UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS PATO BRANCO
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ALINE FORNARI

RODRIGO DELA JUSTINA

SISTEMA PARA CONTROLE DE TESTES DE ABSORÇÃO E DRIP EM CARCAÇAS DE AVES

PATO BRANCO
2011

ALINE FORNARI

RODRIGO DELA JUSTINA

SISTEMA PARA CONTROLE DE TESTES DE ABSORÇÃO E DRIP EM CARCAÇAS DE AVES

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Campus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: profa. Beatriz T. Borsoi

PATO BRANCO
2011

ATA Nº: **191**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DOS ALUNOS **ALINE FORNARI e RODRIGO DELA JUSTINA**.

Às 09:55 hrs do dia 7 de dezembro de 2011, Bloco S da UTFPR, Campus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Robison Cris Brito (Convidado) e Tarlis Tortelli Portela (Convidado), para avaliar o Trabalho de Diplomação da aluna Aline Fornari, matrícula 1030698 e do aluno Rodrigo Dela Justina, matrícula 1030817, sob o título **Sistema para Controle de Testes de Absorção e Drip em Carcaças de Aves**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Coordenação de Informática. Após a apresentação os candidatos foram entrevistados pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 10:45 hrs foi encerrada a sessão.

Beatriz Borsoi

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Prof. Robison Cris Brito, M.Sc.
Convidado

Tarlis T. Portela

Prof. Tarlis Tortelli Portela, Esp.
Convidado

Prof. Omero Francisco Bertol, M.Sc.
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

FORNARI, Aline; DELA JUSTINA, Rodrigo. Sistema para controle de testes de absorção e drip em carcaças de aves. 2011. 66 f. Monografia de Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Pato Branco, 2011.

O abate de aves é um processo composto por várias atividades e o controle sanitário e a inspeção de qualidade são realizados em diversos pontos desse processo. Muitas vezes os dados para controle como o de absorção de água, precisam ser coletados diretamente nos locais em que a atividade é realizada e em muitos desses locais o uso de um dispositivo móvel é uma solução ideal. Isso porque se tornaria bastante difícil, pelas condições do ambiente como umidade e temperatura, manter equipamentos como *desktops* e *notebooks*. Considerando a utilidade de um sistema automatizado para o controle de testes de absorção e *drip* e as facilidades oferecidas por dispositivos móveis para a coleta de dados em ambientes como linha de abate de aves. Neste trabalho é proposto o desenvolvimento de um sistema para o controle desses testes. O sistema é composto por dois módulos: um módulo *desktop* desenvolvido em Delphi e um módulo para dispositivos móveis desenvolvido em JavaME. O módulo para dispositivos móveis visa prover a coleta dos dados e o módulo para *desktop* o acesso aos dados coletados para análise e visualização dos resultados dos cálculos realizados com os esses dados.

Palavras-chave: Java ME. Linguagem Delphi. Aplicativo para dispositivos móveis.

ABSTRACT

FORNARI, Aline; DELA JUSTINA, Rodrigo. Software to control water absorption and drip test in poultry. 2011. 66 f. Monografia de Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Pato Branco, 2011.

The killing of poultries is a process comprising several activities and sanitary control and quality inspection is carried out in various parts of this process. Often the data must be collected directly in the places where the activity is performed in many of these places and the use of a mobile device is an ideal solution. This would become very difficult because the conditions of the environment as a unit and maintain temperature equipment such as desktops and notebooks. Considering the usefulness of an automated system to control absorption and drip tests and the facilities offered by mobile devices for data collection in environments such as poultry slaughter line, we propose the development of a system for this control. The system consists of two modules: a desktop module developed in Delphi and a module for mobile devices developed in JavaME. The mobile module aims to the collection of data and the desktop module the access to data collected for analysis and visualization of the results of calculations performed with the data collected.

Palavras-chave: Java ME. Delphi Language. Mobile software.

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
ADO	*ActiveX Data Objects*
API	*Application Programming Interface*
AWT	*Abstract Window Toolkit*
BDE	*Borland Database Engine*
CDC	*Connected Device Configuration*
CLDC	*Connected Limited Device Configuration*
CLX	*Component Libraries for Cross-platform*
Eof	*End of File*
HTML	*HyperText Markup Language*
IBX	*Interbase Express*
IDE	*Integrated Development Environment*
JavaME	*Java Micro Edition*
JDBC	*Java Database Connectivity*
JFC	*Java Foundation Classes*
JVM	*Java Virtual Machine*
MIDP	*Mobile Information Device Profiles*
NDR	*Nevrona Designs Report*
ODBC	*Open Data Base Connectivity*
OLEDB	*Object Linking and Embedding, Database*
PDF	*Portable Document Format*
PHP	*Hypertext Preprocessor PHP*
PSQL	*Procedural Structural Query Language*
RAD	*Rapid Application Development*
RMS	*Record Management System*
RTF	*Rich Text Format*
RUP	*Rational Unified Process*
SGBD	*Sistema Gerenciador de Banco de Dados*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
VCL	*Visual Component Library*

LISTA DE FIGURAS

LISTAGENS DE CÓDIGO

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais com uma visão geral do trabalho, os objetivos, a sua justificativa e a organização do texto.

1.1 Considerações Iniciais

Sistemas computacionais - denominados, também, de *software*, aplicativos ou aplicações - são ferramentas auxiliares na realização de atividades de negócio. Considerando o volume de dados, de informações e de tarefas envolvidas em um negócio, a abrangência de mercado das empresas (globalização) e a necessidade de conquistar novos clientes e de manter os existentes, os aplicativos computacionais têm se tornado uma parte muito importante de um negócio.

O tipo de negócio da empresa e o volume de dados manipulados são fatores que auxiliam a determinar as atividades ou partes do negócio que serão informatizadas. Não necessariamente todo o processo de negócio de uma empresa precisa ser informatizado. As atividades que serão automatizadas dependem de uma série de fatores, incluindo o tipo de negócio, a relação entre o custo do sistema e o benefício que o mesmo pode proporcionar e o valor que a empresa pode ou está disposta a investir.

Partindo da idéia de informatizar partes de um processo de negócio verificou-se a possibilidade de implementar um sistema que auxiliasse na realização de testes de controle de qualidade em um abatedouro de aves. Um abatedouro é uma indústria de processamento com atividades que possuem intensiva participação humana. Contudo, partes do processo podem ser realizadas com o auxílio de sistemas computacionais.

Os testes de qualidade, por exemplo, são realizados em dois momentos específicos da linha de abate: no pré-resfriamento e no túnel de congelamento. Para facilitar a coleta dos dados para realização dos testes será utilizado um dispositivo móvel, um aparelho celular. Assim, a ênfase deste trabalho é a implementação do módulo para dispositivo móvel.

Os dados coletados pelo aplicativo implementado no dispositivo móvel serão armazenados em uma aplicação *desktop*. O módulo *desktop* foi implementado como trabalho de estágio pelos autores deste trabalho.

1.2 Objetivos

O objetivo geral se refere ao resultado principal obtido com o desenvolvimento do trabalho e os específicos o complementam.

1.2.1 Objetivo Geral

Implementar um sistema para dispositivos móveis para coletar dados para testes realizados em abatedouros de aves.

1.2.2 Objetivos Específicos

- Apresentar conceituações e exemplos de arquitetura de *software*, exemplificando o termo visões de arquitetura e modelos que representam essas visões.
- Exemplificar o uso de RMS para o armazenamento dos dados no aparelho celular que serão exportados para o módulo *desktop*.
- Fornecer um mecanismo para controle dos testes de absorção e *drip* realizados.
- Integrar o módulo do aplicativo para *desktop* implementado em linguagem Delphi com o módulo para dispositivos móveis implementado com Java ME, por meio de um web services.

1.3 Justificativa

O processo de abatimento de aves abrange várias atividades. Esse processo inicia com a colocação da ave viva para ser abatida na esteira de pendura e é finalizado com a ave embalada e pronta para ser comercializada. Essas atividades, em essência de realização manual, aos poucos vão sendo realizadas com o auxílio de equipamentos e sistemas computadorizados. A automatização se refere tanto aos equipamentos que realizam automaticamente determinadas atividades do processo como aos sistemas computacionais que auxiliam na gestão e controle das atividades.

Algumas das atividades realizadas nesse ciclo, da chegada da ave até sua embalagem final, são monitoradas por procedimentos e padrões de controle de qualidade. O controle é feito por coleta de dados e testes. Dentre esses testes está o que verifica a quantidade de água retida na ave antes de a mesma ser congelada, denominado teste de absorção. E o teste que verifica a quantidade de água retida na ave após o seu congelamento, denominado Drip teste.

O sistema desenvolvido como objeto deste trabalho será utilizado na coleta dos dados para a realização dos testes, com o objetivo de verificar se os limites de água absorvida pelas aves estão de acordo com os padrões definidos.

Considerando que o sistema é implementado em duas partes distintas (*desktop* e dispositivo móvel) verificou-se a importância de modelar a solução do sistema com auxílio de conceitos de arquitetura de *software*. A definição de uma arquitetura do *software* auxilia a organizar a solução proposta para o sistema por meio das suas visões distintas que endereçam interesses específicos dos envolvidos direta e indiretamente com o uso do sistema.

1.4 Organizações do Texto

Este texto está organizado em capítulos, dos quais este é o primeiro e apresenta a ideia do sistema, incluindo os objetivos e a justificativa. No Capítulo 2 está o referencial teórico sobre arquitetura de *software*. No Capítulo 3 está o método, que é a sequência geral de passos para realizar o trabalho. Os materiais se referem ao que é necessário para modelar e implementar o sistema, incluindo as tecnologias, as ferramentas e os ambientes de desenvolvimento utilizados. O Capítulo 4 contém o resultado do trabalho que é a modelagem do sistema e a sua implementação utilizando as linguagens Delphi e Java. No Capítulo 5 está a conclusão com as considerações finais.

2 ARQUITETURA DE *SOFTWARE*

Este capítulo contém o referencial teórico do trabalho e abrange conceitos de arquitetura referenciados na literatura e um conceito proposto. O capítulo inclui, ainda, sobre conceitos sobre representações de arquitetura, visões e modelos e as visões propostas para representar uma arquitetura de *software*.

2.1 Conceitos de Arquitetura de *Software*

Uma das publicações mais referenciadas sobre arquitetura de *software*, em termos de conceitos, é de David Garlan e Mary Shaw (GARLAN, SHAW, 1994). Embora essa não tenha sido a primeira publicação sobre esse assunto - o trabalho de Perry e Wolf, por exemplo, é datado de 1992 (PERRY, WOLF, 1992) - Garlan e Shaw (1994) se destacam, além do certo pioneirismo e pelo próprio conceito, pelas muitas referências ao seu trabalho.

Garlan e Shaw (1994) se referem à arquitetura de *software* como um nível de projeto que inclui questões que vão além dos algoritmos e das estruturas de dados da computação. É a projeção e a especificação da estrutura geral do sistema, incluindo a sua organização e a estrutura de controle global, protocolos de comunicação, sincronização e acesso a dados, atribuição de funcionalidade a elementos de projeto, distribuição física do sistema e a possibilidade de seleção entre alternativas de projeto.

O conceito de arquitetura relacionado a partes de sistema e a interação e organização dessas partes é encontrado em várias definições de arquitetura de *software*. Dentre as quais estão Perry e Wolf (1992), Garlan e Shaw (1994), IEEE 1471-2000 (2000) e Bass, Clements e Kazman (2003).

Para Perry e Wolf (1992), a arquitetura de *software* é um conjunto de elementos arquiteturais que possuem alguma organização. A definição que esses autores propõem consiste na fórmula a seguir e na explicação de seus termos:

Arquitetura = {Elementos, Organização, Decisões}

A ideia de fórmula é apenas para indicar que arquitetura é composta por elementos, pela organização desses elementos e pelas decisões arquiteturais de como organizar esses elementos de forma a representar o sistema. As chaves são utilizadas apenas para agrupar os elementos e as vírgulas para separá-los.

Os elementos e sua organização são definidos por meio de decisões que satisfazem objetivos e restrições do sistema. Para esses autores, os tipos de elementos arquiteturais são:

elementos de processamento que usam ou transformam informação; elementos de dados que contêm a informação a ser usada e transformada; e elementos de conexão que ligam os elementos entre si. A organização determina as relações entre os elementos. Essas relações possuem propriedades e definem como os elementos devem interagir de forma a satisfazer os objetivos do sistema.

A Figura 1 contém a representação da arquitetura de um sistema de *software* para locação de filmes, definida de acordo com o conceito de Perry e Wolf (1992).

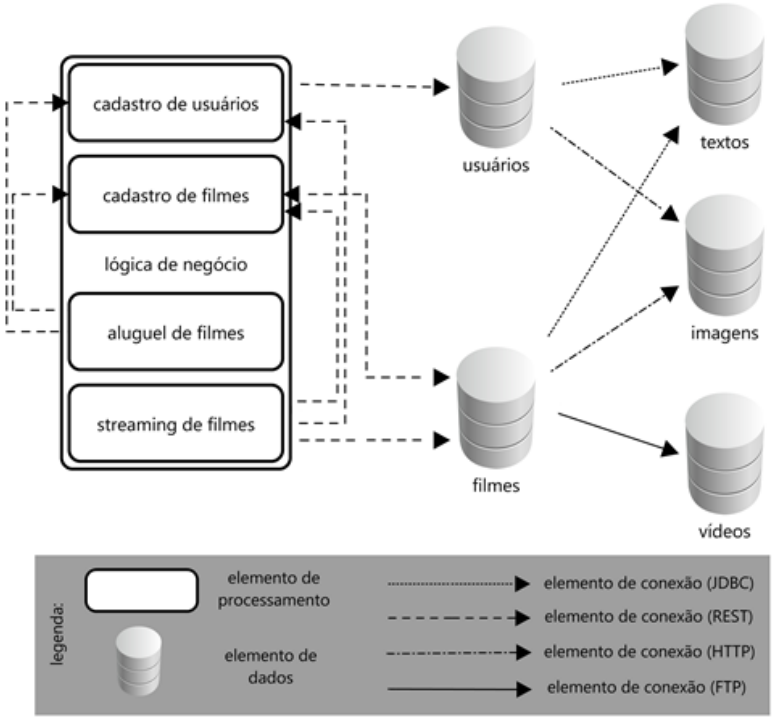

Figura 1 – Exemplo de arquitetura de *software* pelo conceito de Perry e Wolf
Fonte: Germoglio (2010).

Na representação da Figura 1, elementos de processamento são componentes (módulos, partes) do sistema; elementos de dados são repositórios de dados; e elementos de conexão são os elementos que interligam os outros elementos.

Para Garlan e Shaw (1994), arquitetura de *software* se torna necessária quando o tamanho e a complexidade dos sistemas de *software* crescem. Assim, o problema de construir sistemas transpõe a escolha dos algoritmos e das estruturas de dados adequados. Esse problema envolve decisões sobre a estrutura geral do sistema, incluindo controle¸ os protocolos de comunicação, a sincronização e o acesso a dados, a atribuição de funcionalidade a elementos do sistema e a distribuição física dos elementos do sistema. E, também, decisões que impactarão no comportamento do sistema em termos de escala e desempenho, dentre

outros atributos de qualidade.

Para Bass, Clements e Kazman (2003) a arquitetura de sistema computacional é a estrutura do sistema, que é composta de elementos de *software*, as propriedades externamente visíveis desses elementos e os relacionamentos entre eles. A Figura 2 apresenta a arquitetura de um sistema de *software* para locação de filmes definida de acordo com o conceito de arquitetura de Bass, Clements e Kazman (2003).

Figura 2 – Exemplo de arquitetura de *software* pelo conceito de Bass, Clements e Kazman
Fonte: Germoglio (2010).

No modelo da Figura 2, estão representados módulos (partes) do sistema e o relacionamento entre essas partes funcionais que são: cadastro de usuários, cadastro de filmes, aluguel de filmes, transmissão de filmes e sugestão de filmes. Esses módulos provêem serviços e informações a outras partes do sistema, o que justifica os relacionamentos entre eles. Por exemplo, uma operação de aluguel ou de transmissão de filmes deve atualizar o histórico presente na conta do usuário, porque o módulo de sugestão usará esse histórico para gerar listas de filmes de acordo com as preferências do usuário.

A IEEE 1471-2000 (IEEE, 2000) também tem como base para a definição de arquitetura elementos (no sentido de partes) e as interações entre essas partes, mas está centrada na forma de documentar a arquitetura, de organizar as suas representações em modelos de acordo com agrupamentos de interesses, que são as visões. Essa norma sugere um modelo conceitual para descrição arquitetural de sistemas intensivos em *software*. Assim, a IEEE 1471-2000 (2000) define arquitetura como a organização fundamental de um sistema incorporada em seus componentes, os relacionamentos desses componentes entre si e com o ambiente e os princípios que conduzem seu projeto e evolução.

Há ainda a definição adotada pelo RUP (*Rational Unified Process*) (JACOBSON, BOOCH, RUMBAUGH, 1999), em que a arquitetura de um sistema de *software* é vista como a organização ou a estrutura dos componentes significativos do sistema que interagem por meio de interfaces, com elementos constituídos de componentes e interfaces sucessivamente

menores.

As definições de arquitetura apresentadas neste texto são, de certa forma, convergentes. Isso porque elas se referem à arquitetura como compreendendo a estrutura (elementos, componentes), as relações entre esses componentes e as decisões (ou princípios) que determinam como esses elementos estão relacionados. Porém, o conceito apresentado pela IEEE 1471 (2000) também se refere à condução da evolução do *software*. Isso no sentido de permitir acompanhar as várias versões e alterações que um sistema de *software* pode possuir. A evolução está relacionada à adição e remoção de funcionalidades e com a manutenção do código ao longo do ciclo de vida do *software*.

Considerando que um dos principais objetivos de se projetar uma arquitetura é alcançar a qualidade desejada pelos interessados no sistema, se torna claro o papel da arquitetura em conduzir a evolução do *software*. Isso porque ela conterá decisões que contribuirão na obtenção e manutenção da qualidade definida para o sistema durante o seu ciclo de vida.

Assim, de forma simplificada, esses conceitos podem ser resumidos em que a arquitetura de *software* é a organização essencial do sistema fundamentada em seus componentes e os relacionamentos entre estes. As decisões que são tomadas para atender certos atributos de qualidade requeridos pelas pessoas envolvidas no processo de desenvolvimento de software e dos usuários desse sistema, também fazem parte da definição de arquitetura.

Como forma de sistematizar o conceito e possibilitar representações distintas da arquitetura, para este trabalho arquitetura de *software* é definida como a representação de um sistema a partir de elementos relacionados entre si definindo modelos que visam atender interesses específicos.

Os elementos e os seus relacionamentos quando definidos de acordo com determinados interesses compõem uma representação específica da arquitetura de *software*. Portanto, a arquitetura de um sistema de *software* pode ter representações distintas ou várias representações. Na definição proposta é utilizado o termo elemento para caracterizar genericamente as partes que podem compor os diversos modelos de uma arquitetura. Cada modelo representa um conjunto de interesses ou interesses específicos que são as visões.

Para Standard Computer Dictionary (IEEE, 1991) um componente é uma parte que compõe o sistema. Assim, componente pode ser entendido como subsistema, módulo, unidade ou elemento de *software*. É esse o significado de componente utilizado no padrão IEEE 1471-2000 e que foi adotado no conceito proposto neste trabalho. Contudo, como o conceito de

arquitetura de *software* proposto abrange visões distintas, o conceito de elemento não fica restrito ao código do *software*, mas sim a partes que possam compor os modelos que representam essas visões.

Essas partes podem ser desde subsistemas, ou mesmo sistemas, até componentes com uma funcionalidade bem específica. Para Chen (2003), componente é o principal elemento da arquitetura de *software e* um componente pode ser uma classe, função, procedimento, módulo, processo, grupo de objetos ou um sistema pequeno (aplicação de *software*). Apesar da categorização de classe como componente, Chen (2003) ressalta que componentes são diferentes de objetos, porque eles podem prover serviços.

Já Bass, Clements e Kazman (2003) e Perry e Wolf (1992) se referem a elementos de *software* permitindo uma interpretação mais ampla do termo, facilitando representar modelos de arquitetura de acordo com os interesses dos diversos envolvidos ou interessados em um sistema de *software*.

Considerando que um sistema de *software* pode ser complexo e amplo e que provavelmente haverá diversos envolvidos com interesses distintos, verificou-se que o termo elemento poderia ser mais adequado que componente. Componente está, muitas vezes, relacionado a código de *software*. Os elementos de um sistema de *software* podem ser: *threads* executando em máquinas diferentes, partes do sistema distribuídas em máquinas distintas, o processo de negócio que representa as funcionalidades essenciais do sistema, as tecnologias utilizadas para implementar as diferentes partes do sistema, dentre outros. Assim, elaborar um único modelo para a arquitetura de um sistema pode tornar a representação muito complexa, extensa ou mesmo incompleta para determinados interessados. Uma maneira de endereçar os diversos interesses é definir modelos distintos. Esses modelos representam, então, visões da arquitetura.

2.2 Visões em Arquitetura de So*ftware*

Uma visão arquitetural é uma representação do sistema de acordo com necessidades ou interesses dos envolvidos (interessados) no sistema. As visões de arquitetura podem ser vistas como abstrações ou simplificações do sistema, que enfatizam determinados aspectos e desconsideram detalhes que não são relevantes para a visão sendo representada.

As visões capturam as principais decisões estruturais de projeto mostrando como a arquitetura de *software* é decomposta em componentes e como os componentes são ligados de

forma a representar os interesses envolvidos. Essas alternativas de projeto devem estar associadas aos requisitos funcionais e não funcionais e políticas. As políticas definem restrições, permissões e proibições dos usuários em relação ao sistema e aos seus requisitos e com o ambiente.

Na visão em que o sistema é dividido em partes funcionais, podem ser percebidos aspectos como a partição do sistema e a comunicação entre os seus elementos. Já na visão em que o sistema é dividido em processos observam-se outros aspectos, como propriedades de comunicação e de interação entre as partes do sistema. Assim, uma visão pode apresentar os módulos lógicos (pacotes, classes, componentes bibliotecas) que compõem o sistema, além das relações de comunicação e as restrições entre eles; e outra como o sistema está dividido fisicamente, quais partes desse sistema estão executando em quais computadores e as conexões físicas entre esses computadores.

No *Rational Unified Process*, o RUP, (JACOBSON, BOOCH, RUMBAUGH, 1999), o modelo de visões é denominado "4+1" (KRUCHTEN, 1995). Esse modelo é composto pelas visões: casos de uso, lógica, implementação, processos e implantação. Além dessas quatro visões, uma visão de caso de uso sobrepõe essas outras visões e relaciona o projeto com o seu contexto e os seus objetivos de negócio (KRUCHTEN, CAPILLA, DUEÑAS, 2009).

2.2.1 Visões Propostas para Arquitetura de Software

A seguir estão as visões sugeridas por meio deste trabalho para representar a arquitetura de *software*. As visões sugeridas são: processos, tecnologias, componentes e distribuição.

a) **Processos** – representa o negócio sob a forma de processos que interagem entre si. Os processos são compostos por atividades organizadas sequencialmente. Essa visão representa os processos principais ou mais significativos que definem o negócio relacionado ao sistema de *software* implementado. Os processos ou suas divisões menores como atividades são os elementos e as interações entre essas partes são as conexões que definem a sequência de realização dos mesmos.

b) **Tecnologias** – representa o sistema por meio das tecnologias utilizadas para implementá-lo e executá-lo. Por exemplo, um sistema três camadas, pode haver tecnologias distintas para implementar e executar a camada interface, a camada de aplicação e a camada de dados.

c) **Componentes** – representa o sistema como um conjunto de elementos de *software* relacionados entre si. Esses elementos representam as funcionalidades do sistema e as interações entre eles. É uma visão que mostra os principais módulos funcionais do *software*. Os componentes do sistema são os seus elementos e as trocas de mensagens entre eles são as conexões.

d) **Distribuição** – representa a distribuição do sistema nos equipamentos utilizados para executá-lo. É a distribuição física das partes do sistema. As partes do sistema e/ou máquinas que hospedam essas partes são os elementos. A comunicação entre as suas partes são as conexões.

A composição ou definição dos modelos para representar as visões propostas para a arquitetura é baseada em elementos e relacionamentos entre eles. O Quadro 1 representa os elementos e relacionamentos de cada uma dessas visões.

Visão	**Modelos que representam as visões compostos por:**	
	Elemento	**Relacionamento**
Processo	Atividade (que compõem os processos) ou em uma visão de mais alto nível podem ser processos.	Conexão entre atividades ou entre processos.
Tecnologia	Tecnologias (utilizadas para implementar e executar o sistema)	Trocas de mensagens, comunicação, entre as tecnologias
Componentes	Partes de *software* (componentes, módulos, subsistemas)	Comunicação entre as partes que compõem o *software*
Distribuição	Equipamentos que possuem componentes de *software* alocados ou dispositivos necessários à execução do sistema.	Comunicação entre as partes do sistema.

Quadro 1 – Visões com seus elementos e relacionamentos

3 MATERIAIS E MÉTODO

Este capítulo contém as ferramentas, as tecnologias e o método utilizados para a análise e a implementação do sistema. Os materiais se referem às ferramentas e as tecnologias, incluindo linguagem de programação, banco de dados, ambiente de desenvolvimento e ferramentas para realizar e documentar a análise e a modelagem do sistema. O método se refere aos principais procedimentos e atividades realizados, abrangendo da definição dos requisitos à implantação do sistema.

3.1 Materiais

As ferramentas e as tecnologias utilizadas para modelar e implementar o sistema foram:

a) Jude Community para a modelagem;

b) Firebird como banco de dados no módulo *desktop*;

c) IBExpert como gerenciador do banco de dados;

d) Delphi como linguagem e IDE para implementar o módulo *desktop*;

e) NetBeans como IDE para implementar o módulo dispositivo móvel;

f) JavaME (*Java Micro Edition*) como linguagem para implementar o módulo para dispositivos móveis;

g) RMS para repositório de dados no módulo dispositivo móvel;

h) Web services – como tecnologia para integração de dados entre o aplicativo cliente e o aplicativo servidor;

i) Rave Reports – para geração dos relatórios.

3.1.1 Jude Community

Jude é uma ferramenta de modelagem de código fonte aberto para projeto de sistemas orientados a objeto e que suportam a UML (*Unified Modeling Language*). A partir da modelagem em UML é gerado o código Java. Jude é uma IDE para modelagem de dados criada com Java e de uso fácil e intuitivo (CAMPOS, 2011).

Com a ferramenta Jude é possível definir os seguintes tipos de diagramas: diagrama de

classe (objeto, pacote, subsistemas); diagrama de casos de uso; diagrama de estados; diagrama de atividade; diagrama de sequência; diagrama de colaboração (comunicação); diagrama de componente; diagrama de implantação e diagrama de estrutura composta.

A Figura 3 apresenta a tela principal da ferramenta Jude, com as partes principais destacadas.

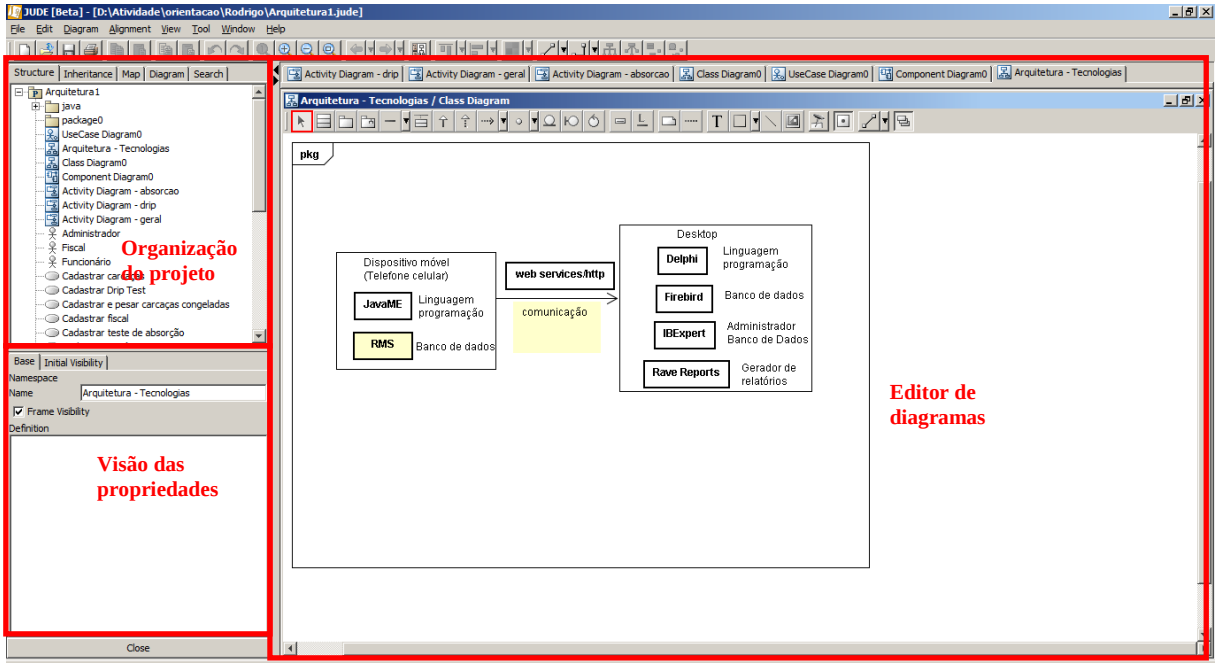

Figura 3 – Tela principal da ferramenta Jude

A ferramenta Jude está organizada em três partes básicas, conforme destacado na Figura 3:

a) Organização do projeto – apresenta as diferentes visões do projeto, são elas: *Support Structure Tree* (árvore de estrutura do projeto), *Inheritance Tree* (exibe as heranças identificadas), *MapView* (exibe todo o editor de diagrama), *DiagramList* (lista de diagramas do projeto) e *Search* (para localização de modelos e substituição de nomes).

b) Visão das propriedades – é a área que possibilita visualizar e alterar as propriedades dos elementos dos diagramas. Por exemplo, com um diagrama de classes em edição e uma classe selecionada são exibidas as suas propriedades: nome, visibilidade, atributos e operações.

c) Editor do diagrama – é a área na qual são exibidos os diagramas. Um diagrama exibido na lista de diagramas é carregado nessa área e todos os seus elementos são mostrados.

3.1.2 Firebird

O banco de dados Firebird é derivado do código do Borland InterBase 6.0 e possui o código fonte aberto e gratuito. O Firebird é um SGBD completo e oferece suporte nativo para os vários sistemas operacionais, incluindo o Windows, Linux, Solaris, MacOS, além de *triggers* de conexão e transação (CANTU, 2011).

Dentre os recursos do Firebird destacam-se: transações compatíveis com ACID (Atomicidade, Consistência, Isolamento e Durabilidade); integridade referencial; utiliza poucos recursos de processamento; linguagem nativa para *stored procedures* e *triggers* (PSQL - *Procedural Structured Query Language*); suporte para funções externas; versão *embedded* do SGBD; diversas formas de acesso ao banco de dados, como: nativo/API (*Application Programming Interface)*, dbExpress, ODBC (*Open Data Base Connectivity)*, OLEDB (*Object Linking and Embedding, Database*), .Net *provider*, JDBC (*Java Database Connectivity)* nativo tipo 4, Python module, PHP (*Hypertext Preprocessor PHP)*, Perl; *backups* incrementais; e tabelas temporárias.

3.1.3 Sistema Gerenciador de Banco de Dados IBExpert

IBExpert (IBEXPERT, 2011) é uma ferramenta para gerenciamento de bancos de dados Interbase e Firebird que pode ser obtida gratuitamente a partir do site oficial da ferramenta (www.ibexpert.com). Para utilizar o gerenciador é necessário registrar o banco de dados.

A interface do aplicativo é bastante simplificada, consistindo basicamente de uma barra de ferramentas e do DBExplorer para criar e gerenciar usuários e tabelas. A Figura 4 apresenta a tela inicial da ferramenta IBExpert.

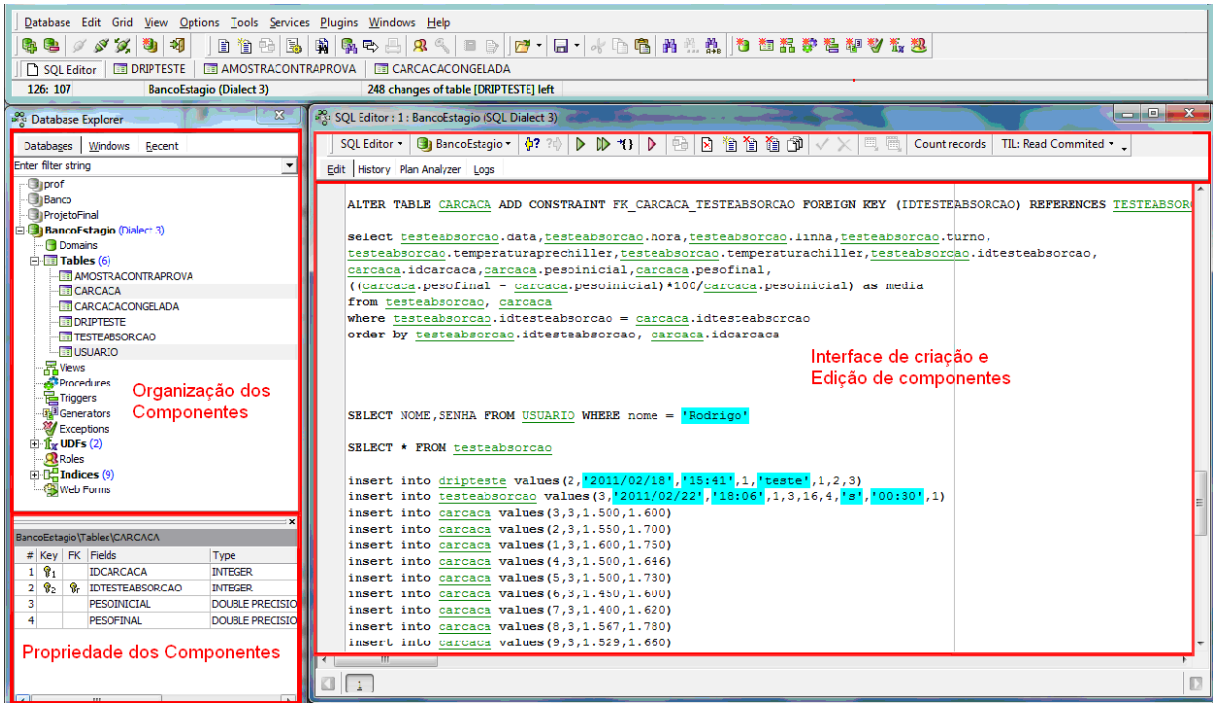

Figura 4 – Tela inicial do Sistema gerenciador de banco de dados IBExpert

O sistema gerenciador de banco de dados IBExpert está organizado em três partes básicas, destacadas na Figura 4:

a) Organização dos componentes – possibilita visualizar os bancos de dados já criados, criar um novo banco e realizar alterações em bancos de dados já existentes. As tabelas podem ser criadas no modo visual.

b) Propriedades dos componentes – permite acesso às propriedades do componente selecionado na tela de organização dos componentes.

c) Interface de criação e edição de componentes - para criar tabelas por meio de linha de código (SQL – *Structured Query Language*), bem como inserir e editar dados, selecionar visualização das tabelas e dos dados que estão armazenados nas tabelas.

3.1.4 Delphi 7

A linguagem Delphi, que está vinculada a um ambiente de desenvolvimento, se baseia em uma extensão orientada a objetos da linguagem de programação Pascal, também conhecida como Object Pascal (CANTU, 2003). Esse ambiente de desenvolvimento é composto de várias ferramentas agregadas.

Como ambiente de desenvolvimento, Delphi é um compilador e também uma IDE

para o desenvolvimento de *software*. A linguagem utilizada é *Object Pascal* (Pascal com extensões orientadas a objetos) que a partir da versão 7.0 passou a se chamar Delphi Language. A linguagem Delphi possui edições que são:

a) Delphi Personal: destinada aos iniciantes em Delphi e programadores eventuais. Não tem suporte à programação com banco de dados e para quaisquer outros recursos avançados do Delphi.

b) Delphi Professional: destinada a desenvolvedores profissionais. Inclui todos os recursos básicos e também suporte à programação com banco de dados.

c) Delphi Enterprise: versão mais completa, com acesso a bases de dados em servidores remotos, voltada para a construção de aplicativos corporativos.

Dentre as características da linguagem Delphi (CANTU, 2000; CANTU, 2003) destacam-se:

a) IDE (*Integrated Development Environment*) - ambiente integrado de desenvolvimento.

b) RAD (*Rapid Application Development*) que agiliza o desenvolvimento de aplicações.

c) Construtor visual de interface com o usuário, baseado em formulários e componentes. O uso de componentes facilita a reutilização e a manutenção de código.

d) Compilador de código nativo.

e) *Drag-and-drop design*: o código é gerado automaticamente durante a montagem do formulário.

f) *Tow-way tools*: alternar entre a visualização de um formulário e seu código (*unit*).

g) Biblioteca de componentes visuais: A VCL (*Visual Component Library*) consiste de objetos reutilizáveis incluindo objetos padrão de interface com o usuário, gerenciamento de dados, gráficos e multimídia, gerenciamento de arquivos. E é possível visualizar a hierarquia dos objetos na VCL.

h) Arquitetura aberta. Possibilidade de adicionar componentes e ferramentas de terceiros.

i) Linguagem de programação orientada a objeto (Object Pascal - 4ª geração).

j) Depurador gráfico. O *debugger* auxilia a encontrar e a eliminar erros no código.

k) Gerenciador de projetos: oferece uma visualização de todos os formulários e as *units* de um determinado projeto e oferece um mecanismo para gerenciar projetos.

l) Geradores de relatórios: o Quick Report e o Rave Reports são ferramentas de geração de relatórios.

m) BDE (*Borland Database Engine*).

n) SGBD (*Sistema Gerenciador de Banco de Dados*), Database Explorer.

o) Help *on-line*: sensível ao texto.

A Figura 5 apresenta a tela inicial de IDE da linguagem Delphi.

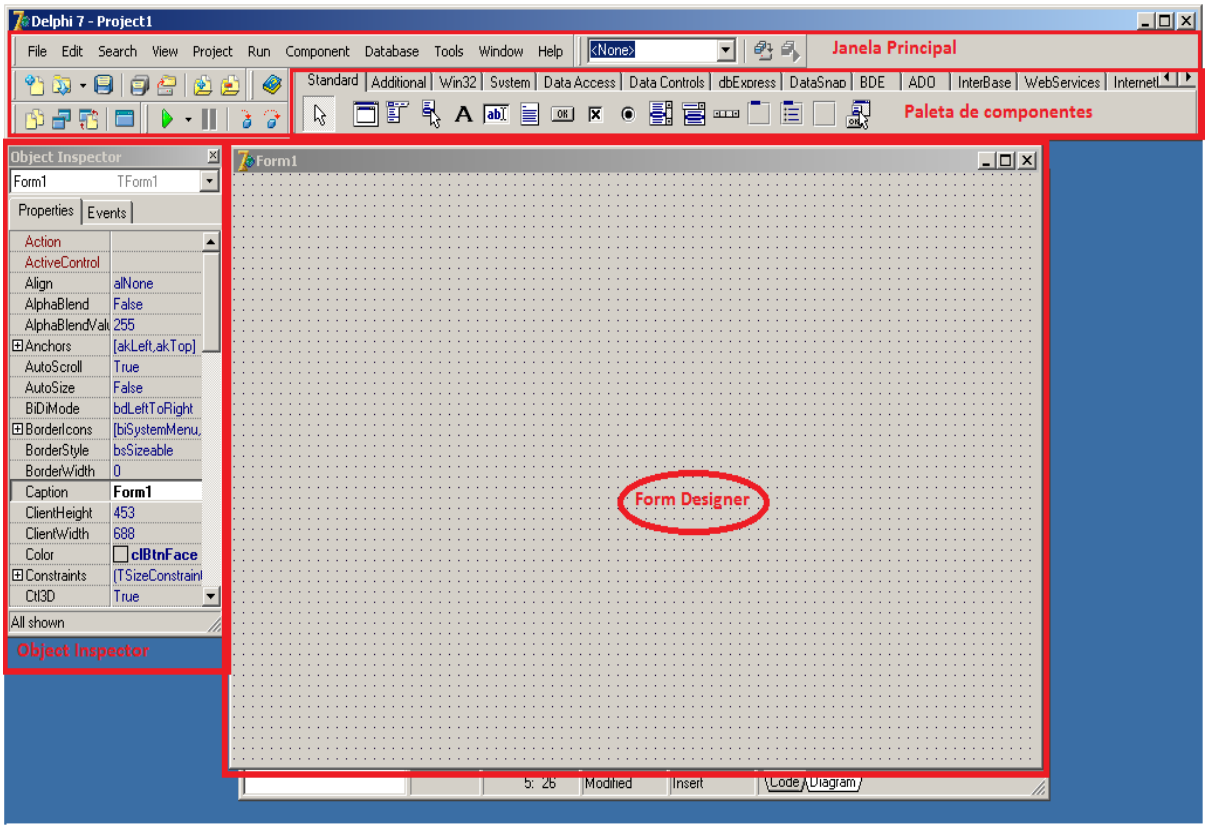

Figura 5 – Tela inicial de IDE da linguagem Delphi

As partes destacadas na Figura 5 são:

a) Paleta de componentes – a paleta de componentes é uma barra de ferramentas com componentes a serem utilizados no desenvolvimento de aplicações. A paleta é constituída de várias guias. Cada guia possui os componentes agrupados de acordo com sua finalidade.

b) *Form designer* - o *form* é uma janela onde a interface gráfica da aplicação é desenvolvida. No *form* são colocados os componentes que farão parte da interface com o usuário.

c) *Object inspector* - define propriedades e eventos para os componentes. O *object inspector* muda de acordo com o componente escolhido. O *object inspector* oferece acesso a todas as propriedades publicadas (*published*) do componente selecionado.

3.1.5 NetBeans

A IDE NetBeans é um ambiente para desenvolvimento de software de código aberto e gratuito. Essa IDE oferece ferramentas para desenvolvimento de aplicações móveis, *web* e *desktop*. A Figura 6 apresenta a tela principal da IDE NetBeans 6.9.1, com as suas partes principais destacadas.

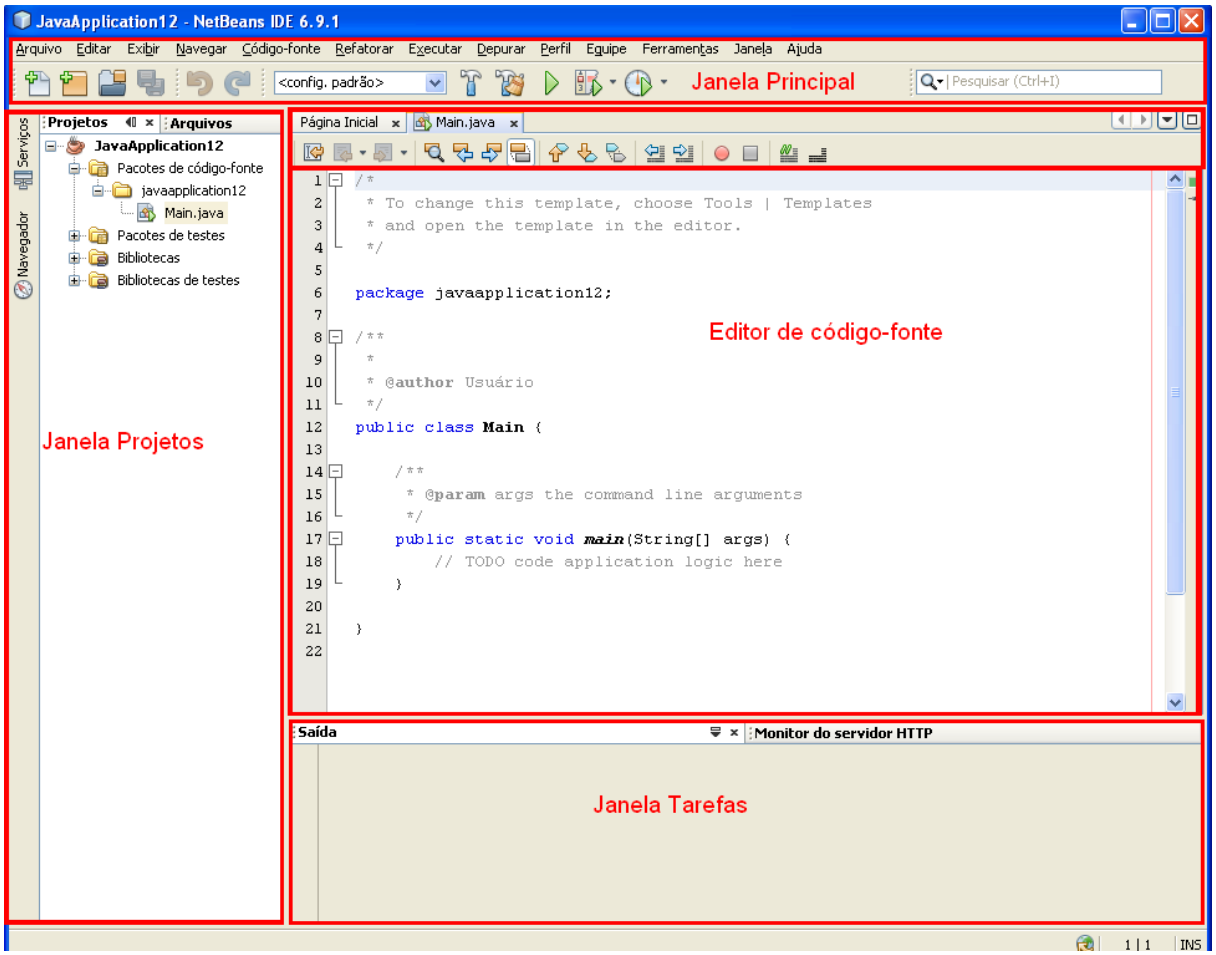

Figura 6 – Tela inicial de IDE NetBeans 6.9.1

As partes destacadas da Figura 6 são:

a) Janela principal - possui uma barra de menus e uma barra de ferramentas. Por meio delas é possível executar todas as tarefas gerais.

b) Janela projetos - contém a visualização de todos os componentes do projeto, incluindo os arquivos de código-fonte e as bibliotecas que o código precisa.

c) Janela de tarefas - Lista os erros de compilação bem como outras tarefas marcadas com palavras-chave.

d) Janela de editor de código-fonte - Local onde é escrito o código-fonte, também é

possível realizar o tratamento de eventos gerados, visualizar erros de compilação (código sublinhado em vermelho) e realizar a chamada dos métodos necessários para a execução.

3.1.6 JavaME

JavaME é a versão de Java projetada para aparelhos compactos (GOMES, 2005) e é baseada em três elementos:

Esta versão possibilita o desenvolvimento de aplicativos para dispositivos móveis. Tratando-se se uma tecnologia bastante abrangente, uma vez que pode-se utilizar o Java ME para desenvolver aplicativos para dispositivos muito limitados, como pager e celulares, até dispositivos não tão limitados, como TV Digital e Smartphone, esta tecnologia foi dividida em duas configurações para o desenvolvimento, sendo elas:

a) CDC *(Connected Device Configuration* ou Configuração para dispositivos conectados), que inclui comunicadores sem fio, navegação automobilística, equipamentos de televisão, entre outros;

b) CLDC (*Connected Limited Device Configuration* ou configurações com conexão limitada), usada em celulares;

Para o presente trabalho, como trata-se do desenvolvimento para dispositivos limitados, sendo utilizado para testes aparelhos celulares, foi utilizado o CLDC. Este possui uma camada de software que definem os componentes visuais, protocolos de acesso à rede, ciclo de vida de um aplicativo, entre outras característica. Este protocolo é chamado de MIDP (*Mobile Information Device Profiles*).

Para o desenvolvimento de aplicativos na plataforma Java ME, uma IDE que recebe destaque é o Netbeans, o qual possui um *plugin* chamado *Mobility Pack*, que permite o desenvolvimento visual de aplicativos, através de recursos como o clicar e arrastar.

A Figura 7 apresenta a IDE do NetBeans com o desenvolvimento de um aplicativo Java ME.

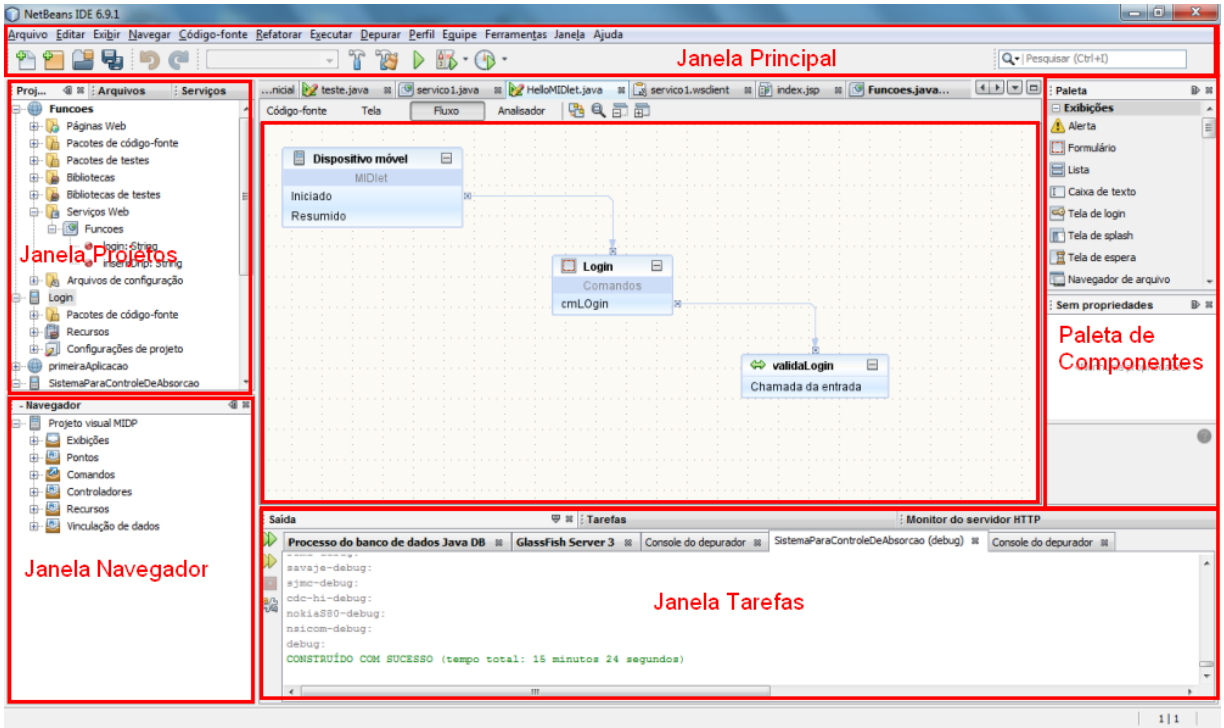

Figura 7 – Tela inicial de IDE NetBeans 6.9.1 implementando JavaME

Utilizando o *plugin* de desenvolvimento visual de aplicativos para celular, a interface gráfica da IDE é alterada, e os principais recursos apresentadas na Figura 7 são:

a) Janela principal - possui uma barra de menus e uma barra de ferramentas.

b) Janela projetos - contém a visualização dos componentes do projeto que são os arquivos de código-fonte e as bibliotecas.

c) Janela navegador - Facilita a navegação rápida entre os elementos que pertencem à classe selecionada.

d) Janela de tarefas - Lista os erros de compilação bem como outras tarefas marcadas com palavras-chave.

e) Janela de editor de código-fonte - Local onde é escrito o código-fonte, também é possível realizar o tratamento de eventos gerados, visualizar erros de compilação (código sublinhado em vermelho) e realizar chamada dos métodos necessários para a execução.

f) Janela paleta de componentes - Contém uma lista dos componentes disponíveis. É possível personalizar, criar, reorganizar e remover as categorias exibidas na paleta. Essa janela possui abas para o gerenciador do leiaute, AWT (*Abstract Window Toolkit*), JavaBeans e JFC (*Java Foundation Classes*)/Swing.

3.1.7 RMS

O MIDP (*Mobile Information Device Profile*) possui um conjunto de classes para realizar a persistência de dados no celular. A persistência de dados utilizando MIDP é baseada em arquivos textos. Cada registro é um *array* de *bytes* no arquivo de persistência e deve ser utilizado algum método para dividir o registro em campos. Para trabalhar com persistência em MIDP é utilizado um conjunto de classes chamado RMS (*Record Management System*).

O mecanismo de armazenamento do RMS é implementado como uma coleção de registros em que cada registro é organizado como um *array* de *bytes*. O tamanho desse *array* pode variar para cada registro e não existem limitações para o seu conteúdo. O *Record Store*, que corresponde as tabelas em um banco de dados tradicional, é representado pela classe **javax.microedition.rms.RecordStore**. Por meio do método **openRecordStore** é possível instanciar um objeto dessa classe.

Não existe limite de *Record Store* por aplicação. E um MIDlet acessa o Record Store apenas da sua aplicação, conforme representado na Figura 8. Contudo, a partir da versão 2 do MIDP, um MIDlet pode compartilhar e acessar os *record stores* de outros MIDlet que se encontram no mesmo MIDlet suíte.

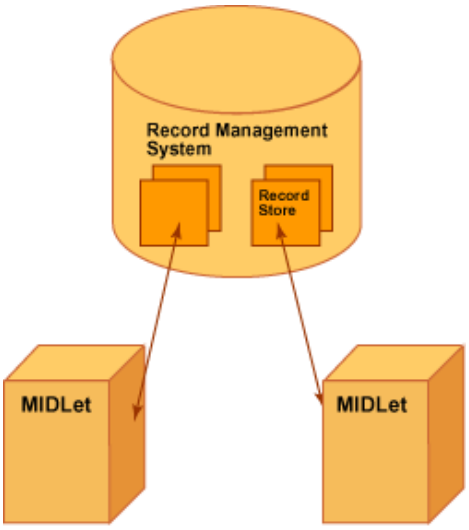

Figura 8 – MIDlets e seus respectivos RecordStore

A plataforma de software do aparelho celular é responsável por implementar a maneira como o RMS será persistido no dispositivo. Assim, os mecanismos para recuperação de erros, como reinicialização do aplicativo ou falta de bateria, devem ser implementados pelo software. Independente das características físicas do aparelho, o acesso aos dados do RMS

ocorre de forma padronizada.

Ao ser armazenado um registro no RMS, além do *array* de *bytes* correspondente ao registro, também é armazenada uma identificação única para o acesso. Essa identificação possibilita recuperar os registros e realizar operações de exclusão e de alteração de dados.

3.1.8 Web Services

Web services provem uma fundamentação conceitual e uma infraestrutura tecnológica para computação orientada a serviços (IBM, 2000 citado por THOMAS, THOMAS, GHINEA, 2003). Isso permite que programas escritos em linguagens diferentes e em plataformas distintas possam comunicar-se de forma padronizada. Web services são considerados como componentes de software reusáveis na Internet (THOMAS, THOMAS, GHINEA, 2003).

3.1.9 Rave Reports

O Rave Reports é uma ferramenta de geração de relatórios integrada a linguagem Delphi. Essa ferramenta permite tanto CLX (*Component Libraries for Cross-platform*) quanto VCL para plataformas Linux e Windows. As principais características e funcionalidades do Rave Reports são:

a) Permite o desenvolvimento de relatórios com acesso direto ao banco de dados, utilizando diversas tecnologias, como: BDE, dbExpress, ADO (*ActiveX Data Objects*) e IBX (*Interbase Express*);

b) Possui um editor visual para criação de instruções SQL integrado ao ambiente;

Os relatórios podem ser salvos nos formatos RTF (*Rich Text Format*), HTML (*HyperText Markup Language*), PDF (*Portable Document Format*), texto e um formato proprietário, o NDR (*Nevrona Designs Report*);

c) Possui diversos recursos para formatação dos relatórios, como alinhamento e posicionamento de objetos;

d) Os relatórios são baseados em páginas, regiões e bandas. É possível visualizar e testar os relatórios em tempo de projeto a partir do Rave Visual Designer;

Disponibiliza a criação de páginas padrão para serem usadas como base para vários relatórios;

e) Possibilita acesso aos objetos do relatório a partir da aplicação Delphi;

f) Permite reutilizar conteúdo e objeto entre os relatórios de um projeto Rave;

g) Possui componentes para suporte a código de barras, linguagem de programação própria a Rave Language para codificação de eventos do relatório;

h) Disponibiliza um ambiente de desenvolvimento de relatórios para o usuário final, no qual o usuário pode alterar e criar seus próprios relatórios. São disponibilizados três níveis de acesso: *beginner*, *intermediate* e *advanced*.

O Rave Report trabalha com o conceito de projetos. Um projeto pode conter vários relatórios. Os relatórios podem ser distribuídos separadamente do executável da aplicação ou incorporados ao mesmo. Todos os relatórios da aplicação podem ser salvos em um único arquivo Rave (.rav).

3.2 Método

A realização deste trabalho foi organizada em etapas. Essas etapas possuem como base o processo unificado que é iterativo e incremental (JACOBSON, BOOCH, RUMBAUGH, 1999). Diversas iterações foram realizadas até o sistema ser implementado por completo. A seguir essas etapas estão listadas de maneira sequencial, embora não tenha sido esta a sua forma de execução. Elas estão assim colocadas para facilitar a exposição.

a) **Planejamento**

Na fase de planejamento, a abrangência do sistema e a sua arquitetura geral foram discutidas. Em termos de abrangência definiu-se que o sistema atenderia os testes de Absorção e Drip Teste. A arquitetura referiu-se à definição das partes do sistema e sua distribuição nos elementos de *hardware*. O sistema seria segmentado em dois grandes blocos principais: o módulo *desktop* e o módulo para dispositivos móveis. Também ficou decidido, após análise de aspectos prós e contras, as linguagens e o banco de dados. Esses prós e contras se referiam aos próprios recursos das linguagens utilizadas durante o curso e que poderiam ser importantes para a implementação do sistema e da familiaridade dos autores deste trabalho com as mesmas. Assim, as linguagens Java e Delphi foram selecionadas.

Em termos das iterações, ficou estabelecido que:

1) Primeira iteração - definir os requisitos necessários para representar uma visão geral do sistema, independentemente da linguagem de programação utilizada; uma primeira versão da modelagem do sistema (enfatizando as visões arquiteturais de processo e tecnologia); revisão da linguagem Delphi e estudo teórico sobre arquitetura de *software*. Em termos de

monografia seria realizado um esboço do Capítulo 1.

2) Segunda iteração - complementar a modelagem do sistema, escrever o referencial teórico do trabalho, com ênfase no entendimento dos conceitos de arquitetura de *software* e suas visões, visando propor conceito e visões próprios. Além disso, a definição da interface do cadastro de usuários e a implementação do código para inclusão de usuários.

3) Terceira iteração - na revisão do referencial teórico e na descrição das tecnologias empregadas e implementação das funcionalidades de exclusão e de consulta de usuários.

4) Quarta iteração - revisão dos requisitos do módulo dispositivo móvel, a complementação da descrição da modelagem, a finalização da implementação do módulo *desktop* do sistema, incluindo os relatórios.

5) Quinta iteração - verificação dos requisitos e da modelagem, a implementação do módulo para dispositivos móveis (celular) e realização dos testes. Além da complementação da monografia.

O Quadro 1 apresenta as principais realizações de cada iteração.

	Iteração 1	**Iteração 2**	**Iteração 3**	**Iteração 4**	**Iteração 5**
Requisitos do sistema	Visão geral do sistema do ponto de vista do usuário	Revisão dos requisitos módulo *desktop*		Revisão dos requisitos módulo dispositivo móvel	Verificação de atendimento de requisitos
Modelagem	Visão processo Visão tecnologia	Completar modelagem para módulo *desktop*		Completar modelagem Finalizar modelagem	
Implementação		Interface e inclusão e alteração de usuários	Exclusão e consulta de usuários	Finalizar módulo *desktop* Elaborar relatórios	Testes para verificação do código e ajustes no código
Estudo	Revisão Delphi Arquitetura de *software*			Revisão de Java	
Monografia	Primeira versão do Capítulo 1	Escrever Capítulo 2 Escrever Capítulos 3 e 4 para o módulo *desktop*	Revisão Capítulo 2 Escrever Capítulo 3 e Seções 4.1 e 4.2	Completar Seção 4.2	Completar monografia

Quadro 2 – Iterações realizadas

b) Levantamento bibliográfico sobre arquitetura de software

O levantamento bibliográfico se centrou em publicações sobre arquitetura de *software* visando levantar o estado da arte no assunto. Do estudo desse referencial foi elaborado o conteúdo do Capítulo 2, incluindo o conceito de arquitetura e as visões propostas.

c) Levantamento de requisitos e definição do processo de negócio

O levantamento de requisitos teve como base as atividades realizadas em um abatedouro de aves, os requisitos foram definidos focados no sistema que seria implementado. Como definição de um primeiro modelo para representar a arquitetura de *software* foi definido o modelo da visão processo. Essa visão representa como ocorre o processo de abate de aves, sem considerar que partes seriam automatizadas ou como seria o sistema que as implementaria. A visão tecnologias foi definida na segunda iteração, quando foram estabelecidas as linguagens a serem utilizadas. Em seguida, à medida que a modelagem foi implementada, as demais visões foram representadas.

O levantamento dos requisitos ocorreu como atividade de estágio, para o trabalho de conclusão de curso esses requisitos foram revisados e complementados quando da implementação do módulo para dispositivos móveis.

d) Análise e projeto do sistema

O levantamento dos requisitos do sistema levou em consideração o processo de abatimento das aves, bem como a forma manual de realização dos testes de absorção e Drip teste. Também foi realizada uma revisão bibliográfica na legislação que trata da absorção de água em carcaça de aves. Com base nos dados coletados na revisão bibliográfica e no processo manual foram organizados os requisitos em casos de uso. Os casos de uso se referem às principais operações do sistema, os processos mais importantes são levados em consideração quando se organiza os requisitos em caso de uso.

Após a definição dos processos principais do sistema (casos de uso), os principais objetos identificados foram organizados em um diagrama de classes. As classes definem o estado e o comportamento dos objetos, implementando métodos e atributos. Os atributos indicam as possíveis informações que os objetos em questão possuem, já os métodos, são as operações executadas pelos objetos de uma classe. Também foram definidas as tabelas do banco de dados com seus campos.

e) Implementação

O módulo *desktop* do sistema foi implementado utilizando a linguagem Delphi. A implementação do módulo para dispositivos móveis, celular, foi realizada utilizando a linguagem JavaME. Na fase de implementação também ocorreu a realização dos testes. Esses testes se basearam na verificação de código por meio de testes unitários realizados pelos autores deste trabalho.

4 RESULTADO

Este capítulo apresenta o resultado obtido com o desenvolvimento deste trabalho, exemplificados pela apresentação, descrição da modelagem e da implementação do sistema.

4.1 Apresentação do Sistema

O sistema para controle de absorção nas carcaças de aves está centrado no cadastro dos dados utilizados nos testes, no cadastro de usuários, na emissão de relatórios e na realização dos testes. A realização dos testes tem o objetivo de verificar se os limites água absorvidos foram ultrapassados.

A coleta dos dados para a realização dos testes é executada em um dispositivo móvel e os dados são armazenados em um banco de dados no próprio dispositivo móvel. Esses dados são posteriormente descarregados no banco de dados do módulo *desktop*. Com os dados armazenados, a parte do sistema que será executada no *desktop* os acessa e faz as verificações. Essas verificações determinam se os limites de absorção estão dentro dos parâmetros estabelecidos pela legislação.

4.2 Modelagem do Sistema

A modelagem do sistema foi definida por meio da listagem dos requisitos, da definição da arquitetura, dos casos de uso, do diagrama de classes e do descritivo das tabelas do banco de dados.

4.2.1 Requisitos

Os requisitos funcionais identificados foram:

a) O acesso ao sistema é realizado por *login* e senha;
b) Cadastro de carcaças, incluindo congeladas;
c) Emissão de relatórios;
d) Cadastro de Drip teste e teste de Absorção;

e) Coleta dos dados para os respectivos testes;

f) Realização de testes de contraprova.

g) Realização dos cálculos vinculados aos testes.

Dentre os requisitos não-funcionais identificados destacam-se:

a) O usuário precisar estar logado para usar o sistema;

b) O usuário fiscal pode cadastrar um funcionário como usuário do sistema;

c) Um usuário fiscal é cadastrado pelo usuário administrador;

e) Devem ser cadastradas doze carcaças para um teste de absorção. Esse número de carcaças é definido pela Legislação do Ministério da Agricultura.

f) Devem ser cadastradas seis carcaças para um Drip teste. Esse número de carcaças é definido pela Legislação do Ministério da Agricultura.

g) Devem ser coletadas três amostras (cada uma com seis carcaças) para a realização do Drip teste, sendo uma para a realização do teste (prova) e as outras amostras para contra provas. A quantidade de amostrar é definida pela Legislação do Ministério da Agricultura.

4.2.2 Arquitetura

Para representar a arquitetura do sistema foram definidos modelos para as visões processos, tecnologias, componentes e distribuição.

a) Visão processos

A visão processos é representada pelos processos de negócio relacionados à atividade principal de uma empresa que abate e industrializa aves. Os estabelecimentos que abatem e industrializam aves seguem um processo, desde a recepção das aves vivas até a expedição dos produtos oriundos do abate desses animais. As etapas principais desse processo estão apresentadas na Figura 9.

Figura 9 – Processo de um abatedouro de aves

As duas etapas circuladas por uma elipse pontilhada na Figura 9 representam as atividades nas quais ocorrerá o uso do aplicativo desenvolvido. As etapas do processo representado na Figura 9 são:

a) Pendura viva - É o início do processo em que as aves que são transportadas em caminhões. São descarregadas e penduradas em linhas que as levarão ao abate.

b) Escaldagem e depenagem - Após a morte dos animais, eles são transportados para o interior da indústria pelas linhas de transporte. Nesta etapa as aves são escaldadas em tanques com água quente e posteriormente são depenadas, ou seja, as suas penas são retiradas.

c) Evisceração - Após a escaldagem e a depenagem, as aves são abertas com um corte abdominal e as suas vísceras são retiradas. É, também, nesta etapa que é realizada a inspeção sanitária dos animais por parte da Inspeção Federal.

d) Pré-Resfriamento - Nesta etapa do processo as aves são depositadas em tanques com água gelada para que a temperatura das mesmas seja reduzida drasticamente e também para que absorvam água em quantidade permitida pela legislação. É nesta etapa que são coletadas as aves para o teste de Absorção. A automatização dessa etapa do processo é parte do sistema proposto como resultado deste trabalho.

e) Embalagem e rotulagem – Nesta etapa as aves são embaladas e rotuladas.

f) Túnel de congelamento – No túnel de congelamento as aves são congeladas. É na saída deste túnel que são coletadas as amostras utilizadas no Drip Teste.

g) Embalagem Final - Nesta etapa as carcaças já congeladas e embaladas primariamente são colocadas em caixas com várias unidades, embalando-as secundariamente.

O processo de negócio para a realização do teste de absorção que ocorre na fase de pré-resfriamento é representado na Figura 10. Esse processo também faz parte da visão processos da arquitetura porque é um detalhamento de uma atividade do processo mais geral.

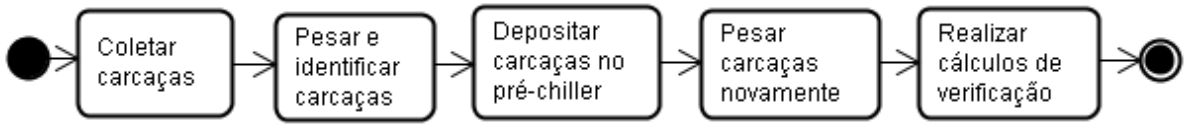

Figura 10 – Processo para realização do teste de absorção

Descrição das atividades do processo para realização do teste de absorção representado na Figura 10.

a) Coletar carcaças – Nessa etapa do processo são coletadas as doze carcaças usadas no teste de absorção.

b) Pesar e identificar as carcaças - Após coletar as carcaças essas devem ser pesadas e identificadas.

c) Depositar carcaças no pré-*chiller* – é o sistema de pré-resfriamento por água (pré-

chiller).

d) Pesar carcaças novamente - Depois de terem passado pelo processo de pré-resfriamento, as carcaças são identificadas e pesadas novamente, obtendo-se o peso final.

e) Realizar cálculos de verificação - De posse do peso inicial e do final, realiza-se o teste para verificar se as carcaças absorveram água em excesso no processo de pré-resfriamento.

O outro teste realizado é o Drip teste que também é automatizado pelo sistema. Esse teste ocorre na atividade identificada como "Túnel de congelamento" do processo representado na Figura 10. O processo representado na Figura 11 também faz parte da visão processos da arquitetura.

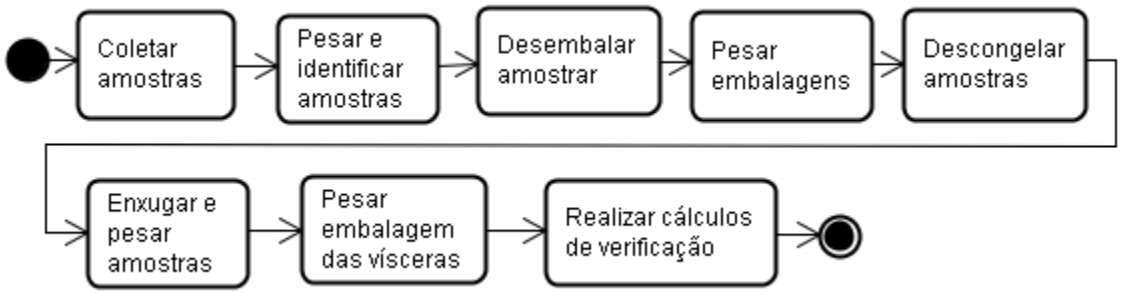

Figura 11 – Processo para a realização do Drip Teste

As atividades representadas no processo da Figura 11 são:

a) Coletar amostras - Devem ser coletadas seis carcaças congeladas para a realização do Drip teste, sendo coletadas mais duas amostras de seis carcaças. Essas constituirão a prova e contraprova.

b) Pesar e identificar amostras - Nesta etapa serão pesadas e identificadas individualmente as carcaças.

c) Desembalar amostras - Após pesá-las e identificá-las, as carcaças serão retiradas de suas embalagens e colocadas em um saco plástico.

d) Pesar embalagens - As embalagens que continham as carcaças devem pesadas.

e) Descongelar amostras - Após as carcaças terem sido retiradas de suas embalagens originais elas devem ser colocadas em sacos plásticos para serem descongeladas.

f) Enxugar e pesar amostras - Depois de terem sido descongeladas as carcaças devem ser enxugadas para se retirar o excesso de água e pesadas novamente.

g) Pesar embalagens de vísceras - Devem ser pesadas as embalagens que contém as vísceras das aves.

h) Realizar cálculos de verificação - De posse dos dados coletados das carcaças, serão

executados os métodos para verificar a quantidade de água absorvida pelas carcaças congeladas.

b) Visão tecnologias

A visão tecnologias, representada na Figura 12, contém as principais tecnologias utilizadas para implementar o sistema.

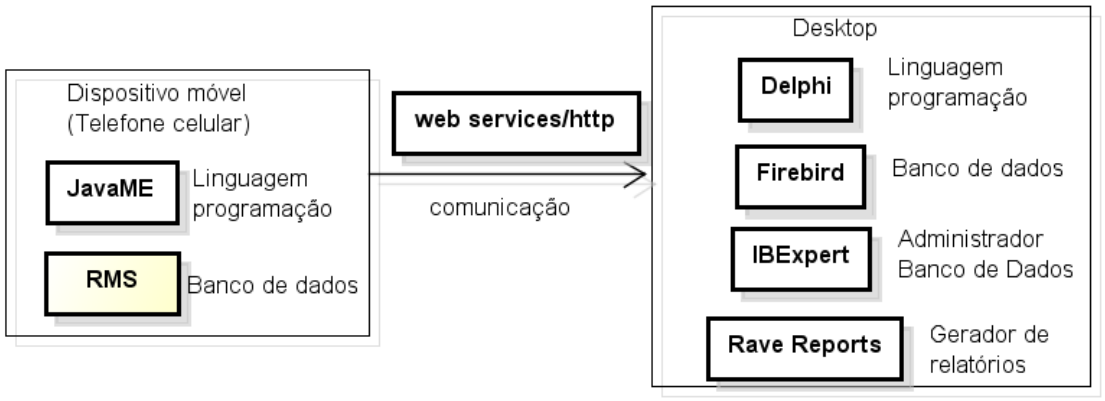

Figura 12 – Tecnologias utilizadas

Conforme apresentado na Figura 12 o sistema está dividido em dois módulos. O módulo *desktop* que tem a implementação realizada com linguagem Delphi e o módulo dispositivos móveis com a implementação realizada utilizando a linguagem Java. A comunicação entre os dois módulos é realizada por um *web services* utilizando *http*.

c) Visão distribuição

A Figura 13 apresenta a visão distribuição, os dois módulos definidos para o sistema serão implementados em dispositivos físicos distintos. O módulo *desktop* ficará em um computador que disponibilizará a aplicação para acesso aos usuários e o respectivo banco de dados utilizado pelos módulos *desktop* e móvel. O módulo dispositivo móvel estará instalado em um aparelho celular. A forma de comunicação é assíncrona. Os dados coletados para os testes são armazenados no dispositivo móvel e posteriormente exportados para o sistema *desktop*.

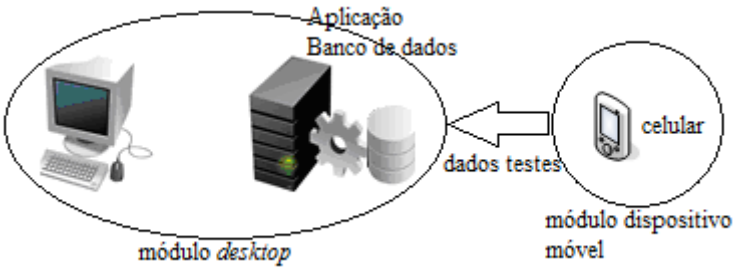

Figura 13 – Distribuição do sistema

4.2.3 Casos de Uso, Classes e Tabelas do Banco de Dados

A Figura 14 contém os casos de uso definidos para o sistema. No diagrama da Figura 14 estão definidas as fronteiras do sistema para o módulo *desktop* e para o módulo dispositivos móveis. Os casos de uso estão agrupados pelos respectivos módulos nos quais eles estão implementados.

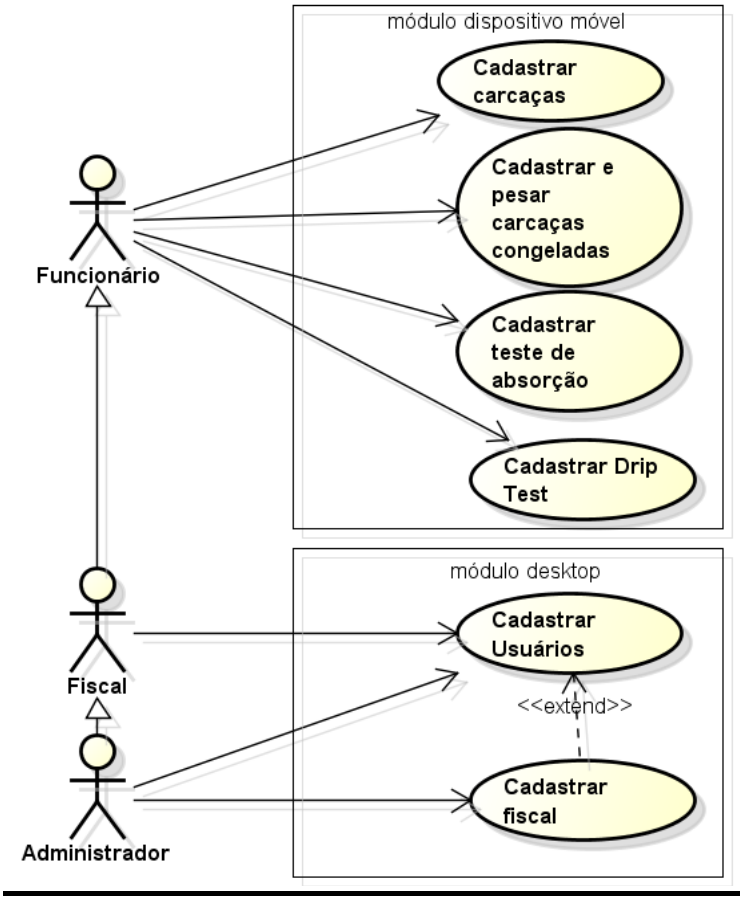

Figura 14 – Diagrama de casos de uso do sistema

De acordo com a Figura 14 foram identificados três atores: administrador, fiscal e funcionário. O ator funcionário pode cadastrar carcaças não congeladas, cadastrar carcaças congeladas, realizar pesagens e cadastrar testes de absorção e o Drip teste. O ator fiscal herda os casos de uso do ator administrador e também pode cadastrar usuários, que são funcionários. O ator administrador herda as funcionalidades do ator fiscal e também estende o caso de uso cadastrar funcionário para realizar o cadastro de fiscal. Os casos de uso apresentados na Figura 14 estão descritos nos Quadros 3 a 8.

Caso de Uso: Cadastrar carcaças. São cadastradas as carcaças utilizadas na realização dos testes de absorção, são registrados os pesos iniciais e finais dessas carcaças. A identificação da carcaça é manual (com a colocação de um número sequencial: 1, 2,3 ... 12) e realizada pelo funcionário que as pesa.
Atores: Funcionário
Pré-condições: As carcaças devem ser coletadas.
Pós-condições: As carcaças foram cadastradas.
Requisitos correlacionados: cadastrar teste de absorção
Fluxo Principal: 1- o funcionário responsável pelos testes acessa o sistema com sua senha. 2- o funcionário seleciona a qual teste as carcaças pertencem. 3- o funcionário coleta as carcaças que comporão o teste e registra o seu peso, gerando assim o peso inicial e as identifica, posteriormente as deposita no pré-*chiller*. 4- o funcionário coleta as carcaças usadas anteriormente depois de passarem pelo processo de pré-resfriamento em água gelada, identifica-as e pesa, gerando assim o peso final 5- o sistema retorna uma mensagem para o funcionário que o cadastro das carcaças foi efetuado.
Tratamento de Exceções: 1 a. Funcionário esqueceu a senha. 1 a.1 o funcionário acessa o botão que ajuda a recuperar a senha; 1 a.2 retorna ao fluxo principal. 4b. Uma das carcaças foi perdida dentro do pré-*chiller* 4 b.1 o referido teste é descartado; 4b.2 o funcionário coleta novas carcaças e inicia todo o processo novamente.

Quadro 3 – Caso de uso Cadastrar carcaça

Caso de Uso: Cadastrar e pesar carcaças congeladas. No sistema é cadastrado o peso das carcaças congeladas utilizadas na realização do Drip teste
Atores: Funcionário
Pré-condições: As carcaças congeladas em condições de serem cadastradas.
Pós-condições: As pesagens foram realizadas.
Requisitos Correlacionados:
Fluxo Principal: 1- o funcionário acessa o sistema com seu *login*. 2- o funcionário cadastra as carcaças usadas no DRIP, e pesa as carcaças com a embalagem, gerando o peso MO. 3- o funcionário retira a ave da embalagem e pesa a embalagem, obtendo assim o peso M1. 4- o funcionário coloca as carcaças para descongelar.

5- o funcionário pesa as carcaças já descongeladas com suas vísceras e a embalagem, gerando o M2 6- o funcionário pesa a embalagem de vísceras, gerando o M3.
Tratamento de Exceções: 2a. o funcionário esqueceu da sua senha de acesso ao sistema. 2a.1 o funcionário acessa a funcionalidade recuperar senha; 2a.2 retorna ao fluxo principal.

Quadro 4 – Caso de uso Cadastrar e pesar carcaça congelada

Caso de Uso: Cadastrar teste de absorção. No sistema é registrada a data da realização do teste, também a hora e turno de trabalho a linha a que as carcaças pertencem, bem como a temperatura do pré-*chiller* e do *chiller* e o tempo de permanência das carcaças no interior do pré-*chiller*.
Atores: Funcionário
Pré-condições: Necessidade de realizar o teste.
Pós-condições: Os dados do teste de absorção foram cadastrados.
Requisitos Correlacionados: cadastrar carcaças, cadastrar usuário.
Fluxo Principal: 1- o funcionário acessa o sistema com seu *login*. 2- o funcionário insere no sistema os dados que comporão o teste. 3- o sistema informa que os dados foram gravados com sucesso.
Tratamento de Exceções: 1a. o funcionário esqueceu da sua senha de acesso ao sistema. 1a.1 o funcionário acessa a funcionalidade recuperar senha; 1a.2 retorna ao fluxo principal. 2a temperatura do pré-*chiller* esta acima da permitida 2a.1 o funcionário pára o abate e segue com o teste. 2a.2 retorna ao fluxo principal. 2b temperatura do *chiller* esta acima da permitida 2b.1 o funcionário pára o abate e também pára o *chiller*. 2b.2 o funcionário segue com o teste 2b.3 retorna ao fluxo principal.

Quadro 5 – Caso de uso Cadastrar teste de absorção

Caso de Uso: Cadastrar Drip Test**.** No sistema são registradas a data da coleta das amostras, a hora e o turno de produção, também o nome oficial do produto coletado, bem como o número dos lacres das amostras. Serão coletadas carcaças para três amostras.
Atores: Funcionário
Pré-condições: Necessidade de realizar o teste.
Pós-condições: Os dados do Drip teste foram cadastrados.
Requisitos Correlacionados: cadastrar carcaças, cadastrar usuário.
Fluxo Principal: 1- o funcionário acessa o sistema com seu *login*. 2- o funcionário insere no sistema os dados que comporão o teste. 3- o sistema informa que os dados foram gravados com sucesso.
Tratamento de Exceções: 1a. o funcionário esqueceu da sua senha de acesso ao sistema. 1a.1 o funcionário acessa a funcionalidade recuperar senha;

1a.2 retorna ao fluxo principal.

Quadro 6 – Caso de uso Cadastrar Drip Test

Caso de Uso: Cadastrar Usuários. No sistema serão cadastrados os usuários que realizarão os testes. O cadastro deve gerar um nome de usuário exclusivo e uma senha de acesso.
Atores: Fiscal Federal Agropecuário e Administrador do sistema
Pré-condições: O funcionário foi cadastrado no sistema pelo administrador.
Pós-condições: O funcionário foi cadastrado como usuário do sistema.
Requisitos Correlacionados:
Fluxo Principal: 1- o funcionário inicializa o sistema e cadastra *login* e senha. 2- o sistema verifica a disponibilidade do *login*. 4- o sistema retorna para o usuário se o *login* pode ser usado.
Tratamento de Exceções: 2a. o *login* já esta cadastrado 2a.1 o funcionário altera o *login* e a senha; 2a.2 retorna ao fluxo principal.

Quadro 7 – Caso de uso Cadastrar usuários

Caso de Uso: Cadastrar fiscais. No sistema serão cadastrados os fiscais. O cadastro deve gerar um nome de usuário exclusivo e uma senha de acesso.
Atores: Administrador do sistema
Pré-condições: O funcionário foi cadastrado no sistema pelo administrador.
Pós-condições: O funcionário foi cadastrado como fiscal do sistema.
Requisitos Correlacionados:
Fluxo Principal: 1- o funcionário inicializa o sistema e cadastra *login* e senha. 2- o sistema verifica a disponibilidade do *login*. 4- o sistema retorna para o usuário se o *login* pode ser usado.
Tratamento de Exceções: 2a. o *login* já esta cadastrado 2a.1 o funcionário altera o *login* e a senha; 2a.2 retorna ao fluxo principal.

Quadro 8 – Caso de uso Cadastrar fiscais

A Figura 15 apresenta o diagrama de classes definido para o sistema.

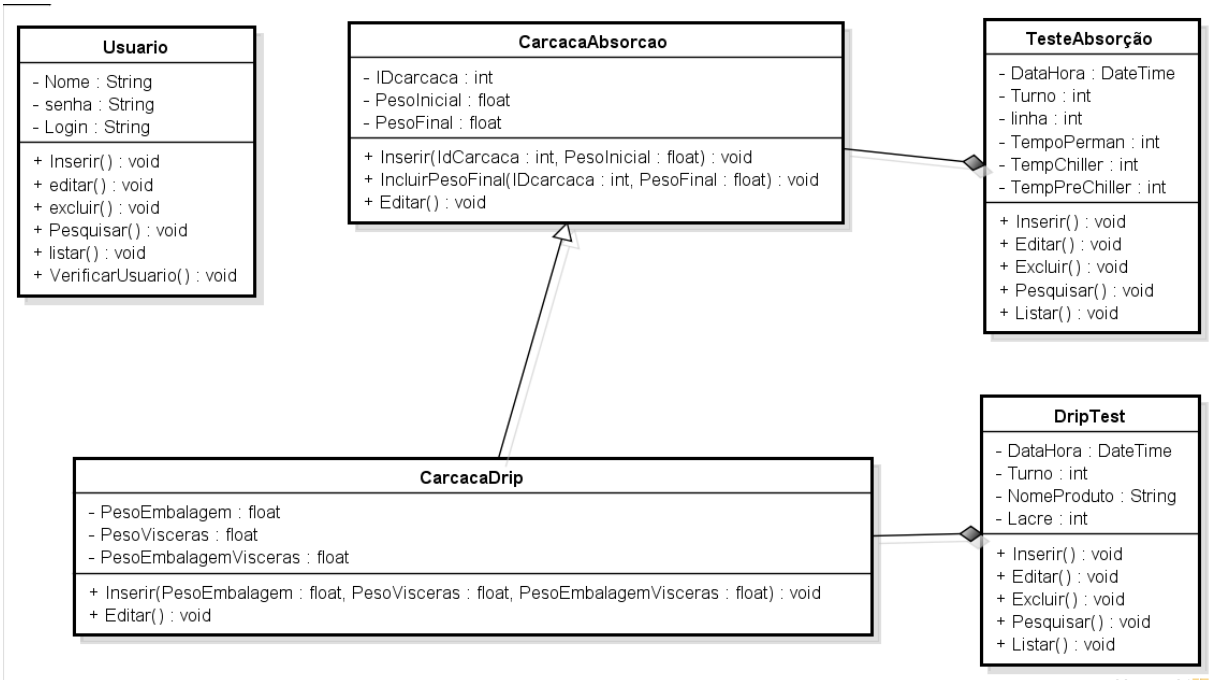

Figura 15 – Diagrama de classes

Os objetos que compõem o sistema são organizados em cinco classes (conforme apresentado no diagrama da Figura 15). Uma classe **Usuario**, para cadastro dos usuários que tem acesso ao sistema. Uma classe **CarcacaAbsorcao** que é herdada pela classe **CarcacaDrip** que é composta por **DripTest**. Uma classe **TesteAbsorção** que é composta por **CarcacaAbsorcao** para realizar os teste de absorção.

Os Quadros 9 a 13 apresentam as tabelas do banco de dados. As tabelas e seus campos foram definidos com base no diagrama de classes da Figura 15.

Dado	**Tipo/tamanho**
IDUsuario(PK)	Integer
NomeUsuario	String[40]
Tipo	Char[1]
Senha	Integer
Endereço	String[50]
Telefone	String[20]

Quadro 9 – Tabela Cadastrar usuário

Dado	**Tipo/tamanho**
IDCarcacaCongelada(PK)	Integer
PesoCarcaca	Double
PesoEmbalagem	Double
PesoDescongelada	Double
PesoEmbalagemVisceras	Double
IDDrip(FK)	Integer

Quadro 10 – Tabela Cadastrar carcaça congelada

Dado	Tipo/tamanho
IDCarcaca(PK)	Integer
PesoInicial	Double
PesoFinal	Double
TestID(FK)	Integer

Quadro 11 – Tabela Cadastrar carcaça

Dado	Tipo/tamanho
IDTeste(PK)	Integer
Data	Date
Hora	DateTime
Turno	Integer
Linha	Integer
TemperaturaPre	Float
TemperaturaChiller	Float
TesteRealizado	Char[1]

Quadro 12 – Tabela Cadastrar teste de absorção

Dado	Tipo/tamanho
IDDrip(PK)	Integer
Data	Date
DataProducao	Date
Hora	DateTime
Turno	Integer
TurnoProdução	Integer
Lacre1	Integer
Lacre2	Integer
Lacre3	Integer
Observacao	Varchar[200]
AcaoFiscal	Varchar[300]
NomeProduto	Varchar[40]

Quadro 13 – Tabela Cadastrar drip teste

4.3 Descrição do Sistema

A descrição do sistema é apresentada por módulos. Primeiro é apresentado o módulo *desktop* e em seguida o módulo para dispositivos móveis.

4.3.1 Módulo *Desktop*

Para acessar o sistema o usuário precisa conectar-se por *login* que é realizado por um nome de usuário e uma senha previamente cadastrados na base de dados. Após logado o usuário terá acesso à interface principal do sistema que contém os seguintes menus: teste, relatório, cadastro e e-mail.

O **menu Teste** apresenta os seguintes itens de menus: **Teste de absorção** e **Drip teste**. O item de menu teste de absorção apresenta a tela para realização do teste de absorção (Figura 16). Na parte inferior dessa tela são listados os testes cadastrados, bem como as carcaças relacionadas ao teste selecionado. O usuário poderá selecionar o teste que deseja realizar e clicar no botão gerar teste para serem realizados os cálculos para verificar se o limite de água absorvida pelas carcaças ultrapassou o estabelecido na legislação.

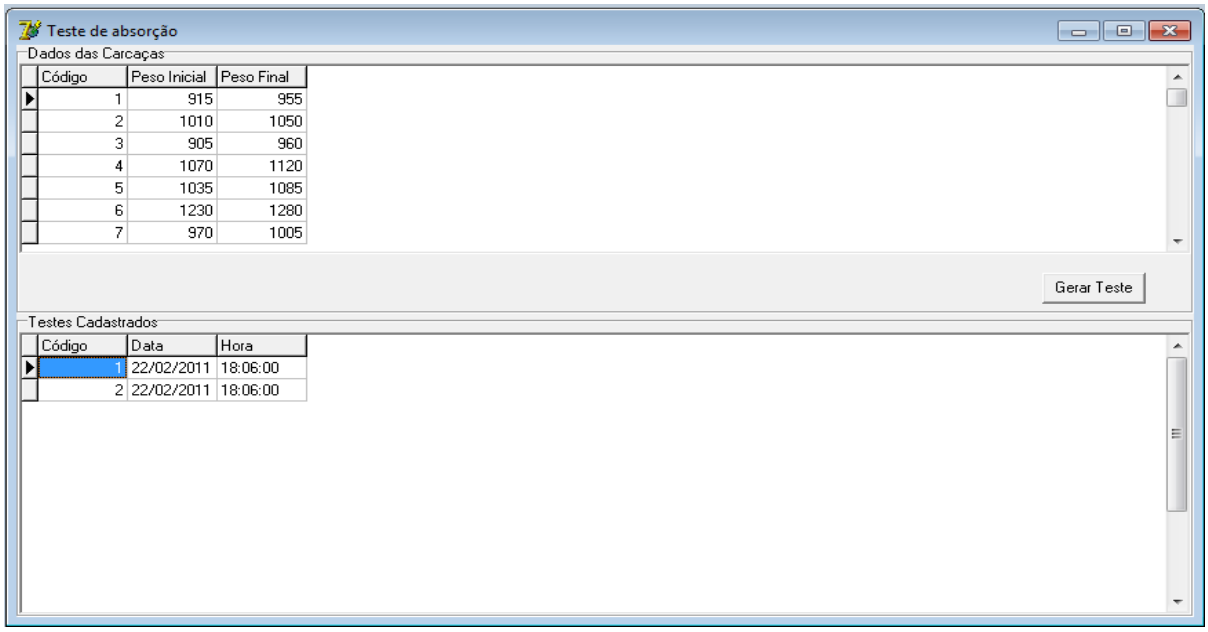

Figura 16 – Tela de teste de absorção

O item de **menu Drip teste** apresenta a tela para realização do Drip teste (Figura 17). Essa tela apresenta os testes já cadastrados e as carcaças relacionadas ao teste selecionado. Por meio do botão **Gerar Drip Teste** são realizados os cálculos para verificar se o limite de água absorvida no processo de pré-resfriamento ultrapassou o estabelecido na legislação. Caso ultrapasse, uma caixa de texto é apresentada na interface para que o usuário cadastre a ação fiscal e um campo para observação.

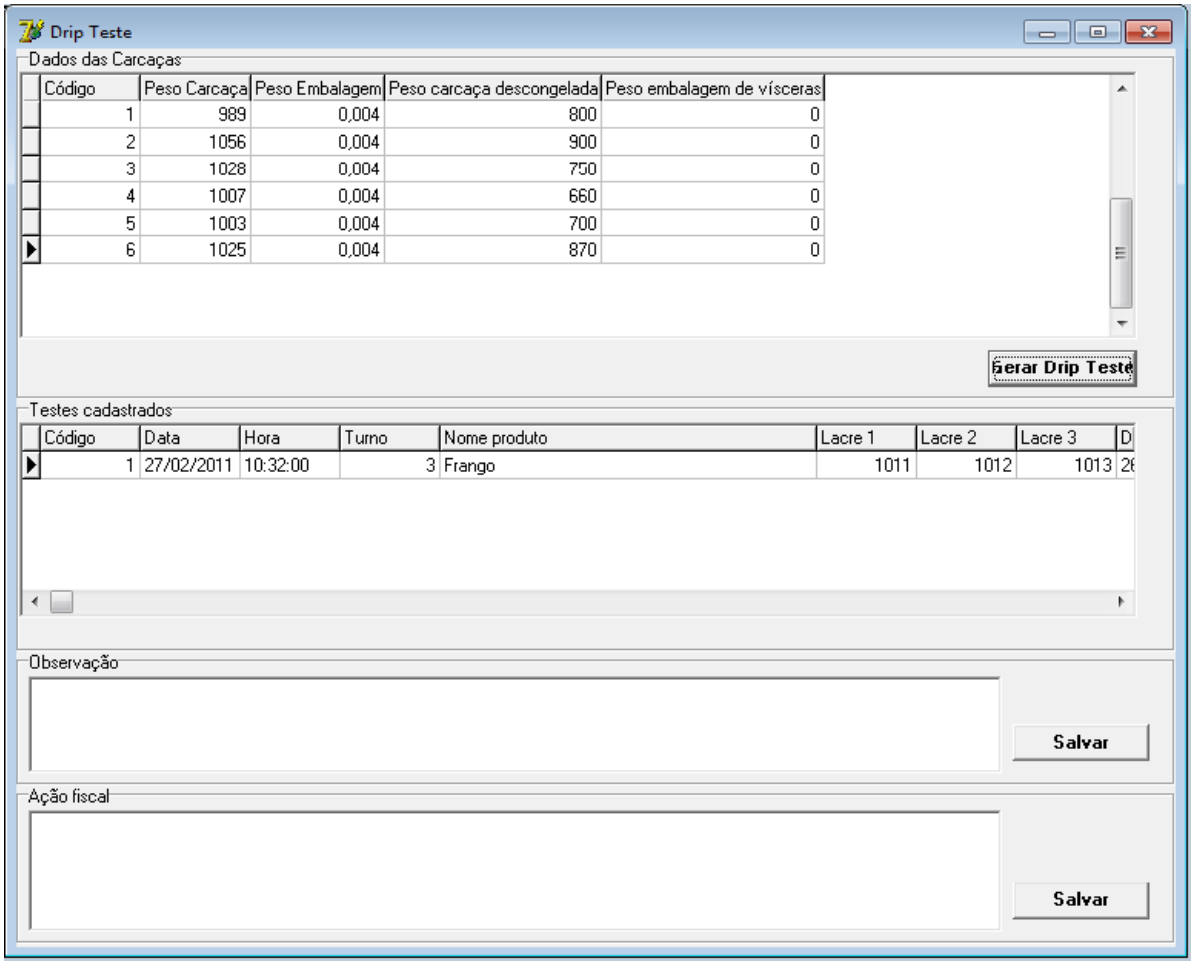

Figura 17 – Tela de drip teste

O menu relatório possui os seguintes itens de menu: **Absorção** e **Drip**. O item absorção do menu relatório apresenta a interface para seleção do relatório a ser gerado. Após selecionar o teste basta pressionar o botão **Gerar** Relatório para que o teste seja gerado. A interface da tela para gerar relatórios é descrita na Figura 18.

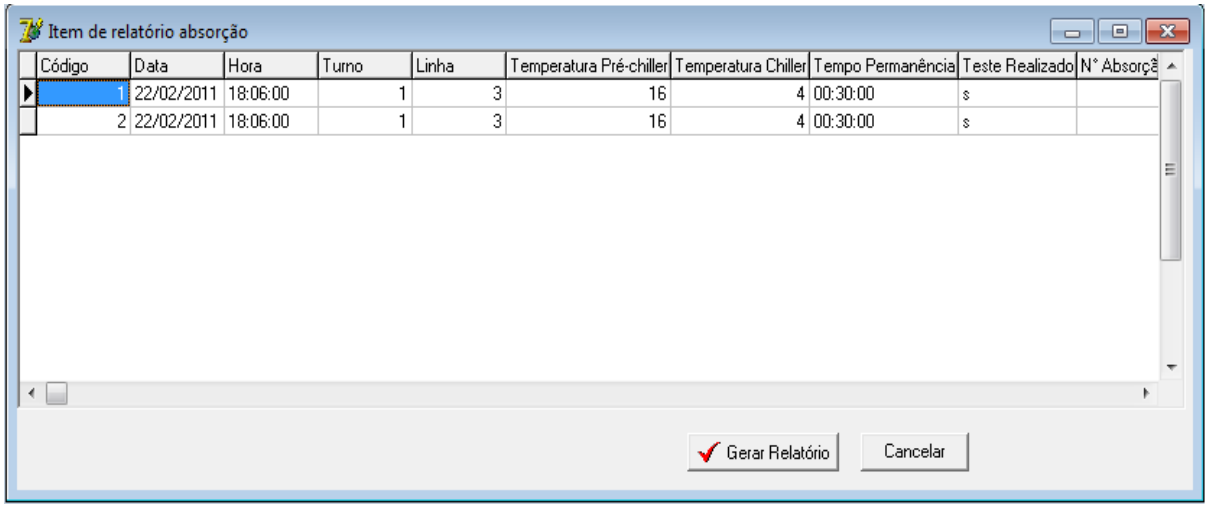

Figura 18 – Tela de relatório de absorção

O **item Drip** do menu relatório apresenta a interface da tela para gerar relatórios dos Drip testes gerados. Nesta tela o usuário poderá selecionar um **Drip teste** para o qual queira gerar o relatório. A Figura 19 apresenta a interface da tela para gerar relatórios do **Drip teste**.

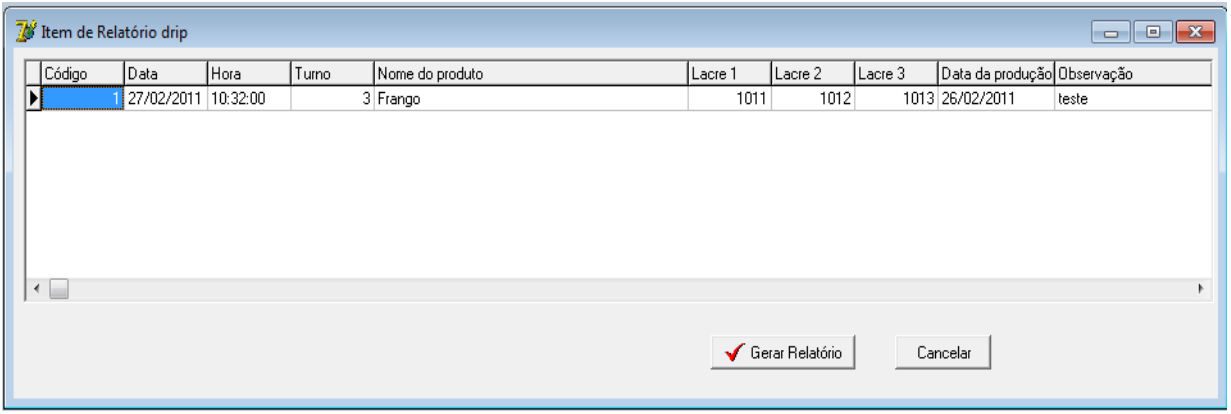

Figura 19 – Tela de relatório do drip teste

O menu **Cadastro** apresenta o item usuário (Figura 20) para cadastrar usuários do sistema que será executado em um dispositivo móvel. Nesta tela, o usuário poderá inserir, alterar e excluir cadastro. A interface possui um componente *grid* que apresenta todos os registros de usuários já armazenados.

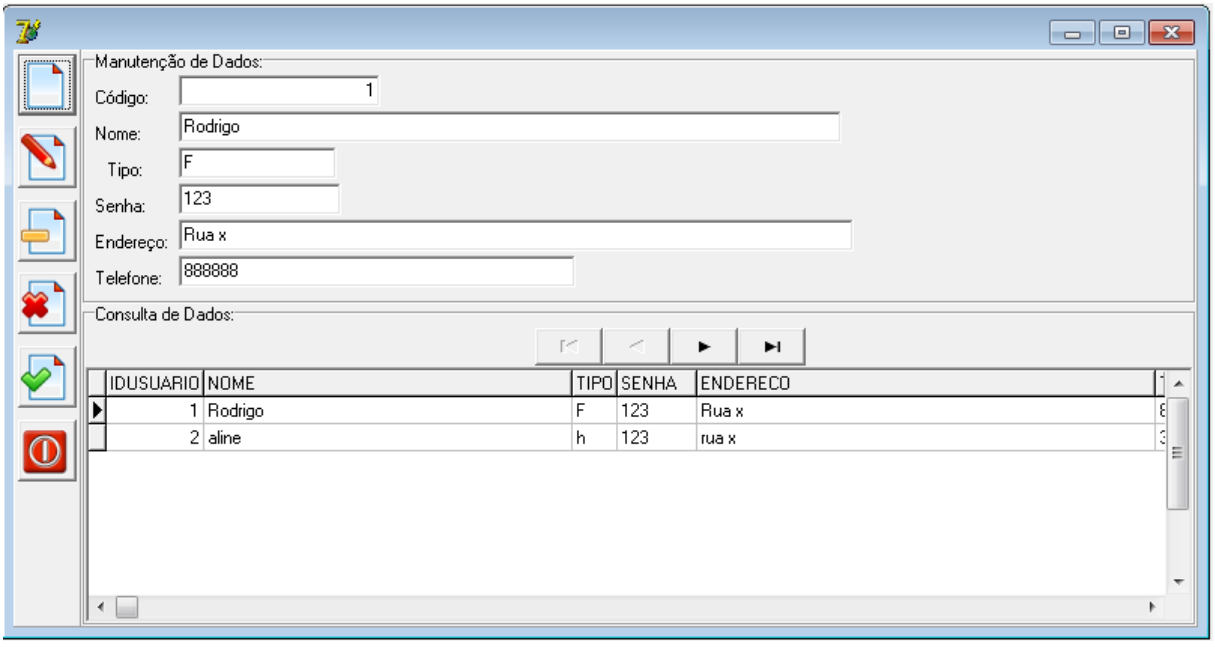

Figura 20 – Tela de cadastro de usuário

O sistema permite o envio de *email* com a possibilidade de anexar os relatórios gerados a partir dos testes. Também são preenchidos dados de origem e de destino bem como campos para assunto e a mensagem. Para enviar o *e-mail* o usuário terá que informar o *host*

bem como a porta de acesso, o usuário e a sua senha de acesso ao *e-mail*.

4.3.2 Módulo Dispositivo Móvel

A Figura 21 apresenta a interface principal do módulo para dispositivo móvel do sistema para controle de absorção de água em carcaças de aves. Nessa tela são apresentados os itens de menu: **Teste de Absorção**, **Drip Teste**, **Drip Teste Contra Prova** e **Descarregar**.

Figura 21 – Tela inicial do módulo para dispositivo móvel

Na Figura 21:

a) O item de menu **Teste de Absorção** direcionará para a funcionalidade de cadastro de carcaças para realização do teste de absorção.

b) O item de menu **Drip Teste** permitirá acesso à funcionalidade de cadastro de carcaças congeladas para a realização do **Drip Teste**.

c) O item de menu **Drip Teste Contra Prova** permitirá localizar um **Drip Teste** já cadastrado. Essa pesquisa pode ser utilizada para buscar os dados iniciais de um teste para a realização do teste contraprova.

d) O item de menu **Descarregar** é utilizado para exportar os dados dos testes de Absorção e Drip no banco de dados do módulo *desktop* da aplicação.

A Figura 22 apresenta a tela de **Opções de absorção**. Nesta tela o usuário terá acesso às opções: **Cadastrar** para cadastrar um teste de absorção, **Listar** para listar os testes já

cadastrados e **Voltar** o qual retorna para a interface principal do sistema.

Figura 22 – Tela de Opções de Absorção

A Figura 23 apresenta a tela de **Cadastro de Carcaças** que possui as seguintes opções de menu: **Carcaças Dados Iniciais** para cadastrar os dados inicias das doze carcaças coletadas, **Carcaça Dados Finais** (para cadastrar os dados finais das doze carcaças coletadas) e **Voltar** que retorna para a interface de Opções.

Figura 23 – Tela de Cadastro de carcaça

A Figura 24 apresenta a tela de **Teste de absorção – Dados iniciais**. Nesta interface o usuário informará os dados iniciais do teste de absorção, sendo todos de preenchimento obrigatório. Ao clicar na opção avançar o sistema validará se todos os campos foram preenchidos. Quando todos esses dados estiverem preenchidos, o usuário tem a opção de gravar os dados iniciais do teste em um banco intermediário. Esses dados serão complementados com os dados finais do teste.

Figura 24 – Telas de Teste de absorção – dados iniciais

A Figura 25 apresenta a tela de **Teste de absorção – Dados finais**. Nesta interface o usuário informará os dados finais do teste de absorção, sendo todos os campos de preenchimento obrigatório. A validação é feita ao clicar na opção avançar. Após a inserção dos dados o usuário terá a opção de inserir o teste no banco de dados RMS definitivo.

Figura 25 – Tela de Teste de absorção – Dados Finais

Para que os dados coletados no dispositivo móvel, que ficam armazenados em um banco de dados RMS, sejam exportados para o servidor, o usuário deverá informar um nome de usuário e sua respectiva senha. A exportação é realizada pela opção **Logar** do menu que inicia um método da classe do **Cliente WebService.** Esse método acessa um serviço *web* que valida o usuário e a senha.

4.4 Implementação do Sistema

A seguir estão exemplos dos principais códigos gerados na implementação do sistema, agrupados no módulo *desktop* (Seção 4.4.1) e módulo dispositivo móvel (Seção 4.4.2).

4.4.1 Módulo *Desktop*

A Listagem 1 apresenta a função que gera a média das carcaças utilizadas em um Drip teste, executando os procedimentos para verificar se a quantidade de água absorvida pelas carcaças ultrapassou os limites estabelecidos pela legislação. Inicialmente é feita a declaração da função no escopo privado do formulário **frmDripTeste** e para isso é utilizada a palavra reservada *Function* seguida do nome da função. E como funções retornam um parâmetro,

deve-se informar qual é o tipo de retorno, no caso esta retorna um valor do tipo *double*.

A função **validaDripTeste** utilizará dados que estão armazenados no banco de dados em uma tabela específica. Para isso utilizará um componente do Delphi que permite acessar os dados de uma tabela. O componente *TIBTable* está relacionado com a tabela **CarcacaCongelada** do bando de dados que sejam obtidos os dados necessários.

```
function TfrmDripTeste.ValidaDripTeste: Double;
var
  MO, M1, M2,M3, Media,Total : Double;
begin
  ibtDadosCarcDrip.First;
  Total := 0;
  while not ibtDadosCarcDrip.Eof do
    begin
      MO := ibtDadosCarcDrip.fieldByname('PESOCARCACA').AsFloat;
      M1 := ibtDadosCarcDrip.fieldByname('PESOEMBALAGEM').AsFloat;
      M2                                                              :=
ibtDadosCarcDrip.fieldByname('PESOCARCACADESCONGELADA').AsFloat;
      M3 := ibtDadosCarcDrip.fieldByname('PESOEMBALAGEMVISCERAS').AsFloat;

      Total := Total + ((MO - M1 - M2)/(MO - M1 - M3)*100);
      ibtDadosCarcDrip.Next;
    end;
  Media := Total/6;
  Result := Media;

  end;
```

Listagem 1 – Implementação da função TfrmDripTeste

Como mostra o código da Listagem 1, os dados que serão coletados do banco devem ser armazenados em variáveis locais e, portanto, elas são declaradas como *var*. Nesta função serão utilizadas seis variáveis do tipo **double** para armazenar dados provenientes do banco e outras para armazenar os resultados dos cálculos. Ao iniciar o procedimento para coletar dados no banco coloca-se o componente *TIBTable* no primeiro registro armazenado na tabela com a função **First** e posteriormente utiliza-se uma estrutura de repetição para percorrer toda a tabela iniciando do primeiro registro até o último registro. Para verificar se o registro corrente é o último, utiliza-se a função **Eof** (*End of File*). Enquanto a tabela é percorrida os dados são armazenados nas variáveis. O acesso aos campos do registro atual é realizado pela opção *FieldByname*, passando por parâmetro o campo da tabela desejado. Em seguida são executados os cálculos de verificação. Após calcular o total de todos os registros, a função retorna a média de absorção das carcaças, dividindo o total de absorção pelo número seis, que representa o total de carcaças padrão em um Drip teste.

A Listagem 2 apresenta o evento **onClick** do botão gerar **Drip teste**. Nesse evento é declarada uma variável que receberá o retorno da função **validarDripTeste** invocada no

início do procedimento. Se esse retorno é maior que 6%, o sistema apresentará uma mensagem informando que o limite de absorção ultrapassou o limite da legislação e será apresentada a quantidade absorvida. Posteriormente o sistema apresentará um componente de caixa de texto, um botão e um separador *GroupBox* para que o usuário informe a ação fiscal e a salve. Caso o percentual absorvido seja inferior a 6%, apenas será apresentada uma mensagem informativa e o percentual absorvido.

```
procedure TfrmDripTeste.btnGerarDriTestClick(Sender: TObject);
var
  per : Double;
  rel : string;
begin
  rel := ibtTestesCadDrip.fieldByname('testerealizado').AsString;
  per := ValidaDripTeste;
  if Per > 6 then
  begin
    MessageDlg('Atenção, o teste ultrapassou o limite estabelecido na
legislação  '+ FloatToStr(Per),
      mtWarning, [mbOK],0);
      if rel = '' then
      begin
        gbAcaofiscal.Visible := true;
       end
     end
   else
    MessageDlg('Atenção, O teste esta dentro dos limites estabelecidos pela
legislação  '+ FloatToStr(Per),
      mtInformation, [mbOK],0);
end;
```

Listagem 2 – Evento onClick do botão

A Listagem 3 contém o evento **onClick** do botão salvar ação fiscal. Nesse evento é verificado se o componente de caixa de texto **TMemo** está preenchido. Caso não esteja é apresentada uma mensagem informando que a ação fiscal deve ser preenchida e logo após será atribuído o foco ao componente de caixa de texto com a função **SetFocus**. Em seguida será informado o campo a ser preenchido por meio da opção **FieldByName** que posteriormente será salvo no banco de dados com a função *Post*. Após gravar a ação fiscal, o componente caixa de texto, o botão e o *groupbox* ficarão invisíveis (propriedade *visible* recebendo *false*) para que o usuário não faça mais nenhuma alteração na ação fiscal. Esse procedimento é executado dentro de uma clausula *try except* para tratamento de erro.

```
procedure TfrmDripTeste.btnObservacaoClick(Sender: TObject);
var
  teste : string;
begin
try
  begin
    teste := ibtTestesCadDrip.fieldByname('testerealizado').AsString;
      if teste = 's' then
```

```
      begin
        ShowMessage('A oberservação e a ação fiscal do referido teste ja
foram realizadas');
        menObservacao.Text := '';
      end
      else
      begin
        if menObservacao.Text  = '' then
          begin
            ShowMessage('Informe a obeservação!');
            menObservacao.SetFocus;
          end
          else
          begin
            ibtTestesCadDrip.Edit;
            ibtTestesCadDrip.FieldByName('observacao').AsString          :=
menObservacao.Text;
            ibtTestesCadDrip.FieldByName('testerealizado').AsString := 's';
            ibtTestesCadDrip.Post;
            ShowMessage('Observação gravada com sucesso!');
            menObservacao.Text := '';
          end;
            if menAcaoFiscal.Visible = true then
              begin
              if menAcaoFiscal.Text  = '' then
                begin
                  ShowMessage('Informe a acão fiscal!');
                  menAcaoFiscal.SetFocus;
                end
              else
          begin
            ibtTestesCadDrip.Edit;
            ibtTestesCadDrip.FieldByName('acaofiscal').AsString          :=
menAcaoFiscal.Text;
            ibtTestesCadDrip.Post;
            ShowMessage('Ação fiscal gravada com sucesso!');
            menAcaoFiscal.Text := '';
            menAcaoFiscal.Visible := false;
            gbAcaofiscal.Visible := False;
          end
        end;
        end;
        end;
    except
      on E: Exception do
      ShowMessage('Erro ao gravar o registro: ' + e.Message);
           end;
     end;
end.
```

Listagem 3 – Evento onClick do botão salvar ação fiscal

Na Listagem 3 também está representa o evento **onClick** do botão **Salvar Observação**. É verificado se o componente de caixa de texto **TMemo** está preenchido. Se não estiver é apresentada uma mensagem de texto informando que a observação deve ser preenchida e em seguida é atribuído o foco ao componente de caixa de texto com a função **SetFocus**. Se estiver preenchida a tabela **DripTeste** do banco, representada pelo componente

TIBTable, será editada com a função **Edit**. Em seguida será informado o campo alterado por meio da opção **FieldByName** e este receberá a observação que foi especificada no componente de caixa de texto que será salvo no banco de dados com a função **Post**. Esse procedimento é executado dentro de uma cláusula *try except* para tratamento de erro.

A Listagem 4 apresenta o evento **formClose** do formulário. Este procedimento fecha as tabelas **CarcacaCongelada** e **DripTeste** utilizando a função **close** e posteriormente libera da memória o formulário **frmDripTeste** com a ação **caFree** e atribuição do valor *nil* à variável **frmDripTeste**.

```
procedure TfrmDripTeste.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  ibtDadosCarcDrip.Close;
  ibtTestesCadDrip.Close;
  Action := caFree;
  frmDripTeste := nil;

end;
```

Listagem 4 – Evento formClose do formulário

A Listagem 5 apresenta o evento **formShow** do formulário. Este procedimento abre as tabelas **CarcacaCongelada** e **DripTeste** com a função **open**. Caso ocorra algum erro e não seja possível abri-las, o procedimento prossegue na cláusula *except* que é um tratamento de erro. E é exibida uma mensagem informando que não foi possível abrir as tabelas.

```
procedure TfrmDripTeste.FormShow(Sender: TObject);
begin
try
    ibtDadosCarcDrip.Open;
    ibtTestesCadDrip.Open;
  except
    on E: Exception do
      MessageDlg('Erro ao tentar abrir as tabelas'+E.Message, mtError,
        [mbOK], 0);
  end;

 end;
```

Listagem 5 – Evento formShow do formulário

4.4.2 Módulo Dispositivo Móvel

A Listagem 6 apresenta o método que faz as validações dos campos iniciais do cadastro do teste de Absorção. Nesse método, a variável **validaCampo3** é utilizada para fazer as validações de campos obrigatórios. A função **getString()** é utilizada para capturar o que o usuário informou no campo texto, a função **trim()** para retirar os espaços e a função **equals()** para verificar se o campo esta vazio. Caso o campo esteja vazio a variável do tipo *boolean* irá receber o valor *true*.

```
public void validar1() {

        boolean validaCampos3 = false;
        Display display = Display.getDisplay(this);

        if (getTfPesoInicial1().getString().trim().equals("")) {
            validaCampos3 = true;
        }
        if (getTfPesoInicial2().getString().trim().equals("")) {
            validaCampos3 = true;
        }
        if (getTfPesoInicial3().getString().trim().equals("")) {
            validaCampos3 = true;
        }
        if (getTfPesoInicial4().getString().trim().equals("")) {
            validaCampos3 = true;
        }

        if (validaCampos3) {
            Alert al = new Alert("Atenção", "Dados de preenchimento
obrigatório.", null, null);
            al.setTimeout(2000);
            display.setCurrent(al, frmDadosCarcacaIniciais);
            return;
        }
        psi.setPeso1(getTfPesoInicial1().getString());
        psi.setPeso2(getTfPesoInicial2().getString());
        psi.setPeso3(getTfPesoInicial3().getString());
        psi.setPeso4(getTfPesoInicial4().getString());

        switchDisplayable(null, getFrmDadosCarcaIniciais1());
    }
```

Listagem 6 – Método Validar1()

Ainda na Listagem 6, pós verificar todos os campos de texto, verifica-o valor da variável **validaCampo3**, se *true* (verdadeiro), então é apresentada uma mensagem para o usuário de que todos os campos são de preenchimento obrigatório. Se o valor dessa variável é *false* (falso), o sistema armazenará os dados que o usuário informou no objeto da classe **PesosAbs** com os métodos "*sets*", classe esta que possui atributos para armazenar os dados.

A Listagem 7 apresenta o método que faz as validações dos dados do teste absorção. O

método valida se os campos com os pesos iniciais das carcaças foram informados por procedimento igual ao exposto na Listagem 7. Na sequência é inicializada a cláusula de tratamento de erro (*try*) e declara-se um objeto da classe **ByteArrayOutputStream**. Essa classe cria um fluxo de saída de *bytes* que é utilizado para converter os dados em *array* de *bytes* e na sequência é feita a declaração de um objeto da classe **DataOutputStream** que é vinculada com a classe **ByteArrayOutputStream** que adiciona os dados no registro. Posteriormente são utilizados dois métodos *gets* para obter os dados que foram informados pelo usuário que são gravados pelo método **writeUTF()** do objeto da classe **DataOutputStream**. Em seguida declara-se um *array* de *bytes* que é adicionado no objeto da classe **RecordStore** que organiza os registros e é responsável pela persistência de dados.

```
public void validar3() {
        boolean validaCampos1 = false;
        Display display = Display.getDisplay(this);

        if (getTfPesocarcaca9().getString().trim().equals("")) {
            validaCampos1 = true;
        }
        if (getTfpesoCarcaca10().getString().trim().equals("")) {
            validaCampos1 = true;
        }
        if (getTfPesoCarcaca11().getString().trim().equals("")) {
            validaCampos1 = true;
        }
        if (getTfPesoCarcaca12().getString().trim().equals("")) {
            validaCampos1 = true;
        }

        if (validaCampos1) {
            Alert aler = new Alert("Atenção", "Dados de preenchimento
obrigatório.", null, null);
            aler.setTimeout(2000);
            display.setCurrent(aler, frmDadosCarcacaIniciais2);
            return;
        } else {

            psi.setPeso9(getTfPesocarcaca9().getString());
            psi.setPeso10(getTfpesoCarcaca10().getString());
            psi.setPeso11(getTfPesoCarcaca11().getString());
            psi.setPeso12(getTfPesoCarcaca12().getString());
            try {
                ByteArrayOutputStream bout = new ByteArrayOutputStream();
                DataOutputStream dout = new DataOutputStream(bout);

                dout.writeUTF(psi.getData());
                dout.writeUTF(psi.getTurno());
                dout.writeUTF(psi.getNumAbs());
                dout.writeUTF(psi.getHora());
                dout.writeUTF(psi.getLinha());
                dout.writeUTF(psi.getTempPreChiller());
                dout.writeUTF(psi.getTempChiller());
                dout.writeUTF(psi.getTempoPerm());
                dout.writeUTF(psi.getPeso1());
                dout.writeUTF(psi.getPeso2());
```

```
                dout.writeUTF(psi.getPeso3());
                dout.writeUTF(psi.getPeso4());
                dout.writeUTF(psi.getPeso5());
                dout.writeUTF(psi.getPeso6());
                dout.writeUTF(psi.getPeso7());
                dout.writeUTF(psi.getPeso8());
                dout.writeUTF(psi.getPeso9());
                dout.writeUTF(psi.getPeso10());
                dout.writeUTF(psi.getPeso11());
                dout.writeUTF(psi.getPeso12());

                byte[] registroPesos = bout.toByteArray();
                rs.addRecord(registroPesos, 0, registroPesos.length);
                limpaCamposAbsorcao();
            } catch (Exception ex) {
                System.out.println("Erro: " + ex.getMessage());
            }
            switchDisplayable(null, getLsCarcacaAbsorcao());
        }
    }
```

Listagem 7 – Método Validar3()

A Listagem 8 apresenta o método que efetua um filtro no **RecordStore** que armazena os testes de absorção. O filtro utiliza uma classe **RecordFilter()**. Para filtro são usados os campos data e hora, que diferenciam os testes entre si. No topo do método é instanciado o objeto da classe **PesosAbs()**, que irá armazenar os dados que estão armazenados no **RecordStore()** selecionado.

Na sequência é inicializada a cláusula de tratamento de exceção *try* e armazena-se em uma variável do tipo *string* a data e a hora do teste que está selecionado no componente *list*. A função **IndexOf()**, que retorna a posição de caractere em uma *string*, é usada para localizar o caractere "-" que é separa a data e a hora no componente *list*. Com a função **substring()** esse caractere é eliminado, permanecendo apenas a data e a hora a serem passadas no método construtor da classe que efetuará o filtro.

Posteriormente declara-se um objeto da classe **RecordEnumeration**, que armazena listas de registros. Para verificar se há registros na classe **RecordEnumeration** é utilizada a função **hasNextElement()** e para recuperar o *id* do registro é utilizada a função **nextRecordID()**. Com o *id* do registro são recuperados os dados que estão armazenados no **RecorStore**. Isso é feito pela função **getRecord()** do objeto da classe **RecordStore** passando por parâmetro o *id* recuperado.

Em seguida declara-se um objeto da classe **ByteArrayInputString** e em seu método construtor é passado o *array* de *bytes* do registro recuperado do **RecordStore**. Esse objeto da classe **ByteArrayInputString** possui um *buffer* interno que armazena os *arrays* de *bytes* e em conjunto com o objeto da classe **DataInputStream,** que recebe em seu método construtor o

objeto da classe **ByteArrayInputString,** é transforma os dados armazenados em *bytes* em caracteres alfanuméricos. Após serem recuperados todos os dados do respectivo teste eles são armazenados no objeto da classe **PesosAbs**. Em seguida a cláusula de tratamento de erro é terminada com o finalizador *catch* e caso ocorra alguma exceção é apresentado uma mensagem.

```
public void EditarAbs() {

        psf = new PesosAbs();
        int i;
        String ind;
        String filt;
        try {
            String                          dataHora                          =
getLsTestesDefinitivos().getString(getLsTestesDefinitivos().getSelectedInde
x());

            i = dataHora.indexOf("-");
            ind = dataHora.substring(i + 1, dataHora.length());
            filt = dataHora.substring(0, i);

            Filtrar filtro = new Filtrar(filt + ind);
            RecordEnumeration   re   =   rst.enumerateRecords(filtro,   null,
false);

            if (re.hasNextElement()) {

                int indice = re.nextRecordId();
                byte data[] = rst.getRecord(indice);

                ByteArrayInputStream bin = new ByteArrayInputStream(data);
                DataInputStream din = new DataInputStream(bin);

                psf.setData(din.readUTF());
                psf.setTurno(din.readUTF());
                psf.setNumAbs(din.readUTF());
                psf.setHora(din.readUTF());
                psf.setLinha(din.readUTF());
                psf.setTempPreChiller(din.readUTF());
                psf.setTempChiller(din.readUTF());
                psf.setTempoPerm(din.readUTF());
                psf.setPeso1(din.readUTF());
                psf.setPeso2(din.readUTF());
                psf.setPeso3(din.readUTF());
                psf.setPeso4(din.readUTF());
                psf.setPeso5(din.readUTF());
                psf.setPeso6(din.readUTF());
                psf.setPeso7(din.readUTF());
                psf.setPeso8(din.readUTF());
                psf.setPeso9(din.readUTF());
                psf.setPeso10(din.readUTF());
                psf.setPeso11(din.readUTF());
                psf.setPeso12(din.readUTF());
                psf.setPeso1f(din.readUTF());
                psf.setPeso2f(din.readUTF());
                psf.setPeso3f(din.readUTF());
                psf.setPeso4f(din.readUTF());
                psf.setPeso5f(din.readUTF());
```

```
                psf.setPeso6f(din.readUTF());
                psf.setPeso7f(din.readUTF());
                psf.setPeso8f(din.readUTF());
                psf.setPeso9f(din.readUTF());
                psf.setPeso10f(din.readUTF());
                psf.setPeso11f(din.readUTF());
                psf.setPeso12f(din.readUTF());

                getTfEdicaoTurnoAbs().setString(psf.getTurno());
                getTfEdicaoNumAbsorcao().setString(psf.getNumAbs());
                getTfEdicaoLinhaAbs().setString(psf.getLinha());

getTfEdicaoTemPreChiAbs().setString(psf.getTempPreChiller());
                getTfEdicaoTemChilAbs().setString(psf.getTempChiller());
                getTfEdicaoPermaAbs().setString(psf.getTempoPerm());
            }
        } catch (Exception ex) {
            System.out.println("Erro:" + ex.getMessage());
        }

        switchDisplayable(null, getFrmEditarAbs());
        // write post-action user code here
    }
```

Listagem 8 – EditarAbs

A Listagem 9 apresenta o método de validação do *login*. Este método tem um objeto da classe de cliente *web service*. Este objeto possui um método responsável pelo *login* e para isso requer dois parâmetros: usuário e senha. A partir desses dados, um serviço *web* para acesso ao banco de dados e a tabela que armazena os usuários cadastrados é acessado.

```
public void validaLogin() {
        Boolean resp = null;
        try {

            if(tfUsuario.getString().trim().equals("")                    ||
tfSenha.getString().trim().equals("")){
                Alert aler = new Alert("Atenção", "usuário e senha devem
ser informados.", null, null);
                aler.setTimeout(2000);
                getDisplay().setCurrent(aler, getFrmLoginContraProva());
                return;
            }
            resp         =         servCliente.login(tfUsuario.getString(),
tfSenha.getString());
        } catch (Exception ex) {
            System.out.println("Erro:" + ex.getMessage());
            Alert  aler  =  new  Alert("Atenção",  "O  servidor  não  esta
disponível.", null, null);
            aler.setTimeout(2000);
            getDisplay().setCurrent(aler, getFrmLoginContraProva());
            return;
        }
        if (resp.booleanValue()) {
            getTfUsuario().setString("");
            getTfSenha().setString("");
            switchDisplayable(null, getFrmSelect());
        } else {
```

```
            Alert aler = new Alert("Atenção", "Usuário inválido.", null,
null);
            aler.setTimeout(2000);
            getDisplay().setCurrent(aler, getFrmLoginContraProva());
            return;
        }
    }
```

Listagem 9 – Validar login

A Listagem 10 apresenta o serviço *web* responsável por validar o usuário e a senha informados no dispositivo móvel. O usuário e a senha são passados por parâmetro quando o usuário tentar acessar as funcionalidades do sistema para controle de absorção. Para validar se o usuário e a senha estão corretos, o sistema terá que fazer uma busca na tabela de usuários.

No topo do método é instanciado o objeto da classe **Connection** com *null*. Inicializa-se a cláusula de tratamento de erro *try* e carrega-se o *drive* do banco Firebird com a função **class.forname** passando por parâmetro uma *string* com o nome do *drive*. O objeto da classe conexão recebe uma instância de conexão a partir do caminho do banco, nome e senha passados por parâmetro na função **DriverManager.getConnection()**. Na sequência é utilizada a função **preparedStatement** do objeto da classe **Connection** para criar instruções SQL parametrizadas.

Para atribuir os parâmetros na cláusula SQL é usada a função **setstring** do objeto da classe **PreparedStatement** e em seguida é executada a cláusula SQL com a função **executequery**() do objeto da classe **PreparedStatement** e o resultado desta consulta é armazenado no objeto da classe **ResultSet**. Se houver retorno dessa consulta significa que o usuário e a senha informados são válidos.

```
@WebMethod(operationName = "login")
    public Boolean login(@WebParam(name = "usuario") String usuario,
@WebParam(name = "senha") String senha) {
        con = null;

        try {
            Class.forName("org.firebirdsql.jdbc.FBDriver");
            con                                                    =
DriverManager.getConnection("jdbc:firebirdsql:localhost/3050:D:/Delphi/proj
ects/Estagio/BancoEstagio.fdb",
                    "sysdba",
                    "masterkey");
            PreparedStatement stmt = con.prepareStatement("select * from
usuario where nome = ? and senha = ?");
            stmt.setString(1, usuario);
            stmt.setString(2, senha);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return true;

            } else {
                return false;
```

```
            }
        } catch (Exception ex) {
            System.out.println("Erro" + ex.getMessage());
            return false;
        }
    }
```

Listagem 10 – Servidor web - Validar login

As Listagens 11 e 12 apresentam o método do serviço *web* que faz a inserção do Drip Teste no banco de dados. Esse método recebe os dados do sistema em execução no dispositivo móvel. No início do método são declaradas duas variáveis que irão armazenar os *ids* das tabelas. Também é instanciado o objeto da classe conexão, este recebendo *null*. Na sequência é iniciada a cláusula de tratamento de erro *try* bem como é carregado *drive* do banco de dados Firebird com a função **class.ForName** passando o nome do *drive* por parâmetro. Em seguida, o objeto da classe **Connection** recebe uma instância de conexão com o banco a partir da função **DriverManager.getConnection** passando por parâmetro o diretório no qual está o banco de dados, bem como o nome da base de dados, a senha e o usuário.

```
@WebMethod(operationName = "inserirDripTeste")
    public Boolean inserirDripTeste(@WebParam(name = "data") String data,
@WebParam(name = "hora") String hora,
            @WebParam(name = "turno") int turno, @WebParam(name =
"nomeProduto") String nomeProduto,
            @WebParam(name = "lacre1") double lacre1, @WebParam(name =
"lacre2") double lacre2,
            @WebParam(name = "lacre3") double lacre3, @WebParam(name =
"dataProducao") String dataProducao,
            @WebParam(name = "turnoProducao") int turnoProducao,
@WebParam(name = "pesoCarcacaCong1") double pesoCarcacaCong1,
            @WebParam(name = "pesoEmbalagem1") double pesoEmbalagem1,
@WebParam(name = "pesoCarcacaDescong1") double pesoCarcacaDescong1,
            @WebParam(name = "pesoEmbalagemVisc1") double
pesoEmbalagemVisc1, @WebParam(name = "pesoCarcacaCong2") double
pesoCarcacaCong2,
            @WebParam(name = "pesoEmbalagem2") double pesoEmbalagem2,
@WebParam(name = "pesoCarcacaDescong2") double pesoCarcacaDescong2,
            @WebParam(name = "pesoEmbalagemVisc2") double
pesoEmbalagemVisc2, @WebParam(name = "pesoCarcacaCong3") double
pesoCarcacaCong3,
            @WebParam(name = "pesoEmbalagem3") double pesoEmbalagem3,
@WebParam(name = "pesoCarcacaDescong3") double pesoCarcacaDescong3,
            @WebParam(name = "pesoEmbalagemVisc3") double
pesoEmbalagemVisc3, @WebParam(name = "pesoCarcacaCong4") double
pesoCarcacaCong4,
            @WebParam(name = "pesoEmbalagem4") double pesoEmbalagem4,
@WebParam(name = "pesoCarcacaDescong4") double pesoCarcacaDescong4,
            @WebParam(name = "pesoEmbalagemVisc4") double
pesoEmbalagemVisc4, @WebParam(name = "pesoCarcacaCong5") double
pesoCarcacaCong5,
            @WebParam(name = "pesoEmbalagem5") double pesoEmbalagem5,
@WebParam(name = "pesocarcacaDescong5") double pesocarcacaDescong5,
```

```
            @WebParam(name = "pesoEmbalagemVisc5") double
pesoEmbalagemVisc5, @WebParam(name = "pesoCarcacaCong6") double
pesoCarcacaCong6,
            @WebParam(name = "pesoEmbalagem6") double pesoEmbalagem6,
@WebParam(name = "pesoCarcacaDescong6") double pesoCarcacaDescong6,
            @WebParam(name = "pesoEmbalagemVisc6") double
pesoEmbalagemVisc6) {

        int id = 0;
        int iddrip = 0;
        con = null;
        try {
            Class.forName("org.firebirdsql.jdbc.FBDriver");
            con =
DriverManager.getConnection("jdbc:firebirdsql:localhost/3050:D:/Delphi/proj
ects/Estagio/BancoEstagio.fdb",
                    "sysdba",
                    "masterkey");
```

Listagem 11 – Servidor web – Inserir drip teste (parte 1)

As variáveis que armazenarão os *ids* das tabelas são validadas com os *ids* que as funções **iddrip()** e **pegaidCarcaccadrip()** retornam e estas serão utilizadas na inserção dos dados nas tabelas. Em seguida é utilizado o objeto da classe conexão para fazer com que não se efetive a inserção dos dados até que todos os dados que foram enviados tenham sido inseridos com sucesso. Para isso é utilizada a função **setAutoCommit(False)**. Na sequência prepara-se uma instrução SQL com a função **PreparedStatement** do objeto da classe **Connection** que é armazenada no objeto da classe **PreparedStatement**. E com o objeto da classe **PreparedStatement** insere-se os dados na instrução SQL com as funções **setString()** e **setDouble**, dependendo do tipo de dado. Em seguida é executada a instrução SQL e armazena-se o retorno desta função em uma variável do tipo *int*.

```
  iddrip = iddrip();
            id = pegaidCarcaccadrip();

            con.setAutoCommit(false);
            PreparedStatement  stmt  =  con.prepareStatement("insert  into
dripteste (iddrip,data,hora,turno,nomeproduto,lacre1,lacre2,lacre3,"
                    +                          "dataproducao,turnoproducao)
values(?,?,?,?,?,?,?,?,?,?)");

            stmt.setInt(1, iddrip);
            stmt.setString(2, data);
            stmt.setString(3, hora);
            stmt.setInt(4, turno);
            stmt.setString(5, nomeProduto);
            stmt.setDouble(6, lacre1);
            stmt.setDouble(7, lacre2);
            stmt.setDouble(8, lacre3);
            stmt.setString(9, dataProducao);
            stmt.setInt(10, turnoProducao);
            int qtdAfetados1 = stmt.executeUpdate();
            PreparedStatement  stmt1  =  con.prepareStatement("insert  into
```

```
carcacacongelada (idcarcacacongelada,iddrip,pesocarcaca,pesoembalagem,"
                    +        "pesocarcacadescongelada,pesoembalagemvisceras)
values(?,?,?,?,?,?)");
            stmt1.setInt(1, id);
            stmt1.setInt(2, iddrip);
            stmt1.setDouble(3, pesoCarcacaCong1);
            stmt1.setDouble(4, pesoEmbalagem1);
            stmt1.setDouble(5, pesoCarcacaDescong1);
            stmt1.setDouble(6, pesoEmbalagemVisc1);
            int qtdAfetados2 = stmt1.executeUpdate();
```

Listagem 12 – Servidor web – Inserir drip teste (parte 2)

A Listagem 13 apresenta a efetivação da inserção dos dados do Drip teste. Inicialmente verifica-se com a cláusula *if* qual é o valor das variáveis do tipo inteiro. Caso alguma delas possua o valor zero significa que ocorreu algum erro na execução da instrução SQL, sendo assim é desfeita a efetivação utilizando-se a função **rollback()**. Caso contrário efetiva-se a operação com a função **commit()**.

```
         if (qtdAfetados1 == 0 || qtdAfetados2 == 0 || qtdAfetados3 == 0 ||
                    qtdAfetados4 == 0 || qtdAfetados5 == 0 || qtdAfetados6
== 0 || qtdAfetados7 == 0) {
                con.rollback();
                return false;
            } else {
                con.commit();
                return true;
            }
        } catch (Exception ex) {
            System.out.println("Erro:" + ex.getMessage());
        }
        return null;
    }
```

Listagem 13 – Servidor web – Inserir drip teste (parte 3)

5 CONCLUSÃO

O objetivo principal deste trabalho era automatizar determinadas atividades, especificamente relacionadas à verificação da quantidade de água retira por aves abatidas, em uma linha de produção de um abatedouro. Para facilitar o uso do sistema, o mesmo foi implementado em dois módulos: um deles um sistema *desktop* e outro para dispositivos móveis.

Considerando as várias atividades (ou etapas) do processo de abate de aves e o sistema ser implementado em plataformas e com tecnologias distintas, verificou-se para a modelagem do sistema poderiam ser utilizados conceitos de arquitetura de software. Conceitos de visões arquiteturais auxiliaram a definir um processo e neste foram identificadas as atividades que seriam informatizadas e para definir as tecnologias que seriam utilizadas na implementação de cada um dos módulos. Assim, foi mais fácil entender o escopo do problema e a solução definida.

Com o desenvolvimento do sistema resultado deste trabalho, verificou-se que um dos principais objetivos de definir a arquitetura de um sistema de software é alcançar atributos de qualidade de *software* estabelecidos para esse sistema. Isso porque as diferentes visões podem atender aos interesses dos diversos envolvidos no processo e essas visões auxiliam na comunicação entre esses envolvidos. A modelagem da arquitetura abrangeu o processo de cadastro dos dados que é executado em um dispositivo móvel, bem como o processamento dos dados que é realizado no módulo *desktop*.

REFERÊNCIAS

BASS, Len, CLEMENTS, Paul, KAZMAN, Rick. **Software architecture in practice,** Second Edition. Addison-Wesley Professional, 2003.

CAMPOS, Augusto. **A IDE para modelagem de dados JUDE**. Disponível em: <http://br-linux.org/linux/node/3335>. Acesso em: 25 out. 2011.

CANTU, Carlos H. **Get to know Firebird in 2 minutes (Conheça o Firebird em dois minutos)**, disponível em: <http://www.firebirdnews.org/docs/f>. Acesso: 26 fev. 2011.

CANTU, Marco. **Dominando o Delphi 6 "A Bíblia".** São Paulo. Makron Books, 2000.

CANTU, Marco. **Dominando o Delphi 7 "A Bíblia".** São Paulo: Makron Books, 2003.

CHEN Hai-Shan. **Survey on the style and description of software architecture**. 8th International Conference on Computer Supported Cooperative Work in Design Proceedings, p. 698-700, 2003.

GARLAN, David, SHAW, Mary. **An introduction to software architecture**. SEI Relatório Técnico CMU/SEI-94-TR-21, 1994.

GERMOGLIO, Guilherme **Fundamentos de arquitetura de software**. Disponível em <http://cnx.org/content/m17524/latest/#bid5>. Acesso: 10 ago. 2011.

IBEXPERT. **IBExpert dveloper sudio**. Disponível em: <http://www.ibexpert.com/>. Acesso: 14 jun. 2011.

IBM **Web Services Architecture Tam**. **Web services architecture overview. The next stage of evolution for ebusiness**, IBM Technical Document, Web Architecture Library, 2000

IEEE Standard Computer Dictionary: **Compilation of IEEE standard computer glossaries**. Institute of Electrical and Electronics Engineers Inc., 1991.

IEEE. INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Std 1471-2000: recommended practice for architecture description of software-intensive system**, IEEE Computer Society, 2000.

JACOBSON, Ivar, BOOCH, Grady, RUMBAUGH, James. **The unified software development process**. Addison Wesley, 1999, 463.

KRUCHTEN, Philippe. The 4+1 View Model of Architecture, **IEEE Software**, vol. 12, no. 6, 1995, p. 45-50.

KRUCHTEN, Philippe, CAPILLA, Rafael, DUEÑAS, Juan C. The decision view's role in software architecture practice. **IEEE Software**, p. 36-42, março/abril 2009.

PERRY, Dewayne E., WOLF, Alexander L. Foundations for the study of software architecture. **SIGSOFT Software Engineering Notes**, vol. 17, nun 4, p. 40-52, out. 1992.

THOMAS, Johnson P.; THOMAS, Mathews; GHINEA, George. **Modeling of web services flow**. In IEEE International Conference on E-Commerce (CEC 2003), 2003, p. 391-398.