UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Flávio Exterkoetter

Blendwork: Framework Orientado a Objetos para Desenvolvimento Rápido de Aplicações Comerciais Cliente/Servidor

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação.

Willy Arno Sommer, Dr.
Orientador

Luiz Carlos Zancanella, Dr.
Co-orientador

Florianópolis, Maio de 2003

Blendwork: Framework Orientado a Objetos para Desenvolvimento Rápido de Aplicações Comerciais Cliente/Servidor

Flávio Exterkoetter

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Fernando Alvaro Ostuni Gauthier, Dr.

Banca Examinadora

_____

Willy Arno Sommer, Dr. (Orientador)

_____

Luiz Carlos Zancanella, Dr. (Co-orientador)

_____

Ricardo Pereira e Silva, Dr.

_____

Patrícia Vilain, Dra.

Dedico este trabalho ao Arthur, meu filho.

Agradecimentos

A Universidade Federal de Santa Catarina, em especial ao Departamento de Pós-graduação em Ciência da Computação.

Ao orientador Prof. Willy Arno Sommer.

Ao co-orientador Prof. Luiz Carlos Zancanella.

A minha mãe e a toda minha família que sempre me apoiou.

Aos amigos Pablo Henrique Sell e Sérgio dos Santos pelos incentivos.

A todos que direta ou indiretamente contribuíram para a realização deste.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Atualmente, com o crescente aumento da complexidade dos sistemas computacionais, a tecnologia de frameworks orientados a objetos tem se mostrado cada vez mais importante tanto para empresas quanto para a área acadêmica devido aos bons resultados obtidos no desenvolvimento de aplicações.

Este trabalho apresenta o projeto do Blendwork, um framework orientado a objetos que tem como principal propósito fornecer uma infra-estrutura para facilitar o desenvolvimento de aplicações cliente/servidor em ambientes visuais de desenvolvimento orientados a objetos que se beneficiam da metodologia de Desenvolvimento Rápido de Aplicações (RAD). O Blendwork provê a arquitetura necessária ao desenvolvimento de aplicações cliente/servidor que interajam com bancos de dados para armazenamento de informações.

O projeto e a construção do Blendwork foram definidos através da análise de um domínio de aplicações previamente existentes e da experiência do autor desta dissertação no desenvolvimento de outras aplicações do mesmo domínio. A implementação, validação e evolução do framework advieram do desenvolvimento do Blendus, um sistema de gerenciamento de benefícios composto de seis aplicações, parcialmente apresentadas neste trabalho.

Palavras-chave

Framework orientado a objetos, Aplicações Cliente/Servidor, Desenvolvimento Rápido de Aplicações - RAD

Abstract

Currently, with the never-ending increase in computer software complexity, object-oriented framework technology has shown itself ever more important to companies and the academic world alike due to the satisfactory results it has obtained in application development.

This work presents the design of Blendwork, an object-oriented framework that has as its main purpose the providing of an infrastructure to facilitate the development of client/server applications in object-oriented visual builder environment that utilizes the Rapid Application Development (RAD) methodology. Blendwork provides the necessary architecture for the development of client/server applications that interact with database for information storage.

Blendwork's designing and building was defined both through the domain analysis of previously existing applications and this dissertation author's experience in the development of other applications from the same domain. The implementation, validation and evolution came from the development of Blendus, a benefits management system made up of six applications, partially presented in this work.

Key-words

Object-oriented frameworks, Client/Server Applications, Rapid Application Development - RAD

1. Introdução

1.1. Visão Geral

Mesmo com a grande oferta de tecnologia disponível no mercado, muitas empresas ainda se defrontam com sérios problemas no desenvolvimento de aplicações. Em virtude da crescente complexidade dos softwares, faz-se necessária a adoção de soluções mais eficientes para diminuição dos esforços e dos custos destinados a este desenvolvimento.

Durante as duas últimas décadas, muitas técnicas e tecnologias foram desenvolvidas. A promessa era diminuir os custos envolvidos no desenvolvimento de software através de uma eficiente reutilização dos recursos já desenvolvidos e, conseqüente, aproveitamento das soluções concretizadas. Algumas delas – incluindo programação orientada a objetos e linguagens de quarta geração – tiveram sucesso limitado e não trouxeram benefícios substanciais. As bibliotecas de classes reutilizáveis, em particular, falharam parcialmente devido à dificuldade de implementação de classes suficientemente genéricas para atender a todos os contextos possíveis.

Frameworks orientados a objetos tentam resolver este problema ampliando o sucesso parcial das bibliotecas de classes reutilizáveis pela redução do seu escopo e pela colaboração entre as classes. De acordo com GAMMA (1995), um framework é um conjunto de classes cooperativas que formam um projeto reutilizável para uma categoria específica de software, fornecendo direcionamento arquitetural pelo particionamento do projeto em classes abstratas, definindo suas responsabilidades e colaborações. Assim, um desenvolvedor customiza um framework para uma aplicação específica, especializando e compondo instâncias destas classes.

Os ambientes visuais de desenvolvimento disponíveis no mercado que utilizam a metodologia de Desenvolvimento Rápido de Aplicações, fornecem um grande conjunto de classes necessárias ao desenvolvimento de aplicações, porém não fornecem direcionamento arquitetural nem uma colaboração satisfatória para aplicações de domínios específicos. Este trabalho objetiva reduzir a lacuna existente entre estes

ambientes visuais de desenvolvimento e a arquitetura necessária para aplicações comerciais cliente/servidor.

1.2. Motivação

São vários os problemas que podem surgir durante e após o desenvolvimento de uma aplicação, dentre os quais podemos citar:

- Falta de treinamento da equipe de desenvolvimento;
- Falta de metodologia de desenvolvimento;
- Falhas nas etapas de análise e projeto anteriores ao desenvolvimento;
- Prazos reduzidos;
- Falta de documentação dos problemas já resolvidos;
- Dificuldade na reutilização de soluções comuns nas aplicações;
- Despreparo das empresas quanto a uma estrutura básica e padronizada de suas aplicações;
- Falta de integração entre aplicações implantadas em diversos setores de uma mesma organização.

Problemas, como os descritos acima, necessitam de soluções com custos viáveis, para que as empresas sobrevivam no mercado e melhorem a qualidade de suas aplicações. Uma empresa de informática que tenha duas ou três aplicações não padronizadas e de difícil manutenção implantadas nos seus clientes corre o risco de precisar alocar grande parte de sua equipe na manutenção destas aplicações. Isto tende a causar uma diminuição nos lucros, pois geralmente são os novos projetos que geram maior receita, entretanto, os mesmos poderão estar parcialmente comprometidos, uma vez que parte da equipe já está alocada.

Alguns dos problemas citados agravam-se ainda mais quando não existe a ciência, por parte dos dirigentes das empresas de informática, da importância da resolução destes problemas e da necessidade da adoção de medidas que possibilitem suas soluções

efetivas. Soluções estas que podem envolver diversos ciclos da metodologia de desenvolvimento adotada pela empresa.

Uma das medidas adotadas pela Extersoft Informática, empresa do ramo de desenvolvimento de software, foi a decisão de melhorar a reutilização das funcionalidades implementadas com sucesso em projetos anteriores, agrupando-as e documentando-as para que pudessem ser reutilizadas de uma forma mais efetiva no desenvolvimento de novas aplicações. O trabalho desta dissertação visa atender exatamente esta necessidade.

1.3. Desafios

Desenvolver aplicações é uma tarefa difícil, mas construir frameworks de alta qualidade, extensíveis e reutilizáveis para um determinado domínio de aplicações, é ainda mais difícil (FAYAD, 1999).

Considerando a metodologia de Projeto Dirigido por Exemplo (JOHNSON, 1993), uma das adotadas neste trabalho, um framework é geralmente desenvolvido de forma repetitiva, num processo de evolução. Depois de pronta a primeira versão do framework, inicia-se o desenvolvimento de uma ou mais aplicações onde o framework será avaliado e testado, podendo ser identificadas novas abstrações e/ou correções que, quando implementadas, trazem conseqüências para as aplicações já desenvolvidas. A construção e evolução do Blendwork ocorreram desta forma, sendo que as aplicações foram disponibilizadas aos usuários finais depois de concluídas. Várias das alterações e novas abstrações incorporadas ao Blendwork durante a sua evolução resultaram em modificações nas aplicações que já haviam sido concluídas, aumentando assim os custos. Logo, um dos grandes desafios foi conciliar a evolução do Blendwork mantendo as aplicações já desenvolvidas compatíveis com o mesmo.

1.4. Objetivos do Trabalho

1.4.1. Objetivo Geral

O principal objetivo deste trabalho é a construção de um framework orientado a objetos, flexível e extensível, que forneça uma infra-estrutura para o desenvolvimento de aplicações cliente/servidor de duas camadas, com bancos de dados, em ambientes visuais de desenvolvimento orientado a objetos.

1.4.2. Objetivos Específicos

Para a concretização do objetivo geral mencionado acima foram estabelecidos os seguintes objetivos específicos:

- Estudar os principais conceitos relacionados aos frameworks orientados a objetos;
- Definir uma arquitetura adequada para o desenvolvimento de aplicações do domínio estudado;
- Identificar as funcionalidades comuns em aplicações já existentes no domínio a que o framework proposto se destina;
- Reduzir os custos no desenvolvimento e manutenção de aplicações RAD;
- Padronizar as interfaces gráficas de usuário para a entrada, consulta e manipulação de dados inerentes a aplicação final e ao dicionário de dados framework;
- Padronizar as saídas de dados (relatórios) das aplicações sobre ele desenvolvidas;
- Controlar o acesso às aplicações e suas funcionalidades (segurança da aplicação);
- Fornecer controle do fluxo de execução das aplicações (chamadas para a criação dos formulários e navegação entre eles).

1.5. Apresentação do Trabalho

A presente dissertação está organizada em nove capítulos. Os primeiros quatro apresentam a introdução e a fundamentação teórica; o quinto apresenta alguns exemplos de frameworks disponíveis no mercado; do sexto ao oitavo são relatadas e documentadas as etapas e decisões do projeto de desenvolvimento do Blendwork.

O capítulo 2 cobre os assuntos relacionados a frameworks orientados a objetos. Nele são apresentados conceitos, benefícios, classificações e tecnologias relacionadas ao uso de frameworks, bem como os seus pontos fortes e fracos. Mostra ainda a evolução e as arquiteturas internas de construção de um framework.

A arquitetura de aplicações cliente/servidor e seus modelos são detalhados no Capítulo 3, onde são apresentadas vantagens e desvantagens de cada modelo.

No capítulo 4, um breve histórico e fundamentação da metodologia de Desenvolvimento Rápido de aplicações são apresentados, juntamente com algumas recomendações sobre seu uso.

O estudo de alguns frameworks orientados a objetos existentes no mercado, com a análise de seu escopo é apresentado no capítulo 5. Este estudo tem o objetivo de facilitar a identificação de um domínio de aplicação específico que ainda não tenha sido diretamente beneficiado.

O capítulo 6 apresenta o processo de construção do Blendwork de forma conceitual, descrevendo a metodologia utilizada, o cenário onde ele foi construído e as abstrações identificadas no domínio de aplicações analisado.

A implementação do Blendwork é apresentada no capítulo 7, através de diagramas de classes, interfaces gráficas padronizadas, diagramas de entidade e relacionamento do dicionário de dados e a menção dos padrões de projeto utilizados.

O projeto e implementação do Blendwork são validados no capítulo 8, onde são apresentados alguns resultados, ou seja, aplicações desenvolvidas utilizando o Blendwork.

Finalmente, no capítulo 9 constam as considerações finais e sugestões para trabalhos futuros.

2. Frameworks

"Parece que nunca temos tempo para fazer as coisas direito,
mas parece que sempre temos tempo para refazê-las".

Gause e Weinberg

2.1. O que é um Framework Orientado a Objetos?

2.1.1 Definição

De acordo com GAMMA (1995) um framework pode ser definido como:

> *Um conjunto de classes que cooperam entre si e compõem um projeto reutilizável para uma categoria específica de software. Um framework fornece direcionamento arquitetural do software, através do particionamento do projeto em classes abstratas e da definição de suas responsabilidades e colaborações. Um desenvolvedor customiza o framework, para uma aplicação particular, através da especialização e da composição de instâncias de classes do mesmo.*

Dentre outras definições de framework há a de JOHNSON & FOOTE (1988), bastante referenciada e amplamente aceita: "um framework é um conjunto de classes que incorpora um projeto abstrato para soluções destinadas a uma família de problemas relacionados".

Frameworks de aplicação orientados a objetos constituem uma tecnologia promissora para concretização de projetos de software testados e implementados, com objetivo de custos e melhor qualidade de software (JOHNSON & FOOTE, 1988). Técnicas anteriores de OO (orientação a objetos), como as bibliotecas de classes, tendem a ser construídas para serem utilizadas de forma bastante genérica, em quaisquer tipos de aplicação. Já os frameworks são direcionados para um domínio específico de aplicações – como interfaces gráficas com usuários – ou para determinada área de negócio – como comércio eletrônico ou aplicações comerciais.

Numa definição um pouco mais detalhada, JOHNSON & FOOTE (1988) definem um framework como um projeto reutilizável de um sistema que descreve como o sistema é decomposto em um conjunto de objetos que se interagem, podendo ser uma aplicação completa ou somente um subsistema. O framework descreve ambos através da interação dos objetos componentes, expondo a interface de cada objeto e a colaboração entre eles, e mostrando também como as responsabilidades do sistema são mapeadas para os objetos.

As partes comuns do framework são as partes estáveis, também conhecidas como *cold spots.* As partes mantidas flexíveis, ou adaptáveis dentro do framework são conhecidas com *hot spot,* e podem ser incompletas. O usuário de um framework irá estender as funcionalidades já implementadas através destas partes flexíveis, produzindo uma aplicação completa. As técnicas utilizadas para estender um framework serão tratadas posteriormente neste capítulo.

Um framework captura decisões de projeto que são comuns ao seu domínio de aplicação, enfatizando a reutilização de projetos em relação à reutilização de código, embora, geralmente, incluindo subclasses concretas (GAMMA, 1995). Deutsch ressalta que a parte mais importante de um framework refere-se à maneira que um sistema é dividido em seus componentes. Os frameworks também reutilizam implementação, mas isto é menos importante que a reutilização das interfaces internas de um sistema e o modo como suas funções são divididas aos seus componentes (DEUTSCH, 1989 *apud* FAYAD, 1999).

2.1.2 Indivíduos envolvidos

Segundo SILVA (2000), existem dois tipos de indivíduos envolvidos no desenvolvimento tradicional de aplicações: aquele que desenvolve a aplicação e aquele que a utiliza. Os encarregados do desenvolvimento devem apurar as exigências de uma aplicação, desenvolvê-la e entregá-la aos usuários. O modo de interação dos usuários ocorrerá apenas através da interface de usuário da aplicação. A Figura 2.1 apresenta os indivíduos envolvidos.

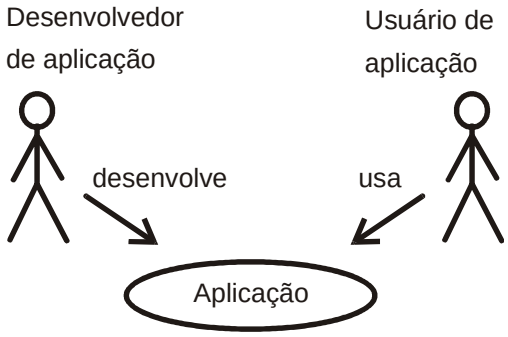

Figura 2.1 – Elementos do desenvolvimento tradicional de aplicações.

Fonte: (SILVA, 2000)

Já com a utilização de frameworks, SILVA (2000) ressalta que além do desenvolvedor e do usuário de aplicação, há outro indivíduo envolvido: o desenvolvedor de framework. Neste contexto, o papel do usuário permanece o mesmo, como descrito anteriormente, porém, modifica-se o papel do desenvolvedor de aplicações, justamente pela inserção do framework. Neste caso, ele passa a ser um usuário de um framework, devendo estender e adaptá-lo para desenvolver suas aplicações. As funções do desenvolvedor de aplicações permanecem inalteradas, isto é, ele deve continuar a obter os requisitos da aplicação, desenvolvê-la – só que usando o framework – e elaborar a documentação da aplicação. Assim, o desenvolvedor de framework tem a responsabilidade de produzir frameworks e de determinar, de algum modo, uma forma de ensino para utilizá-los na produção de aplicações. A Figura 2.2 apresenta os indivíduos envolvidos neste caso.

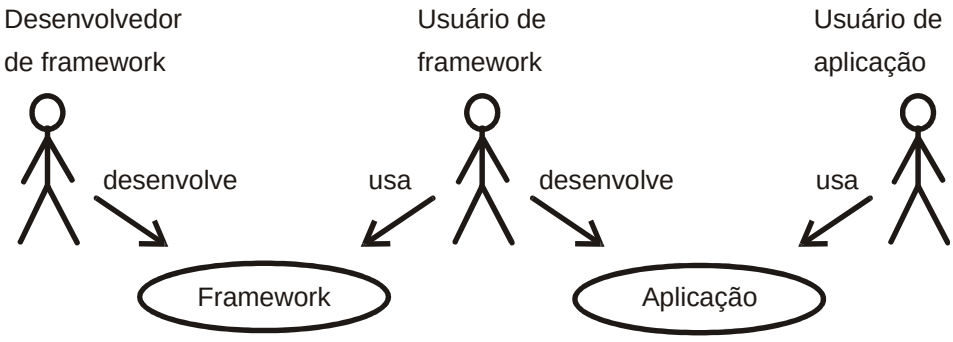

Figura 2.2 – Elementos do desenvolvimento de aplicações baseado em frameworks.

Fonte: (SILVA, 2000)

2.2. Classificando Frameworks

De acordo com FAYAD (1999), embora os benefícios e princípios de projeto sobre frameworks sejam largamente independentes do domínio no qual eles são aplicados, é importante classificá-los de acordo com o escopo, como segue:

- **frameworks de infra-estrutura de sistema:** simplificam o desenvolvimento de infra-estruturas de sistemas portáveis e eficientes, como por exemplo, um sistema operacional, frameworks de comunicação e frameworks para interfaces de usuário e ferramentas de processamento de linguagens;
- **frameworks de integração *middleware*:** comumente usados para integrar aplicações e componentes distribuídos, encapsulando as implementações necessárias neste contexto. Exemplos comuns incluem implementações de frameworks ORB, *middleware* orientado a mensagens e transações de banco de dados;
- **frameworks de aplicação empresarial:** estes frameworks destinam-se a um largo domínio de aplicações destinadas a usuários finais – como telecomunicações, manufatura, engenharia financeira – e representam a base para as atividades de negócios das empresas.

Independente do escopo, FAYAD (1999) ressalta que os frameworks também podem ser classificados pelas técnicas utilizadas para estendê-los, que variam em continuidade de frameworks caixa-branca para frameworks caixa-cinza e para frameworks caixa-preta, conforme descrito abaixo:

- **Caixa-branca:** são extensíveis através da utilização de herança. As funcionalidades existentes são reutilizadas e estendidas por herança de classes base do framework e sobrescrevendo métodos *hook*[1]. Para estender frameworks caixa-branca é recomendado que os usuários de framework tenham um profundo conhecimento da estrutura interna do framework. As

[1] Métodos *hook* são pontos onde um framework orientado a objetos pode ser adaptado pelo desenvolvedor de aplicações. Os métodos *hook* são invocados pelos *template*.

aplicações produzidas sob frameworks caixa-branca estão fortemente ligadas a hierarquia de classes do framework;

- **Caixa-preta:** suportam extensibilidade pela definição de interfaces para componentes que podem ser conectados em um framework através de composição de objeto. Os objetos compostos não revelam detalhes internos uns aos outros e, assim, são análogos a “caixas pretas”. Outras funcionalidades existentes são reutilizadas pela definição de componentes que irão se adequar a uma determinada interface e pela interação destes componentes em um framework. Os frameworks caixa-preta são geralmente mais fáceis de usar e estender do que os caixa-branca, porém são mais difíceis de desenvolver, já que devem prever uma maior variação das funcionalidades a serem implementadas na aplicação;
- **Caixa-cinza:** são caracterizados pela utilização das duas técnicas de reutilização: herança de classe e composição de objetos. Desta forma, algumas funcionalidades do framework caixa-cinza podem ser estendidas pela especialização das classes do framework, e outras pela composição dos objetos providos pelo framework. Os itens 2.3.3, 2.3.4 e 2.3.5 fazem uma avaliação mais detalhada dos conceitos e aplicabilidade das caixas.

2.3. A Evolução de um Framework

EARLES (2000b) resume a evolução de um framework, conforme as subseções a seguir, baseando-se num padrão para desenvolvimento de frameworks orientados a objetos de ROBERTS & JOHNSON (1996).

2.3.1. O Nascimento – Repetição Observada

Um framework é concebido como qualquer outro projeto que tenha uma necessidade de negócio. A diferença é que enquanto um projeto normal é resultado de uma necessidade de negócio individual, um projeto de framework é resultado de uma necessidade de negócio observada repetitivamente. Sem uma observação repetitiva, a criação de um framework é confundida com uma solução a ser aplicada uma única vez,

resultando numa significativa demora no ciclo de desenvolvimento, além de uma solução mais complexa.

Quando algumas implementações são necessárias em vários sistemas, há o surgimento de um padrão e, como todos os padrões, isso deve ser identificado como uma excelente oportunidade para reuso.

Esta etapa da descoberta do reuso é crucial para a criação do processo de desenvolvimento da solução que poderá ser reutilizada. Sem um planejamento para a reutilização, as oportunidades de reuso serão tardiamente identificadas pelo processo de desenvolvimento, resultando em duplicação de dados e comportamentos pela organização.

2.3.2. Projeto

Existe uma grande diferença entre o projeto e arquitetura de uma aplicação e de um framework. O principal desafio no projeto de um framework é permitir que futuras aplicações sejam desenvolvidas mais facilmente. Para isso, o framework deve incorporar flexibilidade, extensibilidade e facilidade na configuração.

Padrões de projeto formam a espinha dorsal de uma arquitetura de framework flexível, encapsulando as partes adaptáveis dentro das classes ou componentes. Este encapsulamento permite a troca de componentes com a mesma interface, enquanto que a sua implementação varia.

2.3.3. Adolescência – Caixa Branca

Frameworks caixa-branca implicam que os usuários de framework terão acesso ao código fonte e usarão herança para derivar novas variações das classes base ou abstratas. O uso de um framework caixa-branca requer um grande conhecimento da sua implementação, exigindo documentação detalhada, como os *cookbooks*, e uma longa curva de aprendizado para tornar possível o seu uso de forma eficiente e efetiva.

O estágio caixa-branca do ciclo de vida de um framework é similar à adolescência humana. Ele está tentando determinar sua razão de existência, compartilhando seus

valores e individualidade. Durante esta fase é necessário continuar desenvolvendo aplicações sob o framework, pois é somente com essa repetição que conseguirar-se-á solidificar a adaptabilidade (*hot spots*) do estável (*cold spots*), e o genérico do específico.

Deste ponto em diante, o framework passa a justificar seus investimentos, e deveria ter seu próprio gerenciamento e equipe de desenvolvimento. No desenvolvimento de novas aplicações, durante a repetição das fases de análise, projeto e implementação, os frameworks serão estendidos para atender as necessidades específicas de cada aplicação, e os desenvolvedores de framework devem ser notificados de qualquer falha importante. Depois da aplicação estar completa, os resultados do reuso devem ser reavaliados. A reavaliação permitirá um contínuo refinamento e evolução do framework, identificando mais pontos adaptáveis, incorporando-os no framework (para uso futuro) ou até mesmo retirando do framework partes que se consideravam genéricas.

2.3.4. Maturidade – Caixa Preta

Os frameworks caixa-branca dependem da herança ou combinação estática para permitir reuso de código, mais a habilidade de sobrepor funcionalidades genéricas por específicas. A desvantagem é que uma subclasse está profundamente interligada com sua superclasse em termos da forma de implementação e dependência.

A medida em que os frameworks evoluem, ganhando experiência de múltiplas aplicações e ciclos de reavaliação, eles começam a se tornar mais fáceis para uma alternativa de modelo de reuso de código, que é a composição polimórfica. Com a composição, depende-se da possibilidade de substituição das interfaces e a capacidade de selecionar a interface apropriada durante a implantação ou em tempo de execução.

Este modelo de composição é bastante aplicado no mundo dos componentes, especialmente os de negócio, que são construídos em torno dos conceitos de interfaces, composição e delegação. Isso é que torna parcialmente possível uma das grandes vantagens dos componentes: funcionamento independente da linguagem.

Uma vez que o framework tenha evoluído suficientemente, ele pode se tornar estático, visto que não é requerida herança do framework. Qualquer configuração pode ser feita através da troca de componentes com a mesma interface obtendo assim, um novo comportamento. À primeira vista, isso parece limitar severamente a utilidade do framework, mas na realidade o framework entrou altamente “em contato” com o seu domínio. As várias implementações utilizando o framework validaram as partes onde a adaptabilidade era ou não necessária. Como as interfaces foram ricamente especificadas, o framework se tornou mais fácil de usar e aprender, e pode ser migrado de versão mais facilmente.

2.3.5. Caixa Cinza

Caixa-branca e caixa-preta formam os limites extremos do projeto de frameworks e suas técnicas de uso. A maioria dos frameworks vão provavelmente estar entre estes dois extremos, como um framework caixa-cinza.

Um framework baseado em herança (caixa-branca) é mais fácil de construir, atinge maior grau de flexibilidade, porém é mais difícil de utilizar. Já o baseado em composição de objetos (caixa-preta) é mais fácil estender para uma nova aplicação, mas é mais difícil de construir. Como no mundo dos negócios tempo e dinheiro precisam ser considerados, um framework pode parar de evoluir quando se torna funcional podendo beneficiar-se das duas técnicas utilizadas para estendê-los: herança de classes e composição de objetos, tornando-se assim um framework caixa-cinza.

2.4. Metodologias de Desenvolvimento

SILVA (2000), descreve três metodologias voltadas ao desenvolvimento de frameworks:

- **Projeto Dirigido por Exemplo:** Proposto por JOHNSON (1993), o framework é projetado observando-se aplicações existentes relacionadas àquele domínio de aplicações. Os aspectos semelhantes de diferentes aplicações podem dar origem a classes abstratas, sendo que as particularidades

podem ser implementadas em classes concretas no desenvolvimento de uma nova aplicação utilizando o framework.

- **Projeto Dirigido por *Hot Spot*:** Os *hot spots* são as partes do framework mantidas flexíveis, ou seja, as partes que diferem entre as aplicações. Esta metodologia, proposta por PREE (1995 *apud* FAYAD, 1999), objetiva construir frameworks identificando estas partes flexíveis na estrutura de classes de um domínio. As etapas a serem seguidas são, respectivamente: identificação das classes, identificação dos *hot spots*, (re)projeto do framework (onde podem ser aplicados os padrões de projeto) e adaptação do framework. Durante a quarta etapa pode-se constatar a necessidade de um maior grau de flexibilidade, devendo-se então retornar à etapa de identificação dos *hot spots* até que o framework esteja concluído.
- **Metodologia de Projeto da Empresa Taligent:** Objetiva desenvolver aplicações através de múltiplos frameworks, estruturalmente menores e mais simples. Dessa forma, por serem menores e mais flexíveis, os frameworks podem ser utilizados mais freqüentemente (TALIGENT, 1994). Logo, esta metodologia está direcionada a desenvolver frameworks pequenos englobando apenas aspectos específicos do domínio.

2.5. Uso de Padrões de Projeto na Construção de Frameworks

Os padrões de projeto representam soluções recorrentes para problemas de desenvolvimento de software dentro de um determinado contexto. Padrões e frameworks juntos facilitam o reuso através da captura de estratégias de sucesso no desenvolvimento de software. A principal diferença entre eles é que os frameworks estão focados no reuso de um projeto concreto, algoritmos e implementações em uma linguagem de programação específica, enquanto os padrões estão focados no reuso de projetos abstratos e micro-arquiteturas de software. Por este motivo, os padrões são mais abstratos que framework (FAYAD, 1997).

GAMMA (1995) explica o importante papel que os padrões de projeto desempenham no desenvolvimento de frameworks citando, primeiramente, pontos críticos no projeto de um framework, a saber:

- a arquitetura da aplicação é definida pelo framework;
- divisões do framework em classes e objetos;
- responsabilidades-chave das classes dos objetos e suas colaborações;
- fluxo de controle do framework;
- evolução do framework e conseqüente evolução das aplicações tornando importantíssimo o fraco acoplamento;
- reutilização de projeto;
- reutilização de código.

Sobre estes pontos críticos na construção de um framework, GAMMA (1995) afirma que:

> *"Um framework que os trata através do uso de padrões de projeto tem maior probabilidade de atingir altos níveis de reusabilidade de projeto e código, comparando 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."*

Ainda segundo GAMMA (1995), os padrões de projeto diferenciam-se de frameworks principalmente em três aspectos:

- Padrões de projeto são mais abstratos que frameworks: Os frameworks são concretizados através de sua implementação (código fonte), já no caso dos padrões só os exemplos podem ser materializados através de código;
- Padrões de projeto são elementos de arquitetura menores que frameworks: um framework pode conter vários padrões, mas um padrão nunca pode conter um framework;

- Padrões de projeto são menos especializados que frameworks: Os padrões de projeto podem ser utilizados em qualquer tipo de aplicação, mas os frameworks são direcionados a um domínio de aplicações específico.

Além dos padrões de projeto, FAYAD (1997) ressalta a importância de outras duas abordagens de reuso relacionadas aos frameworks, que são: bibliotecas de classes e componentes.

2.6. Benefícios no Uso de Frameworks

Os principais benefícios dos frameworks de aplicações orientados a objetos vêm da modularidade, reusabilidade, extensibilidade e inversão de controle que eles provêm para os desenvolvedores, como descrito abaixo (FAYAD, 1999):

- **modularidade:** através do uso de interfaces estáveis, os frameworks possibilitam maior modularidade encapsulando os detalhes de implementação volátil, melhorando a qualidade do software e identificando os pontos de maior impacto das alterações de projeto e implementação;
- **reusabilidade:** as interfaces estáveis providas pelos frameworks facilitam a reutilização de componentes genéricos em novas aplicações. As definições das interfaces e a maneira com que suas funções são atribuídas entre os componentes são mais importantes que a implementação, também provida pelos frameworks;
- **capacidade de extensão:** um framework aumenta a capacidade de extensão pelo fornecimento explícito de métodos *hook* (PREE, 1995 apud FAYAD, 1999), que permitem às aplicações estender suas interfaces estáveis;
- **inversão de controle:** é uma característica dos frameworks que embute a vantagem de implementar o fluxo de controle da aplicação. Diferentemente de uma biblioteca de classes – onde as aplicações chamam suas funções e instanciam suas classes e onde não há fluxo de controle, interações ou colaborações pré-definidas – a arquitetura em tempo de execução dos frameworks podem efetuar chamadas a funções das aplicações, controlar o

fluxo de execução, definir interações e colaborações entre as classes que compõem o framework e/ou entre as classes estendidas pelo usuário de framework (TALIGENT, 1994). Esta inversão de controle em tempo de execução é referenciada como o Princípio de Hollywood: "*Don't call us, we'll call you*".

2.7. Pontos Fracos de Frameworks

Os frameworks orientados a objetos podem aumentar significativamente a qualidade das aplicações e reduzir seu esforço de desenvolvimento, todavia existem alguns desafios a serem vencidos para se empregar os frameworks de maneira eficiente. Para se obter sucesso na construção de utilização de frameworks, precisa-se reconhecer e resolver alguns desafios como o esforço de desenvolvimento, curva de aprendizado, integrabilidade, manutenibilidade, validação e remoção de defeitos, eficiência e falta de padronização, os quais estão detalhados abaixo conforme (FAYAD, 1999):

- **esforço de desenvolvimento:** desenvolver softwares complexos já é bastante difícil, mas desenvolver frameworks de alta qualidade, extensíveis e reutilizáveis para aplicações complexas é ainda mais difícil. O conhecimento necessário para o sucesso na produção de frameworks geralmente fica restrito aos desenvolvedores experientes;
- **curva de aprendizado:** o esforço de aprendizado requerido dos usuários de framework é considerável. Faz-se necessária a disponibilização da documentação do framework e de alguns exemplos de aplicações implementadas. Dependendo da complexidade do framework, mentores e cursos de treinamento são necessários para ensinar aos usuários de framework como utilizá-lo de forma eficiente;
- **integrabilidade:** é preciso levar em consideração a integração de múltiplos frameworks, bibliotecas de classes, sistemas legados e componentes existentes empregados no desenvolvimento de uma aplicação;

- **manutenibilidade:** É difícil manter a compatibilidade entre os frameworks e as aplicações sobre ele desenvolvidas, pois os frameworks evoluem e tornam-se mais maduros com o tempo e as aplicações devem evoluir com ele (MATTSON, 1996);
- **validação e remoção de defeitos:** embora frameworks bem projetados e modularizados possam minimizar o impacto dos defeitos das aplicações, a validação e a depuração de frameworks podem ser mais problemáticas, pois componentes genéricos são mais difíceis de validar de forma abstrata. A inversão de controle provida pelos frameworks também pode dificultar a depuração.
- **eficiência:** os frameworks aumentam a extensibilidade empregando níveis adicionais não direcionados. Por exemplo, ligação dinâmica é comumente usada para permitir aos desenvolvedores criar subclasses e personalizar interfaces existentes. Todavia, a generalidade e a flexibilidade reduzem a eficiência.
- **falta de padronização:** atualmente, não existe padronização amplamente aceita para projeto, implementação, documentação e adaptação de frameworks.

2.8. Considerações

ROBERTS & JOHNSON (1996) ressaltam que apesar de a construção de um bom framework ser cara, pode-se reduzir o custo no desenvolvimento de várias aplicações, reutilizando projeto e implementação. Frameworks não requerem uma nova tecnologia, pois podem ser implementados com linguagens orientadas a objetos existentes no mercado. Um framework deve incorporar as abstrações de um domínio de problemas relacionados, e é sempre o resultado de uma análise de exemplos concretos deste domínio. Eles devem ser suficientemente simples para que possam ser aprendidos, mas devem implementar funcionalidades que tornem o desenvolvimento de aplicações rápido, e pontos adaptáveis para que haja possibilidade de estendê-los.

3. Aplicações Cliente/Servidor

3.1. História e Definição

É importante mencionar, antes de explicar e definir a arquitetura cliente/servidor, as arquiteturas anteriores a ela:

- **Arquitetura *Mainframe*:** nas arquiteturas de software *mainframe* todo o processamento está centralizado em um computador central. Terminais, também conhecidos como “terminais burros”, são usados para acessar os programas e processos, para entrada de dados e visualizar relatórios; no entanto, todo processamento é feito em um único computador central. Os terminais apenas possibilitam a entrada de dados e exibem os resultados do processamento executado pelo servidor. As interações também podem ser feitas através de computadores pessoais ou estações UNIX por meio de emuladores, porém o processamento é centralizado (STEELE, 2000);
- **Arquitetura de Compartilhamento de Arquivos:** Quando os computadores pessoais começaram a ser interligados em rede surgiu a arquitetura de compartilhamento de arquivos, em que um servidor compartilha um conjunto de arquivos e os disponibiliza aos computadores clientes. O processamento da aplicação é concentrado no computador cliente, incluindo a interface com o usuário, lógica de negócio e os dados que são transferidos do servidor. Esta arquitetura foi popularizada com os produtos Xbase, como dBASE, FoxPro e Clipper. Ela funciona adequadamente com um pequeno grupo de computadores clientes e baixa transferência de volume de dados.

Como resultado das limitações das arquiteturas anteriores – *mainframe* e compartilhamento de arquivos – surgiu a arquitetura de aplicações cliente/servidor. Com esta nova abordagem, os arquivos compartilhados foram substituídos por um servidor de banco de dados. A arquitetura cliente/servidor diminui o tráfego na rede fornecendo respostas a consultas ao invés de transferir todo o arquivo. No seu modelo mais simples, dois computadores (um cliente e um servidor) dividem o processamento das aplicações, onde a aplicação cliente interage com o usuário através de sua interface,

enviando solicitações de dados ao servidor que, por sua vez, envia os dados solicitados ao cliente. Em outros modelos mais complexos desta arquitetura o processamento pode ser distribuído a uma grande quantidade de servidores. Em todos os modelos a característica chave da arquitetura cliente/servidor é a separação do processamento das funções computacionais das aplicações entre computadores clientes e servidores.

Um cliente é definido como um solicitador de serviços, e um servidor como um provedor de serviços. Dependendo da configuração do software, um computador pode funcionar como um cliente e ou como um servidor, mas tipicamente um cliente é uma aplicação executada em um computador pessoal e depende de um servidor para processar algumas operações. Por exemplo, um cliente de correio eletrônico, como o Outlook Express, é uma aplicação cliente que possibilita o envio e recebimento de mensagens através de um servidor de correio eletrônico. Embora aplicações cliente/servidor possam ser implementadas para diversos fins, como servidor de impressão, acesso a páginas da internet, servidor de fax, esta dissertação será direcionada a aplicações envolvendo servidor de banco de dados, onde o SQL (*structured query language*) é tipicamente utilizado para a comunicação entre cliente e servidor.

GALLAUGHER (1996) define a arquitetura cliente/servidor como:

> *Na sua forma mais fundamental, cliente/servidor envolve uma entidade de software (cliente) fazendo uma solicitação específica que é atendida por outra entidade de software (servidor). O processo cliente envia a solicitação para o servidor. O servidor interpreta a mensagem e então tenta atender a solicitação. Para atender a solicitação, o servidor pode ter que acessar um banco de dados, processar dados (cálculos), controlar periféricos ou fazer uma solicitação adicional a outro servidor. Em muitas arquiteturas, um cliente pode fazer solicitações a múltiplos servidores e um servidor pode servir múltiplos clientes.*

Outra definição é a de CAMBIOTIS (2001):

> *O termo arquitetura cliente/servidor é uma descrição genérica de um sistema de rede onde um programa cliente inicia contato com outro programa servidor (possivelmente em outro computador) para um função*

> *ou propósito específico. O cliente existe no papel de solicitador do serviço, fornecido pelo servidor.*

A arquitetura cliente/servidor começou a ter aceitação no final dos anos 80 e, comparando-se com a arquitetura centralizada dos *mainframes* e do compartilhamento de arquivos, ela é uma estrutura modular, versátil e baseada em mensagens que se destina a melhorar a usabilidade, flexibilidade, interoperabilidade e escalabilidade (SADOSKI, 1997).

3.2. Modelos da Arquitetura Cliente/Servidor

A grande maioria das aplicações para usuário pode ser dividida em três partes: apresentação, processamento ou lógica de negócio e dados. A arquitetura cliente/servidor pode ser classificada pela maneira com que estes três elementos podem ser divididos entre as entidades de software e distribuídos pela rede. Esta seção trata de classificar e descrever estes modelos de arquitetura cliente/servidor.

3.2.1. Modelo Cliente/Servidor Duas Camadas

Essa abordagem foi desenvolvida no início dos anos 80 com um grande avanço em relação ao compartilhamento de arquivos, substituindo o servidor de arquivos por um servidor de banco de dados. Nesta abordagem o cliente envia uma solicitação (através de comandos SQL) ao servidor, que a processa retornando exatamente os dados que foram solicitados, ao invés de todo o arquivo, como acontecia no compartilhamento de arquivos. Isto reduz o tráfego da rede e reflete um ambiente para múltiplos usuários. Este modelo possui subdivisões, a saber:

- **Cliente Robusto** (*fat client*): A aplicação cliente implementa, além da apresentação (interface com o usuário), toda a lógica de negócio, deixando para o servidor apenas a função de armazenar os dados e fazer validações básicas de dados (chaves primárias, chaves estrangeiras, campos que não podem ser nulos) que garantam a integridade dos dados.
- **Cliente Fraco** (ou *thin client*): O servidor é responsável pelo armazenamento dos dados e pela implementação da lógica de negócio. O

cliente trata apenas da apresentação e depende do servidor para as outras duas funções.

- Além das duas subdivisões apresentadas imediatamente acima, uma aplicação cliente/servidor pode ser implementada dividindo a lógica de negócio entre o cliente e o servidor, geralmente avaliando-se onde teria maior eficiência.

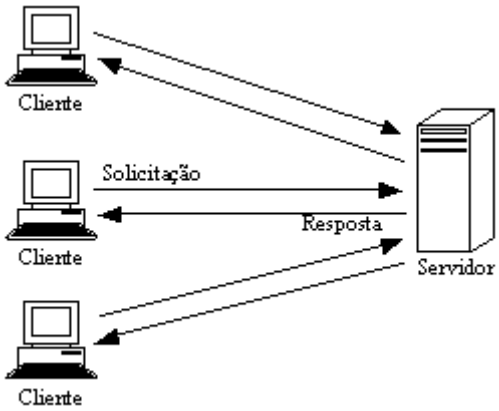

Figura 3.1 – Arquitetura cliente/servidor duas camadas

A Figura 3.1 ilustra o modelo de arquitetura cliente/servidor duas camadas.

Aplicações com as seguintes características são adequadas para desenvolver em arquitetura cliente/servidor de duas camadas:

- Ambiente relativamente homogêneo (GALLAUGHER, 1996);
- Lógica de negócio/lógica da aplicação relativamente estática (GALLAUGHER, 1996);
- Suporte a uma quantidade limitada de usuários (não mais que algumas centenas);
- Aplicação funcional na mesma rede local onde está o servidor de banco de dados;
- Necessidade de um nível normal de segurança;

- Baixo volume de trafego na rede.

Principais vantagens deste modelo:

- Produtividade: muitas ferramentas avançadas possuem recursos para acelerar o desenvolvimento da aplicação trabalhando com o modelo de duas camadas (Visual Basic, Power Builder, Delphi) e utilizando-se de técnicas de desenvolvimento rápido de aplicações (RAD);
- Melhor reuso: quando a lógica de negócio é implementada somente no servidor é possível iniciá-la de muitas aplicações clientes.

Principais desvantagens deste modelo:

- Nova versão da aplicação a cada alteração de lógica de negócio, no caso de um cliente robusto (GALLAUGHER, 1996);
- Impossibilidade de implantar a lógica de negócio em um servidor de aplicação, separando-o da camada de apresentação e da camada de dados;
- Falta de uma segurança robusta;
- Falta de escalabilidade.

De acordo com GALLAUGHER (1996), o tipo de aplicação mais comum implementada em duas camadas é o cliente robusto (ou seja, a implementação da lógica de negócio é no cliente), dividido em duas entidades de software: aplicação cliente e o servidor de banco de dados, conforme a Figura 3.2.

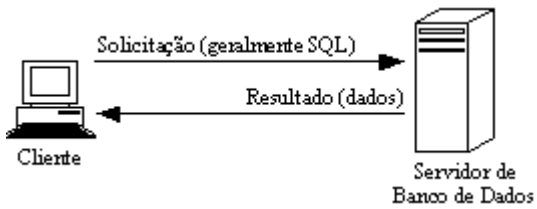

Figura 3.2 – Topologia de acesso de dados para arquitetura duas camadas.

3.2.2. Modelo Cliente/Servidor Três Camadas

Para suprir as limitações da arquitetura do modelo de duas camadas foi criada uma camada intermediária entre a camada de apresentação e a camada de dados para implementar a lógica de negócio da aplicação. Esta nova camada é também chamada de servidor de aplicação e comunica-se tanto com a camada de apresentação quanto com a camada de dados.

Se o número de usuários for pequeno, o servidor de aplicação pode ser instalado no computador cliente ou junto ao servidor de banco de dados, reduzindo a quantidade de conexões e tornando o sistema mais rápido. Se a quantidade de usuários for grande, o servidor de aplicação pode ser instalado num computador separado, ou ainda em vários computadores.

Mesmo com uma camada intermediária para a lógica de negócio, não é necessário que toda a lógica seja implementada no servidor de aplicação. Por exemplo, a integridade de dados é mais eficiente quando implementada no banco de dados.

Abaixo, os benefícios do modelo de três camadas (CAMBIOTIS, 2001):

- **Escalabilidade:** Neste modelo os servidores de aplicação podem ser implantados em vários computadores. O banco de dados necessita manter-se conectado apenas com alguns servidores de aplicação, ao invés de manter conexões com todos os clientes;
- **Integridade de dados:** Pelo fato de todas as atualizações no banco de dados serem efetuadas através de uma camada intermediária, esta camada pode garantir que somente dados válidos sejam atualizados, removendo o risco de corrupção de dados por aplicações clientes não autorizadas;
- **Segurança:** A segurança pode ser implementada em múltiplos níveis, fazendo com que seja mais difícil para um cliente acessar dados não permitidos. A lógica de negócio pode ser implantada num computador mais seguro.
- **Redução da distribuição de versões:** As alterações da lógica de negócio podem ser centralizadas em um só local.

- **Ocultação da estrutura do banco de dados:** A estrutura do banco de dados não é visível à aplicação cliente; logo, as mudanças no banco de dados são transparentes para ela.

Segundo Herb Edelstien e George Schussel, uma limitação do modelo de três camadas é uma maior dificuldade no uso de um ambiente de desenvolvimento comparando-se ao desenvolvimento orientado a visual das aplicações de duas camadas (SADOSKI, 1997).

3.3. Considerações

A arquitetura cliente/servidor ganhou popularidade nas organizações nos últimos anos por aumentar a capacidade das redes compostas por muitas estações com bom poder de processamento – que tiveram seus custos reduzidos – e um limitado número de servidores (LEWANDOWSKI, 1998). Com esta popularidade, houve uma grande oportunidade para o aumento do conhecimento coletivo e o amadurecimento das ferramentas que apóiam o desenvolvimento de aplicações na arquitetura cliente/servidor, resultando numa melhoria de qualidade.

Apesar de algumas desvantagens, conforme exposto neste capítulo, a arquitetura cliente/servidor é extremamente adequada para muitas aplicações (LEWANDOWSKI, 1998).

4. Desenvolvimento Rápido de Aplicações – RAD

4.1. O que é RAD?

O termo Desenvolvimento Rápido de Aplicações (*Rapid Application Development* - RAD) se aplica a projetos que têm prazos curtos, e que geralmente envolvem o uso de prototipagem e ferramentas de desenvolvimento de alto nível.

James Martin, na primeira vez que usou o termo, escreveu: “Desenvolvimento Rápido de Aplicações (RAD) é um ciclo de desenvolvimento projetado para proporcionar um desenvolvimento mais rápido e resultados de mais alta qualidade que os alcançados no ciclo de desenvolvimento tradicional. Ele foi projetado para obter o máximo proveito dos softwares de desenvolvimento poderosos que evoluíram recentemente” (CASEMAKER, 2000).

KETTEMBOROGH (1997), define o RAD como “uma abordagem para construir sistemas de computador que combina ferramentas e técnicas CASE (*Computer-Assisted Software Engineering*), protótipos dirigidos ao usuário e prazos curtos numa fórmula potente, testada e confiável para alta qualidade e produtividade. O RAD aumenta drasticamente a qualidade do sistema final enquanto reduz o tempo necessário para construí-los”.

De acordo com a Online Knowledge, RAD é “uma metodologia que permite as organizações desenvolverem mais rapidamente sistemas estrategicamente importantes reduzindo os custos de desenvolvimento e mantendo a qualidade. Isto é alcançado usando-se uma série de técnicas de desenvolvimento de aplicações comprovadas, juntamente com uma metodologia bem definida” (CASEMAKER, 2000).

4.2. História do RAD

Os ciclos tradicionais de desenvolvimento que surgiram nos anos 70, e ainda largamente utilizados atualmente são baseados numa abordagem estruturada passo a passo para o desenvolvimento de sistemas. Esta rígida seqüência de passos força um

usuário a aprovar depois de cada passo estar completo e antes de iniciar o próximo passo do desenvolvimento. Estes requisitos e projeto são então congelados e o sistema é programado, testado e implantado. Com métodos tão convencionais, existe um grande tempo antes que o usuário veja algum resultado e o processo de desenvolvimento poderia demorar tanto que as regras de negócios do cliente poderiam ter mudado antes mesmo do sistema estar pronto (CASEMAKER, 2000).

Em resposta a este modelo seqüencial rígido de desenvolvimento, Barry Boehm introduziu um modelo denominado Espiral. O Modelo Espiral é uma abordagem dirigida aos riscos, em oposição à dirigida à programação, que utiliza modelagem de processos ao invés de fases metodológicas. Através deste modelo, implementou-se a prototipagem de software como uma maneira de reduzir os riscos. O processo de desenvolvimento do Modelo Espiral separa o produto em partes críticas ou níveis enquanto faz-se análise de riscos, prototipagem e os mesmos passos em cada um destes níveis. Similarmente, o Ciclo de Vida Evolucionário de Tom Gilb é baseado numa prototipagem evolucionária em que o protótipo vai crescendo e sendo refinado em um produto final (CASEMAKER, 2000).

A origem do RAD está na metodologia chamada *Rapid Iterative Production Prototyping* (RIPP) desenvolvida e utilizada na empresa DuPont do meio ao final dos anos 80. James Martin, então estendeu o trabalho num processo mais formalizado, que se tornou conhecido como Desenvolvimento Rápido de Aplicações (RAD). O RAD comprimiu o desenvolvimento passo a passo dos métodos convencionais em um processo repetitivo, conforme mostra a Figura 4.1, onde JAD refere-se a *workshops* com um grupo de usuários e os desenvolvedores objetivando elaborar a análise de requisitos das aplicações de forma rápida. A abordagem RAD inclui o desenvolvimento e refinamento dos modelos de dados, modelos de processos e protótipos num processo repetitivo (CASEMAKER, 2000).

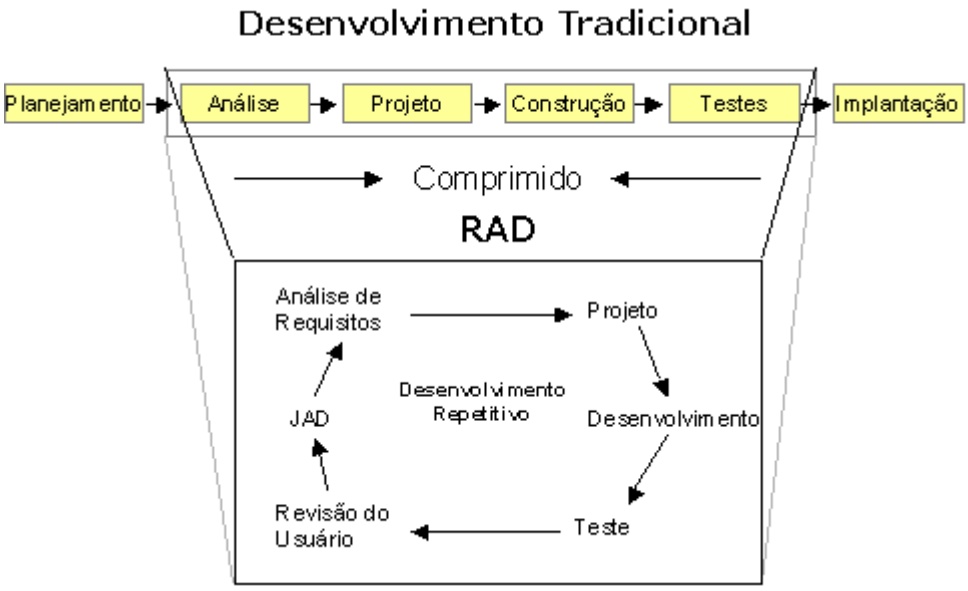

Figura 4.1 – Comparação entre desenvolvimento tradicional e RAD

Fonte: (CASEMAKER, 2000)

4.3. Quando RAD deveria ser utilizado?

Segundo DAVIES (1998), o RAD não é adequado para todos os tipos de projeto. Diversos fatores são úteis para ajudar a decidir se é adequado ou não a utilização de RAD em determinado projeto:

- **Tipo de sistema:** RAD é adequado para sistemas onde:
    a. A aplicação é interativa;
    b. As funcionalidades são claramente visíveis na interface de usuário;
    c. O grupo de usuários está claramente definido;
    d. As funcionalidades do sistema não são computacionalmente complexas.
- **Envolvimento da gerência:** necessita do apoio da gerência para que haja comprometimento dos usuários finais;
- **Envolvimento do usuário final:** os desenvolvedores devem ter fácil acesso aos usuários finais;

- **A equipe de desenvolvimento:** é ideal que a equipe de desenvolvimento seja a mesma do início ao fim do projeto;
- **Gerenciamento do Projeto:** é necessário o gerenciamento do projeto para que o mesmo se mantenha focado no seu objetivo;
- **Outorga de poder:** A equipe de desenvolvimento necessita ter autoridade para fazer decisões de projeto básicas do dia-a-dia sem precisar consultar seus superiores;
- **Tamanho do Projeto:** RAD é mais adequado para pequenos a médios projetos. Para grandes projetos deveria ser considerado somente se há possibilidade de dividi-lo em componentes menores, cada um dos quais implantado independentemente;
- **Tamanho da equipe:** A equipe de desenvolvimento deve ser pequena a fim de reduzir problemas de comunicação e gerenciamento, e também aumentar o comprometimento com o desenvolvimento.

4.4. Quando RAD não deveria ser utilizado?

Ainda segundo DAVIES (1998), RAD não deveria ser utilizado:

- Para sistemas em tempo-real ou críticos de segurança;
- Para sistemas de grande infra-estrutura;
- Para sistemas computacionalmente complexos;
- Para aplicações que precisam ter suas especificações completas antes de iniciar o desenvolvimento;
- Quando existe falta de comprometimento dos usuários ou eles não estão disponíveis;
- Quando não há um patrocinador do projeto que garanta seu andamento;
- Quando muitas tecnologias novas estão sendo introduzidas no escopo do projeto.

4.5. Ferramentas RAD

As ferramentas RAD diminuem as diferenças entre a prototipagem e a criação das aplicações por fazerem com que a criação das aplicações seja mais fácil. No entanto, a construção de uma aplicação ainda é mais difícil que a construção de um protótipo e tratando-se de desenvolvimento rápido de aplicações, a prototipagem não deveria ser dispensada, mesmo com as facilidades existentes hoje nas ferramentas RAD (BLOM, 2000). Um problema mencionado por BLOM (2000) é que várias ferramentas RAD dificultam uma boa separação entre a interface do usuário e a lógica de negócio da aplicação.

4.6. Considerações

As ferramentas que apóiam o Desenvolvimento Rápido de Aplicações evoluíram consideravelmente nos últimos anos, principalmente nos recursos de interface gráfica de usuário e transações com banco de dados. Existe a tendência de direcionar os desenvolvedores a implementar a lógica de negócio juntamente com a interface de usuário, tornando a aplicação um cliente robusto (onde a lógica de negócio reside no cliente). Esta tendência é um fator determinante para a agilidade no desenvolvimento de aplicações nestas ferramentas.

Observa-se que ambientes de desenvolvimento visual amplamente difundidos comercialmente, tais como Delphi, Visual Basic e Power Builder, possuem um grande enfoque no desenvolvimento de aplicações na arquitetura cliente/servidor, tratando-se de aplicações que se comunicam com banco de dados.

5. Exemplos de Frameworks Existentes

5.1. Visão Geral

Este capítulo tem por objetivo citar alguns exemplos de frameworks. Vários frameworks já foram desenvolvidos; alguns disponíveis comercialmente e outros para fins acadêmicos. Alguns já contribuem substancialmente ao desenvolvimento de software no mercado, como: MacApp, Interviews, ACE, MFC e DCOM da Microsoft, RMI da JavaSoft e implementações do CORBA da OMG.

Foram escolhidos dois frameworks para aplicações empresarias para um maior detalhamento neste trabalho, sendo eles: IBM San Francisco e SAP Business Framework.

5.2. IBM San Francisco

Conforme sumário técnico (IBM, 1997a), o framework San Francisco, da IBM, oferece soluções para desenvolvimento de aplicações de negócio através de componentes de processos de negócio, que fornecem uma infra-estrutura orientada a objetos, um modelo de programação de aplicações consistente e algumas regras de negócio que podem ser usadas para começar a construir uma aplicação. Esses componentes facilitam a migração para tecnologia orientada a objetos, disponibilizando serviços e funções bem testadas.

O framework San Francisco é implementado usando a linguagem Java. Isso faz com que as aplicações por ele desenvolvidas sejam multiplataforma. Também permite que os desenvolvedores utilizem as várias ferramentas e bibliotecas de classes disponíveis no mercado para Java. Ele está sendo desenvolvido em parceria com vários fabricantes de software para projetar, desenvolver e validar componentes, criar ferramentas de desenvolvimento e desenvolver pequenas e médias soluções de negócios (IBM, 1997b).

O framework San Francisco possui três camadas de código reutilizável, conforme Figura 5.1:

- *Base*: provê a infraestrutura e serviços que são necessários para construir aplicações distribuídas e multiplataforma;
- *Common Business Objects*: provê definições de objetos de negócios que podem ser usados em aplicações de diferentes domínios e também como fundação para interoperabilidade entre aplicações;
- *Core Business Processes*: provê objetos de negócios e regras de negócio padronizadas para determinados domínios. Inicialmente, o framework está disponível com componentes de negócio nos domínios de contas a receber, contas a pagar, contabilidade, gerenciamento de pedidos (compra e venda) e gerenciamento de estoque. Com o tempo, estes componentes serão estendidos e melhorados para processos adicionais de negócio, objetos e acesso a mais interfaces de framework, proporcionando grande flexibilidade.

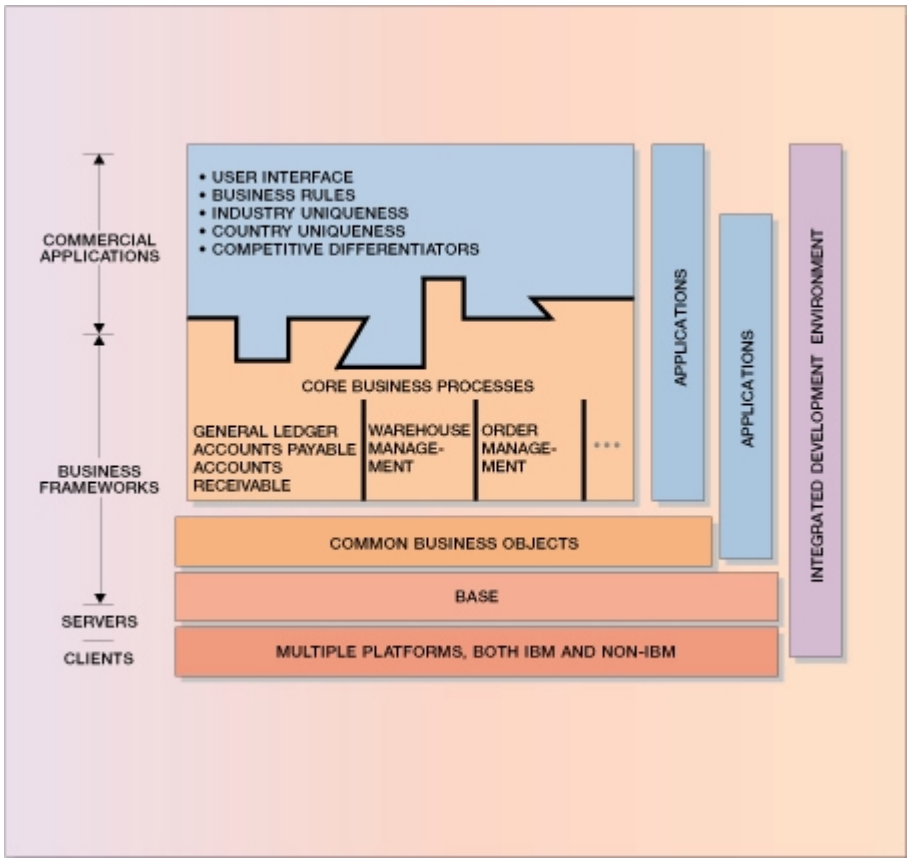

Figura 5.1 – Visão da arquitetura do IBM San Francisco

Fonte: (IBM, 1997b)

A seguir as três camadas do San Francisco Framework serão descritas de forma mais detalhada (IBM, 1997a) (IBM, 1997b).

5.2.1. Camada Base

A camada *Base* isola uma aplicação das complexidades das tecnologias de rede multiplataforma e permite que os desenvolvedores concentrem-se nos elementos que tragam benefícios aos usuários. É o mais baixo nível do framework que pode ser usado pelos desenvolvedores, e provê a infraestrutura fundamental para as duas camadas acima dela: *Common Business Objects* e *Core Business Processes.*

Duas categorias de funções estão incluídas nesta camada e são utilizadas diretamente pelos desenvolvedores: modelo de classes dos objetos básicos e utilitários. Para dar suporte a distribuição e requerimentos de missões críticas, a camada também possui serviços de *kernel*, acessados através das interfaces do modelo de objetos básicos para garantir compatibilidade entre versões.

O modelo de classes dos objetos fornece a estrutura base para os objetos do framework San Francisco, e definem o modelo da aplicação. As classes contêm métodos concretos e abstratos que podem ser sobrepostos ou implementados pelos desenvolvedores das aplicações.

O modelo de classes dos objetos inclui:

- Entidade: são objetos independentes e compartilhados (pessoas, coisas) usados nas operações de negócio. Entidades estão geralmente associadas com dados e podem ser armazenadas em um banco de dados através de uma associação a um mecanismo de persistência.
- Dependente: contém informações adicionais de uma entidade. Por serem dependentes não podem existir fora do escopo de uma entidade, nem serem compartilhadas ou referenciadas diretamente.
- Comando: conjunto de operações executadas sobre um objeto ou coleção de objetos, podendo conter funções e procedimentos da aplicação e/ou do framework.

- Coleção/*Iteractor*: conjunto de objetos estruturados de maneira que possam ser acessados por uma chave. *Iteractors* são usados para acessar os elementos da coleção.
- *Factory*: gerencia as instâncias dos objetos (entidades, comandos e coleções) em tempo de execução.

Os utilitários fornecem serviços necessários na maioria das aplicações construídas sob o framework San Francisco, sendo disponibilizadas para utilização e não para estender ou modificar.

5.2.2. Camada Common Business Objects

A camada intermediária do framework contém objetos de negócio genéricos, sendo composta por vários frameworks que podem ser categorizados em: objetos de negócio comuns a vários domínios; serviços comuns de aplicações.

Um “parceiro de negócio” é um exemplo de um objeto de negócio utilizado em vários domínios, e encapsula características comuns a um cliente ou a um fornecedor.

Os serviços comuns de aplicações auxiliam a automatização de processos de negócio.

5.2.3. Camada Core Business Processes

No nível mais alto do framework, os componentes de negócios podem ser especializados para necessidades específicas de negócio, sendo facilmente estendidos para o desenvolvimento de uma solução completa. Nesta camada, as classes implementam a estrutura e comportamentos básicos a um determinado domínio.

5.2.4. Desenvolvendo Aplicações Sob o Framework San Francisco

A maneira mais simples de desenvolver uma aplicação sobre o San Francisco é utilizando seus objetos da maneira que são disponibilizados. Para isso o desenvolvedor cria uma aplicação cliente que usa a *factory* para manipular os objetos de negócio do framework.

A segunda abordagem seria estender o framework para criar um novo domínio de aplicações a partir do modelo de classes dos objetos. Neste casso os desenvolvedores precisam ter um conhecimento maior do framework.

A terceira abordagem seria estender o framework para atender necessidades específicas de um domínio já atendido pelo framework. Pode ser necessário adicionar atributos em algumas classes ou modificar a lógica de negócio de alguns métodos.

5.3. SAP Business Framework

O sistema SAP R/3[2] é o mais bem sucedido produto ERP (*Enterprise Resource Planning*) disponível no mercado e se destina a uma grande quantidade de domínios de negócio. Preocupada com as necessidades do mercado e com a concorrência, a SAP tem trabalhado continuamente para alcançar interoperabilidade, flexibilidade e abertura. Assim, a SAP projetou um novo framework arquitetural orientado a objetos transformando um sistema relativamente monolítico numa plataforma de integração para o uso de objetos de negócio e aplicações. Esta arquitetura de componentes e objetos de negócios é chamada de Business Framework (ALEKSY & KORTHAUS, 1999).

O Business Framework, da SAP, objetiva a "componenterização" de funcionalidades R/3 em uma família integrada de componentes de negócio que podem ser misturados e combinados – e estendidos com componentes de terceiros compatíveis – para trazer, rapidamente, novas funcionalidades aos clientes. O Business Framework fornece benefícios de integração de sistemas de negócio com processos de negócio extensíveis baseados no SAP Business Objects – que é parte do framework – e são desenvolvidos para utilizar interfaces padronizadas existentes e futuras suportando interoperabilidade entre os componentes. Com este framework, os clientes da SAP podem manter vários componentes R/3 de diferentes versões, ao invés de atualizar todo o sistema R/3 de uma só vez (SAP, 1997).

[2] R/3 significa arquitetura real-time em três camadas, sendo: apresentação, aplicação e banco de dados.

Conforme (ALEKSY & KORTHAUS, 1999), o SAP Business Framework é composto de:

- Componentes de Negócios SAP (*SAP Business Components*): representam uma aplicação para um domínio empresarial. Cada componente é composto por vários Objetos de Negócios SAP que juntos formam a interface do componente. Cada componente pode ser mantido de forma independente.
- Objetos de Negócios SAP (*SAP Business Objects*) e Interfaces de Programação de Aplicações de Negócio (BAPIs): Os objetos de negócio encapsulam dados e funcionalidades que representam conceitualmente os negócios do mundo real. Cada objeto de negócio pode ser acessado através de sua interface, chamada de *Business Application Programming Interfaces* (BAPIs). As BAPIs determinam toda a aparência externa do objeto de negócio.
- Tecnologias de comunicação e integração: para possibilitar acessar as BAPIs de forma genérica, alguns serviços de comunicação – como o DCOM da Microsoft, o RFC da SAP, etc. – foram implementados no framework. A integração dos objetos de negócios distribuídos em processos de várias aplicações de diferentes fabricantes pode ser alcançada pelo serviço de integração chamado *Application Link Enabling* (ALE).

5.3.1. Objetos de Negócios SAP

Os objetos de negócios da SAP são abstrações dos domínios de negócio, como empregado, produto ou cliente. Além de representar as interfaces de programação para o sistema R/3, os objetos de negócio fornecem também uma visão orientada a negócios de alto nível adequada para modelagem e engenharia de negócios.

Um objeto de negócio é caracterizado pela sua identificação, atributos, métodos e eventos. A identificação de cada objeto é alcançada pelo *SAP Business Object Identifiers*, composto do *SAP System Identifier*, o nome, e tipo do objeto e uma chave. Os objetos de negócios SAP são estruturados nas seguintes camadas, conforme a Figura 5.2 (SAP, 1997):

- *Kernel* do objeto de negócio: contém os dados do objeto e o núcleo da lógica de negócio;
- Camada de integridade: contém regras de negócio e restrições necessárias para manipulação dos dados de forma consistente;
- Camada de interface: define os serviços fornecidos pelos objetos, os eventos de controle de entrada e os eventos de saída publicados pelo objeto;
- Camada de acesso: descreve as metodologias (COM, DCOM, Java, CORBA) que podem ser usadas para acessar o objeto.

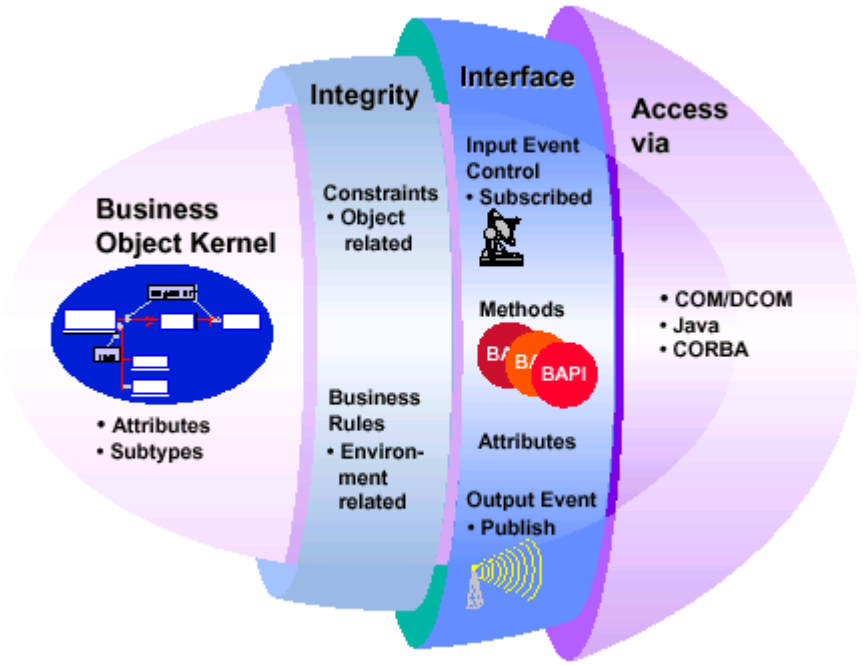

Figura 5.2 – As camadas dos Objetos de Negócio SAP

Fonte: (SAP, 1997)

O acesso aos dados do objeto de negócio é efetuado através de métodos públicos. Qualquer aplicação que utilize os serviços do objeto de negócio precisam especificar o nome do método e os parâmetros, mas não precisam conhecer sua implementação.

5.3.2. Interfaces de Programação de Aplicações de Negócio (BAPIs)

As interfaces de programação de aplicações de negócio fornecem as informações necessárias para acessar os Objetos de Negócio SAP. As interfaces são estáveis, fazendo com que as alterações na implementação do objeto não afetem outras partes do sistema.

Os objetos de negócio e suas BAPIs são armazenados no Repositório de Objetos de Negócios (*Business Objects Repository* – BOR). A implementação dos métodos dos objetos de negócio são baseadas em *SAP Function Modules*, que não é orientado a objetos. Porém, a camada para visualização e acesso ao sistema R/3 na forma de objetos de negócio é orientado a objetos.

Existem duas maneiras de acessar os objetos de negócio: orientado a objetos chamando uma BAPI via repositório (BOR); ou efetuando uma chamada convencional via RFC (*Remote Funcion Call* – protocolo da SAP) num nível mais baixo que implementa as BAPIs.

5.3.3. Repositório de Objetos de Negócio (BOR)

Como parte do repositório do R/3, o Repositório de Objetos de Negócios trabalha como um sistema de armazenamento de informações orientadas a objetos que contém todos os Objetos de Negócios SAP, seus métodos, apresentados como BAPIs, objetos técnicos (textos, anotações, etc) e metaobjetos. Como ponto central de acesso para aplicações externas, o BOR tem duas funções principais:

- Identificação e descrição dos tipos de objetos de negócio. As propriedades, métodos e eventos de um objeto podem ser consultado no tipo de objeto, documentado por um metaobjeto no repositório.
- O repositório é o ambiente de execução onde as instâncias dos objetos são criadas quando solicitados pelas aplicações clientes.

O BOR fornece também uma *Dynamic Invocation Interface* para permitir o acesso a interfaces de Objetos de Negócios SAP que não eram conhecidas em tempo de compilação.

5.4. Considerações

Os frameworks IBM San Francisco e o SAP Business Framework são produtos maduros e solidificados no mercado. O IBM San Francisco é comercializado através de parceiros, porém o SAP Business Framework acompanha o sistema SAP R/3 e não é comercializado separadamente. Os custos de aquisição de licença, customização e implantação de produtos como os citados requerem grandes investimentos.

Vários outros frameworks têm surgido no mercado e no meio acadêmico e podem ser uma boa opção em termos de custo, mas outros critérios também precisam ser avaliados.

Muitas empresas acabam por decidir pela construção de suas próprias soluções, tendo como benefício um produto mais direcionado, personalizado e com melhor aproveitamento das funcionalidades.

6. O Projeto do Blendwork

6.1. Visão Geral

O Blendwork é um framework orientado a objetos projetado para facilitar o desenvolvimento de aplicações cliente/servidor em ambientes orientado a objetos visuais que utilizam a metodologia de Desenvolvimento Rápido de Aplicações (RAD). Ele provê uma extensão às funcionalidades já disponibilizadas em ferramentas RAD, definindo funções comuns de infra-estrutura para a construção de aplicações comerciais.

Inicialmente implementado na linguagem Object Pascal no Delphi 5 (ambiente de desenvolvimento visual orientado a objetos da Borland), o Blendwork pode ser portado para outras linguagens de programação, desde que as singularidades de cada linguagem sejam observadas e convertidas de forma adequada ao novo ambiente de desenvolvimento. Pelo fato do Blendwork ser um framework orientado a objetos é pré-requisito, no caso de portá-lo para outra linguagem, que esta seja orientada a objetos; e recomenda-se a utilização de um ambiente visual de desenvolvimento.

6.2. Cenário

A Extersoft Informática foi contratada por uma Fundação (empresa sem fins lucrativos, não governamental) para a manutenção evolutiva de suas aplicações. Esta empresa é responsável pelo gerenciamento de vários benefícios providos aos funcionários de uma empresa pública estadual. Suas aplicações foram em parte desenvolvidas em Power Builder 5 (ferramenta de desenvolvimento visual da Sybase), e em Delphi 3 (ferramenta de desenvolvimento visual da Borland). A plataforma utilizada na Fundação é totalmente Windows, com uma rede local de aproximadamente 25 computadores gerenciada por um servidor Microsoft Windows 2000 e interligada à internet. Utiliza os bancos de dados Oracle e Microsoft SQL Server.

O principal objetivo desta Fundação é disponibilizar e gerenciar benefícios aos funcionários (denominados de associados) da empresa governamental e seus dependentes. As aplicações devem gerenciar estes benefícios integrando os seguintes

departamentos da empresa: Administrativo, Saúde, Seguridade e Financeiro. Cada um dos departamentos possui sua própria aplicação com acesso ao mesmo banco de dados, compartilhando assim, informações comuns como o cadastro de associados e seus dependentes. Outra necessidade de integração das aplicações que compõem o sistema de gerenciamento de benefícios da Fundação é a que está relacionada com as operações financeiras, realizadas somente pelo departamento financeiro. Todas as operações financeiras de créditos e débitos aos associados são encaminhadas a este departamento para sua posterior execução. Vale também citar a aplicação gerencial, disponibilizado à diretoria, para fazer o acompanhamento dos benefícios e facilitar o atendimento e avaliações financeiras dos associados.

Foram constatados alguns problemas relacionados ao antigo sistema utilizado pela Fundação, dentre eles:

- falta de flexibilidade, escalabilidade, dificuldade de integração ao ambiente Windows;
- dificuldade no reuso de funcionalidades comuns;
- deficiência em alguns módulos deste sistema no quesito de atendimento às necessidades de negócio.

Devido a esses problemas, decidiu-se, baseado neste cenário, construir um framework orientado a objetos para fornecer suporte à infra-estrutura básica à implementação de um novo sistema para o gerenciamento dos benefícios. A este novo sistema foi dado o nome de Blendus e o framework utilizado no seu desenvolvimento foi chamado de Blendwork. Logo, o Blendus é um sistema de gerenciamento de benefícios para fundações e foi desenvolvido utilizando o framework Blendwork.

Ambos, o Blendus e o Blendwork, foram construídos observando-se as funcionalidades do sistema previamente utilizado pela Fundação e aproveitando-se das experiências no desenvolvimento de aplicações da Extersoft anteriores a este projeto, trabalhos que por sua vez não serão citados por estarem fora do escopo deste trabalho.

6.3. Ferramentas Utilizadas

O desenvolvimento do Blendwork deu-se utilizando as seguintes ferramentas computacionais:

- **Ferramenta de modelagem UML:** o ModalMaker, ferramenta produzida pela ModelMaker Tools, foi utilizado para criar e documentar o conjunto de classes que fazem parte do framework, bem como seus relacionamentos e colaborações. A integração com os códigos fonte e o ambiente de desenvolvimento do Delphi foram primordiais à escolha do ModelMaker no processo de implementação do Blendwork;
- **Ambiente visual de desenvolvimento orientado a objetos:** o Borland Delphi 5 foi usado para a implementação do framework. Sua escolha se deve ao fato de ele já ser previamente utilizado pela Extersoft no desenvolvimento de aplicações. Algumas outras ferramentas e componentes foram instalados no ambiente de desenvolvimento do Delphi para auxiliar na implementação, a saber:
  - **Report Builder:** ferramenta de relatórios da Digital Metaphors, caracterizada por um conjunto de componentes e ferramenta visual que se integram ao IDE (*Integrated Development Environment*) do Delphi;
  - **ExDBGrid:** Componente que exibe informações do banco de dados em forma de planilha ou grade, produzida pela GJL Software;
  - **RxLibrary:** Biblioteca de componentes para o Delphi produzido por Fedor Kozhevnikov, Igor Pavluk e Serge Korolev.
- **Ferramenta de modelagem de dados relacional:** o ErWin foi usado para a elaboração do modelo de entidade e relacionamentos que garante a persistência dos dados utilizados nas classes concretas do framework;
- **Banco de dados:** o Blendwork foi desenvolvido utilizando-se o banco de dados Oracle para a persistência do dicionário de dados e das informações

inerentes ao negócio gerenciados pelo Blendus (sistema composto de 6 aplicações desenvolvido sobre o Blendwork). Isso não implica na incompatibilidade do Blendwork com outros bancos de dados, visto que qualquer gerenciador de banco de dados relacional poderá ser utilizado desde que seja suportado pelo BDE (*Borland Database Engine*) e SQL Links; e também devendo ser compatível com SQL Ansi.

6.4. Metodologia de Desenvolvimento

Dentro do cenário anteriormente mencionado, dispunha-se de cinco exemplos de aplicações implementadas no domínio de aplicações ao qual o Blendwork é destinado. Este conjunto de aplicações compõe um sistema, integrado através das informações persistidas em um banco de dados, que objetiva automatizar o gerenciamento dos benefícios providos pela Fundação, que em alguns casos mantém convênios com fornecedores para a prestação de serviços especializados aos seus associados. Estes convênios e os pagamentos aos fornecedores provenientes destes convênios, também são gerenciados neste sistema. Ele é chamado Sistema Integrado de Fundações – SIF e, portanto, neste trabalho será tratado como SIF.

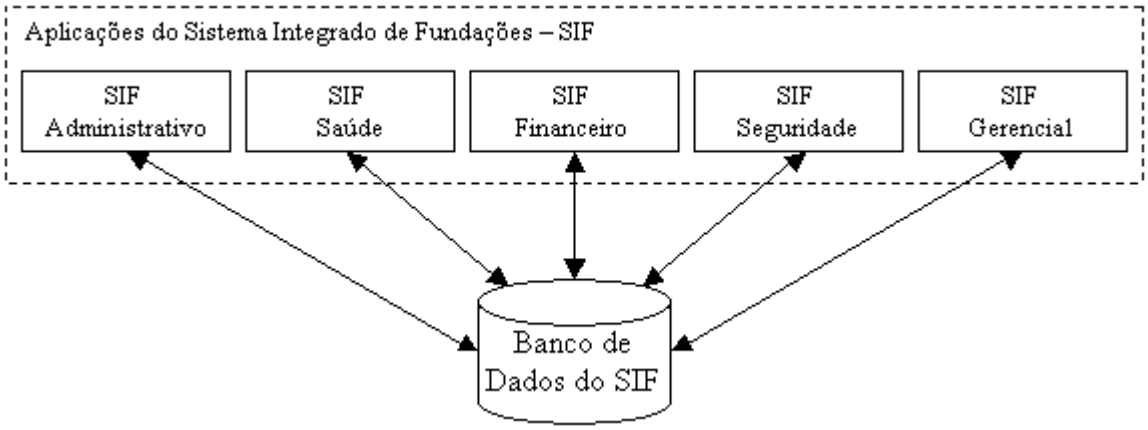

Figura 6.1 – Arquitetura do Sistema Integrado de Fundações

Conforme mostra a Figura 6.1, o SIF é dividido em cinco aplicações, cada uma das quais compostas por módulos destinados a automatizar processos específicos de negócio dentro da Fundação, a saber:

- **SIF Administrativo:** Programa de Alimentação do Trabalhador - PAT, Auxílios, Locação do Ginásio;
- **SIF Saúde:** Planos de Saúde, Planos Odontológicos, Convênio Farmácia, Reembolso Farmácia;
- **SIF Financeiro:** Empréstimos, Contas a Pagar, Contas a Receber, Controle de Saldo Vencido;
- **SIF Seguridade:** Plano de Incentivo à Aposentadoria, Plano de Auxilio Desemprego, Plano de Pensão e Pecúlio;
- **SIF Gerencial:** acompanhamento individual dos benefícios usufruídos pelos associados.

Com esta facilidade, de ter cinco aplicações implementadas disponíveis para análise, optou-se, para a concepção da primeira versão, pelo desenvolvimento de um framework orientado a objetos utilizando a metodologia de Projeto Dirigido por Exemplo (JOHNSON, 1993). Cada uma das aplicações do SIF foi analisada e comparada abstraindo delas as funcionalidades comuns e implementando-as no Blendwork. Além de analisar as aplicações existentes, também foram assimiladas abstrações já conhecidas, resultado da experiência de desenvolvimento do autor em projetos anteriores no mesmo domínio de aplicações. Para a evolução do framework, após sua primeira versão estar completa, utilizou-se a metodologia de Projeto Dirigido por *Hot Spot*, onde foram identificadas as partes flexíveis na estrutura de classes do framework.

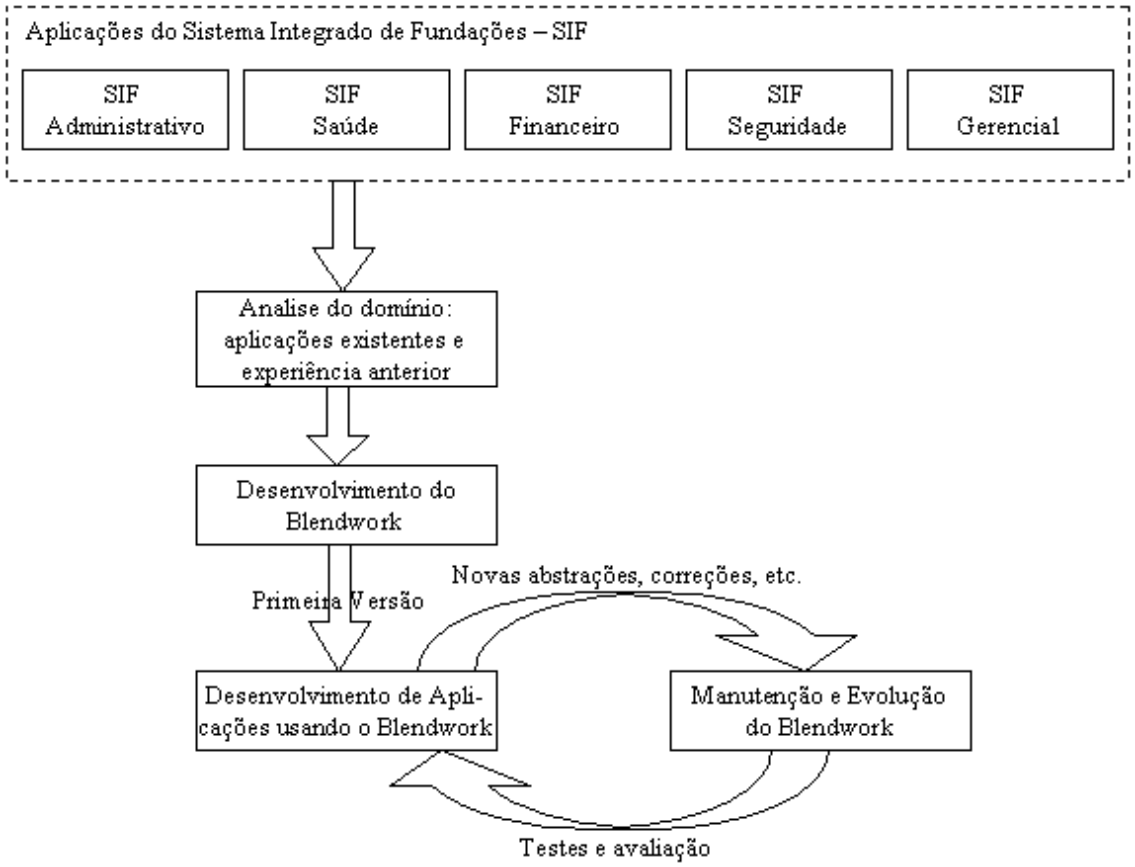

Figura 6.2 – Desenvolvimento e Evolução do Blendwork

A Figura 6.2 apresenta as etapas processo de desenvolvimento e evolução do Blendwork. Veja abaixo a descrição de cada etapa:

- **Analise do Domínio:** A partir da análise das aplicações do SIF foram abstraídas as funcionalidades comuns das aplicações, levando em consideração também a experiência do autor em aplicações anteriores;
- **Desenvolvimento do Blendwork:** Projeto da estrutura de classes e implementação resultando na primeira versão do Blendwork;
- **Desenvolvimento de Aplicações usando o Blendwork:** Quando o Blendwork já continha algumas das funcionalidades implementadas iniciou-se o desenvolvimento de uma aplicação piloto para testá-lo e avaliá-lo. A primeira aplicação desenvolvida foi o Blendus Administrativo. Ele foi eleito como "piloto" devido a algumas mudanças nas regras de negócios ocorridas nos

processos por ele gerenciados. Como estas mudanças eram significativas, optou-se pelo seu desenvolvimento em nova tecnologia ao invés de despender-se um grande esforço para a manutenção do SIF Administrativo;

- **Manutenção e Evolução do Blendwork:** Com os testes e avaliação do framework através do desenvolvimento de aplicações foi possível refiná-lo, corrigindo-se erros e incorporando-se novas abstrações que não haviam sido identificadas durante o processo de desenvolvimento. Após sua evolução resultante do desenvolvimento da primeira aplicação, deu-se continuidade ao processo de desenvolvimento das outras aplicações que compõem o Blendus, através das quais foi possível evoluir e validar o Blendwork de forma repetitiva.

Estando finalizado o desenvolvimento do Blendus Administrativo, partiu-se para o desenvolvimento das outras aplicações que compõem o Blendus: Gestor, Seguridade, Saúde, Financeiro e Gerencial. Cabe salientar que o Blendus Gestor não existia anteriormente e foi desenvolvido com a finalidade de gerenciar as informações comuns do Blendus e o dicionário de dados (informações sobre o próprio sistema) do Blendwork.

6.5. Análise do Domínio

Uma importante decisão de projeto do Blendwork surgiu durante o processo de análise do domínio de aplicações: construí-lo para dar suporte ao desenvolvimento de aplicações em arquitetura cliente/servidor duas camadas. Como as aplicações do SIF baseiam-se nesta arquitetura, constatou-se que seria inviável a construção de um framework em outra arquitetura, já que as abstrações resultantes da análise das aplicações existentes poderiam ser diferentes das abstrações comuns em outra arquitetura. Uma implicação desta decisão é a recomendação de utilizar o Blendwork para a construção de aplicações de pequeno ou médio porte, e que a quantidade de usuários finais das aplicações seja cuidadosamente avaliada. Uma vantagem é o suporte que as ferramentas de desenvolvimento disponíveis no mercado possuem para o desenvolvimento nesta arquitetura de aplicações.

Como resultado da análise do domínio de aplicações identificaram-se as abstrações comuns a todas as aplicações e que, conseqüentemente, devem fazer parte do Blendwork. Estas abstrações estão documentadas abaixo:

- **Módulo de Negócio:** Conjunto de classes concretas com o objetivo de atender a uma necessidade específica de negócio. Uma aplicação é composta de um ou mais módulos de negócio. No escopo deste trabalho, quando houver alguma referência a módulo entenda-se como módulo de negócio.
- **Interface Gráfica Principal da Aplicação:** Esta abstração representa a janela principal da aplicação e é exibida logo após a aplicação ser iniciada. Durante toda a execução da aplicação a Janela Principal é a interface ativa e pela qual o usuário invoca as tarefas implementadas na aplicação, que são geralmente exibidas dentro dos limites desta interface principal.
- **Formulários:** São as janelas da aplicação utilizadas para interagir com os usuários finais (interfaces gráficas de usuário) e implementar as funcionalidades da aplicação. Os formulários foram classificados de acordo com seu objetivo preliminar:
    - **Cadastros:** utilizados para manipular os dados armazenados no banco de dados, validando-os de acordo com a lógica de negócio. Também implementa facilidades para consulta dos dados através dos campos exibidos e através de funções específicas do banco de dados;
    - **Sub-formulários:** são recipientes (*containers*) que podem ser incluídos nos Formulários de Cadastro que necessitam de controle de acesso impostos de acordo com as permissões cedidas a cada usuário. Como exemplo podemos citar as informações salariais de um associado que porventura não possam estar visíveis a todos os usuários que possuem acesso ao cadastro de associados;
    - **Chamada de Relatórios:** são formulários usados para visualizar e imprimir os relatórios, com a possibilidade de filtrá-los de acordo com os parâmetros fornecidos nestes formulários;

  - **Execução de Processos:** são formulários usados para executar procedimentos específicos para a geração de dados, como por exemplo: geração de descontos odontológicos e geração de ordem de pagamento bancária. Estes processos podem estar implementados na própria aplicação ou no banco de dados;
  - **Localização de dados:** usado para localizar dados no banco de dados, como por exemplo: Localização de Associados, onde o usuário pode informar parte do nome de um associado e conseguir identificar sua matrícula funcional.

- **Tarefas:** Cada um dos formulários é representado de forma não visual por uma tarefa dentro da aplicação. Esta tarefa é persistida em um dicionário de dados do framework. Dentro do framework uma tarefa contém informações sobre o formulário. Além dos formulários, uma tarefa pode também representar um procedimento implementado no banco de dados;
- **Gerenciamento de Formulários:** Gerencia a criação dos formulários e a interação entre os formulários da aplicação através das Tarefas persistidas no dicionário de dados;
- **Relatórios:** Constatou-se a necessidade de criação de vários layouts para a emissão de relatórios de uma forma padronizada. Dentre os layouts já definidos no Blendwork, estão:
  - **Formal e Normal:** Relatórios genéricos que são emitidos pelas aplicações desenvolvidas com o Blendwork. Quando for no estilo Normal, o cabeçalho do relatório exibe apenas o nome da empresa; no estilo Formal, exibe o relatório com o logotipo da empresa no cabeçalho;
  - **Extrato:** Este modelo de relatório foi criado por causa da necessidade de se enviar certas informações detalhadas aos associados, como status de utilização de benefícios ou de débitos ou créditos. Um exemplo deste tipo de relatório é o Extrato Mensal de Descontos, enviado mensalmente aos associados informando os valores de descontos a

serem efetuados em sua folha de pagamento ou em sua conta corrente bancária;

- **Carta:** Modelo criado para emitir cartas com informações do banco de dados através de um processador de texto, mais especificamente através do Microsoft Word.

- **Componentes Visuais:** Os componentes visuais são os objetos de interface gráfica utilizados para exibir e/ou manipular os dados armazenados no banco de dados. Estão disponíveis nas ferramentas de desenvolvimento visual, mas foram inseridos como uma abstração por causa da necessidade de adaptação destes para uma melhor interação com as outras classes do framework;
- **Agrupamento de Componentes Visuais:** Alguns dos componentes visuais necessitam cooperar com outros componentes para a exibição e manipulação dos dados, ou ainda precisam ser combinados para a execução de alguma tarefa específica. Agrupando-se estes componentes em estruturas visuais é possível reutilizá-los de forma uniforme por toda a aplicação;
- **Domínio de Valores:** São listas de valores constantes ou pouco variáveis exibidas ao usuário de forma descritiva, mas armazenadas de forma codificada para economizar recursos no banco de dados e otimizar o processamento dos dados. Um exemplo pode ser a identificação do sexo de um associado, que é apresentado ao usuário como Masculino ou Feminino, mas é armazenado como 1 ou 2, respectivamente;
- **Parâmetros:** suporte a parâmetros do próprio framework e também a parâmetros necessários para as aplicações a serem desenvolvidas sobre ele;
- **Usuário:** usuário final que utiliza a aplicação. Esta abstração foi identificada pela necessidade de se autenticar o usuário antes de entrar na aplicação e de controlar seu acesso às tarefas disponíveis em cada uma das aplicações.

Através da generalização destas abstrações identificadas no processo de análise do domínio foram modeladas as classes que compõem a estrutura do Blendwork, tratada no próximo capítulo.

6.6. Considerações

Ao se analisar um conjunto de aplicações a um mesmo domínio, foi possível determinar a partir de exemplos concretos, as particularidades e as similaridades de cada uma delas. Apesar das aplicações do SIF terem a mesma arquitetura e compartilhar de muitas funcionalidades semelhantes, a reutilização de código não estava sendo aplicada de forma adequada. Além das questões sobre a infra-estrutura da aplicação, observou-se também a replicação de funcionalidades concretas ao invés do reuso. Um exemplo é o cadastro de associados, disponível em quatro das cinco aplicações do SIF analisadas. O inconveniente é que a implementação deste cadastro era fisicamente copiada entre as aplicações, sendo que quando houvesse uma alteração pertinente ao cadastro, todas as cópias do cadastro nas aplicações teriam que ser alteradas. A análise do domínio teve o intuito de eliminar esta repetição observada.

Durante o processo de análise, várias decisões de projeto foram influenciadas pela condição comercial do projeto. Havia uma certa expectativa de resultados a serem obtidos que justificassem o investimento na construção de um framework. Por este motivo o projeto foi direcionado a soluções práticas e recorrentes focando-se no domínio analisado, viabilizando o seu andamento e garantido os investimentos.

7. Implementação do Blendwork

7.1. Projeto da Estrutura de Classes

Conforme as abstrações identificas no capítulo anterior, iniciou-se o projeto da estrutura de classes que compõem o Blendwork. Para melhor visualização dos diagramas de classes, eles foram separados em quatro diagramas relacionados com funcionalidades específicas:

- **Diagrama da Estrutura Arquitetural do Framework;**
- **Diagrama dos Componentes Visuais;**
- **Diagrama dos Agrupamentos de Componentes;**
- **Diagrama do Dicionário de Dados.**

Como a abordagem de frameworks insere um conjunto de requisitos de modelagem não atendidos por metodologias OOAD em geral (SILVA, 2000), foram usadas as extensões propostas por SILVA (2000):

- **Redefinibilidade de Classe:** identifica se a classe pode ou não originar subclasses no desenvolvimento de uma aplicação sob o framework.
- **Essencialidade da Classe:** uma classe do framework pode ser essencial ou não, sendo que somente as classes redefiníveis podem ser classificadas como essenciais. Uma classe essencial (ou uma subclasse desta) deve ser utilizada em aplicações desenvolvidas a partir deste framework.
- **Classificação da Classe:** Além da redefinibilidade e essencialidade, as classes podem ser classificadas como abstratas, concretas ou externas;
- **Classificação dos Métodos:** Os métodos podem ser classificados como regular, *template* ou abstratos;

Nos diagramas apresentados neste trabalho, as classes redefiníveis serão representadas com a letra «R» acima do nome da classe; as redefiníveis e essenciais serão representadas com «R E»; as abstratas terão o nome em itálico e as concretas

serão representadas normalmente de acordo com as notações UML; já as externas serão representadas com a palavra «Externa».

Para melhor visualização dos diagramas, foram também introduzidas cores nas classes que fazem parte dos diagramas. As classes verdes fazem parte do Blendwork e as amarelas são da biblioteca de classes do Delphi. Na implementação, várias das classes do Blendwork foram especializadas a partir das classes do Delphi, conforme será ilustrado nos diagramas deste capítulo.

O Blendwork é um framework orientado a objetos caixa-cinza, e por isso, sua adaptabilidade está na especialização das classes através de herança e na composição de objetos. As principais propriedades e métodos no que diz respeito à adaptabilidade e colaboração entre as classes estão ilustradas nos diagramas. As classes externas, principalmente as que fazem parte da biblioteca de classes do Delphi, geralmente disponibilizam alguma adaptabilidade através de eventos que são disparados automaticamente através de mensagens ou métodos *template*.

7.2. Diagrama da Estrutura Arquitetural do Blendwork

A Figura 7.1 apresenta as principais classes, e seus relacionamentos, implementadas no Blendwork. Este conjunto de classes forma sua estrutura arquitetural. Todas as subclasses da classe externa *TForm* (classe da biblioteca de objetos do Delphi) contêm uma interface gráfica que em algum momento da execução da aplicação, caso sejam classes concretas, podem ser instanciadas e exibidas aos usuários. As classes *TfrmSplash* e *TfrmSenha* são instanciadas somente durante a inicialização da aplicação, sendo que a primeira exibe uma barra de progressão e uma figura e a segunda autentica o usuário com a mesma conta da rede. Atualmente a autenticação está implementada para contas do Microsoft Windows NT.

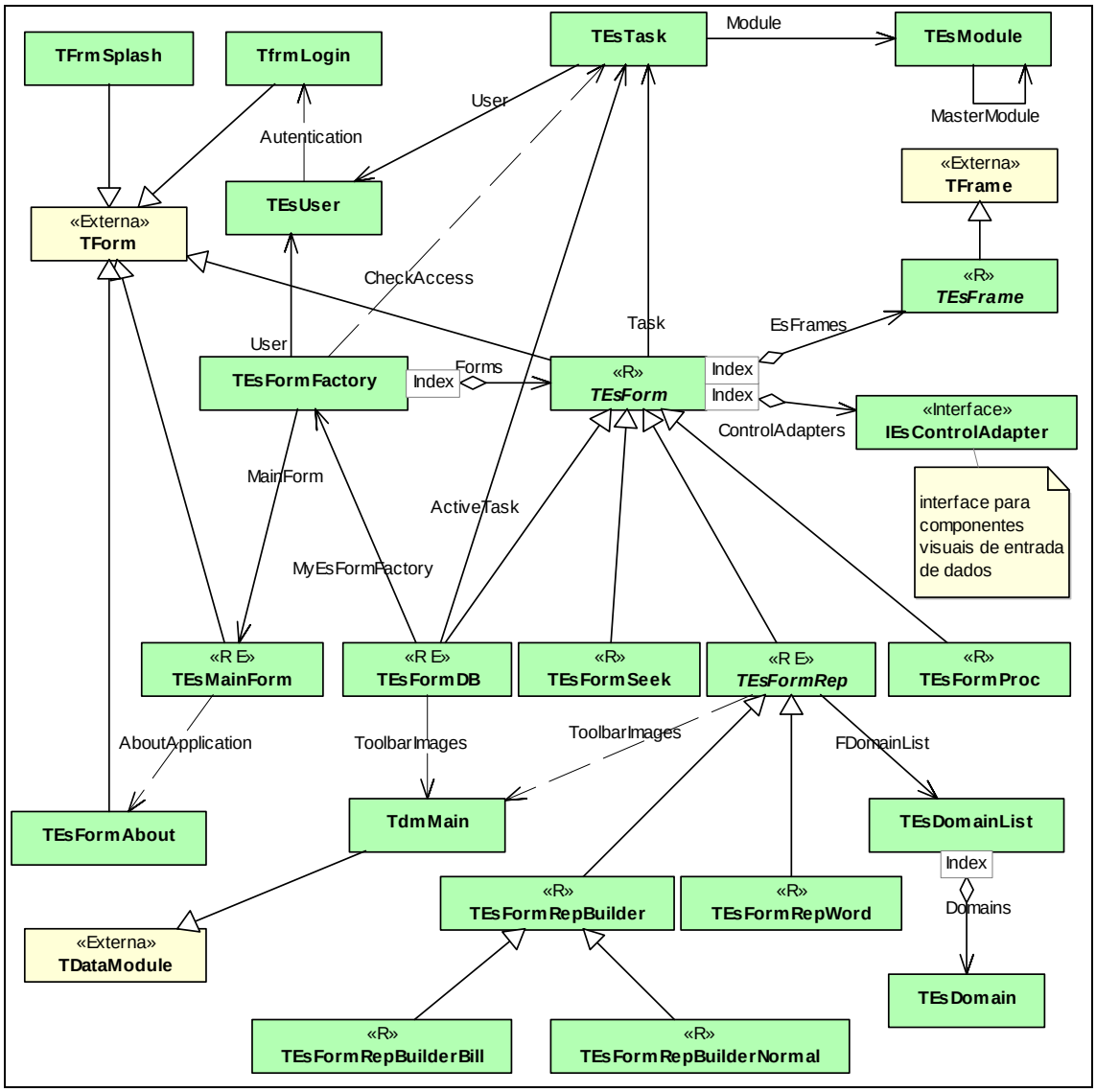

Figura 7.1 – Diagrama de Classes da Estrutura Arquitetural do Blendwork

Existem quatro classes que são instanciadas pelo Blendwork durante a inicialização da aplicação e suas instâncias permanecem ativas até o término da execução da aplicação. São elas:

- uma instância de uma subclasse de *TEsMainForm*, que deverá ser especializada pelo desenvolvedor para representar a interface gráfica (janela) principal da aplicação, onde serão adicionados os itens de menu com chamadas para a instância global de *TEsFormFactory*. A propriedade *Tag* do item de menu deve conter o código da tarefa que será chamada;

- uma instância global de *TEsFormFactory*, responsável pela verificação de acesso, criação, ativação e interação entre os formulários do tipo *TEsForm* dentro da janela principal. A verificação de acesso é validada pela instanciação da classe *TEsTask* que representa o formulário chamado. A criação dos formulários foi implementada baseando-se no padrão de projeto *Factory Method* (GAMMA, 1995). Para que seja possível a criação dos formulários, suas classes devem ser registradas em uma lista de classes na inicialização da aplicação;
- uma instância da classe *TEsUser*, representa o usuário que está utilizando a aplicação, instância esta que é mantida e referenciada através de uma propriedade da classe *TEsFormFactory*;
- uma instância da classe *TdmMain*, onde há funções comuns do Blendwork e listas de figuras que são exibidas nas barras de ferramentas dos formulários da classe *TEsFormDB* e *TEsFormRep* e suas subclasses.

A classe *TEsForm* é a classe base para os formulários do Blendwork e das aplicações sob ele desenvolvidas. Ela implementa funcionalidades comuns para as interfaces gráficas, interagindo uniformemente com os componentes visuais para edição de dados que implementam a interface *IEsControlAdapter* e também com os agrupamentos de componentes visuais descendentes da classe *TEsFrame*. Esta interação é importante para agilizar o desenvolvimento das aplicações, pois torna possível controlar de forma genérica os componentes visuais e seus agrupamentos. Cabe ressaltar, também, a possibilidade de se criar um formulário *TEsForm* embutido em recipientes (*containers*) gráficos, o que facilita a visualização e manipulação dos dados para o usuário.

A Figura 7.2 exibe as principais propriedades e métodos da hierarquia de classes descendentes de *TEsForm*, onde podemos observar alguns métodos abstratos (os métodos em itálico) sendo definidos nas subclasses e alguns métodos sendo sobrescritos. Os métodos *CreateEsForm* e *CreateEmbutido* são construtores. O primeiro cria os formulários dentro das fronteiras físicas da interface principal da aplicação e o segundo os cria dentro de outros recipientes gráficos, ou seja, embutidos.

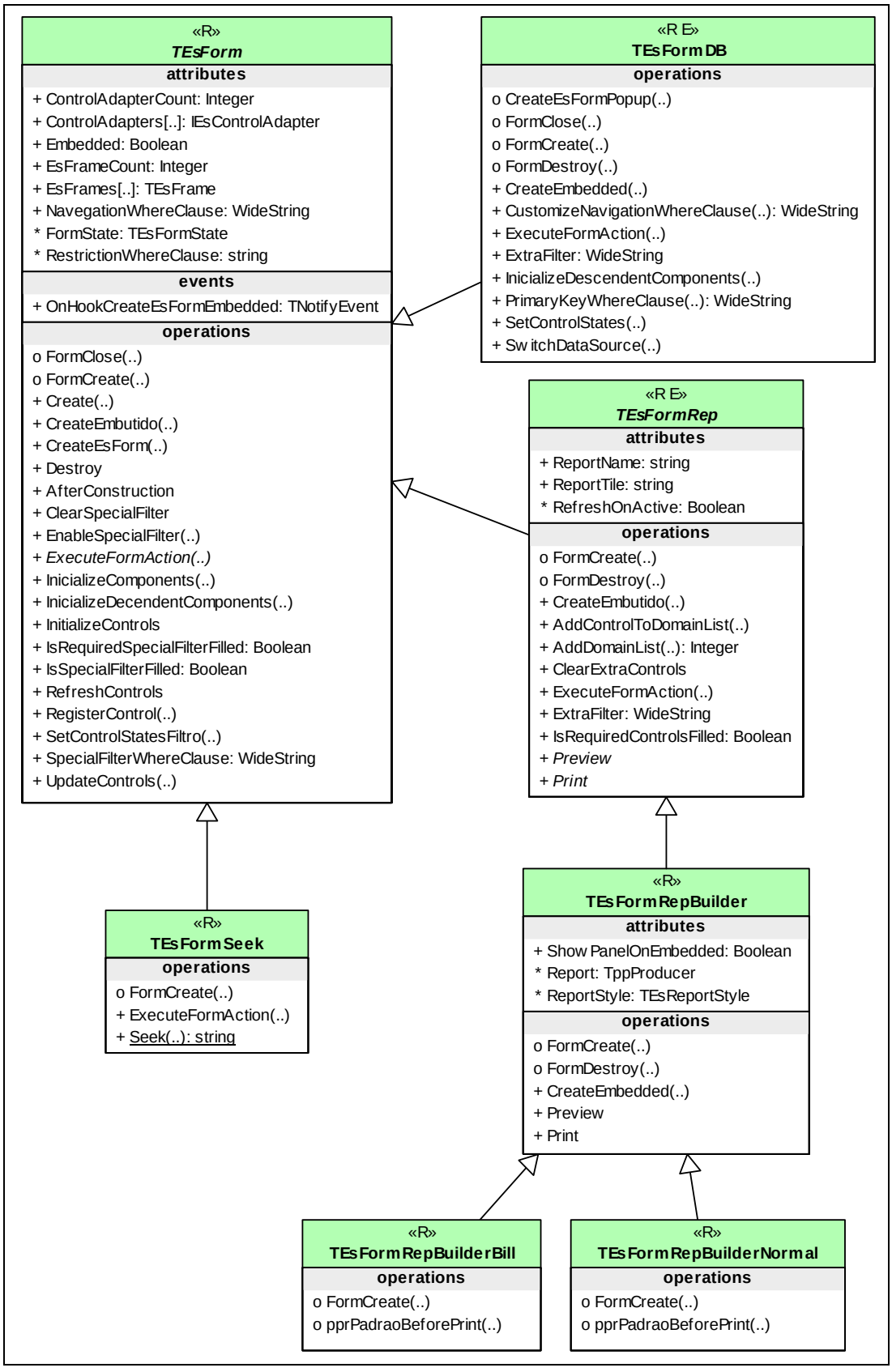

Figura 7.2 – Diagrama de Classes dos formulários exibindo as principais propriedades e métodos

As subclasses de *TEsForm* disponibilizadas no Blendwork objetivam disponibilizar recursos para concretizar as implementações comuns a cada tipo de formulário e/ou relatório identificados como abstrações durante a análise do domínio, detalhada no capítulo anterior. A classe *TEsFormDB* objetiva fornecer a interface gráfica e funcionalidades comuns aos cadastros para a manipulação das informações armazenadas no banco de dados e fornecer direcionamento para a implementação da lógica de negócio das aplicações. Algumas destas funcionalidades são executadas a partir da barra de ferramentas, conforme ilustrado na Figura 7.3, e detalhada abaixo:

- **Navegação entre os registros:** os dados são exibidos como registros ou linhas da tabela do banco de dados, e estes registros são alternados através dos dois primeiros botões da barra de ferramentas;
- **Manipulação dos dados:** a classe *TEsFormDB* prevê a inclusão, exclusão, alteração e consulta de registros das tabelas do banco de dados.
- **Filtro de dados:** para restringir a quantidade de registros retornados do banco de dados, é possível filtrá-los através dos próprios componentes visuais para edição de dados (os quais são adicionados nas subclasses de *TEsFormDB*). Além disso, pode-se adicionar outras opções de filtro através dos agrupamentos de componentes visuais (*frames* descendentes da classe *TEsFrame*);
- **Navegar entre formulários:** durante a especialização da subclasse é possível adicionar chamadas a formulários que manipulam dados relacionados. Pode-se, então, "navegar" de um formulário para outro (também descendente de *TEsFormDB*) filtrando-se os dados de acordo com o registro selecionado;
- **Chamar relatórios:** também durante a especialização da subclasse pode-se adicionar chamadas a relatórios que exibam dados relacionados. Os dados dos relatórios são filtrados de acordo com o registro selecionado no formulário de cadastro.

Para melhorar a ergonomia da interface gráfica, também foram adicionadas teclas de atalho que disparam as mesmas ações configuradas para os botões, dentre outras, como: atualizar as informações que estão sendo exibidas, alternar para o primeiro ou o

último registro. Para conhecer as teclas de atalhos, o usuário pode clicar com o botão direito do mouse sobre a barra de ferramentas e exibidas todas as ações disponíveis – em um menu – e os respectivos atalhos de teclado especificados ao lado de cada item de menu.

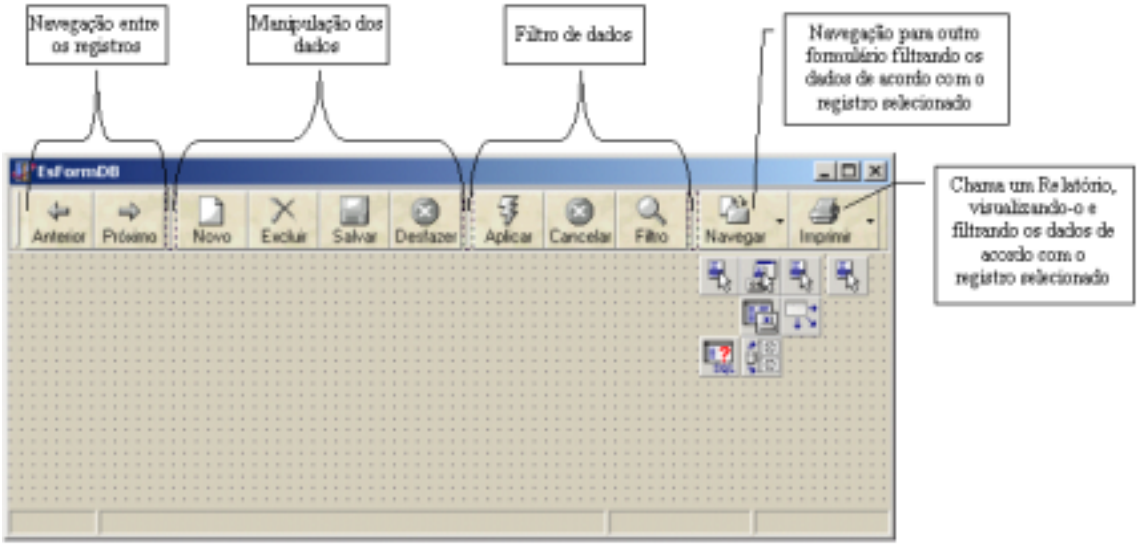

Figura 7.3 – Interface Gráfica da classe *TEsFormDB*

Os componentes mostrados na Figura 7.3 abaixo da barra de ferramentas não são visuais em tempo de execução, sendo que os primeiros quatro são utilizados em conjunto com a barra de ferramentas (lista de ações e itens de menu), e os quatro inferiores são utilizados para a interação com o banco de dados. Durante a especialização das classes, estes últimos devem ser configurados através dos comandos SQL.

A classe *TEsFormRep*, também descendente de *TEsForm*, objetiva fornecer funcionalidades comuns e interface gráfica para a visualização e impressão de relatórios, facilitando as opções de filtro dos dados a serem exibidos nos relatórios através das subclasses de *TEsFrame*, que geram comandos SQL para restringir a seleção de registros. Veja a interface gráfica da classe *TEsFormRep* ilustrada na Figura 7.4, que além da barra de ferramentas para visualizar ou imprimir o relatório, limpar os campos preenchidos e fechar a janela, inclui componentes não visuais em tempo de execução que são utilizados para configurar a interação com o banco de dados através dos

comandos SQL. O resultado da seleção de dados através destes componentes é exibido no relatório.

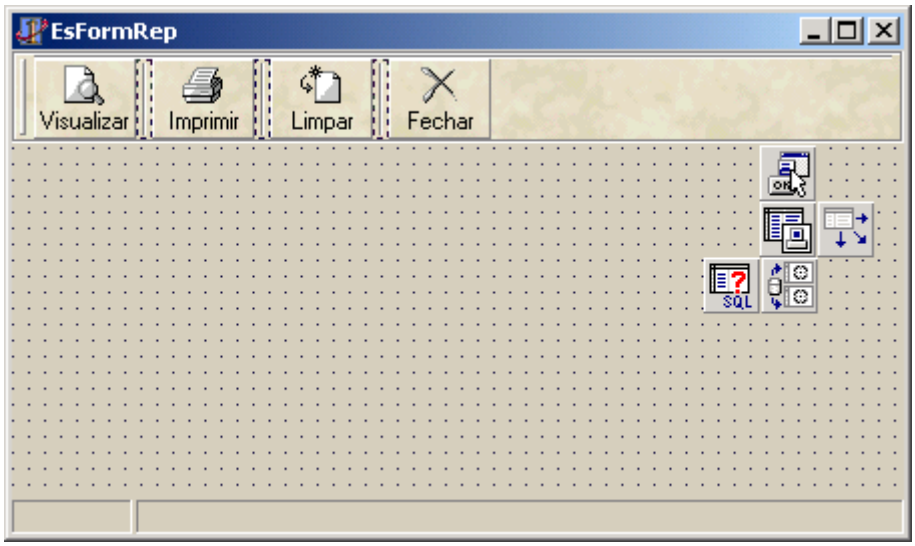

Figura 7.4 – Interface Gráfica da classe *TEsFormRep*

A classe *TEsFormRepWord* permite a impressão de relatórios em formato de carta através do processador de textos Microsoft Word. A classe *TEsFormRepBuilder*, subclasse de *TEsFormRep*, interage com a ferramenta de relatórios Report Builder, citada no capítulo anterior. As subclasses de *TEsFormRepBuilder* fornecem layout pré-definidos para os relatórios.

A classe *TEsFormSeek* também fornece uma interface gráfica e objetiva ser a classe base aos formulários utilizados para localizar registros em determinadas tabelas no banco de dados. A interface gráfica desta classe é ilustrada na Figura 7.5. Ao clicar no botão “Pesquisar”, os componentes que interagem com o banco de dados, também mostrados na figura, executam o comando SQL configurado para retornar os dados que satisfaçam as condições especificadas. Para ampliar as opções de pesquisa podem ser usadas subclasses de *TEsFrame* durante a especialização de um formulário de localização. Quando o usuário clica no botão “OK”, o registro selecionado na grade é retornado como resultado à função que o invocou.

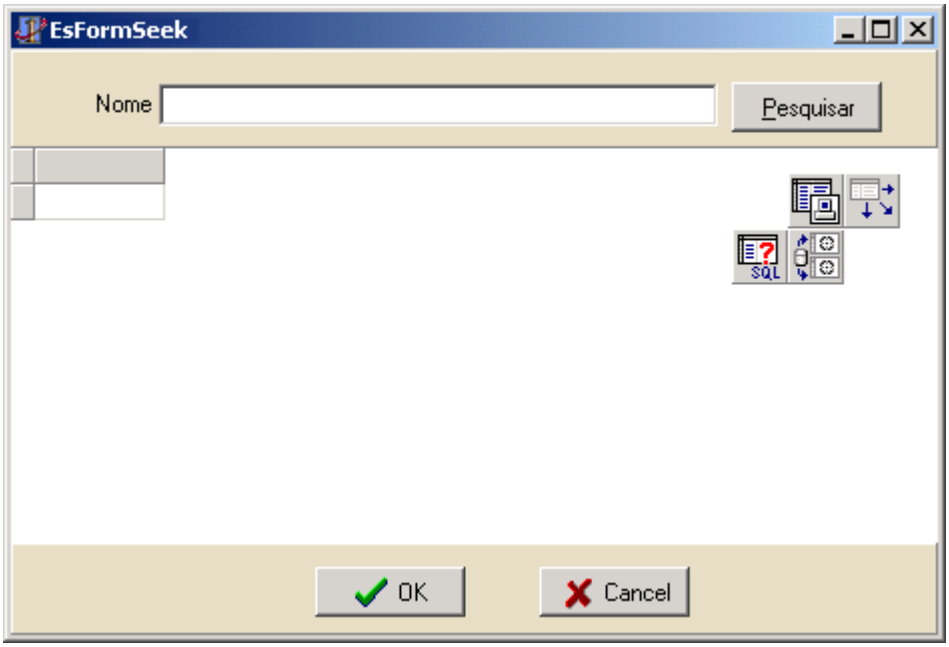

Figura 7.5 – Interface Gráfica da classe *TEsFormSeek*

A classe *TEsFormProc* não possui componentes visuais inseridos na sua interface gráfica, mas seu objetivo é ser especializada para execução de processos, tanto os implementados na aplicação cliente, quanto os procedimentos implementados no servidor de banco de dados.

Caso o desenvolvedor da aplicação utilize as subclasses de *TEsFormDB*, *TEsFormRep* e *TEsFormSeek* para implementar seus cadastros, relatórios e localização de dados, a interface gráfica será apresentada de forma padronizada por toda a aplicação, facilitando a aprendizagem aos usuários finais.

Finalmente, as classes *TEsDomain* e *TEsDomainList* implementam os domínios de valores, que são utilizados nos relatórios para exibir os dados de forma descritiva e também em dois componentes visuais – tratados na próxima seção.

7.3. Diagrama dos Componentes Visuais

Os componentes visuais para edição de dados foram adaptados utilizando-se o padrão de projeto *Adapter* (GAMMA, 1995). Para isso, os componentes necessários no

desenvolvimento de aplicações sob o Blendwork implementam a interface *IEsControlAdapter*, conforme o diagrama na Figura 7.6. Desta forma, os componentes são tratados de maneira uniforme e interagem de maneira genérica com os formulários da classe *TEsForm* e suas subclasses. A propriedade *ReadOnlyFlags* da interface *IEsControlAdapter*, adicionada em todas as classes que a implementam, determina se o estado somente para leitura ou não do campo para edição conforme o estado dos formulários descendentes de *TEsForm*. Outras propriedades e métodos podem ser adicionados na interface e implementados nas classes.

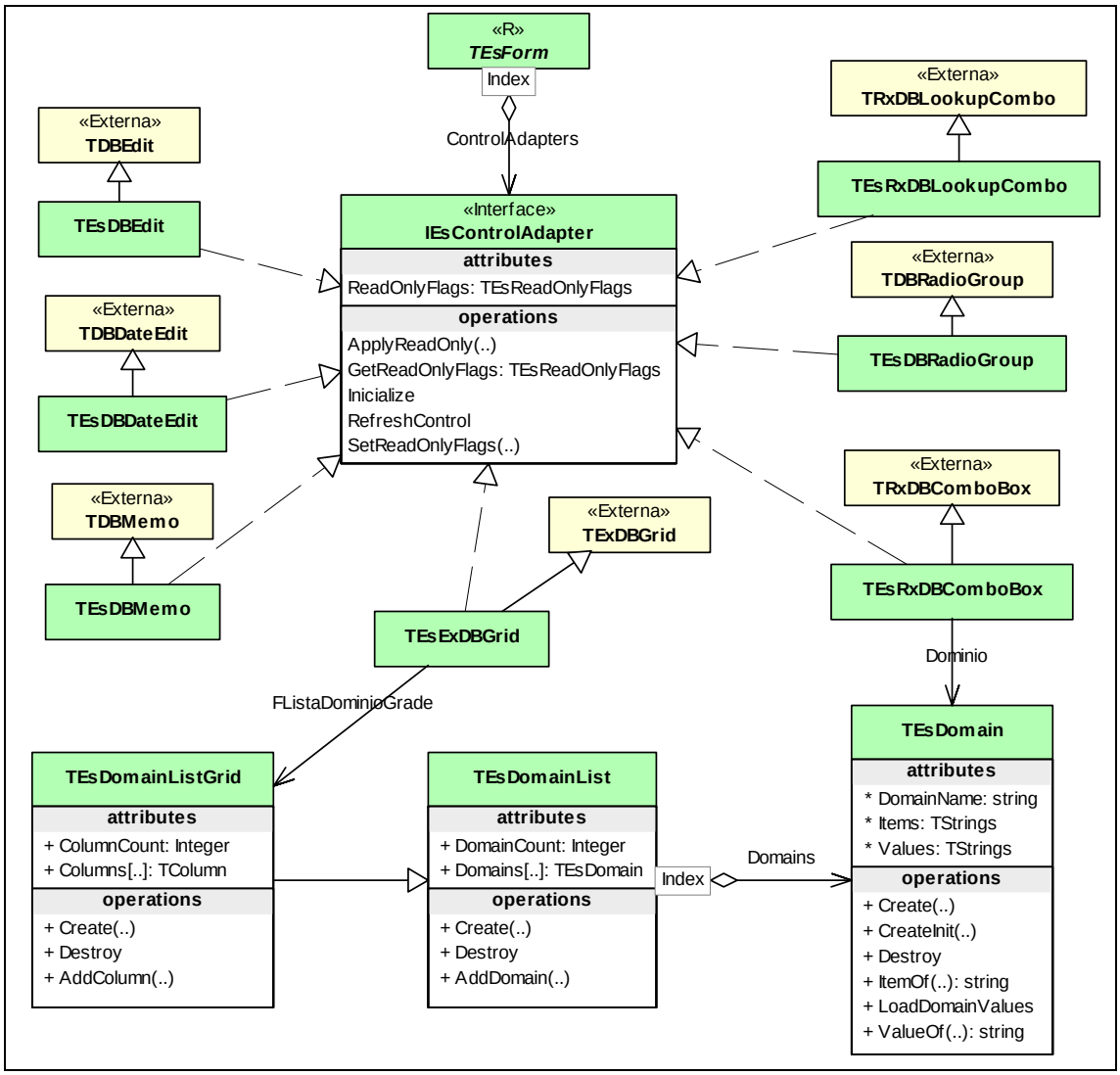

Figura 7.6 – Diagrama dos componentes visuais de edição

As classes *TEsExDBGrid* e *TEsRxDBComboBox* estão associadas às classes que implementam os domínios de valores, concretizando a utilização dos domínios de valores nas aplicações a serem desenvolvidas.

7.4. Diagrama dos Agrupamentos de Componentes

Dentre os benefícios providos pelas subclasses da classe *TEsFrame* podemos citar a padronização das interfaces gráficas e a funcionalidade de filtro de dados. A Figura 7.7 mostra o diagrama de classes de agrupamentos de componentes utilizadas no Blendwork e as suas classes concretas.

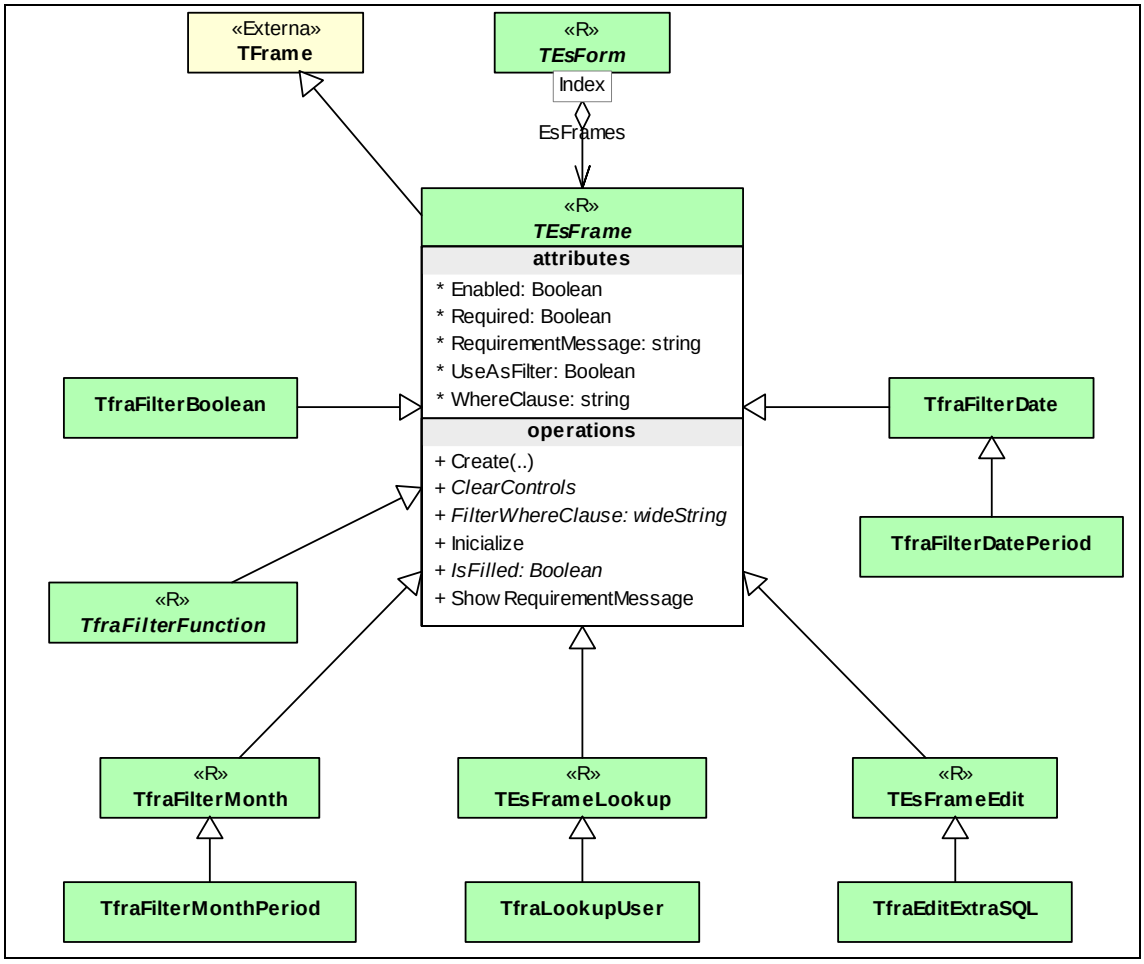

Figura 7.7 – Diagrama dos agrupamentos de componentes visuais

As subclasses que possuem "*filter*" no nome são utilizadas exclusivamente para filtrar dados, mas de maneira genérica. Um exemplo é a classe *TfraFilterMonth*, que possui dois componentes visuais: um rótulo (*label*) e um campo para edição. Quando colocado numa subclasse de *TEsFormRep* (formulário de relatórios) e preenchido pelo usuário, o agrupamento retornará uma linha para filtrar os dados dos relatórios, exibindo apenas os dados do mês selecionado. As subclasses de *TEsFrameLookup*, por sua vez, podem ser utilizadas para filtrar dados e para editá-los. Seu objetivo é exibir os registros de uma tabela do banco de dados numa lista para que o usuário selecione um dos valores. Um exemplo é o cargo ocupado por um funcionário. Recomenda-se criar uma subclasse de *TEsFrameLookup*, e especializá-la efetuando a seleção dos dados da tabela "CARGO" através de comandos SQL. Quando esta subclasse for colocada no formulário de cadastro de funcionários (subclasse de *TEsFormDB*), poder-se-á alterar o cargo do funcionário selecionando um cargo diferente na lista de cargos.

7.5. Diagrama do Dicionário de Dados

O dicionário de dados compreende as informações armazenadas no banco de dados referentes ao próprio Blendwork. Na versão atual do framework, as seguintes informações estão sendo armazenadas: usuários, tarefas, permissões, parâmetros do Blendwork e das aplicações, módulos de negócio e os domínios de valores.

A Tabela 7.1 apresenta os prefixos utilizados no nome das colunas das tabelas no banco de dados. Nos diagramas do Blendwork foram utilizados os mesmos prefixos adotados como padrão pela empresa Extersoft Informática.

Prefixo	Significado	Exemplo
CD	Código	CD_MODULO, CD_TAREFA, CD_ASSOCIADO
DS	Descrição	DS_LOGRADOURO, DS_TAREFA
DT	Data	DT_INCLUSAO, DT_NASCIMENTO, DT_MOVTO
IB	Identificador Boleano	IB_SITUACAO, (sempre 0 ou 1)
ID	Identificador	ID_PROCESSADO, ID_SEXO, ID_MOVTO
NM	Nome	NM_ASSOCIADO, NM_BANCO, NM_USUARIO
NR	Número	NR_CPF, NR_FONE, NR_ORDEM
PC	Percentual	PC_DESCONTO, PC_TITULAR, PC_PREMIO
QT	Quantidade	QT_VALE, QT_DIA_CARENCIA, QT_TALAO
VL	Valor	VL_SALARIO, VL_DESCONTO, VL_MOVTO

Tabela 7.1 – Prefixos para Atributos do Modelo Relacional

A Figura 7.8 ilustra o Diagrama de Entidade Relacionamento do Dicionário de Dados, onde o nome das tabelas é precedido das siglas ESD (Extersoft Dicionário).

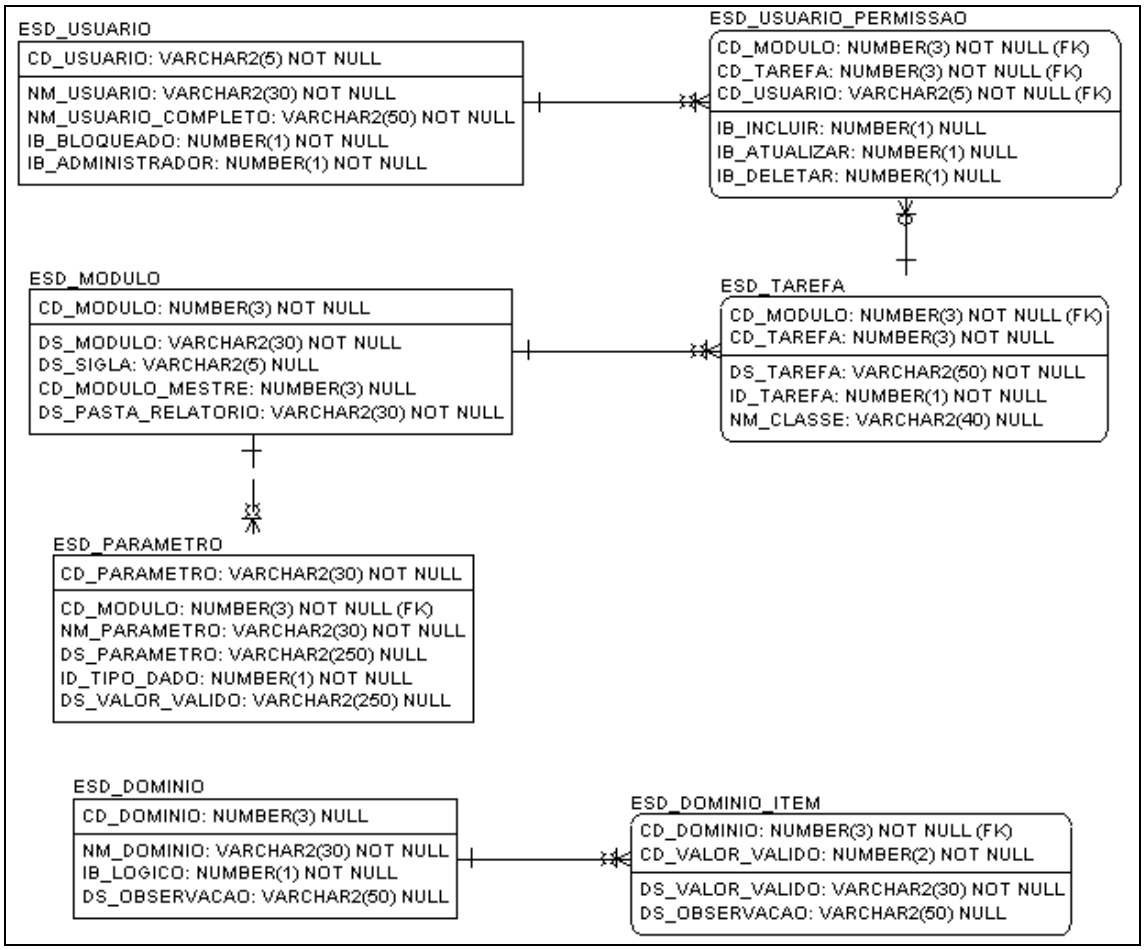

Figura 7.8 – Diagrama de Entidade Relacionamento do Dicionário de Dados

As informações armazenadas no Dicionário de Dados são recuperadas pelas classes do Blendwork para sua utilização em tempo de execução. A Tabela 6.2 mostra o mapeamento entre as classes e as tabelas do banco de dados.

Classe	**Tabelas**
TEsTask	ESD_TAREFA, ESD_USUARIO_PERMISSAO
TEsUser	ESD_USUARIO
TEsDomain	ESD_DOMINIO, ESD_DOMINIO_ITEM
TdmMain	ESD_PARAMETRO

Tabela 7.2 – Mapeamento entre o Dicionário de Dados e as classes do Blendwork

Cabe salientar que as classes apresentadas na tabela acima somente recuperam as informações do dicionário, não efetuando inclusões, exclusões ou alterações das informações lá armazenadas. Logo, como o Blendwork necessita de classes concretas para efetuar estas operações, foram criadas subclasses da classe *TEsFormDB* para o cadastro das informações e subclasses de *TEsFormRepBuilderNormal* para os relatórios, conforme ilustra a Figura 7.9.

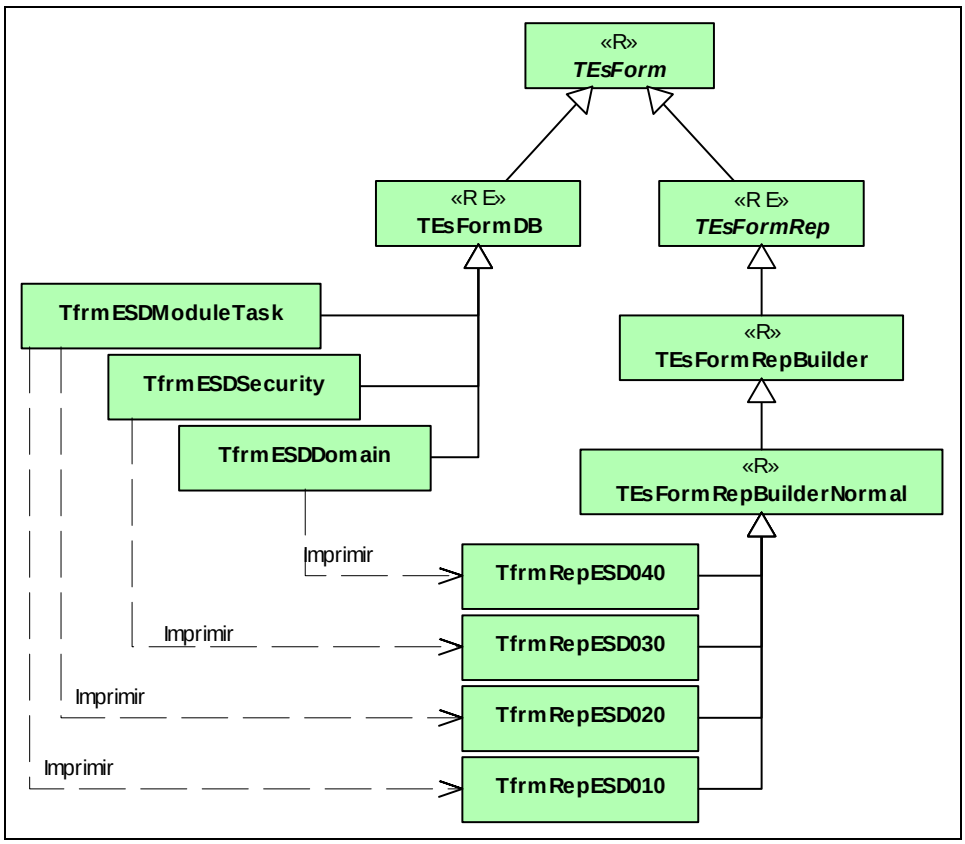

Figura 7.9 – Diagrama de Classes do Dicionário de Dados

Numa instância da classe *TfrmESDModuleTask*, pode-se manipular os dados referentes aos módulos de negócio, tarefas e parâmetros. Este cadastro é implementado em mestre/detalhe, em que os módulos de negócio correspondem ao mestre e as tarefas e os parâmetros são detalhes. Isso significa que quando um módulo de negócio estiver selecionado, somente as tarefas e os parâmetros a ele relacionados serão exibidos. As classes *TfrmRepESD010* e *TfrmRepESD020* são utilizadas para emitir os relatórios dos módulos e suas tarefas e dos módulos e seus parâmetros, respectivamente. Através da classe *TfrmESDSecurity* são cadastrados os usuários e suas permissões em mestre/detalhe. A classe *TfrmRepESD030* é usada para emitir relatórios das permissões liberadas aos usuários. Já na classe *TfrmESDDomain* são cadastrados os domínios de valores e seus itens, também em mestre/detalhe e a classe *TfrmRepESD040* emite relatórios sobre estes domínios.

7.6. Considerações

Dentre as funcionalidades previstas e implementadas no Blendwork, assim como os benefícios que elas proporcionam, podemos citar:

- As classes do Blendwork provêem a padronização das interfaces gráficas de usuário, fazendo com que as aplicações sob ele desenvolvidas sejam mais uniformes, facilitando o aprendizado dos usuários finais;
- As interações das classes com o banco de dados trazem agilidade no desenvolvimento de aplicações, reduzindo assim seus custos;
- Os agrupamentos dos componentes visuais incentivam e facilitam a reutilização de interface de usuário e de código fonte;
- Novos componentes visuais podem ser adicionados ao Blendwork, adaptando-os, caso necessário, através da implementação da interface *IEsControlAdapter*. Desta forma eles poderão interagir e colaborar com os formulários.

8. Validação do Blendwork: o Blendus

8.1. Visão Geral

A validação das implementações das classes do Blendwork ocorreu através do desenvolvimento do Blendus, um sistema de gerenciamento de benefícios. A Figura 8.1 ilustra a arquitetura do Blendus, onde cada uma das aplicações que o compõem foram construídas utilizando-se o Blendwork.

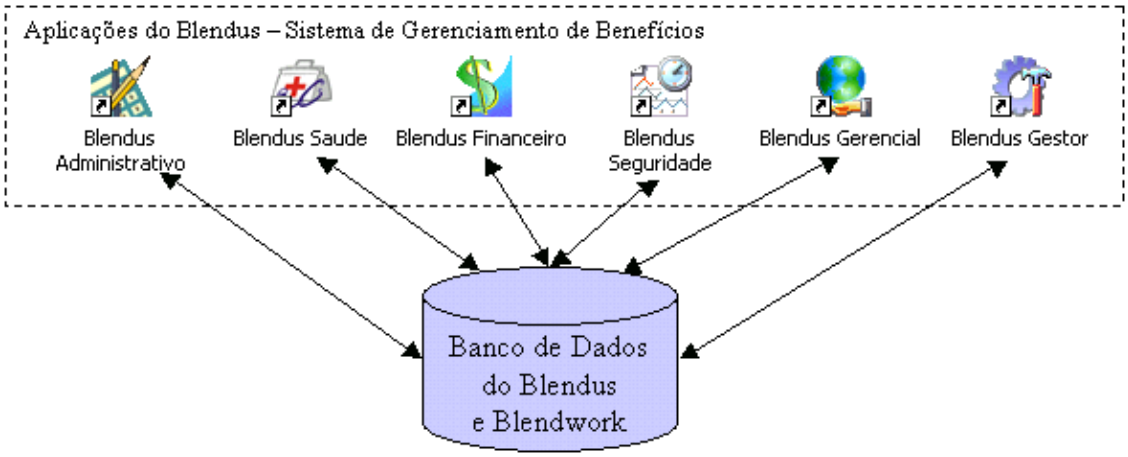

Figura 8.1 – Aplicações do Blendus

Cada aplicação do Blendus é composta de um ou mais módulos destinados a atender determinadas áreas de negócios, distintas entre si. Os módulos de negócio foram distribuídos entre as aplicações de acordo com a responsabilidade atribuída a cada departamento da Fundação, sendo que uma aplicação está implantada somente em um departamento. Um módulo de negócio, ou algumas de suas classes, também pode estar presente em mais de uma aplicação. As aplicações e os respectivos módulos de negócio que já estão implantados serão tratados nas seções subseqüentes.

Os diagramas de entidade relacionamento do Blendus não serão ilustrados por motivos de sigilo e propriedade intelectual da empresa, porém os diagramas de classes necessários à ilustração das extensões advindas do Blendwork o serão. Mesmo assim, somente alguns dos módulos de negócio serão detalhados.

Além das cores introduzidas nos diagramas de classes do Blendwork no capítulo anterior, surgiu a necessidade de ilustrar mais claramente a diferenciação entre as classes do Blendus. Assim, foi adotada a cor de preenchimento azul claro para diferenciar as classes específicas de um módulo de negócio e a cor laranja para as classes do Blendus que são compartilhadas entre as aplicações.

Para facilitar a nomenclatura das classes no Blendus, cada um dos módulos de negócio recebeu uma sigla, e foi adotada a seguinte padronização de nomes:

- Todos os nomes de classes descendentes de *TEsForm* são precedidos de *Tfrm*;
- Todos os nomes das classes descendentes de *TEsFormDB* são precedidos, além de *Tfrm*, da sigla do módulo ao qual estão vinculados;
- Todos os nomes das classes descendentes de *TEsFormRep* possuem o seguinte nome: o prefixo *TfrmRep,* mais a sigla do módulo de negócio ao qual está vinculado e o número da tarefa que a representa;
- Os nomes das classes descendentes de *TEsFormSeek* são precedidos de *TfrmSeek*;
- Os nomes das classes descendentes de *TFrame* são precedidos por *Tfra;*
- Os nomes das classes descendentes de *TEsFrameLookup* são precedidos de *TfraLookup*.

Alguns diagramas de classes não exibem todas as classes que são utilizadas no módulo, pois se tornam muito complexos e de difícil visualização por conter uma grande quantidade de classes. Foram então exibidas as classes mais relevantes em cada um dos casos apresentados.

8.2. Blendus Gestor

O Blendus Gestor foi desenvolvido para gerenciar as informações comuns que são compartilhadas entre todas as aplicações do Blendus. Atualmente está disponibilizada para a equipe de informática, responsável por manter o sistema e gerenciá-lo. Nesta aplicação foi criado um módulo chamado Cadastros Genéricos, que implementa o

conjunto de classes necessárias ao gerenciamento das informações compartilhadas em todo o sistema, e que eventualmente a própria classe pode ser disponibilizada em todas as aplicações do sistema.

As classes concretas que compõem o Dicionário de Dados do Blendwork formam um módulo que está incluído no Blendus Gestor. É através dele que são feitas as instanciações das classes do dicionário – detalhadas no capítulo anterior – para a manutenção das informações lá armazenadas.

A Figura 8.2 ilustra o Blendus Gestor e os módulos nele incluídos: Cadastros Genéricos – GEN, e o Dicionário de Dados do Blendwork – ESD. O pacote nomeado Agrupamentos Genéricos é um conjunto de classes descendentes de *TFrame* e *TEsFrame*. São, portanto, agrupamentos de componentes visuais, que são utilizados em todo o sistema Blendus, independentemente de módulos.

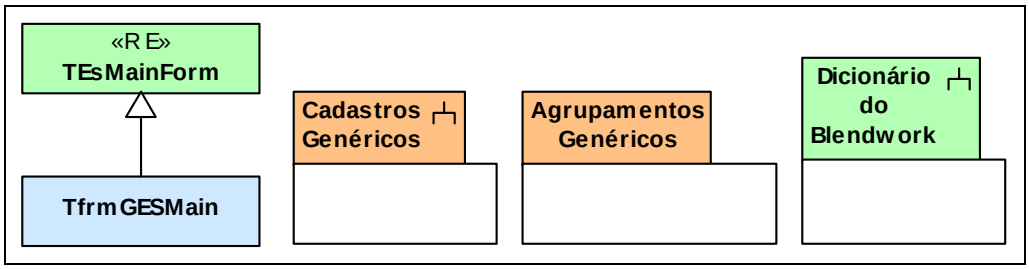

Figura 8.2 – Blendus Gestor e seus módulos

Cabe salientar que em todas as figuras ilustrativas de aplicações do Blendus mostradas neste capítulo haverá a ilustração de uma subclasse de *TEsMainForm*, que é a representação da interface gráfica principal da aplicação, classificada como redefinível e essencial.

Na seqüência serão apresentados os diagramas dos Agrupamentos Genéricos e dos Cadastros Genéricos.

8.2.1. Agrupamentos Genéricos

Estas classes não formam um módulo, mas um conjunto de classes que pode ser utilizado em todas a aplicações do Blendus. Pode-se observar na Figura 8.3 que a maioria das classes são descendentes das classes do Blendwork. O nome das classes tende a ser auto-explicativo, como no exemplo de *TfraLookupEmpresa*: é uma classe de agrupamento de componentes visuais (*frame*), subclasse de *TEsFrameLookup* e exibe uma lista das empresas cadastradas no Blendus.

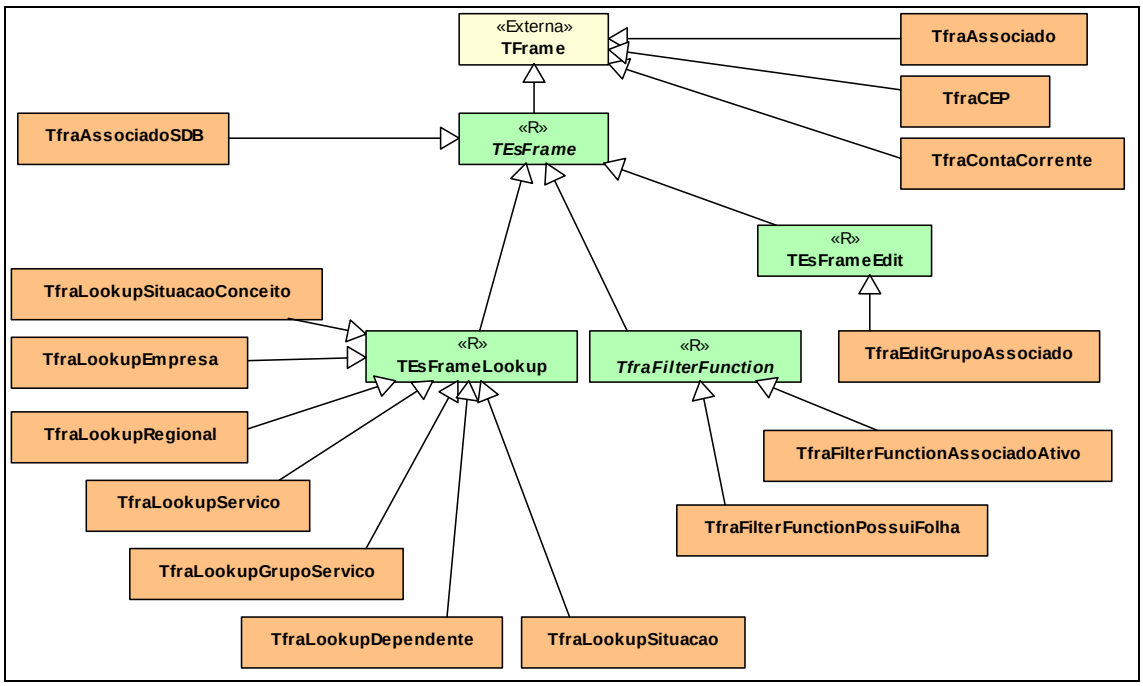

Figura 8.3 – Diagrama de Classes dos Agrupamentos Genéricos

8.2.2. Cadastros Genéricos – GEN

A Figura 8.3 ilustra as principais classes do módulo de cadastros genéricos. Estas classes são utilizadas em várias aplicações, senão em todas, do Blendus. Já percebemos a utilização de algumas classes apresentadas no diagrama anterior e como elas são utilizadas para compor as funcionalidades implementadas nos formulários de cadastros (subclasses de *TEsFormDB*).

Como mostra o diagrama (Figura 8.4), existem duas subclasses de *TEsFormSeek*: uma para localização de associados (*TfrmSeekAssociado*) e outra de seus dependentes

(*TfrmLocDependente*). A localização de associados é sempre invocada pela classe *TfraAssociado* ou *TfraAssociadoSDB*.

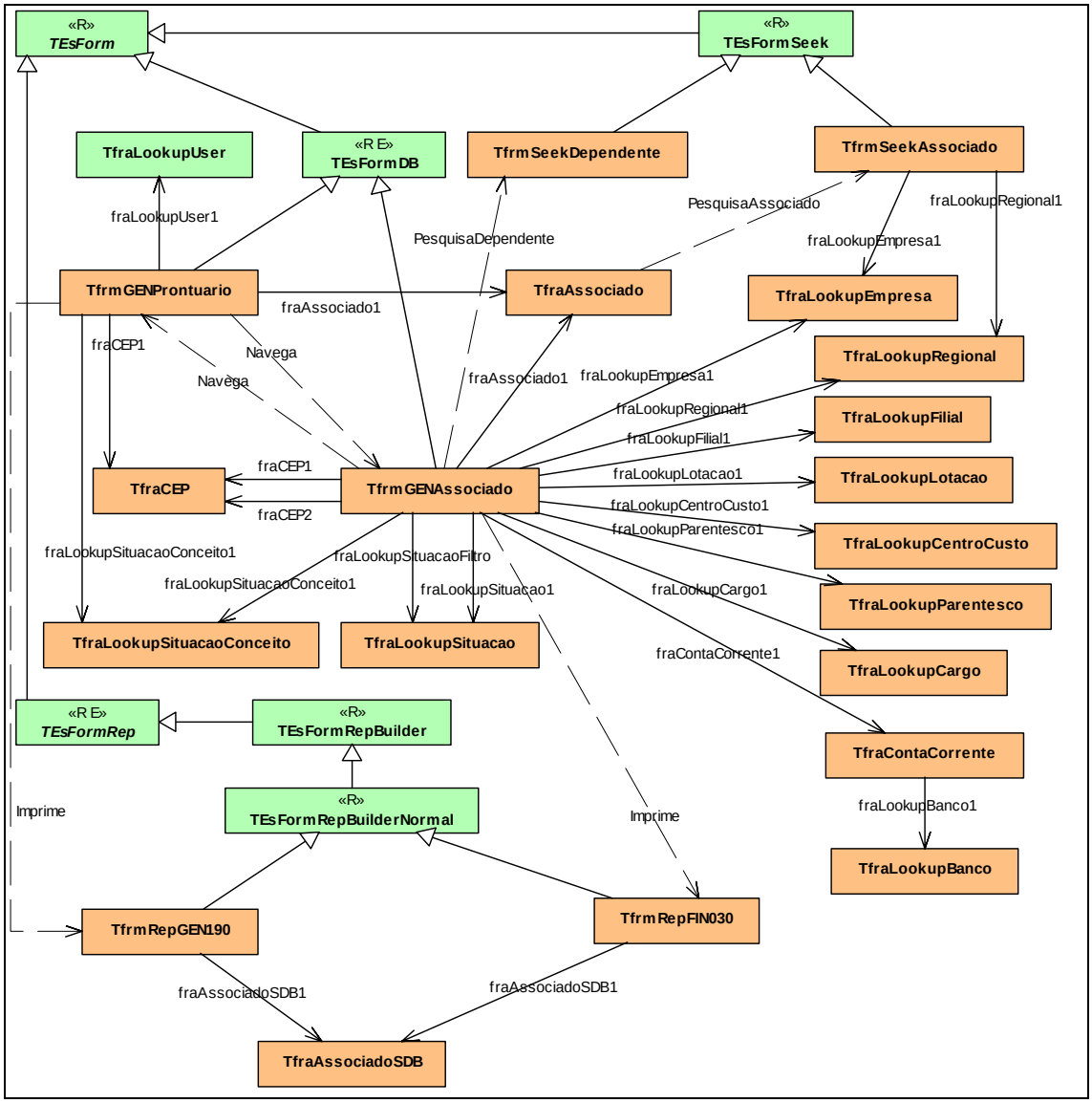

Figura 8.4 – Diagrama de Classes do módulo de Cadastros Genéricos

As classes de cadastro *TfrmGENAssociado* e *TfrmGENProntuario* são utilizadas em todas as aplicações do Blendus, em que a primeira gerencia as informações do cadastro de associados da Fundação e a segunda mantém um histórico de atendimentos a cada um dos associados. Por este motivo suas interfaces gráficas foram selecionadas para ilustração neste trabalho. A Figura 8.5 ilustra a interface da classe

TfrmGENAssociado. Os círculos vermelhos indicam exemplos de instâncias das classes de agrupamento de componentes visuais, sendo *TfraAssociado* e *TfraLookupEmpresa*, respectivamente.

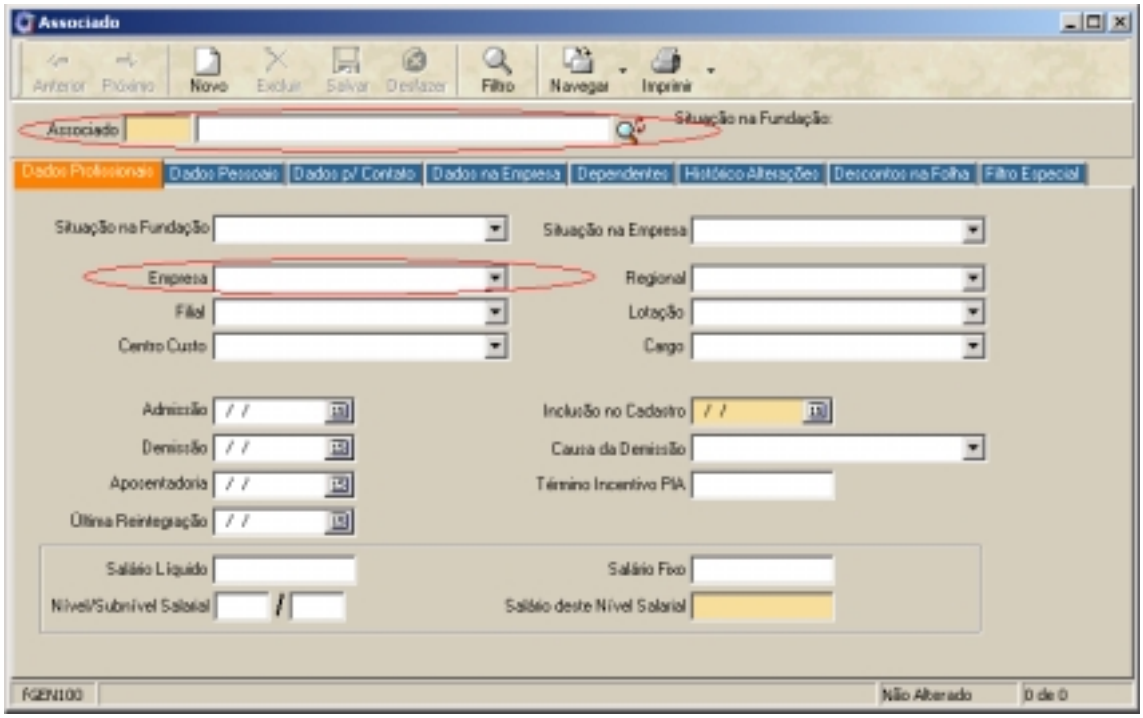

Figura 8.5 – Interface gráfica da classe *TfrmGENAssociado*

A Figura 8.6 ilustra a interface da classe *TfrmGENProntuario*. O círculo vermelho indica uma instância das classes de agrupamento de componentes visuais, *TfraLookupUsuario*, que é uma classe concreta do Blendwork, conforme ilustrado em um diagrama de classes do capítulo anterior.

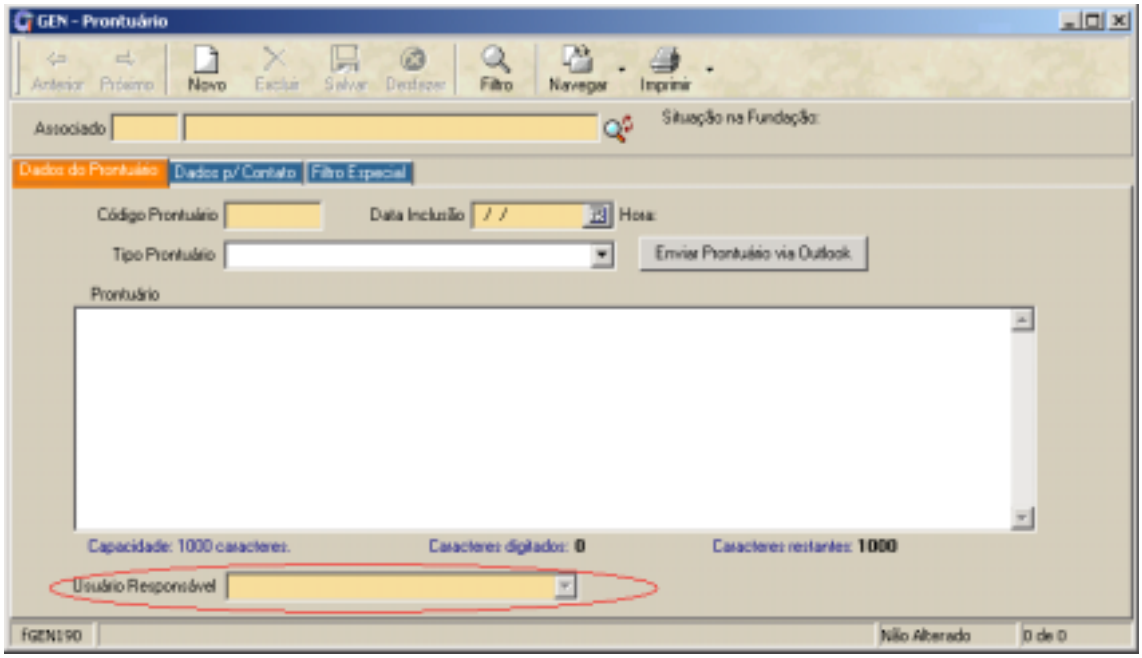

Figura 8.6 – Interface gráfica da classe *TfrmGENProntuario*

Os componentes de edição preenchidos de cor laranja, na Figura 8.5 e 8.6, estão em estado somente para leitura. Este estado é determinado de acordo com as propriedades de cada campo (definida pelo desenvolvedor da aplicação) e de acordo com o estado do formulário. Este recurso é controlado pela classe *TEsForm* interagindo com os componentes de edição que implementam a interface *IEsControlAdapter*.

As classes de relatório *TfrmRepGEN190* e *TfrmFIN030* também são utilizadas em todo o sistema. A primeira visualiza e emite relatórios dos prontuários cadastrados para cada um dos associados e a segunda visualiza e emite a folha de pagamento dos associados.

8.3. Blendus Administrativo

A aplicação Blendus Administrativo está implantada no departamento Administrativo da Fundação e é composta dos seguintes módulos, conforme ilustra a Figura 8.7: Aluguel Ginásio – GIN, Seguro de Vida – VID, Auxílios – AXL, Programa de Alimentação do Trabalhador – PAT.

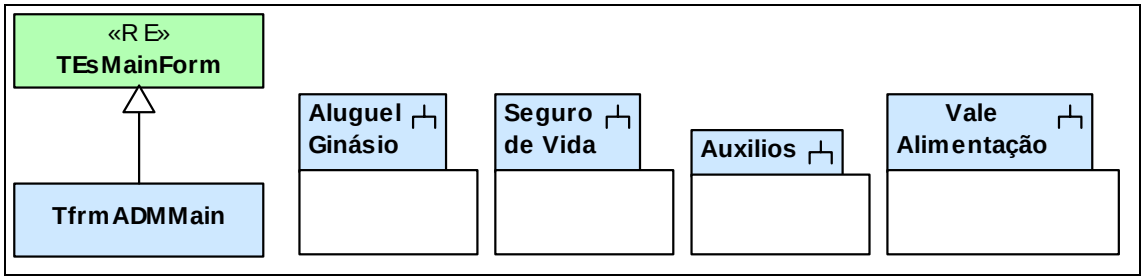

Figura 8.7 – Blendus Administrativo e seus módulos

Abaixo, estão apresentados os diagramas de cada um dos módulos do Blendus Administrativo.

8.3.1. Aluguel Ginásio – GIN

Este módulo tem como objetivo gerenciar a locação, por horários pré-determinados, do ginásio de esportes aos associados da Fundação.

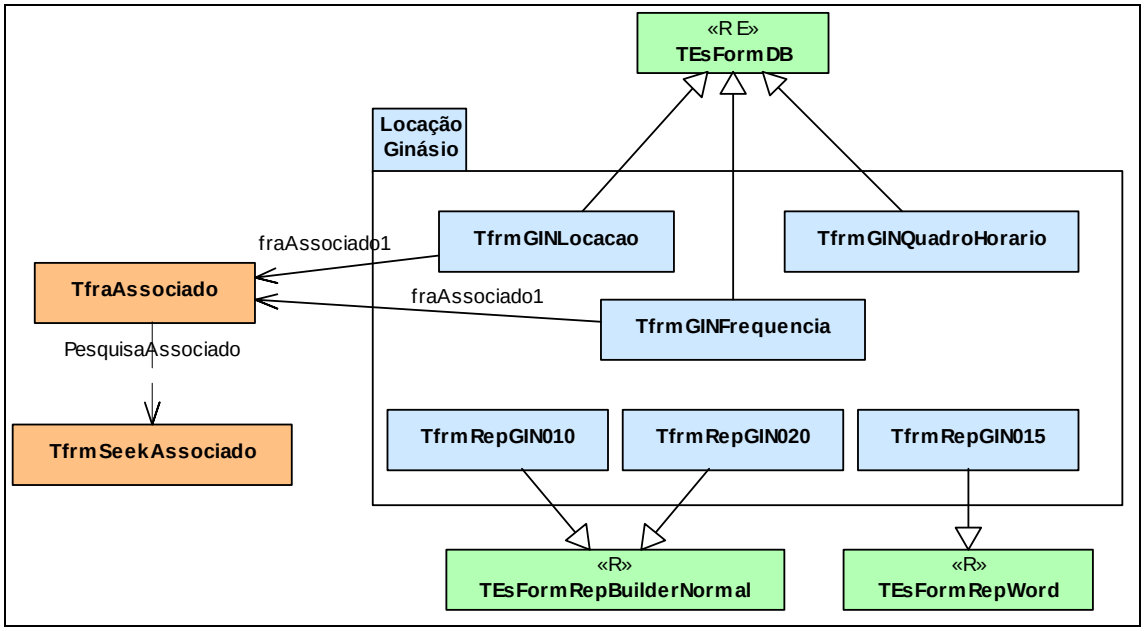

Figura 8.8 – Diagrama de Classes do módulo Aluguel Ginásio – GIN

Podemos observar na Figura 8.8 três formulários de cadastros implementados no módulo Aluguel Ginásio. A classe *TfrmGINQuadroHorario* objetiva cadastrar todos os horários em que o ginásio pode ser locado para jogos. Através da classe *TfrmGINLocacao* é possível cadastrar os grupos que formam os times para os jogos. Se houver um grupo de associados inscritos para jogar no mesmo horário, a cobrança da locação é feita em rodízio. Cada mês um deles é responsável pelo pagamento. A classe *TfrmGINFrequencia* registra a freqüência com que o grupo tem comparecido aos jogos. Caso haja muitas ausências, o horário pode ser cancelado para aquele grupo.

As classes de relatórios emitem a relação dos grupos inscritos, a relação de descontos de um determinado mês e uma carta de aviso de desconto. Os valores são descontados em débito automático na conta corrente bancária do associado.

8.3.2. Seguro de Vida – VID

O módulo de Seguro de Vida objetiva gerenciar os inscritos em um seguro de vida em grupo administrado pela Fundação, e suas classes estão ilustradas na Figura 8.9.

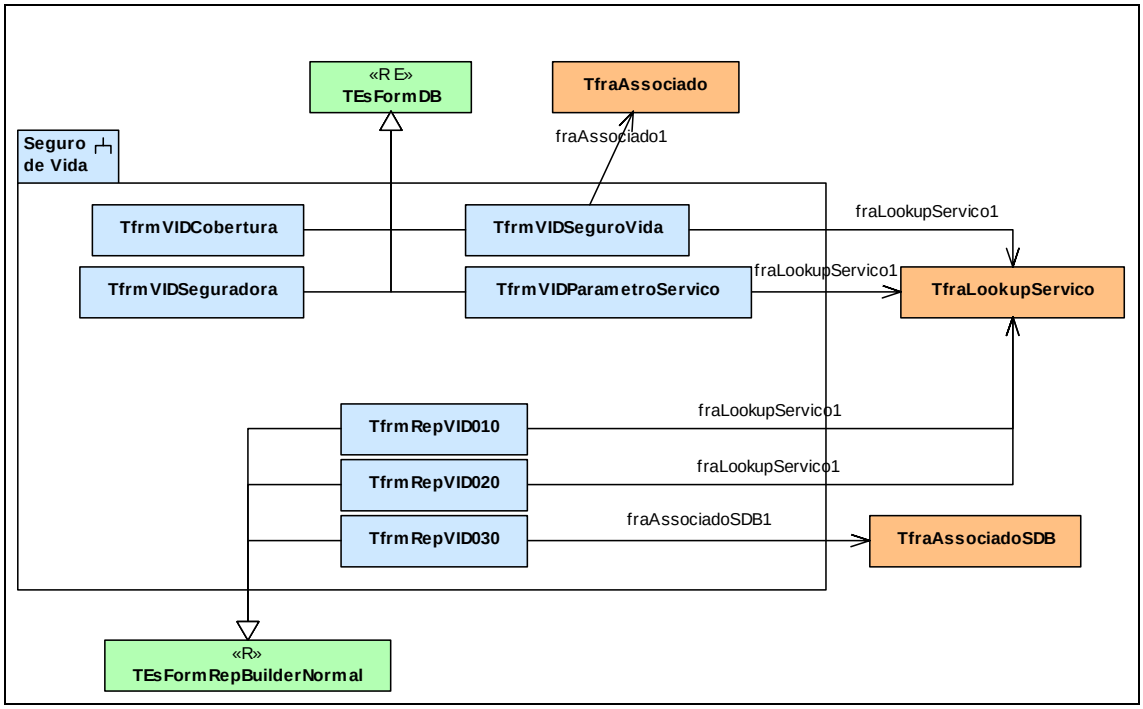

Figura 8.9 – Diagrama de Classes do módulo Seguro de Vida – VID

Como existem duas modalidades de seguro, foi preciso criar um cadastro específico para parametrização dos serviços de seguro. Cabe salientar que todos os benefícios gerenciados pelo sistema são cadastrados como serviço. Por exemplo, todas as modalidades de auxílios aos associados são cadastradas como serviços.

Este módulo também calcula as coberturas em valores de cada um dos associados, levando em consideração o valor da mensalidade por ele paga. Os descontos são efetuados na folha de pagamento dos associados.

8.3.3. Auxílios – AXL

Através deste módulo são gerenciados os auxílios financeiros fornecidos pela Fundação aos associados. Estes auxílios se caracterizam em ajuda de custo para determinadas necessidades. Dentre os auxílios podemos citar: Auxílio Natalidade, Auxílio Funeral, Auxílio Oftalmológico, etc. Nos dois primeiros casos, na ocorrência de nascimento ou falecimento de algum membro da família do associado, é fornecida uma certa ajuda de custo. No terceiro, na compra de óculos ou lentes de contato, um percentual do valor é ressarcido ao associado.

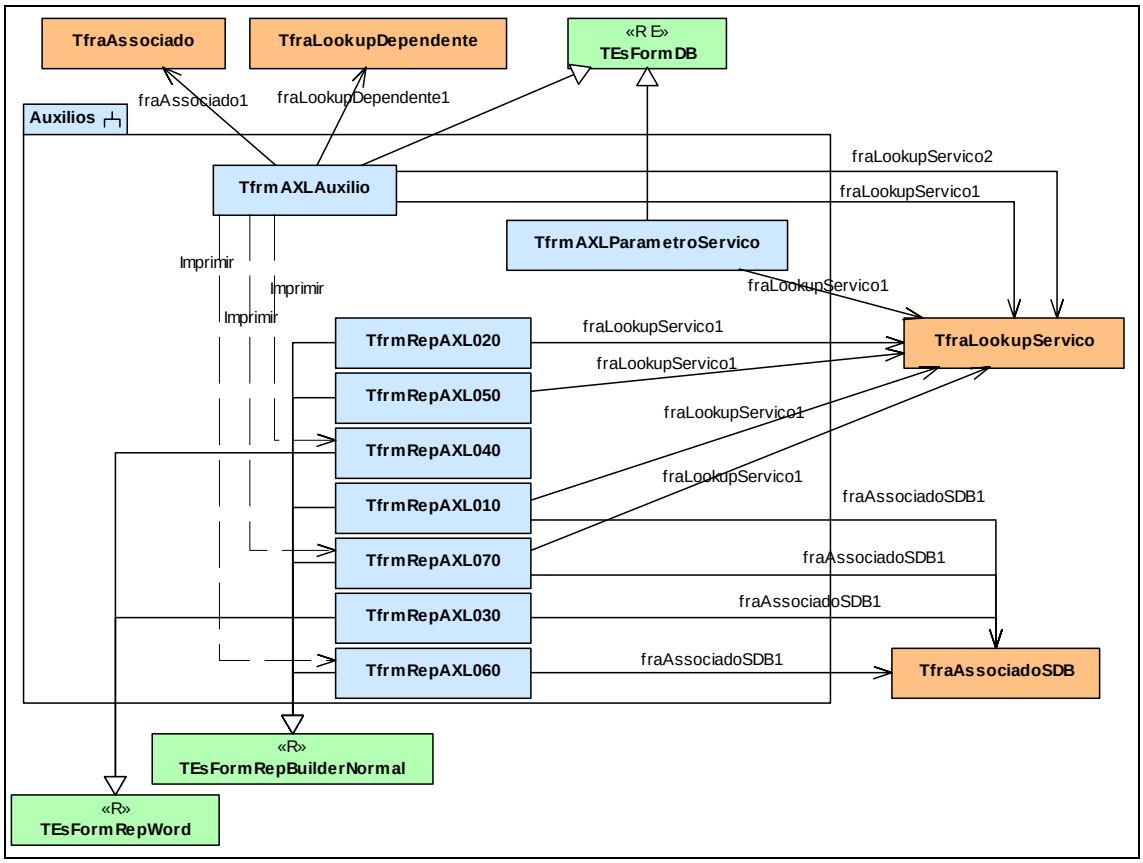

Figura 8.10 – Diagrama de Classe do módulo Auxílios – AXL

A Figura 8.10 ilustra o diagrama, em que se observam duas classes de cadastros – uma para a parametrização dos serviços de auxílios e outra para o cadastramento dos auxílios – e várias classes de relatórios.

Mas adiante, veremos que este mesmo módulo também é utilizado no Blendus Saúde, para gerenciamento dos auxílios sob responsabilidade do setor de Saúde da empresa.

8.3.4. Programa de Alimentação do Trabalhador – PAT

A empresa gerencia, para sua empresa mantenedora, o pedido e distribuição dos vales alimentação do Programa de Alimentação do Trabalhador – PAT. A cada mês os pedidos são registrados, incluindo-se os funcionários da empresa mantenedora que tem direito ao recebimento do vale alimentação, e após isso o pedido é enviado para a empresa fornecedora.

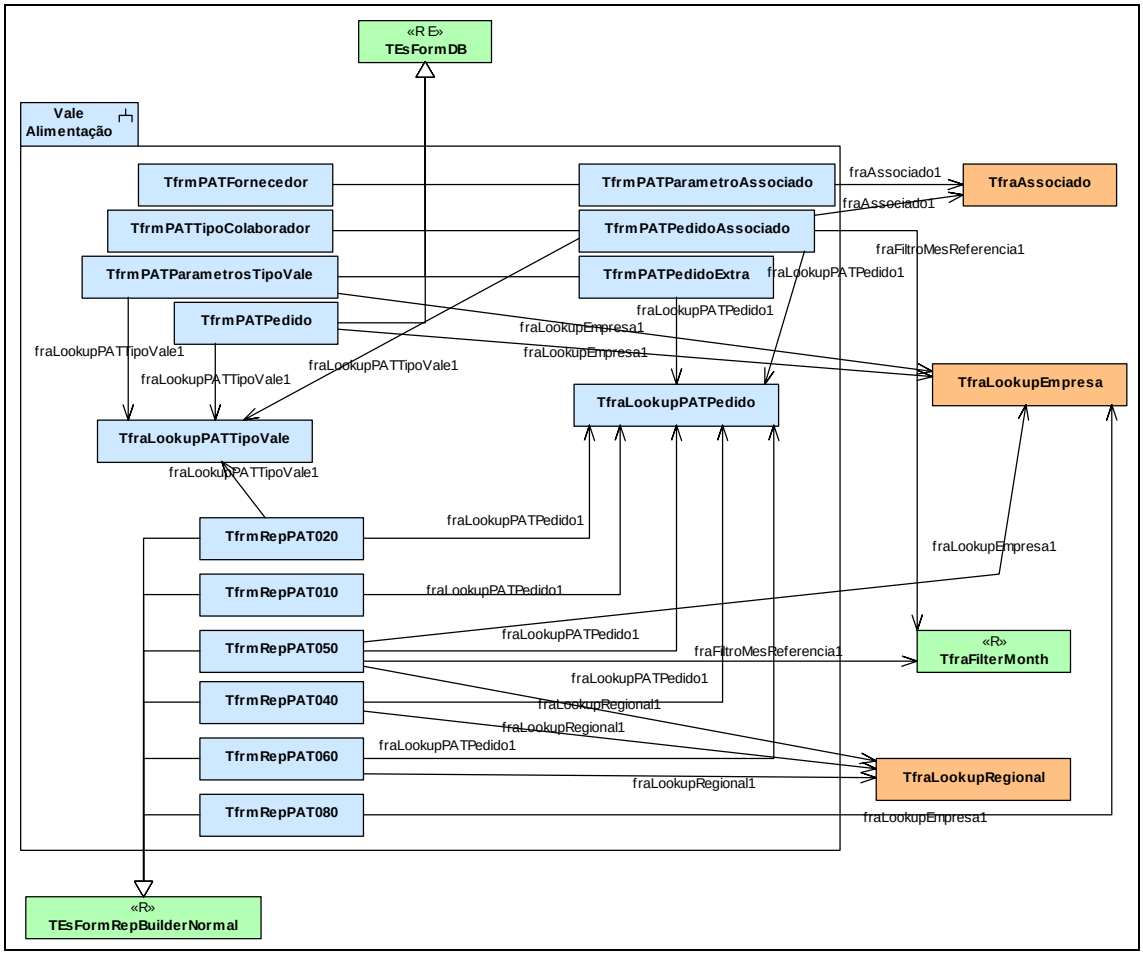

Figura 8.11 – Diagrama de Classes do módulo Vale Alimentação – PAT

Pela utilização das instâncias das classes ilustradas na Figura 8.11 é que se faz o gerenciamento deste processo.

8.4. Blendus Saúde

Atualmente o Blendus Saúde, implantado no departamento de Saúde da empresa, possui os seguintes módulos implantados: Planos de Saúde, Digitalização de Documentos, Auxílios. O módulo de Planos de Saúde está implantado parcialmente, sendo a classe do formulário de Autorizações de Procedimentos Médicos a mais utilizada.

A Digitalização de Documentos é empregada para arquivar e catalogar as solicitações de autorização de procedimentos, que são atualmente recebidas por fax, e-mail ou pessoalmente. As autorizações que chegam em papel são digitalizadas através da utilização de scanners.

Os módulos desta aplicação estão ilustrados na Figura 8.12, assim como a classe da interface gráfica principal.

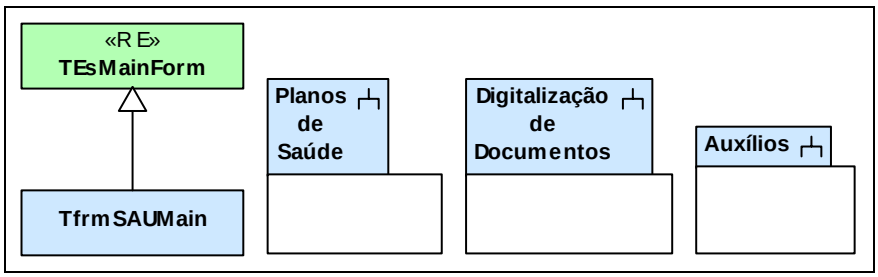

Figura 8.12 – Blendus Saúde e seus módulos

Depois da aplicação já estar implantada, surgiu uma nova necessidade: foram criados novos auxílios que seriam gerenciados pelo departamento de saúde. Como já havia um módulo gerenciador de auxílios implementado e implantado, mas no Blendus Administrativo, ele foi adaptado para atender às duas necessidades. O principal problema estava no fato que as informações dos auxílios gerenciados por uma aplicação não poderiam estar acessíveis na outra. A solução foi incrementar o Blendwork com uma funcionalidade extra: restringir informações de acordo com a aplicação onde a classe estava sendo instanciada. Para que isso fosse possível, foram adicionadas propriedades na classe *TEsFormDB* e uma tabela associativa entre módulos e serviços no banco de dados.

8.5. Blendus Financeiro

O Blendus Financeiro gerencia tanto as informações de cobrança e pagamento aos associados como a concessão de empréstimos através dos módulos Controle de Cobrança e Empréstimos, respectivamente.

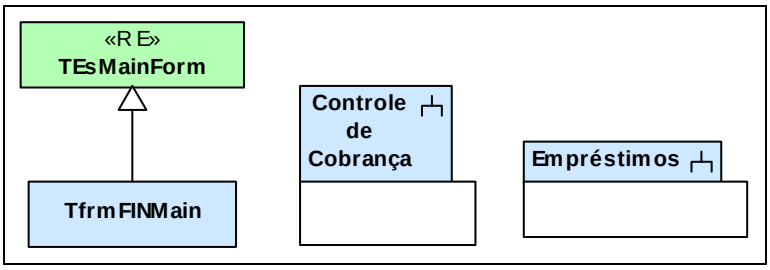

Figura 8.13 – Blendus Financeiro e seus módulos

A Figura 8.13 ilustra os módulos do Blendus Financeiro e a classe que representa sua interface gráfica principal.

8.6. Blendus Seguridade

Utilizando os módulos do Blendus Seguridade, o departamento de Seguridade da empresa gerencia os planos de pensão e pecúlio, auxílio desemprego e documenta os processos jurídicos eventualmente movidos contra a empresa.

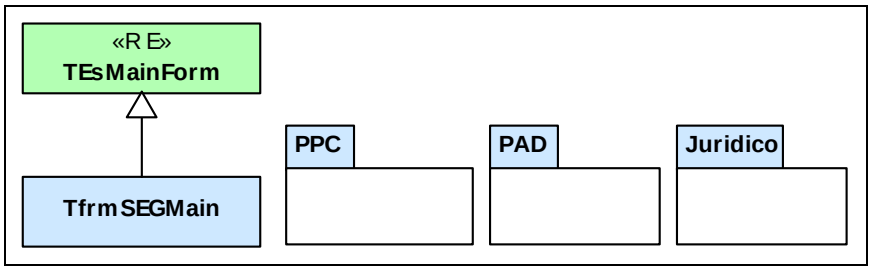

Figura 8.14 – Blendus Seguridade e seus módulos

A Figura 8.14 ilustra os módulos: Plano de Pensão e Pecúlio – PPC, Plano Auxílio Desemprego – PAD e Jurídico.

8.7. Blendus Gerencial

Com o Blendus Gerencial surgiram novas necessidades para o Blendwork. Almejando-se uma interface gráfica de usuário que fosse mais interativa e facilitasse a navegação entre os diversos relatórios e cadastros, utilizou-se o recurso de embutir um

formulário em outro. Como resultado, todos os relatórios ficam dentro das fronteiras de uma só janela, contudo somente um é exibido de cada vez.

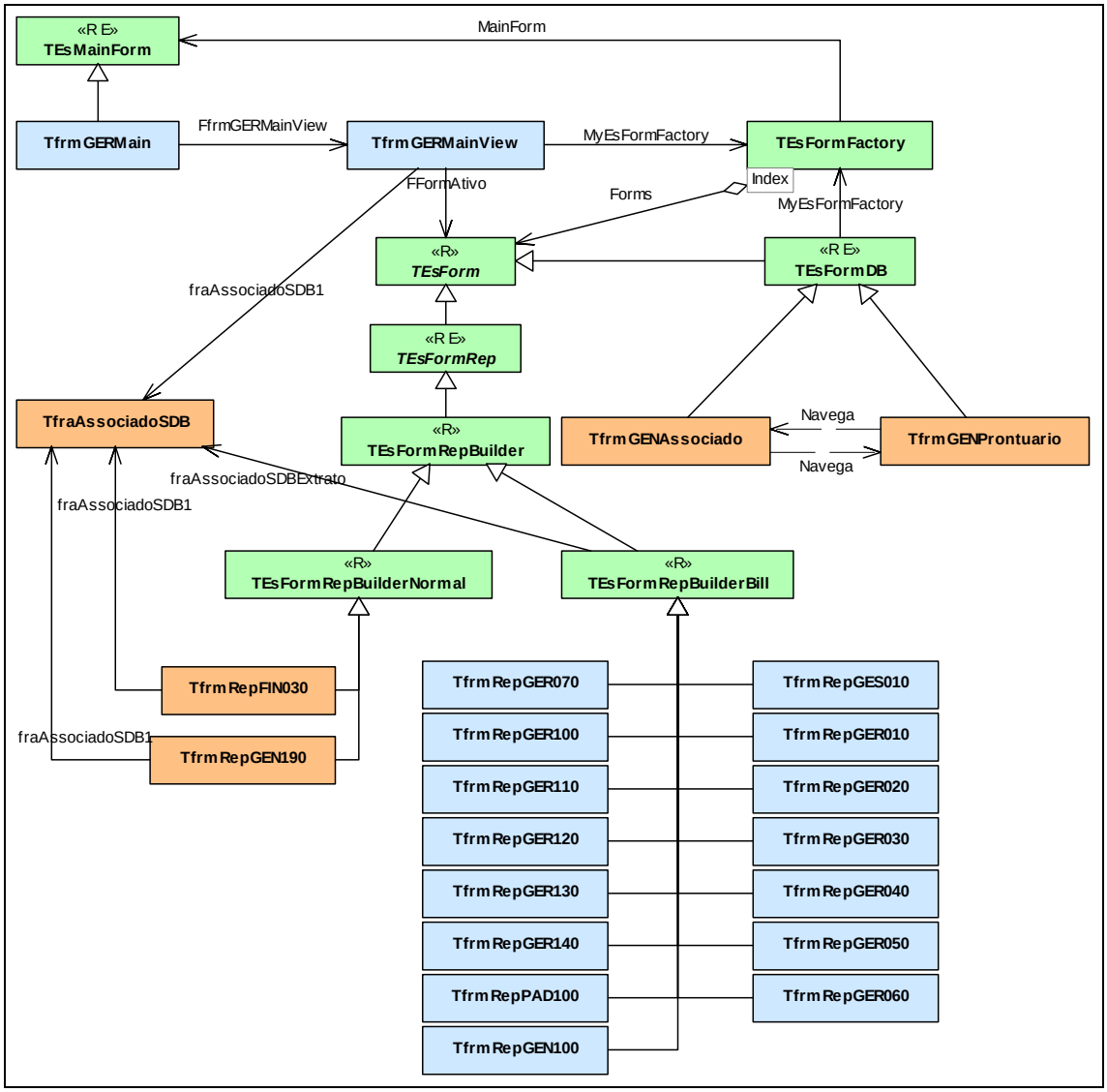

Figura 8.15 – Diagrama de Classes do Blendus Gerencial

A Figura 8.15 ilustra o diagrama de classes do Blendus Gerencial, onde se observa a classe *TfrmGERMainView*, que possui seu próprio gerenciamento de formulários. A Figura 8.16 ilustra a interface gráfica desta classe. Os relatórios e/ou cadastros são invocados através dos botões relacionados à esquerda da interface gráfica. Caso a classe do formulário chamado já esteja instanciada, somente as informações são atualizadas

quando da mudança da matrícula do associado, disponível através de uma instância da classe *TfraAssociadoSBD*, na parte superior da interface.

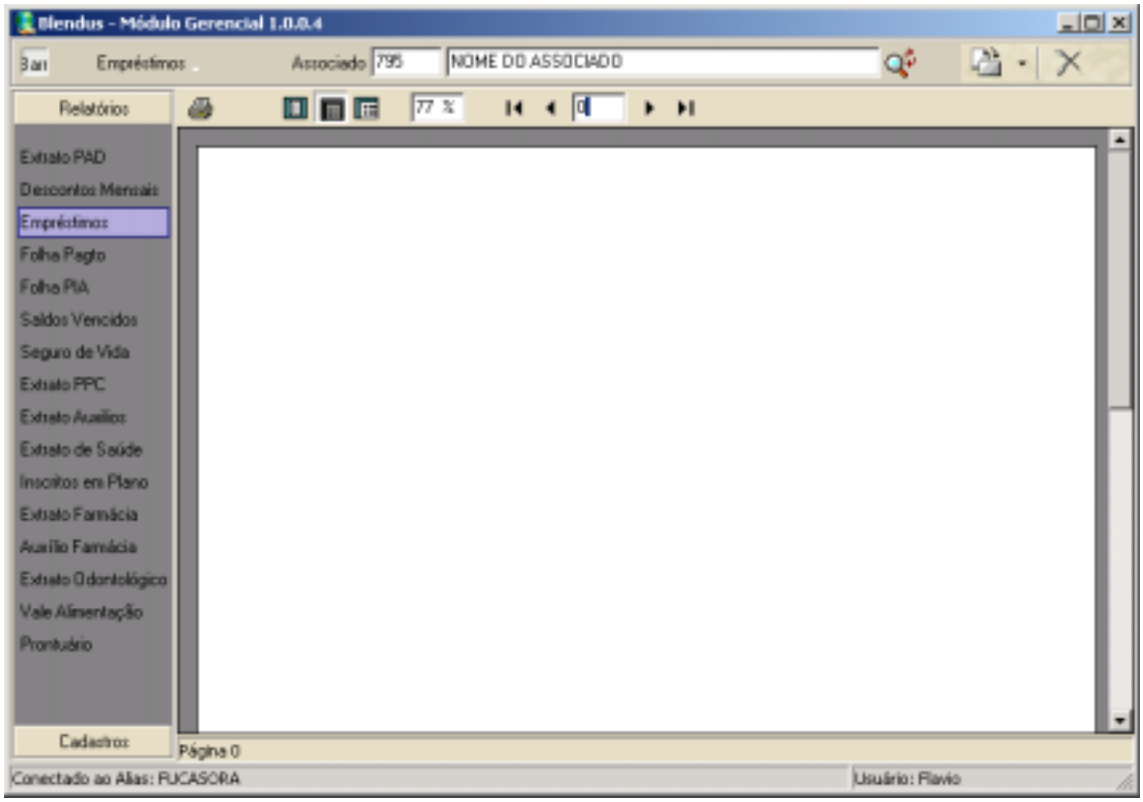

Figura 8.16 – Interface gráfica da classe *TfrmGERMainView*

8.8. Considerações

Os módulos do Blendus foram selecionados para desenvolvimento conforme as necessidades da empresa e conforme a problemática apresentada pelos mesmos módulos no sistema anterior.

Apesar do processo inicial de desenvolvimento do Blendus ter despendido mais tempo que o normal, pelo fato de ter sido desenvolvido enquanto o Blendwork ainda evoluía, considera-se satisfatórios os resultados, pois o tempo empregado na manutenção diminuiu quando comparado com o SIF (sistema anterior).

9. Considerações Finais

9.1. Conclusão

O objetivo geral deste trabalho foi atendido através do projeto e implementação do Blendwork, apresentados nos capítulos 6 e 7. O Blendwork é um framework orientado a objetos que provê uma infra-estrutura para o desenvolvimento de aplicações cliente/servidor de duas camadas. Seu conjunto de classes fornece fluxo de controle e colaborações entre suas classes e as subclasses estendidas e especializadas para o desenvolvimento de uma aplicação. Sua infra-estrutura envolve interface gráfica de usuário, transações com o banco de dados e controle de acesso e permissões.

Considera-se que os objetivos específicos também foram alcançados, pois os levando em consideração e a este trabalho, podemos reiterar:

- Foram estudados os principais conceitos relacionados aos frameworks orientados a objetos;
- Para o domínio de aplicações estudado, foi definida uma arquitetura de aplicações adequada e que atende as necessidades avaliadas;
- Foram identificadas as similaridades encontradas num conjunto de aplicações existentes, no mesmo domínio em que o Blendwork foi construído;
- Quanto aos custos de desenvolvimento e manutenção de aplicações RAD, conclui-se que o Blendwork é um importante aliado, pois através dele torna-se ainda mais fácil a prototipagem e a evolução deste protótipo para tornar-se uma aplicação completa;
- As interfaces gráficas de usuário para entrada de dados foram padronizadas através dos formulários, componentes visuais e agrupamentos de componentes visuais;
- O framework possui leiautes de relatórios que podem ser utilizados para padronizá-los em toda a aplicação;
- A segurança das aplicações – em termos de controle de acesso e permissões – é implementada através do Blendwork;

- O Blendwork provê um controle do fluxo de execução das colaborações existentes entre suas classes, sendo que as classes especializadas para a construção das aplicações são chamadas pelas classes do framework.

A validação do Blendwork, concretizada através do desenvolvimento do Blendus, conforme apresentado no capítulo 8, foi primordial para a correção de erros e a sua evolução; porém, algumas dificuldades foram encontradas durante este processo. Para se manter a compatibilidade entre as aplicações já desenvolvidas e implantadas aos usuários finais e o Blendwork, cada alteração substancial efetuada no framework, principalmente em termos de quantidade ou tipos de parâmetros de métodos e/ou funções, resultava na alteração de todas as classes que as utilizavam. Com o aumento da quantidade de módulos e aplicações, esta dificuldade tornou-se cada vez mais aparente. Tanto que, ao final deste trabalho, uma importante consideração a ser feita é a seguinte: para a continuidade da evolução e aprimoramento do Blendwork, recomenda-se desvinculá-lo das primeiras aplicações sob ele desenvolvidas para evitar o aumento dos custos desta evolução.

Desta forma, conclui-se que o custo da evolução do Blendwork a partir deste ponto seria justificável pelo desenvolvimento de novas aplicações cliente/servidor de duas camadas com banco de dados.

Pela utilização do Blendwork no desenvolvimento de uma aplicação, comparando-se com o desenvolvimento de uma aplicação utilizando somente os recursos proporcionados pelos ambientes de desenvolvimento visual, observa-se os seguintes benefícios:

- Redução do tempo de desenvolvimento;
- Redução da quantidade de código para posterior manutenção da aplicação;
- Padronização das interfaces gráficas de usuário;
- Ergonomia das aplicações, por sua interface gráfica e interação/navegação entre os formulários;
- Padronização dos leiautes de relatórios;

Assim, constatados os benefícios reais que podem ser alcançados pela utilização do Blendwork, acredita-se ele ser uma solução prática para alguns dos problemas defrontados no desenvolvimento e manutenção de aplicações.

No contexto deste trabalho, as seguintes contribuições podem ser identificadas:

- O Blendwork, um produto final, implementado em um ambiente de desenvolvimento visual (Borland Delphi) utilizado comercialmente, que o torna acessível àqueles que queiram utilizá-lo para o desenvolvimento de novas aplicações cliente/servidor de duas camadas;
- O projeto arquitetural de um framework orientado a objetos, que pode ser utilizado para implementá-lo em outros ambientes de desenvolvimento visuais;
- A disponibilidade de um trabalho científico – que pode ser continuado e aprimorado – apresentando o conhecimento resultante da análise de um domínio de aplicações.

9.2. Limitações

O Blendwork apresenta algumas limitações que são características da arquitetura cliente/servidor duas camadas e outras que advém da não automatização de algumas funcionalidades já implementadas. A seguir estas limitações serão apresentadas juntamente com possíveis formas de superá-las.

- Limitações provenientes da arquitetura: uma nova versão a cada alteração de negócio, podendo ser superado através da automatização da distribuição de novas versões, onde a própria aplicação verifica durante a inicialização a existência de uma versão mais atualizada. Pode-se também agrupar as classes de negócio em bibliotecas separadas do executável, facilitando o processo de atualização e redistribuição; outra limitação é a impossibilidade de, na versão atual do Blendwork, separar a lógica de negócio da camada de apresentação, pois são implementadas nos formulários – responsáveis também pela persistência dos dados. Isto resulta em dificuldades de integração e

comunicação de outros sistemas para com os desenvolvidos sobre o Blendwork. Uma solução seria as outras aplicações acessarem diretamente as informações no banco de dados, tendo como inconveniente a não validação das regras de negócio.

- Funcionalidades não automatizadas: a interação entre as classes de formulários implementada no Blendwork precisa ser configurada em tempo de programação, o que acaba gerando um esforço de desenvolvimento adicional. Uma maior flexibilidade pode ser alcançada através da parametrização destas interações em tempo de execução, armazenando-as no dicionário de dados, automatizando assim a utilização desta funcionalidade. Da mesma forma, a chamada dos formulários a partir da interface gráfica principal também poderia ser automatizada.

9.3. Trabalhos Futuros

Como trabalhos futuros, poderão ser cogitadas questões do Blendwork como:

- Separação da camada de apresentação da camada de lógica de negócio, possibilitando assim a adequação da atual arquitetura cliente/servidor de duas camadas para cliente/servidor três ou multi-camadas;
- Criação de uma camada de persistência, responsável pela comunicação com o banco de dados, flexibilizando a compatibilidade com bancos de dados de diferentes fabricantes;
- Ampliação do Dicionário de Dados para facilitar a parametrização das funcionalidades do framework, inclusive em tempo de execução;
- Incremento do ambiente de desenvolvimento visual para dar suporte a uma maior interação com o Blendwork, facilitando o desenvolvimento de aplicações e o aprendizado do framework;
- Incorporação de outras funcionalidades, como auditoria de acesso e de erros, mecanismos flexíveis e parametrizados para importação e exportação de dados em arquivos texto, dentre outras que possam vir a ser identificadas.

Referências Bibliográficas

(ALEKSY & KORTHAUS, 1999) ALEKSY, Markus; KORTHAUS, Axel. **Interoperability of Java-Based Applications and SAP's Business Framework – State of Art and Desirable Developments.** 1999, Mannheim: University of Mannheim. Disponível em: <http://citeseer.nj.nec.com/332233.html>. Acesso em: 20 de julho de 2003.

(AMBLER, 2000a) AMBLER, Scott W. **Mapping Objects To Relational Databases**. 2000. White-paper. Disponível em <http://www.ambysoft.com/mappingObjects.pdf>. Acesso em: 02 dez. 2002.

(AMBLER, 2000b) AMBLER, Scott W. White paper: **The Design of a Robust Persistence Layer For Relational Databases**. 2000. White paper. Disponível em: <http://www.ambysoft.com/persistenceLayer.pdf>. Acesso em: 04 jan. 2003.

(BLOM, 2000) BLOM, R.E. Survey of prototyping and RAD tools. 2000, Amsterdam: Vrije Universiteit Amsterdam. Disponível em: http://cs.vu.nl/~mmc/mci/content_pages/opdrachtvoorbeelden/R.BOOM/paper.html. Acesso em: 04 abr. 2003

(BOLDSOFT, 2002) Empresa: Boldsoft, **Products: Bold Architecture.** Disponível em: <http://www.boldsoft.com/products/bold/index.html>. Acesso em: 03 de maio de 2002.

(CAMBIOTIS, 2001) CAMBIOTIS, John. **An Introduction to Client/Server Architecture Goals and Specifications.** 2001. Disponível em <http://lancs.ac.uk/postgrad/cambioti/client-server/client-server-architecture.html>. Acesso em: 27 mar. 2003.

(CASEMAKER, 2000) CASEMaker Inc.. **What is Rapid Application Development**, White Paper, 2000. Disponível em: http://www.casemaker.com/download/products/totem/rad_wp.pdf Acesso em: 12 mar. 2003.

(DAVIS, 1998) DAVIS, Paul Beynon. **Rapid Application Development.** 1998. Disponível em: <http://comp.glam.ac.uk/SOC.Server/ research/ gisc/RADbrf.htm> Acesso em: 07 abr. 2003.

(DEUTSCH, 1989) DEUTSCH, L. Peter. **Design reuse and frameworks in the Smalltalk-80 programming system.** Addison Wesley, 1989. *apud* (FAYAD, 1999).

(EARLES, 2000a) EARLES, John. **Frameworks! Make room for another Silver Bullet.** 2000. Empresa: Castek. Disponível em: <http://www.cbd-hq.com/articles/2000/000301je_frameworks.asp>. Acesso em: 25 março 2002.

(EARLES, 2000b) EARLES, John. **Framework Evolution! One Box, Two Box, White Box, Black Box.** 2000. Empresa: Castek. Disponível em: <http://www.cbd-hq.com/articles/2000/000401je_frameworks2.asp>. Acesso em: 25 março 2002.

(FAYAD & SCHMIDT, 1997) FAYAD, Mohamed E.; SCHMIDT, Douglas C.; JOHNSON, Ralph E. **Object-Oriented Application Frameworks.** 1997. Disponível em: < http://www.cs.wustl.edu/~schmidt/CACM-frameworks.html>. Acesso em: 20 de março de 2002.

(FAYAD, 1999) FAYAD, Mohamed E.; SCHMIDT, Douglas C.; JOHNSON, Ralph E. **Building Application Frameworks: OO Foundations of Framework Design.** New York, NY: John Wiley and Sons, 1999.

(GAJIC, 2000) GAJIC, Zarko. Birth, Life and Death of a Form. **Delphi For Beginners.** 2000. Disponível em <http://delphi.about.com/library/weekly/aa060600a.htm>. Acesso em: 24 abr. 2002.

(GALLAUGHER, 1996) GALLAUGHER, John. **The Critical Choice of Client Server Architecture: A Comparison of Two and Three Tier Systems.**1996. Disponível em <http://www2.bc.edu/~gallaugh/ research/ism95/cccsa.html>. Acesso em: 27 mar. 2003.

(GAMMA, 1995) GAMMA, Erich; et al., 1995. Tradução de Luiz A. Meireles Salgado. **Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos.** Porto Alegre: Bookman, 2000.

(HALBACH & NEWTON, 1997) HALBACH, Daniel S.; NEWTON, Eric C. **A Practical Alternative to Business Objects – An Experience Report.** 1997. Disponível em: http://www.pareto.com/exprep97.html. Acesso em: 01 mai. 2002.

(IBM, 1997a) Empresa: IBM. **IBM SanFrancisco Technical Summary.** 1997. Disponível em: <http://www-3.ibm.com/software/ad/sanfrancisco/prd_summary.html>. Acesso em: 28 de abril de 2002.

(IBM, 1997b) Empresa: IBM. **IBM Business Framework: San Francisco Project – Technical Overview.** 1997, Rochester: IBM AS/400 Division. Disponível em: < http://www.research.ibm.com/journal/sj//363/forum.html>. Acesso em: 28 de junho de 2003.

(JENSEN, 1998) JENSEN, Cary. Delphi Database Development**. Delphi Informant**, v. 4, n. 8, agosto 1998. Disponível em: <http://www.delphizine.com/features/1998/08/di199808cj_f/di199808cj_f.asp>. Acesso em: 18 abr. 2002.

(JENSEN, 1999) JENSEN, Cary. Delphi Frames**. Delphi Informant,** v. 5, n. 12, dezembro 1999. Disponível em: <http://www.delphizine.com/features/1999/12/di199912cj_f/di199912cj_f.asp>. Acesso em: 19 de abril de 2002.

(JOHNSON & FOOTE, 1988) JOHNSON, Ralph E., FOOTE, Brian. Designing reusable classes. **Journal of Object-Oriented Programming**, v.1, p. 22-35, junho/julho 1988.

(JOHNSON, 1993) JOHNSON, Ralph E. **How to Design Frameworks**, Tutorial Notes, OOPSLA '93, Washington, 1993. Disponível em: www.cse.msu.edu/~cse870/Materials/Frameworks/how-to-design-fw-tutorial.ps. Acesso em: 28 ago. 2002.

(KETTEMBOROGH, 1997) Kettemborough, Cliff . **Rapid Application Development (RAD)**. Disponível em: http://newton.uor.edu/FacultyFolder/CKettemborough/WhPp/RAD.html. Acesso em: 12 mar. 2003.

(LEWANDOWSKI, 1998) LEWANDOSKI, Scott M.; **Frameworks for Component-Based Client/Server Computing**. ACM Computer Surveys, v. 30, n. 1, março 1998.

(LISCHNER, 2000) LISCHNER, Ray. **Delphi in a Nutshell: A Desktop Quick Reference**. 3ed. Sebastopol, CA: O'Reilly & Associates, 2000.

(MATTSSON, 1996) MATTSON, Michael. **Object-Oriented Frameworks – A survey of methodological issues**. 1996, 130p. Tese (Licenciatura) – Department of Computer Science, Lund University, Lund.

(MATTSSON, 2000) MATTSON, Michael. **Evolution and Composition of Object-Oriented Frameworks**. 2000, 216 f. Tese (Doutorado em Engenharia) – Department of Software Engineering and Computer Science, University of Karlskrona/Ronneby, Karlskrona.

(MORRIS, 1996) MORRIS, Richard A. Delphi: Type Safe Programming. **7th Annual Borland Developers Conference**, 1996. Disponível em: <http://www.khiron.com/rtti.htm>. Acesso em: 20 abr. 2002.

(PREE, 1995) PREE, Wolfgang. **Design Patterns for Object-Oriented Software Development.** Reading, MA: Addison-Wesley, 1995. *apud* (FAYAD, 1999).

(ROBERTS & JOHNSON, 1996) ROBERTS, Don; JOHNSON, Ralph. **Evolving Frameworks. A Pattern Language for Developing Objetc-Oriented Frameworks.** Procedings of the 3rd International Conference on Pattern Languages for Programming, Monticelli, IL, USA, September 1996. Disponível em: http://st-www.cs.uiuc.edu/users/droberts/evolve.html. Acesso em: 28 ago. 2002.

(SADOSKI, 1997) SADOSKI, Darleen. **Client/Server Software Architectures – An Overview.** 1997. Disponível em: http://www.sei.cmu.edu/str/descriptions/clientserver_body.html. Acesso em: 23 abr. 2002.

(SAP, 1997) Empresa: SAP, **R/3 System SAP Business Objects**, Disponível em: <http://www.biz.uiowa.edu/class/6k180_park/SAP.pdf> Acesso em: 02 de maio de 2002.

(SAP, 2000) Empresa: SAP, **mySAP Technology**, Key Capabilities: Internet-Business Framework Disponível em: http://www.sap.com/solutions/technology/keycapabilities/ibf.asp Acesso em: 02 de maio de 2002.

(SCHMIDT, 1997) SCHMIDT, Douglas C. **Applying Design Patterns and Frameworks to Develop Object-Oriented Communication Software**. New York: Macmillan Computer Publishing, 1997. *apud* (FAYAD, 1999).

(SILVA & PRICE, 1998) SILVA, Ricardo Pereira e.; PRICE, Roberto Tom. **A busca de generalidade, flexibilidade, e extensibilidade no processo de desenvolvimento de frameworks orientados a objetos.** Proceedings of Workshop Iberoamericano de Engenharia de Requisitos e Ambientes de Software (IDEAS'98). Torres: apr. 1998. v.2, p.298-309.

(SILVA & PRICE, 1999) SILVA, Ricardo Pereira e.; PRICE, Roberto Tom. **Suporte ao desenvolvimento e uso de componentes flexíveis.** Proceedings of XIII Simpósio Brasileiro de Engenharia de Software. Florianópolis: out. 1999. p.13-28.

(SILVA, 2000) SILVA, Ricardo Pereira e. **Suporte ao Desenvolvimento e Uso de Frameworks e Componentes**. 2000. 262 f. Tese (Doutorado em Ciência da Computação) - Programa de Pós-graduação em Ciência da Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre.

(SPENCE, 2002a) SPENCE, Rick. Visual Form Inheritance: Part I. **Delphi Informant.** v. 6, n. 2, fevereiro 2000. Disponível em: <http://www.delphizine.com/features/2000/02/di200002rs_f/di200002rs_f.asp>. Acesso em: 18 abr. 2002.

(SPENCE, 2002b) SPENCE, Rick. Visual Form Inheritance: Part II. **Delphi Informant.** v. 6, n. 3, março 2000. Disponível em: < http://www.delphizine.com/features/2000/03/di200003rs_f/di200003rs_f.asp>. Acesso em: 18 abr. 2002.

(STEELE, 2000) STEELE SCHARBACH ASSOCIATES LLC. **Client-Server Architecture: Bringing Order to the Bamble Bush.** White paper. 2000. Disponível em <http://www.ssa-lawtech.com/wp/wp3-5.htm>. Acesso em: 27 mar. 2003.

(TALIGENT, 1994) **Building Object-Oriented Frameworks**. White-paper: Taligent Inc, 1994. Disponível em: http://www-106.ibm.com/developerworks/java/library/j-oobuilding/ Acesso em: 11 jul. 2002.

(TALIGENT, 1995) **Leveraging Object-Oriented Frameworks**. White-paper: Taligent Inc, 1995. Disponível em: http://www-106.ibm.com/developerworks/java/library/j-ooleveraging/ Acesso em: 11 jul. 2002.

(TODD, 1999) TODD, Bill. RTTI Gets Easier. **Delphi Informant**, v. 5, n. 11, novembro 1999. Disponível em: <http://www.delphizine .com/ features/1999/11/di199911bt_f/di199911bt_f.asp>. Acesso em: 18 abr. 2002.

(WOOD, 2000) WOOD, Keith. Database Persistent Objects: Part I. **Delphi Informant**, v. 4, n. 9, setembro de 2000. Disponível em: < http://www.delphizine.com/features/2000/09/di200009kw_f/di200009kw_f.asp>. Acesso em: 19 abr. 2002.