UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Marcelo Luís Theobald

PRODAGRO: Framework Orientado a Objetos para a Gestão e Comercialização de Produtos Agropecuários

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

Ricardo Pereira e Silva
Orientador

Florianópolis, setembro de 2003

Agradecimentos

Agradecer, creio ser umas formas mais legais de demonstrar o carinho, gratidão e respeito à aqueles que contribuíram para o êxito da conquista. No meu caso, não foram poucos os colaboradores/participantes, tanto diretos quanto indiretos, mas com certeza, todos eles vitais para alcançar esse feito tão grandioso. Por isso, cada um, indistintamente, pode considerar-se parte ativa da conquista.

Primeiramente, eu gostaria de agradecer aos meus pais, Inácio e Maria, que além de rezarem por mim, são os responsáveis pela minha existência. Eles, que durante a minha faculdade sofreram muito, haja visto que a situação financeira sempre fora difícil. E, sem isso, o Mestrado não seria possível. Valeu Pai! Valeu Mãe! Muito Obrigado!

De igual forma, queria agradecer aos meus irmãos, são seis(6), todos eles sempre me incentivando e motivando para o alcance da conquista. De modo especial, ao mano Elemar, que foi o responsável pelo meu ingresso na área de Informática, indicando o caminho para a Universidade. Sem o apoio dele, provavelmente eu teria feito outro curso. Igualmente, minha gratidão ao irmão Alceste, a cunhada Márcia e o sobrinho Bruno, que por inúmeras vezes, reservaram parte do seu tempo para cuidar do meu filho Cristian, possibilitando-me dar seqüência nos estudos/pesquisas. Valeu estimados manos, Obrigado!

À minha querida e amada companheira e esposa Neidi e meu filho Cristian, vocês que me acompanharam durante todo o período de Mestrado, o meu Muito Obrigado de coração. Você Neidi, sabe o quanto lutamos para chegar onde cheguei. Sei o quanto você se esforçou para trocar horários no seu serviço, trabalhar noites seguidas, enfim mover forças para que eu conseguisse trilhar o caminho com pleno sucesso. Você Cristian, por muitas vezes serviu como um anti-stress, pois após uma rotina diária de 8 horas, era um prazer poder chegar em casa e brincar com você. Valeu, Obrigado!

Agradeço ao meu orientador Prof. Dr. Ricardo Pereira e Silva, uma pessoa muito especial, pelo seu conhecimento, sinceridade e comprometimento com a pesquisa.

Obrigado por me proporcionar esses anos de convivência, possibilitando o enriquecimento pessoal e profissional.

Aos colegas de luta e agora também Mestres, Renato Rockemback e Dário Beutler, o meu Muito Obrigado pela parceria nas horas difíceis, pelas horas de estudo compartilhadas, seja à noite, feriados, sábados e muitas e muitas vezes aos domingos. Visto que, todos tínhamos outros afazeres além do término do Mestrado. Valeu gente boa!

Ao meu tio Orides e tia Margarete, que igualmente foram pessoas motivadoras durante esse ciclo do Mestrado, dando apoio e sabendo transmitir energias para continuar o trabalho.

A minha cunhada Natanieli, mas conhecida por Pati, que também não mediu esforços na primeira etapa do Mestrado, auxiliando nos cuidados do filho Cristian, permitindo assim que eu pudesse estudar e pesquisar. Muito Obrigado!

Às empresas Camera Agroalimentos e Coopermil, nas pessoas dos Senhores Roberto Kist e Jaime Stochero, respectivamente, um Muito Obrigado em especial, pois várias vezes precisei de uma folga para poder viajar ou pesquisar e, prontamente fui atendido. Sei que vocês também ganharam com isso, mas de qualquer forma fica registrada a minha gratidão.

Aos amigos e companheiros, indistintamente, meu agradecimento. Sei que vocês às vezes não gostavam quando eu dizia não para uma cerveja, um jogo de futebol, enfim um lazer de final-de-semana. Mas, hoje vocês entendem muito bem isso. Obrigado pela compreensão!

Agradeço também a Deus, que sempre esteve ao meu lado. Servindo como um guia, uma referência para não desistir da luta nunca.

RESUMO

Atualmente, com o crescente aumento da complexidade dos sistemas computacionais, a abordagem de frameworks orientado a objetos tem-se mostrado cada vez mais importante, visto que a mesma propicia principalmente o reuso de software. Isso permite o desenvolvimento das aplicações com menor custo e tempo. No presente trabalho é apresentado um framework denominado **PRODAGRO**, para o domínio de gestão e comercialização de produtos agropecuários. O propósito do mesmo é servir como suporte aos desenvolvedores de aplicações desse domínio. Seu desenvolvimento foi motivado pela carência de ferramentas que auxiliam o desenvolvimento destas aplicações. Através do **PRODAGRO**, é possível desenvolver as aplicações com maior produtividade, visto que o mesmo contempla algumas características importantes, como o reuso de projeto e de código. Conseqüentemente, em função do reuso de software promovido pelo framework, é possível desenvolver aplicações do ramo agropecuário com mais rapidez, por não ser necessário desenvolver toda a aplicação. Com isso, os usuários das aplicações também podem ser atendidos com maior rapidez e agilidade. Cabe salientar que software agropecuário, é uma área muito pouco explorada pelos pesquisadores de Engenharia de Software e, portanto, um campo propício para pesquisas. O trabalho desenvolvido na presente dissertação, além do framework, apresenta duas aplicações desenvolvidas com o mesmo, descritas no decorrer da dissertação, com o intuito de avaliar o framework.

Palavras-chave: framework orientado a objetos, reuso, orientação a objetos, aplicação agropecuária.

ABSTRACT

Nowadays, due to the increase in the complexity of computing systems, object-oriented framework technology has become more and more important, since it allows for the reuse of software which causes a decrease in time and in cost of the development of the applications. This study presents a framework named PRODAGRO, which is used for managing and marketing agricultural products, and whose purpose is to give support to developers of this kind of computer applications.The development of PRODAGRO was motivated by the lock of tools that facilitate their development. With the use of PRODAGRO, it is possible to develop applications with more productivity, since it has some important characteristics such as the reuse of project and code. Consequently, due to the reuse of software provided by the framework, it is possible to develop applications for agribusiness faster as it is not necessary to develop the whole application. As a consequence, users of these applications can be supplied to with more agility, satisfying their needs as the developers can develop them faster. It is important to emphasize that this area is not much explored by researchers of software engineering and, therefore, it is an attractive area for research. The work presented in this dissertation provides the framework, as well as two examples of applications developed with it, aiming the assessment of the framework.

Key-words: object-oriented framework, reuse, object-oriented, agricultural application.

Sumário

LISTA DE ABREVIATURAS

- ACE – Adaptive Communication Environment
- AWT - Abstract Window Toolkit
- CNPTIA - Centro Nacional de Pesquisa Tecnológica em Informática para a Agricultura
- CORBA - Common Object Request Broker Architecture
- DCOM - Distributed Component Object Model
- DI – Departamento de Informática
- EDA – Estruturas de Dados Abstratos
- EMBRAPA - Empresa Brasileira de Pesquisa Agropecuária
- ESP - Soluções Embrapa para a Agroinformática
- GRN - Gestão de Recursos de Negócios
- GUI - Graphical User Interface
- IBM – Information Business Management
- LAMA - Library Administration and Management Association
- LES - Laboratório de Engenharia de Software
- MFC - Microsoft Foundation Classes
- MVC - Model-View-Controller
- ODMG – Object Data Management Group
- OMT - Object Modeling Technique
- OO - Orientação a Objetos
- OOAD – Object Oriented Analysis and Design
- OOHDM – Object Oriented Hypermedia Design Model
- OOPSLA – Object Oriented Programming Systems Languages and Applications
- OOSE - Object-Oriented Software Engineering
- ORB - Object Request Broker
- PUC – Universidade Pontífice do Rio de Janeiro

- RMI - Remote Method Invocation
- TDA - Tipos de Dados Abstratos
- TI – Tecnologia da Informação
- UML – Unified Modeling Language
- UNICAMP - Universidade de Campinas (SP)
- WWW - World Wide Web

LISTA DE FIGURAS

1 INTRODUÇÃO

São inúmeras as forças que impõem novas concepções e valores à sociedade e às empresas, sendo a mais importante delas a rapidez nas mudanças exigidas. As novas tecnologias têm promovido mudanças e adaptações das empresas em ritmo sem precedentes. Elas têm exercido forte impacto até mesmo sobre as estruturas empresariais mais conservadoras, cujas estratégias e regras de gerenciamento e administração foram desenvolvidas e organizadas para atender mercados e tecnologias estáveis, que se modificam gradualmente, em ritmo bastante lento. Os sistemas de informação vêm alterando a natureza da administração e afetando a direção e a cadência das mudanças.

A principal fonte provedora desta mudança tem sido o intenso emprego do recurso de informação associado a tecnologias facilitadoras da coleta, processamento, armazenamento e disseminação, conhecidas como Tecnologias da Informação (TI). Cada vez mais as empresas têm recorrido a estas tecnologias para a adequação de seus empreendimentos ao novo formato de operação exigido pelo mercado nacional e internacional.

O tema deste trabalho situa-se na área de Engenharia de Software, onde o desenvolvimento de framework orientado a objetos está sendo estudado e difundido entre os pesquisadores, analistas, projetistas e desenvolvedores da área. Pesquisas sobre frameworks têm sido amplamente conduzidas (PREE, 1999; ROBERTS & JOHNSON, 1998; SCHMID, 2002) com o intuito de descobrir técnicas mais eficientes não somente para construí-los, mas também para torná-los mais fáceis de utilizar e manter por desenvolvedores não familiarizados com sua estrutura.

Embora os primeiros frameworks tenham surgido para domínios mais amplos - entre eles, interface com o usuário - existe uma preocupação cada vez maior em produzir frameworks para domínios de aplicações específicas, como negócios, engenharia, medicina, seguros, etc. Em particular, o ambiente de negócios está se tornando mais dinâmico com o aumento da competição mundial e mudanças no mercado, demandando novas tecnologias a

fim de diminuir o esforço necessário para o desenvolvimento de aplicações com maior rapidez e eficiência.

1.1 Contextualização

O presente trabalho tem como contexto o desenvolvimento de software para a gestão e comercialização de produtos agropecuários, em que se constata a falta de ferramentas que possibilitem o desenvolvimento das aplicações do domínio com maior produtividade, uniformidade e rapidez. Framework Orientado a Objetos se mostra uma abordagem adequada para suportar o desenvolvimento de software deste domínio de aplicações, visto que possibilita aglutinar as características comuns e especificidades das aplicações e, por conseguinte, servir de suporte reusável para os desenvolvedores de aplicações agropecuárias tornarem seus softwares mais manuteníveis, em um processo da mais alta produtividade em função do reuso possibilitado e, conseqüentemente, com menor custo.

Hoje, faz-se necessário que o administrador de uma fazenda, propriedade ou empresa agropecuária tome suas decisões amparado em informações quantificadas e agregadas sobre o seu processo produtivo. Além disso, as informações sobre sua produção devem ser atualizadas e confiáveis de modo que possam ser inseridas nos sistemas de planejamento de seu ramo de atividade. O mercado competitivo exige tanto dos produtores rurais quanto dos dirigentes de cooperativas, mais profissionalismo na condução dos seus negócios, o que implica na obtenção de informações mais precisas, organização e utilização eficiente. Embora a aplicação da informática seja ainda bastante incipiente no meio rural, principalmente quando comparada a outros setores, não se pode negar a existência de um bom número de softwares específicos no mercado nacional. Corroborando essa afirmativa, Vale & Santos relatam a existência de 146 títulos de programas, destinados exclusivamente ao setor agropecuário (AGROSOFT, 1997).

Desenvolver software não é tarefa das mais fáceis. Um dos fatores que gera esta dificuldade é que muitas vezes o entendimento do problema não está muito claro. Além disso, há uma escassez grande na documentação dos problemas e de possíveis soluções para

os mesmos. Com isso, problemas que muitas vezes se repetem, geram esforços adicionais para a implementação de suas soluções. As tentativas de reuso das boas soluções e do acúmulo de experiências sobre determinados problemas são, na maioria das vezes, iniciativas isoladas de alguns desenvolvedores (JUNQUEIRA, 1998).

Uma das características tida como das mais importantes da Orientação a Objetos (OO) é a de promover e aumentar a reusabilidade de software. Reutilização é um ponto chave para produtividade, sendo também um dos atributos considerados na avaliação de qualidade de software (PREE, 1995). Na prática, a criação de software através da reunião e composição de artefatos existentes não é muito fácil de ser obtida. Mesmo antes da orientação de objetos, algumas formas de reutilização já existiam em desenvolvimento de software. A organização de funções e procedimentos em bibliotecas é uma das formas mais primitivas, que antecede a reutilização das chamadas Estruturas de Dados Abstratos (EDA) e Tipos de Dados Abstratos (TDA) (PREE, 1995; MATTSON, 1996). Essas formas de reutilização favoreciam principalmente a reutilização de código. Outras tecnologias de reutilização surgiram juntamente e, de certo modo paralelamente com a OO, procurando contemplar a reutilização de outros elementos envolvidos na concepção de software.

Simplesmente projetar software orientado a objetos não significa que ele vai ser potencialmente reutilizável e flexível. Diversas técnicas, heurísticas, princípios de bom projeto, devem ser aplicados para criar software reutilizável e flexível (JOHNSON, 1988; RIEL, 1996; GAMMA, 1994; MEYER, 1997). Como Johnson (1988) argumenta, "*reutilização de software não acontece por acidente*", o que é complementado por Gamma (1994), afirmando que "*a criação de software orientado a objeto é difícil, mas a criação de software orientado a objeto reutilizável é mais difícil ainda*". É sabido que só o que é bem concebido e modelado levando em conta generalidade e flexibilidade, está apto a ser reutilizável.

O domínio em questão - software para produtos agropecuários - atua em dois campos distintos da agricultura, um envolvendo produtos agrícolas e outro, produtos pecuários, bem como nos produtos derivados dos mesmos. Nos produtos agrícolas, é possível produzir

aplicações envolvendo diversos subtipos como oleaginosas, cereais, fruticultura, olericultura, etc. Já nos produtos pecuários, trabalha-se basicamente com três finalidades, que seguem: pecuária de corte, leite e reprodução. Por fim, os diferentes tipos de derivados, entre eles: farelo, farinha, óleo, carne, ovos, etc. Portanto, a gama de aplicações que podem ser desenvolvidas é muito ampla, sendo assim, faz-se necessário então um levantamento das peculiaridades existentes em cada um dos ramos de atuação, para com isso modelar uma ferramenta realmente capaz de atender às necessidades e aos anseios dos desenvolvedores das aplicações.

Existem no mercado nacional softwares para o gerenciamento de rebanhos. Todavia, apesar de eventualmente eficientes na organização da informação para criadores, eles não se caracterizam com o sistema de informação - software baseado nos processos reais, que consideram o relacionamento entre setores/atividades/estágios de produção, e, neste sentido, realizam a consistência dos dados coletados entre si e com aqueles previamente existentes, ou seja, numa avaliação estruturada conforme padrões de conformidade preestabelecidos com base em critérios biológicos, cronológicos e técnico-operacionais ou normativos. Assim, no âmbito das associações de criadores, constata-se a necessidade de se aplicar os recursos da tecnologia da informação no armazenamento dos registros de desempenho produtivo, reprodutivo, de conformação e genealogia dos animais, para estruturação de um banco de dados com informações sobre animais, e prover um mecanismo para a transferência dessas informações entre as instituições participantes de programas de desenvolvimento da pecuária. Esta avaliação, de grande auxílio na tomada de decisões, possibilita a elaboração de estratégias de ajuste do manejo dos rebanhos, visando obter ganhos em produtividade.

De certa forma, os produtores rurais, cooperativas agropecuárias e associações de criadores que não fizerem uso da informática, seja por atualização dos recursos de hardware ou desenvolvimento de sistemas de seu interesse, estarão desprovidas de um recurso importante no contexto atual de mudanças, em que a falta de informação implica atraso tecnológico. O poder de mercado, por exemplo, tem se deslocado dos processadores para as

redes de distribuição, resultante do acesso rápido e direto às informações quanto aos hábitos de consumo e coordenação dos fluxos de bens (AGROSOFT, 2002).

Os agricultores internautas são seres desconhecidos para o governo, mas as multinacionais do setor agrícola, fabricantes de máquinas, sementes, defensivos, fertilizantes e outros artigos agrícolas encomendaram recentemente ao instituto alemão Kleffmann[1], especializado em estudos do mercado agrícola, um perfil do produtor brasileiro de soja. O levantamento mostra que dos 1.400 produtores de soja pesquisados, 32% têm microcomputador e quase 19% (261) conexão à internet. Destes, 16% já compraram produtos pela rede e o principal uso da WEB é para a busca de informações climáticas, estatísticas, técnicas e sobre produtos agrícolas. Entre os produtores de soja conectados, apenas 5% são considerados pequenos, isto é, cultivam até 100 hectares de terra. A situação é crítica porque justamente quem mais se beneficiaria das informações são os pequenos produtores, que não dispõem de outras fontes de informação (FRANCISCO, 2002).

1.2 Um pouco da história do reuso de software

Inicialmente, o reuso consistia apenas em linhas de código copiadas de um outro programa, o que ocasionava a criação de programas extensos e mal escritos. Em seguida, pedaços de código reutilizados passaram a localizar-se em subrotinas. Depois, funções inteiras genéricas, relacionadas e elaboradas para servir a várias situações em programas diferentes, chamadas bibliotecas, começaram a ser utilizadas como uma boa forma de reutilizar código.

A partir dos anos 80, a revolução OO deu novos ares à reusabilidade, permitindo reusar conceitos inteiros através de classes, utilizando técnicas como herança e composição, assim como, permitiu reutilizar contratos de relacionamento entre os objetos, através do uso de interfaces, que possibilitou o surgimento do polimorfismo.

[1] Líder em pesquisas agropecuárias.

Porém, com todas estas técnicas, a arte de programar ainda consistia uma tarefa muito árdua, que dependia muito do programador. Isto é, apenas excelentes programadores conseguiam construir programas realmente reutilizáveis.

Na década de 90, técnicas foram criadas e catalogadas para resolver estes problemas. Elas consistiam de boas idéias de projeto obtidas pelos melhores programadores do mundo ao longo de suas vidas. Intituladas como Padrões de Projeto (Design Patterns), consistiam de micro-arquiteturas que definiam as colaborações entre classes e objetos. Desta forma, a granularidade do reuso foi aumentada de classes individuais ou hierarquia de classes para várias classes e suas colaborações. Por outro ângulo, passou-se a reutilizar idéias.

No final da década de 90, frameworks orientado a objetos foram criados para aumentar ainda mais esta granularidade. Framework consiste de uma solução comum para uma família de problemas semelhantes. Construir um framework implica em implementar e testar várias aplicações, fatorar o que elas têm em comum em um único código e deixar os ganchos para que o programador da aplicação adicione as diferenças relativas a sua aplicação. Desta forma, passou-se a reusar análise, arquitetura, design inteiro, semântica de interação e até testes (SAUVÉ, 2003).

1.3 Objetivo

Este estudo tem por objetivo principal disponibilizar aos desenvolvedores de aplicações agropecuárias, uma arquitetura estrutural que permita reuso de projeto e código. Isso será possível mediante a construção de uma ferramenta denominada **PRODAGRO**, isto é, *framework para a gestão e comercialização de produtos agropecuári*os. Com isso, os desenvolvedores desse tipo de aplicação poderão utilizar o mesmo para produzir suas aplicações finais num tempo possivelmente menor do que se as implementassem sem reuso de sotware. O objetivo estabelecido foi alcançado a partir do desenvolvimento de duas aplicações usando o framework, em que se verificou a capacidade da arquitetura desenvolvida de promover reuso para diferentes contextos dentro do domínio tratado.

Um segundo objetivo é a interação do autor desta com os conceitos de modelagem OO, Frameworks, Padrões de Projeto, Programação Orientada a Objetos, Linguagens Orientadas a Objetos, mais especificamente JAVA, entre outros. Enfim, familiarizar-se com essas tecnologias, pois despertam bastante curiosidade e interesse do mesmo.

1.4 Motivação

A motivação para desenvolver a presente pesquisa deve-se principalmente pela inexistência de ferramentas que auxiliem os desenvolvedores de softwares agropecuários a desenvolver suas aplicações com maior rapidez, agilidade, flexibilidade e portabilidade. O desenvolvimento de um framework, além das características recém citadas, pode facilitar outras questões, como manutenibilidade da aplicação desenvolvida, produtividade, uniformidade na estrutura do software, redução da complexidade do desenvolvimento, maior consistência e integração entre aplicações e aproveitamento da experiência de especialistas do domínio. Mas, para isso, o mesmo deve ser bem projetado e codificado. Outra motivação presente é a de que essa área ainda é muito pouco explorada pelos pesquisadores e engenheiros de software. Sabe-se que a área agropecuária pode gerar bastante produção, necessitando com isso, softwares capazes de auxiliar no gerenciamento e controle das informações ou das necessidades do cliente. Cabe salientar também a familiaridade do autor desta dissertação com relação ao tema, pois o mesmo trabalha com desenvolvimento de software agrícola.

1.5 Organização da Dissertação

A presente dissertação é constituída por seis capítulos, cujos conteúdos estão descritos a seguir:

O capítulo 2 apresenta os conceitos e características do desenvolvimento de software baseado em frameworks orientados a objetos. Faz-se necessário esse estudo, essa aquisição de conhecimentos e experiência, tendo em vista de que servirão como base para o entendimento dos objetivos e a posterior realização do trabalho proposto nesta dissertação. Também apresenta conceitos e características do desenvolvimento de software baseado em

padrões de projeto. Enfim, visa discriminar claramente os conceitos de design patterns e framework orientado a objetos, possibilitando assim um maior conhecimento dessas tecnologias e, posteriormente, utilizar as mesmas para o desenvolvimento de aplicações.

O capítulo 3 apresenta a caracterização de produtos agropecuários e uma descrição das principais características existentes no domínio de aplicações para gestão e comercialização de produtos agropecuários. Características estas que devem ser abrangidas pelo framework a ser desenvolvido.

O capítulo 4 relata as características que envolvem o projeto do desenvolvimento do framework de gestão e comercialização de aplicações agropecuárias. Descreve todo o projeto e implementação do framework PRODAGRO.

O capítulo 5 apresenta a experiência do autor com o desenvolvimento de duas aplicações usando o framework produzido. Enfim, relata basicamente aquilo que foi necessário desenvolver ou usar para chegar ao objetivo previamente definido, que é prover reuso de projeto e código através de uma ferramenta, que é o framework PRODAGRO.

O capítulo 6 trata da conclusão da presente dissertação, mencionando que o objetivo fora atingido e a importância do mesmo no cenário de softwares agropecuários. Além disso, menciona as vantagens em produzir um framework, as limitações existentes no framework produzido, bem como alguns trabalhos futuros que possam vir a serem desenvolvidos por algum pesquisador.

2 FRAMEWORKS ORIENTADOS A OBJETOS

Um *framework* pode ser considerado como "*um projeto reutilizável de uma parte ou de todo um sistema, que é representado por um conjunto de classes abstratas e pelo modo que elas interagem*" (JOHNSON, 1997). Outra definição é "*uma aplicação reutilizável, incompleta, que pode ser especializada para produzir aplicações específicas*" (FAYAD & SCHMIDT, 1997). Também define-se que *framework* é "*um conjunto de classes que definem um projeto abstrato de soluções para uma família de problemas relacionados*" (JOHNSON, 1988).

Pode-se dizer que um *framework* é um conjunto de classes (abstratas e concretas) que representa genericamente um (sub)sistema de software, cujas classes podem ser adaptadas e estendidas para representar um (sub)sistema específico. *Frameworks* são abstrações de (sub)sistemas de software.

Frameworks orientados a objetos são uma tecnologia promissora para a instanciação de designs de software bem elaborados com vistas à redução de custos e melhoria da qualidade do software. Um *framework* é uma aplicação reutilizável e semi-completa que pode ser especializada para a produção de aplicações customizadas (LUCENA & MILIDIU, 1998).

Frameworks são desenvolvidos focando principalmente o aspecto de *reutilização de softwar*e. Desenvolver software pensando em reutilização leva a duas diferentes disciplinas, que são desenvolvimento para reutilização e desenvolvimento com reutilização, algumas vezes chamadas de *Application Engineering* (Engenharia de Aplicação) e *Application Development* (Desenvolvimento de Aplicação) (MATTSON, 1996). Os papéis executados por desenvolvedores nestas disciplinas normalmente são nomeados como *abstractor* e *elaborato*r, respectivamente (BECK & JOHNSON, 1994). Em resumo, desenvolvedores *abstractors* criam *frameworks* genéricos e reutilizáveis que serão usados e adaptados por *elaborators* para contextos específicos.

A tecnologia de *frameworks* orientados a objetos (JOHNSON, 1988; GAMMA, 1994; FAYAD & SCHMIDT, 1997; PREE, 1995; JOHNSON, 1997) endereça reutilização em maior escala, preocupando-se principalmente com a reutilização de projeto de software, o qual envolve muito mais empirismo e criatividade por parte do desenvolvedor, quando comparado com a tarefa de implementação. *Frameworks* podem ser definidos como "*um projeto reutilizável de uma parte ou de todo um sistema, que é representado por um conjunto de classes abstratas e pelo modo que elas interagem*" (JOHNSON, 1997). Eles definem os aspectos fixos de um (sub)sistema genérico de software, que são as responsabilidades comuns das classes e as colaborações básicas existentes, e deixam em aberto aspectos variáveis que devem ser determinados no processo de adaptação de um *framework* para representar um (sub)sistema concreto.

2.1 Introdução

Frameworks diferem de bibliotecas de classes porque apresentam uma inversão do fluxo de controle da aplicação, em relação ao reuso convencional de classes. Projetos baseados em bibliotecas de classes tradicionalmente forçam o desenvolvedor a definir todo o fluxo de controle da aplicação. Ao contrário, projetos baseados em frameworks colocam no framework o controle do fluxo, onde o desenvolvedor escreve apenas métodos especiais que adaptam funcionalidades previstas no framework, os quais são chamados pelo próprio framework (GAMMA, 1994; JOHNSON, 1988). Por esta razão, geralmente é dito que frameworks obedecem o "Princípio de Hollywood" ("Don't call us, we call you.") (APPLETON, 1997).

Após a popularização da linguagem Smalltalk, no início da década de 80, paralelamente às bibliotecas de classes, começaram a ser construídos frameworks de aplicação, que acrescentam às bibliotecas de classes os relacionamentos e interação entre as diversas classes que compõem o framework. Com os frameworks, reutilizam-se não somente as linhas de código, como também o projeto abstrato envolvendo o domínio de aplicação. *Framework* é definido por Coad & Yourdon (1992) como um esqueleto de classes, objetos e relacionamentos agrupados para construir aplicações específicas e por

Johnson (1988) como um conjunto de classes abstratas e concretas que fornecem uma infra-estrutura genérica de soluções para um conjunto de problemas. Essas classes podem fazer parte de uma biblioteca de classes ou podem ser específicas da aplicação. Frameworks possibilitam reutilizar não só componentes isolados, como também toda a arquitetura de um domínio específico.

Diversos *frameworks* têm sido desenvolvidos nas duas últimas décadas, visando o reuso de software e conseqüentemente a melhoria da produtividade, qualidade e manutenibilidade. Neste capítulo são definidos os *frameworks* bem como diversas abordagens para entender sua finalidade. A seguir são dados exemplos de alguns frameworks existentes e o framework *Model-View-Controller* é visto em maiores detalhes.

Um dos primeiros *frameworks* que pode ser encontrado na literatura é o *Model-View-Controller (MVC*), disponível em ambientes de desenvolvimento *Smalltal*k, para o desenvolvimento de interfaces para o usuário (GUI). Outros *frameworks* famosos endereçando o domínio de GUI que podem ser citados são o MacApp, para aplicações gráficas no ambiente *Macintos*h, ET++, *Interview*s, *Microsoft Foundation Classe*s, *Java AW*T, etc. (LEWIS, 1995). Apesar dos *frameworks* pioneiros endereçarem principalmente criação de interfaces gráficas, atualmente existem *frameworks* para diversos domínios, como editores diagramáticos *Hotdraw* (JOHNSON, 1992), *Unidraw* (LEWIS, 1995), sistemas operacionais *(Choice*s), engenharia financeira, objetos distribuídos e comunicação CORBA e Java RMI.

2.2 Conceitos e terminologia

Um “*Framework*” é o projeto de um conjunto de objetos que colaboram entre si para execução de um conjunto de responsabilidades. Um framework promove o reuso de projeto e código. Ele promove o reuso de projeto porque descreve os tipos de objetos importantes e como um problema maior pode ser dividido em problemas menores, contém algoritmos abstratos e descreve a interface que o programador deve implementar e as restrições a serem satisfeitas pela implementação. Ele promove o reuso de código porque todo o

framework é implementado. Portanto, quando estamos usando o mesmo, está sendo reusado seu código. Apesar de todos os tipos de reuso serem importantes, o reuso de projeto é mais relevante (JOHNSON, 1992).

2.3 Classificação de frameworks quanto a forma

Um *framework* define os seus *aspectos fixos* ou estáveis, que correspondem aos aspectos comuns a todos os contextos em que o *framework* será reutilizado, e deixa em aberto os seus *aspectos variáveis*, específicos de contexto de utilização do *framework*, sobre os quais os aspectos fixos podem estar baseados. Os aspectos fixos são definidos nas classes de um *framework* como pontos de estabilidade, enquanto aspectos variáveis são previstos nas classes como pontos de variabilidade (PREE, 1995). Um desenvolvedor reutiliza um *framework* adaptando-o ao contexto específico em que será usado, através do preenchimento dos pontos de variabilidade que foram previstos e adicionando novas funcionalidades.

Os pontos de estabilidade e variabilidade de um *framework* são representados nas classes que o compõem na forma de métodos (ou operações) chamados respectivamente de métodos *template* e *hook* (PREE, 1995). Métodos *template* implementam os pontos de estabilidade, e normalmente são baseados em métodos *hook*, que representam os pontos de variabilidade, os quais devem ser implementados no processo de adaptação e extensão das classes do *framework* para um uso específico.

A identificação de métodos *template* e *hook* é a separação e o encapsulamento de aspectos fixos e variáveis de um *framework*, que variam e evoluem independentemente. Pode-se dizer, que a identificação e definição dos pontos de estabilidade e variabilidade em *frameworks* é bastante importante, principalmente para criar *frameworks* flexíveis, evitando tarefas de reprojeto e reimplementação em cada situação em que forem detectados novos usos ou novas possibilidades de reutilização de um *framework*.

2.3.1 Caixa Branca (White-Box)

No *framework caixa branca* o reuso é provido por herança, ou seja, o usuário deve criar subclasses das classes abstratas contidas no *framework* para criar aplicações específicas. Pode-se dizer que o uso de herança faz com que aspectos internos das superclasses sejam expostos às suas subclasses, pois "herança quebra encapsulamento" (JOHNSON, 1988). Para tal, ele deve entender detalhes de como o *framework* funciona para poder usá-lo. Os métodos *hook* são representados em classes abstratas por operações abstratas ou com uma implementação padrão, as quais devem ser sobrescritas em subclasses concretas. Preencher um ponto de variabilidade na abordagem *white-box* significa sobrescrever e implementar um método *hoo*k.

2.3.2 Caixa Preta (Black-Box)

Já no *framework caixa preta* o reuso é por composição, ou seja, o usuário combina diversas classes concretas existentes no *framework* para obter a aplicação desejada. Assim, ele deve entender apenas a interface para poder usá-lo. Portanto, numa abordagem *black-bo*x, classes são adaptadas pela composição de objetos que estão de acordo com uma interface pré-definida para os pontos de variabilidade. As operações definidas nesta interface são os métodos *hook* das classes a serem adaptadas, variando as implementações das operações de acordo com o objeto que está sendo composto. Em outras palavras, neste contexto preencher um ponto de variabilidade significa compor um objeto que respeita a interface de adaptação que implementa os métodos *hook* correspondentes.

2.3.3 Caixa Cinza (Gray-Box)

Frameworks caixa-cinza já tem partes prontas e permite que se insiram novas funcionalidades a partir da criação de classes. Este tipo de *framework* fornece subclasses concretas e permite a criação de novas subclasses concretas. O *framework* OCEAN possui estas características e, portanto, é classificado como *caixa-cinza* (SILVA, 2000).

Através da utilização do *framework caixa-cinza*, será possível obter reuso por herança, associação dinâmica e definição de interfaces. De acordo com um levantamento realizado por Yassin e Fayad em 1999: 55% dos *frameworks são caixa-cinza*, 30% dos *frameworks são caixa-branca* e 15% dos *frameworks são caixa-preta* (MASIERO & BRAGA, 2001a).

2.3.4 Considerações

Normalmente, *frameworks caixa-branca* são mais difíceis de usar que caixa-preta, uma vez que forçam o desenvolvedor a ter bastante conhecimento sobre detalhes de implementação do *framework*. Subclasses ficam ligadas fortemente às suas superclasses, fazendo com que qualquer mudança na superclasse cause impacto nas subclasses. Além disso, o uso de herança para combinar funcionalidade pode levar a uma proliferação de subclasses, aumentando a complexidade de software.

Atualmente, diversos especialistas em orientação a objetos defendem a utilização de *frameworks black*-box *à white-box*, formando sistemas mais flexíveis e dinâmicos (JOHNSON, 1988; GAMMA, 1994; FAYAD & SCHMIDT, 1997). Em trabalhos como (JOHNSON, 1988; ROBERTS & JOHNSON, 1998), é destacado o fato de que *frameworks* imaturos, onde o desenvolvedor ainda não tem muito conhecimento sobre o domínio que será usado, nem sobre quais os seus aspectos fixos e variáveis, devem começar com a abordagem *white-box*, já que ainda não se tem condições de elaborar uma interface estável de adaptação do *framework*. Há ainda a possibilidade de desenvolver frameworks *gray-box*, que resultam da combinação de *caixa-branca* e *caixa-preta*.

Resumindo, um *framework caixa branca* pode ser mais fácil de projetar, pois não há necessidade de prever todas as alternativas de implementação possíveis. Já o *framework caixa preta* é mais difícil de projetar por haver a necessidade de fazer essa previsão. Por outro lado, o framework caixa preta é mais fácil de usar, pois basta escolher a implementação desejada, enquanto no caixa branca é necessário fazer essas implementações completas. *Frameworks caixa branca* podem evoluir para se tornar cada

vez mais caixa preta (JOHNSON, 1997). Isso pode ser conseguido de forma gradativa, implementando-se várias alternativas que depois são aproveitadas na instanciação do *framework.* Ao mesmo tempo não se fecha totalmente o *framework,* permitindo ao usuário continuar usando-o como *caixa-branca*. Após um certo tempo, estarão disponíveis diversas alternativas e então pode-se decidir realmente em torná-lo *framework caixa-preta*. A medida em que o *framework* vai se tornando mais *caixa-preta,* diminui o número de objetos criados, embora aumente a complexidade das suas interconexões. Além disso, o "scripting" fica mais importante, por permitir a especificação das classes que farão parte da aplicação e sua interconexão. Pode-se também construir ferramentas de apoio que ajudem na especificação da aplicação. Essas ferramentas têm como objetivo facilitar a instanciação do *framework,* por meio do uso de componentes visuais que sejam mais fáceis de usar do que os comandos do "script".

2.4 Classificação de um framework quanto a finalidade

Alguns autores tentam classificar *frameworks* quanto ao contexto, caracterizando tipos distintos de *frameworks* de acordo com os seus propósitos. Desta forma, *frameworks* podem ser divididos em *framework de infra-estrutura de sistema*, *framework de integração de "middleware"* e *framework de aplicaçã*o ou *aplicação empresarial* (FAYAD & SCHMIDT, 1997).

2.4.1 Framework de infra-estrutura

São usados para dar suporte a diversos tipos de aplicação, independentemente dos domínios endereçados por estas aplicações. Eles simplificam o desenvolvimento da infra-estrutura de sistemas portáveis e eficientes, como por exemplo os sistemas operacionais, sistemas de comunicação, interfaces com o usuário e ferramentas de processamento de linguagem. Em geral são usados internamente em uma organização de software e não são vendidos a clientes diretamente. O *framework* AWT seria classificado como um framework de infra-estrutura de sistema.

2.4.2 Framework de integração de “middleware”

São usados para integrar aplicações e componentes distribuídos, como *frameworks* CORBA ORB, DCOM, implementações do padrão ODMG, etc. Eles são projetados para melhorar a habilidade de desenvolvedores em modularizar, reutilizar e estender sua infra-estrutura de software para funcionar “seamlessly” em um ambiente distribuído.

2.4.3 Framework de aplicação ou aplicação empresarial

Estão voltados a domínios de aplicações mais amplos e são a pedra fundamental para atividades de negócios das empresas, como telecomunicações, aviação, manufatura e engenharia financeira. *Frameworks* deste tipo capturam elementos invariantes de domínio, e deixam em aberto aspectos específicos de cada aplicação. São também mais caros para desenvolver ou comprar, mas podem dar um retorno substancial do investimento, já que permitem o desenvolvimento de aplicações e produtos diretamente.

2.4.4 Considerações

Framework de infra-estrutura e integração preocupam-se basicamente com problemas internos de desenvolvimento de software e são independentes de domínio de aplicação, enquanto *framework de aplicação* tem o objetivo de apoiar o desenvolvimento de aplicações dirigidas ao usuário e produtos em domínios específicos. Comparados com as outras categorias, *framework de aplicação* são os mais difíceis e custosos para se desenvolver e comercializar (FAYAD & SCHMIDT, 1997; JOHNSON, 1997).

2.5 Framework MVC (Model-View-Controller)

Para ilustrar o funcionamento básico de um *framewor*k, esta seção apresenta os componentes básicos do *frameworks* MVC (KRASNER & POPE, 1988), que é provavelmente um dos *frameworks* mais conhecidos na comunidade de software. O MVC é usado para a construção de interfaces gráficas, e surgiu no ambiente do *Smalltalk-80* (KRASNER & POPE, 1988). Ele divide uma aplicação em três abstrações, que são *model*

(modelo), *view* (visão) e *controller* (controlador). A abstração *Modelo* define a semântica da aplicação, mantém seu estado e define seu comportamento. Cada abstração *Visão* permite uma forma de apresentação visual do modelo para os usuários da aplicação, podendo um mesmo modelo estar associado a várias visões. *Controladores* definem o modo de interação do usuário com os modelos e visões da aplicação, e estão relacionados com dispositivos de entrada de dados (teclado, mouse, etc.).

Cada uma destas abstrações corresponde a uma classe abstrata no *framework* MVC, e elas cooperam através de um protocolo de interação bem definido. As classes são definidas da seguinte maneira:

- **Classe *Model*:** um objeto *Model* pode manter objetos dependentes *(views* e *controller*s), e enviar mensagens de notificação de modificações no modelo para os dependentes. A classe *Model* define um protocolo de mensagens específico para estes propósitos. Um modelo de uma aplicação é uma instância de uma subclasse de *Model*, que deve implementar comportamento específico da aplicação, ou seja, as subclasses de *Model* são as abstrações do domínio do problema.

- **Classe *View*:** um objeto *View* pode ser decomposto em outros objetos *Vie*w, chamados de subvisões. A classe *View* define protocolo para interagir com objetos *controller* e *model*, interagir com suas subvisões, coordenando ações de transformação e apresentação. Um objeto *View* em uma aplicação é uma instância de uma subclasse de *Vie*w. A subclasse tem que fornecer a implementação para o protocolo de apresentação, para definir detalhes específicos do modelo que representa.

- **Classe *Controller*:** define protocolo para manipular objetos *Model* e *Vie*w, a partir de mensagens de interação enviadas pelo usuário através de dispositivos de entrada de dados.

A arquitetura MVC é mais referenciada no desenvolvimento de sistemas interativos. Ela divide uma interface em um conjunto de tríades de objetos, cada objeto do sistema pertencente às classes *model*, *view* ou *controller*, ilustradas na figura

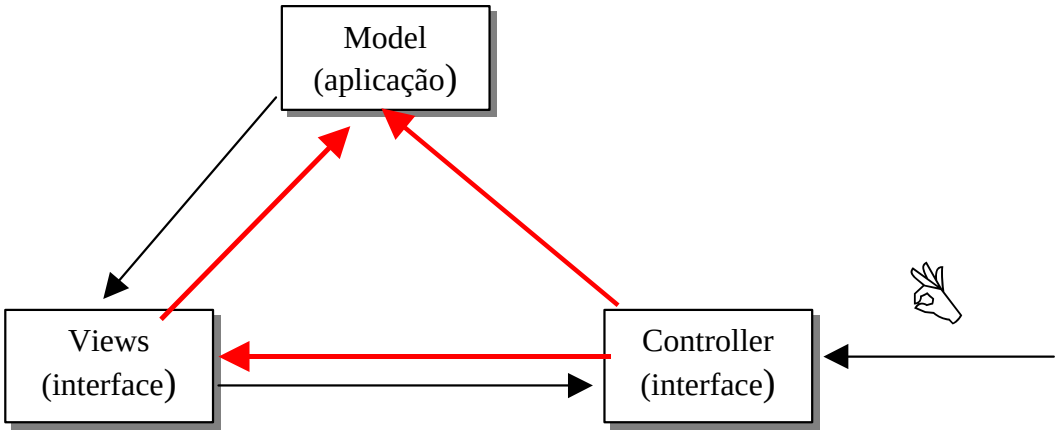

Figura 2.1: Modelo esquemático do modelo MVC

Conforme a Figura 2.1, as considerações pertinentes à aplicação (objeto *Model*, representado por exemplo, por tabelas em banco de dados), são isoladas daquelas referentes à interface (objeto *Controller* ou *View*). O código que trata das entradas do usuário é da responsabilidade de objetos *Controller* e o código responsável pelas respostas do sistema, é de responsabilidade dos objetos *View*. Por exemplo, quando o usuário pressiona um botão, o respectivo objeto *Controller* chama o método pertinente do objeto *Model* para tratar o evento do usuário. Um objeto *Controller* chama os métodos de objetos *View* para solicitar a modificação da apresentação em função de mudanças no contexto da interface (BASS & COUTAZ, 1991). Na Figura 2.1 as setas internas (ou com maior espessura) representam um forte acoplamento entre os objetos, já as setas externas (ou com menor espessura) significam um baixo acoplamento.

O desenvolvimento de uma interface pode se tornar eventualmente mais complicado pelo fato de o usuário não ter uma idéia exata dos requisitos necessários para o sistema em desenvolvimento (BASS & COUTAZ, 1991).

2.6 A importância da experiência para desenvolver softwares

Boas técnicas e heurísticas de projeto de software orientado a objeto são obtidas através de experiência. Desenvolvedores experientes se depararam com diversos problemas recorrentes de projeto, e criaram soluções satisfatórias para resolvê-los. Eles não desenvolvem aplicações partindo do zero: reutilizam a sua experiência, aplicando e adaptando as soluções desenvolvidas na resolução de problemas recorrentes semelhantes (GERBER, 1999).

A reutilização de experiência é o que diferencia desenvolvedores experientes dos novatos. Muitas vezes os novatos têm que passar pelos mesmos problemas encontrados pelos experientes, quando não existe registro e transferência do conhecimento desses desenvolvedores experientes para os novatos desfrutarem (GERBER, 1999).

Toolkit é uma biblioteca de rotinas usadas por programadores para a implementação de detalhes de baixo nível. Em geral, quando se emprega um *toolkit* na implementação de uma interface, o controle reside no componente computacional (ou aplicação), que define a funcionalidade do sistema, ou seja, funções que auxiliam os usuários na realização de suas tarefas. A implementação resultante é um conjunto de *widgets*. *Widget* é um objeto de interação com apresentação e comportamento bem definidos. Implementam elementos típicos de uma interface como botões, menus e outros. Um *toolkit* ainda pode possuir ferramenta que facilita o posicionamento destes objetos de interação (*widgets*) na figura. A interface desenvolvida com o uso de um *toolkit* geralmente interage com a aplicação através de *callbacks*. Ou seja, funções fornecidas pelo programador para serem executadas em pontos específicos da interação do sistema com o usuário, em resposta a algum evento ocorrido (PAUSCH, 1992).

Frameworks e *Toolkits* são o resultado da experiência de desenvolvedores, que disponibilizam produtos para outros desenvolvedores reutilizarem e adaptarem às particularidades das suas aplicações. O desenvolvedor que usa estes produtos não sabe como é sua construção interna, quais as decisões de projeto tomadas, etc., pois concentra-se principalmente em como adaptar esses componentes de software às suas necessidades. Isto é importante quando ele quer tratá-los realmente como “caixas-pretas”, porém em muitas situações os desenvolvedores se sentem desencorajados a reutilizar algo que não sabem como foi construído, o “porquê” de um determinado projeto (GERBER, 1999).

Os projetos de desenvolvimento de software têm progressivamente aumentado de tamanho e complexidade, sendo cada vez mais comum sua realização por equipes de médio porte (entre dez e vinte desenvolvedores) e grande porte (acima de vinte desenvolvedores) (MURTA, 2000). Com as facilidades de comunicação proporcionadas pela Internet, a

necessidade de experiência em diversas áreas de conhecimento e a pressão por cronogramas mais restritos, alguns projetos são realizados por diversas equipes trabalhando concorrentemente. Ainda mais, as dificuldades de reunir os especialistas necessários em um mesmo local físico e a delegação do desenvolvimento de determinados componentes para outras empresas, são exemplos de fatores que podem exigir que as equipes participantes de um projeto estejam geograficamente distribuídas (MURTA, 2000).

A existência de mecanismos eficientes de comunicação, como a Internet, não é suficiente para solucionar os problemas de desenvolvimento concorrente de projetos de software. Ao contrário, o novo cenário traz novos problemas para o processo de desenvolvimento. Por exemplo, o trabalho concorrente de equipes geograficamente distribuídas dificulta o controle de alterações nos componentes de um projeto em desenvolvimento. Mesmo quando precedida de uma precisa definição das interfaces entre os componentes, a realização de um projeto pode exigir que diversos desenvolvedores alterem simultaneamente os mesmos componentes. Esta situação exige a adoção de políticas para manter a consistência entre os componentes da versão atual do projeto ou permitir que essa consistência seja posteriormente restituída (MURTA, 2000).

2.7 Síntese de trabalhos afins

Diversos trabalhos na literatura têm dado enfoque à construção e uso de *frameworks*. O *framework* de aplicação OOHDM-Java, cujo domínio é composto pelas aplicações hipermídia modeladas com o método OOHDM e disponibilizadas na WWW, considera uma separação entre os dados da aplicação, os objetos navegacionais e os objetos de interface, resultando em uma arquitetura mais flexível. Ele pode ser classificado como um *framework caixa-branca*, pelo fato de ser necessário que o programador entenda o seu projeto e a sua implementação, até um determinado grau de detalhe, para poder utilizá-lo. No entanto, o *framework* é composto internamente por alguns componentes considerados caixa-preta, já que estes requerem apenas o conhecimento da sua interface externa para serem utilizados. O artigo traz detalhes de implementação e outros ligados ao OOHDM, que são mais difíceis de entender se o leitor não domina Java ou o método OOHDM. O

interessante é que a implementação do *framework* foi bem elaborada e consistente com a fase de projeto. Usa várias tecnologias e realmente enfatiza a importância de fazer a modelagem antes de implementar uma aplicação baseada nesse *framework*. HotDraw (JOHNSON, 1992) é um *framework* gráfico bidimensional para editores de desenho estruturado, escrito no VisualWorks Smalltalk. Ele tem sido usado para criar diversos editores diferentes, desde ferramentas CASE até “HyperCard clone”. Pode-se facilmente criar novas figuras e ferramentas especiais de manipulação para seus desenhos. Os desenhos do HotDraw podem ser animados.

O **Framework GREN,** classificado como caixa-branca, tem por objetivo construir um *framework* para apoiar a linguagem de padrões Gestão de Recursos de Negócios (GRN), de forma que aplicações concretas no domínio de gestão de recursos de negócios possam ser construídas com o menor esforço possível. O Ambiente utilizado para o desenvolvimento foi VisualWorks 5i.3 NC, usando a Base de dados MySQL. Como documentação existente, há o diagrama de classes e o cookbook: passo a passo o que deve ser feito para instanciar o framework para uma aplicação específica (MASIERO & BRAGA, 2001b).

O **Compass Framework** agiliza e documenta todo o processo de projeto, reduzindo custos e prazos de implementação. Segundo estudo da empresa de pesquisa Standish Group International, 40% dos projetos de desenvolvimento são cancelados antes de estarem concluídos, 33% sofrem atraso ou estouram o orçamento e apenas 27% são bem-sucedidos. Atenta a essa realidade, Cumerlato & Schuster desenvolveram ao longo de mais de 13 anos de experiência, uma arquitetura completa de desenvolvimento de sistemas, que é o **Compass Framework.** Essa arquitetura visa minimizar os erros e aumentar a produtividade durante todas as etapas de análise, projeto e implementação de sistemas. O **Compass Framework** é um conjunto integrado de procedimentos, técnicas e ferramentas que contempla todas as etapas do desenvolvimento de sistemas de informação (CUMERLATO & SCHUSTER, 2002).

Projeto MOBILE: Frameworks, Design Patterns e Componentes**.** O projeto se propõe a desenvolver e avaliar métodos distintos para o design de *frameworks de software*. Ao final do período de 4 anos estes métodos terão sido extensamente refinados e amplamente validados pela sua utilização no desenvolvimento de *frameworks* para diferentes domínios de aplicação. Espera-se que os métodos desenvolvidos se constituam em uma importante contribuição para a produção de software de qualidade e para o aumento da produtividade no desenvolvimento de software (LUCENA & MILIDIU, 1998). Adicionalmente, pretende-se desenvolver diversos frameworks para os seguintes domínios de aplicação:

- Educação baseada na Web e Comércio Eletrônico.
- LUA (Componentware).
- Frameworks para famílias de algoritmos.
- Frameworks para a manipulação de documentos multimídia/hipermídia.
- Frameworks para agentes em ambientes 3D.
- Frameworks para serviços de redes de comunicação.
- O projeto pretende também desenvolver uma família de ferramentas (um *framework*) para dar suporte ao desenvolvimento de novos frameworks.

Frameworks para Comércio Eletrônico: O projeto é coordenado pelos professores Carlos Lucena e Ruy Milidiu e, constituído por dez sub-projetos patrocinados pela IBM Brasil (LUCENA & MILIDIU, 1998). Os mesmos são mencionados a seguir.

- **2BuyNet** - Framework para Geração e Administração de lojas na Internet.
- **Vmarket** - Framework para geração de mercados virtuais "Consumer to Consumer" na Internet.

- **CommercePipe** - Framework para geração de mercados virtuais "Business to Consumer" na Internet.

- **ContentNet** - Framework para interoperabilidade de conteúdos educacionais utilizando a plataforma EDUCAUSE –IMS.

- **Portalware Development Framework** - Framework para o Desenvolvimento de Portais através da instanciação de **Portalware Services**.

- **VBroker** – Um framework para suporte a decisão no Mercado Financeiro.

- **IDeal –** Um Portal para Negócios.

- **PkiEnabled -** Um framework para aplicações de comércio eletrônico baseado em chave pública.

- **VGroups** - Framework para a criação de Grupos Virtuais de Consumo.

- **eBP** – eBusiness Provider. Um framework para mercados B2B.

Além dos projetos citados nos dois parágrafos anteriores, relacionados diretamente com frameworks, existem igualmente outros projetos que o LES do DI da PUC do Rio de Janeiro está desenvolvendo na área de Engenharia de Software.

Taligent (1995), relata os benefícios obtidos versus os custos de construção de *frameworks*.

Benefícios:

- Economia à longo prazo.

- Aproveitamento da experiência de especialistas do domínio.

- Maior consistência e integração entre aplicações.

- Redução da manutenção: menos linhas de código nas aplicações, reparos no framework se propagam pelas aplicações.

- Melhora da produtividade: os programadores podem se concentrar nas características específicas de suas aplicações.

Custos:

- Mais esforço para construção e aprendizado.

- Programas mais difíceis de depurar.

- Necessária documentação de manutenção e apoio.

2.8 Exemplos de frameworks

A maioria dos *frameworks* existentes é para domínios técnicos tais como interfaces com o usuário ou distribuição, como por exemplo o framework MacApp, específico para aplicações Macintosh, o qual é “free”, o Lisa Toolkit, o Smalltalk Model View Controller, o Interviews, o LAMA, para desenvolvimento de software, também é “free”, o Amulet, para para interface gráfica, igualmente “free”, etc. A maioria dos *frameworks* para aplicações específicas não é de domínio público. Exemplos são o ET++, ACE, Microsoft Foundation Classes (MFC) e DCOM, JavaSoft’s RMI e implementações do OMG’s CORBA, HBOC: framework para área médica, PrismTech BOF e CORBA: frameworks para indústria e manufatura e IBM San Francisco: aluguel pelo tempo que durar o desenvolvimento da aplicação específica.O Model-View-Controller (MVC) (KRASNER & POPE, 1988) foi o primeiro framework largamente utilizado. Ele foi inicialmente implementado em Smalltalk-80, mas atualmente conta com implementações em todas as versões existentes de Smalltalk, como o VisualWorks, VisualAge, Dolphin, Squeak, etc. O MVC é usado pelo Smalltalk como interface com o usuário, tendo mostrado a adequação da orientação a objetos para implementação de interfaces gráficas com o usuário.

2.9 Padrões de projeto (Design Patterns)

2.9.1 Introdução

Conforme Maldonado (MALDONADO, 2001):

> *"A origem dos padrões de projeto deu-se com o trabalho feito pelo arquiteto Christopher Alexander no final dos anos 70. Ele escreveu dois livros: "A Pattern Language"* (ALEXANDER, 1977) *e "A Timeless Way of Building" (ALEXANDER, 1979) que, além de exemplificarem, descrevem seu método para documentação de padrões. O trabalho de Alexander, apesar de ser voltado para a arquitetura, possui uma fundamentação básica que pode ser abstraída para a área de software. Os padrões permaneceram esquecidos por um tempo, até que ressurgiram na conferência sobre programação orientada a objetos (OOPSLA) de 1987, mais especificamente, no Workshop sobre Especificação e Projeto para Programação Orientada a Objetos (BECK, 1987). Nesse trabalho, Beck e Cunningham falam sobre uma linguagem de padrões para projetar janelas em Smalltalk. A partir de então, muitos artigos, revistas e livros têm aparecido abordando padrões de software, que descrevem soluções para problemas que ocorrem freqüentemente no desenvolvimento de software e que podem ser reusadas por outros desenvolvedores. Um dos primeiros trabalhos estabelecendo padrões foi o de Coad & Yourdon (1992), no qual são descritos sete padrões de análise. Em 1993, Gamma e outros introduzem os primeiros de seus vinte e três padrões de projeto (GAMMA, 1993), que seriam publicados em 1995 (GAMMA, 1995)*".

Um padrão descreve uma solução para um problema que ocorre com freqüência durante o desenvolvimento de software, podendo ser considerado como um par "problema/solução" (BUSCHMANN, 1996). Projetistas familiarizados com certos padrões podem aplicá-los imediatamente a problemas de projeto, sem ter que redescobri-los (GAMMA, 1995). Um padrão é um conjunto de informações instrutivas que possui um

nome e que capta a estrutura essencial e o raciocínio de uma família de soluções comprovadamente bem sucedidas para um problema repetido que ocorre sob um determinado contexto e um conjunto de repercussões (APPLETON, 1997).

Padrões de software podem se referir a diferentes níveis de abstração no desenvolvimento de sistemas orientados a objetos. Assim, existem padrões arquitetônicos em que o nível de abstração é bastante alto, padrões de análise, padrões de projeto, padrões de código, entre outros. As diversas categorias de padrões são discutidas a seguir. O uso de padrões proporciona um vocabulário comum para a comunicação entre projetistas, criando abstrações num nível superior ao de classes e garantindo uniformidade na estrutura do software. Além disto, eles atuam como blocos construtivos a partir dos quais projetos mais complexos podem ser construídos (GAMMA, 1995).

2.9.2 Conceito

O conceito de padrões (ALEXANDER, 1977; APPLETON, 1997; GAMMA, 1994; BUSCHMANN, 1996) surgiu com o propósito de dar algum formalismo para o registro e disseminação de experiência. Um padrão pode ser definido como o encapsulamento da descrição abstrata e estruturada de uma solução satisfatória para um problema que ocorre repetidamente dentro de um contexto, dado um conjunto de forças ou restrições que atuam sobre o problema. O conceito de padrões foi proposto inicialmente na área de arquitetura (ALEXANDER, 1977), e vem sendo gradualmente incorporado na comunidade de software. Em resumo, padrões de software visam à reutilização de experiência em desenvolvimento de software (GERBER, 1999).

Um dos trabalhos mais famosos em padrões de software é o catálogo de *design patterns* de Gamma (1994), que apresenta um grupo de 24 padrões que endereçam problemas de projeto de software orientado a objeto, independentes de domínio. Na sua grande maioria, estes padrões tratam de problemas relacionados com reusabilidade e flexibilidade de *framework*s, e pode-se dizer que foram abstraídos de projetos bem sucedidos de *frameworks* executados pelos pesquisadores que publicaram o catálogo.

2.9.3 Identificação

Padrões são identificados por nomes, que, com o tempo, são incorporados no vocabulário dos desenvolvedores. Além disso, padrões descrevem o "porquê" das decisões tomadas para resolver os problemas, quais as justificativas, que estão diretamente relacionadas com as forças que atuam sobre os problemas. No projeto de *framework*s, as forças principais são normalmente relacionadas com aspectos de *genericidade* e *facilidade de adaptaçã*o (GERBER, 1999).

Enquanto um padrão endereça um problema elementar, uma linguagem de padrões (ALEXANDER, 1977; APPLETON, 1997) endereça um problema complexo, sendo formada por um conjunto de padrões interligados colaborando para resolver o problema complexo. Dentro do projeto de software, pode-se comparar padrões a classes individuais e linguagens de padrões a *framework*s, de acordo com os papéis que exercem para resolver um problema de projeto. Linguagens de padrões têm sido consideradas pela comunidade de software como uma boa alternativa aos tipos de documentação tradicionais usados em desenvolvimento de software, como guias estilo "how-to", "dicas-truques-segredos", "Gems", etc... Particularmente, algumas linguagens de padrões publicadas mostram como construir *frameworks* genéricos (ROBERTS & JOHNSON, 1998) ou como utilizar (adaptar e estender) *frameworks* (JOHNSON, 1992), em diversos domínios de aplicação. Em Engenharia de Software, diversos trabalhos em padrões e linguagens de padrões têm sido publicados, endereçando vários domínios de aplicação (Comunicação, Distribuição, Interfaces, Hipermídia, etc.), como também as diversas fases e atividades relacionadas com desenvolvimento de software (Organização de equipes, análise de requisitos, análise, projeto, implementação, etc.) (ALEXANDER, 1977; GAMMA, 1994; MARTIN, 1995; BUSCHMANN, 1996).

2.9.4 Vantagens

As principais vantagens de usar *Design Patterns* para o desenvolvimento de aplicações, segundo Araújo (2001), estão descritas abaixo:

- ✓ Documentar experiência de desenho válidas e bem conhecidas.
- ✓ Especificar conceitos acima do nível de classes e objetos.
- ✓ Descrever estrutura e comportamento de objetos que cooperam entre si.
- ✓ Prover um vocabulário comum.
- ✓ Abordar propriedades específicas da solução de um problema.
- ✓ Cobrir vários níveis de abstração e de escala.**Metapadrões**

Existe também o conceito de metapadrão (metapattern) (PREE, 1995) que é uma abordagem complementar à abordagem dos padrões de projeto (GAMMA, 1994). Pree observou e classificou alguns arranjos de funcionalidade sobre métodos, que se repetem em diferentes padrões de projeto – o que denominou metapadrões. Estes trabalham em um nível de abstração superior aos padrões de projeto. Eles definem como manter a flexibilidade das classes, encapsulando a funcionalidade sujeita a alteração em métodos hook[2], chamados por métodos template[3].

Um framework bem projetado deve conter um conjunto flexível de métodos template e hook. Métodos template implementam os pontos fixos (ou imutáveis) – frozen spots (PREE, 1995) do framework, ou seja, as partes comuns das aplicações. Já os métodos hook implementam os pontos adaptáveis (PREE, 1995) – hot spots que são as partes que podem ser estendidas para cada aplicação específica.

---

[2] O método hook sempre será invocado por um método template, seu objetivo é proporcionar a flexibilização a um método template.

[3] O método template ao ser executado invoca pelo menos um outro método – um método hook. Este método comporta a parte imutável de um procedimento.

Segundo Pree (1995), os metapadrões complementam a abordagem de Gamma, pois os mesmos permitem a visualização dos pontos adaptáveis de um framework. Isto facilita o entendimento e conseqüentemente a utilização do framework. Cada metapadrão ocorre provavelmente várias vezes em um framework, logo os metapadrões podem ser vistos como um meio de capturar e classificar o projeto de um framework.

2.9.6 Padrões x Metapadrões

Tanto padrões de projeto quanto metapadrões podem ser utilizados no projeto de um framework. Ambas as abordagens têm por objetivo tornar o framework mais flexível. Cabe ressaltar alguns aspectos:

- Metapadrões podem contribuir para documentar o projeto de qualquer framework, independente de seu domínio (PREE, 1995).

- Metapadrões são mais abstratos que padrões de projeto. Um mesmo metapadrão pode ser utilizado no projeto de um framework na solução de diferentes problemas, enquanto que padrões de projeto são aplicados a apenas um único problema. Por outro lado, como os padrões de projeto são mais concretos são também mais fáceis de manusear e entender.

- Padrões de projeto são um guia mais concreto para a compreensão dos pontos adaptáveis de um framework (SCHMIDT, 1995).

- Metapadrões são mais flexíveis que os padrões de projeto. Porém esta vantagem é também uma desvantagem, uma vez que para problemas similares, soluções diferentes podem ser desenvolvidas (SCHMID, 1996).

- O catálogo de padrões de projeto, como um complemento para as metodologias de análise e projeto orientado a objetos é insuficiente para apoiar o desenvolvimento de frameworks (PREE, 1995).

- O catálogo de padrões de projeto tem se tornado cada vez mais completo (SCHMID, 1996).

Em suma, as duas abordagens apresentam vantagens e desvantagens, sendo mais adequado, no caso de frameworks, utilizar uma combinação de ambas, conforme proposto por (SCHMIDT, 1995; SCHMID, 1996).

2.9.7 Padrões (Design Patterns) x Frameworks

Os conceitos de *design patterns* e *frameworks* são confundidos em algumas vezes, apesar de existirem algumas diferenças fundamentais. Ambos são elementos de projeto que contêm micro-arquiteturas de classes e objetos cooperantes para resolver um problema recorrente de projeto OO.

Em Gamma (1994), são apresentadas algumas diferenças importantes entre *design patterns* de propósito geral e *framework*s:

a) *Design patterns* são mais abstratos do que *framework*s. *Frameworks* podem conter código (implementação), mas apenas exemplos de *design patterns* podem ser implementados. Uma vantagem de *frameworks* é que eles podem ser implementados em uma linguagem de programação e não apenas estudados, mas executados e reutilizados diretamente. Em contraste, *design patterns* tem que ser implementados cada vez que são reutilizados. Eles explicam também intenção, forças, e conseqüências do seu projeto.

b) *Design patterns* são elementos que promovem o reuso de projeto e de código em um nível de granularidade menor que *framework*s. Um *framework* comum geralmente contém diversos *design pattern*s, mas o contrário não é verdadeiro.

c) *Design patterns* são menos especializados do que *framework*s. *Framework* sempre endereça um particular domínio de aplicação. Um *framework* de editor gráfico pode ser usado em uma simulação de fábrica, mas não será confundido com um *framework* de simulação. Em contraste, *design patterns* podem ser usados em praticamente qualquer tipo de aplicação. Um *framework* é um conjunto de *design patterns* de um domínio de

aplicações. Apesar de que *design patterns* mais específicos de domínio existem, mesmo assim não ditam a arquitetura de uma aplicação, da maneira que um *framework* faz.

Fazendo uma comparação entre padrões e frameworks, Johnson (1997) diz que padrões são mais abstratos do que *frameworks*, porque são soluções genéricas, válidas para diferentes contextos e problemas. Já *frameworks,* são voltados para problemas específicos. Os padrões são menores do que *frameworks*. Em geral padrões são compostos por 2 ou 3 classes, enquanto os *frameworks* envolvem geralmente um número maior de classes, podendo englobar diversos padrões. Os padrões são menos especializados do que *frameworks*, já que os *frameworks* geralmente são desenvolvidos para um domínio de aplicação específico e os padrões (ou muitos deles) podem ser usados em diversos tipos de aplicação. Apesar de haver padrões mais especializados, eles não são capazes de ditar a arquitetura de uma aplicação, ao contrário dos *frameworks*. Comparando *frameworks* com bibliotecas de programação, nas bibliotecas os componentes não estão inter-relacionados, enquanto em *frameworks* os componentes cooperam entre si. Em uma biblioteca de programação o desenvolvedor é responsável pela chamada de componentes e fluxo de controle do programa enquanto em um *framework* há a inversão de papéis: o programa principal é reutilizado e o desenvolvedor decide quais componentes são chamados e, deriva novos componentes. O *framework* determina a estrutura geral e o fluxo de controle do programa.

2.9.8 Frameworks x Geradores de Aplicação

Comparando *frameworks* com geradores de aplicação[4], pelo menos duas semelhanças são encontradas (MASIERO & BRAGA, 20001a):

- ambos envolvem análise de domínio para sua construção, na qual diferentes aplicações dentro do mesmo domínio são estudadas e posteriormente generalizadas;

- ambos resultam em código que integrará uma aplicação específica, embora com o uso de um *framework* novas classes possam ser criadas e nele embutidas para uso futuro;

- nos geradores de aplicação é necessário fornecer uma especificação do sistema em alto nível para obter o produto final, enquanto no *framework* são apenas informados os pontos variáveis, com possível criação de novas classes, para a instanciação do *framework* para uma aplicação específica;

- *frameworks* implicam o uso da orientação a objetos, já que sua definição envolve classes, enquanto geradores de aplicação são independentes do paradigma de desenvolvimento;

- nos *frameworks* o código é herdado enquanto em geradores de aplicação o código é gerado. Assim, grande parte do código de um *framework* é incorporado ao produto final, enquanto grande parte do código do gerador de aplicação existe apenas para fornecer mecanismos de geração do código do produto final.

2.10 Considerações finais

Através desse capítulo teve-se uma noção a respeito da abordagem de OO, da importância de construir um framework, bem como os principais itens que estão envolvidos

na construção do mesmo. Percebe-se que através da produção de um framework, tem-se não apenas o reuso da arquitetura de classes, mas também a minimização do esforço para o desenvolvimento de aplicações, dado que é definido o protocolo de controle da aplicação (definição da arquitetura), liberando o desenvolvedor de software desta preocupação (SILVA, 2000). No próximo capítulo, será abordado o contexto a ser abrangido pelo framework PRODAGRO, qual seja, gestão e comercialização de produtos agropecuários.

[4] Geradores de aplicação são ferramentas de apoio para o desenvolvimento de aplicações, semelhantes aos geradores de relatórios.

3 APLICAÇÕES DE GESTÃO E COMERCIALIZAÇÃO DE PRODUTOS AGROPECUÁRIOS

3.1 Introdução

A informática tornou-se uma ferramenta indispensável na agricultura, visto que as necessidades de controlar os processos produtivos das propriedades rurais e fazendas, bem como, os processos de semeadura, armazenagem e comercialização, são cada vez maiores. Além disso, é necessário ter disponível o maior volume possível de informações para tomadas de decisão acerca de uma melhora no desenrolar dos processos a serem executados. Os agricultores estão usando os computadores para contabilizar os custos da produção, controlar o armazenamento, o transporte e até o uso racional do solo. Com os crescentes avanços tecnológicos, cada vez mais surgem meios de abranger as necessidades existentes no agronegócio.

Nos Estados Unidos, uma parcela considerável dos produtores rurais usa a Internet para se comunicar através do e-mail, bem como para ter acesso aos mapas metereológicos atualizados a cada hora, contendo dados de radar, satélite e estações metereológicas. Diante disso, será possível obter informações sobre o regime de ventos e de chuvas e a previsão de tempo.

O sistema brasileiro de pesquisa agropecuária possui um enorme estoque de informação produzido ao longo de mais de um século de existência. Este fato aliado à importância estratégica que o agronegócio representa para o Brasil, não apenas no equilíbrio da balança comercial mas também como fator de desenvolvimento regional, impõe enorme responsabilidade às instituições de pesquisa: fazer com que o conhecimento seja trabalhado em produtos de informação adequados ao consumo dos produtores agrícolas, contribuindo para o desenvolvimento sustentado.

3.2 Produtos Agropecuários

Os produtos agropecuários são oriundos da agricultura e produção animal. A agricultura engloba as culturas provenientes dos cereais, oleaginosas, olericultura e fruticultura. Já a produção animal, divide-se por finalidades, sendo elas de corte, leite e reprodução.

3.2.1 Classificação dos produtos agropecuários

3.2.1.1 Produtos agrícolas

A soja, trigo e o milho, são as principais plantas de lavoura e estão associadas a produtores bem estruturados tecnicamente e com melhor capacidade gerencial. A grande expansão da soja deve-se à integração de esforços interinstitucionais na seleção de cultivares adaptadas as diferentes regiões do relevo nacional e às tecnologias geradas para, por exemplo, em algumas regiões, como no Cerrado, corrigir a baixa fertilidade natural dos solos. Os níveis de tecnologia e de uso de insumos nessas culturas tendem a se intensificar devido à constante procura por maiores rendimentos (EMBRAPA, 2001). Os principais derivados ou subprodutos dessas culturas são o óleo de soja, farelo de soja e trigo, casca de soja, farinha de trigo e milho e torta de soja.

A fruticultura tem grande importância econômica, pois tem uma expressiva função social ao permitir fonte de emprego durante o ano todo. Dentre as diversas frutíferas, é possível destacar as seguintes: manga, citrus, abacaxi, banana, acerola, maçã e graviola (EMBRAPA, 2001).

Com um grau de relevância menor, mas igualmente importantes, aparecem outras culturas como o feijão, arroz, café, cana-de-açúcar, algodão, milheto, sorgo, girassol, amendoim, gergelim, guandu, urucum, seringueira, pupunha e olerícolas. Se elas não têm uma expressão econômica significativa, mas representam uma alternativa para a diversificação da agricultura.

O Brasil está entre os cinco maiores produtores mundiais de café, cana-de-açúcar, milho, algodão, cacau, frutas tropicais, soja, mamona e algumas outras culturas. Entretanto, comparada à agricultura de outros países, muito ainda tem de ser feito pela agricultura do país. Os principais problemas encontrados são os físicos, causados por secas, geadas, enchentes, erosão, etc...; os humanos, causados pelas condições desfavoráveis para os trabalhadores; e os fundiários, causados pela concentração de grandes áreas nas mãos de proprietários que não as exploram, mantendo-as improdutivas.

3.2.1.2 Produção animal

A produção animal se ocupa de diversas subáreas, como a bovinocultura (bois, vacas, bezerros, etc...), suinocultura (suínos), avicultura (aves), caprinocultura (caprinos), ovinocultura (ovelhas), eqüinocultura (eqüinos ou cavalos), piscicultura (peixes), apicultura (abelhas), ranicultura (rã) e sericicultura (bicho da seda). Portanto, quando falamos em bovinos de corte, nos referimos aos bois tratados para comercialização. A produção animal pode ser extensiva ou intensiva. Na produção animal extensiva, o gado é criado solto, ao ar livre, sem técnicas especializadas, gerando baixa produtividade. Já na produção animal intensiva, o animal é criado com cuidados e técnicas avançadas, passando por um processo de seleção de raças e cruzamento. A diferença é que nessa última, devemos investir bem mais, mas o retorno também será mais imediato e maior. Já nos animais com a finalidade leiteira, devemos abordar o controle leiteiro de algumas categorias, como por exemplo, as vacas.

3.2.1.3 Produtos derivados

Os produtos derivados são aqueles originados de algum processo de industrialização de um dos produtos agropecuários, ou seja, provenientes de alguma das culturas envolvidas na agricultura e pecuária. Alguns exemplos são os subprodutos da fruticultura como o suco, néctar e geléia; da produção animal, como a carne, ovos, leite; dos cereais, como o farelo, a farinha, a casca; das oleaginosas, como o óleo.

3.3 Qualidade dos softwares agropecuários

De acordo com o Guia Agrosoft[5], em 1997 foram cadastrados 198 softwares, sendo 54 para administração rural, 7 para mapeamento geográfico, 15 para o gerenciamento do meio ambiente, 15 para o ensino e a pesquisa agropecuária, 38 para o gerenciamento de lavouras e 7 para outras aplicações. A produção animal é, de longe, o setor mais beneficiado pela agroinformática. Atualmente, existem 62 programas cadastrados para o gerenciamento de animais.

Com o aumento da oferta de softwares para o setor, também cresceu a quantidade de produtos de baixa qualidade. Para evitar que o produtor "leve gato por lebre" e para fomentar o desenvolvimento de softwares de alta qualidade, a Embrapa criou o Soluções Embrapa para a Agroinformática (ESP). O programa está sendo feito em parceria com a Agrosoft.

O objetivo é testar e avaliar os programas disponíveis e fornecer um selo de qualidade para o produto aprovado. "Dessa forma, é mais fácil convencer o produtor de que determinado software realmente funciona, pois ele foi validado pela Embrapa, uma instituição com alta credibilidade junto ao setor", diz Pedroso, diretor do Centro Nacional de Pesquisa Tecnológica em Informática para a Agricultura (CNPTIA), da Embrapa.

3.4 Aplicações agropecuárias do domínio de produtos agropecuários

Nesta seção serão descritas as principais aplicações que envolvem o domínio de produtos agropecuários e, que foram, senão totalmente, mas parcialmente analisadas pelo autor, a fim de serem contempladas ou não na produção do framework.

[5] Empresa de Desenvolvimento de Softwares para a Agricultura e a Pecuária. O endereço da Internet é www.agrosoft.com.br.

3.4.1 GV Farm System – Sistema de Gerenciamento Agropecuário

Esse sistema foi desenvolvido numa parceria da Embrapa em conjunto com a empresa Vale Verde de Juiz de Fora, Minas Gerais. A seguir, estão descritas as principais características e funcionalidades do mesmo.

O GV Farm System é um software de administração de fazendas de pecuária leiteira ou de corte, propriedades com atividades exclusivamente agrícolas ou com grande diversidade de atividades. Ele tem como objetivo gerenciar todos os fatores produtivos de uma propriedade, visando otimizar custos de produção de pequenos, médios e grandes produtores. As principais funções do sistema, divididas em módulos, são as seguintes:

- Fazendas, com dados das propriedades.
- Agricultura, com manejo de culturas, controle de glebas, plantio, colheita, máquinas e clima.
- Produção animal, com todo o controle de rebanhos (tanto de corte como de leite), controle sanitário, alimentar, reprodutivo, leiteiro e ganho de peso, com evolução automática de rebanho e, financeiro, com contas a pagar, receber, fluxo de caixa e controle de estoques.
- Todos os módulos possuem recursos de saída em relatórios e planilhas (CNPGL, 2002; VALE, 2002).

Esse software pode ser usado em quase todas as atividades agropecuárias, desde pequenas propriedades, até grandes conglomerados. Fazendas de pecuária leiteira, pecuária de corte, fazendas com atividades puramente agrícolas ou florestais e propriedades que têm uma grande diversidade de atividades. O enfoque é sempre para propriedades voltadas para sistemas produtivos, que utilizam ou pretendem utilizar recursos tecnológicos. O GV Farm System é uma importante ferramenta para racionalizar o uso destes recursos, permitindo às pessoas que gerenciam suas atividades a tomada de decisões.

O Módulo Pecuária, possibilita o acompanhamento das condições reprodutivas, sanitárias, nutricionais e produtivas do rebanho, bem como a evolução do plantel e o controle de peso. Este módulo permite a avaliação constante do rebanho ou de cada um dos animais que compõem o plantel. No momento da comercialização de qualquer um dos animais do rebanho, os dados cadastrados sobre esse animal podem orientar o criador a determinar o melhor preço para a sua venda.

No controle de ganho de peso é possível a entrada de peso por animais, por lote, ou por grupo de engorda. Os lotes são conjuntos de animais identificados dentro do sistema, e os grupos de engorda são animais não identificados, tratados como uma única entidade, normalmente usado por criadores de corte. O sistema informa as diferenças de peso da última pesagem, o ganho médio diário e outras informações. Para os criadores de corte é possível informar o desmame dos animais e a movimentação de animais entre os grupos de engorda. A evolução de peso dos animais ou dos grupos pode ser visualizada em gráficos. Os grupos podem ser alocados em pastagens cadastradas no módulo de agricultura, e é possível saber a lotação de cada pastagem.

O Módulo Agricultura reúne informações sobre culturas, solos, preços e safras, dentre outras. Este modo permite o cadastramento de culturas, sejam perenes, anuais, áreas de pastagem ou capineiras.

Portanto, é possível o cadastramento das culturas, das glebas, das safras e dos manejos empregados na fazenda. Para cada gleba é possível cadastrar todas as análises de solos efetuadas e as adubações realizadas. Em cada safra, é possível cadastrar os plantios (por gleba) e a colheita realizada, informando a produção e a produtividade. É possível visualizar os plantios e as colheitas por safra, gleba e cultura.

Além disso este módulo possui rotinas de planejamento de custos, de atividades e acompanhamento. É possível o planejamento do custo de uma determinada cultura, por hectare e por área total. Os insumos e os serviços são cadastrados no estoque do sistema (módulo financeiro). As atividades a serem executadas podem ser agendadas para uma determinada cultura, e o acompanhamento é feito para um determinado plantio. Por fim, é

possível cadastrar informações climáticas, tais como temperatura e precipitação pluviométrica, com visualização gráfica.

O usuário pode sempre tratar os animais individualmente, tendo, a partir da ficha individual, todas as informações relativas a um animal específico, não necessitando recorrer a funções diretamente. Portanto, se desejarmos saber sobre a produção leiteira ou sobre o histórico sanitário, esta informação estará disponível através da ficha individual. Por outro lado, se temos informações agrupadas de diversos animais, divididos em lotes ou não, podemos entrar com estas informações no modo de operação em lotes, acionando para isto a função específica. A Figura 3.1, Figura 3.2 e Figura 3.3, mostram alguns processos disponíveis na aplicação.

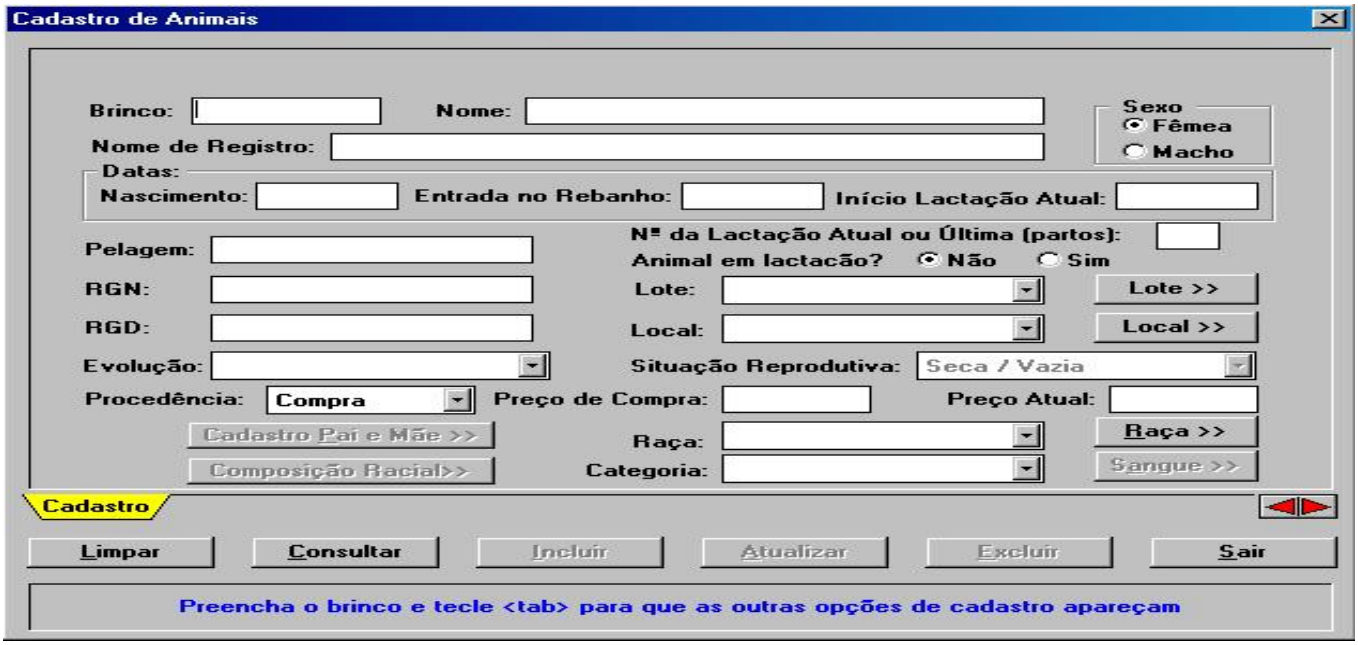

Figura 3.1: Cadastro de um produto pecuário (animal bovino)

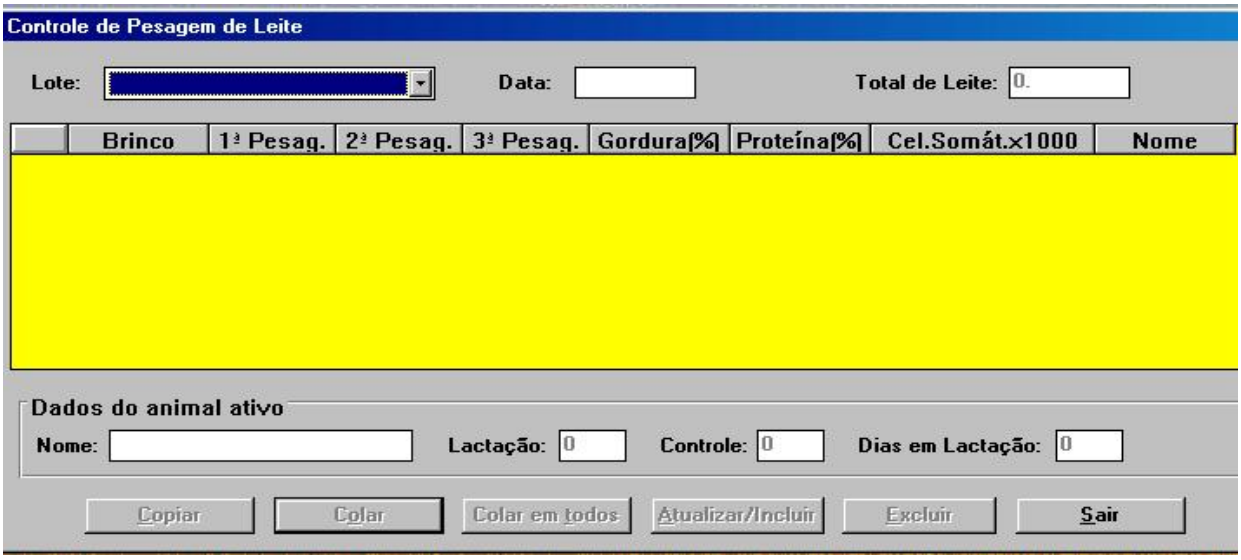

Figura 3.2: Controle da pesagem do leite

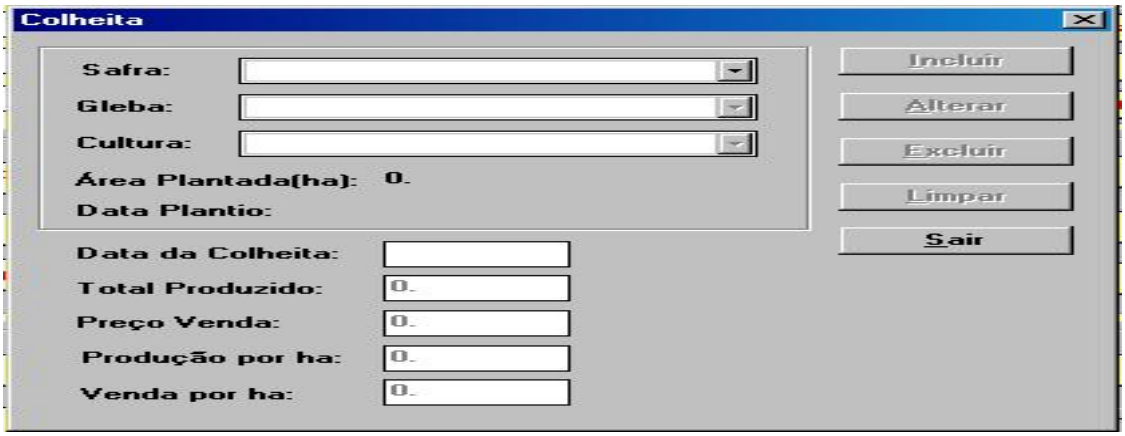

Figura 3.3: Controle de dados da colheita

3.4.2 CONGADO

O CONGADO é um software para controle da produção animal de corte, produção animal de leite e custos de produção. Além disto, é o software oficial da pesquisa de rastreabilidade e identificação eletrônica de bovinos da EMBRAPA - GADO DE CORTE (ALMA, 2002).

Por ter sido desenvolvido por uma equipe de analistas de sistemas, em conjunto com veterinários, zootecnistas e produtores rurais, o CONGADO é ao mesmo tempo um software extremamente técnico e prático de usar.

O CONGADO é utilizado pela Universidade Federal de Lavras no controle de suas fazendas experimentais de gado de leite. Este programa realiza todo o gerenciamento da

pecuária, tanto de corte, como de leite, levando em consideração a produção, reprodução e dados econômicos. As suas principais características são (ZOOTEC, 2002; ALMA, 2002):

- ✓ Trabalho individual ou por grupo de animais.
- ✓ Controle total da produção de leite e desenvolvimento dos animais.
- ✓ Controle total da reprodução: acompanhamentos reprodutivos, cios, coberturas, diagnóstico de gestação e transferências de embriões.
- ✓ Selecionador de grupo de animais por critérios de produção e reprodução, facilitando a formação de lotes, comparações e operações diárias - apartações, mudanças de pastos, controles sanitários, etc.
- ✓ Relatórios objetivos de tarefas do dia-a-dia da fazenda (vacas a diagnosticar, previsão de parto, encerramento de lactações), análises gerenciais (avaliação de inseminadores, estação de monta, composição do rebanho, índice de retorno), análises zootécnicas (ganho de peso, análise de lactações, desempenho de touros e matrizes).
- ✓ Fichas para coleta de informação no campo.
- ✓ Gráficos para análises gerenciais.
- ✓ Controle de custos em um plano de contas e fluxo de caixa.

3.4.3 ADM-AGRÍCOLA

Produto voltado para a administração técnica/financeira da atividade agrícola. O Adm-Agrícola, assim como o Adm-Rebanho alia a facilidade de uso com um conjunto de funções que o tornam um valioso elemento tanto no nível operacional, no dia a dia da fazenda, quanto no nível gerencial. As características principais do Adm-Agrícola (AGRISOFT, 2003):

- Versatilidade para uso nos mais diversos tipos de culturas tais como soja, milho, trigo, feijão, cana, citrus, inhame, etc.

- Registrar todas as atividades e tratos culturais realizados em cada um dos talhões da fazenda. Os produtos utilizados nestas atividades serão automaticamente baixados do estoque conforme as quantidades utilizadas.

- Trabalhar de forma integrada ao Adm-Máquinas (caso possua este programa) de forma a utilizar os custos de operação de cada máquina para valorizar as atividades realizadas nos talhões.

- Apurar os custos de produção por talhão ou ciclo de produção a partir das atividades realizadas e insumos aplicados.

- Acompanhar a colheita ao nível de cada romaneio de carga e controlar os estoques tanto da fazenda quanto externos em cooperativas e outros.

- Registrar as cargas transportadas por cada caminhão de forma a controlar os pagamentos dos fretes.

- Controlar todas as receitas e despesas ocorridas em quaisquer tipos de centros de custos (fazenda, casa, fábrica, escritório, etc...) podendo-se a partir do registro dos lançamentos de receitas e despesas, obter uma série de resultados tais como gráficos de composição e origem dos custos, balancetes por centro de custos, evolução de certos tipos de despesa ao longo do tempo, evolução do preço unitário de aquisição ou venda de produtos de nosso interesse, controle de contas bancárias ou do caixa, e etc.

- Manter um inventário dos bens existentes em cada um dos centros de custos controlados.

- Gerar fluxos de caixa diários de forma a controlar os compromissos a pagar e a receber.

3.4.4 ADM-REBANHO

O Adm-Rebanho é um produto para o gerenciamento da atividade de produção animal. Controlando tanto a parte técnica quanto a área financeira da atividade, é um software que permite ao pecuarista iniciar sua informatização de forma gradativa e implementar gradativamente novos e poderosos controles viabilizando tanto o gerenciamento de gado de seleção quanto de grandes plantéis controlados individualmente. Recursos para entrada de dados de modo extremamente otimizado facilitam o trabalho diário e agilizam a obtenção de resultados. As características principais do Adm-Rebanho (AGRISOFT, 2003):

- Controle do gado de corte e de leite.
- Suporta um número qualquer de fazendas.
- Recursos para entrada rápida de dados que viabilizam o controle de fazendas com grandes rebanhos onde os animais são acompanhados individualmente.
- Controle dos animais tanto a nível individual quanto genérico para fazendas onde os animais não possuem brincos.
- Controle do estoque e da movimentação de animais entre fazendas.
- Controle da produção de leite de forma global ou por animal.
- Controle de lotes confinados tanto a nível de custos quanto de manejo.
- Mudança de categoria animal automática, considerando idade e opcionalmente também o peso dos animais.
- Controle do estoque de sêmem e embriões.
- Controle de composição de sangue dos animais com cálculo automático do sangue dos filhos a partir dos pais.

- Integração com balanças digitais para leitura das pesagens individuais sem necessidade de digitação.

- Controle sanitário, reprodutivo e ponderal com recursos para entrada de dados múltiplas de forma otimizada em fazendas com grandes rebanhos.

- Controle de ocupação dos pastos com relatórios e gráficos.

- O gerenciador de tarefas, é um recurso interessante, pois gera a partir de um único comando, as ordens de serviço para as diversas tarefas a serem realizadas no dia-a-dia de sua fazenda.

- Registro das despesas e receitas ocorridas, de forma classificada pelo tipo de gasto e centro de custos destino de forma a permitir a qualquer momento a geração de gráficos e relatórios demonstrando como e onde se gastou dinheiro e foram geradas receitas em um período de tempo qualquer definido livremente pelo usuário.

- Fluxo de caixa da fazenda com gráficos e relatórios mostrando a situação dia-a-dia.

- Controle das contas bancárias.

3.4.5 PROLEITE - Sistema de acompanhamento e avaliação de rebanhos leiteiros

O PROLEITE (CNPGL, 2002) é um sistema desenvolvido com o objetivo de organizar as informações de desempenho produtivo e reprodutivo dos animais de rebanhos leiteiros. O sistema é um instrumento gerencial para orientar a tomada de decisão dos produtores, mediante análise de relatórios que apresentam indicadores técnicos de desempenho individual dos animais e indicadores de produtividade e eficiência reprodutiva de rebanhos.

O sistema é útil para instituições que cadastram e registram animais, executam os serviços de controle leiteiro e realizam o acompanhamento da produção de rebanhos leiteiros. Além disso, é um instrumento gerencial do produtor/criador para a sua orientação na tomada de decisões, no processo de planejamento das atividades de rotina dos sistemas de produção de leite.

A disponibilização do PROLEITE representa um marco inicial para solidificar o relacionamento das instituições patrocinadoras da pecuária leiteira nacional, além de contribuir para a consolidação das informações que convergem para o Arquivo Zootécnico Nacional de Gado de Leite. Contudo, sua maior importância fundamenta-se na sua grande contribuição para identificar e disseminar reprodutores capazes de gerar populações com maior potencial genético e auxiliar a gerência do processo produtivo para sua maior eficiência, fator estratégico para a melhoria da qualidade e competitividade dos produtos da agroindústria do leite. A seguir, estão disponíveis duas figuras dos processos disponíveis na aplicação.

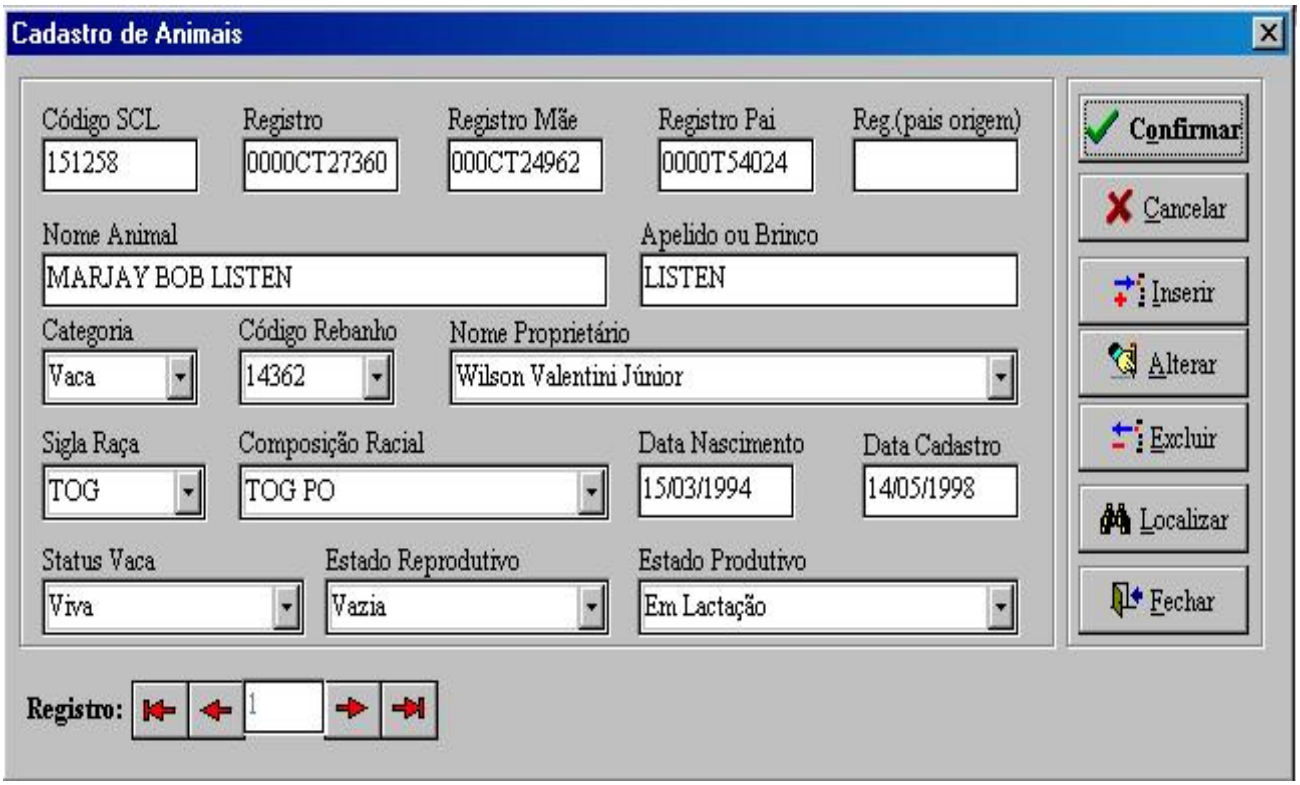

Figura 3.1: Cadastro de animais

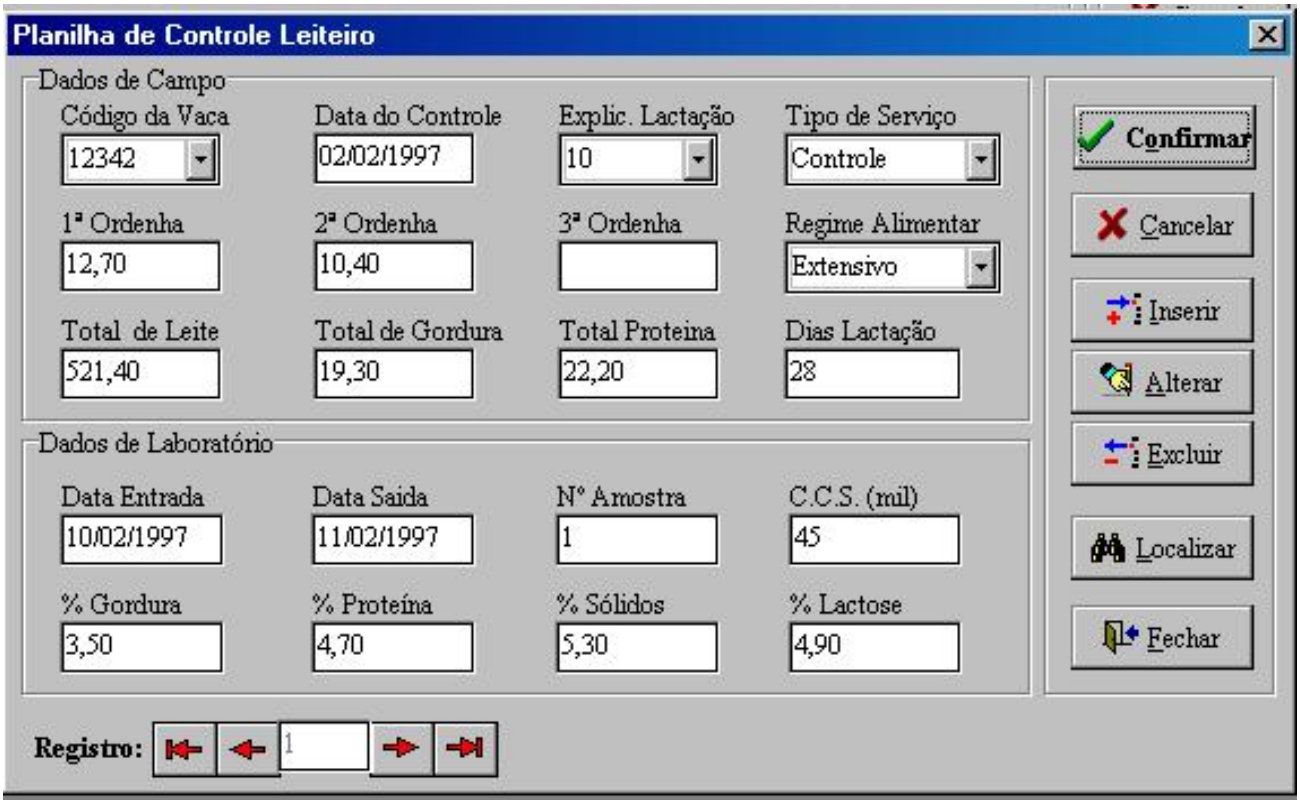

Figura 3.2: Controle leiteiro

3.4.6 Gestão da qualidade: aplicação para a suinocultura na Alemanha

Esse sistema foi apresentado no AGROSOFT 97, que é uma Feira e Congresso de Informática Aplicada à Agropecuária e Agroindústria, pelos senhores Aziz Galvão da Silva Jr., Gerhard Schiefer e Ralf Helbig (AGROSOFT, 1997).

O mercado europeu de carne suína é altamente competitivo. Os maiores produtores são a Alemanha, com 24,7 milhões de animais, seguido da Espanha, Holanda, França e Dinamarca. Apesar de ser o maior produtor, 21% do mercado interno da Alemanha, é abastecido por outros países da União Européia, como Dinamarca, Holanda e Bélgica/Luxemburgo (WINDHORST, 1995). A tendência histórica é de aumento de importações no mercado alemão, em prejuízo da suinocultura interna.

O sistema é composto por sete módulos divididos em três grupos. O primeiro grupo permite o acesso dinâmico a informações relevantes para a gerência da qualidade. Recursos de multimídia foram incluídos no sistema a fim de aumentar a eficiência na identificação de problemas e na implementação de ações preventivas e corretivas. Após a identificação de

um problema, este pode ser analisado pelo segundo grupo, que inclui também um módulo para cálculo do custo relacionado a qualidade. O último grupo permite a documentação das análises efetuadas em conformidade com exigências de normas internacionais da série ISO 9000. Testes em condições de campo e laboratório estão sendo conduzidos na Alemanha.

A partir da análise da questão da qualidade na suinocultura, foi desenvolvido um sistema computadorizado que auxilie o produtor rural a identificar e atender as exigências de seus clientes, aumentar a eficiência do processo produtivo, além de comprovar, através de documentação adequada, a qualidade de seus produtos e processos (SILVA Jr. & HELBIG & SCHIEFER, 1996; SILVA Jr. & HELBIG, 1997). Em cadeias de produção integrada, o sistema possibilita adicionalmente uniformizar e disponibilizar de forma rápida, eficiente e atual os conhecimentos gerados a partir da interação do extensionista com produtores rurais e com especialistas, contribuindo para o aumento da eficiência do trabalho extensionista.

Após análise detalhada da situação da suinocultura na Alemanha, iniciou-se o desenvolvimento do sistema a partir do método "partizipative Entwicklung" (HAUTZER & HELBIG & SCHIEFER, 1997). O método baseia-se na participação do usuário e de especialistas desde a fase inicial do processo. Protótipos do sistema, inicialmente na forma de esboço em papel, passando por protótipos descartáveis até chegar a um protótipo executável com todas as funções básicas do sistema, são avaliados pelos usuários e por especialistas. A participação do usuário é intensificada à medida que o sistema é desenvolvido. As fases podem ser divididas em:

- avaliação da demanda por informações e concepção do sistema;
- desenvolvimento do sistema e
- experimentação. Elementos de uma etapa anterior são repetidos nas fases posteriores, dependendo dos resultados da avaliação do usuário.

A gestão da qualidade total é o enfoque mais adequado, pois busca atender as exigências dos consumidores (internos e externos) e ao mesmo tempo aprimorar a qualidade do processo produtivo, através da diminuição do nível de erros. A qualidade total realiza-se na prática através da utilização sistemática, constante e em todos os níveis da empresa, de métodos para detecção e avaliação de exigências de clientes, prevenção de erros e controle do processo produtivo.

3.4.7 Aplicação para o gerenciamento da produção agrícola de grãos

Esta aplicação possibilita o gerenciamento da comercialização de produtos agrícolas de grãos, desde lançamentos de entradas e saídas, como emissão de nota fiscal, geração de gráficos estatísticos e relatórios da movimentação dos produtos. O funcionamento é feito pelo lançamento dos romaneios de pesagem do produto e o posterior faturamento (venda) conforme preço diário. Podem ser efetuadas vendas a prazo, dependendo da política da empresa. Os produtores que produzirem produtos com qualidade para armazenar o mesmo como semente, podem receber uma bonificação. Os usuários das aplicações são as empresas que comercializam esses produtos, bem como, possivelmente também produtores. As principais características são:

- cadastro de produtos;
- cadastro de produtores;
- cadastro das unidades (filiais) de comercialização (recebimento e faturamento) dos produtos;
- cadastro dos produtos para a unidade base de comercialização, sendo que para os demais, caso existirem, serão efetuados lançamentos de transferência dos produtos, armazenando assim automaticamente os dados do cadastro existente;

- definição da política de preços a serem pagos aos produtores ou clientes, conforme período de faturamento, tipo de produto entregue (para semente ou não);
- faturamento (venda) dos produtos pode ser feito com prazos e preços diferentes;
- lançamento das entradas de produtos do produtor para armazenamento é via os romaneios de pesagem;
- o faturamento do produto produzido pelo produtor ou cliente, é possível imediatamente após o lançamento de romaneios;
- faturamento (venda) de produtos, com a emissão de nota fiscal;
- relatórios de comercialização (entradas e saídas) dos produtos conforme períodos pré-determinados;
- geração de gráfico estatístico, referente aos produtos mais produzidos pelos produtores ou clientes;
- geração de gráfico estatístico mensal do faturamento de um determinado produto;
- geração de gráfico estatístico dos produtores que mais venderam seu produto num determinado período;
- geração de gráfico estatístico do volume de vendas diárias do produto;
- é possível também verificar o que a empresa revendeu para outras empresas;
- é possível também verificar quais os saldos disponíveis para revenda;
- é possível também verificar o que a empresa revendeu para outras empresas, mas ainda não foi retirado;
- é possível estornar lançamentos de entrada ou saída de produtos;

- efetuar transferência de produtos entre produtores e unidades;
- comprar o produto da empresa, ou seja, se um determinado cliente (produtor ou não) deseja comprar o produto da empresa e deixá-lo armazenado para posterior faturamento, também é possível;
- no caso do milho, para sua armazenagem, pode ser efetuado um desconto de quebra de armazenagem;
- no caso do trigo, são pagos os preços conforme o PH do produto. Portanto, a qualidade é medida com base nesse quesito.

3.4.8 Aplicação para comercialização de oleaginosos e cereais

Sistema que gerencia a comercialização de produtos agrícolas como soja, milho, trigo, canola e girassol. Esse sistema é destinado a empresas que atuam no ramo agrícola de grãos (DATACOPER, 2003). A seguir, um relato das principais operações executadas pelo sistema.

- cadastro de cliente, seja ele produtor ou comerciante;
- cadastro de produto agrícola;
- cadastro de operações;
- cadastro de filiais;
- pesagem do produto, efetuando os cálculos dos descontos sobre o produto e do peso do veículo, resultando num peso final a ser depositado na conta do cliente;
- faturamento do produto, ocorrerá quando o cliente desejar vender o produto depositado na empresa. Nesse processo será possível configurar a questão do desconto da porcentagem equivalente ao funrural (2,3%);

- transferência normal de produto de um cliente para outro, com a impressão de um comprovante da quantidade transferida;
- transferência normal de produto entre filiais da empresa, com a impressão de um comprovante da quantidade transferida;
- retirada do produto depositado, sendo possível configurar eventuais descontos;
- compra de produto, quando o cliente deseja comprar produto da empresa e deixar em depósito na mesma;
- controle do saldo do cliente.

3.4.9 AgroLeite - Manejo e controle do rebanho leiteiro

Sistema de gerenciamento e controle de rebanhos leiteiros. Possibilita a organização e monitoramento dos eventos ligados a reprodução, produção e controle sanitário. Integra e compartilha dados com um outro sistema, o AgroGestão. As principais características estão descritas a seguir (AGROSOFT, 2003).

- ✓ Controle do rebanho: ficha individual de animais com históricos detalhados, controle de lotes e categorias, resumo de rebanho com indicadores relacionados a manejo, reprodutivos e produtivos.
- ✓ Controle reprodutivo: estoque de sêmen, exames ginecológicos, inseminações e partos.
- ✓ Controle produtivo: lactações em andamento e encerradas, indicadores da qualidade do leite, produção total.
- ✓ Controle sanitário: exames, vacinas, medicamentos, controle de doenças e carrapatos.

- ✓ Relatórios de avisos: vacas a parir por mês e por período, vacas a secar, animais fora do padrão do lote, animais fora do padrão do rebanho.

- ✓ Base de parâmetros para classificação de lotes e raças e para padronização de peso e altura.

3.4.10 Aplicação para o controle leiteiro

Sistema para controle de rebanho leiteiro com as seguintes funções: registro individualizado, controle sanitário e clínico, movimentação de rebanho, desenvolvimento ponderal, controle reprodutivo, controle leiteiro, controle de exposições, estoques relatórios, consultas e gráficos (DATACOPER, 2003).

3.4.11 Aplicação para o controle bovino leite

WinBov é um auxiliar poderoso para a gestão técnica de efetivos bovinos-leite. Além de possuir fichas individuais de vacas, touros/sêmen e cria/recria, com informações detalhadas de cada um dos animais do efetivo, tais como a genealogia, descendência, imagem, estado atual, etc., também está apto a efetuar previsão de ocorrências, com base em parâmetros definidos pelo utilizador. Esta última particularidade do programa é de manifesta importância, dada a diversidade de critérios técnicos dos criadores. Possui relatórios recapitulativos do ponto da situação, com gráficos gerados automaticamente, tendo como base as ocorrências registradas, mas também de indicadores projetados, fato que é inovador em programas similares. Os animais são acedidos não só pelo "número da casa", mas também pelo "SIA", "LA", "LN," e "Nome", este último no caso dos "touros/sêmen". A aplicação permite ainda o registro dos contrastes oficiais e das produções diárias, com as duas ordenhas. O **WinBov** mantém também atualizada automaticamente uma tabela de existências, cuja listagem é idêntica aos livros oficiais de existências e deslocações de bovinos (SOFTIMBRA, 2003b).

3.4.12 Aplicação para o controle bovino de corte

O **WinBovC** é um auxiliar poderoso para a gestão técnica de efetivos bovinos-carne. Além de possuir fichas individuais de vacas, touros/sêmen e cria/recria, com informações detalhadas de cada um dos animais do efetivo, tais como genealogia, descendência, imagem, estado atual, etc., também está apto a efetuar previsão de ocorrências, com base em parâmetros definidos pelo utilizador. Esta última particularidade do programa é de manifesta importância, dada a diversidade de critérios técnicos dos criadores. Especialmente para os efetivos explorados extensivamente - sem controle dos saltos / cobrições - o programa dispõe de uma opção para atualização das ocorrências, sendo necessário indicar apenas a data de parto, determinando e registrando, o programa, o dia do serviço (SOFTIMBRA, 2003a).

Possui relatórios recapitulativos do ponto da situação, com gráficos gerados automaticamente, tendo como base as ocorrências registradas, mas também de indicadores projetados, fato que é inovador em programas similares.

O **WinBovC** mantém também atualizada automaticamente uma tabela de Existências, semelhante aos livros oficiais. Por esta razão, o programa foi homologado pelos serviços oficiais, podendo esta listagem substituir o preenchimento manual dos livros.

3.4.13 Sistema para o controle da produção de café

O **Siscafé2000 V-2** é um sistema de controle de custos e gerenciamento de dados relativos a produção do café, tendo sido planejado e construído para ser utilizado e analisado pelo produtor de café ou seu gerente técnico. A versatilidade deste sistema é completa em simulações para redução de custos sobre o que fora efetivamente realizado, fornecendo uma visão estratégica ao produtor. O objetivo principal visa estabelecer padrões para avaliações dos itens de controle de todas atividades cafeeiras e seus respectivos resultados, tais como custos diretos, investimentos, produções, produtividades, etc.

Apresentando os resultados em relatórios descritivos e gráficos, torna-se uma poderosa ferramenta para tomar as decisões corretas (SISCAFÉ, 2000).

3.5 Considerações finais

A partir das aplicações agropecuárias e informações contidas no presente capítulo, foi possível abstrair uma parte das características comuns a um determinado conjunto de aplicações e, conseqüentemente, definí-las como sendo parte do escopo de aplicações a serem produzidas na arquitetura do framework proposto.

Tendo em vista desenvolver uma arquitetura para um conjunto de aplicações agropecuárias distintas entre si, mas pertencentes ao mesmo domínio, ou seja, produtos agropecuários, foi proposto produzir no presente trabalho uma ferramenta capaz de auxiliar o processo de desenvolvimento dessas aplicações. Através dessa ferramenta, denominada framework, será possível o reuso de projeto e código, resultando em maior produtividade e flexibilidade no desenvolvimento das aplicações. As aplicações a serem produzidas pelos desenvolvedores de softwares agropecuários, poderão ser utilizadas por diferentes tipos de clientes, tais como agricultores, fazendeiros, engenheiros agronômos, veterinários, técnicos agrícolas e empresas do ramo agropecuário.

Com base no que foi abordado no presente capítulo (domínio de aplicações agropecuárias) e no anterior (contexto de frameworks), tem-se subsídios para atingir o objetivo, que é dispor de uma ferramenta que prove principalmente reuso de projeto e código. Diante disso, o próximo capítulo discrimina todo o processo envolvido na construção do framework PRODAGRO.

4 PRODAGRO: FRAMEWORK PARA GESTÃO E COMERCIALIZAÇÃO DE PRODUTOS AGROPECUÁRIOS

4.1 Projeto e implementação do PRODAGRO

4.1.1 Visão geral

Um framework é uma técnica poderosa para aumentar o reuso de software, visto que inúmeras aplicações distintas podem ser obtidas através do seu reuso. Entretanto, o processo de instanciação é, na maioria das vezes, muito complexo, exigindo conhecimento considerável a respeito do projeto e implementação do framework. Para alcançar a flexibilidade desejada, os frameworks contêm construções especiais que podem dificultar seu entendimento. Eles possuem uma parte fixa que reflete o comportamento comum do domínio e partes variáveis que devem permanecer flexíveis por representarem aspectos que mudam de uma aplicação para outra. Isso permite gerar produtos que podem ser estendidos para clientes com necessidades especiais.

Os frameworks possuem características de flexibilidades para prover uma necessidade de aplicação específica. Para tanto, eles mantêm flexível uma parte da estrutura de classes, cujo domínio é generalizado, de modo que a estrutura possa especializar-se para gerar diferentes aplicações. Isto possibilita ao usuário do framework determinar o comportamento da aplicação (SILVA, 2000).

Este capítulo vai apresentar os principais detalhes da modelagem do framework PRODAGRO.

4.1.2 Ferramentas utilizadas

O desenvolvimento do PRODAGRO ocorreu através do uso das seguintes ferramentas computacionais:

- ✓ **Ferramenta de modelagem UML:** o Rational Rose 2000, ferramenta produzida pela Rational Software Corporation, foi utilizado para criar e documentar o conjunto de classes que fazem parte do framework, bem como seus relacionamentos e colaborações. A opção por essa ferramenta, deve-se principalmente pelo conhecimento que o autor da presente dissertação já tinha da mesma e pela facilidade de uso que ela dispõe (RATIONAL, 2000).

- ✓ **Ambiente visual de desenvolvimento orientado a objetos:** o JBuilder 7.0 Personal, produto da Borland Software Corporation, foi usado na implementação do framework.

- ✓ **Linguagem de programação:** para desenvolver o framework e as aplicações sobre o mesmo, foi utilizada a linguagem de programação Java, por ser orientada a objetos, amplamente usada no mercado acadêmico e até comercial e, servir como um incremento no aprendizado do autor dessa dissertação. Visto que o mesmo praticamente não tinha conhecimento dessa linguagem.

4.1.3 Metodologia de desenvolvimento

Dentre as metodologias de desenvolvimento de frameworks existentes atualmente, o autor usou principalmente a metodologia de Johnson, ou seja, Projeto Dirigido por Exemplo (Example-Driven Design) (JOHNSON, 1993), o qual, atravessa as etapas de análise, projeto e teste. Outra metodologia usada foi o Projeto Dirigido por Hot Spot[6] (PREE, 1994), na qual os hot spots são as partes do framework mantidas flexíveis. A essência dessa metodologia é identificar os hot spots na estrutura de classes do domínio e, a partir disto, construir o framework (SILVA, 2000).

Todo o processo foi desenvolvido de forma iterativa e incremental (baseado no que sugerem o Processo Unificado (RUMBAUGH & BOOCH & JACOBSON, 1999) e outros processos baseados no mesmo, como o de Larman (1998)), ou seja, a cada ciclo de

[6] Em inglês é descrito como Hot Spot Driven Design. A expressão hot-spot foi mantida em Inglês e significa "parte flexível", significado este não obtido a partir de sua tradução literal.

desenvolvimento (iteração) um conjunto novo de problemas era analisado de forma que, no final da iteração, o *framework* incluísse novas funcionalidades (incremento).

Durante o desenvolvimento do *framework* foi bastante utilizada a técnica de *refactoring*. Essa técnica consiste em modificar o software sem introduzir novas funcionalidades, isto é, visa apenas aprimorar o projeto da solução para facilitar tanto a manutenção do software como a introdução de novas funcionalidades ao mesmo (FOWLER, 1999).

Para que a técnica de *refactoring* seja bem sucedida é importante que ela seja apoiada por testes de unidade (FOWLER, 1999). Testes de unidade normalmente são um conjunto de classes com métodos de teste elaborados para exercitar a funcionalidade de outras classes. Os testes de unidade ajudam a verificar se as modificações introduzidas no projeto, afetaram a funcionalidade esperada do software. Isso ajuda a encontrar os problemas de forma mais fácil (BECK & GAMMA, 1998).

No capítulo anterior, seção 3.4, foram descritas diversas aplicações que abrangem o domínio de produtos agropecuários e, com base nesses subsídios, foi possível seguir a metodologia baseada em exemplos e, adotar a técnica de *refactoring*.

Foi feita uma análise das aplicações existentes a fim de abstrair o conjunto de aplicações pertencentes ao domínio em questão, ou seja, gestão e comercialização de produtos agropecuários. A partir disso, foram abstraídas as funcionalidades comuns e implementadas no framework PRODAGRO. Além disso, também foram assimiladas abstrações já conhecidas, resultado da experiência de desenvolvimento do autor em projetos anteriores, no mesmo domínio de aplicações.

De acordo com Gamma (1995), a tarefa de desenvolvimento de um framework é difícil. Ele ainda complementa: "*If applications are hard to design, and toolkits are harder, then frameworks are hardest of all*". Ou seja, se aplicações são difíceis de projetar, e conjuntos mais ainda, então frameworks são os mais difíceis de todos.

A Figura 4.1 ilustra como foi o processo de desenvolvimento do framework PRODAGRO.

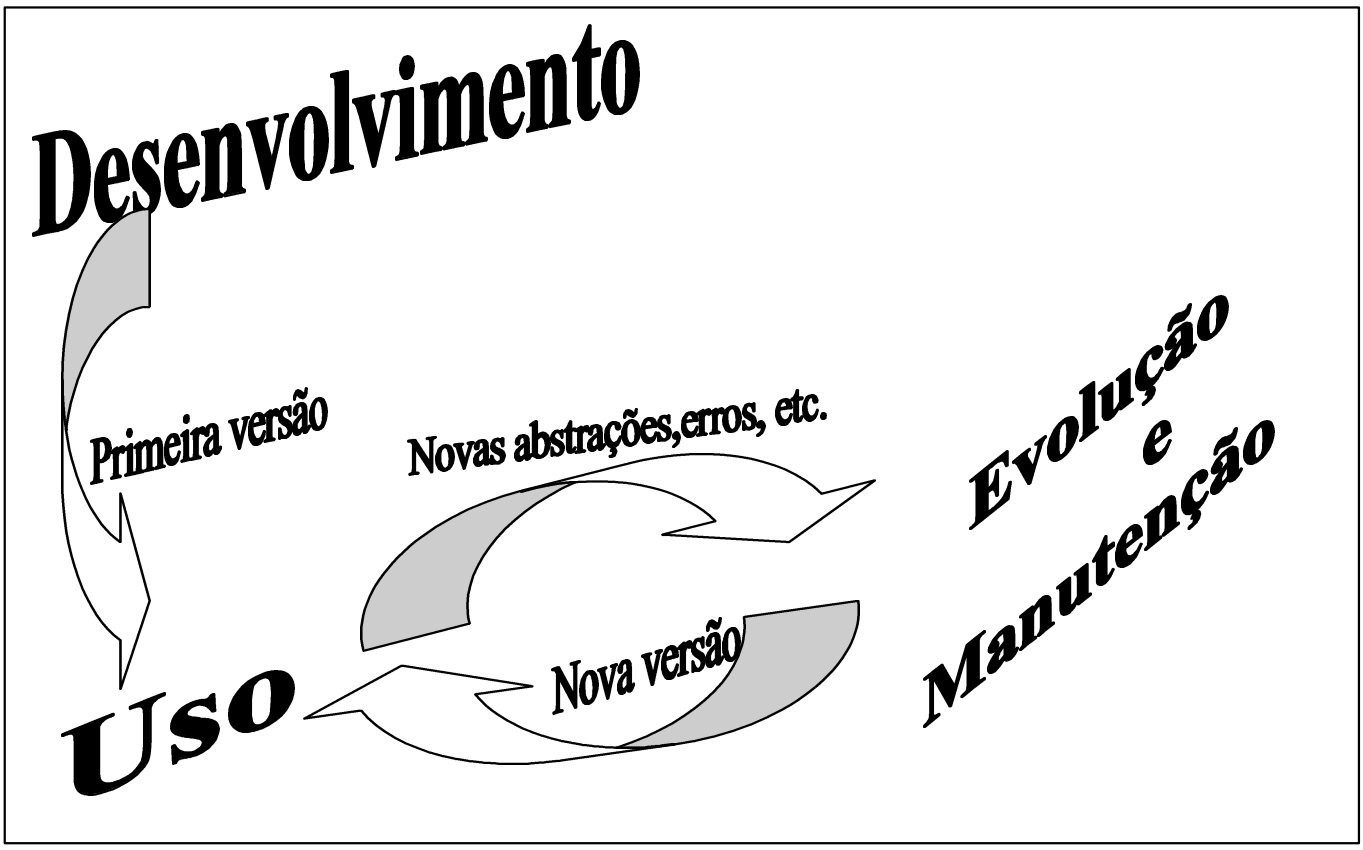

Figura 4.1: O processo de desenvolvimento de um framework (Fonte: DAMIAN, 1999)

4.1.4 Análise do domínio

O framework é a generalização de um modelo de domínio específico de aplicação e por essa razão deve conter a definição das classes que serão instanciadas para a construção da aplicação. O desenvolvedor tem autonomia limitada nesse processo pois o framework projetado pode conter classes que necessitem ser obrigatoriamente instanciadas ou ainda métodos que não possam ser sobrepostos. Este aspecto revela a importância da análise de domínio, pois deve capturar descrições que generalizem um conjunto de aplicações com similaridades de características.

O conjunto de peculiaridades envolvidas no contexto de aplicações para gestão e comercialização de produtos agropecuários é muito grande, resultando em um processo de análise bem abrangente e detalhado, procurando assim englobar todas as características que viriam a ser comuns dentro das aplicações. Para tanto, foi necessário um processo

amplamente interativo, ou seja, mesclando o estudo das aplicações, procurando discernir sobre o que era comum em todas as aplicações e o que seria uma particularidade, e o projeto de implementação do framework PRODAGRO. A figura da seção anterior reflete bem o processo de desenvolvimento do framework.

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

- **Empresa:** representa a generalidade do local onde a aplicação será utilizada, ou seja, propriedade rural, fazenda, unidade empresarial, etc.

- **Unidade Empresarial:** por que não filial? Porque em determinadas aplicações esse nome não abrangeria muito bem o contexto, como no caso, de a aplicação ser usada somente num determinado local. Com o nome filial, subentende-se que teria mais de uma. E, no caso de unidade empresarial, tem-se um escopo mais abrangente.

- **Bem:** tornou-se necessário a partir do momento em que se vislumbrou a necessidade de efetuar operações sobre valores (financeiro) e quantidades (físico) em diferentes situações. Essa abstração envolveu moeda e produto.

- **Cliente:** vai englobar diferentes tipos de pessoa física e pessoa jurídica. No contexto do domínio, pessoas físicas podem englobar produtores rurais, comerciantes, criadores de gado, etc. Com relação a pessoa jurídica, seriam todas aquelas que representam empresas.

- **Produto:** usado para generalizar os tipos de produto existentes, ou seja, produtos agrícolas, produtos pecuários e, os produtos derivados de ambos, assim como a composição de determinados produtos, representados no mundo real por um lote. Um exemplo claro disso seria um lote de bois.

- **Saldo:** é possível extrair a quantidade ou valor de saldo do produto de um determinado cliente, seja qual for a unidade de medida.

- **Operação:** são as operações que acontecem nas aplicações. Dentre elas se enquadram, depósito, venda, compra, transferência de clientes, transferência de unidade empresarial, transferência de local, exportação, importação, pesagem, retirada, devolução, etc.

- **Documento:** pertencem ao conjunto denominado de fiscal e não-fiscal. Fazem parte do primeiro item basicamente a nota fiscal e cupom fiscal. Do segundo tipo, o romaneio, contrato e recibo.

- **Fator de Cálculo:** foi criado com o objetivo de generalizar o comportamento envolvido em taxa e fator de qualidade.

- **Taxa:** representa aquilo que deve ser aplicado no momento de executar alguma operação. Ela se divide em dois tipos, a de crédito e débito. A primeira tem a função de adicionar alguma informação[7], ao passo que a segunda diminui informação. Um exemplo de uma taxa de crédito, seria o frete a ser pago ao cliente no momento da venda do produto. Em contrapartida, uma taxa de débito seria o funrural a ser pago pelo produtor na venda do produto.

- **Fator de Qualidade:** permite identificar quais as qualidades que devem ser avaliadas no momento de executar uma operação. Por exemplo, umidade e impureza, quando for uma operação de depósito de produto agrícola. Já quando for uma aplicação de controle da produção leiteira, deve-se avaliar fatores como gordura e proteína.

- **Tipo de Cálculo:** abstrai o contexto envolvido em cálculos que deverão ser aplicados sobre um valor, seja ele financeiro ou físico. Ou seja, possibilita associado a outros objetos, principalmente do domínio de fator de cálculo, comportar-se como porcentagem ou absoluto.

[7] Entende-se por informação aquilo que é valor ou quantidade (peso).

- **Espécie:** essa classe abrange principalmente o contexto envolvido em produtos pecuários. A especialização da mesma é possível através de classes como por exemplo, bovino, suíno, eqüino, ovino, ave, etc.

- **Usuário:** proprietário de propriedade rural, empresa de comercialização de produtos agropecuários, empresas que comercializam produtos derivados, engenheiros agrônomos e veterinários, técnicos agropecuários, fazendeiros, enfim qualquer usuário envolvido com o contexto de gerenciamento da comercialização de produtos agropecuários.

- **Interface Gráfica 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.

4.1.5 Casos de uso resultantes da abstração dos conceitos do domínio

Após ter feito um estudo do escopo de aplicações a serem abrangidas pelo framework a ser produzido e, conseqüentemente uma análise dos principais comportamentos e requisitos que o mesmo teria, foi possível definir os casos de uso que estariam envolvidos no desenvolvimento do PRODAGRO. Cabe ressaltar que esses casos de uso são basicamente interações entre o usuário da aplicação e o seu cliente. A seguir, os mesmos estão descritos.

- **Criar uma taxa:** as taxas estão vinculadas diretamente a operação, portanto serão executadas no momento em que acontecerá um determinado processo, como por exemplo, faturar um produto agropecuário (soja) ou vender um produto agropecuário (soja). A taxa deve ser vinculada a nenhum, um ou uma lista de produtos, a nenhum, um ou uma lista de clientes, a um tipo de cálculo (porcentagem ou absoluto), se o cálculo deve ser feito sobre o valor ou sobre a quantidade, se a taxa adiciona ou subtrai e, por fim, em qual operação ela estará vinculada. Um exemplo de taxa de débito é o **funrural**

na fatura ou venda do produto. Já como exemplo de taxa de crédito, pode-se citar um frete pago ao cliente na armazenagem do produto (quando acontece a colheita).

- **Criar um fator de qualidade:** é necessário para verificar a qualidade do produto que entra ou sai da empresa. Sempre que houver uma operação depósito, transferência de armazém ou retirada do produto, devem ser verificados os fatores de qualidade.

- **Inserir item no documento:** quando ocorrer qualquer operação na aplicação deve-se criar um item e agregá-lo ao documento que está sendo criado.

- **Depositar produto agropecuário de comerciante:** quando um comerciante trouxer seu produto para depositar na empresa, o usuário da aplicação deve verificar se o produto é próprio ou de terceiros, no caso de algum(s) produtor (es).

- **Depositar produto agropecuário de produtor (pessoa física):** ocorre quando o produtor quer deixar o produto no armazém da empresa.

- **Faturar produto agropecuário:** quando o cliente quer vender o produto que tem depositado na empresa. Quando o cliente for um comerciante e o produto está vinculado a um produtor, a nota fiscal de fatura deve ser efetuada para o produtor.

- **Vender produto agropecuário:** quando a empresa vende o produto depositado pelos clientes para outras empresas (pessoa jurídica).

- **Comprar produto agropecuário:** quando o cliente compra produto pertencente a empresa.

- **Exportar produto agropecuário:** quando a empresa exporta produto para outros países.

- **Pesagem do produto agropecuário:** ocorre quando o cliente traz o produto para depósito ou quando a empresa quer transferir o produto para um outro armazém ou unidade empresarial, ou ainda quando o cliente quer retirar seu produto

- **Retirar produto agropecuário:** deve-se pesar o veículo primeiro e, a seguir, o produto. Verificar ainda os fatores de qualidade, determinando seus valores. A aplicação verifica se existe alguma taxa a ser calculada. Por fim, tem-se os dados para gerar o documento.

- **Criar preço:** deve ser executado para as operações e produtos existentes na aplicação, pois de acordo com a operação o produto tem seu respectivo preço.

- **Atualizar preço:** corresponde a atualização do preço do produto, caso tiver algum indexador configurado para o mesmo, ou seja, tem-se um indexador para atualizar os preços diariamente.

- **Atualizar saldo do cliente:** toda vez que for executada uma operação, deve-se atualizar ou não o saldo do produto e cliente.

- **Definir categoria:** cada espécie tem um conjunto de categorias, nas quais é possível determinar qual a faixa etária que a mesma se enquadra, ou seja, se a espécie é bovino, suas categorias são, bezerro, bezerra, garrote, boi, vaca, etc.

- **Transferir produto para uma unidade (filial) da empresa:** deve-se pesar o veículo primeiro e, a seguir, o produto. Verificar ainda os fatores de qualidade, determinando seus valores. A aplicação verifica se existe alguma taxa a ser calculada. Por fim, tem-se os dados para gerar o documento.

- **Transferir produto agropecuário para uma pessoa física:** ocorre quando um cliente tipo pessoa física quer transferir um produto armazenado na empresa para uma outra pessoa física.

- **Criar um lote de compra (produto composto):** usado para compor um lote de animais oriundos de uma compra. Quando o proprietário compra animais de um outro produtor rural ou fazendeiro.

- **Criar um lote próprio (produto composto):** usado para compor um lote de animais oriundos da própria propriedade.

- **Vender lote:** quando o fazendeiro, proprietário ou produtor rural quer vender um lote de animais.

- **Gerar um documento:** quando acontecer uma operação na aplicação, é gerado um documento.

4.1.6 Projeto de estrutura de classes

De acordo com as abstrações obtidas através do estudo das aplicações envolvidas na construção do framework, tornou-se possível iniciar o projeto da estrutura de classes que compõem o PRODAGRO. Cabe ressaltar, que houve várias versões diferentes de estruturas de classes até se chegar a versão final. Mas, tudo isso se fez necessário, para uma representação melhor do conjunto de aplicações a serem envolvidas no desenvolvimento do framework, bem como o amadurecimento do autor com relação à produção de um framework.

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 na presente dissertação, as classes redefiníveis serão representadas com a letra «R» acima do nome da classe; as redefiníveis e essenciais serão representadas com «RE»; as abstratas terão o nome em itálico e as concretas serão representadas normalmente de acordo com as notações UML. Já os métodos hook, estão representados pelo esteriótipo «H» e os métodos template pelo «T».

O PRODAGRO 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. Através dos atributos e métodos disponíveis em cada classe da arquitetura disponível nos diagramas, será possível obter uma idéia acerca das funcionalidades da mesma e, da interação existente entre os objetos que compõem os diagramas. Toda essa interação ocorre, na abordagem de orientação a objetos, através da troca de mensagens proporcionadas pelos métodos das classes.

4.1.7 Estrutura arquitetural do PRODAGRO

A partir dessa seção, será apresentado passo a passo o conjunto de classes produzidas no framework PRODAGRO. Cabe ressaltar também, que será mostrado a interação que acontece entre os objetos das principais classes, bem como os relacionamentos entre as classes do framework. A seguir, será apresentado o esqueleto principal da arquitetura de classes que foi implementada no framework PRODAGRO.

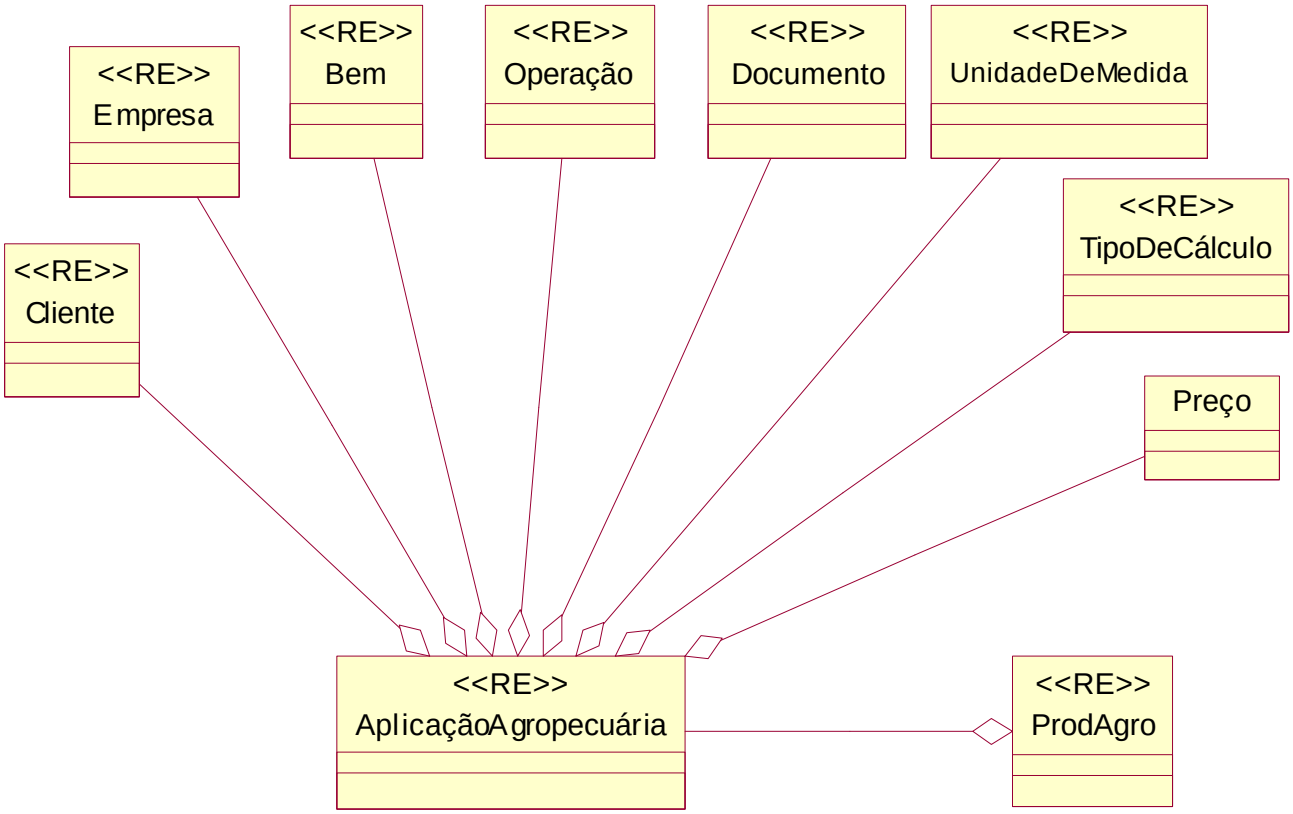

Figura 4.1: Arquitetura principal do framework PRODAGRO

Com base no que está demonstrado na Figura 4.1 é possível ter uma idéia inicial acerca daquilo que está envolvido na construção do framework, dando assim uma visão geral do PRODAGRO. Toda a aplicação que for desenvolvida sob o framework deverá especializar duas classes do framework. Uma delas, é a classe *ProdAgro,* responsável por centralizar todo o controle de inicialização de qualquer aplicação agropecuária. A outra a ser especializada, é a classe *AplicaçãoAgropecuária,* onde está centralizado todo o modelo construído para abranger as características do domínio de aplicações para gestão e comercialização de produtos agropecuários. Na classe concreta resultante da segunda especialização, deverá ser definido um método que será responsável pela visualização da interface produzida para a aplicação e, que permitirá a interação do usuário com a mesma. Ou seja, baseado naquilo que a aplicação deve fazer, o desenvolvedor da aplicação precisa construir uma interface que atenda as necessidades de seu cliente, usando os recursos disponíveis no framewok. Isso deverá ser feito através da invocação de métodos das classes.

Para obter um entendimento melhor do que está envolvido em cada uma das demais classes da arquitetura principal do framework (*Cliente, Bem, Empresa, Documento, Operação, FatorDeCálculo, UnidadeDeMedida e Preço*), optou-se por discriminar suas principais funcionalidades ou comportamentos em itens separados, dispostos a seguir.

- **Empresa:** a estrutura de classes apresentada a seguir, mostra uma visão particular do objeto *Empresa*. Ele representa a abstração do contexto de propriedades rurais ou fazenda e de unidades empresariais, envolvidas nas aplicações do domínio de produtos agropecuários. Cada instância das subclasses de *Empresa*, tem características próprias, onde o objeto *UnidadeEmpresarial* abrange basicamente as aplicações desenvolvidas para empresas do ramo agropecuário. Ao passo que, o objeto *Propriedade*, representa mais especificamente os conceitos de campo, fazenda e propriedade rural. A classe *Empresa*, igualmente mantém um relacionamento de agregação com um objeto *InscriçãoEstadual*, sendo que uma instância do mesmo é parte da instância de um objeto tipo *Propriedade* ou *UnidadeEmpresarial*. Tem-se ainda a classe *Propriedade* que é composta de um conjunto de objetos do tipo *Local*.

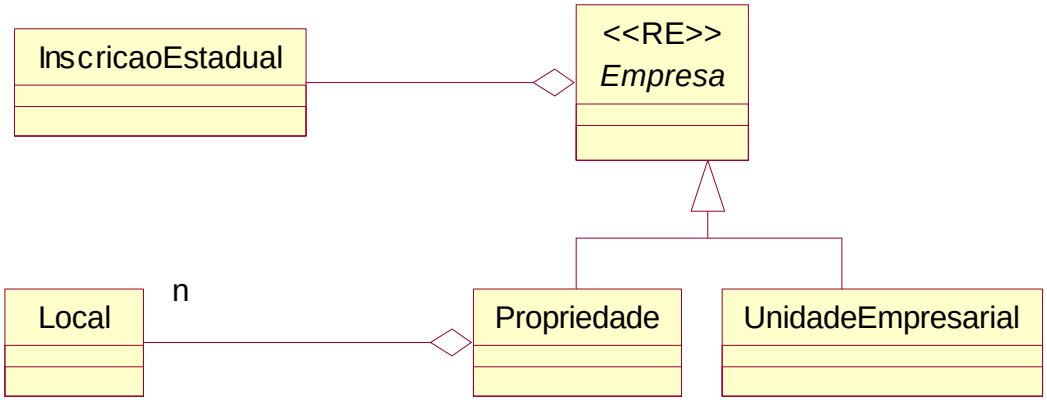

Figura 4.2: Estrutura de classes vinculada a classe Empresa

- **Cliente:** através da estrutura de classes ilustrada abaixo, é possível ter uma noção do que está envolvido na instanciação de objetos tipo *Cliente*. Percebe-se que há duas especializações de *Cliente*, uma que representa objetos do tipo *PessoaJurídica* e, a outra que representa os de *PessoaFísica*. Seguindo, visualiza-se a especialização da classe *PessoaFísica* em duas outras, representadas no domínio em questão por *Produtor* e

Comerciante. Sendo que o relacionamento existente entre ambas, permite identificar que objetos do tipo *Produtor* podem estar vinculados a diversos objetos tipo *Comerciante*, bem como, um objeto *Comerciante* pode ter vínculo com diferentes objetos tipo *Produtor*. Cada um dos objetos *Cliente* pode ter agregado vários objetos tipo *InscriçãoEstadual*, que por sua vez, está associado a um objeto *Cep*. Com isso, fica claro que um objeto *Cep* pode estar associado a diferentes objetos tipo *InscriçãoEstadual*. Outro relacionamento apresentado, é a associação entre objetos *Cliente* e, um objeto *Classificação*, através do qual é possível enquadrar cada objeto *Cliente* a uma determinada *Classificação*. Também pode-se definir comportamentos específicos a cada especialização da classe *Classificação*.

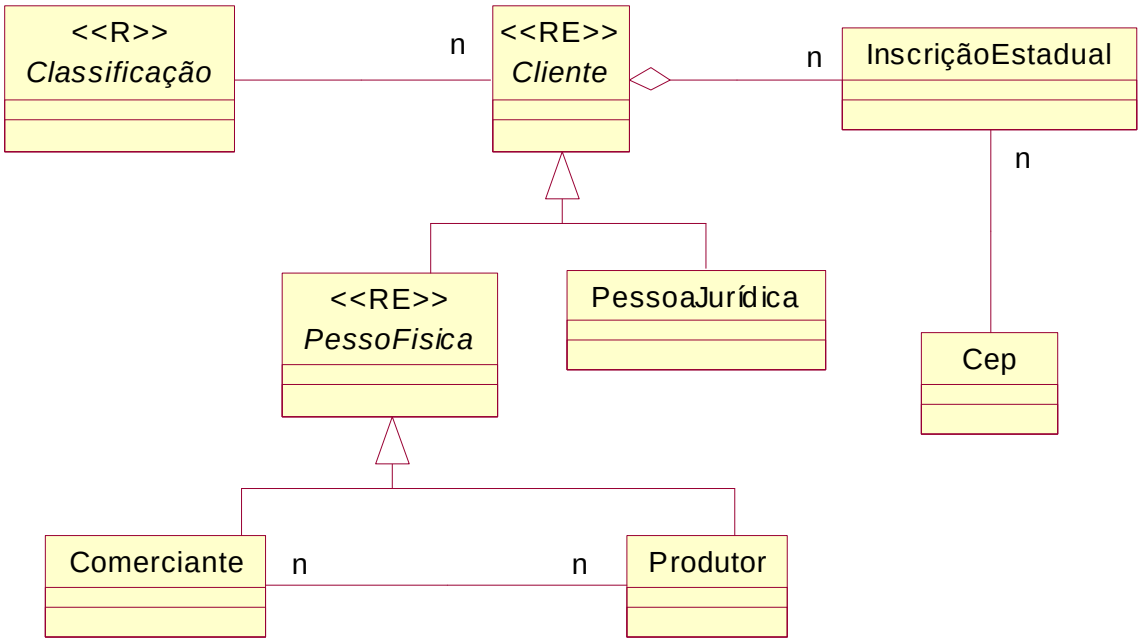

Figura 4.3: Estrutura de classes vinculada a classe Cliente

- **Bem:** por meio dessa classe foi possível abstrair comportamentos muito importantes no processo de execução de uma aplicação agropecuária, especializados pelas subclasses *Moeda* e *Produto*. O comportamento comum em ambas é representado pelos métodos abstratos *retornar(), subtrair() e adicionar()*. Suas funções são respectivamente: retornar o valor correspondente a uma determinada taxa, de acordo com alguns parâmetros (tipo de cálculo, valor da taxa, valor total e quantidade); o segundo método é responsável pela subtração do valor correspondente a uma *Taxa* e o último, efetua a operação de soma do valor da taxa. Na especialização das classes *Moeda* e *Produto*, coube a primeira atuar sobre

o valor financeiro e a segunda sobre o valor físico (quantidade). A classe *Produto,* é especializada por outras classes, sendo elas respectivamente *ProdutoAgrícola*, *ProdutoPecuário, ProdutoComposto* e *ProdutoDerivado*. A primeira engloba as características pertinentes aos produtos agrícolas, a segunda apresenta as características de produtos pecuários (animais), a terceira surgiu pela necessidade de trabalhar com lotes de produtos e, por fim, a última engloba os produtos derivados dos agropecuários. Para tratar do comportamento da composição de produtos da classe *ProdutoComposto,* foi usado o padrão de projeto *Composite,* o que facilitou o processo de implementação dos objetos da classe. A associação existente entre a classe *Produto* e a classe *Espécie*, determina que diferentes objetos *Produto* podem estar vinculados a um objeto *Espécie*. Assim como, um objeto *Espécie* conter uma agregação de objetos tipo *Categoria*. Na especialização da classe *Espécie* é possível definir suas especificidades, sendo que o mesmo também vale para a classe *Categoria*. Com relação a classe *ProdutoPecuário*, percebe-se que a mesma dispõe de dois vínculos, um via associação com a classe *Raça*, e o outro por agregação de um objeto tipo *Finalidade,* especializado por sua vez em *Reprodução*, *Leite* e *Corte*. Todo o processo é visualizado pela Figura 4.4.

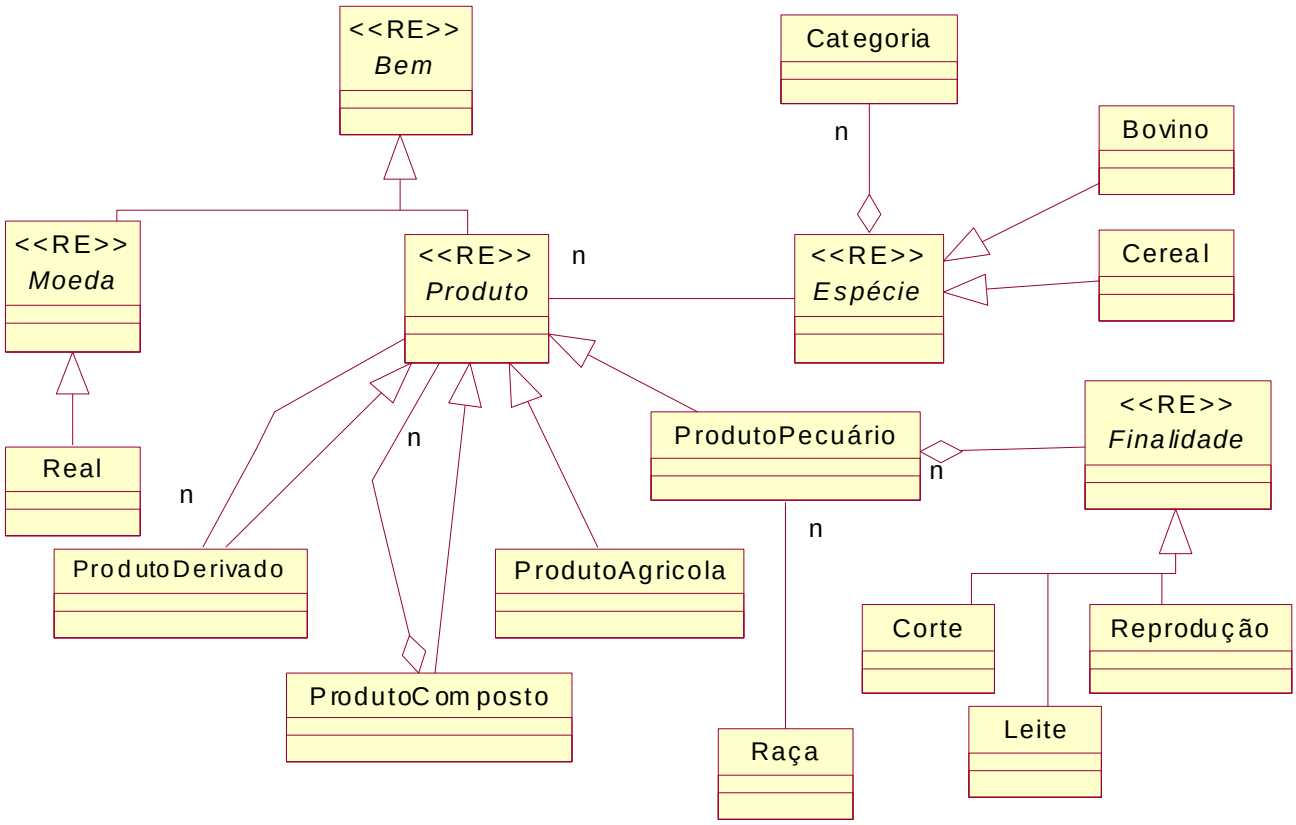

Figura 4.4: Estrutura de classes vinculada a classe Bem (Produto e Moeda)

- **Operação:** é uma das classes mais importantes da arquitetura de classes do framework PRODAGRO, por dispor de comportamentos vitais para a instanciação de uma aplicação. Por ser genérica, ela possibilita especializar seus comportamentos, como por exemplo, do método *gerarDocumento()*. Através do qual é possível gerar o documento pertinente a *Operação* especializada. Por exemplo, numa operação de depósito de produto agrícola de grão, deve-se primeiramente gerar o *Romaneio* de pesagem do produto, para depois gerar a *NotaFiscal* correspondente a mesma. Além disso, outros comportamentos são igualmente importantes para a execução das operações da aplicação, tais como: invocar o método responsável pelo retorno do valor de um tipo de objeto *Taxa* e, invocar o método responsável pelo cálculo do valor de um objeto tipo *FatorDeQualidade*. Outro comportamento relevante, é de que um objeto do tipo *Operação* pode conter um conjunto de objetos *TabelaFiscal*, sendo que a aplicação vai usar sempre o último objeto do vetor. Com isso, quando o governo for alterar características fiscais de uma determinada *Operação*, basta instanciar um objeto do tipo *TabelaFiscal* e adicioná-lo à *Operação*

vigente, mantendo o histórico de objetos tipo *TabelaFiscal* na respectiva *Operação*, proporcionando, caso for necessário, de um eventual reuso. Além disso, a operação também é responsável por invocar o método de atualização do saldo do cliente. A Figura 4.5 ilustra basicamente o que fora descrito.

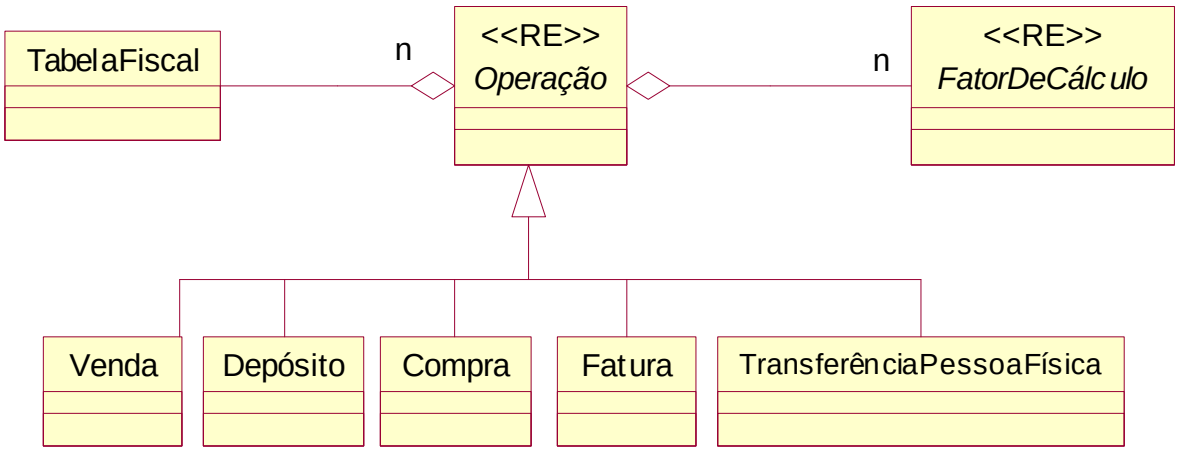

Figura 4.5: Estrutura de classes vinculada a classe Operação

- **Documento:** esta classe abstraiu o contexto envolvido em documentos fiscais e não fiscais e, com isso, projetou-se uma arquitetura de classes baseada em objetos do tipo *DocumentoFiscal* e *DocumentoNãoFiscal*. O primeiro tipo especializa objetos como *NotaFiscal* e *CupomFiscal*. Já o segundo, especializa objetos do tipo *Romaneio*, *Recibo* e *Contrato*. Todo o objeto do tipo *Documento*, comportará algumas características fundamentais, quais sejam: a associação a um objeto tipo *Cliente*, uma associação com um objeto do tipo *Operação*, uma associação com um objeto do tipo *Empresa* e uma agregação de objetos do tipo *Item*, que por sua vez mantêm uma associação com objetos do tipo *Bem*. A Figura 4.6 apresenta a estrutura de classes envolvida em objetos tipo *Documento*.

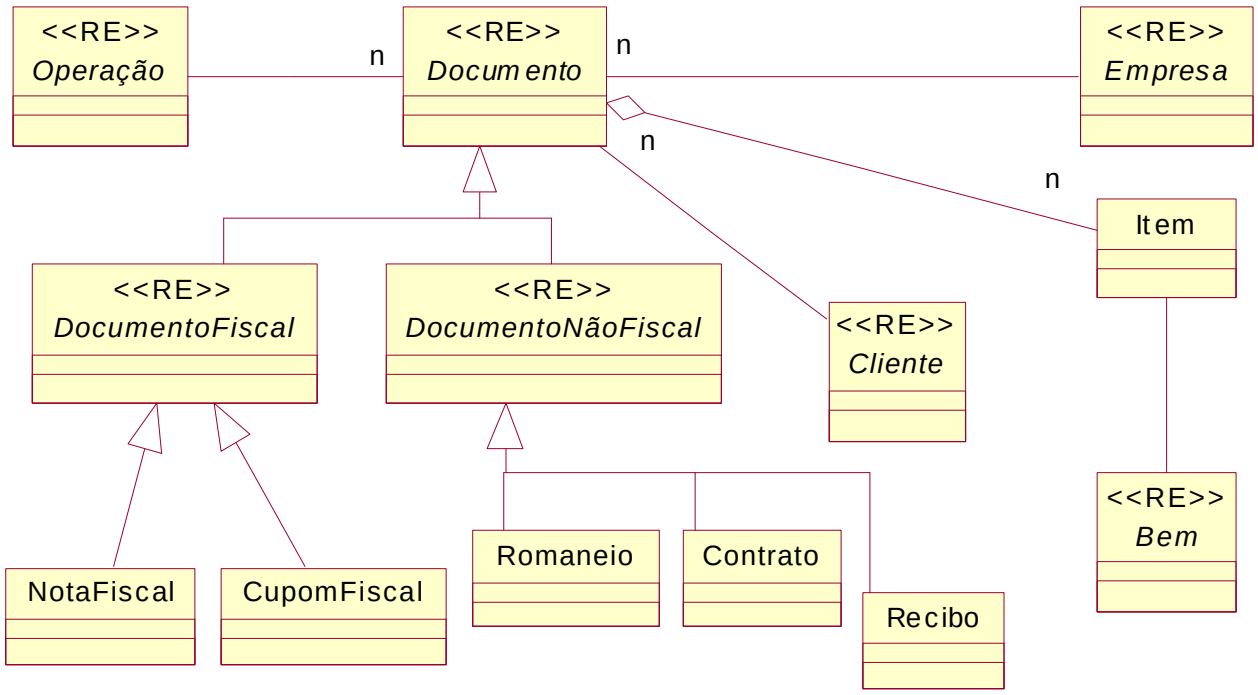

Figura 4.6: Estrutura de classes vinculada a classe Documento

- **FatorDeCálculo:** a arquitetura de classes apresentada na Figura 4.7, representa a abstração de comportamentos envolvidos nas classes *Taxa* e *FatorDeQualidade*. Esse tipo de objeto atua quando estiver sendo executado alguma operação sob a aplicação. Os objetos do tipo *FatorDeCálculo*, são agregados por objetos do tipo *Operação*. O comportamento dos objetos tipo *FatorDeCálculo* é determinado por algumas características comuns e outras particulares. Dentre as comuns, pode-se destacar a associação com os objetos *TipoDeCálculo*, representados pela especialização das classes *Absoluto* e *Porcentagem*. Além disso, também deve-se definir a fórmula de cálculo a ser exercida, ou seja, se for sobre o valor ou sobre a quantidade, através de um objeto do tipo *Bem*. Por fim, a associação com objetos do tipo *Produto*, serve para determinar quais os produtos a serem abrangidos pela mesma, podendo variar entre nenhum, vários ou todos. Já com relação as particularidades de cada classe, faz-se necessário relatar as principais. Os objetos do tipo *Taxa*, podem ser tanto de *Crédito*, quanto de *Débito*, haja visto que o seu comportamento está encapsulado nos métodos *subtrair()* e *adicionar()*. Também é importante salientar uma característica fundamental em objetos do tipo *Taxa*, que é a associação da mesma com a classe *Cliente*, sendo possível determinar em quais objetos do tipo *Cliente* a mesma deve

ser aplicada. Nos objetos do tipo *FatorDeQualidade*, cabe ressaltar características como a agregação de um objeto tipo *TabelaDeQualidade*. Essa classe é útil para, por exemplo, calcular o desconto de um objeto desse tipo, pois deve-se invocar o método *buscaValor()*, que busca o valor correspondente ao mesmo. O comportamento dos objetos tipo *FatorDeQualidade* é definido no momento de especializar uma classe, bastando definir quais dos métodos deve ser executado, o *subtrair()* ou *adicionar()*.

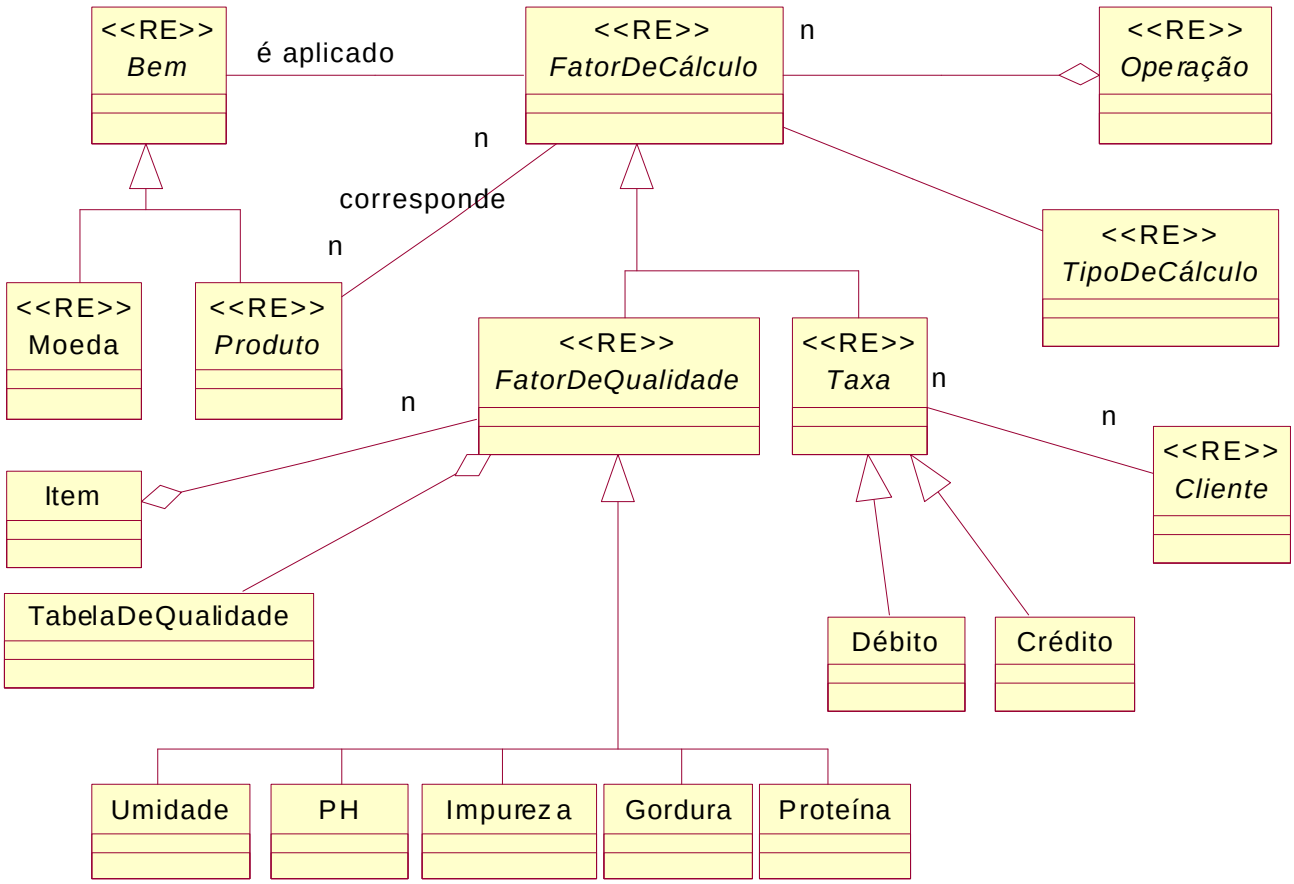

Figura 4.7: Estrutura de classes vinculada a classe FatorDeCalculo

- **UnidadeDeMedida:** essa classe tem através do método abstrato *calcular()*, inserida uma flexibilidade interessante, pois através do mesmo pode-se obter a informação contida em diferentes documentos numa única unidade de medida. Por exemplo, se a aplicação tiver um documento com uma instância de objeto tipo *Quilo* e a outra em *Saca*, pode-se obter o resultado tanto em um quanto em outro (quilo ou saca). A Figura 4.8 ilustra a abstração existente na classe *UnidadeDeMedida*.

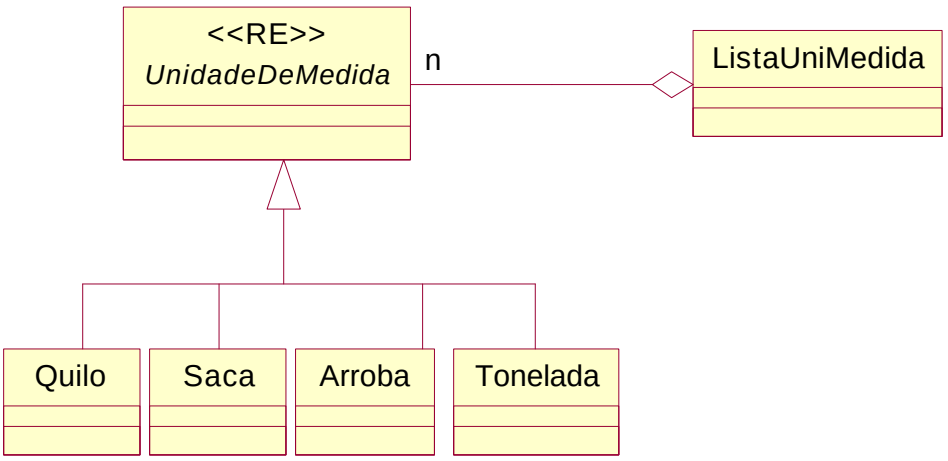

Figura 4.8: Estrutura de classes vinculada a classe UnidadeDeMedida

- **Preço:** a arquitetura de classes da Figura 4.9 permite vislumbrar o processo envolvido na criação de objetos do tipo *Preço*. Cada objeto criado, caracteriza-se por estar associado a outros objetos, como do tipo *Bem*, *Moeda*, *Empresa*, *UnidadeDeMedida*, *Operação* e, pela agregação de objetos do tipo *Indexador*. Este por sua vez, pode ser definido de acordo com objetos do tipo *TipoDeCálculo* (*Porcentagem* ou *Absoluto*) e, por uma *Moeda*. Essa flexibilidade é interessante a partir do momento que o usuário da aplicação queira ter somente um preço base e diariamente usar os indexadores de preço. Para tanto, basta usufruir do método *atualizarPreço()*, que suporta tal comportamento.

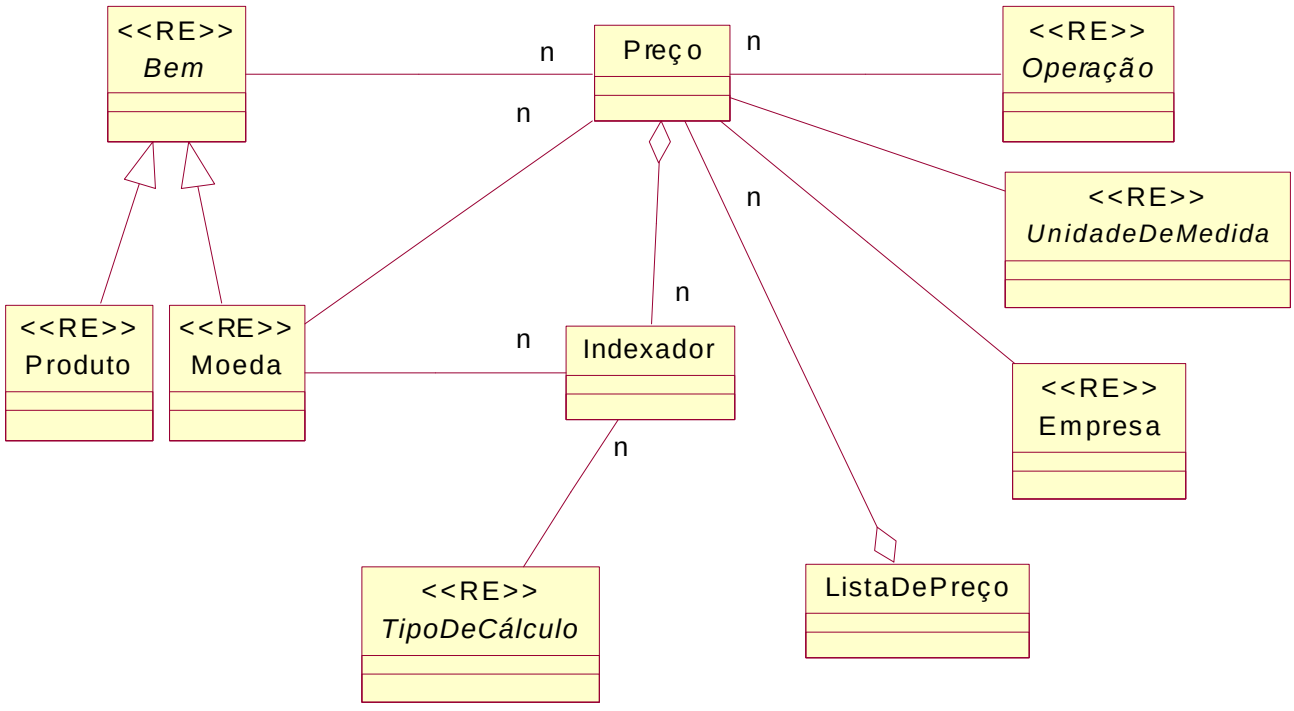

Figura 4.9: Estrutura de classes vinculada a classe Preço

4.1.8 Produzindo aplicações agropecuárias

A classe abstrata *ProdAgro*, contém um comportamento bem definido, o qual é representado pelo método template *inicializar()*. Seu algoritmo comporta a invocação de um método hook, representado pelo método *gerarAlpicação()*.

A implementação do método *gerarAplicação()* do framework PRODAGRO, adota o padrão de projeto *AbstractFactory* (GAMMA, 2000). Em função disso a única responsabilidade do método que deve ser definido na especialização da classe *ProdAgro*, é chamar o método construtor da classe correspondente a especialização da classe *AplicaçãoAgropecuária*. A figura Figura 4.1 ilustra a aplicação do padrão de projeto AbstractFactory no framework, produzindo apenas um acoplamento entre as subclasses de *ProdAgro* e *AplicaçãoAgropecuária*, e não o contrário.

A seguir, através da ilustração da Figura 4.1 e Figura 4.2, será possível vislumbrar o que o usuário do framework ou desenvolvedor de aplicações, deverá fazer para produzir uma aplicação agropecuária.

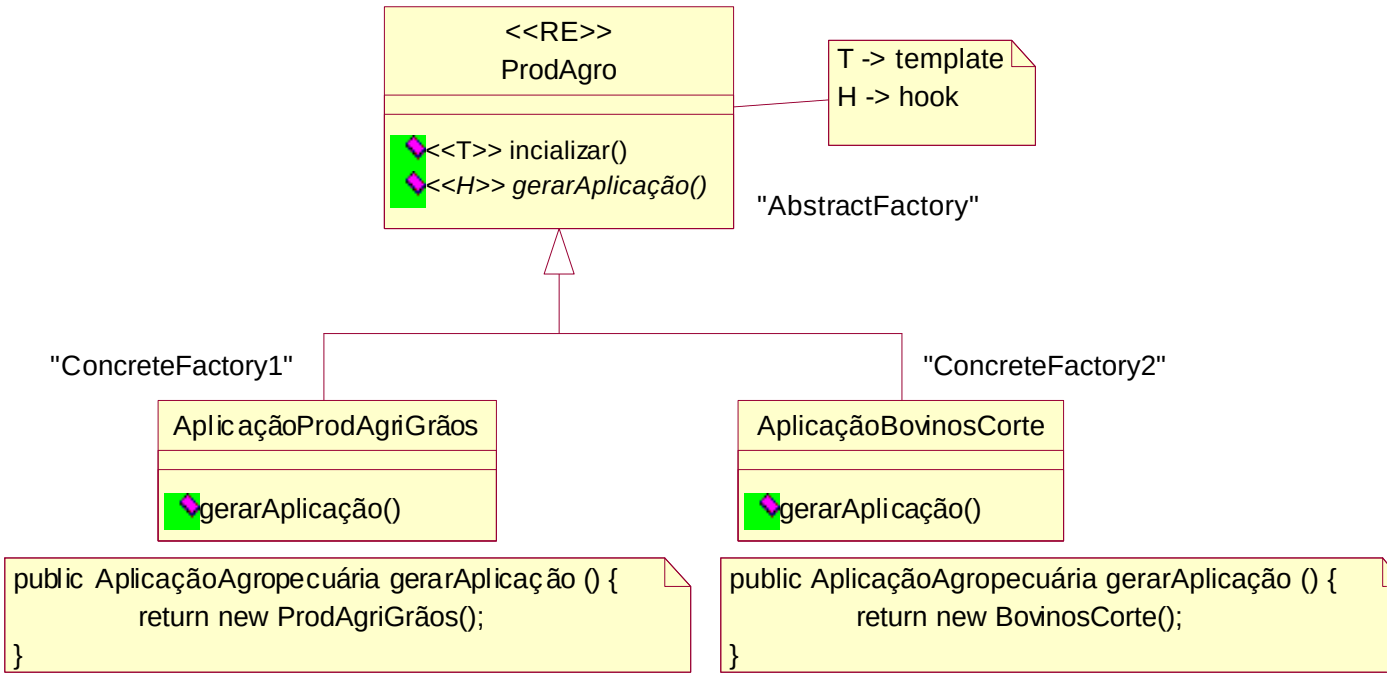

Figura 4.1: Aplicação do padrão de projeto Abstract Factory

Num segundo momento da produção de uma aplicação, temos o comportamento envolvido na classe abstrata *AplicaçãoAgropecuária*, representado pela Figura 4.2, que também utiliza o metapadrão unificação para a sua inicialização, já que tem um procedimento comum para qualquer aplicação agropecuária que desejamos criar. Na instanciação de uma aplicação agropecuária será invocado o método template chamado *inicializar()*. Esse método possui um algoritmo genérico que invoca diversos métodos hook para flexibilizar os aspectos particulares na criação de diferentes aplicações. O método hook *criarInterface()*, obriga o desenvolvedor da aplicação a criar uma interface para a aplicação, a qual possibilitará a interação do usuário com a mesma.

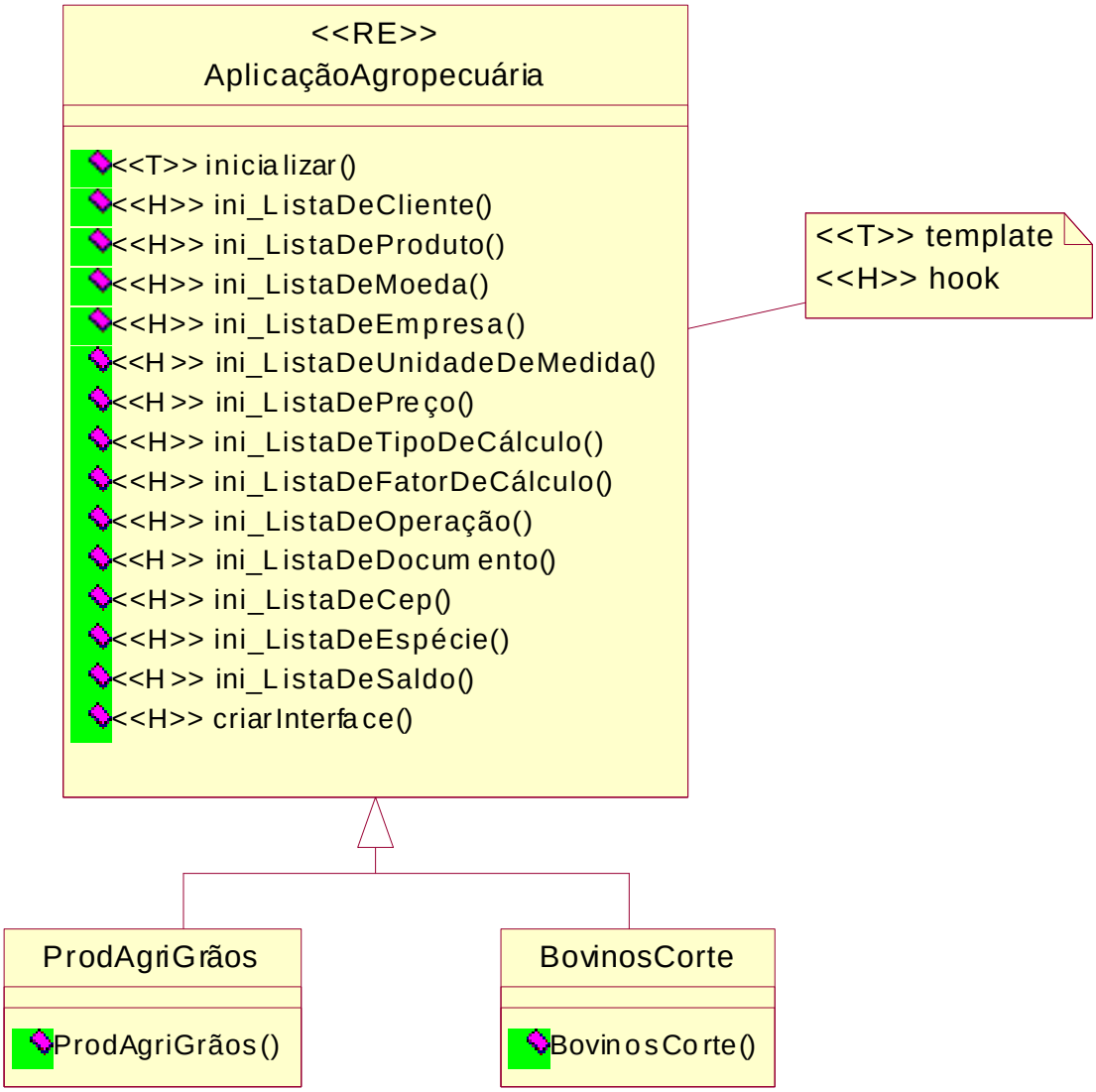

Figura 4.2: Estrutura principal da classe AplicaçãoAgropecuária

O procedimento apresentado na Figura 4.1 representa o comportamento genérico da inicialização de qualquer aplicação agropecuária a ser criada sob o framework PRODAGRO. O método de *inicializar()* de uma aplicação agropecuária é um procedimento comum às aplicações do domínio, porém cada aplicação tem aspectos particulares. Isso caracteriza a possibilidade de uso do metapadrão unificação (PREE, 1995), em que um método template é usado para generalizar a estrutura do algoritmo de inicialização e as particularidades de cada caso ficam sob responsabilidade de métodos hook.

4.1.9 Controle do framework

Uma outra característica importante de frameworks é a inversão do controle entre a aplicação e o framework sobre o qual ela está baseada. O desenvolvedor usualmente escreve o código a ser invocado pelo framework (JOHNSON, 1988). Isso é uma forma totalmente diferente de pensar. Em sistemas convencionais, os desenvolvedores dos próprios programas proviam tudo, desde a estrutura até o fluxo de execução e, assim que necessário faziam chamadas para as bibliotecas de funções. Notem que isto é o principal fluxo de controle. Framework e bibliotecas podem ambas ter chamadas e chamado componentes iguais (DAMIAN, 1999).

É possível destacar que existem ferramentas que permitem o fluxo de controle de um framework orientado a objetos ser implementado em programação procedural, assim como que frameworks não necessariamente requerem uma linguagem de programação orientada a objetos (JOHNSON & FOOTE, 1991). Contudo, OO tem os atributos que são particularmente apropriados para a abordagem de framework.

4.1.10 Armazenamento de dados

A questão envolvendo armazenamento físico de informações, como por exemplo, em arquivos ou banco de dados, não foi contemplada na versão atual do framework em face da complexidade do domínio abordado. Além disso, o objetivo é a produção de uma arquitetura flexível e reusável dos conceitos envolvidos no domínio de gestão e

comercialização de produtos agropecuários. Com isso, vislumbra-se a possibilidade de explorar o mesmo numa versão posterior do PRODAGRO.

Na atual implementação do framework, todo o processo de armazenamento foi feito em memória, usando vetores unidimencionais. Para tratar disso, foi criado um conceito de lista, na qual estão agregados todos os objetos instanciados pertinentes a uma determinada abstração. Por exemplo, para tratar os objetos a serem instanciados pela abstração da classe *Cliente*, foi criado uma classe chamada *ListaDeCliente*. Cada uma dessas classes é responsável por criar um objeto, adicioná-lo ao vetor existente na mesma, bem como ter outras funcionalidades, tais como: retornar uma determinada instância do objeto, excluir uma instância e retornar um vetor de objetos. A Figura 4.1, ilustra claramente o que fora relatado.

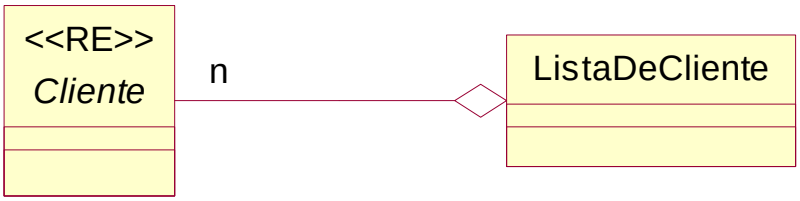

Figura 4.1: Controle dos objetos cliente via um objeto ListaDeCliente

Através do uso dessa estrutura, foi possível manter um baixo acoplamento entre a janela que prove a interface gráfica de interação do usuário com a aplicação e o modelo de domínio, visto que o conceito que estava envolvido é tratado na classe da lista. Ou seja, todas as funcionalidades são de responsabilidade da lista, tais como: inserir um cliente, excluir um cliente, retornar um cliente, retornar uma lista de clientes, retornar clientes de acordo com uma classificação, retornar cliente de acordo com um cep, etc.

4.2 Considerações finais

Além da redução do custo de desenvolvimento, um outro fator muito importante que decorre da reutilização de componentes pré-fabricados é a diminuição do tempo que o software leva para ser concluído. Como grande parte do software já está pronta para ser

usada, via classes concretas, é necessário implementar apenas as características adicionais da nova aplicação, através das classes abstratas.

Adicionalmente, a confiabilidade dos sistemas pode ser melhorada porque as classes concretas já foram submetidas a testes de unidade durante o seu desenvolvimento original e podem já ter comprovado a sua eficácia em outros sistemas em produção (JACOBSON & GRISS & JOHNSON, 1997).

Como resultado da fase de desenvolvimento do framework tem-se um entendimento da necessidade do domínio do problema de produtos agropecuários e como abordá-lo. Desse entendimento, uma proposta de como projetar o framework, seus pontos de flexibilização e suas funcionalidades principais. A divisão do framework em partes é uma proposta de flexibilização e de como num processo de instanciação, uma aplicação deve usar este framework.

Através do presente capítulo, foi possível ver o quanto é importante o domínio acerca das aplicações que estarão envolvidas na produção de um framework. Pois, com isso, será possível obter desde a análise até o projeto e implementação, resultados bem mais confiáveis e condizentes com aquilo que desejamos, que é a construção de uma arquitetura provida de características como reusabilidade, flexibilidade, generalidade, alterabilidade e manutebilidade.

O capítulo que segue apresenta duas experiências de uso sob o framework PRODAGRO, uma abordando o controle da comercialização de produtos agrícolas de grãos e a outra, comercialização de bovinos de corte.

5 EXPERIÊNCIA DE USO DO PRODAGRO

5.1 Visão geral

A finalidade básica de um framework é, ser reutilizado na produção de diferentes aplicações, minimizando tempo e esforço requeridos para isto. Desta forma, procura ser uma descrição aproximada do domínio, construída a partir das informações até então disponíveis (SILVA, 2000).

Uma das vantagens do desenvolvimento de aplicações a partir de frameworks é a possibilidade de reutilização desde a etapa de elaboração da arquitetura da aplicação (PASETTI, 1997), permitindo, ainda, ao usuário adaptar as características de sua aplicação.

O desenvolvimento de aplicações inicia com o entendimento do sistema contido no projeto do framework e segue, no detalhamento das particularidades da aplicação específica, o que é definido por seu usuário. Assim, a implementação de uma aplicação a partir do framework é feita pela adaptação de sua estrutura de classes, fazendo com que esta inclua as particularidades da aplicação (TALIGENT, 1997).

A utilização ideal de desenvolvimento de aplicações a partir de frameworks consiste em completar ou alterar procedimentos e estruturas de dados neles presentes. Sob esta ótica, uma aplicação gerada sob um framework não deveria incluir classes que não fossem subclasses de classes do framework. Todavia, como um framework nunca é uma descrição completa de um domínio, é possível que a construção de aplicações por um framework leve à obtenção de novos conhecimentos do domínio tratado, indisponíveis durante a sua construção (SILVA, 2000).

Cabe observar que, além do desenvolvimento de aplicações a partir de frameworks, estes também podem servir de exemplos para o desenvolvimento de outros frameworks ou fazer parte do desenvolvimento de novos (LARSSON, 1999).

Além das características dos frameworks, de flexibilidade e generalização, outro fator, que influencia na complexidade de desenvolvimento de novas aplicações a partir dos frameworks, é o fato de que eles podem agrupar diferentes quantidades de classes. Isto os torna mais ou menos complexos (SILVA, 2000). Frameworks típicos possuem centenas de classes que devem estar semanticamente relacionadas e interconectadas (PASETTI, 2000).

A experiência de uso da implementação das classes do PRODAGRO ocorreu através do desenvolvimento de duas aplicações pertinentes ao domínio de produtos agropecuários, descritas a seguir. Cabe ressaltar, que nem todas as classes contidas na arquitetura do PRODAGRO fazem parte das aplicações desenvolvidas, pois caracterizam outros artefatos[8] de software do domínio.

5.2 Desenvolvimento de aplicações usando o framework PRODAGRO

Conforme descrito desde o início do presente capítulo, foi disponibilizado um projeto arquitetônico provendo diferentes funcionalidades e, a partir das quais as aplicações do domínio de produtos agropecuários podem ser desenvolvidas sob o PRODAGRO.

Por ser o framework PRODAGRO caixa-cinza, é interessante saber quais classes devem ser geradas, quais classes podem ser geradas, e qual a diferença nos dois casos, bem como, conhecer quais as responsabilidades e as funcionalidades das classes a serem geradas segundo SILVA (2000), que também propõe três questões-chave que devem ser respondidas:

a) Quais as classes?: onde se define entre as classes concretas existentes, quais podem ser criadas ou reutilizadas no framework;

b) Quais os métodos?: ao gerar uma classe concreta, quais os métodos devem ser definidos e quais entre os herdados podem ser reutilizados?

[8] Artefato: Qualquer item criado como parte da definição, manutenção ou utilização de um processo de software. Inclui entre outros, descrições de processo, planos, procedimentos, especificações, projeto de arquitetura, projeto detalhado, código, componente, módulo, aplicação, documentação para o usuário. Artefatos podem ou não ser entregues a um cliente ou usuário final.

c) O que os métodos fazem?: qual o comportamento que os métodos devem assumir, quando eles o fazem e como devem ser implementados.

Assim quando for utilizar um framework tem que se conhecer bem mais do que simplesmente como construir uma aplicação. SILVA (2000) comenta que se deve conhecer os recursos disponibilizados para evitar esforços desnecessários de desenvolvimento.

Os usuários que quiserem produzir aplicações sob o framework PRODAGRO, devem seguir os seguintes passos:

- ✓ Especializar a classe *AplicaçãoAgropecuária*, redefinindo o método hook *criarInterface()*, no qual deve-se invocar a instanciação da interface definida para a interação do usuário com a aplicação. Esse método será invocado pelo método template *inicializar()*, também responsável por inicializar todas as classes de controle dos objetos criados pelo framework e pela aplicação. Esses controles devem ser definidos nos outros métodos hook da classe *AplicaçãoAgropecuária*, que tem por definição inicial "*ini_ListaDe....*".

- ✓ Especializar a classe *ProdAgro*, redefinindo o método hook *gerarAplicação()*, onde no seu corpo se deve instanciar a classe correspondente a especialização da classe *AplicaçãoAgropecuária*, seja pela invocação do seu método construtor. Depois de feita a instanciação, é disparada a invocação do método *inicializar()* da mesma. A Figura 4.1 ilustra a aplicação do padrão de projeto *AbstractFactory* no framework, produzindo apenas um acoplamento entre as subclasses de *ProdAgro* e *AplicaçãoAgropecuária*, e não o contrário. Através da ilustração apresentada nas Figura 4.1 e Figura 4.2, é possível vislumbrar esse processo.

- ✓ Especializar as classes *Operação* e *Documento*, caso as classes concretas das mesmas não contemplem o comportamento desejado. Na classe *Operação* deve-se redefinir o método *gerarDocto()*, responsável por instanciar o(s) documento(s) requerido(s) para a mesma. Com relação a classe *Documento*, tem-se duas divisões, sendo elas as subclasses *DocumentoFiscal* e *DocumentoNaoFiscal*. Isso

se deve pela relação existente entre ambas, onde a segunda tem características e comportamentos particulares, tendo em vista que normalmente está associada à algum *DocumentoFiscal*, já o contrário dificilmente acontece. Por isso, na estrutura de classes disposta na Figura 4.6 há uma associação da classe *Documento* com ela mesma, prevendo a possibilidade de haver relacionamento entre quaisquer tipos de documentos.

- ✓ Especializar a classe *Produto*, desde que a estrutura de classes definida para a mesma, não suporte alguma característica específica. O framework desenvolvido apresenta as classes concretas *ProdutoAgricola*, *ProdutoPecuário*, *ProdutoComposto* e *ProdutoDerivado*. A classe *ProdutoComposto* é responsável pela composição de objetos do tipo *Produto*. Normalmente, nas aplicações ela engloba o contexto de lote, mas no desenvolvimento do presente framework optou-se por *ProdutoComposto*. Na implementação da mesma, usou-se o padrão de projeto *Composite* (GAMMA, 2000), por esse se enquadrar no contexto envolvido na classe, ou seja, poder tratar outros objetos do tipo *Produto*.

- ✓ Especializar as classes *Espécie* e *Categoria*, definindo seus comportamentos. Essas classes são necessárias principalmente para o desenvolvimento de aplicações pertinentes à pecuária. A classe *Espécie* pode ser composta por um conjunto de objetos do tipo *Categoria*.

- ✓ Especializar a classe *FatorDeQualidade*, redefinindo seu comportamento através do método *calcular()*, responsável por executar uma operação de soma ou subtração num determinado valor. Além disso, também definir com quais objetos das outras classes que compõe o relacionamento, o mesmo deva interagir. A Figura 4.7 ilustra o processo de interação com as classes.

- ✓ Especializar a classe *Moeda* caso queira operar com uma moeda que não seja o real (R$). A estrutura dessa classe já contém uma classe concreta *Real*.

- ✓ Especializar a classe *TipoDeCálculo* se aplicação a ser gerada operará com tipos financeiros que não sejam em porcentagem ou valor absoluto e, redefinir o comportamento do método *calcular()*. No framework desenvolvido, a estrutura da classe *TipoDeCálculo* já apresenta duas classes concretas, representadas por *Porcentagem* e *Absoluto*.

A seguir, serão demonstradas duas aplicações geradas sob o framework PRODAGRO, ilustrando o que foi necessário para desenvolvê-las.

5.3 Aplicação para comercialização de produtos agrícolas (grãos)

Este tipo de aplicação envolve basicamente a comercialização de produtos agrícolas de grãos, das espécies oleaginosa e cereal, como por exemplo: soja, milho, trigo, canola e girassol. O autor optou por desenvolver essa aplicação em virtude de sua experiência com aplicações similares. Esse tipo de aplicação adapta-se a necessidade de empresas do ramo agrícola que comercializam grãos.

Para desenvolver essa aplicação, foi possível através da arquitetura do framework PRODAGRO, obter reuso das seguintes estruturas de classe.

- ✓ Foram reusadas 31 classes concretas por composição, sendo elas:
  - Na classe abstrata *Cliente*, usou-se as que envolviam *PessoaJurídica* e *Produtor*.
  - Na classe abstrata *Produto*, usou-se a classe *ProdutoAgricola*.
  - Na classe abstrata *Empresa*, usou-se a classe *UnidadeEmpresarial*.
  - Na classe abstrata *Operação*, usou-se as classes *Fatura, TransferênciaPessoaFísica, Depósito, Transferência, Compra e Venda*.
  - Na classe abstrata *Documento*, usou-se a classe *NotaFiscal* para o tipo *DocumentoFiscal* e a classe *Romaneio* para *DocumentoNaoFiscal*.

- Na classe abstrata *Taxa* do tipo *FatorDeCálculo*, usou-se as duas existentes ou seja, a de *Débito* e *Crédito*.

- Na classe abstrata *TipoDeCálculo*, igualmente usou-se as duas existentes ou seja, a de *Porcentagem* e *Absoluto*.

- Na classe abstrata *UnidadeDeMedida*, usou-se as classes *Quilo* e *Saca*.

- Na classe abstrata *Moeda*, usou-se a classe *Real*.

- *InscricaoEstadual, Cep, Saldo, Preço, Item, TabelaFiscal e TabelaDeUmidade*.

- Na classe abstrata *FatorDeQualidade* do tipo *FatorDeCálculo*, basicamente usada na classe concreta *Depósito*, foram utilizadas as classes *Umidade*, *Impureza*, *GraosVerdes*, *GraosAvariados* e *GraosQuebrados*. Para a classe *Umidade* fez-se necessário usar a classe concreta *TabelaDeQualidade*. O comportamento definido para todas elas é de subtrair do peso bruto da carga a porcentagem informada no momento da operação, com exceção da classe *Umidade*, onde é descontado o valor que consta no vínculo com a classe *TabelaDeQualidade*.

✓ Foram especializadas 2 classes a partir de classes abstratas, sendo elas:

- A classe *ProdAgro* foi especializada em *AplicaçãoProdAgriGraos*.

- A classe *AplicaçãoAgropecuária* foi especializada em *ProdAgriGraos*.

A seguir, serão descritas as principais funcionalidades dessa aplicação.

Figura 5.1: Menu de cadastros

Na Figura 5.1 serão executadas as funcionalidades do sistema de grãos pertinentes ao menu Cadastro, sendo possível efetuar a criação de objetos tipo cliente, unidade empresarial, produto agrícola, preço e criação de taxa. Nos três primeiros itens (cliente, unidade empresarial e produto agrícola) é feita a instanciação de objetos desse tipo. Já as funcionalidades envolvidas nas outras duas opções (taxa e preço), estão ilustradas na seqüência.

A Figura 5.2 possibilita a criação de taxas, sejam elas de débito ou crédito. As taxas de débito executam uma operação de subtração sobre um valor ou quantidade, dependendo do comportamento que lhe fora atribuído. Por outro lado, as taxas de crédito, fazem a operação inversa, adicionando um valor ou quantidade. De acordo com a figura abaixo, a descrição financeiro significa que a mesma atuará sobre o valor total e físico, significa que ela será executada sobre a quantidade. Essas taxas também podem ter diferentes comportamentos quanto ao tipo de cálculo, podendo ser de porcentagem ou absoluto. Além disso, é necessário definir a operação na qual a mesma será atribuída, bem como um nome e valor. Outra flexibilidade importante que a mesma apresenta, é a possibilidade de atuar sobre produtos (nenhum, lista ou todos) e clientes (nenhum, lista ou todos).

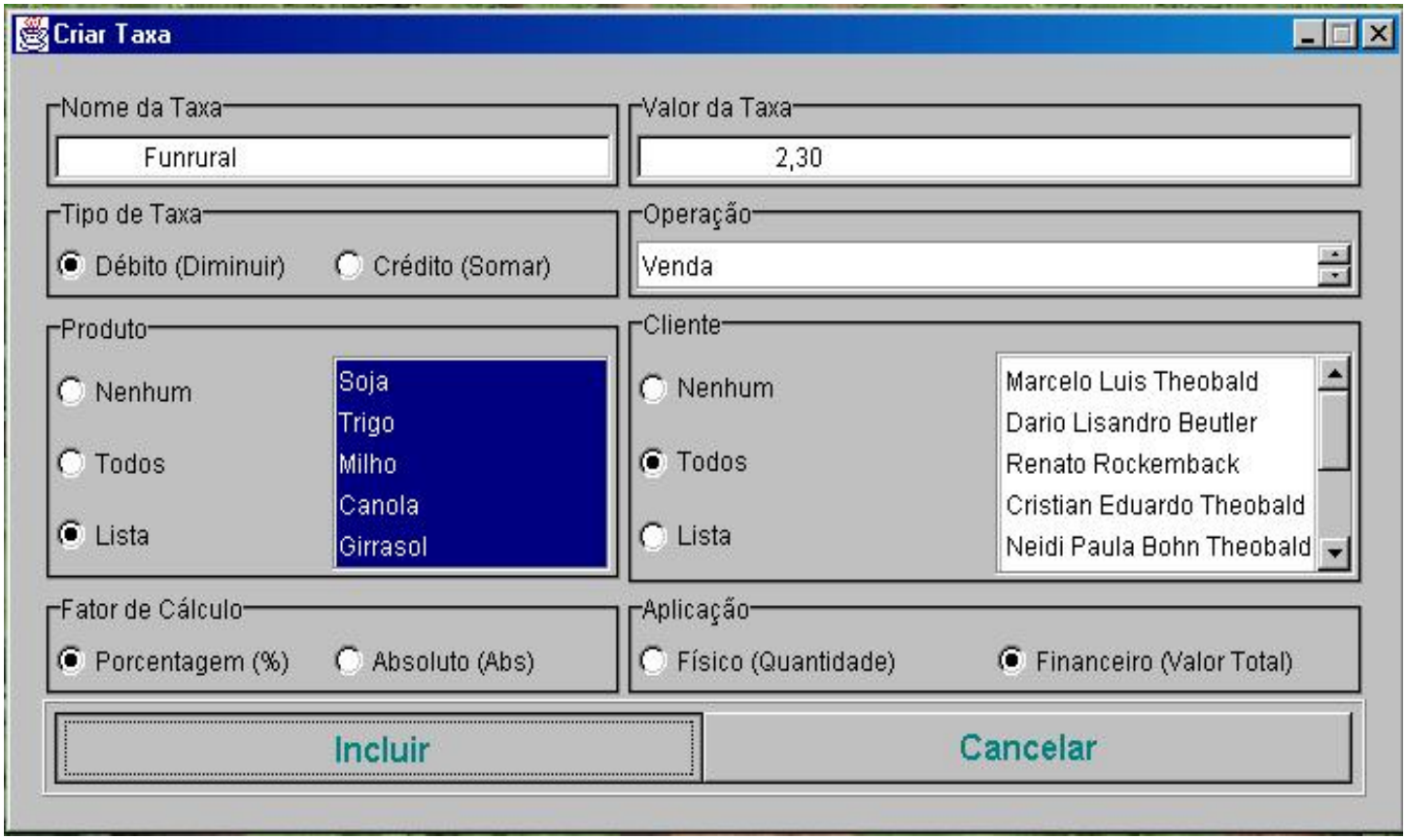

Figura 5.2: Criando taxas de débito e crédito

Para poder criar instâncias do objeto preço, usa-se a Figura 5.3, a qual permite definir tal processo. Para tanto, faz-se necessário determinar os seguintes atributos: o produto, a operação, a unidade de medida (quilo, saca, tonelada, etc.), assim como a moeda em que o mesmo vai vigorar, seu valor e a data. Para que, com isso, o produto possa ser comercializado pelo usuário da aplicação. Na Figura 5.3 não foi abordada a questão de indexadores de preços, mas é possível usufruir dessa característica para outras aplicações ou até mesmo para a atual, caso julgar necessário. A classe concreta *Indexador* possibilita definir um indexador diário para o preço, podendo o mesmo ser em porcentagem ou valor absoluto, definido através dos objetos *TipoDeCálculo*. Com isso, o preço a ser definido na Figura 5.3 serviria como base e o incremento diário seria feito através de instâncias tipo *Indexador*.

Figura 5.3: Criando objetos do tipo Preço

A Figura 5.4 apresenta as principais operações abrangidas pelo sistema de comercialização agrícola de grãos, sendo depósito, fatura, venda, compra, retirada, transferência de cliente, transferência de unidade empresarial, etc. As mesmas serão descritas no decorrer dessa seção.

Figura 5.4: Operações pertinentes a aplicação de grãos

O processo de depósito de produto agrícola, acontece em duas etapas, demonstradas na Figura 5.5 e Figura 5.6. A primeira, inicia-se quando o cliente chega com o produto (por exemplo, soja) na empresa e quer depositar o mesmo. Então o produto é pesado na balança, juntamente com o veículo de transporte. Seguindo, o usuário da aplicação interage com o cliente no sentido de informar seus dados (nome e nota fiscal de produtor), bem como informa ao sistema qual a unidade de medida e o peso bruto da carga. Finalizando essa

etapa, é gerado um documento, mais precisamente um romaneio de pesagem do produto. Uma ilustração disso ocorre na Figura 5.5.

Figura 5.5: Romaneio de pesagem de um produto agrícola

A segunda etapa do processo de depósito, é demonstrada na Figura 5.6, onde o cliente já deixou o produto na empresa e vai finalizar o processo. Para tanto, o cliente deve deixar o seu veículo de transporte na balança, a fim de obter o peso (tara) do mesmo. Nesse momento o usuário da aplicação informa ao sistema o número do romaneio correspondente ao depósito em questão. O sistema então, busca os dados pertinentes a esse documento e os mostra na figura para confirmação do usuário. Seguindo, o usuário informa os demais dados para finalizar o romaneio, que são: tara do veículo (peso) e os valores dos fatores de qualidade (impureza, umidade, grãos verdes, etc.). Após isso, o usuário confirma o processo de depósito e o sistema atualizará os dados do romaneio, bem como vai gerar um outro documento, que é a nota fiscal de depósito. Dessa forma conclui-se a operação de depósito de um produto agrícola. Existem empresas em que a nota fiscal é gerada somente no final do dia, pois o cliente traz diversas cargas para depósito e aí, faz-se necessário gerar somente uma nota fiscal correspondente a todos os romaneios de depósito daquele dia. Isso também foi previsto no framework PRODAGRO.

Figura 5.6: Retorno da pesagem - tara do veículo

O processo de fatura é apresentado na Figura 5.7 e, ocorre quando o cliente quer vender o produto depositado na empresa ou adquirido pelo processo de compra. Para tanto, o usuário interage com o cliente para solicitar ao mesmo qual o produto e a quantidade que ele deseja faturar/vender. Depois o usuário confirma o processo e o sistema, baseado no preço cadastrado para a operação de fatura/venda, faz o cálculo do valor bruto da transação, bem como o cálculo de eventuais taxas (crédito e débito) que estejam vinculadas à tríade operação/produto/cliente. Uma taxa de débito padrão para pessoa física é o funrural, no valor de 2,3 % e, que deve ser aplicada sob o valor bruto. Para finalizar a operação, o sistema vai gerar o documento correspondente a mesma.

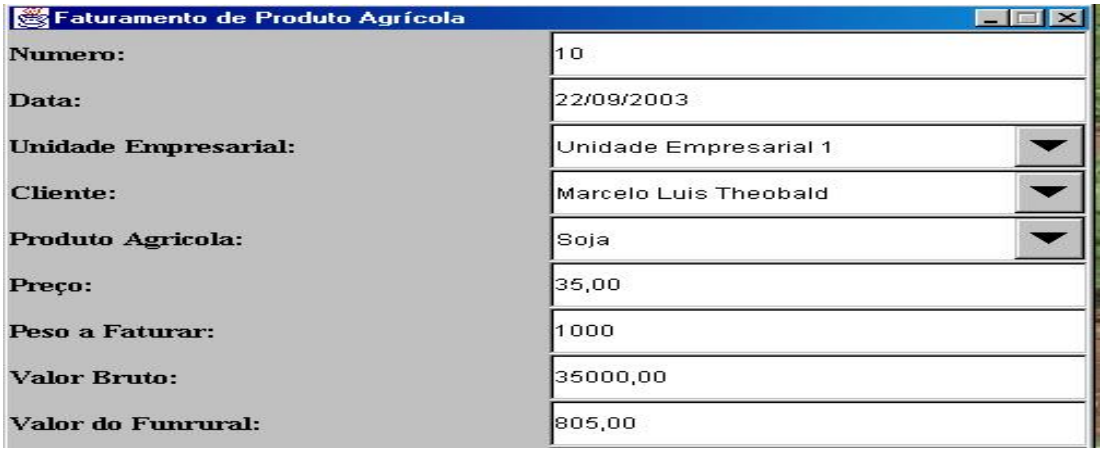

Figura 5.7: Faturamento de produto agrícola - cliente vende o produto em depósito na empresa

Na compra de produtos agrícolas de grãos, o processo ocorre da seguinte forma. O cliente chega até a empresa e conversa com o usuário. Ele repassa seus dados ao usuário e a quantidade de produto requerida. Por conseguinte, o usuário informa os mesmos ao sistema, que por sua vez calcula o valor total, com base no preço de compra cadastrado. Caso houver alguma taxa a ser descontada ou acrescida do valor, o sistema também fará. Quando finalizar a operação, o sistema vai gerar o documento correspondente a mesma, visto que é flexível ao usuário, cabendo a ele decidir sobre isso, através da especialização da superclasse "*operação*". A apresentação disso está na Figura 5.8.

Figura 5.8: Operação de compra de produto agrícola

O processo de transferência de produto de uma unidade empresarial para outra, também ocorre em duas etapas, sendo que a primeira resume-se a pesagem do veículo (tara) e a geração do romaneio de pesagem. Isso é ilustrado na Figura 5.9.

Figura 5.9: Operação de transferência de produto entre unidades (1ª etapa)

Depois, já com o produto carregado no veículo, acontece a segunda pesagem, ilustrado a partir da Figura 5.10. Com isso, o usuário informa ao sistema o romaneio de pesagem do veículo, gerado a partir da interação ocorrida conforme ilustrado na Figura 5.9 e, o sistema, busca os dados do mesmo. O usuário informa os demais campos pertinentes à operação em questão, tais como: produto, peso bruto, fatores de qualidade, unidade empresarial de destino e unidade de medida da pesagem. Por fim, o usuário confirma o processo e, o sistema atualiza o romaneio, bem como gera o documento de transferência, que é a nota fiscal.

Figura 5.10: Operação de transferência de produto entre unidades (2ª etapa)

A Figura 5.11, apresenta o processo de transferência de produto entre clientes tipo pessoas físicas. Para tanto, faz-se necessário que o cliente informe seus dados ao usuário e, este por sua vez informe-os ao sistema. Seguindo, o usuário solicita ao cliente qual é o produto que ele deseja transferir, assim como o cliente recebedor do produto e a respectiva quantidade, passando também a informar ao sistema. Para encerrar, ele confirma a operação e o sistema gera o documento hábil de transferência, qual seja a nota fiscal.

Figura 5.11: Transferência de produto entre pessoas físicas

A Figura 5.12 apresenta uma consulta de saldo à informações pertinentes a um determinado produto de uma unidade empresarial e de um cliente. É possível obter o saldo físico do produto, seja ele em quilos ou sacas, bem como o valor financeiro atualizado. Para tanto, o usuário deve informar à unidade empresarial, o produto, o cliente, o documento (caso quiser saber por documento) e o tipo de moeda, para o sistema buscar as informações requeridas e visualizá-las ao usuário da aplicação.

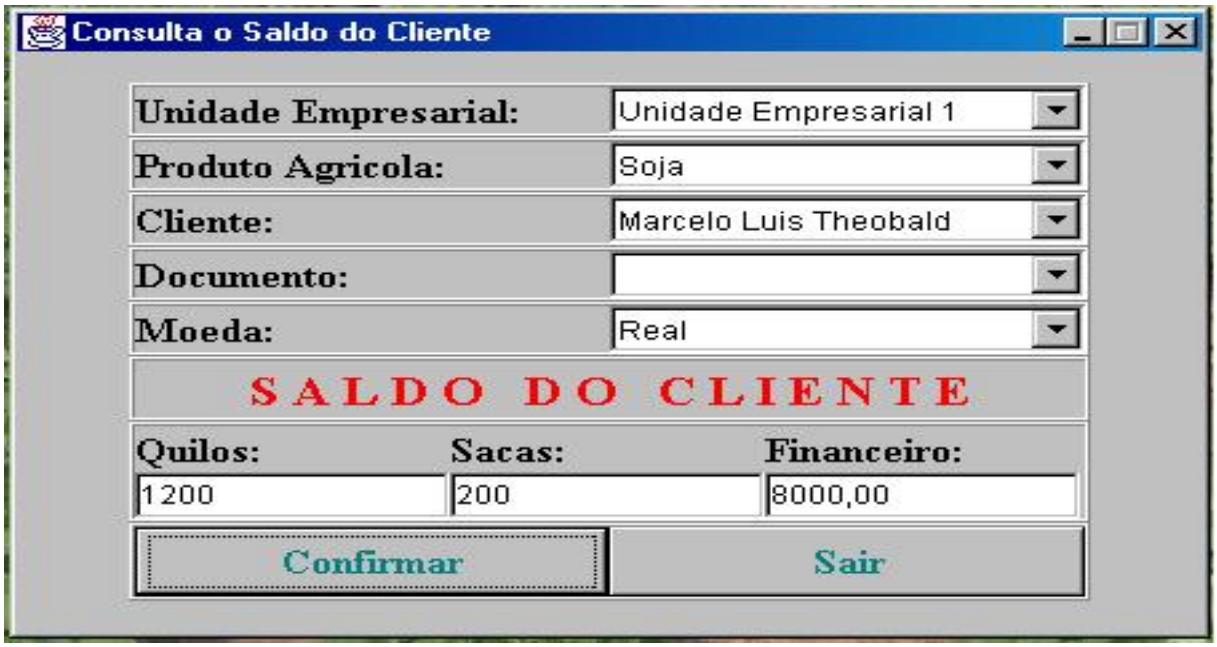

Figura 5.12: Consulta saldo do cliente

5.3.1 Considerações sobre a aplicação de grãos

Cabe ressaltar, que para criar a aplicação descrita na seção anterior, faz-se necessário a especialização de apenas duas(2) classes abstratas do framework PRODAGRO e a instanciação de 31 classes concretas, possibilitando assim um substancial reuso de 33 classes do framework. Com isso, conclui-se que para essa aplicação, relativamente complexa, foi possível atingir o objetivo do presente trabalho, que é dispor de um framework que prove reuso de projeto e código em grande escala.

5.4 Aplicação para comercialização de bovinos de corte

Esta aplicação é destinada principalmente à clientes como por exemplo, fazendeiros e produtores rurais, envolvidos com a comercialização de bovinos de corte. Por exemplo, se considerarmos um fazendeiro que possui diferentes fazendas, sendo que as mesmas podem ser compostas de diversos locais com menores dimensões (exemplo: piquetes). O usuário da aplicação pode executar operações como compra animais de outros fazendeiros ou proprietários rurais, venda de animais da fazenda, transferência de animais entre fazendas, transferência de animais de um local da fazenda para outro. Além disso, tem-se as operações que podem ser executadas sob um lote[9], representando objetos de tipo *ProdutoComposto*.

De acordo com a arquitetura do framework PRODAGRO, foi possível obter reuso das seguintes estruturas de classe para gerar a aplicação.

- ✓ Foram reusadas 26 classes concretas por composição, sendo elas:
  - Na classe abstrata *Cliente*, usou-se as que envolviam *PessoaJurídica* e *Produtor*.
  - Na classe abstrata *Produto*, usou-se a classe *ProdutoPecuário e ProdutoComposto*.

- Na classe abstrata *Empresa*, usou-se a classe *Propriedade*.
- Na classe abstrata *Operação*, usou-se as classes *TransferênciaDeLocal, Transferência, Compra e Venda*.
- Na classe abstrata *Documento*, usou-se a classe *NotaFiscal* para o tipo *DocumentoFiscal* e a classe *Romaneio* para *DocumentoNaoFiscal*.
- Na classe abstrata *Taxa* do tipo *FatorDeCálculo*, usou-se as duas existentes ou seja, a de *Débito* e *Crédito*.
- Na classe abstrata *TipoDeCálculo*, usou-se a de *Porcentagem*.
- Na classe abstrata *UnidadeDeMedida*, usou-se a classe *Quilo*.
- Na classe abstrata *Moeda*, usou-se a classe *Real*.
- Na classe abstrata *Espécie*, usou-se a classe *Bovino*.
- Na classe abstrata *Finalidade*, usou-se a classe *Corte*.
- *InscricaoEstadual, Cep, Local, Raça, Saldo, Preço, Item, e TabelaFiscal*.

✓ Foram especializadas 3 (três) classes a partir de classes abstratas do framework, sendo elas:

- A classe *ProdAgro* foi especializada em *AplicaçãoBovinosCorte*;
- A classe *AplicaçãoAgropecuária* foi especializada em *BovinosCorte;*
- A classe *Categoria* foi especializada pela classe *Boi*;

A seguir, serão descritas as principais funcionalidades dessa aplicação.

---

[9] Corresponde a um conjunto/composição de animais. No framework, é representado pela classe concreta ProdutoComposto.

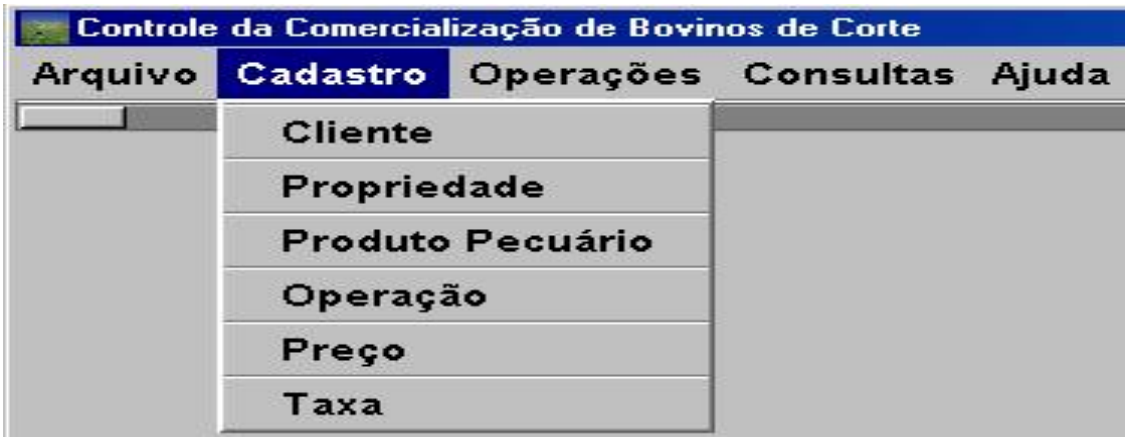

Figura 5.1: Menu de cadastro da aplicação bovinos de corte

Na Figura 5.1 estão ilustradas as opções disponíveis no item de menu Cadastro, das quais diferem, em relação a aplicação descrita na seção 5.3, apenas o item que instancia objetos do tipo propriedade e o item produto pecuário. Portanto, as funcionalidades para criar objetos tipo preço e taxa, apresentadas na aplicação da seção anterior, dispõem nessa aplicação da mesma flexibilidade e generalidade. Com isso, esses itens dispensam uma atenção especial na presente seção. Para instanciar objetos do tipo propriedade, o processo é similar ao executado para instanciar uma unidade empresarial, com exceção dos atributos particulares de cada objeto. O mesmo não acontece para instanciar objetos do tipo produto pecuário (animais) ou produto composto (lotes). Isso é ilustrado na Figura 5.2.

Figura 5.2: Criando instâncias da classe ProdutoPecuário

Através da Figura 5.2 é possível instanciar objetos tipo produto pecuário, ou seja, animais (bois, vacas, garotes, bezerros, etc...). Para tanto, deve-se preencher os campos disponíveis na mesma, onde a propriedade representa a fazenda do proprietário (usuário da aplicação). Já o local, significa as diferentes divisões existentes na propriedade (por exemplo, os piquetes). Seguindo, basta informar os campos que correspondem ao número, brinco (identificação existente na orelha), nome, categoria (vaca, boi, bezerro, garote, etc.), raça, pelagem, sexo, origem/procedência do animal (compra, nascimento ou doação), data de nascimento ou compra, data de desmame, grau de sangue, apelido, ascendente pai, ascendente mãe, RGN (Registro de Nascimento) e RGD (Registro Definitivo). Com isso, é possível registrar o animal (instanciar o objeto) na aplicação. O campo categoria merece um destaque, haja visto que ele é atualizado de acordo com a idade do animal, através de atributos pertencentes a esse objeto, pode-se identificar em qual categoria o mesmo se encontra.

A Figura 5.3 apresenta as principais operações envolvidas no sistema de comercialização de bovinos de corte. Algumas serão descritas no decorrer dessa seção.

Figura 5.3: Menu das operações disponíveis na aplicação

Para efetuar a transferência de um animal de local, basta informar o animal e selecionar um outro local para o mesmo, visto que para essa operação não é necessário gerar um documento.

A pesagem do lote ou individual, é feita com o intuito de acompanhar o desenvolvimento do mesmo possibilitando efetuar uma avaliação mais precisa e consistente, para depois tomar alguma decisão quanto a um possível rearanjo de local e até se preciso, de propriedade.

A seguir, são apresentadas a Figura 5.4 e Figura 5.5, através das quais é realizada a aquisição (compra) de animais ou lotes, sendo necessário informar a data de aquisição e o valor, além dos dados contidos na ficha cadastral do animal ou lote.

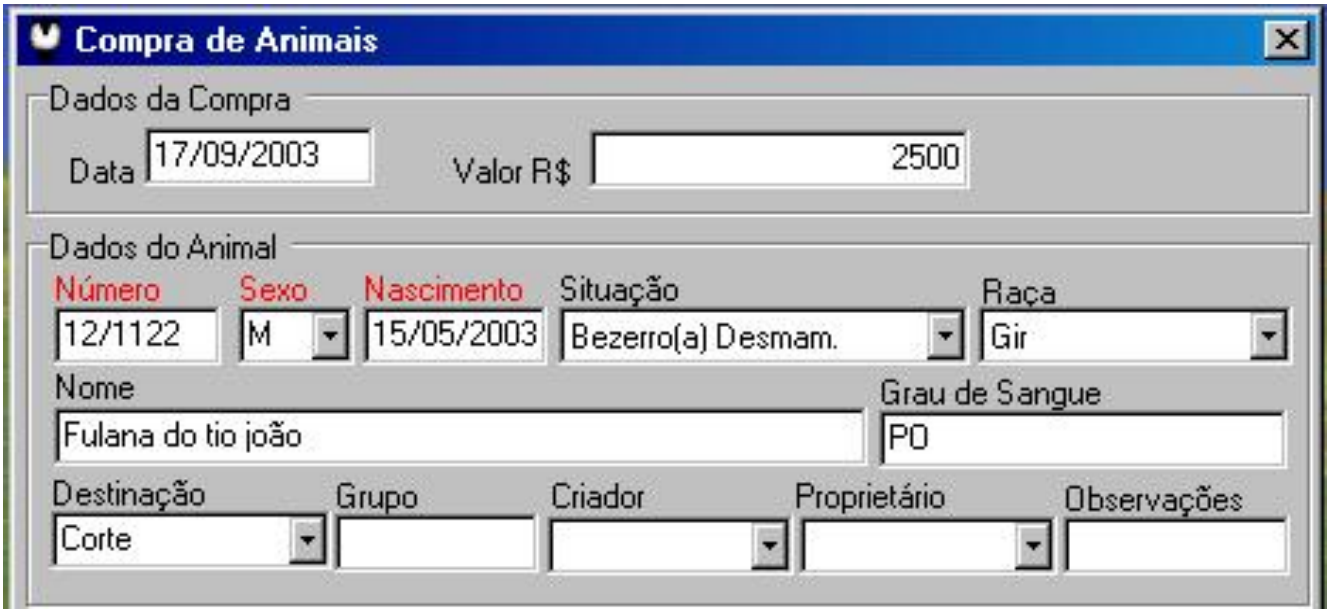

Figura 5.4: Operação de compra de animais (individual)

Na compra de um lote de animais, faz-se necessário informar a quantidade de animais que compõem o mesmo.

Figura 5.5: Operação de compra de animais (lote)

A Figura 5.6 apresenta os dados necessários para criar uma instância da classe *ProdutoComposto* (lote).

Figura 5.6: Criando uma instância de ProdutoComposto (lote)

A Figura 5.7 apresenta uma consulta a informações pertinentes a um determinado lote de animais, bem como o valor financeiro atualizado do mesmo. Para obter as informações do lote, o usuário da aplicação deve informar a propriedade (fazenda, propriedade rural, etc...), o lote de animais e o tipo de moeda para o sistema poder calcular o valor total do lote e o preço médio por animal pertencente ao lote. Além disso, pode-se apresentar outras informações como a quantidade de animais que o lote comporta, a espécie e a categoria.

Figura 5.7: Consulta dados de um lote

5.4.1 Considerações sobre a aplicação de bovinos de corte

Para criar a aplicação descrita na seção 5.4, fez-se necessário a especialização de apenas três(3) classes abstratas do framework PRODAGRO e foi possível o reuso por composição de 26 classes concretas, perfazendo um total de 29 classes reusadas sob o framework. Diante disso, pode-se dizer que para uma aplicação um pouco mais simples do que a descrita na seção 5.3, igualmente foi possível atingir o objetivo do presente trabalho, de prover reuso de projeto e código em grande escala através de um framework.

5.5 Considerações finais

Com o desenvolvimento das aplicações sob o framework PRODAGRO, ficou claro de que o desenvolvedor de um framework necessita primeiramente de um conhecimento acerca do domínio de aplicações que ele deseja envolver no mesmo. Segundo, de que o processo de desenvolvimento do framework é complexo, demorado e requer muita pesquisa, a fim de ter bem claro o que representa e o que é uma arquitetura de um framework.

A partir dessa experiência de desenvolvimento é possível concluir que desenvolver aplicações com base numa arquitetura que prove reuso de projeto e de código, pode vir a ser benéfica, dada a facilidade e rapidez de desenvolvimento que ela proporciona.

O desenvolvimento de sistemas baseado em objetos pode ser empregado com sucesso na grande maioria das aplicações. Pode-se esperar benefícios com um melhor gerenciamento da complexidade dos sistemas, e no aumento da produtividade dos programadores, como conseqüência do encapsulamento e do reuso. A manutenção é facilitada, assim como a capacidade de expansão dos sistemas, reduzindo também os custos a longo prazo. A herança permite que se adicione, ou elimine objetos sem comprometer a aplicação como um todo.

O desenvolvimento do framework PRODAGRO foi bem sucedido dado que:

1) é possível desenvolver aplicações sob o framewrok, como já foi feito;

2) nos dois casos ocorreu um significativo reuso, como pode ser constatado a partir dos números das duas aplicações descritas nas seções 5.3 e 5.4.

6 CONCLUSÕES

6.1 Considerações iniciais

Os objetivos desta dissertação foram alcançados, pois uma arquitetura foi definida e materializada através do desenvolvimento de um protótipo de framework para gestão e comercialização de produtos agropecuários. A experiência de uso ocorreu através do desenvolvimento das duas aplicações, descritas na seção 5.2. Além disso, o autor da dissertação igualmente sente-se capaz de disseminar os conceitos envolvidos neste trabalho no meio que o cerca, a saber, acadêmico e comercial.

A proposta inicial foi atingida e, com isso, espera-se que os desenvolvedores de software para gestão e comercialização de produtos agropecuários, possam desenvolver suas aplicações com menos custo, reduzindo a complexidade de desenvolvimento das partes específicas de cada software, uniformizando a estrutura do software e, além disso, produzir com maior produtividade, visto que, poderão reutilizar projeto e código. Por conseguinte, possivelmente os usuários das aplicações também podem ser atendidos com maior rapidez e agilidade, satisfazendo suas necessidades, dado que provavelmente os desenvolvedores das aplicações agropecuárias, desenvolvem as aplicações com maior rapidez.

Através da presente dissertação pode se ter perspectivas quanto a possibilidade de um aumento de produtividade, de qualidade e de diminuição de custo na produção de software agropecuário. Pois, foi elaborada (produzida) uma arquitetura reusável e, ela se mostrou adequada sob os dois critérios de avaliação de um framework: primeiramente através das duas aplicações produzidas e segundo, ocorreu um reuso expressivo em termos de quantidade de classes reusadas versus quantidade de classes desenvolvidas especificamente para cada aplicação.

O autor da dissertação percebeu que o processo de desenvolvimento de um framework deve ser num contínuo aprendizado, buscando o crescimento na medida em que

as dificuldades se apresentam. É notório e imprescindível passar por todas as etapas para conseguir chegar ao resultado almejado. Como no caso de um framework, não adianta desenvolver o mesmo sem ter conhecimento do domínio do problema, bem como, é vital desenvolver as aplicações para poder concretizar o projeto. Portanto, não podemos "queimar" etapas do processo de desenvolvimento de um framework, ou seja, faz-se necessário passar pela análise, projeto, implementação e testes, para que o produto final desejado seja realmente aquilo que fora inicialmente planejado. Lembrando que isso pode ocorrer paralelamente, ou seja, não necessitando terminar uma etapa para iniciar a posterior.

Dentre os resultados do esforço de pesquisa pode-se destacar como principais:

- ✓ O framework PRODAGRO, capaz de disponibilizar aos desenvolvedores de aplicações agropecuárias suporte, para produzir aplicações de gestão e comercialização de produtos agropecuários, visto ser possível produzi-las através do reuso de código e projeto, aproveitando o que já fora previamente codificado.

- ✓ A concepção arquitetônica do framework, que pode ser reutilizada independentemente da linguagem de programação. Os principais pontos adaptáveis do framework são as diferentes operações que podem ser criadas pelo usuário do mesmo, permitindo definir quais os documentos a serem gerados na mesma. Além disso, o framework provê o controle do fluxo de execução das colaborações existentes entre suas classes, sendo que para a construção de aplicações, faz-se necessário a especialização e ou composição de classes do framework, mas que são chamadas pelas classes do framework (Princípio de Hollywood).

- ✓ A disponibilização de um protótipo de implementação do framework PRODAGRO em linguagem de programação JAVA. Programadores com conhecimento nessa linguagem podem reutilizar não apenas o projeto, mas também o código. Na implementação foram utilizados projetos de classes da

própria linguagem, para com isso facilitar o entendimento dos conceitos envolvidos no projeto.

- ✓ A disponibilização de duas aplicações agropecuárias – ProdAgri e BovinosCorte – desenvolvidas sob o framework PRODAGRO. Desenvolvimento este que comprovou a viabilidade da arquitetura proposta, uma vez que suportou o desenvolvimento das duas aplicações, como também comprovou a ocorrência de reuso significativo.

- ✓ Tornar-se um disseminador dos conceitos e idéias apresentadas na dissertação.

- ✓ O aprendizado do autor desta dissertação referente a temas, como:

  - Desenvolvimento de frameworks.

  - Orientação a Objetos, englobando desde análise até o desenvolvimento de aplicações.

  - Padrões de Projeto ou Design Patterns, desde identificar o uso de algum padrão, como saber se posso criar algum padrão.

  - Conhecimento de uma ferramenta de modelagem orientada a objetos, denominada UML.

  - Linguagem de programação orientada a objetos, JAVA.

  - Conhecimento de diferentes softwares agropecuários, bem como de assuntos relacionados ao domínio de produtos agropecuários.

Para a implementação das aplicações, foi escolhida a linguagem Java, por ser uma linguagem de programação orientada a objetos, capaz de suportar os conceitos do paradigma de orientação a objeto, proporcionando o desenvolvimento de aplicações voltadas para o reuso, como é o caso de frameworks.

6.2 Vantagens de produzir um framework para aplicações agropecuárias

- Espera-se o aumento de produtividade, através do reuso de projeto e código, e aumento da qualidade do software a ser desenvolvido, considerando que o framework esteja testado e depurado.

- Dispor de uma arquitetura contendo uma grande gama de conhecimento sobre o domínio, possibilidade através da flexibilidade e generalidade inserida na mesma, vislumbrar a possibilidade de novas aplicações ou alterar processos que em aplicações anteriores estavam confusas e com a manutenção comprometida.

- Disponibilizar aos desenvolvedores de aplicações agropecuárias uma ferramenta, que pode acelerar o processo de desenvolvimento de software agropecuário.

- Estabelecer uma certa padronização no desenvolvimento de aplicações agropecuárias, através do uso de uma arquitetura pré-definida.

- Possibilidade de diminuir o custo de produção de uma aplicação, visto que grande parte da mesma já foi desenvolvida, evitando certos desgastes desde a análise, projeto, implementação e testes.

6.3 Limitações

O framework PRODAGRO, apesar de prover reuso de projeto e código em grande escala, apresenta algumas limitações, basicamente no que se refere ao desenvolvimento de outras aplicações do domínio envolvido no mesmo. Pois, com isso, poderia ter sido produzida uma arquitetura mais amadurecida e completa. A seguir, estão descritas algumas limitações:

- Sob o framework PRODAGRO, desenvolver aplicações para outras culturas, tais como frutas, hortaliças, café, algodão, bem como, outros ramos de atividade da

pecuária, como leite e reprodução; Com isso será possível obter uma avaliação mais precisa e consistente da arquitetura projetada.

- Indisponibilidade de classes concretas que promovam um reuso mais significativo, no sentido de diminuir o esforço para produzir aplicações que possibilitem controlar o processo alimentar de um animal.
- Desenvolver aplicações que envolvam produtos derivados de produtos agropecuários, abrangendo assim, atividades envolvidas na comercialização destes.
- O domínio de aplicações que envolvem a gestão de insumos agropecuários, tais como, herbicidas, inseticidas, produtos de lojas tipo PetShop, etc., não foram pesquisados e abordados na presente dissertação. Mas, podem ser abrangidas a partir da especialização da superclasse *Produto* e, eventualmente, haver a necessidade de definir algum comportamento específico, sendo que isso pode ser feito via subclasses de *Operação* e *Documento.*
- A integração com balanças eletrônicas não foi projetada. Caso esse framework seja usado para produzir aplicações comerciais, as informações resultantes da pesagem do produto (peso do produto e tara do veículo) deverão ser informadas manualmente.
- A persistência de dados não foi implementada. O framework PRODAGRO não trata da questão envolvida em persistência de objetos, tanto em arquivos quanto em banco de dados, somente em memória. Diante disso, não seria possível usar o mesmo momentaneamente para desenvolver aplicações comerciais.

6.4 Trabalhos futuros

O framework PRODAGRO apresenta-se como um suporte para poder construir aplicações agropecuárias e que na medida do possível, deve ser estendido para futuros esforços de pesquisa.

A seguir são descritas extensões que podem ser realizadas ao trabalho que fora desenvolvido:

- Construir uma ferramenta gráfica (ambiente) para o desenvolvimento de aplicações agropecuárias a partir do framewok PRODAGRO, possibilitando assim, o reuso de projeto e código da interface.
- Em vista de que na versão atual do framework PRODAGRO não fora abordada a persistência de dados, vislumbra-se implementar o armazenamento dos objetos (informações) em banco de dados ou arquivos.
- Adicionar no framework a funcionalidade que proporcione o controle alimentar da atividade pecuária, desde animais de corte, reprodução e leite, conforme limitação comentada na seção 6.3.
- Desenvolver componentes ou ferramentas que possibilitam a interligação com frameworks de controle financeiro (fiscal, contábil e financeiro), para torná-lo mais completo. Como por exemplo, Framework SanFrancisco da IBM (2000).
- Aprofundar a avaliação do framework, submetendo-o ao desenvolvimento de novas aplicações, proporcionando assim o seu amadurecimento.

6.5 Considerações finais

Aplicações modernas possuem natureza dinâmica e necessitam de suporte para se adaptarem a novas demandas funcionais e não-funcionais. Novas demandas funcionais são criadas por usuários das aplicações, que precisam funções não previstas originalmente ou necessitam alterar funções existentes. Durante o ciclo de vida das aplicações, podem também surgir requisitos não-funcionais que, em geral, não estão associados diretamente com a aplicação, como por exemplo, distribuição física de componentes, protocolos de comunicação, qualidade de serviço, disponibilidade ou tolerância a falhas. Modelos mais usuais para o desenvolvimento de software não oferecem as facilidades necessárias para a produção de aplicações que se adaptem dinamicamente a essas demandas. Portanto, uma abordagem que facilite a concepção dessas aplicações se faz necessária **(**SZTAJNBERG & LOQUES & LOBOSCO, 1999).

O autor da presente dissertação procurava sempre lembrar de uma frase dita pelo seu orientador Dr. Ricardo: “*Não adianta ter boas intenções, dizer que se esforçou, que*

tentou. É necessário produzir os resultados desejados". Diante disso, é possível concluir que resultados foram produzidos através da materialização de uma arquitetura que prove reuso de projeto e código e, o posterior desenvolvimento de duas aplicações sob o mesmo, desenvolvimento este que comprovou a viabilidade da arquitetura proposta., uma vez que suportou o desenvolvimento das duas aplicações, como também comprovou a ocorrência de reuso significativo, como discutido no capítulo 5.

O aprofundamento da pesquisa introduzida por essa dissertação, poderá auxiliar a tornar a abordagem voltada à promoção de reuso de software uma prática amplamente difundida, já que apresenta um exemplo prático da abordagem de frameworks orientado a objetos em um domínio específico. Além disso, o framework desenvolvido pode contribuir para o aumento da produtividade e da qualidade no desenvolvimento de aplicações agropecuárias, em função do reuso que é capaz de promover.

REFERÊNCIAS BIBLIOGRÁFICAS

AGRISOFT, Brasil. **Empresa especializada em softwares agropecuários.** Acessado em 11 de setembro de 2003. Disponível por http://www.agrisoft.com.br/.

AGROSOFT, Trabalhos de 1997. **Sistema informatizado de gestão da qualidade: uma aplicação para a suinocultura na Alemanha**. Acessado em 31 de maio de 2003. Disponível por http://agrosoft.softex.br/agrosobr/ver.php?page=59.

AGROSOFT, Guia de 2002. **Empresa de desenvolvimento de software agropecuário para o Rio Grande do Sul.** Disponível por http://www.agrosoft.com.br/guia/software/categorias.php3?desloc=0&cat=Culturas. Acessado em novembro de 2002.

AGROSOFT, Guia de 2003. **Empresa de desenvolvimento de software agropecuário para o Rio Grande do Sul.** Acessado em fevereiro de 2003. Disponível por http://www.agrosoftrs.com.br/agroleite.htm.

ALEXANDER, Cristopher. **A Pattern Language: Towns, Buildings, Construction**s. New York : Oxford University Press, 1977.

ALEXANDER, Christopher. **The Timeless Way of Building.** Oxford University Press, New York, 1979.

ALMA, Informática Ltda. **Empresa que tem por objetivo desenvolver sistemas informatizados de alta qualidade e prestar consultoria em processos de informatização de empresas. 2002.** Acessado em setembro de 2003. Disponível por http://www.almainformatica.com.br/PRODUTOS/CONGADO/congado.html.

APPLETON, Brad. **Patterns and Software: Essential Concepts and Terminology**. Disponível por http://www.enteract.com/~bradapp/. Acessado em junho de 1997.

ARAÚJO, João. **Modelação da Arquitectura**. 2001. Disponível por http://www.ctp.di.fct.unl.pt/~ja/as2/comppag.ppt. Acessado em janeiro de 2003.

BASS, L.; COUTAZ, J. **Developing software for the user interface.** Massachusets, Addison-Wesley, 1991.

BECK, Kent. **Using Pattern Languages for Object-Oriented Programs.** Technical Report nº CR-87-43, 1987. Disponível por http://c2.com/doc/oopsla87.html. Acessado em fevereiro de 2003.

BECK, Kent; JOHNSON, Ralph. **Patterns generate architectures.** In: EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, (ECOOP), 8., 1994. Proceeddings. Berlin: Springer Verlag, 1994.

BECK, Kent; GAMMA, Erich. ***Test Infected: Programmers Love Writing Testes*.** 1998. Disponível em: http://www.junit.org. Acessado em março de 2003.

BUSCHMANN, Frank. **Pattern-Oriented Software Architecture – A System of Pattern**s. 1996. Chichester : John Willey and Sons, 1996.

CNPGL. **A Embrapa Gado de Leite desenvolve e comercializa uma série de produtos em parceria com outras empresas.** 2002. Acessado em setembro de 2003. Disponível por http://www.cnpgl.embrapa.br/publicacoes/.

COAD, P.; YOURDON, E. **Análise baseada em objetos.** Rio de Janeiro: Campus, 1992.

CUMERLATO; SCHUSTER. **Consultoria e Desenvolvimento de Sistemas.** Acessado em julho de 2002. Disponível por http://www.cs.inf.br/compass%20frame%20work.htm.

DAMIAN, Adrian. **An Object Oriented Framework for the Simulation of Network Models. 1999.** University of Calgary. Acessado em julho de 2003. Disponível por http://sern.ucalgary.ca/students/theses/AdiDamian/Pre.htm .

DATACOPER. **Empresa especialista no desenvolvimento de software para comercialização de produtos agropecuários.** Acessado em fevereiro de 2003. Disponível por http://www.datacoper.com.br.

EMBRAPA, Cerrados. Campinas-SP. **Sistemas Produtivos.** 2001. Disponível por http://www.cpac.embrapa.br/sisprod.htm. Acessado em janeiro de 2003.

FAYAD, Mohamed E.; SCHMIDT, Douglas C. **Object-oriented application frameworks.** Communications of the ACM, 40(10):32-38, October 1997.

FOWLER, Martin. ***Refactoring: improving the design of existing code*.** ISBN: 0-201-48567-2, Addison-Wesley, 1999.

FRANCISCO, Vera Lucia F. dos S. **O acesso do setor rural à Internet no Estado de São Paulo**. Pesquisadora Científica do Instituto de Economia Agrícola. Disponível por

http://www.webrural.com.br/webrural/artigos/internet/acesso.pdf. Acessado em agosto de 2002.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns –Abstraction and Reuse of Object-Oriented Design**. LNCS, nº 707, p. 406-431, julho de 1993.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns: Elements of Reusable Object-Oriented Softwar**e. Reading, MA : Addison-Wesley, 1994.

GAMMA, Erich. **Design Patterns – Elements of Reusable Object-Oriented Software**. Reading: Addison Wesley Longman, 1995.

GAMMA, Erich. **Padrões de Projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2000.

GERBER, Luciano. **Uma Linguagem de Padrões para o Desenvolvimento de Sistemas de Apoio à Decisão Baseado em Frameworks**. Porto Alegre, 1999. Dissertação de Mestrado em Informática – Faculdade de Informática, PUCRS. Disponível por http://www.inf.pucrs.br/dissertacoes/arquivos/luciano.pdf. Acessado em janeiro de 2002.

HAUTZER, H. J.; HELBIG, R.; SCHIEFER, G. **Computer based information and report system for the support of agricultural extension: presentation of a prototype.** in Kure, H. et al. (ed) Proceedings of the first european conference for information technology in agriculture, Copenhagen.

JACOBSON, Ivar; GRISS, Martin; JOHNSON, Patrik. ***Software Reuse: Architecture, Process and Organization for Business Success.*** ISBN 0-201-92476-5, Addison-Wesley, 1997.

JOHNSON, Ralph. **Designing Reusable Classes. Journal of Object-Oriented Programming**, New York, v. 1, n. 2, p. 22-35, June/July 1988.

JOHNSON, Ralph E.; FOOTE, Brian. **Designing Reusable Classes.** Journal of Object-Oriented Programming, June/July 1991.

JOHNSON, R. E. **Documenting frameworks using patterns.** SIGPLAN Notices, New York, v.27, n.10, Oct 1992. Trabalho apresentado na OOPSLA, 1992.

JOHNSON, R. E. **How to design frameworks**. 1993. Acessado em dezembro de 1998. Disponível por FTP anônimo em st.cs.uiuc.edu.

JOHNSON, Ralph E**. Components, Frameworks, Patterns**. Department of Computer Sciece, University of Illinois. February, 1997.

JUNQUEIRA, Alvaro R. B.; COSTA, André Fernandes; LIRA, Édson Carlos, 1998. **Design Patterns: Conceitos e Aplicações.** Disponível por http://www.dcc.ufrj.br/~schneide/PSI_981/gp_6/design_patterns.html. Acessado em novembro de 2001.

LARSSON, Johan L. **Reuse, genericity and frameworks: a graduate thesis in computer science.** In: FAYAD, Mohamed E.; SCHMIDT, Douglas C.; JOHNSON, Ralph E. Implementing Applications Frameworks. New York: Wiley Computer Publishing, 1999. CD-ROM.

KRASNER, E. K.; POPE, S.T. **A Cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk-80**. Journal of Object Oriented Programming, p. 26-49, Agosto-Setembro 1988.

LARMAN, Craig. ***Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design*.** ISBN: 0-13-748880-7, Prentice-Hall, 1998.

LEWIS, Ted. **Object-Oriented Application Frameworks. Manning Publications**, 1995.

LUCENA, Carlos J. P. de; MILIDIU, Ruy. **Projetos do Laboratório de Engenharia de Software da PUC do Rio de Janeiro, 1998.** Disponível por WWW em http://www.les.inf.puc-rio.br/projetos.html.

MALDONADO, José C. **Padrões e Frameworks de Software**. Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação**.** Disponível por http://www.icmsc.sc.usp.br/~rtvb/apostila.pdf. Acessado em novembro 2001.

MARTIN , J. J. Odell. **Análise e projeto orientados a objetos**. São Paulo: Makron Books, 1995.

MASIERO, Paulo; BRAGA, Rosana, 2001a. **Frameworks de Software**. Disponível por http://www.icmc.sc.usp.br/~rtvb/Aula8Sce223.ppt. Acessado em abril de 2002.

MASIERO, Paulo; BRAGA, Rosana, 2001b. **GREN Framework**. Disponível por http://www.icmc.sc.usp.br/~rtvb/Aula9Sce223.ppt. Acessado em abril de 2002.

MATTSON, Michael. **Object-Oriented Frameworks: A Survey of Methodological Issue**s. Ronneby, Sweden, 1996. Tese (Doutorado) – Department of Computer Science and Business Administration, University College of Karlskrona/Ronneby.

MEYER, Bertand. **Object-Oriented Software Construction.** New Jersey : Prentice-Hall,1997.

MURTA, L. G. P.; BARROS, M. de O.; WERNER, C. M. L. **Token: Uma Ferramenta para o Controle de Alterações em Projetos de Software em Desenvolvimento, outubro 2000.** XIV Simpósio Brasileiro de Engenharia de Software. COPPE / UFRJ – Departamento de Sistemas e Computação. Disponível por http://www.cos.ufrj.br/~odyssey/publicacoes/SBES2000_Ferramentas_Apresentacao_Token.pdf.

PASETTI, Alessandro; PREE, Wolfgang, 2000. **Framework Methodology Project – Overview.** Disponível por http://www. Softwareresearch.net/FrameworkMethodologyProject/ProjectHomePage. html. Acessado em julho de 2002.

PAUSCH, Randy; CONWAY, Matthew; DELINE, Robert. **Lessons Learned from SUIT, the Simple User Interface Toolkit ACM Transactions on Information Systems**, October 1992, Vol. 10, No. 4, pp. 320-344. Disponível por http://www.ic.unicamp.br/proj-xchart/start/toolkits.html. Acessado em julho de 2002.

PREE, Wolfgang. **Design Patterns for Object-Oriented Software Development.** Reading,MA : Addison-Wesley, 1995.

PREE, Wolfgang. **Building Application Frameworks: Object-Oriented Foundations of Framework Design.** Hot-spot-driven development in M. Fayad, R. Johnson, D. Schmidt, John Willey and Sons, p. 379–393, 1999.

RATIONAL, Software Corporation. **Documentação oficial da UML**. Acesso em agosto de 2000. Disponível por http://www.rational.com/uml.html.

RIEL, Arthur. **Object-Oriented Design Heuristics.** Reading, MA: Addison-Wesley, 1996.

ROBERTS, Don; JOHNSON, Ralph. **Evolving Frameworks: A Pattern Language for Developing Frameworks.** In: Martin, Robert; Riehle, Dirk; Buschmann, Frank. Pattern Languages of Program Design 3. Reading, MA : Addison-Wesley, 1998. p. 471-486.

RUMBAUGH, J.; BOOCH, G.; JACOBSON, I. ***The Unified Software Development Process.*** ISBN: 0-201-57169-2, Addison-Wesley, 1999.

SAUVÉ, Jacques Philippe. **O que é um framework?**. 2003. Acessado em julho de 2003. Disponível por http://www.dsc.ufpb.br/~jacques/cursos/map/html/frame/oque.htm.

SCHMID, Hans Albrecht. **Design patterns for constructing the hot spots of a manufacturing framework.** Journal of Object-Oriented Progamming, 9(3), jan 1996.

SCHMID, Hans Albrecht. **Systematic framework design by generalization.** Communications of the ACM, v. 40, n. 10, 2002.

SCHMIDT, Doug. Using design patterns to develop reusable object-oriented communication software. Communications of the ACM, 38(10):65-74, October 1995.

SILVA, Ricardo Pereira e. Suporte ao desenvolvimento e uso de frameworks e componentes. Porto Alegre: PPGC da UFRGS, 2000. Tese de Doutorado.

SILVA Jr., A. G. da; HELBIG, R.; SCHIEFER, G. 1996. **A prototype computer aided quality assurance system for pig producers.** In Lokhorst, C. (ed.) proceedings of the 6th International Congress for Computer Technology in Agriculture, Wageningen.

SILVA Jr., A. G. da; HELBIG, R. 1997. **Gerenciamento da qualidade na produção de carne suína: análise comparativa entre sistemas integrados na Holanda, Dinamarca e Alemanha.**

SISCAFÉ. **Tecnologia em Softwares Agrícolas, 2000**. Acessado em agosto de 2001. Disponível por http://www.siscafe.com.br/itm_principal.html.

SOFTIMBRA. **WinBovC - Software para comercialização de carne bovina, versão 6.10**. 2003a. Acessado em setembro de 2003. Disponível por http://www.softimbra.pt/p_winboc.htm.

SOFTIMBRA. **WinBov - Software para controle de bovinos-leite, versão 6.20**. 2003b. Acessado em setembro de 2003. Disponível por http://www.softimbra.pt/p_winbov.htm.

SZTAJNBERG, Alexandre; LOQUES, Orlando; LOBOSCO, Marcelo. 1999. **Configurando Protocolos de Interação na Abordagem R-RIO.** Acessado em setembro de 2002. Disponível por http://www.inf.ufsc.br/sbes99/anais/SBES-Completo/03.pdf.

TALIGENT. ***Leveraging object-oriented framework***s. Taligent Inc. white paper, 1995.

TALIGENT, Inc. **Building Object-Oriented Frameworks.** 1997. Acessado em agosto 2002. Disponível por http://www.taligent.com.

VALE. **Vale Verde Assessoria Agropecuária e Informática.** 2002. É sede de importantes centros de pesquisa e desenvolvimento em agropecuária e informática.

Tem parceria com a Embrapa. Acessado em julho de 2002. Disponível por http://www.valeverde.com/produtos.html.

ZOOTEC. **A ZOOTEC possui parceria com a empresa responsável pelo *software* Congado.** 2002. Acessado setembro de 2003. Disponível por http://www.globalsite.com.br/zootec/parceria.htm.

WINDHORST, H. W. **Schweinehaltung und Schweinefleischproduktion in Dänemark um die Mitte der neunziger Jahre: Organisationsformen und räumliche Strukturen. Vechtaer Druckerei und Verlag, Vechta.** 1995.