FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” – UNIVEM
PROGRAMA DE MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

CHRISTIANO FERREIRA SILVA

PROPOSTA DE UMA FERRAMENTA DE APOIO AO DESENVOLVIMENTO DE APLICAÇÕES BASEADAS EM CASOS DE USOS E REGRAS DE NEGÓCIOS

MARÍLIA / SP
2006

CHRISTIANO FERREIRA SILVA

PROPOSTA DE UMA FERRAMENTA DE APOIO AO DESENVOLVIMENTO DE APLICAÇÕES BASEADAS EM CASOS DE USOS E REGRAS DE NEGÓCIOS

Monografia apresentada à Fundação de Ensino Eurípides Soares da Rocha, mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito para obtenção do título de Mestre em Ciência da Computação.

Orientador:

Prof. Dr. Edmundo Sérgio Spoto

MARÍLIA / SP
2006

SILVA, Christiano Ferreira
Proposta de uma Ferramenta de apoio ao Desenvolvimento de Aplicações Baseadas em Casos de Usos e Regras de Negócios / Christiano Ferreira Silva; orientador: Prof. Dr. Edmundo Sérgio Spoto. Marília, SP:[s.n.], 2006.
98 f.

Dissertação de Mestrado em Ciências da Computação – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha

1. Regras de Negócios 2. Sistema de Informação

CDD: 005.11

Christiano Ferreira Silva

Proposta de uma ferramenta de apoio ao desenvolvimento de componentes baseado em casos de usos e regras de negócios

Banca examinadora da dissertação apresentada ao Programa de Mestrado da UNIVEM / FEESR, para obtenção do Título de Mestre em Ciência da Computação.

A Comissão Julgadora:

Prof. Dr. Edmundo Sérgio Spoto

Profa. Dra Maria Istela Cagnin Machado

Prof. Dr. Auri Marcelo Rizzo Vincenzi

Marília, 05 de setembro de 2006.

AGRADECIMENTO

Agradeço, primeiramente, a Deus por ter me dado não apenas o dom da vida, mas também a inteligência para poder escolher as coisas corretas e importantes de minha vida.

Não poderia esquecer de minha esposa Fernanda que sempre esteve ao meu lado, em todos os momentos, motivando-me para que eu pudesse chegar ao final deste trabalho.

Agradeço, também, aos meus pais e parentes que me apoiaram e, não poderia deixar de agradecer ao meu orientador, Edmundo Spoto, o Dino, que acompanhou, de perto, não apenas os momentos de alegria, mas também os grandes momentos de dúvidas e sofrimentos, chegando, algumas vezes, a ver meu desespero nos olhos. Agradeço também a professora Dra. Maria Istela, que teve uma participação importante em momentos de dúvida, com sua presteza, sapiência e paciência se prontificou a me ajudar e tornou mais fácil o término deste trabalho, assim como os alunos Bruno Felipe e Helton Higute, pois sem a ajuda deles teria sido mais complicado concretizar algumas etapas.

SUMARIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API – *Application Programming Interface*

ASP – *Active Server Pages*

CGI – *Commom Gateway Interface* (Interface Comum de Gateway)

CORBA – *Commom Object Request Broker Architecture*

CRUD – *Create, Read, Update, Delete* (Inserir, Listar, Alterar e Excluir)

CSS – *Cascade Style Sheet*

DBA – *Database Administrator* (Administrador de Banco de Dados)

HTML – *HyperText Markup Language* (Linguagem de Formatação de Hipertexto)

HTTP – *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)

JDBC – *Java DataBase Connectivity*

JSP – *JavaServer Pages*

MER – Modelo Entidade Relacionamento

ODBC – *Open DataBase Connectivity*

OMG – *Object Management Group*

OMT – *Object Modelling Technique*

OO – Orientado a Objeto

PHP – *Hypertext Preprocessor*

RUP - *Rational Unified Process*

SGBD – Sistema Gerenciador de Banco de Dados

SGML – *Standard Generalized Markup Language*

SQL - *Structured Query Language* (Linguagem de Consulta Estruturada)

UML - *Unified Modeling Language*

WEB – *Word Wide Web* (Rede de computadores na Internet)

XHTML – *eXtensible HyperText Markup Language*

XML – *eXtensible Markup Language*

SILVA, Christiano Ferreira. **Proposta de uma Ferramenta de apoio ao Desenvolvimento de Aplicações Baseadas em Casos de Usos e Regras de Negócios**. 2006. 98 f. Dissertação de Mestrado em Ciências da Computação – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

RESUMO

Este trabalho apresenta uma proposta de ferramenta de apoio ao desenvolvimento de sistemas de informação baseados em casos de usos e regras de negócios, apresentando os conceitos e terminologias necessários, bem como um conjunto de idéias que objetiva mostrar a composição da arquitetura da ferramenta. Também é apresentado um estudo de caso que exemplifica o uso da ferramenta. Finalmente, são apresentados as conclusões e os trabalhos futuros propostos para a continuação deste projeto.

Palavras-chaves: Regras de Negócios, Sistema de Informação.

SILVA, Christiano Ferreira. **Proposta de uma Ferramenta de apoio ao Desenvolvimento de Aplicações Baseadas em Casos de Usos e Regras de Negócios**. 2006. 98 f. Dissertação de Mestrado em Ciências da Computação – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

ABSTRACT

The present dissertation proposes a tool that supports the development of Information Systems based in Use Case and Business Rules. This paper presents not only concepts but also fundamental terms and several ideas to illustrate the architecture of the tool. We also propose a Study Case, showing how this tool is used. Finally, we expose the conclusion and the future papers that will be proposed in order to continue this project.

Keywords: Information System, Business Rules.

INTRODUÇÃO

Atualmente, as informações estão crescendo em uma velocidade espantosa. Para se ter uma idéia, o volume de informações da *Internet* dobra a cada 100 dias (OBRIEN, 2004), e quanto mais conhecimento a humanidade adquire, mais complexos tornam-se os sistemas e as empresas. Considerando que os conceitos administrativos e as tecnologias não param de evoluir, os ambientes das empresas estão cada vez mais complexos (BISPO, 1998). A informação tornou-se uma ferramenta essencial para a sobrevivência das empresas, e, dessa forma, a Tecnologia da Informação é um instrumento que auxilia os executivos a tratarem as principais informações das empresas como um fator diferencial de competitividade na estratégia do negócio e não apenas como um apoio das atividades de administração cotidiana (DALFORO, 2000).

As mudanças dos conceitos administrativos e legislações fazem com que haja grande necessidade de manutenções em softwares, além das constantes evoluções que vão surgindo de acordo com as necessidades dos usuários. Estas manutenções podem degradar o código fonte e, conseqüentemente, as documentações, fazendo com que a única fonte de documentação confiável seja o próprio código fonte (CAGNIN, 2005).

Durante muito tempo, os executivos administraram as empresas por meio de erros e acertos. A partir da década de 40, surgiram várias técnicas gerenciais. Com a evolução dos conceitos administrativos e dessas técnicas de “como fazer”, os administradores passaram a administrar seus negócios baseados em normas, técnicas e regras**,** ocasionando uma mudança de paradigma de “como fazer” para “o que fazer”, que foi o grande desafio enfrentado pelos executivos da época. A utilização de Regras de Negócios vem justamente ao encontro desta nova realidade dos executivos, ajudando a integrar novos conceitos na administração do negócio.

As Regras de Negócios tem o propósito de automatizar o processo de negócio. Sendo assim, elas propõem que o desenvolvimento de sistemas com o uso da metodologia procedimental, aquela que descreve passo a passo como o trabalho deve ser feito, seja substituído pelo método declarativo, cujo objetivo é apenas informar o que deve ser feito na execução do trabalho e a partir desta informação, gerar o desenvolvimento do sistema.

A maioria dos usuários corporativos surpreende-se ao descobrir que no ramo da Tecnologia e Informação as melhores práticas para escrever códigos de software residem apenas na mente de quem as desenvolve. Porém, ultimamente, com a necessidade de se adaptar às rápidas mudanças de regulamentações do mercado e do governo e para operar com o máximo de eficiência possível, é necessário utilizar mecanismos de regras de negócios, como, por exemplo, repositórios de tais regras, para que os executivos (sejam eles de negócios ou de Tecnologia e Informação) tenham acesso facilitado (HILDRETH, 2005).

Segundo Bispo (1998), no início da década de 90, surgiram as primeiras ferramentas CASE e as Linguagens de Quarta Geração, que prometiam resolver os problemas dos usuários que necessitavam de informações rápidas e não tinham tempo para perder com o desenvolvimento de sistemas específicos. Porém, elas não eram versáteis o suficiente para atender todas as necessidades gerenciais, e, apesar de serem ágeis na manipulação e navegação de dados, eram pobres na análise gerencial.

A construção de uma ferramenta que dê suporte para o desenvolvimento de software baseado em operações de negócio (casos de usos e regras de negócio) é a principal motivação deste trabalho, cujo objetivo é buscar oportunidades de, a partir de projetos de Banco de Dados existentes, construírem os principais tipos de aplicações para a execução das atividades de negócio que servirão como mecanismo principal para as atividades da empresa.

Muitos dos novos conceitos administrativos são facilmente armazenados em regras, regulamentações, políticas e procedimentos, e estes, por sua vez, podem ser reutilizados. A

partir deste panorama, nota-se também uma maior facilidade de documentar tais regras, regulamentações, políticas e procedimentos, tornando o acesso a estas documentações muito mais claro, além de tornar mais ágeis as manutenções.

Motivação

A utilização das Regras de Negócios como apoio ao desenvolvimento de aplicações pode ser observada como uma tendência das ferramentas de desenvolvimento de software, pois elas facilitam não apenas o processo de desenvolvimento, mas também a documentação (fato interessante para a padronização de desenvolvimento), além do aspecto da reusabilidade de regras já cadastradas, o que proporciona uma economia de tempo e esforço de desenvolvimento e nas atividades de VV&T (Validação, Verificação e Testes).

Outro fator importante no desenvolvimento de software é a preocupação com o seu prazo de entrega e com o custo não superior ao que foi estimado (CAGNIN, 2005).

Considerando estes fatores que podem ser trabalhados e analisados no desenvolvimento de software, nota-se que muitos deles podem ser melhorados com a utilização de um repositório de Regras de Negócios. Sendo assim, a criação de uma ferramenta que permita controlar estas regras para reúso em novas aplicações, ou mesmo para as manutenções de aplicações existentes, pode gerar muitos benefícios para empresas que normalmente trabalham com softwares legados[1].

[1] Sistema Legado é um sistema de missão crítica, desenvolvido em algum momento do passado, que ainda é usado e vem sendo modificado ao longo do tempo sem submeter-se a ações sistemáticas de melhoria. (VALLE *et al.*, 2005)

Objetivo

Este trabalho tem como objetivo principal definir uma ferramenta que apóie o desenvolvimento de aplicações baseado na reusabilidade de um repositório de Regras de Negócios, tendo estas, registro de seus casos de uso que facilitem a documentação das aplicações criadas. Além do fator do reuso dessas regras, outro aspecto facilitador criado por esta ferramenta é a manutenibilidade da aplicação para a adaptabilidade às novas necessidades geradas pelas mudanças ocasionadas nos fatores internos e externos dos negócios.

Organização do Trabalho

Este trabalho está organizado em quatro capítulos como descrito a seguir:

No Capítulo 2, apresenta-se a revisão bibliográfica realizada para o desenvolvimento e entendimento deste trabalho.

No Capítulo 3, apresenta-se uma ferramenta criada para apoiar o desenvolvimento de aplicações baseadas em Regras de Negócios, objetivando mostrar os seus aspectos positivos.

No Capítulo 4, apresenta-se um estudo de caso da aplicabilidade da ferramenta e o funcionamento da ferramenta proposta.

No último capítulo, apresentam-se os resultados obtidos e as propostas de trabalhos futuros decorrentes deste trabalho.

No Apêndice A, apresenta-se o Script para a criação dos bancos da ferramenta desenvolvida, e no Apêndice B, o Script para a criação dos bancos de dados dos casos de teste utilizado no Capítulo 4.

2 - REVISÃO BIBLIOGRÁFICA

2.1 - Considerações Iniciais

Neste capítulo é apresentado o embasamento teórico necessário para a compreensão deste trabalho. Na Seção 2.2 são apresentados os conceitos de aplicação de software e também a evolução que vem ocorrendo na melhoria das técnicas de desenvolvimento de softwares baseados em componentes e regras de negócio. Na Seção 2.3 são apresentadas as utilizações e os conceitos de Regras de Negócios no uso de geração de Sistemas de Aplicação, sendo este o principal foco do trabalho. Na Seção 2.4 são apresentados os conceitos da *Unified Modeling Language* (UML), e um enfocando nos casos de uso. Na Seção 2.5 são apresentados os conceitos de Desenvolvimento Web e sua funcionalidade, além de algumas das técnicas utilizadas para este desenvolvimento. Na Seção 2.6 são apresentados alguns conceitos sobre bancos de dados e uma explanação sobre o PostgreSQL. Finalmente, na Seção 2.7 são apresentadas as conclusões finais deste capítulo.

2.2 - Aplicações de Software

Uma aplicação é a implementação de alguma função de negócio, por exemplo, “inserir um item do pedido”, “remover item do pedido” e “atualizar a quantidade disponível de algum pedido”. Em geral, uma aplicação envolve três partes ou componentes: aspectos de

apresentação; aspectos de banco de dados e aspectos específicos da função do negócio em si (DIAS, 2002).

Os aspectos de apresentação são aqueles relacionados com a interface de usuário final: exibindo telas, captando informações, exibindo mensagens de erro, produzindo relatórios impressos, e outros. Os aspectos de banco de dados são aqueles relacionados com a recuperação e a alteração da base de dados, em resposta às solicitações do usuário final e entradas nos formulários (são as partes das aplicações que interagem com o servidor de banco de dados, no uso de SGBD para a aplicação). Finalmente, os aspectos específicos à função do negócio da empresa, entendidos como a formalização da aplicação, são aqueles que especificam o processo atual a ser realizado para implementar a função de negócio ou, em outras palavras, são aqueles que efetivamente implementam as políticas e as práticas do negócio (DIAS, 2002).

De acordo com Pressman (1995), as “técnicas de quarta geração”, é um termo que abrange um amplo conjunto de ferramentas de software que tem algo em comum: cada uma delas possibilita que o desenvolvedor de software especifique alguma característica do software em um nível mais elevado.

Dos três componentes comentados anteriormente, os dois primeiros foram largamente automatizados. Os desenvolvedores de aplicação já não precisam escrever em código detalhado para projetar telas ou procurar mudanças em formulários nas telas: eles simplesmente invocam serviços de apresentação já integrados ao sistema para executar essas tarefas. Igualmente, eles não precisam escrever código detalhado para administrar dados: apenas invocam serviços de banco de dados já integrados ao sistema de banco de dados para executar essas tarefas (DIAS, 2002).

O passo mais recente na evolução dos ambientes de desenvolvimento de software foi representado pelo Microsoft Visual C++, sendo este uma grande e elaborada coleção de

ferramentas de desenvolvimento de software, utilizadas por interfaces providas de janelas. Esse sistema, juntamente com outros sistemas similares como o Visual BASIC, o Delphi e o Java Development Kit, da Sun Microsystems, oferecem maneiras simples de construir interfaces gráficas para os programas (SEBESTA, 2000).

Na visão de Elmasri (2005),um dos principais problemas com os sistemas de banco de dados pioneiros era a mistura entre os relacionamentos conceituais, o armazenamento físico e a localização de registros no disco. Neste caso, apesar de prover acessos muito eficientes, não oferecia flexibilidade suficiente e eficiente para novas consultas. Outra deficiência citada era a de que forneciam somente interfaces para a linguagem de programação.

Os bancos de dados relacionais foram originalmente projetados com o objetivo de separar o armazenamento físico dos dados de sua representação conceitual e prover uma fundamentação matemática para os bancos de dados (ELMASRI, 2005).

Nos anos 80, com a evolução das linguagens de programação orientadas a objetos, e com as necessidades de armazenar e partilhar os objetos complexos estruturados motivou-se o desenvolvimento dos bancos de dados orientados a objeto.

Korth e Silberschatz (1995) afirma que o grande objetivo de um sistema de bancos de dados é prover os usuários com uma visão abstrata dos dados**,** ou seja, o sistema omite certos detalhes de como os dados são armazenados e mantidos. Este conceito tem direcionado o projeto de estruturas de dados complexos para a representação de dados em bancos de dados. Uma vez que muitos dos usuários de bancos de dados não são treinados em computação, a complexidade está escondida por meio de diversos níveis de abstração que simplificam a interação do usuário com o sistema.

O terceiro componente, descrito anteriormente e que trata dos aspectos específicos à função de negócio da empresa, ainda é codificado manualmente, o que significa que os

desenvolvedores têm que escrever muito código procedimental sem apoio computacional (DIAS, 2002).

Escrever código procedimental é tedioso, demorado, propenso a erro e conduz ao famoso problema de *backlog*[2] de aplicação, além de ocasionar muitos outros problemas. A princípio, a solução para estes problemas é eliminar a codificação, ou seja, especificar declarativamente o processo empresarial por meio de regras de negócios, e deixar o sistema compilar essas regras no código procedimental, e executável, necessário (DIAS, 2002).

2.3 - Regras de Negócios

Segundo Bajec e Krisper (2001), a primeira aparição da frase “regras de negócios” no contexto atual foi no ano de 1984, quando Daniel Appelton escreveu um artigo chamado *Business Rule: The Missing Link.* Appelton discutiu problemas que eram causados por uma carência na padronização dos termos de negócios. Ele aponta que os analistas da área não utilizam termos de negócios padronizados para as soluções de aplicações comuns a todas as empresas, existindo, então, vários termos para uma mesma aplicação que varia de empresa para empresa.

Após a publicação desse artigo, vários profissionais da área de Sistemas de Informações perceberam que as regras de negócios podem ser capturadas em processos de máquina, e, desta forma, podem fazer cumprir as regras e assegurar que os processos dos negócios sejam controlados e conduzidos de acordo com os padrões, regras e procedimentos das empresas. Mais tarde, os pesquisadores de Sistemas de Informação e os profissionais desta área começaram a ter novas perspectivas para trabalhar com tais regras (BAJEC e

[2] Registro de atraso, registro de pedidos feitos, mas que por algum motivo ainda não puderam ser atendidos.

KRISPER, 2001).

Várias pesquisas foram realizadas com as ações que podem ser realizadas nos bancos de dados usando componentes ativos, como as *triggers*[3] e *database procedure*[4]. O desempenho dos bancos de dados e as funções de integridade dos dados podem ajudar na geração do código das aplicações, e estas funções são utilizadas como instrumentos para reforçar as regras de negócios (BAJEC e KRISPER, 2001).

Hoje em dia, a área promissora para a aplicação das regras de negócios é a pesquisa de integração entre as definições de negócios das empresas com o suporte dos Sistemas de Informação. Está cada vez mais óbvio que um Sistema de Informação pode atuar de forma a gerar atributos competitivos para as organizações se este estiver de acordo com as necessidades dos negócios, e, desta forma, os trabalhos tornam-se mais fáceis. Normalmente, as políticas, as regras e os procedimentos das empresas estão em constante mudança, os quais ocorrem espontaneamente devido às novas reações de mercado e dos resultados das estratégias, além das decisões tomadas pela empresa. Nem sempre estas mudanças são facilmente adaptadas aos sistemas de informação (BAJEC e KRISPER, 2001).

Essas mudanças que ocorrem nas empresas devem ser suportadas pelos softwares, e, normalmente, esses requerem novas implementações de novas regras, gerando uma necessidade de reprogramação para atingir os novos objetivos, metas e políticas das empresas.

2.3.1 - Conceitos básicos de Regras de Negócio

Segundo o *The Business Rule Project Group* (2000), uma regra de negócio pode ser

[3] Um *trigger* (gatilho) é um comando executado automaticamente pelo sistema com efeito colateral de uma modificação no banco de dados (SIMONETTO, 1998)

[4] *Database Procedure* é um procedimento que é executado toda vez que uma regra do banco de dados é disparada

definida segundo duas perspectivas: a de negócio e a de Sistemas de Informação. De acordo com a primeira, uma regra de negócio é uma diretiva destinada a influenciar ou guiar o comportamento do negócio como suporte à política de negócio que é formulada em resposta a uma oportunidade. De acordo com a segunda perspectiva, uma regra de negócio é uma sentença que define ou restringe algum aspecto do negócio. Pretende-se garantir a estrutura do negócio ou controlar a influência do comportamento do mesmo.

A regra de negócio é uma declaração que controla ou define alguns aspectos de um negócio, além de definir sua estrutura ou reger seu processo meta. Muitos requisitos de aplicações fazem parte de uma categoria conhecida como regra de negócio, a qual expressa requisitos computacionais que determinam ou afetam o modo como um negócio é administrado, por exemplo, ela pode indicar como os clientes de uma empresa são tratados, como os recursos são utilizados em uma linha de produção e como as situações especiais são tratadas pelas aplicações (KLINGER e KROTH, 2000).

Nos últimos anos, as técnicas de análise de sistemas têm evoluído muito com o intuito de melhorar os métodos de descrever os aspectos de vários negócios ou mesmo das agências do governo. Elas são responsáveis pela maior facilidade em projetar os fluxos de informações das organizações, as seqüências das ações, a estruturação das informações operacionais e assim por diante.

A construção de Sistemas de Informação requer flexibilidade, e, dessa forma, o trabalho com as regras de negócios e com a habilidade para criar e dar manutenção a tais regras, objetivando o seu ajuste às necessidades das organizações, e o trabalho com a criação de novas regras de negócios que não tenham ambigüidades. Todos estes fatores facilitam a implementação e também o gerenciamento dos negócios (DATE, 2000).

Recentemente, enfatiza-se a mudança de paradigma que considera as regras de negócios como elementos muito importantes para os Sistemas de Informação. Muitos

processos dos negócios são executados respeitando o que foi prescrito em uma determinada ação ou obrigação para um conjunto de ações possíveis. As regras de negócios são baseadas em éticas, cultura, regulamentos ou crenças organizacionais (HERBST *et al.*, 1994).

As regras de negócios das organizações mudam constantemente, não apenas por novas estratégias, mas também por mudanças no ambiente onde as empresas se encontram. Estas mudanças podem gerar novas regras, ou mesmo mudanças nas regras existentes.

Uma grande vantagem da programação baseada em regras de negócios é o reuso de sistemas de aplicações. Segundo Sommerville (2003), uma vantagem óbvia do reuso de software é que os custos gerais de desenvolvimento são reduzidos.

Para que um projeto ou um desenvolvimento de software seja baseado em reuso, são imprescindíveis três requisitos: (SOMMERVILLE, 2003)

1. Deve ser possível encontrar componentes reutilizáveis apropriados;
2. O responsável pelo reuso dos componentes precisa ter certeza de que os componentes se comportarão como especificado e de que serão confiáveis;
3. Os componentes devem ter a documentação associada para ajudar o usuário a compreendê-los e adaptá-los a uma nova aplicação.

Se uma regra de negócio for bem definida no processo de desenvolvimento, a mesma pode ser aplicada em várias etapas de desenvolvimento do software. Por exemplo, se definida uma regra “Calcule o Valor do ICMS”, ela deve ser utilizada tanto na aplicação de inclusão quanto na alteração ou exclusão de um item no pedido.

Outra vantagem que deve ser avaliada na utilização das regras de negócios é que as mesmas não necessitam, obrigatoriamente, seguir uma ordem para serem citadas no desenvolvimento de uma aplicação, ou seja, elas podem ser criadas e, por suas próprias dependências, deverão ser organizadas para que sejam executadas. Para que uma regra de negócio, como por exemplo, “Calcular o Total do Pedido”, seja executada, é necessário,

primeiramente, que se execute a regra “Calcular o Valor do Item”, e para que esta seja executada, é necessário a quantidade e o valor de cada um dos itens. Tanto a seqüência quanto a dependência devem ser verificadas pela própria ferramenta e a execução deve ser na ordem correta.

Diferentemente do desenvolvimento procedimental, o desenvolvimento baseado em regras de negócios gera uma diminuição da quantidade de linhas de código, como apresentado por Dias (2002). Esta quantidade de linhas torna-se menor, pois na utilização das regras de negócio as declarações são mais concisas.

Tanto Herbst *et al.*(1994) como Oliveira (1998) afirmam que as regras de negócio podem ser divididas em regras de consistência e regras de atividade. As primeiras, também conhecidas como regras de integridade, definem a integridade entre estados válidos da base de dados, enquanto que as segundas descrevem seqüências de operações que devem ser realizadas para atender uma determinada função da aplicação. As regras de consistência ainda são classificadas em ativas e passivas. Uma restrição de integridade passiva previne ações a serem executadas em uma dada seqüência para garantir a integridade do dado, já uma restrição de integridade ativa mantém a consistência do dado durante uma manipulação do mesmo em alguma operação.

As organizações possuem diferentes interesses, especialidades e níveis, os quais podem ser definidos como nível organizacional, nível de conhecimento, nível gerencial e nível estratégico. Para cada um destes níveis existem sistemas estruturados para suportá-los. Atualmente, o nível de conhecimento é o que mais cresce nas organizações (LAUDON e LAUDON, 2004) e as regras de negócio são criadas pelos trabalhadores de conhecimento.

2.3.2 - Análise Comparativa de Desenvolvimento Procedimental e Desenvolvimento baseado em Regras de Negócios.

Dias *et al.* (2002) fez uma análise comparativa do desenvolvimento de Sistemas de Informação utilizando o desenvolvimento Procedimental e de Regras de Negócios declarativas, nesta análise foram comparados os fatores de qualidade de software definidos pela norma ISO/IEC 9126 (ver Apêndice C), e para isto foram comparados especificamente o ambiente Delphi para o desenvolvimento Procedimental e a ferramenta GeneXus para o desenvolvimento baseado em Regras de Negócios, os resultados obtidos estão apresentados no Quadro 2.1.

Segundo Dias *et al.* (2002),

> "GeneXus é uma ferramenta para desenvolvimento de aplicações. Seu objetivo é auxiliar os analistas de sistema na implementação de aplicações no menor tempo e com a melhor qualidade possível. Em linhas gerais, o desenvolvimento de uma aplicação implica tarefas de análise, projeto e implementação. O caminho seguido por GeneXus para atingir estes objetivos é o de liberar as pessoas das tarefas automatizáveis, por exemplo a implementação, permitindo a elas concentrar-se nas tarefas realmente difíceis e não automatizáveis, como compreender o problema do usuário. Do ponto de vista teórico, GeneXus é uma ferramenta associada a uma metodologia de desenvolvimento de aplicações"

As comparações realizadas foram relativas a confiabilidade da representação, a confiabilidade conceitual e também da usabilidade.

Como já mencionado, na comparação de confiabilidade de representação, o desenvolvimento baseado em Regras de Negócios necessitou de um menor número de linhas de código, mas, além disso, também apresentou maior Clareza, uma vez que as Regras de Negócios puderam ser identificadas com maior facilidade, já com relação aos subfatores, estilo e Modularidade (ver Quadro 2.1) apresentaram resultados semelhantes.

Quadro 2.1 - Resultado das Comparações (Baseado em DIAS *et al*, 2002)

	Fatores	Subfatores	SI Procedimental (Delphi)	SI com RN declarativa (GeneXus)
Confiabilidade da Representação	Legibilidade	Clareza	Menor	Maior
		Concisão	Menor	Maior
		Estilo	Semelhantes	
		Modularidade	Semelhantes	
	Manipulabilidade	Disponibilidade	Manual	Automática
		Estrutura	Pior	Melhor
		Rastreabilidade	Pior	Melhor
Confiabilidade Conceitual	Fidedignidade	Precisão	Atende	Atende
		Completeza	Atende	Atende
		Necessidade	Atende	Atende
	Integridade	Robustez	Manual	Automática
		Segurança	Manual	Automática
Utilizabilidade	Manutenibilidade		Pior	Melhor
	Operacionalidade		Semelhantes	
	Portabilidade		Menor	Maior
	Avaliabilidade		Pior	Melhor
	Reutilizabilidade		Semelhantes	

Ainda analisando o Quadro 2.1, nos fatores de Confiabilidade Conceitual, os dois Sistemas de Informação desenvolvidos implementaram satisfatoriamente o que foi especificado e projetado, e nos fatores de usabilidade, o subfator Manutenibilidade é que se torna relevante no que tange ao desenvolvimento baseado em Regras de Negócios (DIAS *et al*,2002).

2.4 - UML (*Unified Modeling Langauage*)

A UML (SOLBERG e BERRE, 1998; FURLAN, 1998) é sucessora de um conjunto de métodos de análise e projetos orientados a objetos. A UML é uma linguagem que unifica

as Metodologias OO de Booch e Rumbaugh por meio de seus diagramas. É, portanto, uma linguagem de modelagem que utiliza uma notação para descrever o modelo pretendido. A UML possui diversos diagramas, como o de caso de uso, de atividades, de colaboração, de seqüência, de estado, de classe, de objeto, de componentes e de implantação.

O Diagrama de Caso de Uso, ou *Use Case*, é um tipo de modelagem usada para identificar como uma aplicação se comporta em várias situações que podem ocorrer durante sua operação. O Diagrama de Caso de Uso descreve tanto a aplicação quanto seu ambiente e a relação entre os dois. Os componentes desse diagrama são os atores e os casos de uso.

Os Casos de Uso são utilizados para descrever um comportamento que o software a ser desenvolvido apresentará quando estiver pronto, ou seja, eles não descrevem como o software será construído, mas sim como deverá se comportar (LARMAN, 2004).

O ator representa qualquer entidade que interage com a aplicação. Pode ser uma pessoa ou uma máquina, por exemplo. Assim, ele deve ter as seguintes características: primeiro, deve relacionar-se com a aplicação; segundo, deve representar os papéis que um usuário da aplicação desempenha, e, terceiro, deve interagir ativamente com a aplicação, visto que também é um receptor passivo de informações.

Cada caso de uso deve conter, no mínimo, um cenário principal, que é o conjunto de passos a serem realizados, caso não haja nenhum problema na execução do programa. Caso haja a possibilidade de existência de alguma não conformidade no programa, estas deverão ser apresentadas nos cenários alternativos ou secundários.

Os atores têm um papel externo às aplicações, porém, são eles que iniciam os casos de uso. A aplicabilidade destes casos é benéfica por serem mais ágeis na captura de requisitos de software, os quais podem ser facilmente adicionados ou removidos de um projeto de software de acordo com as prioridades. Eles também são utilizados para estimar, escalonar e validar os esforços necessários para o desenvolvimento de uma aplicação, além de facilitar o

entendimento das pessoas de negócios, tornando-se, assim, um grande facilitador entre quem desenvolve as aplicações e quem as utiliza (LARMAN, 2004).

Para representar a interação entre os objetos, a UML utiliza o diagrama de seqüência e o diagrama de colaboração. O primeiro mostra a interação entre os objetos ao longo do tempo, apresentando quais são os que participam da interação e a seqüência de mensagens trocadas. O diagrama de colaboração é um modo alternativo para representar a troca de mensagens entre um conjunto de objetos, além de mostrar a interação organizada em torno dos objetos e suas ligações uns com os outros.

Furlan (1998) afirma que o diagrama de classe é a essência da UML resultando de uma combinação de diagramas propostos pela OMT, Booch e vários outros métodos. Trata-se de uma estrutura lógica em uma superfície de duas dimensões mostrando uma coleção de elementos declarativos do modelo.

Para a representação dos aspectos estáticos dos processos de negócios, a UML utiliza o diagrama de classes, e para especificar uma seqüência de trocas de mensagens que representa o comportamento e as interações entre um conjunto de objetos, utiliza o digrama de interação. Este, por sua vez, deve ser usado quando se deseja visualizar o comportamento de vários objetos dentro de um único caso de uso a partir das mensagens que são trocadas.

Para a representação dos aspectos dinâmicos, a UML utiliza o diagrama de atividades. Para Furlan (1998), o diagrama de atividades deve ser utilizado quando a maioria dos eventos representar a conclusão de ações geradas internamente.

Ainda, na UML, tem-se o diagrama de componentes, que é um gráfico de componentes conectados pelos relacionamentos de dependências**,** os quais podem ser associados a outros por retenção física que representa relacionamentos de composição. Este diagrama mostra as dependências entre os componentes de software.

2.4.1 - Casos de Uso

Para entender melhor o conceito de caso de uso, é necessário, inicialmente, entender melhor o ator. Este pode ser uma pessoa, um sistema ou mesmo o que representava na análise estruturada de entidade externa. Ele realiza uma atividade e sempre atua sobre um caso de uso.

Segundo Medeiros (2004), um caso de uso é uma macro-atividade que encerra diversas tarefas ou atividades. Pode ser, também, uma representação descritiva de variadas ações para a realização desta macro-atividade. O limite de um caso de uso é uma decisão pessoal, além do mais, ele deve ser bastante detalhado, o que não significa que se deva escrever um tratado maçante e longo demais para ser lido.

Medeiros (2004) ainda afirma que um caso de uso somente é possível por duas vias: a observação e a entrevista. A observação só é possível em casos em que a atividade é repetitiva, a entrevista somente é possível entre humanos.

Larman (2004) aponta que a idéia de casos de uso para descrever requisitos funcionais foi introduzida em 1986 por Ivar Jacobson. Esta idéia foi amplamente aceita e suas principais virtudes são a simplicidade e a utilidade. Possivelmente, o passo mais influente, abrangente e coerente na definição do que são casos de uso e sobre como escrevê-los veio de Alistair Cockburn.

Em termos informais, um caso de uso é uma coleção de cenários relacionados de sucesso e fracasso, os quais descrevem atores usando uma aplicação como meio para atingir um objetivo.

No Quadro 2.2 é apresentada uma sugestão de estrutura para um caso de uso (MEDEIROS, 2004)

Quadro 2.2 - Sugestão de estrutura de Caso de Uso (MEDEIROS, 2004)

Nome Referencia **Nome do Caso de Uso**	Verbo no infinitivo (informar, comprar, pagar...)
Breve Descritivo	Descrição que informa do que se trata este caso de uso
Pré-condições	Descrição que informa o que é necessário acontecer para que se inicie este caso de uso.
Atores Envolvidos	
Cenário Principal	A descrição de uma tarefa que representa o mundo perfeito, sem exceções. Verbos no presente do indicativo ou substantivos, iniciando a frase com (Registra, Compra, Seleciona, Informa...)
Cenário Alternativo	Qualquer situação que represente uma exceção de um cenário principal.
Requisitos Especiais	Qualquer situação não contemplada anteriormente
Dados	Tipo de dados que forem encontrados durante a descrição do Caso de Uso. Aqui informamos: texto, número, data, etc., ou mesmo o tipo de dado e seu tamanho, conforma a linguagem a ser utilizada, caso vocês a conheçam**.**
Observação **Analista de Negócios: _____** **Entrevistado: _____** **Área: _____** **Data: ____/____/____** **Versão:** XPTO	

O Caso de uso é um conceito amplamente difundido e utilizado para a documentação e o desenvolvimento de requisitos.

Segundo Souza *et al.*(2005), no *Rational Unified Process* (RUP), há 5 (cinco) padrões para as descrições dos casos de uso, são eles: padrão de caso de uso CRUD; padrão para documentação de atributos, padrão caso de uso relatório; padrão caso de uso transação e padrão caso de uso assistente, sendo que todos este possuem um relacionamento que pode ser observado na Figura 2.1 (obtida em (SOUZA *et al.*, 2005)).

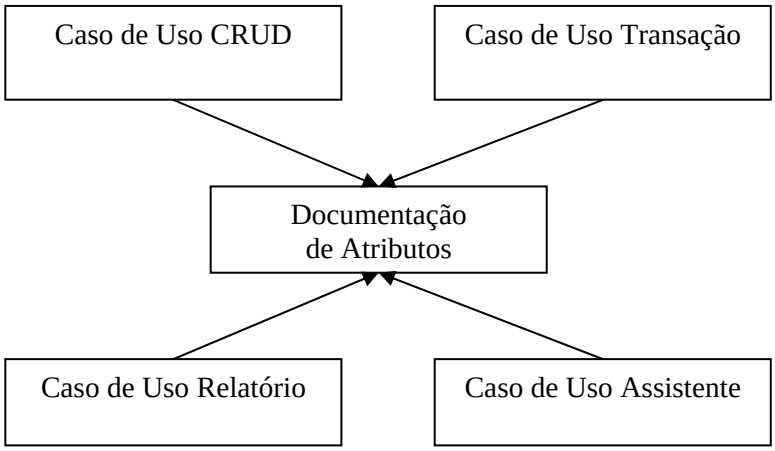

Figura 2.1 - Relacionamento entre os padrões apresentados (SOUZA *et al.*, 2005)

O Padrão Caso de Uso CRUD é utilizado para a documentação dos requisitos de manutenção em sistemas de informação por meio do uso de modelos e especificações de casos de uso. Os requisitos de manutenção são caracterizados por operações de Inclusão, Consulta, Alteração e Exclusão.

O Padrão Documentação de Atributos é utilizado em sistemas de informação. Os atributos das entidades possuem diversas características como: nome, descrição, obrigatoriedade, validações, semânticas, entre outras. Portanto, a documentação desse atributo deve ser elaborada de forma que essas características não sejam esquecidas.

O Padrão Caso de Uso Relatório é utilizado em sistemas de informação, no qual uma grande quantidade de dados é armazenada freqüentemente. Neste contexto, surge a necessidade de visualizar, exportar ou imprimir dados armazenados com o objetivo de conferir, analisar e tomar decisões com base neles.

O Padrão Caso de Uso Transação é utilizado para a documentação dos requisitos de operações que são tratadas como um comando atômico que processa várias transações, tipicamente operações batch e operações que requerem apenas um comando de início do caso de uso pelo usuário, tendo pouca entrada de dados e interação com o sistema.

O Padrão Caso de Uso Assistente é utilizado para a documentação dos requisitos de

operações complexas que são executadas em diversos passos, onde as decisões ou os dados necessitam ser informados em cada passo por meio da interação com o usuário.

Como exemplo de um caso de uso, pode-se citar o processo de comprar coisas em uma loja quando um terminal de ponto de vendas é usado.

Caso de uso:	**Comprar Itens**
Atores:	Cliente, Caixa
Descrição:	Um Cliente chega a um ponto de pagamento, com vários itens que deseja comprar. O Caixa registra os itens de compra e recebe o pagamento. Ao término, o Cliente deixa a loja com os itens.

2.5 - Desenvolvimento WEB

Segundo Conallen (1999), as aplicações *Web* possuem os *Web sites* e os sistemas Web. Nos *Web sites*, um usuário pode acessar e visualizar um documento em um determinado endereço. O *browser* do usuário envia uma solicitação deste documento para outro computador onde está armazenado o documento. Esta requisição é um meio pelo qual um software identifica o *Web server*. Este, por sua vez, executa o serviço solicitado, que monitora as atividades da rede, analisa as necessidades especiais de formatação do documento e mostra o documento solicitado, como pode ser observado na Figura 2.2.

Essa situação utiliza arquivos estáticos, ou seja, documentos que nunca sofrem alteração, independente da forma ou localização da solicitação (ZORZO, 2004).

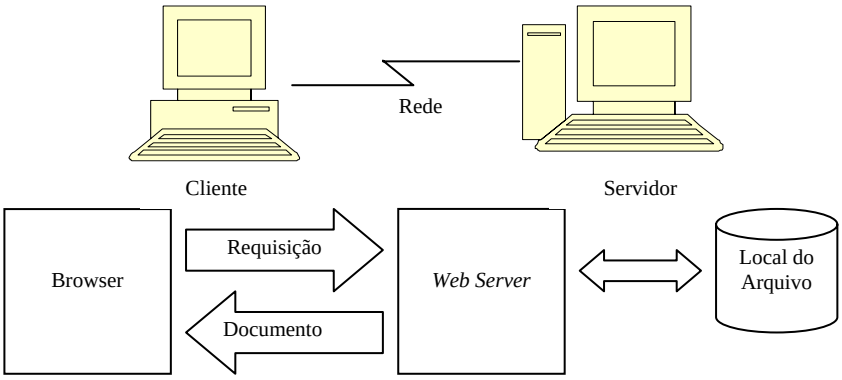

Figura 2.2 - Sistema Básico da Web (CONALLEN, 1999)

A construção das aplicações *Web* ou suas extensões é um sistema *Web* adicionado de funcionalidades de negócios. Simplificando, uma aplicação *Web* é um sistema em que os usuários podem executar a lógica dos negócios com um *browser Web*. Esta não é uma definição muito precisa, mas mostra, de forma sucinta, a concepção de um sistema *Web* (CONALLEN, 1999).

No entanto, a maioria das aplicações exige que os dados sejam fornecidos por servidores *Web* de forma dinâmica. O conteúdo dinâmico exige que o *Web Server* faça algum processamento adicional na solicitação feita. Além da identificação da solicitação, o *Web Server* vai depender de valores transferidos do cliente para o servidor, que são parâmetros para a geração da resposta dinamicamente (ZORZO, 2004).

As aplicações Web utilizam-se de uma página em HTML (*HyperText Markup Language*), interpretada pelo *browser*, para interagir com o usuário. Outras tecnologias podem ser misturadas ao HTML para a construção de uma interface mais poderosa, com um visual mais adequado, além de proporcionar recursos que o HTML isoladamente não é capaz de produzir (Frydrych, 2001).

Os servidores Web trabalham recebendo e enviando requisições e respostas no formato do protocolo *Hypertext Transfer Protocol* (HTTP), sendo este do nível de aplicação,

possuindo objetividade e rapidez necessárias para suportar sistemas de informação distribuídos cooperativos de hipermídia (FIELDING *et al*., 1999).

As aplicações *Web* diferem dos softwares tradicionais por serem executadas em ambientes heterogêneos e autônomos (CHAN, 2006).

Diversas tecnologias têm surgido para o desenvolvimento dessas aplicações, sendo que a maioria utiliza as vantagens fornecidas pelo paradigma Orientado a Objeto (OO). Tem-se, como exemplo, as tecnologias convencionais HTML (*HyperText Markup Language*) e CGI (*Commom Gateway Interface*), que possuem restrições em termos de desempenho, funcionalidade e usabilidade, dentre outros. Como exemplo de tecnologias recentes que utilizam as vantagens da OO, podemos citar como exemplo Java e CORBA (*Commom Object Request Broker Architecture*).

Segundo BOS *et al*.(1998), as linguagens e padrões mais utilizados para tal desenvolvimento são o HTML ou XHTML (que é uma extensão do HTML), CSS (*Cascade Style Sheet*), JavaScript e, para adicionar alguns recursos mais avançados e adicionar dinamismo às páginas, são utilizadas linguagens de programação que rodem no servidor da aplicação Web, tais como PHP, JSP, ASP, ColdFusion, entre outras.

Com a difusão da *Web*, sistemas podem ser entregues e desenvolvidos em quase todos os lugares, contudo, existem algumas restrições. A interação de arquiteturas *Web* é feita de acordo com o paradigma de estímulo/resposta, sem o controle direto do cliente no servidor. O cliente deve iniciar a interação com o sistema e, caso esse não seja o comportamento esperado, outros elementos devem ser acrescentados à arquitetura, gerando sistemas mais complexos e, possivelmente, mais sensíveis (CONALLEN, 1999).

2.5.1 - HTML

HTML (*Hyper Text Markup Language*) é uma linguagem de marcação que descreve a estrutura, o conteúdo e a apresentação de um documento e sua relação com outros documentos.Utiliza os conceitos do Hyper Texto e da Hipermídia para apresentar, em um mesmo ambiente, dados, imagens e outros tipos de mídia, como vídeos, sons e gráficos. O HTML é um subconjunto do *Standard Generalized Markup Language* (SGML) e utiliza rótulos (*tags*) que definem a aparência e o formato dos dados, sendo padronizado pelo *Object Management Group* (OMG). É interpretado por qualquer *browser*, em qualquer plataforma (COSTA, 2001) (CONALLEN, 1999).

2.5.2 - CGI

O mecanismo original para o processamento de entrada dos usuários em um sistema *Web* é o CGI (*Commom Gateway Interface*), o qual é o caminho padrão para os usuários executarem aplicações nos servidores (CONALLEN, 1999).

O CGI é um padrão para interfaceamento de aplicações externadas com servidores, como um *Web Server*, por exemplo. Também funciona como a aplicação mais básica para acessar os recursos do sistema no servidor, e foi também a primeira tecnologia para desenvolvimento de aplicações *Web* (COSTA, 2001).

2.5.3 - Java

Java é uma linguagem segura, orientada a objetos, independente da plataforma. A orientação a objetos é empregada no processo de desenvolvimento de software, no qual um programa é concebido como um grupo de objetos que trabalham juntos, comunicando-se e cooperando para solucionar o problema proposto. A neutralidade de plataforma é a capacidade de um programa ser executado sem modificações em diferentes ambientes computacionais, tanto de software como de hardware. Os programas em Java são compilados para um formato denominado *bytecode* e que podem ser executados por diferentes máquinas virtuais Java, presentes em diferentes plataformas de software e de hardware (ZORZO, 2004).

A linguagem Java da *Sun Microsystems*, utilizada na forma de *applets*, é capaz de estender as funcionalidades dos *browsers*, adicionando recursos que eram impossíveis de serem construídos com o HTML puro. Os *applets* são miniprogramas executados sob o *browser*, por meio da Java Virtual Machine (COSTA, 2001).

Uma desvantagem dessa estratégia é a de aguardar o término da transferência do código do *applet*, residente no servidor, para a máquina do usuário para que, então, seja iniciada a sua execução. Por outro lado, há a vantagem de não ser necessário aguardar o servidor realizar todos os cálculos para transferir a página estática para o cliente. *Applets* funcionam bem em conexões rápidas ao servidor *Web*, como em redes locais, mantendo uma cópia única do código de acesso a todos os usuários. Quando o código for alterado, seus usuários passam a utilizar a nova versão independente da notificação de atualização (ZORZO, 2004).

JavaScript e Java *applets* são utilizados para determinar os clientes dinâmicos. *Java Server Page* (JSP) e *servlets* são utilizados nos servidores (CONALLEN, 1999).

Servlets é um tipo de aplicativo Java que, executado no Web Server, permitem um funcionamento similar ao CGI. Os *Servlets Java* são multiplataforma e oferecem bom

desempenho (COSTA, 2001). Os *Servlets Java* são executados em um servidor web por um módulo especial, o motor de *servlet*, que otimiza a sua execução com o mínimo de recursos do servidor *web*. Há diversos servidores *web* que aceitam *servlets*, devendo ser observado as especificidades de cada um (ZORZO, 2004)

Segundo Zorzo (2004), os *servlets*, por serem escritos em Java, são independentes de plataforma, podendo ser criados, compilados e executados em plataformas diferentes.

De acordo com Costa (2001), *Java Server Pages* (JSP) tem se mostrado uma tecnologia bastante promissora. É baseada em Java com código Java embutido na página HTML, o qual é interpretado a cada requisição pelo *Web Server*. JSP é implementada por meio de *servlets*, mas o programador não precisa mais ter a preocupação de gerar todo o código HTML, como era feito em *servlets*. JSP permite a separação da parte dinâmica, escrita por scripts especiais, das partes estáticas, escritas como HTML usual. As páginas JSP incorporam o código Java no código HTML estático (ZORZO, 2004).

Para utilizar a tecnologia *JavaServer Pages* (JSP) é necessário que o servidor W*eb* esteja integrado a um contêiner JSP. O servidor Apache Tomcat possui contêiner JSP disponível, embora outros servidores também o possuam (ZORZO, 2004).

Segundo Kurniawan (2004), uma das diferenças de trabalhar com JSP ao invés de *servlet*, é que o programador JSP não tem vínculo com o *webdesigner* no que diz respeito à programação, ou seja, quando programa-se em *servlet*, o programador precisa enviar todo código HTML pelo seu programa, sendo assim, mesmo que uma página tenha pouco código, o programador precisa mandar uma longa instrução HTML. Além do mais, quando o w*ebdesigner* quiser fazer qualquer tipo de alteração, é necesário pedir para que o programador *servlet* faça a alteração. Já a programação JSP, além de suportar HTML e XML (*eXtensible Markup Language*) é mais dinâmica, ou seja, o programador JSP adiciona seu código dentro de *tags* especiais em qualquer lugar do código HTML.

Java DataBase Connectivity (JDBC) é uma *Application Programming Interface* (API) desenvolvida pela *Sun* com objetivo de fazer o acesso ao banco de dados de forma padronizada para que seja possível o acesso a qualquer tipo de Sistema Gerenciador de Banco de Dados (KURNIAWAN, 2004).

Visando o ganho de tempo e estabilidade no desenvolvimento desta API, a *Sun* baseou-se em outras APIs bem sucedidas, como por exemplo, a *Open DataBase Connectivity* (ODBC), sendo que esta foi criada para padronizar o acesso a banco de dados no ambiente Windows.

Esta API tem a finalidade de ser flexível e simples para os programadores. De acordo com Reese (2001), “o conjunto das classes que implementam as interfaces JDBC para um determinado mecanismo de banco de dados é chamado driver JDBC”, sendo que estes são fornecidos e desenvolvidos por seus respectivos fabricantes. Como conseqüência, o programador se preocupa somente com sua aplicação, conforme mostrada na arquitetura da Figura 2.3.

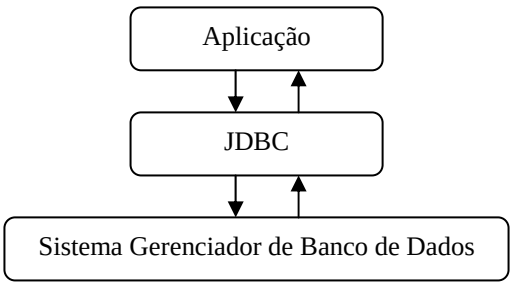

Figura 2.3 - Arquitetura JDBC

Na Figura 2.4 é mostrado o relacionamento entre os componentes e a habilitação das tecnologias com o Servidor Web. O banco de dados na figura está nos recursos do *server-side*, incluindo sistemas internos e outros aplicativos. Na Figura 2.4 é mostrado como o módulo

compilado intercepta as requisições das páginas *Web* de um servidor *Web* e o sentido e ações de um servidor *Web* (CONALLEN, 1999).

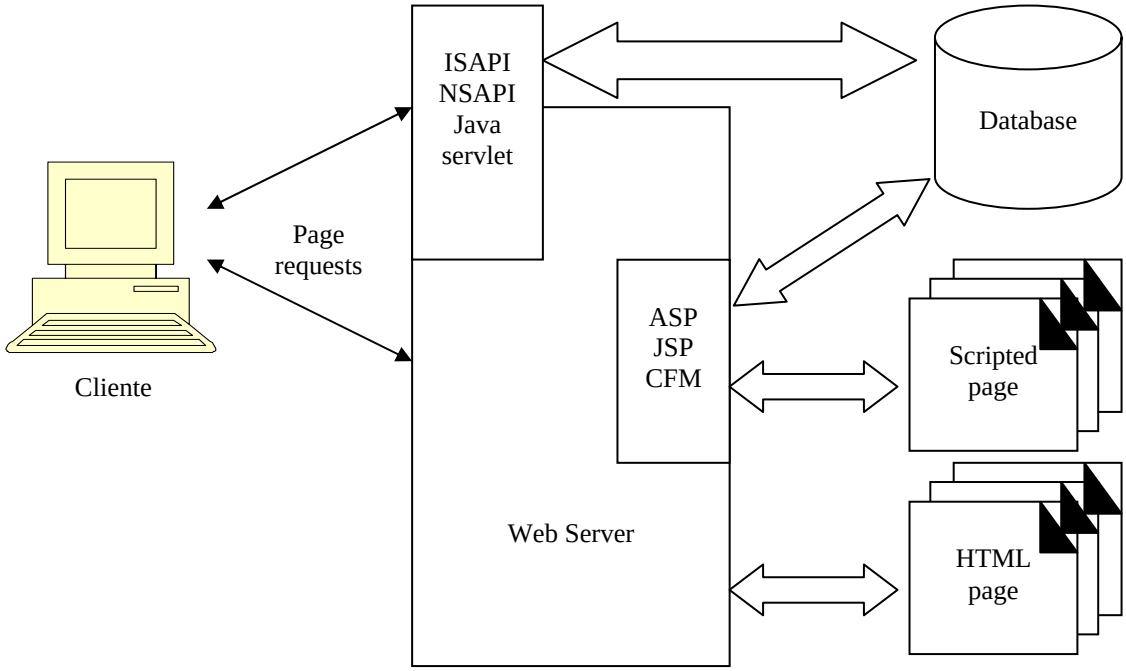

Figura 2.4 – Relacionamento entre os componentes de um Web Server (CONALLEN, 1999)

2.6 - Banco de Dados

Elmasri e Navathe (2005) descreve Banco de Dados como:

> "Um banco de dados é uma coleção de dados relacionados. Os dados são fatos que podem ser gravados e que possuem um significado implícito. Por exemplo, considere nomes, números telefônicos e endereços de pessoas que você conhece. Esses dados podem ter sido escritos em uma agenda de telefones ou armazenados em um computador, por meio de programas como Microsoft Acess ou Excel. Essas informações são uma coleção de dados com um significado implícito, conseqüentemente, um banco de dados."

Além do mais, um banco de dados pode ser gerado e mantido manualmente, ou por

meio de um SGBD (Sistema Gerenciador de Banco de Dados), ou seja, o administrador do Banco de Dados pode criar suas próprias ferramentas ou utilizar o SGBD para gerenciar a definição, construção, manipulação, compartilhamento, proteção e manutenção do banco de dados (ELMASRI e NAVATHE, 2005).

Segundo Harrington (2002), toda preocupação de como os registros estão fisicamente armazenados deve ser do SGBD e não do usuário ou do programa. O papel do usuário é solicitar apenas a manipulação dos dados, é o SGBD que deve preocupar-se com a tradução desta solicitação e o armazenamento físico dos dados.

De acordo com Korth e Silberschatz (1995), todo banco de dados possui restrições de integridade que fornecem meios de assegurar que as mudanças feitas no banco de dados por usuários autorizados não resultem na perda da consistência dos dados. Para o Modelo Entidade-Relacionamento (MER), as restrições são apresentadas na forma de declaração de chaves e na forma de um relacionamento. Além destes dois tipos de restrições utilizadas no Modelo Entidade-Relacionamento, há, também, as restrições de domínio, a integridade referencial, as dependências funcionais, as asserções e os gatilhos (*triggers*).

Segundo Elmasri e Navathe (2005), em um banco de dados relacional são permitidas várias restrições para os valores reais, e estas são derivadas das regras do mini-mundo que o banco de dados representa. Estas restrições em bancos de dados podem, geralmente, ser divididas em três categorias principais:

1. Restrições inerentes baseadas em modelos – inerentes ao modelo de dado;
2. Restrições baseadas em esquemas – normalmente expressadas diretamente nos esquemas dos modelos de dado; e
3. Restrições baseadas em aplicações – não podem ser expressas diretamente nos esquemas do modelo de dados, e, por isso, devem ser expressas e impostas pelos programas de aplicação.

Outra categoria importante de restrições é a de dependência de dados, que incluem as dependências funcionais e as multivaloradas.

Restrições de domínio são a forma mais elementar de restrições de integridade. Elas são testadas facilmente pelo sistema cada vez que um novo item de dado é inserido no banco de dados (KORTH e SILBERSCHATZ, 1995). As restrições de domínio especificam que, dentro de cada tupla, o valor de cada atributo deve ser um valor atômico do domínio (ELMASRI e NAVATHE, 2005).

A restrição de integridade referencial é classificada entre duas relações e é usada para manter a consistência entre as tuplas nas duas relações (ELMASRI e NAVATHE, 2005). A integridade referencial é utilizada devido ao fato de, freqüentemente, desejarmos assegurar que um valor que aparece em uma relação para um dado conjunto de atributos apareça também em um certo conjunto de atributos em outra relação (KORTH, 1995).

O que Korth e Silberschatz (1995) chama de dependência funcional, Elmasri e Navathe (2005) chama de dependência de chave, o que significa que duas tuplas distintas não podem ter valores idênticos para os atributos da chave.

As asserções e gatilhos (*triggers*) são definidos por Elmasri Navathe (2005) como restrições de integridade semântica. Estas restrições podem ser específicas e impostas dentro dos programas de aplicação que atualizam o banco de dados, e, geralmente, são utilizadas nos programas de aplicação, pois são difíceis e complexas de serem usadas corretamente.

Uma asserção é um predicado expressando uma condição de desejo de que o banco de dados sempre satisfaça, já o gatilho (*trigger*) é um comando executado automaticamente pelo sistema como um efeito colateral de uma modificação no banco de dados (KORTH e SILBERSCHATZ, 1995).

Pode-se citar como exemplo de asserção e gatilho a restrição “o salário de um empregado não deve exceder o do supervisor empregado” e “o número máximo de horas que

um empregado pode trabalhar por semana não pode exceder 56”.

Como banco de dados, as opções são muitas: existem os *open-source* como o *Interbase* e o *PostgreSQL*, *freewares*, como o *MySQL*, e comerciais, como *Oracle*, *Informix*, *DB2* e *Sybase*, sendo estes mais voltados ao mercado corporativo. A escolha de cada tipo de banco de dados vai depender da necessidade de segurança, desempenho, escalabilidade da aplicação e das limitações financeiras.

2.6.1 - PostgreSQL

O Sistema Gerenciador de Banco de Dados Orientado a Objeto PostgreSQL é derivado de um pacote Postgres escrito na Universidade da Califórnia em Berkeley. Com mais de uma década, desde o início de seu desenvolvimento, PostgreSQL é o mais avançado banco de dados *open-source*, oferecendo controle de concorrência multi-versão, suportando quase todos os construtores SQL (incluindo *subselects*, transações, e tipos definidos pelos usuários e funções), e tem uma vasta extensão com as diversas linguagens (POSTGRESQL, 2005).

A implementação do SGBD Postgres começou em 1986, sendo que a sua primeira versão de demonstração se tornou operacional em 1987. Postgres tem sido utilizado para implementar várias pesquisas e aplicativos de produção incluindo sistema de análise de dados financeiro, pacote de monitoramento de desempenho de turbinas de jato, uma base de dados de acompanhamento de asteróides, base de dados de informações médicas e diversos sistemas de informações geográficas. Postgres foi também utilizado como uma ferramenta educacional em diversas universidades.

Em 1994, Andrew Yu e Jolly Chen adicionaram um interpretador de linguagem SQL

(*Structured Query Language*) ao Postgres, criando o Postgre95. Em 1996 foi lançada a versão PostgreSQL, que reflete o relacionamento entre a versão original do Postgres e a mais nova versão com as capacidades do SQL.

O PostgreSQL foi criado para a plataforma Linux e a empresa DBExpert gerou uma versão do PostgreSQL para a plataforma Windows; porém, atualmente, a PostgreSQL desenvolveu a Versão 8.0 para a Plataforma Windows.

2.7 - Considerações Finais

Este capítulo apresentou os conceitos básicos necessários para o entendimento e valorização desta dissertação.

Como comentado neste capítulo, notou-se que o desenvolvimento de aplicações pode ser facilitado com a técnica de desenvolvimento baseado em regras de negócios, visto que esta, além de gerar quantidade menor de linhas de programas escritos pelos desenvolvedores e aumentar a reusabilidade, aplica a facilidade proposta em ferramentas baseadas nas “técnicas de quarta geração”.

Outro ponto relevante é a utilização do desenvolvimento Web, que auxilia na portabilidade das ferramentas, e a utilização de um banco de dados *open-source* que tem um interpretador da linguagem SQL.

Muitas regras de negócio podem ser expressas de forma simples na definição do banco de dados. Porém, algumas das regras que são constantemente modificadas nas empresas devem ser trabalhadas nas aplicações, pois facilitam as manutenções.

3 - BRD-TOOL – UMA FERRAMENTA GERADORA DE APLICAÇÕES BASEADA EM REGRAS DE NEGÓCIOS.

3.1 - Considerações Iniciais

Com a evolução dos Sistemas Gerenciadores de Banco de Dados (SGBD) e também com as “técnicas de quarta geração” de desenvolvimento de software, cada vez mais há a necessidade de agilizar e facilitar a criação de novas aplicações. Segundo Cagnin (2005), uma das preocupações da Engenharia de Software é migrar sistemas legados para novas tecnologias e paradigmas. Além disso, as regras de negócios e os requisitos específicos do sistema legado são implementados manualmente pelo engenheiro de software no sistema. Desta forma, este capítulo apresenta uma ferramenta para desenvolvimento de aplicações baseada na definição de regras de negócios.

O capítulo está organizado da seguinte forma: na Seção 3.2 são apresentados os conceitos de funcionamento e as tecnologias utilizadas na Ferramenta BRD-Tool, na Seção 3.3 está apresentada toda a arquitetura da Ferramenta, na Seção 3.4 está apresentado o modelo de banco de dados da BRD-Tool e algumas considerações importantes para entender o funcionamento da mesma.

3.2 - BRD-Tool

A BRD-Tool é uma ferramenta idealizada para a criação de aplicações baseadas em regras de negócios. Estas regras podem estar implícitas nas definições do banco de dados ou podem ser definidas pelos usuários no momento de sua criação. Para facilitar o processo de documentação das aplicações, no momento de sua criação, deve ser registrado o seu caso de uso, não como diagrama, mas sim como descrições narrativas de como a aplicação deve ser manuseada.

A Ferramenta BRD-Tool foi construída com base em aplicação *Web* usando *Java Server Pages* (JSP), sistema gerenciador de banco de dados PostgreSQL 8.0. A decisão de ser uma aplicação *Web* foi baseada na afirmação de Laudon e Laudon (2002) de que a Internet, Intranets e Extranets são as principais plataformas para comércio eletrônico, negócios eletrônicos e empresa digital, pois esta tecnologia proporciona vários benefícios, como por exemplo, a conectividade global, o fácil uso, o baixo custo e a capacidade multimídia que pode ser utilizada para criar aplicações, produtos e serviços interativos.

A escolha do Sistema Gerenciador de Banco de Dados Relacional (SGBDR) PostgreSQL 8.0 aconteceu por vários aspectos. O primeiro destes é a sua condição *open-source*, porém não se pode deixar de citar a possibilidade de trabalhar com as plataformas Windows e Linux, além da facilidade de trabalhar com um interpretador da linguagem SQL.

Por meio desta ferramenta, o desenvolvedor de sistemas, ao criar uma nova aplicação, deixará armazenados todos os dados utilizados em seu desenvolvimento, e o armazenamento destes dados facilitará uma manutenção futura da aplicação. O desenvolvedor também deve cadastrar regras de negócio, que poderão e deverão ser utilizadas no desenvolvimento das aplicações.

A BRD-Tool possui um repositório de regras de negócio que pode ser compartilhado

em projetos diferentes do mesmo desenvolvedor, e, caso a ferramentas seja utilizada por uma fábrica de software, esta poderá compartilhar este repositório com vários de seus clientes.

3.3 - Arquitetura

A Ferramenta BRD-Tool conta com dois atores: o Desenvolvedor e o Administrador, como pode ser observado na Figura 3.1, na qual também apresentam-se os casos de uso “Cadastrar Usuários”, “Cadastrar Projetos” e “Cadastrar Banco de Dados Cliente” que só podem sofrer algum tipo de ação pelo Administrador. Porém, os casos de uso “Cadastrar Regras de Negócio” e “Cadastrar Caso de Uso dos Projetos” podem ser realizados tanto pelo Administrador quanto pelo Desenvolvedor. Entende-se por “Cliente” a empresa para a qual está sendo desenvolvido um Sistema de Aplicação.

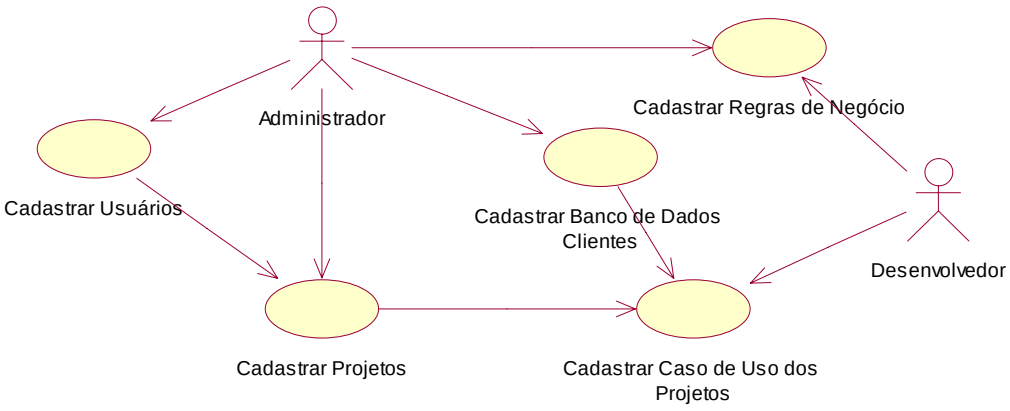

Figura 3.1 - Caso de Uso da Ferramenta BRD-Tool

Na Ferramenta BRD-Tool, serão cadastradas as regras de negócio de cada empresa e a definição de seu banco de dados. Somente após estas informações estarem cadastradas é que o desenvolvedor dará andamento no processo, definindo para quais tabelas serão geradas as

aplicações, quais os dados necessários para a sua interface, qual a necessidade de acrescentar regras de negócio em cada uma destas aplicações. Após todas estas definições, a ferramenta gerará a aplicação. Como foi dito anteriormente, todas estas definições ficarão armazenadas em um banco de dados da Ferramenta, e, em uma possível manutenção, basta acessar estes dados armazenados e gerar novamente a aplicação com as novas necessidades.

Na Figura 3.2 é ilustrada a arquitetura da ferramenta em questão, assim, pode-se visualizar com maior clareza como é feito todo o processamento para gerar uma aplicação com sua respectiva regra de negócio.

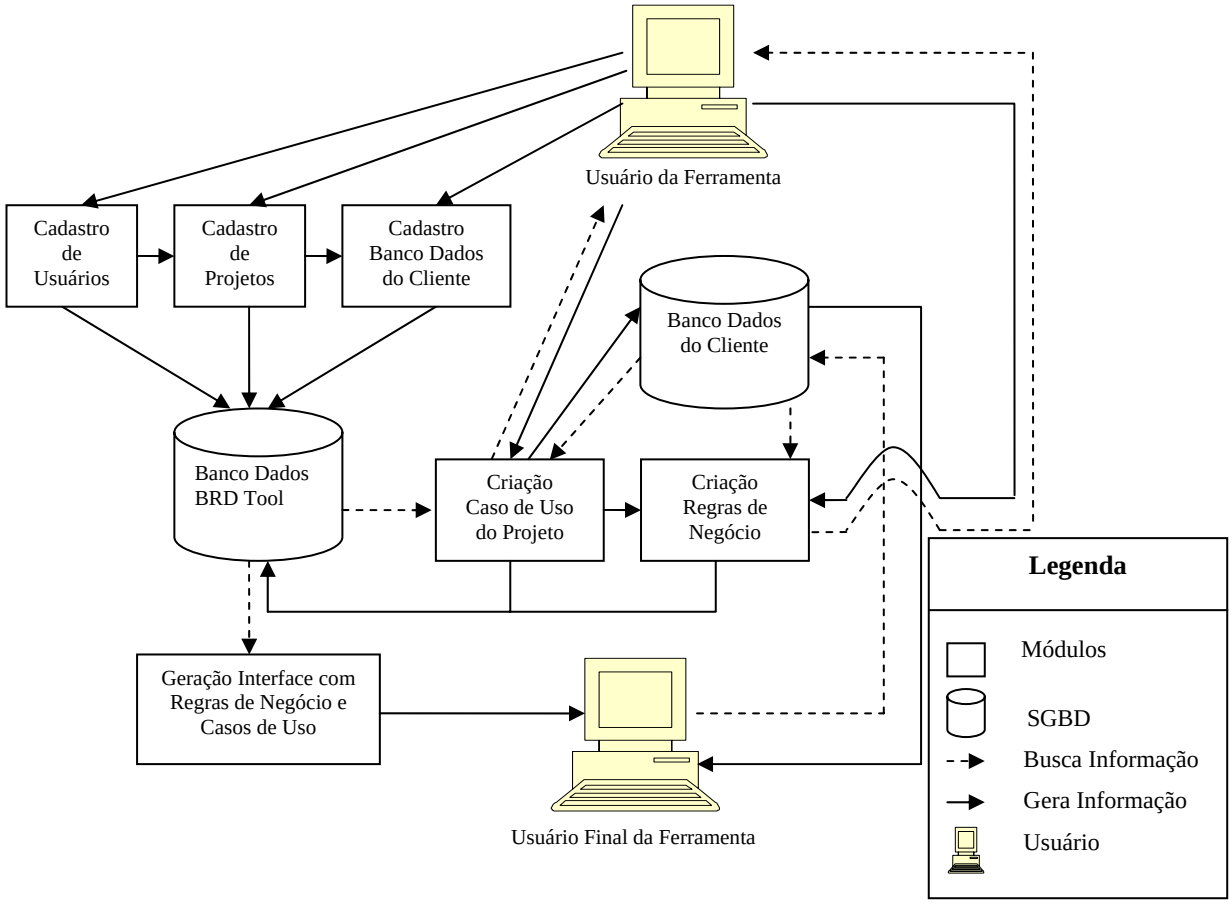

Figura 3.2 - Arquitetura da ferramenta BRD-Tool

No procedimento de instalação da ferramenta é necessário cadastrar o usuário Administrador e também um Projeto**.** A partir desses dois dados cadastrados é que se poderá dar início à criação dos “Caso de Uso do Projeto” das “Regra de Negócio”.

No cadastro são armazenados os dados dos usuários que são responsáveis por determinados projetos que serão determinados no cadastro, isto é, quando se cadastra um projeto, selecionam-se os usuários responsáveis por ele, além de armazenar seus dados. Por exemplo, pode-se criar um projeto chamado “Controle de Pedidos”, o qual pode ter dois usuários “A” e “B”, que são os responsáveis por seu desenvolvimento.

No cadastro do “Banco de Dados do Cliente” são armazenadas as configurações necessárias para se fazer a conexão com o banco de dados do cliente e a extração do metadados, pois é a partir dessas informações que será apresentada toda estrutura do banco de dados primário (ELMASRI e NAVATHE, 2005). São estas informações que facilitam a escolha do banco de dados, seus relacionamentos e as referências de integridade cadastradas.

Na criação dos “Casos de Uso do Projeto” e das “Regras de Negócio”, são utilizadas as configurações do “Banco de Dados do Cliente”. Após a escolha do Projeto pelo Desenvolvedor, são mostradas todas as tabelas existentes para que ele escolha uma para gerar a aplicação. Todos os seus catálogos são apresentados, além dos campos que possuem preenchimento obrigatório, como, por exemplo, as chaves primárias e os campos com restrição “*not null*”, já apresentados na tela, o que torna mais ágil a ferramenta, além de facilitar a visualização do Desenvolvedor sobre as Regras de Negócio que já foram definidas no banco de dados.

Após todas as informações necessárias estarem armazenadas no “Banco de Dados BRD-Tool”, torna-se possível a produção de um formulário ou de um relatório, criado com a linguagem *JSP*, para o usuário final onde todos os dados necessários ficam armazenados.

3.4 - Modelo do Banco de Dados

Como se pode observar nas Figuras 3.1 e 3.2, a Ferramenta BRD-Tool necessita de uma estrutura de tabelas relativamente simples como pode ser visto no Diagrama de Entidade e Relacionamento apresentado na Figura 3.3.

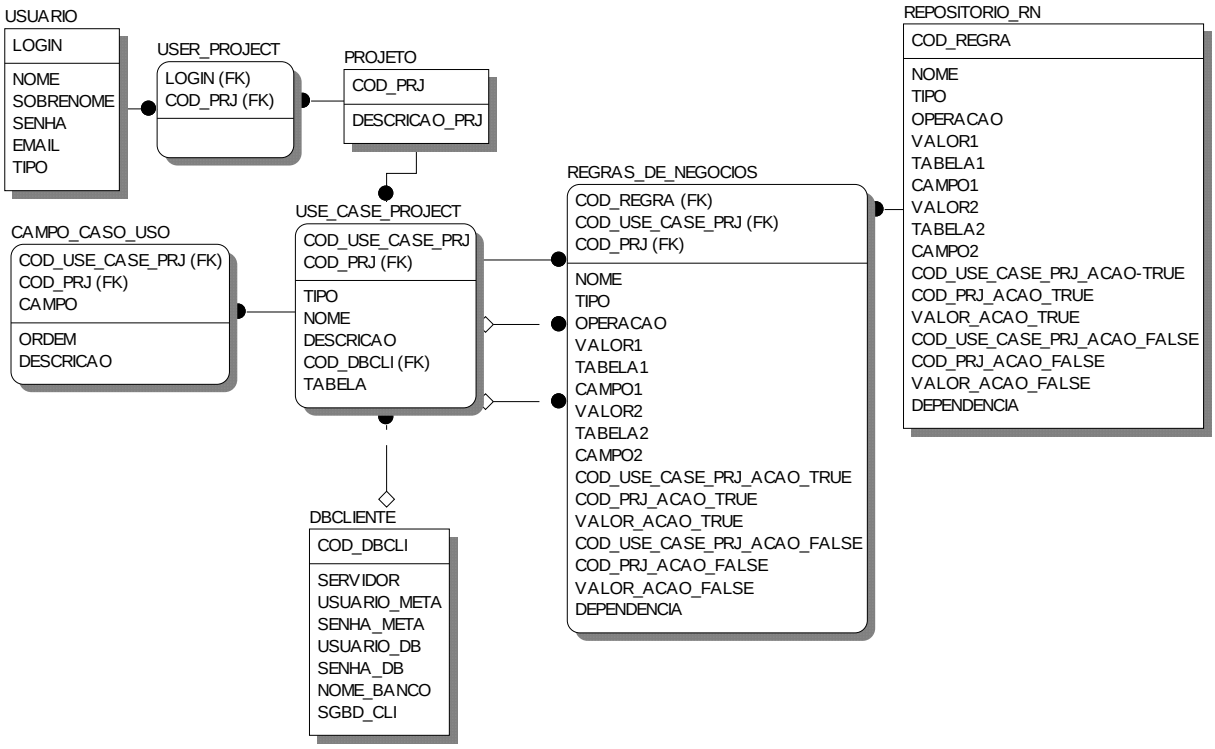

Figura 3.3 - Modelagem Entidade Relacionamento da Ferramenta BRD-Tools

Na Tabela “USUARIO” são armazenados os usuários que possuem acesso ao sistema, sendo que eles serão distribuídos em três categorias:

- **Administrador** – é o usuário que tem acesso à todas operações da Ferramenta. Somente este tipo de usuário pode incluir um novo usuário, um novo projeto e o relacionamento de quais usuários trabalham em cada um dos projetos criados, além também de criar novas regras de negócio e casos de uso do projeto. Este usuário pode também acessar as aplicações criadas pela

ferramenta e trabalhar como um usuário dos sistemas criados;

- **Desenvolvedor** – é o usuário que possui acesso somente à criação de novas regras de negócio e casos de uso do projeto, sendo possível visualizar apenas os projetos desenvolvidos por ele mesmo. Além de poder trabalhar como um usuário dos sistemas criados, também pode acessar as aplicações criadas pela ferramenta;
- **Comum** – este usuário só tem acesso às ferramentas geradas pelo usuários do tipo Administrador e Desenvolvedor, ou seja, é considerado como um usuário final dos sistemas criados.

Na Tabela “PROJETO” estão armazenados todos os projetos que estão sendo desenvolvidos ou mesmo os projetos que já foram finalizados e estão disponíveis para eventuais manutenções.

Um usuário pode e, normalmente, terá acesso a mais de um projeto, da mesma forma como um projeto normalmente terá mais de um usuário. Por este motivo, existe uma tabela de relacionamento de usuário com projetos, a “USER_PROJECT”.

Na Tabela “USE_CASE_PROJECT” estão armazenados todos os casos de uso específicos criados para cada um dos projetos. Por exemplo, pode-se ter um caso de uso “Cadastrar Cliente” que existirá em vários projetos diferentes e para um destes projetos as informações armazenadas são distintas. Nesta tabela as informações de todos os casos de uso são armazenas separadamente por projeto. Os dados armazenados nesta tabela são utilizados para realizar eventuais futuras manutenções nas aplicações que foram geradas. O campo DESCRICAO desta tabela é utilizado para documentação do caso de uso, e estas informações ficam disponíveis para uma consulta on-line da aplicação.

Na Tabela “REPOSITORIO_RN” são cadastradas todas as regras de negócio chamadas de genéricas. Adotou-se a nomenclatura genérica pelo fato das regras cadastradas

nesta tabela não serem específicas para um único projeto, ou seja, esta regra estará disponível para ser utilizada em qualquer projeto criado pela Ferramenta BRD-Tool. Os dados contidos nesta tabela ficam disponíveis para todos os usuários do tipo Administrador ou Desenvolvedor para o auxilio na criação de uma nova regra de negócio para um determinado projeto.

A Tabela “REPOSITORIO_RN” é, na realidade, uma tabela de parâmetros para as regras de negócio de um determinado projeto. Somente os campos COD_REGRA, NOME, TIPO, OPERACAO e DEPENDENCIA, têm seu preenchimento normal, pois, inclusive estes serão utilizados na criação de uma regra de negócio para um projeto. Os outros campos são do tipo *boolean*, e o seu preenchimento indicará se um determinado parâmetro é utilizado ou não na criação de uma nova regra de negócio. Se na Tabela “REPOSITORIO_RN” o campo for como um parâmetro obrigatório, quando esta regra de negócio for utilizada em um projeto específico, o mesmo campo da tabela “REGRAS_DE_NEGOCIOS” deverá ser obrigatóriamente preenchido.

A Tabela “REPOSITORIO_RN” é comum para todos os projetos e é esta característica que permite a reusabilidade da mesma em vários projetos diferentes, além de facilitar a criação de regras de negócio para cada um dos projetos e minimizar a possibilidade de erros nas criações de novas regras de negócio.

Como exemplo, pode existir no “REPOSITORIO_RN” uma regra chamada “Repor Estoque Minimo baseado em Tabela de Parâmetro”. Esta regra será do TIPO “Integridade” e sua OPERACAO será de comparação “<=” e não tem DEPENDENCIA de nenhuma outra regra. Neste exemplo, os campos TABELA1, CAMPO1, TABELA2, CAMPO2 e VALOR_ACAO_TRUE deverão ser preenchidos como obrigatórios. Desta forma, quando o Desenvolvedor implementar esta regra de negócio em um projeto, ele será obrigado a preencher todos estes campos com os valores correspondentes aos dados do projeto, pois de

um projeto para outro os nomes das tabelas e dos campos podem ser diferentes.

A Tabela “CAMPO_CASO_USO**”** é usada para armazenar as seqüências apresentadas pelos campos em um formulário ou relatório, e, também, para indicar qual é sua descrição na aplicação, pois, algumas vezes, o nome do campo no banco de dados não é tão claro para o entendimento do usuário final. Por exemplo, pode-se ter um campo em uma tabela de controle de estoque chamado “EST_MIN”, entretanto, aparecerá para o usuário com a descrição de “Estoque Mínimo”.

Na Tabela “BDCLIENTE” são armazenados os dados referentes aos bancos de dados dos clientes, podendo, inclusive, um cliente possuir mais de um SGBD (Sistema Gerenciado de Banco de Dados).

Na Tabela “REGRAS_DE_NEGOCIOS” estão armazenadas as regras de negócio existentes para cada um dos projetos específicos.

4 - ESTUDO DE CASO

4.1 - Considerações Iniciais

Neste capítulo são apresentados alguns estudos de casos realizados para mostrar o funcionamento da Ferramenta BRD-Tool. Definidos os resultados esperados, esta ferramenta divide-se em duas partes:

A) O cadastramento dos dados referentes aos usuários, projetos e bancos de dados dos clientes (esta etapa foi implementada na ferramenta e é realizada somnte pelo usuário do tipo Administrador),

B) O cadastramento dos “Casos de Uso dos Projetos” e das regras de negócio (esta etapa foi parcialmente implementada).

A Ferramenta BRD-Tool possui duas interfaces diferentes: uma para os usuários do tipo Administrador e Desenvolvedor, e outra para os usuários do tipo Comum, ou seja, os usuários finais dos Sistemas de Aplicação.

O capítulo está organizado da seguinte forma: na Seção 4.2 são apresentados os pré-requisitos e os procedimentos para a instalação da Ferramenta BRD-Tool, na Seção 4.3 está apresentada toda a manutenção dos cadastros básicos da Ferramenta, sendo estes cadastros necessários para o início do uso da Ferramenta. Na Seção 4.4 apresenta-se o funcionamento da Ferramenta para o cadastramento de novos Casos de Uso de um projeto e na Seção 4.5 o cadastramento de novas Regras de Negócios.

4.2 - Instalação da Ferramenta BRD-Tool

O pré-requisito para a utilização da Ferramenta BRD-Tool é a instalação do SGBD PostgreSQL 8.0 e do Java. A instalação do PostgreSQL só é necessária na máquina que servirá como *Web Server*.

Com estas ferramentas instaladas, é preciso instalar a Ferramenta em questão, que se incumbirá de criar todas as tabelas necessárias de acordo com o Apêndice A.

No término da instalação da ferramenta, é indispensável a criação de um usuário, que deverá ser obrigatoriamente do tipo Administrador, além da criação de um projeto.

Após a instalação da ferramenta, pode-se, então, iniciar o seu uso. Por medida de segurança e, também, para a identificação de quais acessos podem ser liberados, a Ferramenta possui uma tela de validação do usuário (Figura 4.1). Tratando-se de uma ferramenta Web, que pode ser acessada de qualquer computador que tenha acesso a Internet, esta etapa é indispensável, pois garante que cada usuário tenha acesso somente aos projetos nos quais esteja relacionado.

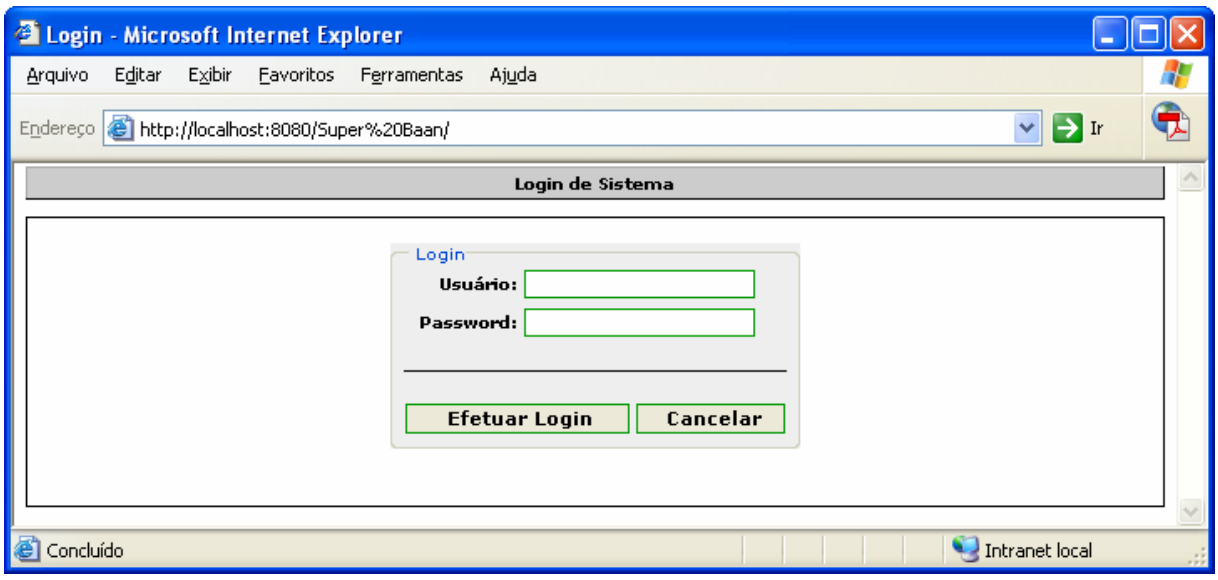

Figura 4.1 - Tela de Login do Usuário

A validação do usuário é necessária também para identificar qual é a interface da Ferramenta e quais projetos este usuário pode ter acesso. A diferença entre as interfaces de um Administrador e Desenvolvedor para um usuário Comum (usuário final) pode ser notada visualmente na disponibilidade dos menus localizados ao lado esquerdo da tela, como apresentados nas Figuras 4.2 e 4.3.

Na Figura 4.2 nota-se a existência de um menu chamado Sistema, o qual é dividido em sub-menus: o de Manutenção e o de Projetos. O primeiro só é liberado para os usuários do tipo Administrador e Desenvolvedor, é nesta opção que são realizados os cadastramentos e as manutenções dos usuários, projetos e as configurações dos bancos de dados dos clientes.

Abaixo do menu Manutenção, encontra-se o menu Projetos, o qual é liberado somente para os usuários do tipo Administrador, Desenvolvedor e Comum. Porém, para os usuários do tipo Comum, as opções são diferenciadas, pois este somente poderá executar as aplicações criadas, já os usuários do tipo Administrador e Desenvolvedor poderão criá-las. Como observado anteriormente, no Capítulo 3, um usuário pode ter acesso a mais de um projeto, e, por este fato, há a separação de cada um destes projetos com os seus respectivos nomes (na Figura 4.3 está representado pelos nomes Calculo de Frete e Controle de Pedidos), e, em cada um deles há os casos de usos que foram criados.

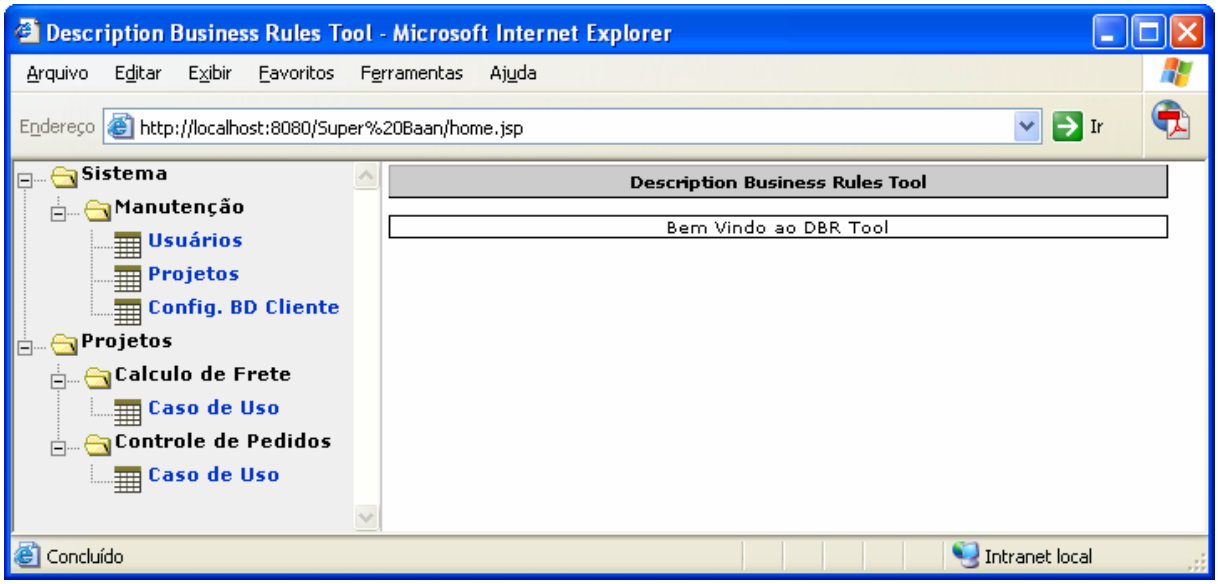

Figura 4.2 - Tela de Manutenção do Sistema

Observe a Figura 4.3: na interface do usuário do tipo comum somente aparecem os projetos que são liberados para ele e dentro de cada um deles, somente estarão disponibilizadas as aplicações criadas. Não há, desta forma, a possibilidade de um usuário comum criar ou modificar uma aplicação já existente.

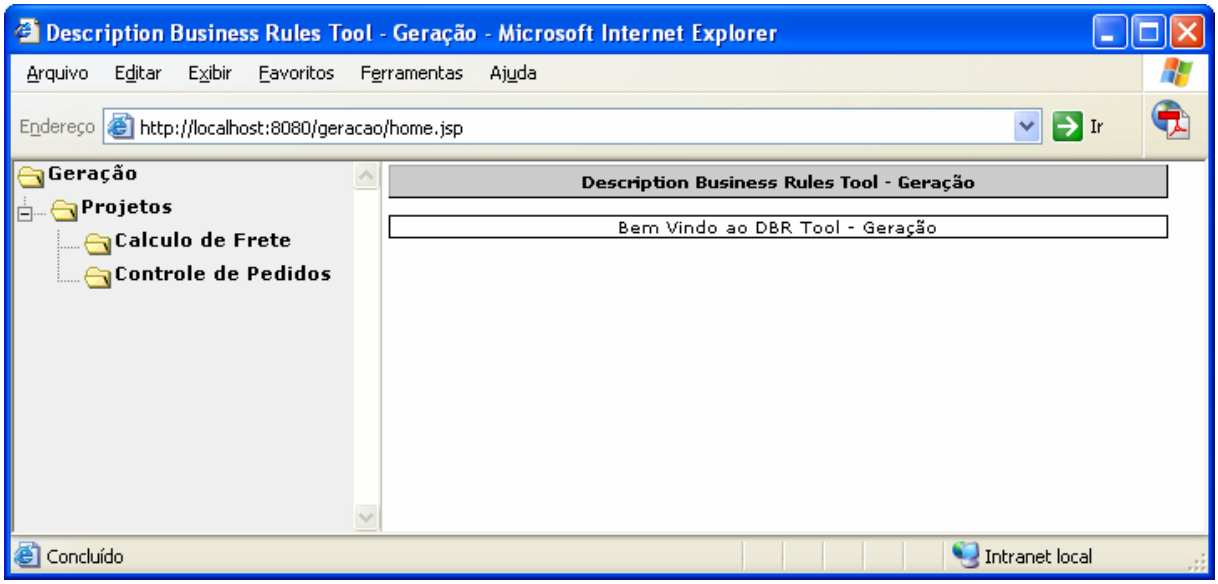

Figura 4.3 - Tela de Usuário Final

4.3 - Cadastros Básicos da Ferramenta

As manutenções necessárias para a liberação dos Sistemas de Informação criados devem ser realizadas por pessoas da área técnica das empresas, ou, ainda, por pessoas terceirizadas que conheçam a infra-estrutura das empresas, pois é necessário o conhecimento, principalmente, das informações sobre o Banco de Dados dos clientes e dos tipos de usuários e projetos disponíveis.

4.3.1 - Cadastramento de Usuários

Quando um usuário habilitado acessar a aplicação de cadastramento de usuários, a interface apresentada será a mostrada na Figura 4.4, na qual estarão aparentes todos aqueles cadastrados no sistema. Nesta tela existe um botão para o cadastramento de novos usuários (“Criar Usuário”) e também para as manutenções daqueles já cadastrados. Neste segundo caso, o procedimento será realizado utilizando o botão “Ver”, o qual exibirá todas as informações do usuário selecionado para, então, realizar a manutenção desejada.

A tela para manutenção de um usuário ou para o cadastramento de um novo é a mesma, somente são alterados os dados apresentados, ou seja, ao realizar a inclusão de um novo usuário não será apresentado dado algum, já no caso de uma manutenção, todos os dados do usuário serão apresentados.

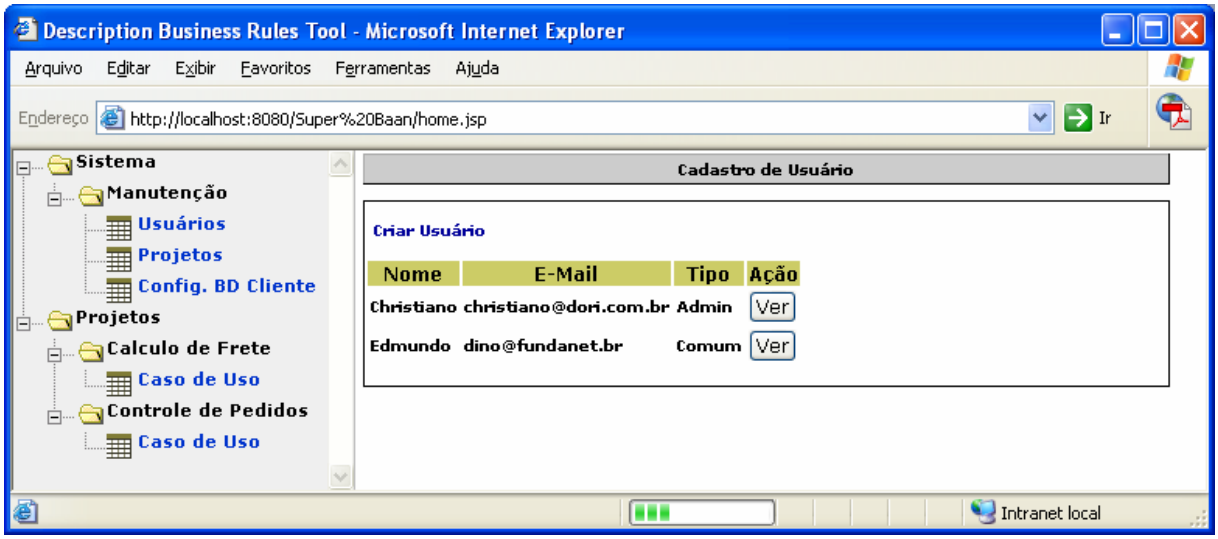

Figura 4.4 - Tela inicial do Cadastro de Usuários

Na Figura 4.5 são apresentados todos os dados necessários para a realização do cadastramento ou manutenção de um usuário.

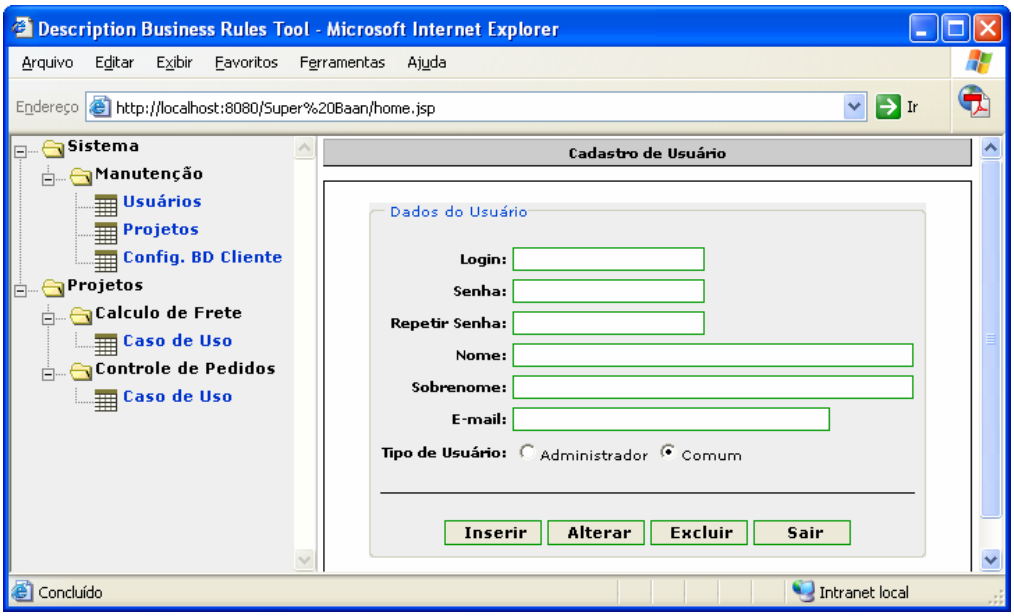

Figura 4.5 - Tela de Manutenção de Usuários

Como na maioria dos Sistemas de Informação, é necessária a criação de um login de usuário e sua senha, além disso, são registrados os seus dados básicos: nome, sobrenome, e-

mail e, por fim, o tipo deste usuário.

Sempre que se for fazer um novo cadastro, a opção que aparecerá será de usuário comum devido ao fato da maioria dos usuários ser deste tipo, pois, sempre há mais usuários finais dos Sistemas de Informação do que desenvolvedores.

4.3.2 - Cadastramento de Projetos

O nome Projeto para esta Ferramenta significa cada um dos módulos[5] que serão criados nos sistemas de informação, ou seja, dentro de uma empresa pode-se ter os módulos de controle de pedidos, cálculo de frete, contas a pagar, contas a receber, etc. Para cada um destes módulos, que serão tratados de forma diferente, deverá ser criado um projeto. O fato de que mais de um projeto criado utilize o mesmo banco de dados dos clientes já é previsto pela Ferramenta, não havendo, assim, nenhum inconveniente ao se trabalhar desta forma, pois apenas serão diferenciadas as tabelas para a criação dos casos de uso.

Como mostrado na Figura 4.6, assim como na tela de manutenção de usuários, quando um usuário do tipo Administrador ou Desenvolvedor entra nesta interface, todos os projetos existentes cadastrados no sistema são apresentados, independentemente dos projetos estarem ou não relacionados para o usuário.

Tal fato ocorre devido a possibilidade do usuário, a qualquer momento, modificar suas atividades e ter de iniciar um trabalho em um novo projeto. Mas, se ele não estiver relacionado com determinado projeto, este não aparecerá ao lado esquerdo da tela (local onde estão todos os projetos relacionados ao usuário). Dessa forma, é impossível que o usuário se engane com os projetos que está trabalhando.

[5] Pequeno trecho de um programa grande que pode, se necessário, funcionar independentemente como um outro programa. (Fonte Dic Michaelis)

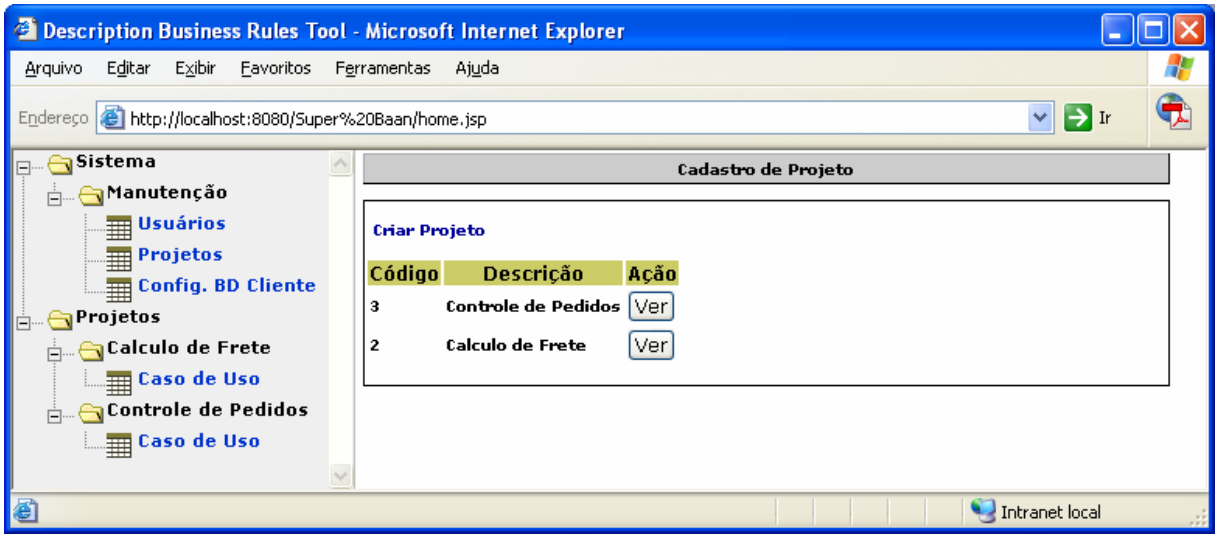

Figura 4.6 - Tela Inicial do Cadastro de Projetos

Observe que a Figura 4.6 também possui os botões “Criar Projeto” e “Ver”, que são as ações idênticas do cadastramento de usuários, modificando apenas a tabela acessada e os dados necessários para preenchimento.

Na Figura 4.7 são apresentados os dados necessários para o cadastramento de um projeto para que o mesmo não fique sem um relacionamento com, ao menos, um usuário. Nesta mesma manutenção é possível fazer o relacionamento do projeto com os usuários que tem acesso.

Nota-se, ainda, que sempre que se entra nesta manutenção, caso seja um novo projeto a ser cadastrado, serão apresentados todos os dados sem preenchimento, com exceção do quadro de usuários, pois neste sempre aparecerão todos aqueles já cadastrados no sistema.

Para que um usuário seja relacionado a um projeto basta clicar sobre o login do usuário e então pressionar o botão “>>”, e no caso de retirar o relacionamento basta clicar no quadro do lado direito, onde estão todos os usuários relacionados e, em seguida, clicar sobre o botão “<<”.

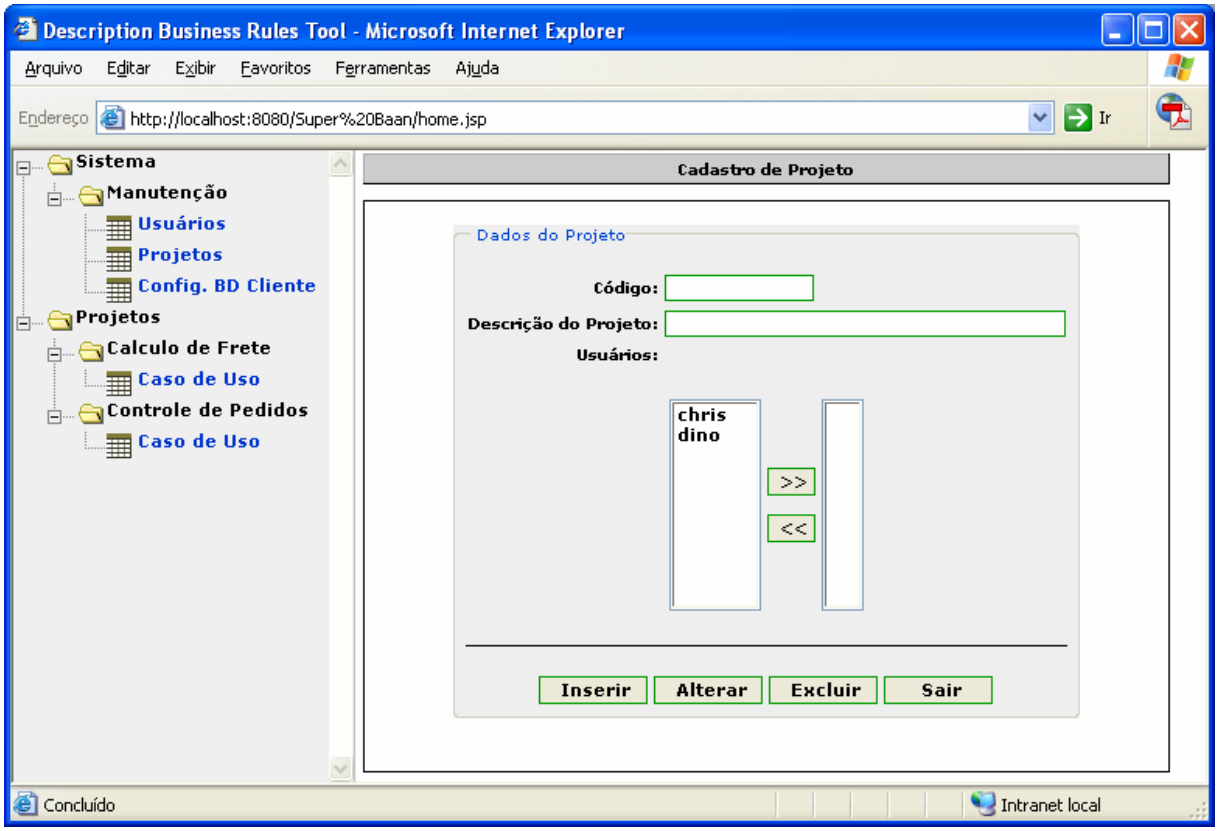

Figura 4.7 - Tela de Manutenção de Projetos

4.3.3 - Cadastramento dos Bancos de Dados dos Clientes

Como já mencionado anteriormente, para vários projetos de um mesmo cliente pode ser utilizada a mesma configuração de um banco de dados, porém, ainda é possível que um único cliente tenha mais de um banco de dados, sendo esta uma condição suportada pela Ferramenta.

Na Figura 4.8 é mostrada a tela inicial do cadastramento de bancos de dados dos clientes. Nesta tela sempre aparecerão todos os bancos de dados de clientes cadastrados que poderão ser de mais de um cliente. Por isso, é importante definir vários dados para poder identificar quem realmente é o cliente.

Os usuários que criam os casos de uso e regras de negócios precisam sempre estar

muito esclarecidos sobre quais são os bancos de dados a serem utilizados, pois alguns podem dizer que uma das falhas no desenvolvimento da ferramenta é não ter um relacionamento de projeto com banco de dados. Mas isto não foi feito para que a ferramenta tivesse uma flexibilidade de em apenas um projeto, ter acesso a mais de um banco de dados, o que se julga ser mais útil.

Não se pode esquecer de que o cadastramento de um banco de dados de cliente é o responsável por identificar qual é o Sistema Gerenciado de Banco de Dados do cliente, e para esta versão da ferramenta, somente os SGBD PostgreSQL e Oracle estão implementados.

Na Figura 4.9 observam-se as necessidades de identificar qual é o servidor e como esta ferramenta foi desenvolvida com a tecnologia *Web* que poderá ser instalada em um servidor fora da empresa e continuar fazendo acesso ao mesmo, o que facilita uma manutenção terceirizada.

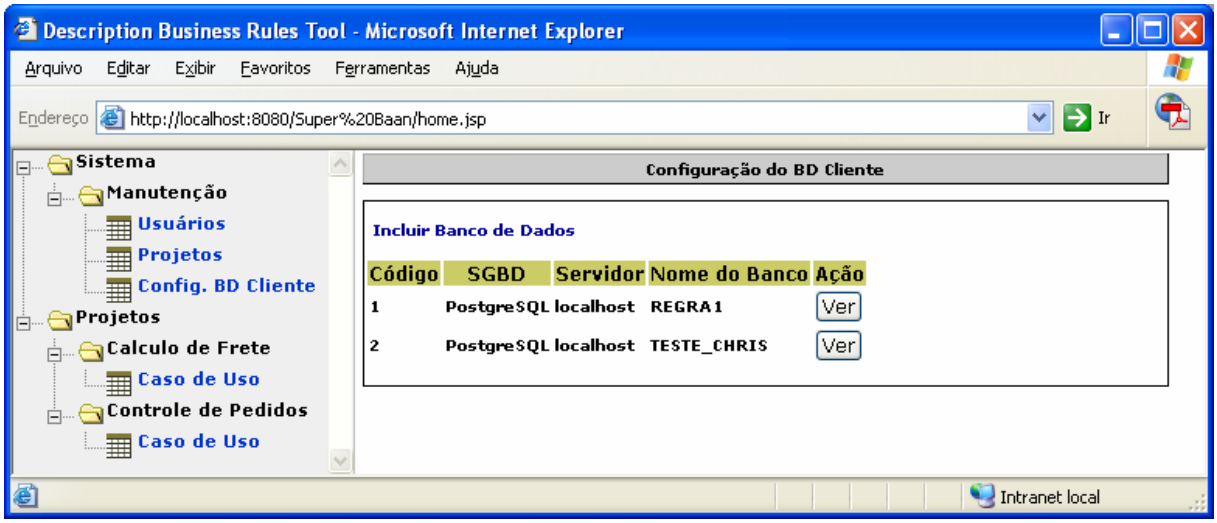

Figura 4.8 - Tela inicial do Cadastro de BD dos Clientes

As informações relacionadas a metadados[6] e do usuário do banco de dados são de grande importância para esta criação, pois se um destes dados não for informado

[6] Metadados são normalmente conhecidos como “informação que descreve informação” (SILVA, 2000)

corretamente, os usuários ficarão impossibilitados de criar os casos de usos e regras de negócios, uma vez que o sistema não conseguirá identificar as tabelas para que sejam cadastradas as manutenções.

É sempre muito importante que o administrador do banco de dados (DBA) mantenha informadas as pessoas responsáveis por estes dados, pois uma mudança de senha, por exemplo, pode comprometer todo o desempenho da Ferramenta.

Na Figura 4.9 são apresentados os campos de configuração do SGBD adotados no projeto em desenvolvimento, informando o nome do servidor, o login e senha dos usuários metadado e do banco de dados e o nome do Banco de Dados.

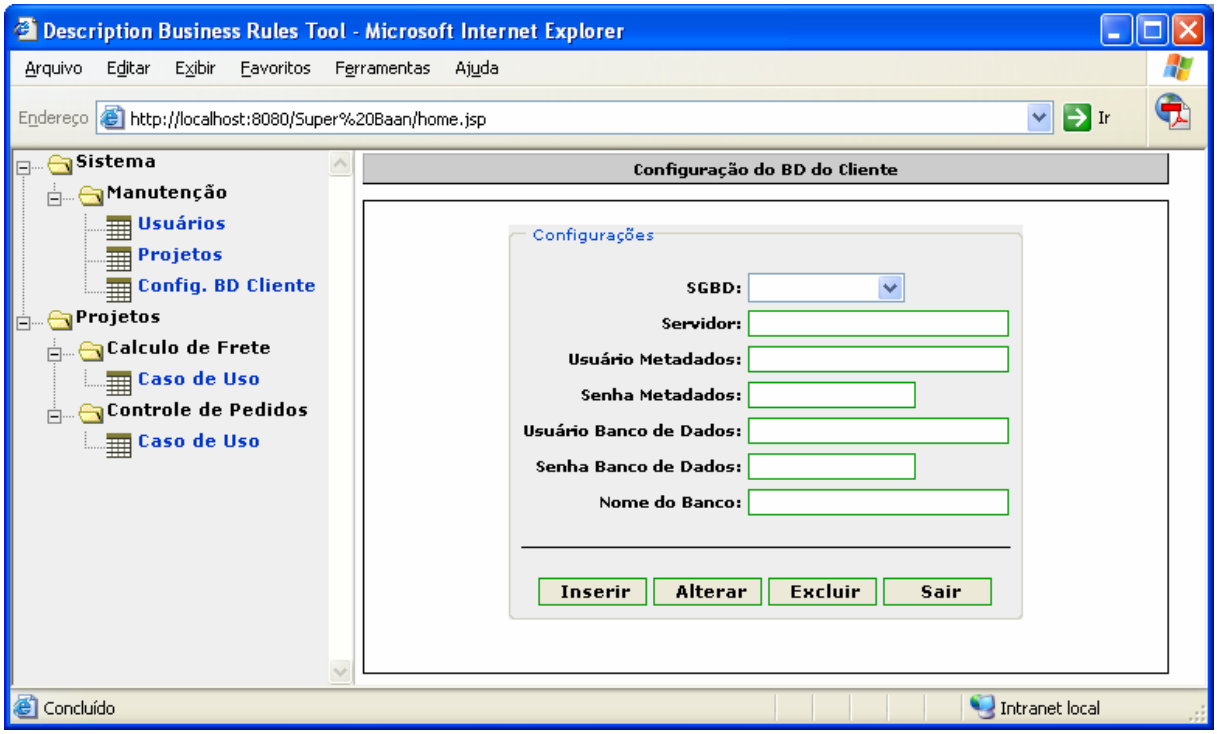

Figura 4.9 - Cadastramento de um Banco de Dados do Cliente

4.4 - Cadastramento Novos Casos de Uso de um Projeto

É apresentado agora o funcionamento da ferramenta para a geração e também os novos casos de usos e regras de negócios de um sistema de informação. Nas etapas apresentadas no tópico “Cadastramento dos Dados Básicos da Ferramenta” foram cadastrados todos os dados necessários para a execução deste teste.

Ressalta-se aqui que parte desta etapa da ferramenta está desenvolvida, mais especificamente a definição dos casos de uso, a seleção do banco de dados, a tabela e também os campos com suas ordenações. Porém, o cadastramento das regras de negócios que é apresentado faz parte do protótipo criado. Será, então, apresentado uma simulação de como as etapas todas deverão ser cadastradas.

4.4.1 - Projeto de Controle de Pedidos

Para uma exemplificação da Ferramenta, será utilizado um projeto de Controle de Pedidos. Para tanto, foram utilizadas apenas quatro tabelas: cliente, item, pedido e item_pedido, como apresentado na Figura 4.10.

A Ferramenta BRD-Tool será utilizada para criar as aplicações de manutenção destes registros, e pode-se notar que neste exemplo simples, existem alguns relacionamentos de tabelas que deverão ser obedecidos pela ferramenta.

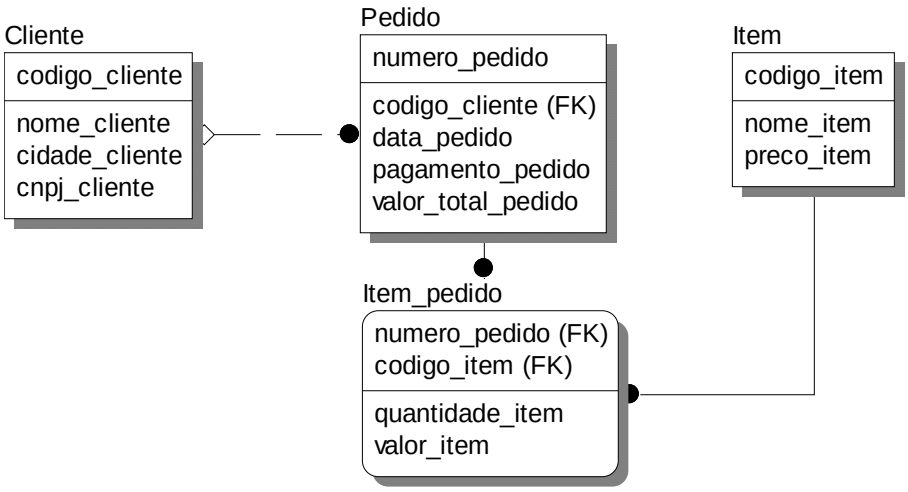

Figura 4.10 - Estrutura do Estudo de Caso Controle de Pedidos

Neste exemplo, são aplicadas as seguintes regras de negócios:

1. A data do pedido não poderá ser menor do que a data atual;
2. O valor do pedido não poderá ser menor que 10;
3. Não é possível ter pedido sem ter itens dos pedidos.

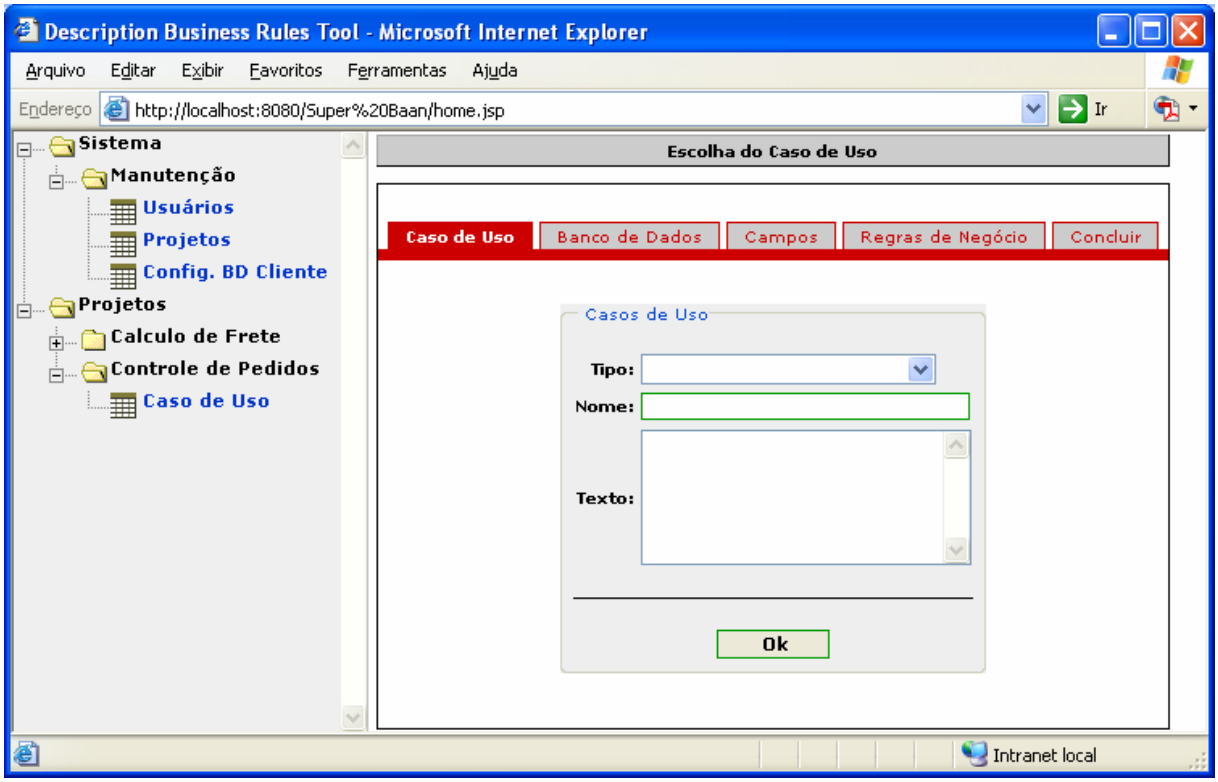

Figura 4.11 - Etapas da criação de uma aplicação

Quando se entra na criação de novos casos de uso, uma tela como a apresentada na Figura 4.11 é mostrada, na qual pode-se perceber todas as etapas necessárias para gerar uma aplicação.

A seqüência a ser seguida para a criação de uma aplicação é muito importante, pois destas etapas é que será gerada a documentação do sistema.

Na tela identificada como caso de uso é preciso identificar, primeiramente, o tipo de caso de uso, que podem ser dois: Manutenção de Registro ou Relatório. Esta opção é que identificará a interface necessária para a aplicação. No caso, será criado primeiramente um caso de manutenção de registro.

Após a escolha do tipo de caso de uso, escolhe-se, então, o nome, ou seja, aquele que será apresentado para o usuário comum (final) para que o mesmo possa saber o que aquela aplicação faz.

O campo texto é que será aproveitado para a documentação. É justamente por causa deste campo que esta tela é denominada de caso de uso. Neste campo deverá ser descrito exatamente o que esta aplicação deve fazer, quem são seus atores e no caso da sessão, deve ser descrito obrigatoriamente o cenário principal, e caso exista, também o cenário secundário.

Na Figura 4.12, tem-se exatamente o exemplo da geração da aplicação de cadastro de cliente. No campo de texto, o caso de uso que foi utilizado é:

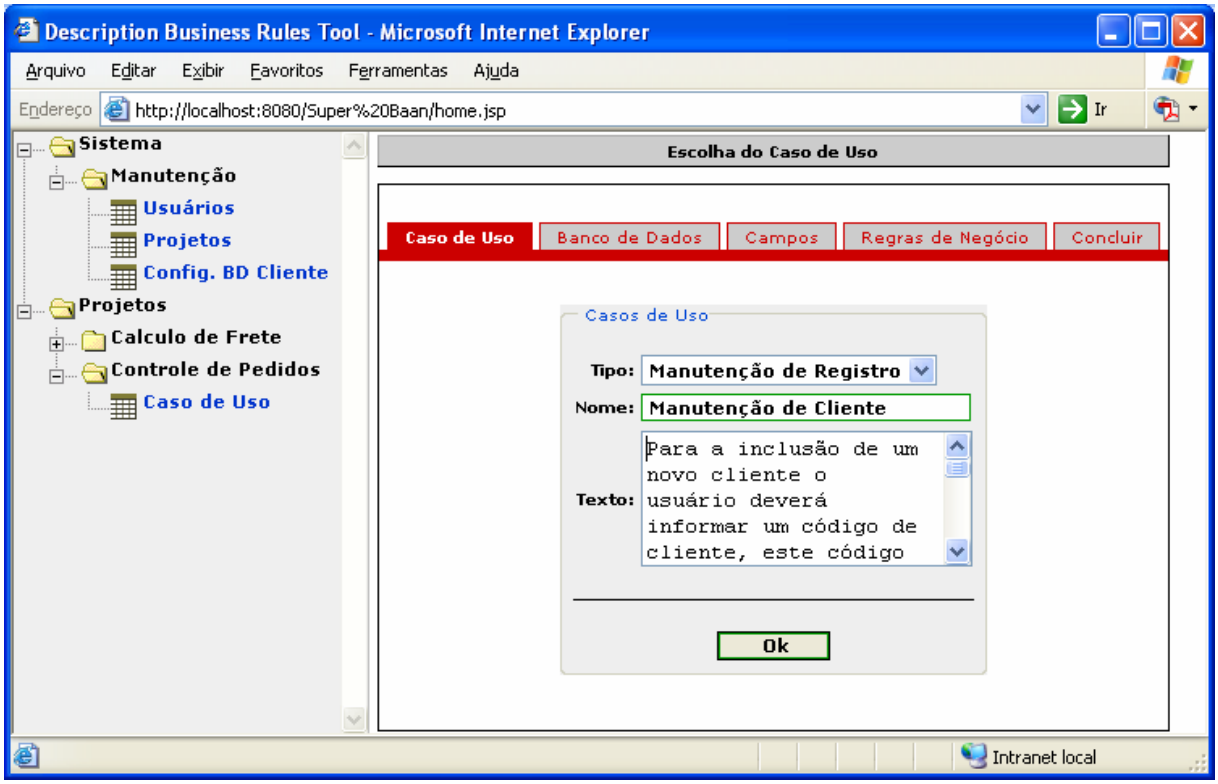

Figura 4.12 - Inclusão da Manutenção de Cliente

"Para a inclusão de um novo cliente o usuário deverá informar um código de cliente, este código deverá ser verificado para saber se não existe. No caso da não existência, será, então, informada o seu Nome, sua Cidade e seu CNPJ". Se o código do cliente já existir, serão apresentados na tela os dados do cliente e somente as opções de exclusão ou alteração serão disponibilizadas. No caso de uma alteração, somente os campos de Nome, Cidade e CNPJ poderão ter alterações. No caso de uma exclusão, a mesma só poderá ser executada caso o cliente não tenha nenhum pedido registrado."

Clicando no botão OK da tela, a ferramenta passa automaticamente para a segunda etapa, que é a informação do Banco de Dados, como pode ser visto na Figura 4.13.

O que se percebe, então, é que não significa apenas a escolha do Banco de Dados, mas também a da tabela.

Quanto o usuário escolher o Banco de Dados, a Ferramenta DBR-Tool se incumbirá

de buscar todas as tabelas existentes neste banco, e, então, o desenvolvedor deverá escolher com qual tabela trabalhará. Nota-se, nesta etapa, a grande importância de se definir claramente o nome do caso de uso, pois o nome da tabela não ficará exposto em nenhum lugar futuramente e, mesmo que ficasse, muitas vezes, os nomes das tabelas não tem uma expressão clara do que ela representa.

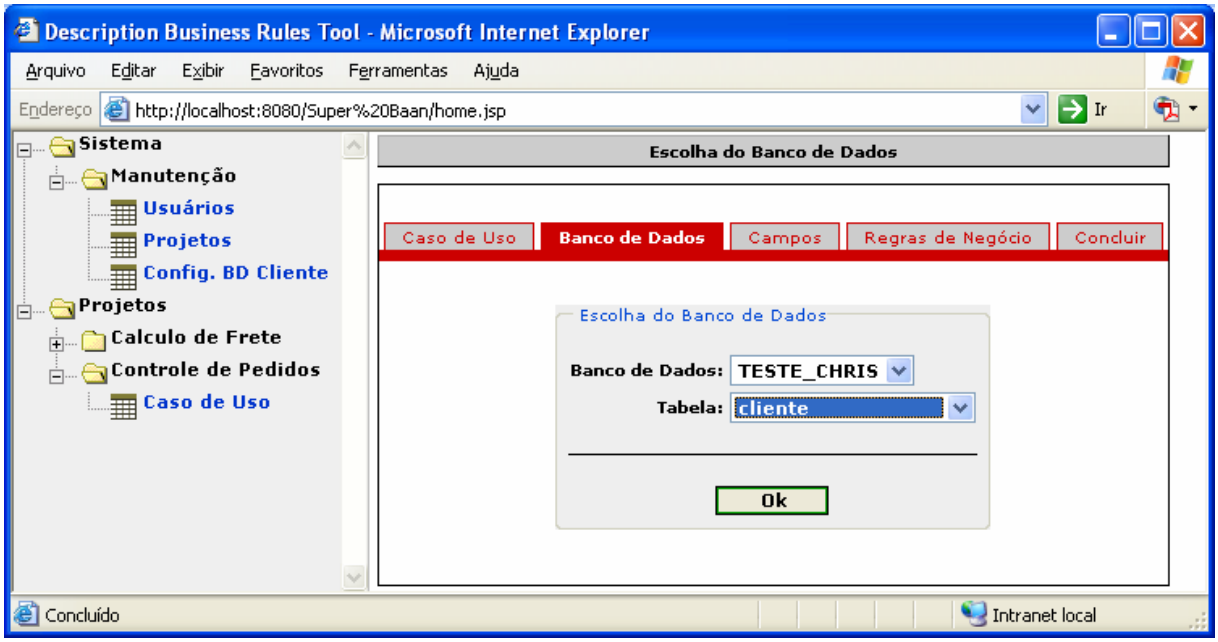

Figura 4.13 - Escolha do Banco de Dados

Novamente, clicando no botão “Ok,” a ferramenta passará para a próxima etapa que é a escolha dos campos que farão parte da aplicação, como apresentado na Figura 4.14. Todos os campos da Tabela estarão disponíveis para serem selecionados. Porém, os campos que forem identificados como chave primária ou chave estrangeira farão, obrigatoriamente, parte da aplicação.

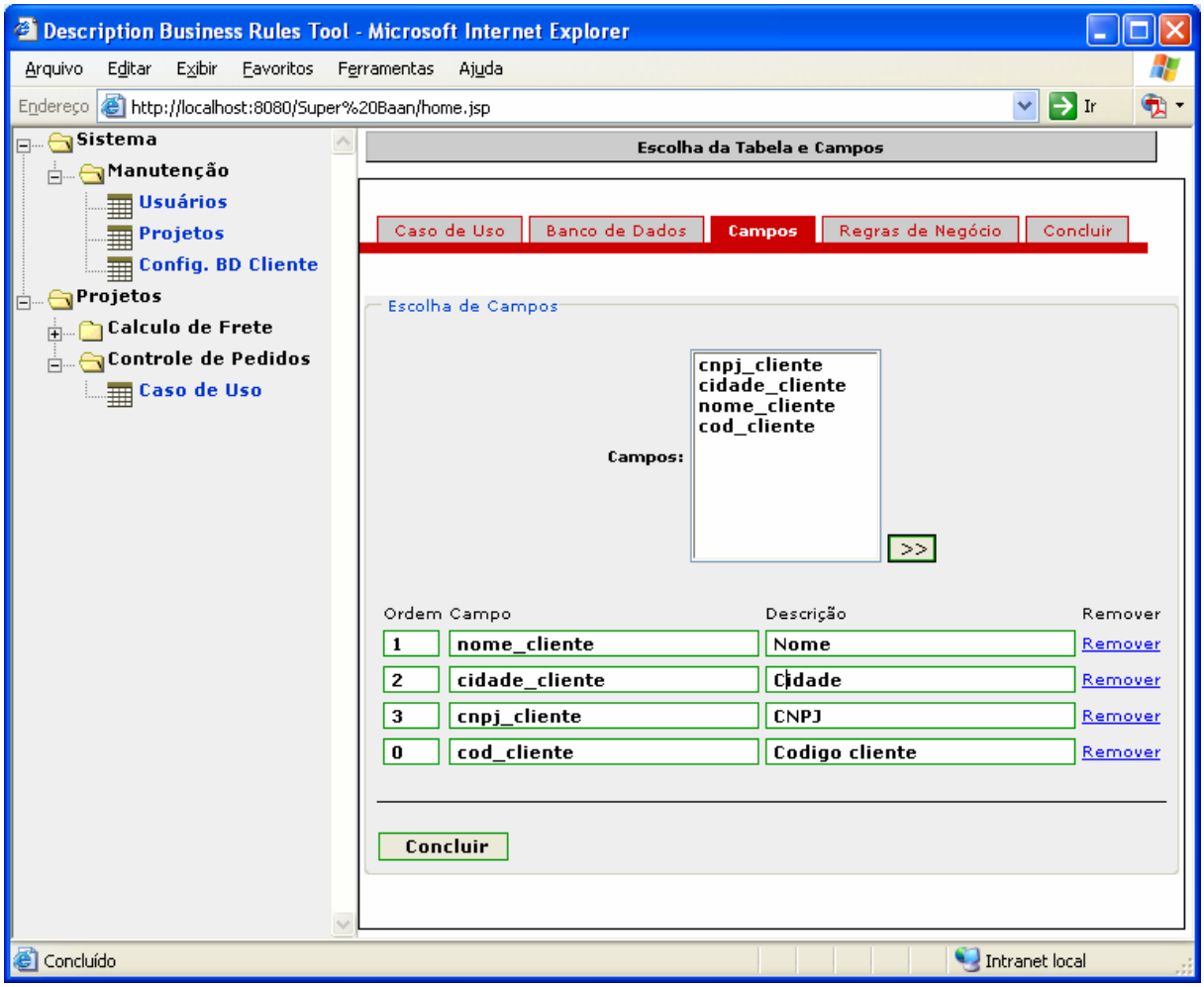

Figura 4.14 - Escolha dos campos e ordenação

Nesta etapa do processo, quando é feita a escolha dos campos, estes também poderão receber uma nova descrição, pois, muitas vezes os novos dados aos atributos da tabela não são os mais apropriados para serem apresentados aos usuários e, muitas vezes, não são claros para o entendimento. Além da escolha da descrição do campo, pode-se também escolher a ordenação do mesmo, pois muitas vezes um atributo da tabela pode não estar em uma seqüência lógica devido a uma série de motivos que não é o foco deste trabalho, e esta ordenação é que permitirá ao usuário colocar estes atributos na melhor seqüência de preenchimento para os usuários finais.

Após o preenchimento desta etapa, clicando no botão “Concluir”, a ferramenta irá para a etapa de cadastramento das regras de negócios. É uma das etapas mais importante desta

ferramenta.

Como é apresentado na Figura 4.15, poderá ser adicionada a um caso de uso uma regra de negócios, podendo ser específica ou genérica. Porém, sempre será nesta tela que as regras de negócios serão cadastradas. Outros comentários sobre o cadastramento das regras de negócios são descritos mais adiante, pois esta atividade não é obrigatória para todas as aplicações que serão geradas, como no caso do exemplo de um cadastramento de cliente, que não possui nenhuma regra específica.

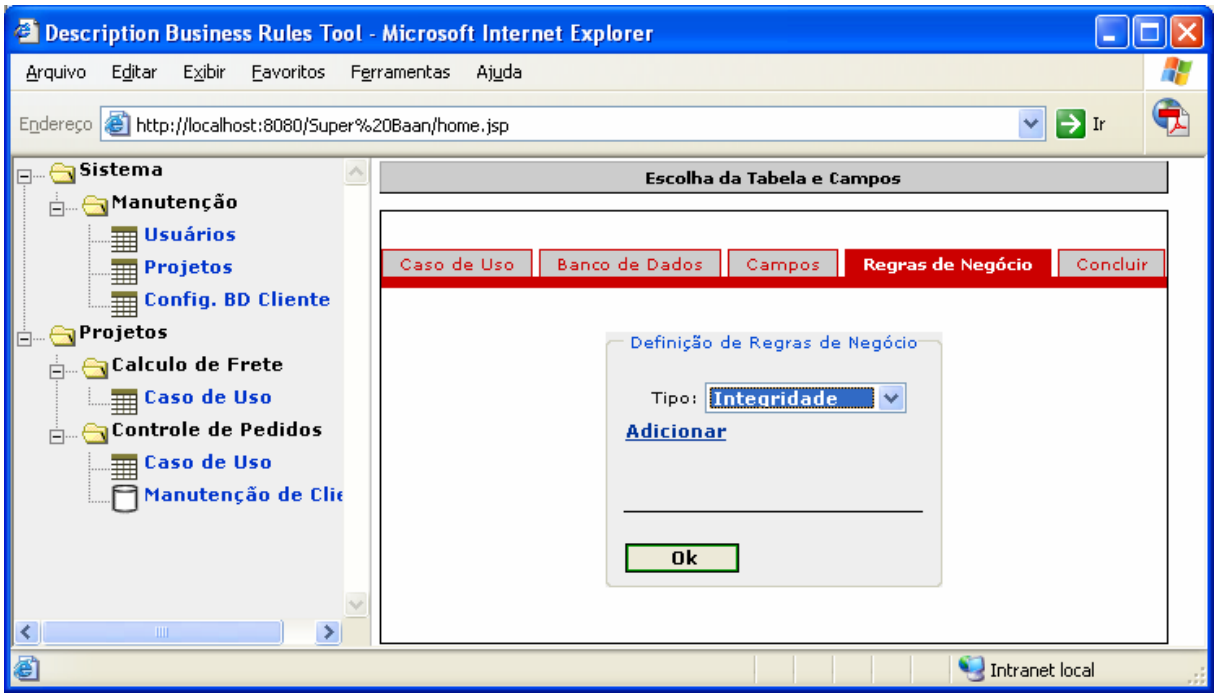

Figura 4.15 - Inclusão de Regras de Negócios.

Quando uma regra de negócios for incluída em alguma aplicação, ela pode ser escolhida de um repositório de regras ou também ser cadastrada. Para o cadastramento de uma nova regra, deve-se especificar se é uma regra de integridade ou de automatização.

Após a inclusão das regras de negócios, deve ser clicado no botão “Ok” para dar continuidade à finalização e criação da interface da aplicação a ser gerada. Clicando no botão

"OK", será apresentada uma tela, como pode ser vista na Figura 4.16. Notamos que nesta tela aparecerão todos os dados necessários para a criação da aplicação. Todas estas informações ficarão armazenadas para uma necessidade futura de manutenção nesta aplicação, ou mesmo para um melhor entendimento de seu funcionamento pelo desenvolvedor.

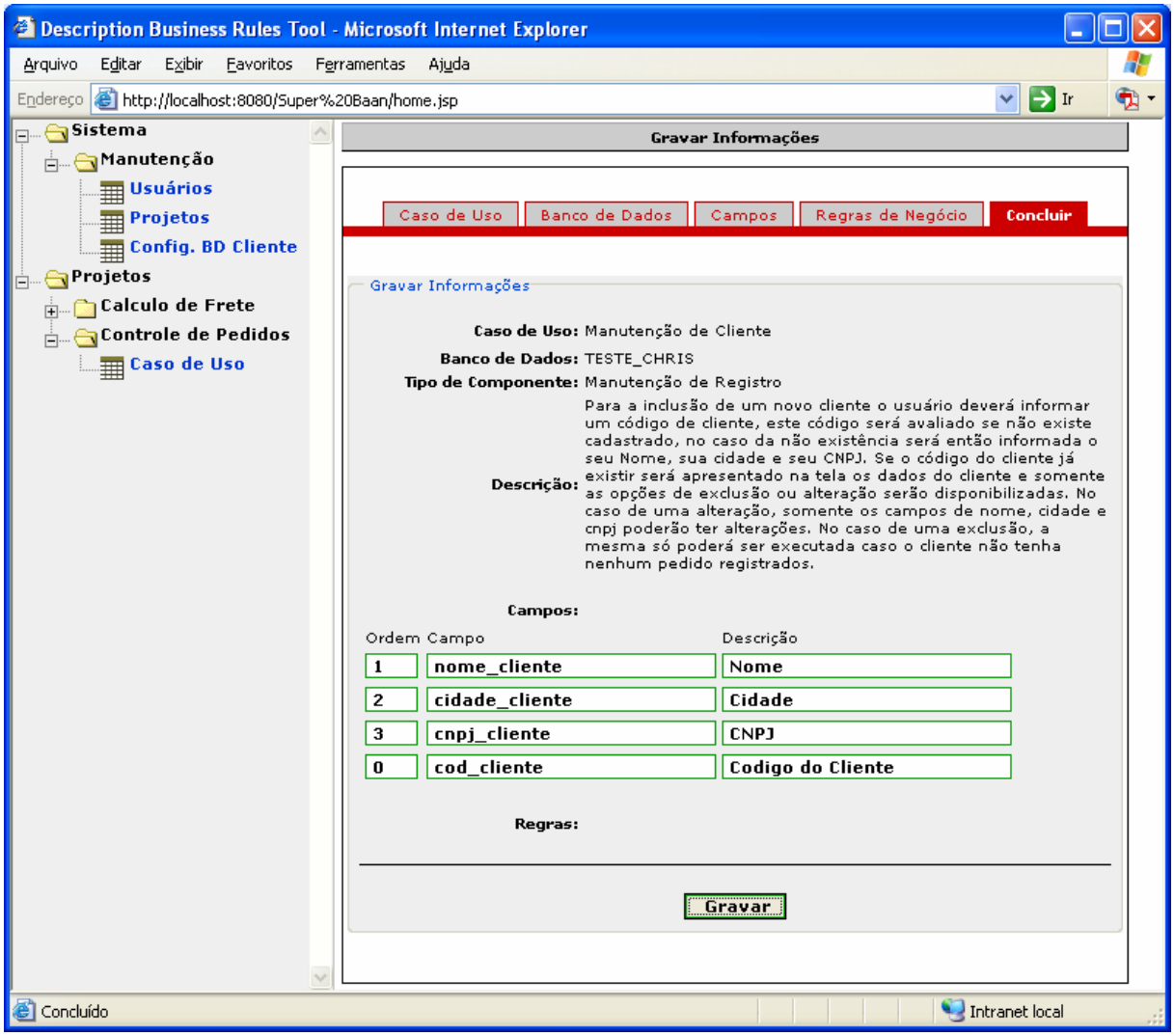

Figura 4.16 - Gravação da aplicação

Após o desenvolvedor confirmar todos os dados, basta clicar no botão "Gravar", este novo caso de uso ficará armazenado e disponível para qualquer desenvolvedor que tenha relacionamento com o projeto em questão.

Como apresentado na Figura 4.17, todos os casos gerados ficam armazenados na

seqüência em que foram criados, e nesta mesma seqüência os aplicativos ficarão disponíveis para os usuários finais (Figura 4.18).

Após a criação destas etapas, notou-se que é necessário criar uma forma diferente de apresentação dos casos de uso para os usuários finais, pois, a seqüência de criação das aplicações no menu dos usuários finais não fica bem organizada.

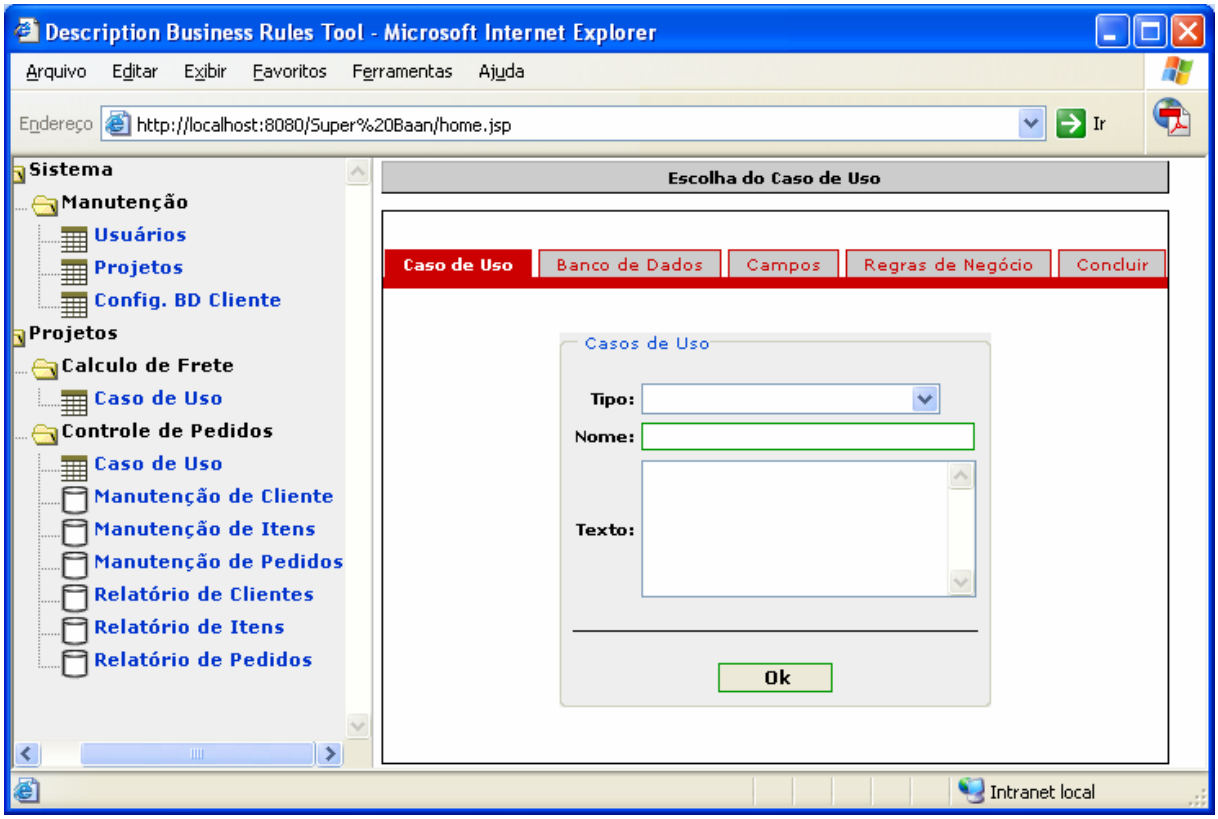

Figura 4.17 - Armazenamento dos Casos de Uso cadastrados para desenvolvedores.

Porém, como uma forma alternativa, o desenvolvedor poderá utilizar a criação de vários projetos diferentes para a criação dos menus. Por exemplo, no caso mostrado na Figura 4.18, poderia ter sido criado um projeto de Manutenções de Controle de Pedidos e outro de Relatórios de Controle de Pedidos. Entretanto, apesar de ser válida, não é a melhor alternativa.

Um exemplo da interface dos aplicativos criados pela ferramenta também está apresentado na Figura 4.18, sendo este um aplicativo de manutenção. Além dos botões

"Insere" e "Limpar", também deverá estar disponível um botão de "Ajuda", e este quando pressionado mostrará o texto que foi cadastrado como texto. Sendo assim, além de poder gerar a documentação para ser criado um manual sobre todo o sistema, os usuários finais ainda terão acesso ao funcionamento de todas as aplicações de forma on-line, o que facilita ainda mais a utilização do sistema, gerando uma quantidade menor de dúvidas, falhas e erros dos usuários.

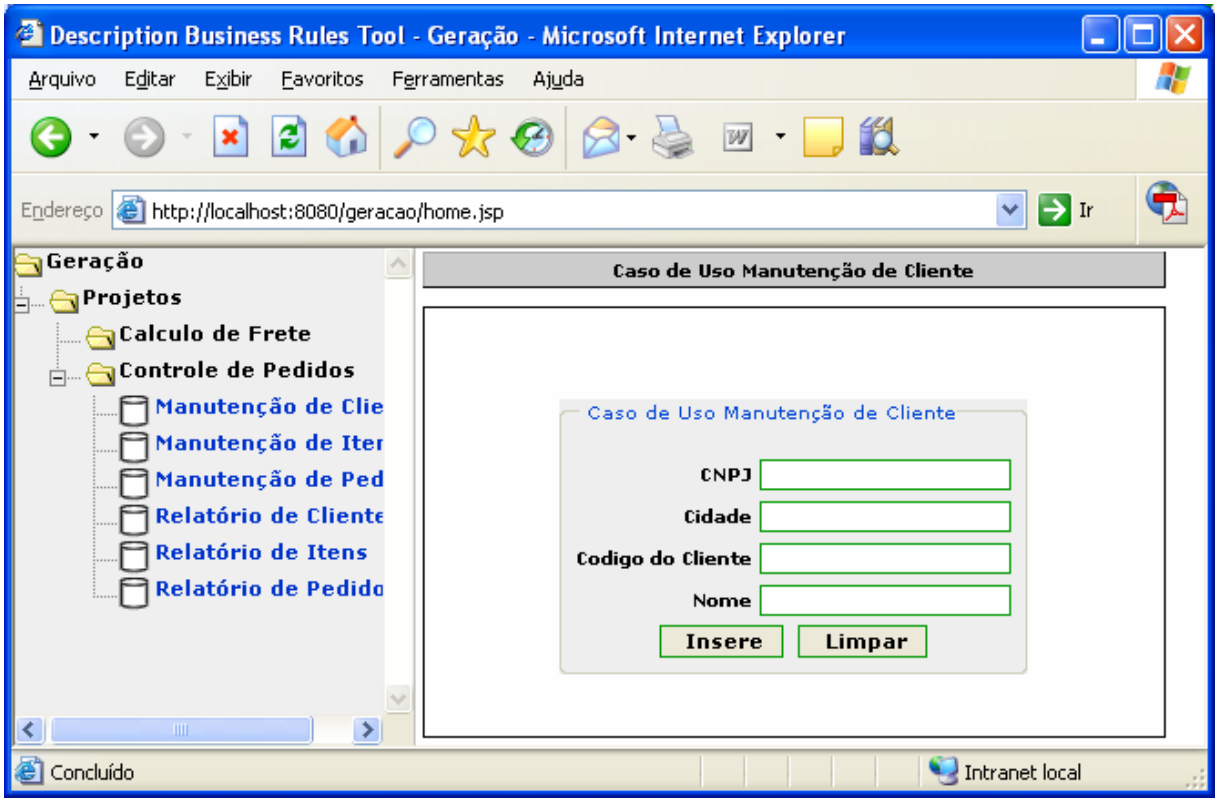

Figura 4.18 - Apresentação dos Casos de Uso para usuários finais – Manutenção de Clientes.

4.5 - Cadastramento de Regras de Negócios.

O cadastramento das Regras de Negócios é a parte mais importante desta ferramenta.

Porém, não está pronto devido às várias mudanças ocorridas desde a primeira concepção do projeto. Muito se evoluiu com os testes realizados, entretanto, muitas definições não foram bem definidas por falta de informações sobre a programação utilizando regras de negócios. Contudo, apesar da não implementação desta etapa da ferramenta, hoje, a proposta para o desenvolvimento da mesma está muito mais amadurecida e várias simulações foram realizadas, e nestes casos simulados, conseguiu-se atender as necessidades.

A primeira falha encontrada foi a de não poder criar regras de negócios que não estivessem relacionadas com as tabelas, e também a condição das regras não serem validadas no momento do preenchimento dos campos, mas sim na finalização da execução do aplicativo, como, por exemplo, na inclusão de um registro. As regras de negócios só seriam validadas após todo o preenchimento dos dados, o que atrapalhava muito no andamento das atividades, da mesma forma como acontece hoje com muitos aplicativos com interface internet que só validam os dados após tudo preenchido.

Após todos estes testes, pode-se dizer que a estrutura da Ferramenta BRD-Tool apresentada supriu várias necessidades que no início do projeto não haviam sido mapeadas como uma dificuldade do desenvolvimento.

Como apresentado na Figura 2.2, temos um repositório de regras de negócios, onde se encontram todas as regras de negócios cadastradas que são chamadas de genéricas. Todas as regras de negócios que ficam cadastradas no repositório não possuem relacionamento com nenhum projeto, e justamente por isto, são chamadas de genéricas, pois em todo e qualquer projeto que sejam criadas, será possível utilizá-las pois já estão previamente cadastradas.

Em um primeiro momento, as regras de negócios cadastradas são as genéricas, liberadas para serem utilizadas futuramente em qualquer projeto. Quando estas regras de negócios forem utilizadas em um projeto específico, será necessário informar com quais tabelas existirão os relacionamentos.

Devido a esta necessidade de cadastramento de regras de negócios genéricas, na Tabela “REPOSITORIO_RN” nota-se a existência dos mesmos campos da Tabela “REGRAS_DE_NEGÓCIOS”. Porém, os tipos dos campos são totalmente diferentes, pois no caso da primeira tabela, todos os dados são parâmetros e na tabela “REGRAS_DE_NEGOCIOS” os dados referentes ao nome das tabelas e aos campos serão cadastrados**.**

O funcionamento acontece da seguinte forma: aproveitando as regras que foram propostas no projeto de controle de pedidos (Seção 4.4.1) no repositório de regras de negócios encontram-se as três regras cadastradas e elas ficarão disponíveis para qualquer novo projeto. Porém, quando estas regras forem cadastradas para um projeto específico, além de gravar o código da regra de negócios, serão também alimentados os campos de código do projeto e do caso de uso específico.

O repositório de regras de negócios terá todos os seus campos preenchidos como parâmetro para facilitar para o desenvolvedor da ferramenta. Quando houver a criação de uma regra de negócios específica para um projeto, deve-se ter o conhecimento de quais campos das regras de negócios deverão ser preenchidos, evitando falhas no desenvolvimento.

Exemplificando, tem-se a regra de negócios de que a data do pedido não pode ser inferior a data atual. No repositório de regras de negócios, esta deverá estar cadastrada com os parâmetros apontando que a regra de negócios é do *tipo* “Integridade”, ou seja, se a condição não for satisfeita será dada uma mensagem, a *operação* será “<=”, neste caso, será necessário informar a *tabela1*, o *campo1*, o *valor2*, e o *valor_acao_true.* Com todos estes parâmetros cadastrados, quando uma regra deste tipo for criada para um projeto específico, ela só poderá ser cadastrada se houver o preenchimento de todos estes valores da forma correta, caso contrário, a regra não poderá ser cadastrada.

Para gerar para o projeto específico do controle de pedidos, ao selecionar a regra do

repositório, ela será copiada na tabela "*REGRAS_DE_NEGOCIOS*"**.** Porém, os campos de caso de uso e projeto serão gravados com os códigos gerados nas tabelas *"USE_CASE_PROJECT"*, os campos de preenchimento obrigatório serão abertos na tela de cadastramento para o seu preenchimento. Neste caso serão o *tabela1, campo1,*e *valor2*. O campo *tabela1* deverá ser preenchido com o nome da tabela de pedidos do referido projeto, e o campo *campo1* com o nome do campo da tabela que represente a data do pedidos.

Suponha que este projeto tenha a estrutura apresentada na Figura 4.10.

Neste exemplo, o campo *tabela1* deverá ser preenchido com "Pedido", o campo *campo1* com "data_pedido", o campo *valor2* com o comando de data do sistema (sysdate). Os outros campos já virão com o preenchimento especificado na tabela de repositório de regras de negócios.

Para a segunda regra de negócios mencionada, o tipo de regra seria de *"automatização"*, pois neste caso não apenas será dada uma mensagem, mas será chamado um novo caso de uso. Os campos de parâmetro serão *tabela1, campo1, valor2, cod_use_case_prj_acao_true*. O campo de *tabela1* deverá ser preenchido com "Pedido", o campo *campo1* com "valor_total_pedido", o campo *valor2* com o valor mínimo estipulado, e o campo *cód_use_case_prj_acao_true* deverá ser preenchido com outro caso de uso, que neste caso será o criado para o cadastramento dos itens dos pedidos.

Se a regra de negócios citada acima não fosse comparar o valor com outro específico, mas sim com um valor existente em uma outra tabela, por exemplo, se houvesse uma regra de negócios comparar estoque com o ponto de ressuprimento, neste caso, seria necessário**,** então, preencher os campos *tabela1, campo1, tabela2, campo2*, o restante seria igual ao mencionado anteriormente, sendo só a diferença que teríamos de preencher o campos *tabela2 e campo2* com a respectiva tabela e campo.

Com a criação do repositório de dados e já pensando na criação da *rule engine*

(máquina de regras), foi criado o campo *DEPENDENCIA*, cuja função é determinar a dependência de uma regra de negócios com a outra, ou seja, quando o usuário criar uma regra de negócios em seu projeto, caso esta regra seja dependente de outra, obrigatoriamente será preciso criar esta que está relacionada.

O exemplo que foi exercitado foi o de calcular o valor total de uma nota fiscal. Para que este valor seja calculado, deve-se, também, calcular o valor das mercadorias, além do valor dos impostos, que, por sua vez, devem ser todos calculados separadamente, visando já os livros fiscais a serem impressos. Sendo assim, devem ser calculados: o valor do IPI, o valor do ICMS, o valor do PIS e o valor do COFINS.

Para um melhor entendimento da tabela *"REPOSITORIO_RN"*, o campo *OPERACAO* tem por finalidade fazer parte da instrução SQL construída com a geração da regra de negócios. Desta forma, como ilustrado, esta operação poderá ser ">; <; >=; <=; =; avg; count", dentre outras.

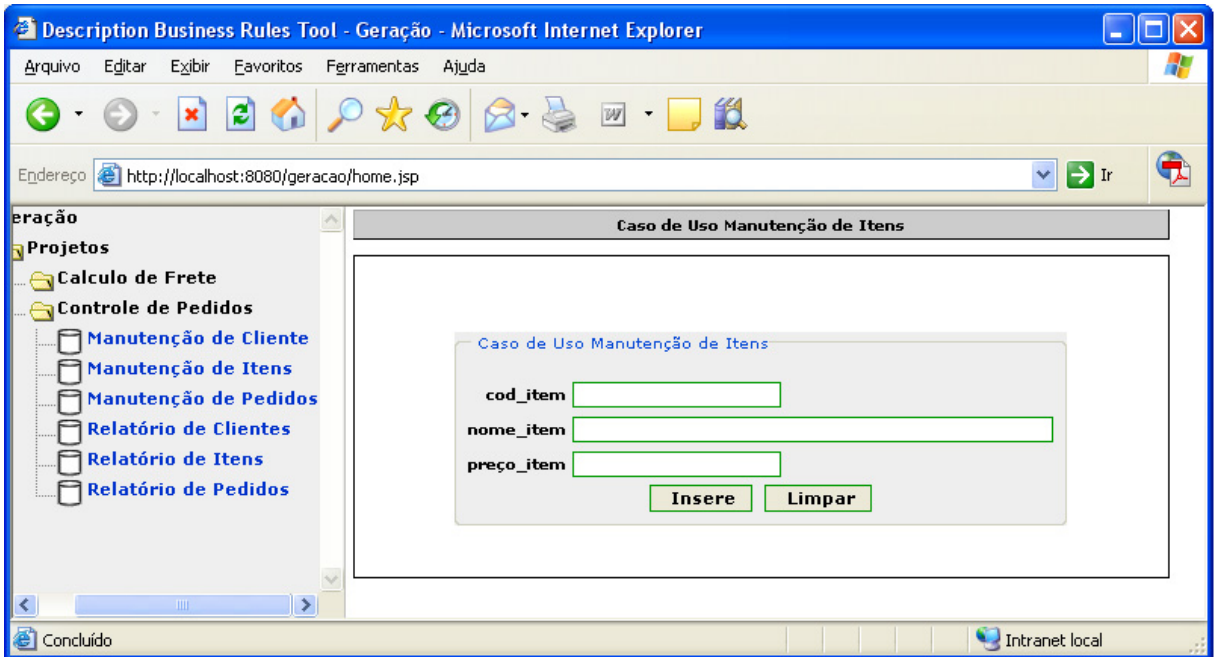

Figura 4.19 - Apresentação dos Casos de Uso para usuários finais – Manutenção de Itens

Após a aplicação do Estudo de Caso apresentado (Figura 4.10), pode-se observar os resultados obtidos nas Figuras 4.18, 4.19, os quais estarão de acordo com os resultados esperados, podendo ser notados os relacionamentos existentes das tabelas.

CONCLUSÃO

A existência de muitas pesquisas e algumas ferramentas já desenvolvidas com o uso da metodologia das regras de negócios demonstra a existência de várias vantagens que podem ser atingidas com esta tecnologia. Inclusive, muitos trabalhos de centros de pesquisas relacionados à instituições de ensino estão trabalhando com esta nova tecnologia e estão conseguindo bons resultados e auxílios à pesquisa. Porém, nota-se que não é algo tão simples de ser trabalhado.

Pode-se perceber que esta tecnologia ainda tem muito que ser explorada e que a sua utilização fará com que os sistemas de informação das empresas não estejam ligados somente às pessoas da área técnica, mas também aos gestores das empresas, de forma que eles possam criar suas ferramentas e entender mais facilmente o funcionamento das empresas.

Os conceitos estudados são apenas os pioneiros nesta área, mas muitos outros surgirão. Dessa forma, há atualmente uma grande dificuldade de se modular uma ferramenta que compreenda todas as necessidades. Porém, elas podem atender as necessidades já descobertas, e, assim, como toda tecnologia, sua evolução deverá ocorrer rapidamente.

A proposta de uma ferramenta de desenvolvimento de software baseada em regras de negócio e casos de usos abre uma etapa que determina os tipos de reúso de código ou objetos, a partir de um conjunto de casos de usos e suas respectivas regras de negócio, antes, apenas, de se utilizar o código (como as API do Java). Tais objetos confeccionados e testados a partir de uma regra de negócio, dificilmente são modificados de empresa para empresa, garantindo, assim, seu reuso sempre que necessário. Outros objetos podem ficar apenas ligados às regras de negócios como um modelo de como utilizar, mas proporcionando uma fácil adaptação de outras aplicabilidades semelhantes em empresas distintas.

Diferentemente de como acontece com as API´s do Java (onde os desenvolvedores

devem estudá-las para entender o seu funcionamento e somente após esta avaliação é possível saber se há, realmente, o funcionamento desejado pelo desenvolvedor para a sua aplicação e quais são as manutenções necessárias para uma adaptação perfeita ao seu funcionamento), quando os objetos gerados por uma ferramenta de desenvolvimento baseada em regras de negócio tornam-se inúmeras vezes mais práticos para o seu entendimento e aplicabilidade, pois é necessário que o desenvolvedor entenda qual regra deverá ser utilizada, e, após esta identificação, basta escolher a regra correta no repositório.

Além da vantagem da reusabilidade, que é um fator de peso para a opção por esta nova metodologia, outra vantagem apresentada na ferramenta proposta é a facilidade da manutenção das regras de negócios e casos de uso definidos.

Tanto as regras de negócios que devem ficar armazenadas em um repositório, quanto as aplicações desenvolvidas possuem uma documentação que se torna quase um passo obrigatório na criação de cada um destes componentes. Tal fato facilita uma manutenção em qualquer que seja o nível de abstração, pois, o desenvolvedor facilmente identificará qual é o comportamento de cada uma destas aplicações. Pelo processo de armazenamento de todos componentes necessários para a geração da aplicação, para que se faça uma manutenção na aplicação, basta apenas fazer a manutenção na etapa necessária para que a ferramenta gere a aplicação com todas as correções.

Esta proposta de uma ferramenta de apoio ao desenvolvimento de aplicações baseada em regras de negócios e casos de uso foi feita fundamentada nos conceitos estudados sobre regras de negócios e também avaliando as dificuldades observadas no entendimento do funcionamento destas regras pelos profissionais técnicos, que, apesar de conhecerem as linguagens de programação, não conhecem o funcionamento das empresas, e também dos outros profissionais que, apesar de conhecerem muito as empresas, não conhecem o desenvolvimento de sistemas.

Com esta nova metodologia, o entendimento entre estes dois profissionais fica muito mais facilitado, pois se cria uma linguagem comum, possibilitando uma diminuição na quantidade de manutenções a serem realizadas por falhas técnicas de desenvolvimento, ou mesmo de expressão do administrador, além de melhorar o cumprimento de prazo e orçamento para entrega das aplicações.

Alguns problemas foram encontrados durante a execução deste trabalho, muitos deles sanados com as pesquisas e os casos de testes realizados. Porém, outros não foram possíveis de serem sanados, mas serão relatados no tópico de trabalhos futuros para que possam ser implementados na continuidade deste trabalho.

Pode-se concluir ainda que a ferramenta que foi desenvolvida já traz grandes resultados para esta linha de pesquisa, e que realmente facilita muito o desenvolvimento de sistemas de informação para as empresas, além de tornar ágil a criação e facilitar a manutenção e portabilidade pelas plataformas tecnológicas utilizadas.

Trabalhos Futuros

Algumas das necessidades para a finalização desta proposta de ferramenta não puderam ser atingidas, devido a grande complexidade e o tempo para a conclusão da mesma, e sendo assim, ficam propostas para melhorar tal ferramenta.

Com a tabela *REGRAS_DE_NEGOCIOS* criada, e para que se possa atingir um padrão de qualidade do software, é necessário criar um aplicativo para a extração das declarações dos casos de uso para a criação da documentação das aplicações criadas.

Outro aspecto que não foi finalizado está relacionado com a manutenção das regras de negócios e casos de usos criados na ferramenta. O modelo entidade relacionamento que foi

criado já prevê esta possibilidade, pois pode-se observar que todas as tabelas registram passo a passo as necessidades para a criação da aplicação.

Neste caso, é preciso apenas a criação da aplicação que faça a engenharia reversa da aplicação criada, verificando todos os dados cadastrados para facilitar a manutenção das aplicações.

Atualmente, fala-se muito da necessidade da criação de uma *rule engine* (máquina de regras) para que a mesma possa verificar a seqüência necessária e o relacionamento existente de uma regra de negócios com outra. Esta *rule engine* é importante para dar mais agilidade ao desenvolvimento de novas aplicações, pois, no caso de dependência de uma regra de negócios não ficaria a cargo do desenvolvedor incluir todas as regras necessárias, além de não precisar dizer o encadeamento lógico que deve ser obedecido para a execução das regras de negócios.

Outra proposta a ser trabalhada nesta ferramenta é a criação de uma ordenação dos aplicativos criados pela ferramenta para serem apresentados para os usuários finais, pois, para esta solução, criou-se a possibilidade de gerar sempre novos projetos. Contudo, sabe-se que não é a melhor alternativa.

Objetivos Atingidos

Na execução deste trabalho, muitos objetivos foram atingidos. Alguns deles não podem ser demonstrados na ferramenta, pois não houve tempo hábil para a implantação devido a muitas mudanças que ocorreram em relação ao projeto inicial. Porém, é possível afirmar que o objetivo principal foi atingido, pois a implantação da ferramenta não é o principal objetivo, mas sim o entendimento do funcionamento correto de como derivar sistemas de aplicação a partir das regras de negócio.

O principal objetivo, entender o funcionamento do desenvolvimento de sistemas de informação baseado em regras de negócios, foi atingido e graças a este entendimento é possível sugerir trabalhos futuros para implantação da parte faltante da ferramenta gerada.

Muitos estudos de casos foram feitos somente baseados em simulações e é por este motivo que se pode dizer que a modelagem criada realmente funciona.

Foram realizados testes de regras de negócios, com e sem o uso de tabelas (utilizando uma ou mais tabelas), e por todos estes estudos, chega-se a conclusão de que esta ferramenta, após ser implementada, será, realmente, de grande auxilio ao desenvolvimento de sistemas de informação para as micro-empresas e pequenas empresas.

REFERÊNCIAS BIBLIOGRÁFICAS

BAJEC, Marko, KRISPER , Marjan. **Managing business rules in enterprises**. University of Ljubljana, Faculty of Computer and Information Science,2001.

BELL, J., BROOKS, D., GOLDBLOOM, E., SARRO, R., WOOK, J. **Re-engineering Case Study – Analysis of Business Rules and Recommendations for treatment or Rules in a Relational Database Environment**, Bellevue Golden: US West Information Technologies Group, 1990.

BISPO, Carlos Alberto Ferreira, **Uma análise da nova geração de sistemas de apoio à decisão.** Dissertação de Mestrado – São Carlos, SP – 1998.

BOS, Bert; LIE, Hakon Wium; LILLEY, Chris; JACOBS, Ian. **Cascading Style Sheets Level 2, CSS Especification**. Disponível em <http://www.w3.org/TR/1998/REC-CSS2-19980512/>. Acesso em: 20 dez 2005.

CAGNIN, Maria Istela; **PARFAIT: uma contribuição para a reengenharia de software baseada em linguagens de padrão e framework.** Tese de Doutorado – São Carlos, SP – 2005.

CHAN, Alessandra; **Um Ambiente de Apoio ao Uso de Padrões de Software e Requisitos de Teste no Desenvolvimento de Aplicações Web.** Qualificação de Mestrado, USP São Carlos, 2006.

CERÍCOLA, Vincent Oswald. **Oracle: Banco de Dados Relacional e Distribuído.** São Paulo: Makron Books, 1995.

CONALLEN, Jim; **Building web applications with UML**. Addison-Wesley, 1999.

COSTA, Claudio Giulliano Alves da; **Desenvolvimento e avaliação tecnológica de um sistema de prontuário eletrônico do paciente, baseado nos paradigmas da World Wide Web e da engenharia de software.** Dissertação de Mestrado – Campinas, SP – 2001.

DALFOVO, Oscar. **Quem tem informação é mais competitivo**. Blumenau. Editora Acadêmica, 2000.

DATE, C. **WHAT NOT HOW, The business Rule Approach in Application Development**, Addison-Wesley, 2000

DIAS, Paulo Roberto. **Sistema de Informação baseado em Regras de Negócios utilizando a ferramenta Genexus**. Tese de Mestrado apresentado ao Programa de Pós-graduação em Engenharia de produção da Universidade Federal de Santa Catarina. 2002.

DIAS, Paulo Roberto; BASTOS, Lia Caetano; DALFORO, Oscar; AZAMBUJA, Ricardo Alencar de. **Comparativo entre Sistema de Informação baseado em Regras de Negócios e Sistemas de Informação Desenvolvido Proceduralmente**. Artigo apresentado no XI SEMINCO (Seminário de Computação) da Universidade Regional de Blumenau. 2002.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistemas de banco de dados.** São Paulo: Addison Wesley, 2005.

FIELDING, R.; IRVINE, U. C.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T. Hypertext **Transfer Protocol – HTTP/1.1.** Copyright (C) The Internet Society, edition of the "Internet Official Protocol Standards", 1999. Disponível em:
<http://www.w3.org/protocols/rfc2616/rfc2616.html>. Acesso em: 06 jan. 2006.

FU, G., SHAO, J., EMBURY, S.M., GRAY, W.A., LIU, X. **A framework for Business Rule Presentation**. Dept. of Computer Science, Cardiff University, UK. 2001

FURLAN, José D. **Unified Modeling Language.** São Paulo: McGrawHill, 1998.

FRYDRYCH, M. **Internet programming**. Lappeenranta Univesity of Technology. Department of Information Technoly, 2001. Acessado em 19/05/2006. Disponível em <http://www.it.lut.fi/opetus/00-01/010577001/.>

THE BUSINESS RULE GROUP, **Guide Business Rules Projetct:** ***Final Report***, revision 1.3, July 2000. Disponível em <http://www.businessrulesgroup.org>.

HARRINGTON, Jan L. **Projetos de Bancos de Dados relacionais: teoria e prática**. Rio de Janeiro: Campus, 2002.

HERBST , H., KNOLMAYER, G., MYRACH , T. and SCHLESINGER, M. **The Specification Of Business Rules: A Comparison Of Selected Methodologies**, Artigo apresentado na IFIP Working Group 8.1 Conference CRIS 94, University of Limburg, Maastricht.

Publicado na: A.A. Verijn-Stuart, T.-W. Olle (Eds.), Methods and Associated Tools for the Information System Life Cycle, Amsterdam et al.: Elsevier 1994, pp. 29-46.

HILDRETH, Sue. **Rounding Up Business Rule.** Computerworld, 2005.

KLINGER, Daniel A.: KROTH, Eduardo. **Um software Assistente Para Especificação de Regras de Negócio.** Trabalho de Conclusão de Curso, Universidade Santa Cruz do Sul – RS, 2000.

KORTH, Henry F, SILBERSCHATZ, Abraham. **Sistema de bancos de dados.** Tradução Maurícia Heihachiro Galvan Abe; revisão técnica Prof. Wlademar W. Setzer – 2ª. ed. ver. – São Paulo: Makron Books, 1995.

KURNIAWAN, Budi. **Programando em JavaServer Faces**. Rio de Janeiro: Editora Ciência Moderna, 2004.

LARMAN, Craig**. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao Processo Unificado** – 2ª. Edição – Porto Alegre: Bookman, 2004.

LAUDON, Kenneth C., LAUDON, Jane P. **Management Information systems : managing the digital firm**. 7th ed. - Prentice Hall. 2002.

LAUDON, Kenneth C. LAUDON, Jane P. **Sistemas de informação gerenciais : administrando a empresa digital**. Tradução Arlete Simille Marques; Revisão técnica Érico Veras Marques, Belmiro João. - São Paulo : Prentice Hall, 2004

MEDEIROS, Ernani Sales de. **Desenvolvendo software com UML 2.0: definitivo.** São Paulo: Pearson Makron Books, 2004.

O'BRIEN, James A. **Sistemas de informação e as decisões gerenciais na era da Internet**; Tradução Célio Knipel Moreira e Cid Knipel Moreira. - 2 ed. - São Paulo: Saraiva, 2004

OLIVEIRA, ANDRÉ COSTA DE. **Metodologia para desenvolvimento de Sistemas de Informação através da utilização de módulos autônomos.** Dissertação de Mestrado, UFRGS, Porto Alegre, 1998.

PRESSMAN, Roger S.; **Engenharia de Software**; tradução José Carlos Barbosa dos Santos; revisão técnica José Carlos Maldonado, Paulo Cesar Masiero, Rosely Sanches. – São Paulo : Makron Books, 1995.

POSTGREESQL, D**ocumentação do PostgreSQL 8.0.0.** The PostgreSQL Global Development Group, traduzido por Halley Pacheco de Oliveira, Revisado por Diogo de Oliveira Biazus e Marcia Tonon, Copyright © 1996-2005.

REESE, George. **JDBC e Java**. São Paulo: Berkeley Brasil, 2001.

SEBESTA, Robert W.; **Conceitos de linguagens de programação.** Trad. José Carlos Barbosa dos Santos. – 4ª. Ed – Porto Alegre: Bookman, 2000.

SILVA, Luís Alexandre Estevão da, **Geração Dinâmica de Interfaces de Bibliotecas Digitais Baseada em Metadados.** Dissertação de Mestrado, Instituto Militar de Engenharia, Rio de Janeiro – RJ, 2000

SIMONETTO, Eugênio de Oliveira, **Uma proposta para a incorporação de aspectos temporais, no projeto lógico de banco de dados, em SGBDs Relacionais**. Dissertação de Mestrado. Pontifícia Universidade Católica do Rio Grande do Sul, 1998.

SOLBERG, Arnor; BERRE, A.J. **Component Bases Metodology Handbook – COMET.** Acesso em Fevereiro de 2005. Disponível em http://www.ifi.uio.no/~samsys/comet/comet.html

SOMMERVILLE, Ian. **Engenharia de Software.** Tradução André Mauricio de Andrade Ribeiro; revisão técnica Kechi Hirama. São Paulo : Addison Wesley, 2003.

SOUZA, Gabriela T. de; PIRES, Carlo Giovano S.; BELCHIOR, Arnaldo Dias. **Padrões de Requisitos para Especificação de Casos de Uso em Sistemas de Informação.** Artigo publicado pela Conferência Sugarloaf-PLoP, 5th Latin American Conference on Pattern Languages of Programming, Campos do Jordão, 2005.

TORRICO, Frans Novillo, TANAKA, Astério Kiyoshi, MOURA, Ana Maria de Carvalho**. Especificação de Regras de Negócio para Banco de Dados Relacional-Objeto**. Instituto Militar de Engenharia (IME), Departamento de Engenharia de Sistemas. 2000

TURBAN, Efraim, Rainer Jr., R. Kelly, Potter, Richard E. **Administração de tecnologia da informação**; tradução Teresa Cristina Felix de Souza. - Rio de Janeiro :

Campus, 2003.

VALLE, Natasha A.Y. do; SOUSA, Raquel C. de; UNSONST, Valéria A.F.; ANQUETIL, Nicolas, **Critérios de Avaliação para Reengenharia de Sistemas Legados.** II Workshop de Manutenção de Software Moderna 2005, Manaus – AM, 2005.

YOUNG, Chu Shao. **Banco de Dados: Organização, Sistemas e Administração**. São Paulo: Editora Atlas S.A., 1983.

ZORZO, Sérgio Donizetti, **Programação para Web.** WebMedia & LA-Web Joint Conference 2004 – Ribeirão Preto – SP – 2004.

APÊNDICE A – SCRIPT PARA CRIAÇÃO DO BANCO DE DADOS DA FERRAMENTA BRD-TOOL

|-- Criar a tabela de usuários.

```
CREATE TABLE usuario

(
  login varchar(50) NOT NULL,
  nome varchar(50),
  sobrenome varchar(50),
  senha varchar(50),
  email varchar(80),
  tipo varchar(13),
  CONSTRAINT usuario_pkey PRIMARY KEY (login)
)

WITHOUT OIDS;
ALTER TABLE usuario OWNER TO postgres;
```

|--- Criar tabela de Projetos.

```
CREATE TABLE projeto

(
  cod_prj int8 NOT NULL,
  descricao_prj varchar(100),
  CONSTRAINT projeto_pkey PRIMARY KEY (cod_prj)
)

WITHOUT OIDS;
ALTER TABLE projeto OWNER TO postgres;
```

|-- Criar tabela de relacionamento de usuários por projeto

```
CREATE TABLE user_project

(
  login varchar(50) NOT NULL,
  cod_prj int8 NOT NULL,
  CONSTRAINT user_project_pkey PRIMARY KEY (login, cod_prj),
  CONSTRAINT user_project_cod_prj_fkey FOREIGN KEY (cod_prj) REFERENCES
projeto (cod_prj) ON UPDATE RESTRICT ON DELETE RESTRICT,
  CONSTRAINT user_project_login_fkey FOREIGN KEY (login) REFERENCES usuario
(login) ON UPDATE RESTRICT ON DELETE RESTRICT
)

WITHOUT OIDS;
ALTER TABLE user_project OWNER TO postgres;
```

|-- Criar tabela do Repositório de Regras de Negócios.

```
CREATE TABLE repositorio_rn
(
  cod_regra serial NOT NULL,
  nome varchar(30),
  tipo int8,
  operacao boolean,
  valor1 boolean,
  tabela1 boolean,
  campo1 boolean,
  valor2 boolean,
  tabela2 boolean,
  campo2 boolean,
  cod_use_case_prj_acao_true boolean,
  cod_prj_acao_true boolean,
  valor_acao_true boolean,
  cod_use_case_prj_acao_false boolean,
  cod_prj_acao_false boolean,
  valor_acao_false boolean,
  dependencia boolean,
  CONSTRAINT repositorio_rn_pkey PRIMARY KEY (cod_regra),
)
WITHOUT OIDS;
ALTER TABLE repositorio_rn OWNER TO postgres;
```

|-- Criar tabela de Casos de Uso dos projetos

```
CREATE TABLE use_case_project

(
  cod_use_case_prj serial NOT NULL,
  cod_prj int8 NOT NULL,
  tipo int8,
  nome varchar(30),
  descricao text,
  cod_dbcli int4 NOT NULL,
  tabela varchar(30),
  CONSTRAINT use_case_project_pkey PRIMARY KEY (cod_use_case_prj, cod_prj),
  CONSTRAINT use_case_project_bdcliente FOREIGN KEY (cod_dbcli) REFERENCES
dbcliente (cod_dbcli) ON UPDATE RESTRICT ON DELETE RESTRICT,
  CONSTRAINT use_case_project_cod_prj_fkey FOREIGN KEY (cod_prj) REFERENCES
projeto (cod_prj) ON UPDATE RESTRICT ON DELETE RESTRICT
)

WITHOUT OIDS;
ALTER TABLE use_case_project OWNER TO postgres;
```

|-- Criar tabela das regras de negócios a serem incluídas.

```
CREATE TABLE regras_de_negocio
(
  cod_regra serial NOT NULL,
  cod_use_case_prj int4 NOT NULL,
  cod_prj int8 NOT NULL,
  nome varchar(30),
  tipo int8,
  operacao varchar(05),
  valor1 varchar(100),
  tabela1 varchar(30),
  campo1 varchar(30),
  valor2 varchar(100),
  tabela2 varchar(30),
  campo2 varchar(30),
  cod_use_case_prj_acao_true int4,
  cod_prj_acao_true int8,
  valor_acao_true text,
  cod_use_case_prj_acao_false int4,
  cod_prj_acao_false int8,
  valor_acao_false text,
  dependencia int8,
  CONSTRAINT regras_de_negocio_pkey PRIMARY KEY (cod_regra,
cod_use_case_prj, cod_prj),
  CONSTRAINT fk1 FOREIGN KEY (cod_use_case_prj, cod_prj) REFERENCES
use_case_project (cod_use_case_prj, cod_prj) ON UPDATE RESTRICT ON DELETE
RESTRICT,
  CONSTRAINT fk2 FOREIGN KEY (cod_use_case_prj_acao_true,
cod_prj_acao_true) REFERENCES use_case_project (cod_use_case_prj, cod_prj)
ON UPDATE RESTRICT ON DELETE RESTRICT,
  CONSTRAINT fk3 FOREIGN KEY (cod_use_case_prj_acao_false,
cod_prj_acao_false) REFERENCES use_case_project (cod_use_case_prj, cod_prj)
ON UPDATE RESTRICT ON DELETE RESTRICT
)
WITHOUT OIDS;
ALTER TABLE regras_de_negocio OWNER TO postgres;
```

|-- Criar tabela de especificação das características de cada campo dos
|-- casos de uso dos projetos

```
CREATE TABLE campo_caso_uso

(
  cod_use_case_prj int4 NOT NULL,
  campo varchar(20) NOT NULL,
  cod_prj int8 NOT NULL,
  ordem int4,
  descricao_campo text,
  CONSTRAINT campo_caso_uso_pkey PRIMARY KEY (cod_use_case_prj, cod_prj,
campo),
  CONSTRAINT campo_caso_uso_cod_use_case_prj_fkey FOREIGN KEY
(cod_use_case_prj, cod_prj) REFERENCES use_case_project (cod_use_case_prj,
cod_prj) ON UPDATE RESTRICT ON DELETE RESTRICT
)

WITHOUT OIDS;
ALTER TABLE campo_caso_uso OWNER TO postgres;
```

```
|-- Criar tabela para especificar o banco de dados dos clientes

CREATE TABLE dbcliente

(
  cod_dbcli serial NOT NULL,
  servidor varchar(15),
  usuario_meta varchar(50),
  senha_meta varchar(50),
  usuario_db varchar(50),
  senha_db varchar(50),
  nome_banco varchar(70),
  sgbd_cli varchar(50),
  CONSTRAINT dbcliente_pkey PRIMARY KEY (cod_dbcli)
)

WITHOUT OIDS;
ALTER TABLE dbcliente OWNER TO postgres;
```

APÊNDICE B – SCRIPT PARA CRIAÇÃO DO BANCO DE DADOS DO TESTE DE CONTROLE DE PEDIDOS

```
CREATE TABLE cliente
(
  cod_cliente int8 NOT NULL,
  nome_cliente varchar(50),
  cidade_cliente varchar(50),
  cnpj_cliente int8,
  CONSTRAINT cliente_pkey PRIMARY KEY (cod_cliente)
)
WITHOUT OIDS;
ALTER TABLE cliente OWNER TO postgres;

CREATE TABLE pedido
(
  cod_pedido int8 NOT NULL,
  data_pedido date,
  cod_cliente_pedido int8 NOT NULL,
  pagto_pedido int2,
  CONSTRAINT pedido_pkey PRIMARY KEY (cod_pedido),
  CONSTRAINT cod_cliente_pedido_fkey FOREIGN KEY (cod_cliente_pedido)
REFERENCES cliente (cod_cliente) ON UPDATE RESTRICT ON DELETE RESTRICT
)
WITHOUT OIDS;
ALTER TABLE pedido OWNER TO postgres;

CREATE TABLE item
(
  cod_item int8 NOT NULL,
  nome_item varchar(50),
  preço_item int8,
  CONSTRAINT item_pkey PRIMARY KEY (cod_item)
)
WITHOUT OIDS;
ALTER TABLE item OWNER TO postgres;

CREATE TABLE item_pedido
(
  item_pedido_cod_pedido int8 NOT NULL,
  item_pedido_cod_item int8 NOT NULL,
  item_pedido_qtde int2,
  item_pedido_valor int8,
  CONSTRAINT item_pedido_pkey PRIMARY KEY (item_pedido_cod_pedido,
item_pedido_cod_item),
  CONSTRAINT item_pedido_cod_pedido_fkey FOREIGN KEY
(item_pedido_cod_pedido) REFERENCES pedido (cod_pedido) ON UPDATE RESTRICT
ON DELETE RESTRICT,
  CONSTRAINT item_pedido_cod_item_fkey FOREIGN KEY (item_pedido_cod_item)
REFERENCES item (cod_item) ON UPDATE RESTRICT ON DELETE RESTRICT
)
WITHOUT OIDS;
ALTER TABLE item_pedido OWNER TO postgres;
```

APÊNDICE C NORMA ISO/IEC 9126

A seguir será descrita a norma na sua íntegra.

- **Parte 1 - Características e Subcaracterísticas da Qualidade**: Este documento é uma revisão da atual ISO/IEC 9126. As principais mudanças são para trazer as subcaracterísticas de um informativo anexo para a parte principal, e para transferir o modelo de avaliação do processo da parte principal para ISO/IEC 14598-1 Revisão Geral. Esta parte da norma está divida em 6 características.

1. **Funcionalidade**
    - Funcionalidade: a capacidade do software em satisfazer quaisquer funções adequando estados e necessidades implicadas quando usados sob condições especıficas.
    - Atributos da funcionalidade: conjunto de atributos de software que influenciam a existência de um conjunto de funções e suas propriedades especıficas. As funções são aquelas que satisfazem estados ou necessidades implicadas.

Notas

1. *Este conjunto de atributos caracteriza "o que" o software faz para preencher as necessidades, considerando os outros grupos, caracteriza principalmente "quando" e "como faz";*
2. *Para estados e necessidades implicados nesta característica, a notação para definição da qualidade aplicada;*
3. *Para um sistema que é operado por um usuário, a combinação de funcionalidade, operabilidade e eficiência pode ser medida externamente pela qualidade em uso.*

A funcionalidade está subdividida nas seguintes subcaracterísticas:

a) **Atributos de Adequação**: atributos de software que influenciam a presença e adequação de um conjunto de funções para tarefas específicas e objetivos do uso.
   Notas
   1. *Exemplos de adequação são tarefas compostas orientadas de funções compostas de subfunções, capacidade de tabelas.*
   2. *Os atributos internos correspondem à adequação para a tarefa na ISSO 9241-10.*
b) **Atributos de Acurácia**: atributos de software que influenciam a provisão de resultados ou efeitos corretos ou concordantes.
   Nota - Por exemplo, isto inclui os dados esperados com o grau de precisão necessário de valores calculados.
c) **Atributos de Interoperabilidade**: Atributos do software que influenciam na aderência a padrões relativos a convenções ou regulamentações legais e prescrições similares.
   Notas - Interoperabilidade é usada no lugar da compatibilidade em condições de permitir possível ambigüidade com capacidade de substituição. Atributos de software que fazem com que o software absorva a aplicação relacionada a normas, convenções ou regulamentações legais e prescrições similares.
d) **Atributos de Conformidade**: Atributos do software que influenciam na aderência a padrões relativos a convenções ou regulamentações legais e prescrições similares.
e) **Atributos de Segurança**: Atributos de software que influenciam na habilidade de prevenir acessos na o intencionados e resistir a ataques intencionados para se ter acesso na

o autorizado é informação confidencial, ou fazer modificações na o autorizadas em informação ou em programa.
Nota - Isto também se aplica para dados em transmissão.

2. Confiabilidade

- Confiabilidade: a capacidade do software em manter seu desempenho quando usado sob condições especıficas.
- Atributos de confiabilidade: um conjunto de atributos de software que influenciam na capacidade do software em manter seu nível de desempenho sob condições especıficas, para um período de tempo especıfico.

Notas

1. *Desgaste ou envelhecimento não acontecem em software. Limitações de confiança são devidas a falhas nas especificações, projetos e implementações. Falhas devidas a estas faltas dependem da forma como o produto de software é usado e do tempo de validade das opções selecionadas.*
2. *Na definição da ISO/IEC DIS 2382-12:1994, confiabilidade é "A habilidade de unidades funcionais de executar uma função requerida...". Neste documento, funcionalidade é somente uma das características da qualidade de software. Entretanto, a definição de confiabilidade foi ampliada para "manter seu nıvel de desempenho..." ao invés de "... executar uma função requerida".*

A confiabilidade está subdividida nas seguintes subcaracterísticas:

a) **Atributos de Maturidade:** atributos de software que influenciam na freqüência de erros devido a falhas no software.
b) **Atributos de Tolerância a Falhas:** atributos de software que influenciam na habilidade de um nível especıfico de desempenho em casos de falhas do software ou por violação de sua interface especıfica.
Nota - O nível especıfico de desempenho pode incluir a capacidade de salvar erros.
c) **Atributos de Recuperabilidade:** atributos de software que influenciam sua capacidade de restabelecer seu nível de desempenho e recuperar os dados diretamente afetados no caso de ocorrer uma falha e no tempo e esforço necessários.
d) **Atributos de Disponibilidade:** atributos de software que influenciam a probabilidade de um produto de software executar uma função requerida, estando em um dado ponto no tempo sob condições específicas de uso.

Notas

1. *Seguido a uma falha, um produto de software estará, à s vezes, fora de uso por um certo período de tempo, a duração durante o qual õ acessado por sua recuparabilidade*
2. *Externamente, disponibilidade pode ser avaliada como sendo a proporção do tempo total durante o qual o produto de software está no estado de recuperabilidade.*
3. *Disponibilidade é entretanto a combinação de confiabilidade (que controla a freqüência de falhas) e recuperabilidade (que controla a duração do tempo de recuperação seguido a cada falha).*

3. Usabilidade

- Usabilidade: a capacidade do software em ser fêcil de usar e satisfazer o usuário, quando usado sob confições especıficas.

- Atributos de usabilidade: um conjunto de atributos de software que influenciam na capacidade do software contido no sistema em ser fácil de usar e satisfazer o usuário, sob condições especıficas.

Notas

1. *Alguns atributos para a funcionalidade, confiabilidade e eficiência também poderiam ter influência sobre usabilidade, porém para esta proposta de Padrão Internacional não são classificados como atributos de usabilidade.*
2. *"Usuários" podem ser interpretados como o mais direto significado de usuários de software interativo. Usuários podem incluir operadores, usuários finais e usuários indiretos que estão sob a influência ou dependência do uso do software. Usabilidade deve abranger todos os ambientes diferentes do usuário que o software pode afetar, que pode incluir preparação para uso e avaliação de resultados.*

A usabilidade está subdividida nas seguintes subcaracterísticas:

a) **Atributos de Compreensibilidade: atributos de software que influenciam na** capacidade do usuário entender se o software Í adequado, e como ele pode ser usado para tarefas e condições de uso particulares.
   Nota - Isto dependerá da documentação e impressão inicial dada pelo software.
b) **Atributos de Aprendizagem:** atributos de software que influenciam a facilidade com a qual o usuário pode aprender suas aplicações (por exemplo, controle de operação, entrada e saída).
   Nota - Os atributos internos correspondem à adequação para aprender em ISO 9241-10.
c) **Atributos de Operabilidade:** atributos de software que influenciam no esforço necessário para o usuário poder operar e manter o controle de operação.
d) **Atributos de Satisfação:** atributos de software que influenciam na capacidade do usuário de gostar de usar o software.

Nota

1. *Os atributos correspondem à habilidade de controle, tolerância de erros e conformidade com as expectativas do usuário na ISO 9241-10.*
2. *Para um sistema que õ operado por um usuário, a combinação de funcionalidade, operabilidade e eficiência podem ser medidos externamente na qualidade em uso.*

4. Eficiência

- **Eficiência**: os recursos usados por um software contido no sistema para alcançar a performance requerida sob condições especıficas.
- **Atributos de eficiência**: um conjunto de atributos de software que influenciam na relação da performance requerida no software e a quantidade de recursos usados, sob condições especıficas.

Notas

1. *Recursos pode incluir outros produtos de software, facilidades de hardware, materiais (isto é papel de impressão, disquetes)*
2. *Para um sistema que é operado por um usuário, a combinação de funcionalidade, operabilidade e eficiência pode ser medida externamente por qualidade em uso: a efetividade do usuário, eficiência e satisfação (Isto é definido como usabilidade na ISO 9241-11).*

A eficiência está subdividida nas seguintes subcaracterísticas:

a) **Atributos do Comportamento em Relação ao tempo:** atributos de software que influenciam no tempo de resposta e processamento e desempenho na execução de suas funções.
b) **Atributos da Utilização de Recursos:** atributos de software que influenciam na quantidade de recursos usados e a duração de tal uso na execução de suas funções.

5. Manutenibilidade

- Manutenibilidade: os recursos necessários para fazer modificações específicas no software.
- Atributos de manutenibilidade: um conjunto de atributos de software que influenciam nos recursos necessários para fazer modificações especıficas.

Nota - Modificações podem incluir correções, aperfeiçoamentos ou adaptações no software para mudanças de ambiente, e em requerimentos e especificações funcionais.

A manutenibilidade está subdividida nas seguintes subcaracterísticas:

a) **Atributos de Legibilidade:** atributos de software que influenciam na necessidade de recursos necessários para diagnostico de deficiências ou causas de falhas, ou para identificação de partes a serem modificadas.
b) **Atributos de Modificabilidade:** atributos de software que influenciam na necessidade de recursos necessários para implantar as modificações especificadas.
   Nota - Valores desta subcaracterísticas podem ser alterados pelas modificações em questão.
c) **Atributos de Entidade:** atributos de software que influenciam no risco de efeitos inesperados das modificações.
d) **Atributos de Testabilidade:** atributos de software que influenciam na necessidade de recursos necessários para validação do software modificado.
   Nota - Valores desta subcaracterísticas podem ser alterados pelas modificações em questão.

6. Portabilidade

- Portabilidade: a capacidade de transferir o software para outros ambientes.
- Atributos de portabilidade: um conjunto de atributos de software que influencia na habilidade do software ser transferido de um ambiente para outro.

Nota - O ambiente pode incluir ambiente organizacional, de hardware e de software.

A portabilidade está subdividida nas seguintes subcaracterísticas:

a) **Atributos de Adaptabilidade:** atributos de software que influenciam na capacidade e esforço necessário para sua adaptação em ambientes diferentes especificados, sem aplicar outros meios ou ações para atingir tal propósito para o software.
   Notas
   1. *Adaptabilidade inclui a seqüência da capacidade interna (isto õ , telas, tabelas, volumes de transação, formatos de relato rios, etc).*
   2. *Adaptabilidade corresponde à disponibilidade para individualização na ISSO 9241-10.*
b) **Atributos de Instalação:** atributos de software que influenciam o esforço necessário para instalar o software no ambiente especificado.

c) **Atributos de Coexistência:** atributos de software que influenciam a habilidade do software de coexistir com outro software independente, em ambiente comum, compartilhando recursos comuns.
d) **Atributos de Conformidade:** atributos de software que fazem o software manter padrões ou convenções relativas é portabilidade.
e) **Atributos da Capacidade de Substituição:** atributos de software que proporcionam a oportunidade e influenciam no esforço requerido para usá-lo em lugar de outro software especıfico no ambiente de tal software.

Notas

1. *Capacidade de Substituição é usada no lugar de compatibilidade para evitar possível ambigüidade com interoperabilidade.*
2. *Capacidade de Substituição não quer dizer que este software está apto a substituir o software sob consideração.*

Capacidade de Substituição pode incluir atributos de ambas as instalação e adaptabilidade. O conceito foi introduzido como uma subcaracterísticas de si mesmo devido a sua importância.

- **Parte 2 - Métricas Externas:** Uma métrica externa Í uma escala quantitativa e o método que pode ser usado para medir uma característica ou subcaracterísticas do software independentemente do comportamento do sistema que contém o software. Esta parte pode ser utilizada por programadores, avaliadores, compradores e usuários. Contém uma coleção de métricas externas e alguns guias para seu uso. Cada métrica pode ser aplicada para medição de uma característica da qualidade, uma subcaracterísticas ou um atributo externo de um produto de software. As métricas dessa parte são para especificar requerimentos da qualidade, avaliar a qualidade de produtos no teste final, e testes de aceitação. Essas métricas podem ser usadas como referencia para desenvolvimento de novas métricas, e para pesquisas e estudos em geral.

- **Parte 3 - Métricas Internas:** Uma métrica interna Í uma escala quantitativa e método que pode ser usado para medir diretamente um atributo ou uma característica do software. Essa parte Í especialmente útil para programadores e alguns avaliadores que podem obter materiais internos tais como especificações e códigos fonte. Essa parte proporciona uma coleção de métricas internas e alguns guias para seu uso. Cada métrica pode ser aplicável para medir um atributo interno do produto de software. Também proporciona características internas e modelo da qualidade que mostram a relação entre eles como um guia. As métricas listadas nessa parte são úteis para propósitos tais como, definição dos objetivos do projeto e revisa o de produtos intermediários. Essas métricas podem ser usadas como referencia para desenvolvimento de métricas novas, e para pesquisas gerais e estudos.