlgType := mtInformation;
      end;
    mdConfirmacao:
      begin
        sTitulo := 'Confirmação';
        cBotoes := [nbSim, nbNao];
        nBotoes := 2;
        eMsgDlgType := mtConfirmation;
      end;
```

```
  mdConfAlteracao:
    begin
      sTitulo := 'Confirmação';
      cBotoes := [nbSim, nbNao, nbCancelar];
      nBotoes := 3;
      eMsgDlgType := mtConfirmation;
    end;
  mdErro :
    begin
      sTitulo := 'Erro';
      eMsgDlgType := mtError;
    end;
  mdAviso:
    begin
      sTitulo := 'Aviso';
      eMsgDlgType := mtWarning;
    end;
end;

fDialogo := CreateMessageDialog(sMsg, eMsgDlgType, []);
try
  fDialogo.caption := sTitulo;

  nWidthBotoes := ((nLARGURA_BOTAO + nDISTANCIA_BOTAO) * nBotoes)
    - nDISTANCIA_BOTAO;
  if fDialogo.ClientWidth < (nWidthBotoes + 20) then
  begin
    fDialogo.ClientWidth := nWidthBotoes + 20;
    nLeft := 10
  end
  else
    nLeft := (fDialogo.ClientWidth div 2) - (nWidthBotoes div 2);

  for eBotao := Low(TNomeBotoes) to High(TNomeBotoes) do
    if eBotao in cBotoes then
    begin
      oBotao := TButton.Create(fDialogo);
      oBotao.Parent := fDialogo;
      oBotao.Height := oBotao.Height - 1;
      oBotao.Width := nLARGURA_BOTAO;
      oBotao.Top := fDialogo.ClientHeight - oBotao.Height - 13;
      oBotao.Left := nLeft;
      nLeft := nLeft + nLARGURA_BOTAO + nDISTANCIA_BOTAO;
      oBotao.ModalResult := mrOk;
      case eBotao of
        nbOk:
          begin
            oBotao.Caption := '&OK';
            oBotao.Cancel := True;
          end;
        nbSim:
          begin
            oBotao.Caption := '&Sim';
            oBotao.ModalResult := mrYes;
          end;
        nbNao:
          begin
            oBotao.Caption := '&Não';
```

```
          oBotao.ModalResult := mrNo;
          if eMensagem = mdConfirmacao then
            oBotao.Cancel := True;
        end;
      nbCancelar:
        begin
          oBotao.Caption := '&Cancelar';
          oBotao.ModalResult := mrCancel;
          oBotao.Cancel := True;
        end;
    end;
  end;

  case fDialogo.ShowModal of
    mrOk    : Result := bpOk;
    mrYes   : Result := bpSim;
    mrNo    : Result := bpNao;
    mrCancel: Result := bpCancelar;
  end;
 finally
  fDialogo.Free();
 end;
end;

function MsgConfirmacao(sMsg: string): Boolean;
begin
 Result := GeraMenssagem(sMsg, mdConfirmacao) = bpSim;
end;

function MsgConfirmAlteracao(sMsg: string): TBotaoPrecionado;
begin
 Result := GeraMenssagem(sMsg, mdConfAlteracao);
end;

procedure MsgInformacao(sMsg: string);
begin
 GeraMenssagem(sMsg, mdInformacao);
end;

procedure MsgAviso(sMsg: string);
begin
 GeraMenssagem(sMsg, mdAviso);
end;

procedure MsgErro(sMsg: string);
begin
 GeraMenssagem(sMsg, mdErro);
end;

end.
```

# Módulo Componentes

## unit ucsEntidade;

```
unit ucsEntidade;
interface

uses
  DesignIntf, DBClient, Provider, ucsAcessoDados, DB, Classes, SysUtils,
  IBQuery, IBDataBase, Contnrs;

type
  EErroValidacao = class(Exception); // para erros de validação.
  EMetdodoNaoImplementado = class(Exception);
  ESelectNaoRegistrado = class(Exception);
  EEntidadePaiFilha = class(Exception);

type
  TcsCustomEntidade = class(TCustomClientDataSet)
  private
    FAcessoDados: TcsAcessoDados;
    FSelects: TStringList; // armazena os selects registrados.
    FSelect: string; // armazena o select selecionado.
    FUltimoSelect: string;
    FOnErroGravacao: TDataSetNotifyEvent;
    procedure SetSelect(psValue: string);
    function GetSelects(Index: Word): string;
    function GetSelectsCount(): Word;
    function GetIBTransaction(): TIBTransaction;
    procedure OnPostErrorEntidade(DataSet: TDataSet; E: EDatabaseError;
      var Action: TDataAction);

  private // Métodos do Provider
    procedure OnUpdateErrorProvider(Sender: TObject;
      DataSet: TCustomClientDataSet; E: EUpdateError; UpdateKind: TUpdateKind;
      var Response: TResolverResponse);
    procedure BeforeUpdateRecordProvider(Sender: TObject;
      SourceDS: TDataSet; DeltaDS: TCustomClientDataSet;
      UpdateKind: TUpdateKind; var Applied: Boolean);

  protected // Utilizar nas classes descendentes.
    FProvider: TDataSetProvider;
    { Componente de acesso ao banco utilizado pela classe. É neste componente
      que os Sqls devem ser implementados e é este componente que será fornecido
      ao provider do conjunto de dados para a obtenção dos dados do banco. }
    FDataSet: TIBQuery;
    { Este método deve ser obrigatoriamente reimplementado pela classe herdade.
      Deve-se especificar o nome tabela do banco de dados a qual a classe
      herdade estará se referenciando. }
    function GetNomeTabela(): string; virtual; abstract;
    { Este método deve ser reimplementado na classe herdada. É através dele
      que os selects a serem executados pelo conjunto de dados são registrados.
      Para retistrar um select, adicione o nome dos procedimentos que executam
      os selects no parâmetro do método. Exemplo:

        poSelects.Add('SelectClientesInadimplentes');
```

```
    poSelects.Add('SelectClientesDeFlorianopolis');}
  procedure RegistraSelects(poSelects: TStringList); virtual;
  { Método onde deve ser atribuir o select do parâmetro especificado
    componente de acesso a dados FDataSet. }
  procedure ConfiguraSelect(psSelect: string); virtual;
  { Este método é muito importante, nele se deve inserir as regras de
    validação. Ele será chamado antes do método Post e ApplyUpdates. Se alguma
    regra de validação for quebrada, você pode cancelar a execução do método
    Post ou ApplyUpdates abribuindo algum valor para o parâmetro
    psMsgResposta. Se isto for feito, será gerada uma exceção EErroValidacao
    com a mensagem informada no parâmetro psMsgResposta. Veja um exemplo de
    implementação deste método na classe TecsUsuario. }
  procedure AntesSalvar(var psMsgResposta: string); virtual;
  { Este método é um ponteiro para o método BeforeUpdateRecord do provider.
    Ele será chamado se nenhum erro como o de campo obrigatório ocorrer, isto
    é, se os dados estiverem consistentes. Este método deve ser sobrescrito
    quando se deseja tratar os dados antes de serem gravados no banco de
    dados. Através deste método também é possível cancelar a operação de
    gravação dos dados no banco, atribuindo True a variável Applied ou
    gerando uma exceção, como no exemplo abaixo:

    if UpdateKind = kModify and
    DeltaDS.FieldByName('vlSalario').AsFloat  < 1500 then
      raise Exception.Create('Este salário é inaceitável.');

    Também se pode executar a validação apenas quando algum erro ocorrer, e
    não antes da tentativa de gravação dos dados. Para isto, sobreescreva o
    método TrataErro. }
  procedure AntesSalvarRegistro(SourceDS: TDataSet;
    DeltaDS: TCustomClientDataSet; UpdateKind: TUpdateKind;
    var Applied: Boolean); virtual;
  { Quando os eventos OnPostError da entidade e OnUpdateError do provider
    ocorrerem, este método será chamado. Deve ser sobreposto quando se deseja
    implementar algum tratamento após algum erro na tentativa de gravação dos
    registros ocorrer. Após este método ser chamado, uma exceção
    EErroValidacao será gerada. Se for especificado algum valor para o
    parâmetro psMsgResposta, a exceção EErroValidacao exibirá a mensagem
    informada no parâmetro, em caso negativo, exibirá a mensagem de erro que
    o Delphi gerou. }
  procedure ErroValidacao(psMsgErro: string; var psMsgResposta: string); virtual;
  { Executado antes do método insert ser executado. }
  procedure AntesInserir(); virtual;
  { Este método é muito útil para abrituir valores iniciais a campos quando
    uma inserção é executada, como por exemplo, atribuir um valor ao campo
    da chave primária, ou inicializar este campo com qualquer valor e atribuir
    o valor final no método AntesAtualizarRegistro. }
  procedure DepoisInserir(); virtual;
  // Tratamento de transação
  procedure StartTransaction();
  function InTransaction(): Boolean;
  procedure Commit();
  procedure Rollback();

protected // Utilização interna.
  procedure DoBeforeInsert; override;
  procedure DoAfterInsert; override;
  procedure DoAfterOpen(); override;
  procedure DoBeforeGetRecords(var OwnerData: OleVariant); override;
```

```
  procedure DoBeforeGetParams(var OwnerData: OleVariant); override;

protected // customizaveis
  property csSelect: string read FSelect write SetSelect;
  property csOnErroGravacao: TDataSetNotifyEvent read FOnErroGravacao
    write FOnErroGravacao;

public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy(); override;
  procedure Post(); override;
  { Devolve o select da entidade indexado pelo parâmetro Index. Se o metadados
    não foi executado, uma exceção é gerada. }
  property Selects[Index: Word]: string read GetSelects;
  { Devolve o número de selects da entidade. Se o metadados não foi executado,
    uma exceção é gerada. }
  property SelectsCount: Word read GetSelectsCount;
  { Se o compoente DataWare utilizado pelo aplicativo for Interbase, este
    método altera o componente IBTransaction associado ao DataSet da Entidade,
    que sendo interbase, é um IBQuery. É utilizado quando se deseja que mais
    de uma entidade pertença a mesma transação. Se o parâmetro
    poIBTransaction for igual a nil, o componente IBQuery irá utilizar o
    seu componente IBTransaction interno, isto é útil quando se alterou o
    transaction do IBQuery e quer restaurá-lo para o seu IBTransaction
    interno.}
  procedure SetIBTransaction(poIBTransaction: TIBTransaction);
  property IBTransaction: TIBTransaction read GetIBTransaction;
end;


TcsEntidade = class(TcsCustomEntidade)
private
  FEntidadePai: TcsEntidade;
  FEntidadesFilhas: TObjectList;
  FValorCampoAutoInc: Double;
  FCampoAutoInc: string;
  procedure SetEntidadePai(poEntidade: TcsEntidade);
  function GetValoresCamposChave(): Variant;
protected
  procedure Notification(AComponent: TComponent;
    Operation: TOperation); override;
  procedure DoAfterInsert(); override;
  procedure SetCampoAutoIncremental(psCampo: string);
public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy(); override;
  function AdicionaEntidadeFilha(poEntidade: TcsEntidade): Boolean;
  function RemoveEntidadeFilha(poEntidade: TcsEntidade): Boolean;
  function ApplyUpdatesAll(pnMaxErrors: Integer): Integer;
  procedure CancelUpdatesAll();
published
  property csEntidadePai: TcsEntidade read FEntidadePai write SetEntidadePai;
  property csSelect;
  property csOnErroGravacao;

 // Propriedades Delphi.
  property Active;
  property Filter;
```

```
    property Filtered;
    property Params;
    property PacketRecords;
    property ReadOnly;
{
    property MasterSource;
    property MasterFields;
    property IndexFieldNames;
}
    property BeforeOpen;
    property AfterOpen;
    property BeforeClose;
    property AfterClose;
    property BeforeInsert;
    property AfterInsert;
    property BeforeEdit;
    property AfterEdit;
    property BeforePost;
    property AfterPost;
    property BeforeCancel;
    property AfterCancel;
    property BeforeDelete;
    property AfterDelete;
    property BeforeScroll;
    property AfterScroll;
    property BeforeRefresh;
    property AfterRefresh;
    property OnCalcFields;
    property OnDeleteError;
    property OnEditError;
    property OnFilterRecord;
    property OnNewRecord;
    property BeforeApplyUpdates;
    property AfterApplyUpdates;
    property BeforeGetRecords;
    property AfterGetRecords;
    property BeforeRowRequest;
    property AfterRowRequest;
    property BeforeExecute;
    property AfterExecute;
    property BeforeGetParams;
    property AfterGetParams;
  end;

procedure Register;

implementation

uses
  ucsEntidadeEditor, ucsAplicacao, ucsDMAplicacao, ucsTrataMsgErro,
  ucsAcessoDadosIB, DBTables, uStringProcs, Variants, Forms, ucsBDProcs;

{ TcsEntidade }

procedure Register;
begin
  RegisterPropertyEditor(TypeInfo(string), TcsEntidade, 'csSelect',  TSelectsProperty);
end;
```

```
constructor TcsCustomEntidade.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FetchOnDemand := False;

  if csDMAplicacao = nil then
  begin
    csAplicacao := TcsAplicacao.Create();
    csDMAplicacao := TcsDMAplicacao.Create(nil);
  end;

  {Cria a classe de Acesso aos Dados de acordo com a classe de DataBase
  utilizada pelo aplicativo.}
  if csAplicacao.ComponenteDataAware = dwInterbase then
  begin
    FAcessoDados := TcsAcessoDadosIB.Create();
    FDataSet := TIBQuery(FAcessoDados.DataSet);
  end
  else if csAplicacao.ComponenteDataAware = dwBDE then
  begin

  end
  else if csAplicacao.ComponenteDataAware = dwDBExpress then
  begin

  end;

  FProvider := TDataSetProvider.Create(Self);
  FProvider.DataSet := FDataSet;
  FProvider.BeforeUpdateRecord := BeforeUpdateRecordProvider;
  // Tratar os erros.
  FProvider.OnUpdateError := OnUpdateErrorProvider;
  OnPostError := OnPostErrorEntidade;

  try
    FAcessoDados.SetNomeTabela(GetNomeTabela());
  except
    raise EMetdodoNaoImplementado.Create('O método (GetNomeEntidade) não foi '
      + 'implementado na classe (' +  ClassName + ').');
  end;

  FSelects := TStringList.Create();
  RegistraSelects(FSelects);
  FUltimoSelect := '';
  FSelect := '';
  FDataSet.SQL.Clear();
  FDataSet.SQL.Add('select * from ' + GetNomeTabela());
end;

destructor TcsCustomEntidade.Destroy();
begin
  FAcessoDados.Free();
  FSelects.Free();
  FProvider.Free();
  inherited Destroy();
end;
```

```
procedure TcsCustomEntidade.DoAfterOpen();
begin
  inherited DoAfterOpen();
  if csAplicacao.ComponenteDataAware = dwInterbase then
    TIBQuery(FDataSet).Transaction.Active := False;
end;
procedure TcsCustomEntidade.DoBeforeGetRecords(var OwnerData: OleVariant);
begin
  if Active = False then
    SetProvider(FProvider);
  inherited;
end;

procedure TcsCustomEntidade.DoBeforeGetParams(var OwnerData: OleVariant);
begin
  if Active = False then
    SetProvider(FProvider);
  inherited;
end;

procedure TcsCustomEntidade.OnUpdateErrorProvider(Sender: TObject;
  DataSet: TCustomClientDataSet; E: EUpdateError; UpdateKind: TUpdateKind;
  var Response: TResolverResponse);
var
  sMsgResposta: string;
begin
  Response := rrAbort;

  sMsgResposta := '';
  ErroValidacao(E.Message, sMsgResposta);
  if sMsgResposta = '' then
    raise EErroValidacao.Create(E.Message)
  else
    raise EErroValidacao.Create(sMsgResposta);

  if Assigned(FOnErroGravacao) then
    FOnErroGravacao(DataSet);
end;

procedure TcsCustomEntidade.OnPostErrorEntidade(DataSet: TDataSet;
  E: EDatabaseError; var Action: TDataAction);
var
  sMsgResposta: string;
begin
  Action := daAbort;
  sMsgResposta := '';
  ErroValidacao(E.Message, sMsgResposta);
  if sMsgResposta = '' then
    csTrataMsgErro.TrataErro(E.Message);
  if Assigned(FOnErroGravacao) then
    FOnErroGravacao(DataSet);
end;

procedure TcsCustomEntidade.ErroValidacao(psMsgErro: string;
  var psMsgResposta: string);
begin

end;
```

```
procedure TcsCustomEntidade.BeforeUpdateRecordProvider(Sender: TObject;
  SourceDS: TDataSet; DeltaDS: TCustomClientDataSet;
  UpdateKind: TUpdateKind; var Applied: Boolean);
begin
  AntesSalvarRegistro(SourceDS, DeltaDS, UpdateKind, Applied);
end;

procedure TcsCustomEntidade.AntesSalvarRegistro(SourceDS: TDataSet;
  DeltaDS: TCustomClientDataSet; UpdateKind: TUpdateKind;
  var Applied: Boolean);
begin
  { UpdateKind: kModify, ukInsert, ukDelete. }
end;

procedure TcsCustomEntidade.SetSelect(psValue: string);
begin
  if psValue = FSelect then
    Exit;

  if (psValue <> '') and (FSelects.IndexOf(psValue) = -1) then
  begin
    csSelect := FSelect;
    raise ESelectNaoRegistrado.Create('O select (' + psValue + ') não é um '
      + 'select registrado pela classe (' + Self.ClassName + ').');
  end;

  FSelect := psValue;
  Active := False;
  FDataSet.Close();

  if FSelect <> '' then
  begin
    FDataSet.Sql.Clear();
    ConfiguraSelect(FSelect);
  end
  else
  begin
    FDataSet.SQL.Clear();
    FDataSet.SQL.Add('select * from ' + GetNomeTabela());
  end;
  FAcessoDados.AtualizaUpdateSql();

  Params.Clear();
  FieldDefs.Update();
  FetchParams();

  FUltimoSelect := FSelect;
end;

function TcsCustomEntidade.GetSelects(Index: Word): string;
begin
  Result := FSelects.Strings[Index];
end;

function TcsCustomEntidade.GetSelectsCount(): Word;
begin
  Result := FSelects.Count;
end;
```

```
procedure TcsCustomEntidade.SetIBTransaction(poIBTransaction: TIBTransaction);
begin
  FAcessoDados.SetIBTransaction(poIBTransaction)
end;

function TcsCustomEntidade.GetIBTransaction: TIBTransaction;
begin
  Result := FAcessoDados.IBTransaction;
end;

procedure TcsCustomEntidade.RegistraSelects(poSelects: TStringList);
begin

end;

procedure TcsCustomEntidade.ConfiguraSelect(psSelect: string);
begin

end;

procedure TcsCustomEntidade.Post();
var
  sMsgResposta: string;
begin
  AntesSalvar(sMsgResposta);
  if sMsgResposta <> '' then
    raise EErroValidacao.Create(sMsgResposta);
  inherited Post();
end;

procedure TcsCustomEntidade.AntesSalvar(var psMsgResposta: string);
begin

end;

procedure TcsCustomEntidade.DoBeforeInsert();
begin
  AntesInserir();
  inherited DoBeforeInsert();
end;

procedure TcsCustomEntidade.DoAfterInsert();
begin
  DepoisInserir();
  inherited DoAfterInsert();
end;

procedure TcsCustomEntidade.AntesInserir();
begin

end;

procedure TcsCustomEntidade.DepoisInserir();
begin

end;
```

```
function TcsCustomEntidade.InTransaction(): Boolean;
begin
  Result := False;
  if csAplicacao.ComponenteDataAware = dwInterbase then
    Result := FAcessoDados.IBTransaction.InTransaction
  else if csAplicacao.ComponenteDataAware = dwBDE then
    Result := TDataBase(csAplicacao.DataBase).InTransaction
  else if csAplicacao.ComponenteDataAware = dwDBExpress then
  begin

  end;
end;

procedure TcsCustomEntidade.StartTransaction();
begin
  if csAplicacao.ComponenteDataAware = dwInterbase then
    FAcessoDados.IBTransaction.StartTransaction()
  else if csAplicacao.ComponenteDataAware = dwBDE then
    TDataBase(csAplicacao.DataBase).StartTransaction()
  else if csAplicacao.ComponenteDataAware = dwDBExpress then
  begin

  end;
end;

procedure TcsCustomEntidade.Commit();
begin
  if csAplicacao.ComponenteDataAware = dwInterbase then
    FAcessoDados.IBTransaction.Commit()
  else if csAplicacao.ComponenteDataAware = dwBDE then
    TDataBase(csAplicacao.DataBase).Commit()
  else if csAplicacao.ComponenteDataAware = dwDBExpress then
  begin

  end;
end;

procedure TcsCustomEntidade.Rollback();
begin
  if csAplicacao.ComponenteDataAware = dwInterbase then
    FAcessoDados.IBTransaction.Rollback()
  else if csAplicacao.ComponenteDataAware = dwBDE then
    TDataBase(csAplicacao.DataBase).Rollback()
  else if csAplicacao.ComponenteDataAware = dwDBExpress then
  begin

  end;
end;

{ TcsEntidade }

constructor TcsEntidade.Create(AOwner: TComponent);
begin
  FCampoAutoInc := '';
  inherited Create(AOwner);
  FEntidadesFilhas := TObjectList.Create();
  FEntidadesFilhas.OwnsObjects := False;
end;
```

```
destructor TcsEntidade.Destroy();
begin
  if FEntidadePai <> nil then
    FEntidadePai.RemoveEntidadeFilha(Self);
  FEntidadesFilhas.Free();
  inherited;
end;

procedure TcsEntidade.Notification(AComponent: TComponent;
  Operation: TOperation);
begin
  inherited Notification(AComponent, Operation);
  if (Operation = opRemove) and (AComponent is TcsEntidade) then
  begin
    if (FEntidadePai <> nil) and (AComponent = FEntidadePai) then
    begin
      FEntidadePai := nil;
      SetIBTransaction(nil);
    end;
  end;
end;

function TcsEntidade.GetValoresCamposChave(): Variant;
var
  i, nNumCampos: Integer;
  sCamposChave, sCampo: string;
begin
  nNumCampos := NumeroRepeticoes(';', FAcessoDados.CamposChave);
  Result := VarArrayCreate([0, nNumCampos-1], varVariant);
  sCamposChave := FAcessoDados.CamposChave;
  for i:=0 to nNumCampos-1 do
  begin
    sCampo := Parse(sCamposChave, ';');
    Result[i] := FieldByName(sCampo).AsString;
  end;
end;

function TcsEntidade.ApplyUpdatesAll(pnMaxErrors: Integer): Integer;
var
  i: Integer;
  vValores: Variant;
begin
  if not InTransaction() then
    StartTransaction();

  Result := ApplyUpdates(pnMaxErrors);

  for i:=0 to FEntidadesFilhas.Count-1 do
    if TcsEntidade(FEntidadesFilhas.Items[i]).Active then
      Result := TcsEntidade(FEntidadesFilhas.Items[i]).ApplyUpdates(pnMaxErrors);

  Commit();

  {Dá um refresh nas tabelas e posiciona o cursor devolta ao registro corrrente após o refresh.}
  if not IsEmpty then
  begin
    vValores := GetValoresCamposChave();
    Refresh();
```

```
    Locate(FAcessoDados.CamposChave, vValores, [loCaseInsensitive]);
  end;
  for i:=0 to FEntidadesFilhas.Count-1 do
    if (TcsEntidade(FEntidadesFilhas.Items[i]).Active)
    and (not TcsEntidade(FEntidadesFilhas.Items[i]).IsEmpty) then
    begin
      vValores := GetValoresCamposChave();
      TcsEntidade(FEntidadesFilhas.Items[i]).Refresh();
      TcsEntidade(FEntidadesFilhas.Items[i]).Locate(FAcessoDados.CamposChave,
        vValores, [loCaseInsensitive]);
    end;
end;

procedure TcsEntidade.CancelUpdatesAll();
var
  i: Integer;
begin
  if InTransaction() then
    Rollback();

  for i:=0 to FEntidadesFilhas.Count-1 do
    if TcsEntidade(FEntidadesFilhas.Items[i]).Active then
      TcsEntidade(FEntidadesFilhas.Items[i]).CancelUpdates();

  CancelUpdates();

  Refresh();
  for i:=0 to FEntidadesFilhas.Count-1 do
    TcsEntidade(FEntidadesFilhas.Items[i]).Refresh();

  if csAplicacao.ComponenteDataAware = dwInterbase then
    FAcessoDados.IBTransaction.Active := False;
end;

procedure TcsEntidade.SetEntidadePai(poEntidade: TcsEntidade);
begin
  if FEntidadePai <> poEntidade then
  begin
    if poEntidade = Self then
      raise EEntidadePaiFilha.Create('A entidade pai não pode ser ela mesma');

    if FEntidadesFilhas.IndexOf(poEntidade) <> -1 then
      raise EEntidadePaiFilha.Create('Você não pode setar uma entidade como ' +
        #13 + 'pai se  ela já for uma entidade filha');

    { Se a entidade já possui um pai, remove a entidade do seu pai antigo.}
    if FEntidadePai <> nil then
      FEntidadePai.RemoveEntidadeFilha(Self);

    if poEntidade = nil then
    begin
      FEntidadePai := nil;
      // Restaura a transação da entidade.
      SetIBTransaction(nil);
    end
    else
    begin   // Torna a transacao da entidade a mesma do pai.
      SetIBTransaction(poEntidade.IBTransaction);
```

```
      FEntidadePai := poEntidade;
      poEntidade.AdicionaEntidadeFilha(Self);
      poEntidade.FreeNotification(Self);
    end;
  end;
end;

function TcsEntidade.AdicionaEntidadeFilha(poEntidade: TcsEntidade): Boolean;
begin
  if (poEntidade <> nil) and (poEntidade <> Self) and (poEntidade <> FEntidadePai)
  and (FEntidadesFilhas.IndexOf(poEntidade) = -1) then
  begin
    FEntidadesFilhas.Add(poEntidade);
    Result := True;
  end
  else
    Result := False;
end;

function TcsEntidade.RemoveEntidadeFilha(poEntidade: TcsEntidade): Boolean;
var
  nIndex: Integer;
begin
  Result := False;
  if poEntidade = nil then
    Exit;

  nIndex := FEntidadesFilhas.IndexOf(poEntidade);
  if nIndex <> -1 then
  begin
    FEntidadesFilhas.Delete(nIndex);
    Result := True;
  end;
end;

procedure TcsEntidade.DoAfterInsert();
begin
  if FCampoAutoInc <> '' then
  begin
    if FValorCampoAutoInc = -717 then
      FValorCampoAutoInc := csBDProcs.GetMax(GetNomeTabela(), FCampoAutoInc);

    FValorCampoAutoInc := FValorCampoAutoInc + 1;
    FieldByName(FCampoAutoInc).AsFloat := FValorCampoAutoInc;
  end;
  inherited DoAfterInsert();
end;

procedure TcsEntidade.SetCampoAutoIncremental(psCampo: string);
begin
  if psCampo <> FCampoAutoInc then
  begin
    FCampoAutoInc := psCampo;
    FValorCampoAutoInc := - 717;
  end;
end;

end.
```

# unit ucsEntidadeQuery;

```
unit ucsEntidadeQuery;

interface

uses
  ucsEntidade, Classes, SysUtils;

type
  ESqlIncorreto = class(Exception);

  TecsQuery = class(TcsCustomEntidade)
  private
    FSQL: TStrings;
  protected
    function GetNomeTabela(): string; override;
    procedure SetActive(Value: Boolean); override;
    procedure SetSQL();
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy(); override;
    procedure Execute(); override;
    property SQL: TStrings read FSQL write FSQL;
  end;

implementation

uses
  ucsAplicacao, IBQuery, DBTables;

{ TecsQuery }

function TecsQuery.GetNomeTabela(): string;
begin
  Result := '';
end;

constructor TecsQuery.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FSQL := TStringList.Create();
end;

destructor TecsQuery.Destroy();
begin
  FSQL.Free();
  inherited Destroy();
end;

procedure TecsQuery.SetActive(Value: Boolean);
begin
  if Self.Active <> Value then
    SetSQL();

  try
    inherited SetActive(Value);
```

```
  except
    on E: Exception do
      raise ESqlIncorreto.Create(E.Message);
  end;
end;

procedure TecsQuery.SetSQL();
begin
  if csAplicacao.ComponenteDataAware = dwInterbase then
  begin
    if TIBQuery(FDataSet).SQL.Text <> FSQL.Text then
      TIBQuery(FDataSet).SQL.Assign(FSQL);
  end
  else if csAplicacao.ComponenteDataAware = dwBDE then
  begin
    if TQuery(FDataSet).SQL.Text <>  FSQL.Text then
      TQuery(FDataSet).SQL.Assign(FSQL);
  end
  else if csAplicacao.ComponenteDataAware = dwDBExpress then
  begin

  end;
end;

procedure TecsQuery.Execute();
var
  bExecutarCommit: Boolean;
begin
  SetSQL();
  SetProvider(FProvider);
  // Só inicia transação se a execução não estiver dentro de uma outra transação
  if not InTransaction then
  begin
    StartTransaction();
    bExecutarCommit := True;
  end
  else
    bExecutarCommit := False;

  inherited Execute();

  if bExecutarCommit then
    Commit();
end;

end.
```

# unit ucsAcessoDados;

```
unit ucsAcessoDados;

interface

uses
  Classes, DB, IBDataBase, SysUtils;
```

```
type
  EAcessoDados = class(Exception);

  TcsAcessoDados = class
  private
    { Private declarations }
  protected
    FDataSet: TDataSet;
    FNomeTabela: string;
    FCamposChave: string;
    FCamposTabela: string;
    { Este método é responsável pela criação do componente DataSet
      utilizado pela classe. Deve ser obrigatoriamente implementado
      pela classe filha. É através deste método que a classe filha
      especifica qual compoente de acesso aos dados deseja utilizar. }
    function CriaDataSet(): TDataSet; virtual; abstract;
    { É sobreposto pela classe TcsAcessoDadosIB. }
    function GetIBTransaction(): TIBTransaction; virtual;
    { Método responsável pela obtenção dos campos da tabela física, atribuindo
      os campos encontrados as variáveis FCamposChave e FCamposTabela. Deve
      ser sobreposto pela classe herdade para a execução deste cálculo de acordo
      com o compnente de acesso aos dados utilizado. }
    procedure ObtemCamposTabela(); virtual;
  public
    { Public declarations }
    constructor Create();
    destructor Destroy(); override;
    { Deve ser informado o nome da tabela a ser utilizada para a geração dos
      comandos sqls do componente UpdateSql. }
    procedure SetNomeTabela(psNome: string);
    { Esta propriedade é utilizada pela classe TcsEntidade para pegar o
      DataSet criado. }
    property DataSet: TDataSet read FDataSet;
    { Configura os sqls do componente UpdateSql. }
    procedure AtualizaUpdateSql(); virtual;
    { É sobreposto pela classe TcsAcessoDadosIB. }
    procedure SetIBTransaction(poIBTransaction: TIBTransaction); virtual;
    property IBTransaction: TIBTransaction read GetIBTransaction;
    { Armazena os campos que compõem a chave primária da tabela, separados por
      ponto e vírgula (;). }
    property CamposChave: string read FCamposChave;
    { Armazena todos os campos que compõem a tabela física, não apenas os
      campos da tabela incluídos no select. }
    property CamposTabela: string read FCamposTabela;
  end;

implementation

{ TcsAcessoDados }

constructor TcsAcessoDados.Create();
begin
  FDataSet := CriaDataSet();
  FNomeTabela := '';
  FCamposChave := '';
  FCamposTabela := '';
end;
```

```
destructor TcsAcessoDados.Destroy();
begin
  FDataSet.Free();
end;

procedure TcsAcessoDados.SetNomeTabela(psNome: string);
begin
  FNomeTabela := UpperCase(psNome);
  ObtemCamposTabela();
end;

procedure TcsAcessoDados.ObtemCamposTabela();
begin

end;

procedure TcsAcessoDados.AtualizaUpdateSql();
begin

end;

function TcsAcessoDados.GetIBTransaction(): TIBTransaction;
begin
  raise EAcessoDados.Create('A Classe de acesso a dados não é Interbase');
end;

procedure TcsAcessoDados.SetIBTransaction(poIBTransaction: TIBTransaction);
begin
  raise EAcessoDados.Create('A Classe de acesso a dados não é Interbase');
end;

end.
```

## unit ucsAcessoDadosIB;

```
unit ucsAcessoDadosIB;

interface

uses
  ucsAcessoDados, DB, IBQuery, IBDataBase, IBUpdateSQL;

type
  TcsAcessoDadosIB = class(TcsAcessoDados)
  private
    { Private declarations }
    FIBTransaction: TIBTransaction;
    FIBUpdateSQL: TIBUpdateSQL;
  protected
    { Protected declarations }
    function CriaDataSet(): TDataSet; override;
  public
    { Public declarations }
    destructor Destroy(); override;
    procedure ObtemCamposTabela(); override;
```

```
    procedure AtualizaUpdateSql(); override;
    procedure SetIBTransaction(poIBTransaction: TIBTransaction); override;
    function GetIBTransaction(): TIBTransaction; override;
  end;

implementation

uses
  Classes, SysUtils, IBTable, ucsAplicacao, uStringProcs;

{ TcsIBDataSet }

function TcsAcessoDadosIB.CriaDataSet(): TDataSet;
begin
  Result := TIBQuery.Create(nil);

  FIBTransaction := TIBTransaction.Create(nil);
  FIBTransaction.DefaultDatabase := TIBDataBase(csAplicacao.DataBase);

  FIBUpdateSQL := TIBUpdateSQL.Create(nil);

  TIBQuery(Result).Database := TIBDataBase(csAplicacao.DataBase);
  TIBQuery(Result).Transaction := FIBTransaction;
  TIBQuery(Result).UpdateObject := FIBUpdateSQL;
  TIBQuery(Result).UniDirectional := True;
end;

destructor TcsAcessoDadosIB.Destroy();
begin
  FIBTransaction.Active := False;
  FIBTransaction.Free();
  FIBUpdateSQL.Free();
end;

procedure TcsAcessoDadosIB.ObtemCamposTabela();
var
  oIBTable: TIBTable;
  i: Integer;
begin
  if FNomeTabela = '' then
    Exit;

  oIBTable := TIBTable.Create(nil);
  try
    FCamposChave := '';
    FCamposTabela := '';
    oIBTable.Database := TIBDataBase(csAplicacao.DataBase);
    oIBTable.Transaction := FIBTransaction;
    oIBTable.TableName := FNomeTabela;
    oIBTable.IndexDefs.Update();
    // pega campos que compõem a chave primária da tabela.
    for i:=0 to oIBTable.IndexDefs.Count-1 do
      if ixPrimary in oIBTable.IndexDefs[i].Options then
        FCamposChave := FCamposChave + oIBTable.IndexDefs[i].Fields + ';';

    // pega os campos da tabela.
    for i:=0 to oIBTable.FieldDefs.Count-1 do
      FCamposTabela := FCamposTabela + oIBTable.FieldDefs[i].Name + ';';
```

```
  finally
    oIBTable.Free();
  end;
end;

procedure TcsAcessoDadosIB.AtualizaUpdateSql();
var
  i: Integer;
  sCamposChave, sCampo: string;
begin
  sCamposChave := FCamposChave;

  FIBUpdateSQL.ModifySQL.Clear();
  FIBUpdateSQL.InsertSQL.Clear();
  FIBUpdateSQL.DeleteSQL.Clear();

  // Configura Sqls.
  FIBUpdateSQL.ModifySQL.Add('update ' + FNomeTabela + ' set');
  FIBUpdateSQL.InsertSQL.Add('insert into ' + FNomeTabela + ' (');
  FIBUpdateSQL.DeleteSQL.Add('delete from ' + FNomeTabela);

  FDataSet.Open();
  FDataSet.FieldDefs.Update();
  for i:=0 to FDataSet.Fields.Count-1 do
  begin
    // Se o campo pertence a tabela principal.
    if Pos(FDataSet.Fields[i].FieldName, FCamposTabela) > 0 then
    begin
      sCampo := UpperCase(FDataSet.Fields[i].FieldName);
      // Se se está inserindo o primeiro campo.
      if FIBUpdateSQL.ModifySQL.Count = 1 then
      begin
        // Modify
        FIBUpdateSQL.ModifySQL.Add(sCampo + ' = :' + sCampo);
        // Insert
        FIBUpdateSQL.InsertSQL.Add(sCampo);
        FIBUpdateSQL.InsertSQL.Add(') values (');
        FIBUpdateSQL.InsertSQL.Add(':' + sCampo);
      end
      else
      begin
        // Modify
        FIBUpdateSQL.ModifySQL.Add(',' + sCampo + ' = :' + sCampo);
        // Insert
        FIBUpdateSQL.InsertSQL.Insert(i+1, ',' + sCampo);
        FIBUpdateSQL.InsertSQL.Add(',:' + sCampo);
      end
    end
    else
      FDataSet.FieldDefs[i].Required := False;

  end; // end for.
  FIBUpdateSQL.InsertSQL.Add(')');

  FDataSet.Close();

  // Gera comando Where.
  FIBUpdateSQL.ModifySQL.Add('where');
```

```
  FIBUpdateSQL.DeleteSQL.Add('where');
  sCampo := Parse(sCamposChave, ';');
  repeat
    if sCamposChave = '' then
    begin
      FIBUpdateSQL.ModifySQL.Add(sCampo + ' = :OLD_' + sCampo);
      FIBUpdateSQL.DeleteSQL.Add(sCampo + ' = :OLD_' + sCampo);
    end
    else
    begin
      FIBUpdateSQL.ModifySQL.Add(sCampo + ' = :OLD_' + sCampo + ' and');
      FIBUpdateSQL.DeleteSQL.Add(sCampo + ' = :OLD_' + sCampo + ' and');
    end;

    sCampo := Parse(sCamposChave, ';');
  until sCampo = '';
end;

procedure TcsAcessoDadosIB.SetIBTransaction(poIBTransaction: TIBTransaction);
begin
  if poIBTransaction = nil then
    TIBQuery(FDataSet).Transaction := FIBTransaction
  else if TIBQuery(FDataSet).Transaction <> poIBTransaction then
  begin
    FIBTransaction.Active := False;
    TIBQuery(FDataSet).Transaction := poIBTransaction;
  end;
end;

function TcsAcessoDadosIB.GetIBTransaction(): TIBTransaction;
begin
  Result := TIBQuery(FDataSet).Transaction;
end;

end.
```

## unit ucsMensagens;

```
unit ucsMensagens;

interface

const
  // Erro
  serCAMPO_OBRIGATORIO
    = 'Campo obrigatório (%p1) não preenchido';
  serVIOLACAO_FOREIGN_KEY
    = 'Violação da chave extrangeira (%p1) na tabela (%p2)';
  serKEY_VIOLATION
    = 'Chave primária duplicada';

  // Aviso
  savMENSAGEM
    = 'Mensagem';
```

```
  // Informação
  sinMENSAGEM
    = 'Mensagem';

  // Confirmação
  scfMENSAGEM
    = 'Mensagem';

function GetMensagem(const psMsg: string; psParametros: array of string): string;

implementation

uses
  uStringProcs, SysUtils;

function GetMensagem(const psMsg: string; psParametros: array of string): string;
var
  i, nPosFim: Integer;
begin
  Result := psMsg;
  if Result = '' then
    Exit;

  nPosFim := High(psParametros);
  for i := 0 to nPosFim do
    Result := SubstituiString(Result, '%p' + IntToStr(i+1), psParametros[i]);
end;

end.
```

## unit ucsTrataMsgErro;

```
unit ucsTrataMsgErro;

interface

uses
  SysUtils;

type
  TcsTrataMsgErro = class
  private
  public
    procedure TrataErro(psMsgErro: string); overload;
    procedure TrataErro(Sender: TObject; E: Exception); overload;
  end;

var
  csTrataMsgErro: TcsTrataMsgErro;

implementation

uses
  ucsMsgDlg, uStringProcs, ucsMensagens;
```

```
{ TcsTrataErro }

procedure TcsTrataMsgErro.TrataErro(Sender: TObject; E: Exception);
begin
  TrataErro(E.Message);
end;

procedure TcsTrataMsgErro.TrataErro(psMsgErro: string);
begin
  if Pos('must have a value', psMsgErro) > 0 then
    psMsgErro := GetMensagem(serCAMPO_OBRIGATORIO,
      [ParsePos(psMsgErro, '''', 2)])
  else if Pos('Field value required', psMsgErro) > 0 then
    psMsgErro := GetMensagem(serCAMPO_OBRIGATORIO, ['???'])
  else if Pos('Key violation', psMsgErro) > 0 then
    psMsgErro := GetMensagem(serKEY_VIOLATION, [])
  else if Pos('FOREIGN KEY', psMsgErro) > 0 then
    psMsgErro := GetMensagem(serVIOLACAO_FOREIGN_KEY,
      [ParsePos(psMsgErro, '''', 2), ParsePos(psMsgErro, '''', 4)]);

  MsgErro(psMsgErro);
end;

initialization
  csTrataMsgErro := TcsTrataMsgErro.Create();

finalization
  csTrataMsgErro.Free();

end.
```

# Módulo Componentes

## unit ucsPrincipal;

```
unit ucsPrincipal;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, ExtCtrls, ucsForm;

type
  TfcsPrincipal = class(TForm)
    pnFormularios: TPanel;
  private
  protected
    {//??? Pendente - Comentário pendente devido ao implementação definitiva do
    método não estar definida.}
    procedure AbreForm(sNomeForm: string);
  public
    constructor Create(AOwner: TComponent); override;
    { Descrição
```

```
    Quando uma nova instância do aplicativo for iniciada, ela irá
    verificar se já existe uma instância do mesmo rodando, caso
    exista, ela chamará este método, que irá abrir a instância em
    andamento, evitando assim que duas instâncias do mesmo
    aplicativo sejam executadas simultaneamente na mesma máquina. }
   procedure RestauraJanelaAplicativo(var Msg: TMessage); message WM_USER;
   { Descrição
    Este método é chamado pelo formulário da classe TfcsForm toda
    vez que o mesmo for fechado, possibilitando ao formulário
    principal customizar sua interface e seu comportamento quando
    este evento ocorrer.

    Parâmetros
    oForm :   formulário que chamou o método (formulário fechado). }
   procedure FormFechou(oForm: TfcsForm); virtual;
  end;

var
  fcsPrincipal: TfcsPrincipal;

implementation

uses
  ucsMsgDlg, ucsTrataErro, ucsAplicacao;

{$R *.dfm}

procedure TfcsPrincipal.RestauraJanelaAplicativo(var Msg: TMessage);
begin
  Application.Restore();
end;

constructor TfcsPrincipal.Create(AOwner: TComponent);
begin
  inherited;
  Caption := csAplicacao.NomeComVersao;
  Application.Title := csAplicacao.NomeComVersao;
  Application.OnException := csTrataErro.TrataErro;
end;

procedure TfcsPrincipal.FormFechou(oForm: TfcsForm);
begin
  //??? Comando Incompleto - Joáber.
  {Antes de mostrar o paDescktop, deve-se testar se ainda existem forms abertos}
  {Esta solução é temporária, deve ser testada, no caso do form
   ControleJanelasAbertas, como ficaria ?}
  if Screen.CustomFormCount = 2 then
    pnFormularios.Show;
end;

procedure TfcsPrincipal.AbreForm(sNomeForm: string);
var
  oFormClass: TFormClass; // Referencia à class TForm, pg39 livro Componentes.
  oForm: TForm;
  i, nHeight, nWidth: Integer;
begin
  try
    oFormClass := TFormClass(FindClass('T' + sNomeForm));
```

```
except
  on EClassNotFound do begin
    MsgErro('Formulário (' + sNomeForm + ') não existe ou ' +
    '(RegisterClass)' + #13 + 'do fomulário não implementado.');
    Exit;
  end;
end;

// Pesquisa o aplicativo para ver se o formulário já está criado.
oForm := nil;
for i:=0 to Application.MainForm.ComponentCount-1 do
  if Application.MainForm.Components[i].ClassName = oFormClass.ClassName then
  begin
    oForm := Application.MainForm.Components[i] as TfcsForm;
    Break;
  end;

// Se o formulário já foi criado apenas o mostra, senão o cria e mostra.
if oForm <> nil then
begin
  if oForm.WindowState = wsMinimized then
    oForm.WindowState := wsNormal
  else
  begin
    oForm.Show;
    oForm.BringToFront;
  end;
end
else begin
  Screen.Cursor := crHourGlass;
  oForm := oFormClass.Create(Application.MainForm);
  nHeight := oForm.Height;
  nWidth := oForm.Width;
  if oForm.Visible then
    MsgErro('A Propriedade "visible" do Formulário "' + oForm.Name +
      '"' + #13 + 'deve ser igual a "false" para que a sequência de ' +
      #13 + 'chamadas dos métodos de abertura funcione corretamente.');
  Screen.Cursor := crDefault;

  //??? Comando Incompleto - Joáber.
  {Se o arquvo ini ainda não capturou a posicao do form o abre
  centralizado (Padrão)}
  if oForm.BorderStyle = bsDialog then
  begin
    oForm.Position := poScreenCenter;
    oForm.ShowModal;
  end
  else
  begin
    pnFormularios.Hide;
    oForm.FormStyle := fsMDIChild;
    oForm.Height := nHeight;
    oForm.Width := nWidth;

    // Configura o posicionamento do form.
    if pnFormularios.Width < oForm.Width  then
      oForm.Left := 10
    else
```

```
    oForm.Left := (pnFormularios.Width div 2) - (oForm.Width div 2);
   if pnFormularios.Height < oForm.Height  then
    oForm.Top := 10
   else
    oForm.Top := (pnFormularios.Height div 2) - (oForm.Height div 2);
  end;
 end;
end;

end.
```

## unit ucsPrincSeguranca;

```
unit ucsPrincSeguranca;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ucsPrincipal, ImgList, Menus, StdCtrls, ExtCtrls, ComCtrls;

type
  TfcsPrincSeguranca = class(TfcsPrincipal)
    procedure FormCreate(Sender: TObject);
  private
    FAcessoPermitido: Boolean;
    function VerificaAcesso(): Boolean;
  public
    property AcessoPermitido: Boolean read FAcessoPermitido;
  end;

var
  fcsPrincSeguranca: TfcsPrincSeguranca;

implementation

uses
  ucsVerificaAcesso;

{$R *.dfm}

procedure TfcsPrincSeguranca.FormCreate(Sender: TObject);
begin
  inherited;
  FAcessoPermitido := VerificaAcesso();
end;

function TfcsPrincSeguranca.VerificaAcesso(): Boolean;
begin
  fcsVerificaAcesso := TfcsVerificaAcesso.Create(nil);
  try
    if fcsVerificaAcesso.ShowModal() = mrOk then begin
      Result := True;
    end
    else
```

```
    Result := False;
  finally
    FreeAndNil(fcsVerificaAcesso);
  end;
end;

end.
```

## unit ucsForm;

```
unit ucsForm;
{-----------------------------------------------------------------------------
** Criação: 01/11/2002
** Definição: Formulário topo da hierarquia de janelas do framework
** Objetivo: Todos os demais formulários do framework, com exceção do formulário
 principal (fcsPrincipal), serão derivados a partir deste formulário. Este
 formulário implementa a interação com janela principal, possibilitando que
 a mesma exerça um controle sobre as janelas do sistema.
** Especificações:
 * Se a propriedade BorderStyle for igual a bsDialog o Formulário será modal.
-----------------------------------------------------------------------------}

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  StdCtrls;

type
  TfcsForm = class(TForm)
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  protected
    { Protected declarations }
    procedure DoShow(); reintroduce; override;
  public
    constructor Create(AOwner: TComponent); override;
  end;

var
  fcsForm: TfcsForm;

implementation

uses
  ucsPrincipal, ucsSeguranca, ucsMsgDlg, ucsTiposDefinidos;

{$R *.dfm}

constructor TfcsForm.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
end;
```

```
procedure TfcsForm.DoShow();
var
  i: Integer;
begin
  inherited DoShow();

  if csSeguranca.PermissaoVisualizar(Self.Name) = False then
  begin
    MsgAviso('Você não tem permissão para acessar esta tela.');
    Close();
    Exit;
  end;

  for i:=0 to ComponentCount-1 do
  begin
    if Components[i] is TControl then
      case csSeguranca.EstadoComponente(Name, Components[i].Name) of
        ecDesabilitado: TControl(Components[i]).Enabled := False;
        ecInvisivel   : TControl(Components[i]).Visible := False;
      end;
  end;
end;

procedure TfcsForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Action := caFree;
  TfcsPrincipal(Application.MainForm).FormFechou(Self);
end;

initialization
  RegisterClass(TfcsForm);

end.
```

## unit ucsCadastro;

```
unit ucsCadastro;

interface

uses
  Windows, Messages, SysUtils, Variants, ucsForm, DB, ucsBotao, Dialogs,
  ucsBotaoCadastro, Controls, ExtCtrls, Classes, DBClient, IBDataBase, DBTables,
  Contnrs, ucsEdicao;

type
  TfcsCadastro = class(TfcsEdicao)
    dsCadastro: TDataSource;
    btSalvar: TcsBotaoCadastro;
    btRestaurar: TcsBotaoCadastro;
    procedure btFecharcsOnClick(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure btSalvarcsOnClick(Sender: TObject);
    procedure btRestaurarcsOnClick(Sender: TObject);
  private
```

```
    FEntidades: TObjectList;
    FBotoesSeta: TObjectList;
    { Verifica se existe algum DataSet associado ao componente dsCadastro. Em
      caso negativo, uma exceção é gerada. }
    procedure VerificaDataSetAssociado();
    { Obtem a lista de todas os botões seta (TcsBotaoDataSource) da tela. }
    procedure ObtemListaBotoesSeta(poBotoesSeta: TObjectList);
    { Obtem a lista de todas as entidades da tela. A entidade principal, que
      está associada ao dsCadastro, será sempre a primeira da lista, as demais
      entidades serão ordenadas de acordo com suas propriedades
      csOrdemGravacao. }
    procedure ObtemListaEntidades(poEntidades: TObjectList);
    { Percorre todas as entidades Ativas verificando se os registros foram
      alterados. Se pelo menos uma das entidades tiver seus registros alterados,
      a função devolte True, se não houver nenhuma entidade com registros
      alterados, a função devolve False. }
    function RegistrosAlterados(): Boolean;
  protected
    FPermInserir: Boolean;
    FPermEditar: Boolean;
    FPermExcluir: Boolean;
    procedure DoShow(); reintroduce; override;
    procedure InsereNovoRegistro(); virtual;
    procedure EditaRegistros(); virtual;
    { Restaura os dados dos registros de todas as entidades Ativas para o estado
      do último salvamento. }
    procedure RestauraRegistros(); virtual;
    { Salva as alterações feitas nos registros de todas as entidades Ativas. }
    procedure SalvaRegistros(); virtual;
    { Exibe uma caixa de mensagem de confirmação para a exclusão do registro,
      devolvendo True caso o usuário precione o botão Sim.}
    function ConfirmaExclusao(): Boolean;
    { Exibe uma caixa a mensagem de confirmação para a exclusão do registro, se
      o usuário precionar o botão Sim, o método ExcluiRegistro é chamado. }
    procedure ConfirmaAntesExcluir();
    { Exclui o registro corrente. }
    procedure ExcluiRegistro(); virtual;
    procedure KeyDown(var Key: Word; Shift: TShiftState); override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy(); override;
  end;

  ECadastroErro = class(Exception);

var
  fcsCadastro: TfcsCadastro;

implementation

uses
  ucsEntidade, ucsMsgDlg, ucsAplicacao, DBGrids, dxDBGrid, ucsBotaoDataSource,
  ucsCadastroGrid, ucsCadastroGridEdit, ucsSeguranca;

{$R *.dfm}

{ TfcsCadastro }
```

```
constructor TfcsCadastro.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FEntidades := TObjectList.Create();
  FEntidades.OwnsObjects := False;
  ObtemListaEntidades(FEntidades);

  FBotoesSeta := TObjectList.Create();
  FBotoesSeta.OwnsObjects := False;
  ObtemListaBotoesSeta(FBotoesSeta);
end;

destructor TfcsCadastro.Destroy();
begin
  FEntidades.Free();
  FBotoesSeta.Free();
  inherited;
end;

procedure TfcsCadastro.ObtemListaEntidades(poEntidades: TObjectList);
var
  i: Integer;
begin
  for i:=0 to ComponentCount-1 do
    if Components[i] is TcsEntidade then
      poEntidades.Add(Components[i]);
end;

procedure TfcsCadastro.VerificaDataSetAssociado();
begin
  if dsCadastro.DataSet = nil then
    raise ECadastroErro.Create('Nenhum (DataSet) foi associado ao ' +
      'componente DataSouce (dsCadastro).');
end;

procedure TfcsCadastro.InsereNovoRegistro();
begin
  VerificaDataSetAssociado();
  TcsEntidade(dsCadastro.DataSet).Insert();
end;

procedure TfcsCadastro.EditaRegistros();
begin
  VerificaDataSetAssociado();
  TcsEntidade(dsCadastro.DataSet).Edit();
end;

function TfcsCadastro.ConfirmaExclusao(): Boolean;
begin
  Result := MsgConfirmacao('Deseja excluir o registro.');
end;

procedure TfcsCadastro.ConfirmaAntesExcluir();
begin
  if ConfirmaExclusao() then
    ExcluiRegistro();
end;
```

```
procedure TfcsCadastro.ExcluiRegistro();
begin
  VerificaDataSetAssociado();
  TcsEntidade(dsCadastro.DataSet).Delete();
end;

function TfcsCadastro.RegistrosAlterados(): Boolean;
var
  i: Integer;
begin
  Result := False;
  for i:=0 to FEntidades.Count-1 do
    if TcsEntidade(FEntidades.Items[i]).Active
    and (TcsEntidade(FEntidades.Items[i]).ChangeCount > 0) then
    begin
      Result := True;
      Exit;
    end;
end;

procedure TfcsCadastro.SalvaRegistros();
begin
  VerificaDataSetAssociado();
  TcsEntidade(dsCadastro.DataSet).ApplyUpdatesAll(-1);
end;

procedure TfcsCadastro.RestauraRegistros();
begin
  VerificaDataSetAssociado();
  TcsEntidade(dsCadastro.DataSet).CancelUpdatesAll();
end;

procedure TfcsCadastro.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
begin
  if dsCadastro.DataSet = nil then
    Exit;

  if RegistrosAlterados() then
    case MsgConfirmAlteracao('Os registros foram alterados,  deseja salvá-los.') of
      bpSim: SalvaRegistros();
      bpNao: RestauraRegistros();
      else
         CanClose := False;
  end
  else
    inherited;
end;

procedure TfcsCadastro.btSalvarcsOnClick(Sender: TObject);
begin
  SalvaRegistros();
end;

procedure TfcsCadastro.btRestaurarcsOnClick(Sender: TObject);
begin
  RestauraRegistros();
end;
```

```
procedure TfcsCadastro.btFecharcsOnClick(Sender: TObject);
begin
  Close();
end;

procedure TfcsCadastro.KeyDown(var Key: Word; Shift: TShiftState);

  function GetDataSourceGridFocused(poComponent: TComponent): TDataSource;
  begin
    Result := nil;
    if ((poComponent is TdxDBGrid) or (poComponent is TDBGrid))
    and TWinControl(poComponent).Focused then
      if poComponent is TdxDBGrid then // TdxDBGrid
        Result := TdxDBGrid(poComponent).DataSource
      else // TDBGrid
        Result := TDBGrid(poComponent).DataSource;
  end;

var
  i, j: Integer;
  bExcecutou: Boolean;
  oDataSource: TDataSource;
begin
  { Se for precinado Alt + Ins ou Alt + Del, verifica se a tela possui altum
   TcsBotaoInsSeta ou TcsBotaoDelSeta, se tiver, associa o click ao botão
   apropriado. }
  if ssAlt in Shift then
  begin
    if Key = VK_INSERT then
    begin
      Key := 0;
      bExcecutou := False;
      for i:=0 to ComponentCount-1 do
      begin
        oDataSource := GetDataSourceGridFocused(Components[i]);
        if oDataSource <> nil then
        begin
          for j:=0 to FBotoesSeta.Count-1 do
          begin
            if TcsBotao(FBotoesSeta.Items[j]).csEnabled
            and (FBotoesSeta.Items[j] is TcsBotaoDataSource)
            and (TcsBotaoDataSource(FBotoesSeta.Items[j]).csDataSource = oDataSource) then
            begin
              TcsBotaoDataSource(FBotoesSeta.Items[j]).Click();
              bExcecutou := True;
              Break;
            end; // fim IF, fim FOR.
          end;
          Break;
        end;  // fim IF, fim FOR.
      end;

      { Se for precionado Alt + Ins mas o foco não estava na Grid, é verificado
        se a classe é TfcsCadastroGrid ou TfcsCadastroGridEdit, se for uma das
        duas, por default deve-se executar o click do botão btInserir. }
      if bExcecutou = False then
      begin
        if (Self is TfcsCadastroGrid)
```

```
    and (TfcsCadastroGrid(Self).btInserir.Enabled) then
    begin
      TfcsCadastroGrid(Self).btInserir.Click();
      TfcsCadastroGrid(Self).grCadastro.SetFocus();
    end
    else if (Self is TfcsCadastroGridEdit)
    and (TfcsCadastroGridEdit(Self).btInserir.Enabled) then
    begin
      TfcsCadastroGridEdit(Self).btInserir.Click();
      TfcsCadastroGridEdit(Self).grCadastro.SetFocus();
    end;
  end;
end
else if Key = VK_DELETE then
begin
  Key := 0;
  bExcecutou := False;
  for i:=0 to ComponentCount-1 do
  begin
    oDataSource := GetDataSourceGridFocused(Components[i]);
    if oDataSource <> nil then
    begin
      for j:=0 to FBotoesSeta.Count-1 do
      begin
        if TcsBotao(FBotoesSeta.Items[j]).csEnabled
        and (FBotoesSeta.Items[j] is TcsBotaoDataSource) and
        (TcsBotaoDataSource(FBotoesSeta.Items[j]).csDataSource = oDataSource) then
        begin
          TcsBotaoDataSource(FBotoesSeta.Items[j]).Click();
          bExcecutou := True;
          Break;
        end; // fim IF, fim FOR.
      end;
      Break;
    end;  // fim IF, fim FOR.
  end;

  if bExcecutou = False then
  begin
    if (Self is TfcsCadastroGrid)
    and (TfcsCadastroGrid(Self).btExcluir.Enabled) then
    begin
      TfcsCadastroGrid(Self).btExcluir.Click();
      TfcsCadastroGrid(Self).grCadastro.SetFocus();
    end
    else if (Self is TfcsCadastroGridEdit)
    and (TfcsCadastroGrid(Self).btExcluir.Enabled) then
    begin
      TfcsCadastroGridEdit(Self).btExcluir.Click();
      TfcsCadastroGridEdit(Self).grCadastro.SetFocus();
    end;
  end;
 end;
end;
inherited KeyDown(Key, Shift);
end;

procedure TfcsCadastro.ObtemListaBotoesSeta(poBotoesSeta: TObjectList);
```

```
var
  i: Integer;
begin
  for i:=0 to ComponentCount-1 do
    if Components[i] is TcsBotaoDataSource then
      poBotoesSeta.Add(Components[i]);
end;

procedure TfcsCadastro.DoShow();
var
  i: Integer;
begin
  inherited DoShow();

  FPermEditar := csSeguranca.PermissaoEditar(Self.Name);

  if FPermEditar = False then
  begin
    FPermInserir := False;
    FPermExcluir := False;
    btSalvar.csEnabled := False;
    btRestaurar.csEnabled := False;

    if dsCadastro.DataSet <> nil then
    begin
      TcsEntidade(dsCadastro.DataSet).ReadOnly := True;

      for i:=0 to FEntidades.Count-1 do
        if (FEntidades.Items[i] <> TcsEntidade(dsCadastro.DataSet))
        and (TcsEntidade(FEntidades.Items[i]).csEntidadePai = TcsEntidade(dsCadastro.DataSet)) then
          TcsEntidade(FEntidades.Items[i]).ReadOnly := True;
    end;
  end
  else
  begin
    FPermInserir := csSeguranca.PermissaoInserir(Self.Name);
    FPermExcluir := csSeguranca.PermissaoExcluir(Self.Name);
  end;
end;

end.
```

# unit ucsSeguranca;

```
unit ucsSeguranca;

interface

uses
  ucsTiposDefinidos, ucsPermissoes;

type
  TcsSeguranca = class
  private
    FPermissao: TcsPermissoes;
```

```
    FIdUsuario: Integer;
    FNomeUsuario: string;
    FLoginUsuario: string;
    FUsuarioAtivo: Boolean;
    FIdGrupoUsuario: Integer;
    FNomeGrupoUsuario: string;
  public
    constructor Create();
    destructor Destroy(); override;
    { Se Login e Senha corretos, busca todos os dados do Usuário e devolve true,  senão devolve false. }
    function LogarUsuario(psLogin, psSenha: string): Boolean;
    // Criptografia
    function Criptografa(psString: string): string;
    function Descriptografa(psString: string): string;
    // Usuário ativo
    property IdUsuario: Integer read FIdUsuario;
    property NomeUsuario: string read FNomeUsuario;
    property LoginUsuario: string read FLoginUsuario;
    property UsuarioAtivo: Boolean read FUsuarioAtivo;
    property IdGrupoUsuario: Integer read FIdGrupoUsuario;
    property NomeGrupoUsuario: string read FNomeGrupoUsuario;
    // Permissões
    procedure AtualizaPermissoes();
    function PermissaoVisualizar(psNomeForm: string): Boolean;
    function PermissaoInserir(psNomeForm: string): Boolean;
    function PermissaoEditar(psNomeForm: string): Boolean;
    function PermissaoExcluir(psNomeForm: string): Boolean;
    function EstadoComponente(psNomeForm,
      psNomeComponente: string): TcsEstadoComponente;
  end;

var
  csSeguranca: TcsSeguranca;

implementation

uses
  uCrypt32, ucsUsuario, ucsEntidade;

{ TcsSeguranca }

constructor TcsSeguranca.Create();
begin
  FPermissao := TcsPermissoes.Create();

  FIdUsuario := 0;
  FNomeUsuario := '';
  FLoginUsuario := '';
  FUsuarioAtivo := True;
  FIdGrupoUsuario := 0;
  FNomeGrupoUsuario := '';
end;

destructor TcsSeguranca.Destroy();
begin
  FPermissao.Free();
  inherited;
end;
```

```
function TcsSeguranca.Criptografa(psString: string): string;
begin
  Result := Crypt32.Encrypt(psString);
end;

function TcsSeguranca.Descriptografa(psString: string): string;
begin
  Result := Crypt32.Decrypt(psString);
end;

function TcsSeguranca.LogarUsuario(psLogin, psSenha: string): Boolean;
var
  ecsUsuario: TecsUsuario;
  sSenha: string;
begin
  Result := False;
  ecsUsuario := TecsUsuario.Create(nil);
  try
    ecsUsuario.csSelect := 'SelectLogin';
    ecsUsuario.Params[0].AsString := psLogin;
    ecsUsuario.Open();
    if not ecsUsuario.IsEmpty then
    begin
      sSenha := Descriptografa(ecsUsuario.FieldByName('deSenha').AsString);
      if psSenha = sSenha then
      begin
        FLoginUsuario := psLogin;
        FIdUsuario := ecsUsuario.FieldByName('idUsuario').AsInteger;
        FNomeUsuario := ecsUsuario.FieldByName('nmUsuario').AsString;
        FUsuarioAtivo := ecsUsuario.FieldByName('flAtivo').AsString = 'S';
        FIdGrupoUsuario := ecsUsuario.FieldByName('idGrupoUsuario').AsInteger;
        FNomeGrupoUsuario := ecsUsuario.FieldByName('nmGrupoUsuario').AsString;

        FPermissao.AtualizaPermissoes(FIdGrupoUsuario);

        Result := True;
      end;
    end;
    ecsUsuario.Close();
  finally
    ecsUsuario.Free();
  end;
end;

procedure TcsSeguranca.AtualizaPermissoes();
begin
  FPermissao.AtualizaPermissoes(FIdGrupoUsuario);
end;

function TcsSeguranca.PermissaoVisualizar(psNomeForm: string): Boolean;
begin
  Result := FPermissao.PermissaoVisualizar(psNomeForm);
end;

function TcsSeguranca.PermissaoInserir(psNomeForm: string): Boolean;
begin
  Result := FPermissao.PermissaoInserir(psNomeForm);
end;
```

```
function TcsSeguranca.PermissaoEditar(psNomeForm: string): Boolean;
begin
  Result := FPermissao.PermissaoEditar(psNomeForm);
end;

function TcsSeguranca.PermissaoExcluir(psNomeForm: string): Boolean;
begin
  Result := FPermissao.PermissaoExcluir(psNomeForm);
end;

function TcsSeguranca.EstadoComponente(psNomeForm,
  psNomeComponente: string): TcsEstadoComponente;
begin
  Result := FPermissao.EstadoComponente(psNomeForm, psNomeComponente);
end;

initialization
  csSeguranca := TcsSeguranca.Create();

finalization
  csSeguranca.Free();

end.
```