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

RAFAEL ANTONIO BELOKUROWS

APLICATIVO MÓVEL PARA CONTROLE DE DADOS DE TIME DE FUTEBOL AMERICANO

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2011

RAFAEL ANTONIO BELOKUROWS

APLICATIVO MÓVEL PARA CONTROLE DE DADOS DE TIME DE FUTEBOL AMERICANO

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.

Orientador: **Prof. Tarlis Tortelli Portela**

PATO BRANCO
2011

ATA Nº: **194**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **RAFAEL ANTONIO BELOKUROWS**.

Às 14:00 hrs do dia 15 de dezembro de 2011, Bloco S da UTFPR, Campus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Tarlis Tortelli Portela (Orientador), Luís Carlos Ferreira Bueno (Convidado) e Silvio Luiz Bragatto Boss (Convidado), para avaliar o Trabalho de Diplomação do aluno Rafael Antonio Belokurows, matrícula 910058, sob o título **Aplicativo Móvel para Controle de Dados de Time de Futebol Americano**; 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 o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 15:00 hrs foi encerrada a sessão.

Tarlis T. Portela

Prof. Tarlis Tortelli Portela, Esp.
Orientador

Prof. Luís Carlos Ferreira Bueno, M.Sc.
Convidado

Silvio Luiz Bragatto Boss

Prof. Silvio Luiz Bragatto Boss, M.Sc.
Convidado

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

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

AGRADECIMENTOS

Agradeço primeiramente a Deus e minha família, que deram todo o apoio e às vezes o empurrãozinho necessário para fazer o trabalho. Também agradeço ao professor Tarlis Tortelli Portela, que me orientou, mesmo às vezes à distância, e que me encaminhou rumo à finalização desse trabalho. Além de todos os professores da UTFPR – Câmpus Pato Branco, mais especificamente do curso de Tecnologia em Análise e Desenvolvimento de Sistemas, que proporcionaram todas as oportunidades de aprendizado e que mostraram o caminho, muitas vezes árduo, da sabedoria.

RESUMO

Belokurows, Rafael. **Aplicativo móvel para controle de dados de time de futebol americano**. 2011. Monografia de Trabalho de Conclusão de Curso. Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas. Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2011.

Toda a competitividade que existe nos esportes profissionais nos dias de hoje está sendo vista nos esportes amadores, e assim como entre os profissionais, a organização, controle e análise bem feita de informações pode fazer a diferença entre a vitória e a derrota, o sucesso e o fracasso dos times. Dentre esses esportes se encontra o futebol americano, esporte pouco conhecido e menos ainda praticado no Brasil, mas que está crescendo de forma impressionante. Sendo assim, identificou-se a necessidade do desenvolvimento de um *software* para coleta e apresentação de dados referentes aos atletas de um time de futebol americano. Esse sistema foi desenvolvido utilizando a plataforma Java ME, que permite total integração com as funcionalidades dos celulares e *smartphones* mais atuais, tendo como único requisito o funcionamento da plataforma Java no aparelho. A análise foi feita utilizando o programa gratuito DIA, o desenvolvimento foi feito utilizando a IDE Netbeans, a gravação de dados foi feita em um banco de dados MySQL e posteriormente em registros RMS

Palavras-chave: Mobilidade. Futebol Americano. Cadastro e controle de atletas. Java. Análise Orientada a Objetos.

LISTA DE SIGLAS

AFAB	Associação de Futebol Americano do Brasil
API	Application Programming Interface
HTML	HyperText Markup Language
IDE	Integrated Development Environment
J2ME	Java Mobile Edition
JDBC	Java Database Connectivity
JDK	Java Development
JVM	Java Virtual Machine
JRE	Java Runtime Environment
KVM	Kilobyte Virtual Machine
OMG	Object Management Group
PDA	Personal Digital Assistant
PHP	Hypertext Preprocessor
POO	Programação Orientada a Objetos
RMS	Record Management System
RUP	Rational Unified Process
SMS	Short Message Service
SQL	Structured Query Language
UML	Unified Modeling Language
XML	Extensible Markup Language
XP	Xtreme Programming

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1. INTRODUÇÃO

Nos últimos anos, os participantes de esportes profissionais ganharam um aliado muito importante na busca pela perfeição e resultados melhores nas suas competições, a tecnologia. Os mais variados tipos de sistemas de análise de desempenho, criação de jogadas, formações e estratégias estão sendo usados por equipes profissionais tanto no Brasil como no mundo inteiro.

Para algumas dessas funções e usos esses sistemas já estão disponíveis, e o mais importante, acessíveis para praticantes de esportes amadores também. Na maior parte são sistemas criados pelos próprios praticantes do esporte sem nenhum fim lucrativo ou gerencial.

Além dos esportes mais tradicionais no hemisfério sul, como o futebol e suas variações, voleibol, basquetebol, entre outros, alguns esportes menos conhecidos e praticados principalmente pelos americanos estão se destacando no Brasil nos últimos tempos. Casos esses do beisebol e futebol americano, dois dos esportes mais populares nos Estados Unidos, e que no Brasil estão conquistando seus adeptos e fãs graças a presença cada vez maior de times praticantes em todas as regiões do país.

Para esses esportes se faz necessário um controle dos dados referentes aos treinamentos e jogos pelos treinadores. É proposto por esse trabalho auxiliar na organização e preparação de um time amador desse esporte que está em franco crescimento no Brasil: o futebol americano.

1.1 OBJETIVOS

Desenvolver um aplicativo móvel de coleta e apresentação de dados de atletas para um time de futebol americano amador.

1.1.1 Objetivo Geral

Desenvolver um aplicativo móvel, para uso em celulares e *smartphones* em geral, que auxilie na coleta e apresentação de dados de atletas.

1.1.2 Objetivos Específicos

- Coletar e analisar as informações e requisitos iniciais para desenvolvimento do *software*;
- Modelar e implementar o cadastro de atletas; telas para inserção de dados; e telas para listagem das informações coletadas;
- Desenvolver o aplicativo móvel em sincronia com as informações coletadas, composto pelas seguintes funções propostas:
    - Cadastro de atletas e atividades;
    - Coleta de dados referentes às atividades (estatísticas e desempenhos em treinos e jogos);
    - Módulo de visualização dos resultados;

1.2 JUSTIFICATIVA

Para os praticantes de esporte de fim de semana, já existem várias formas de controlar desempenho e estatísticas por meio de planilhas e anotações. Mas para o futebol americano, por ser um esporte menos conhecido e praticado, isso ainda é pouco encontrado.

Faz-se necessário um *software* específico para esse esporte, com o qual possa controlar dados sobre os treinamentos e jogos. Tal *software* traria grande auxílio a qualquer time quase profissional desse esporte, especialmente a um time iniciante. Assim, é mais fácil de traçar estratégias, definir qual jogador se adapta melhor em cada posição, definir as jogadas mais efetivas para cada força do time, dentre outras melhorias possíveis.

Através do aplicativo de coleta de dados e controle das informações referentes aos atletas, pretende-se extinguir a necessidade de planilhas e anotações em incontáveis folhas de papel. Com ele, o técnico poderá conhecer melhor seus atletas, saberá exatamente o desempenho e os dados físicos de cada um – peso, altura e tempo em exercícios relacionados a agilidade - e poderá levar pra dentro de campo esses dados, tendo acesso a eles diretamente da linha lateral em um jogo ou mesmo em um treino.

2. REFERENCIAL TEÓRICO

Uma das principais ações a ser tomada quanto ao desenvolvimento de um *software* é em relação ao modelo de desenvolvimento e arquitetura que ele irá utilizar. Existem vários tipos de modelos, e cada um poderia ser usado para qualquer projeto, mas existe um método mais correto para cada tipo de *software* que será desenvolvido.

Os principais processos de *software* definidos, e os mais utilizados são:

- Cascata
- Espiral
- Iterativo e Incremental
- Prototipagem
- RUP
- XP
- SCRUM

Para esse trabalho em particular, será utilizado o modelo em cascata, que mesmo já defasado, é o que apresenta a melhor relação tempo gasto por funcionalidade, pela simplicidade dos requisitos desse projeto e pela pequena quantidade de funções, decisões e implementações que serão feitas.

2.1 MODELO DE CICLO DE VIDA EM CASCATA

O modelo de ciclo de vida em Cascata foi o primeiro conhecido e é provavelmente o primeiro método organizado de desenvolvimento de *software*. Até o dia de hoje é o modelo mais utilizado, pela sua facilidade e abordagem sistemática, que não requer muito conhecimento da área de Engenharia de *Software* para implementar.

Ele é um modelo sequencial no qual o desenvolvimento do *software* sempre segue um fluxo pré-definido para frente – por isso a alusão à cascata no nome. Todas as fases desse ciclo são executadas em sequência, e as fases anteriores podem ser revisitadas para correções de erros ou possíveis adaptações necessárias.

É um método indicado para projetos nos quais há domínio dos requisitos do sistema que será desenvolvido e quando o pessoal envolvido no projeto é fraco tecnicamente, devido a baixa complexidade do modelo. Além disso, pode ser empregado para situações nas quais há um bom conhecimento do *software* que será desenvolvido e das tecnologias que serão utilizadas.

Como vantagem dessa abordagem mais direta e sequencial, caso a análise dos requisitos do programa a ser desenvolvido seja boa e, ou, seja uma aplicação simples de desenvolver, não haverá problemas futuros com esse projeto. Também oferece maior previsibilidade de prazos e custo. Além disso, a duração do projeto pode ser consideravelmente menor.

Por ser um método demasiadamente linear, ele não é adequado a projetos grandes ou projetos que podem ter seus requisitos várias vezes alterados. Possui pouca flexibilidade quanto a essas alterações e mudanças na metodologia e nas ferramentas utilizadas, como no caso de adequações a novas tecnologias, pois essa possibilidade não foi prevista no início do projeto.

Esse Modelo foi primeiramente definido por Winston W. Royce, nos anos 70, que havia proposto um método com sete fases bem distintas:

- Especificação de Requisitos
- Projeto
- Construção
- Integração
- Teste e depuração
- Instalação
- Manutenção

Nesse método original de Royce, cada fase deveria ser terminada para posteriormente ser iniciada a fase posterior, mas hoje existem várias versões modificadas do modelo em cascata original, incluindo versões que permitem esse retorno à fase anterior para alterações nos requisitos (SOMMERVILLE, 2007, p. 44)

2.2 ORIENTAÇÃO A OBJETOS

A Programação Orientada a Objetos (POO), idealizada inicialmente por Alan Kay, criador da linguagem de programação *Smalltalk*, é a programação implementada pelo envio de mensagens a objetos. Ela tem como objetivo aproximar o mundo real do mundo virtual, e faz isso através de objetos, classes e mensagens. Apesar de algumas linguagens de programação trabalharem especifica e mais abertamente com a POO, como Java, Python, Pascal e Ruby, pode-se desenvolver

um programa razoavelmente orientado a objetos mesmo em uma linguagem que seria tipicamente uma linguagem estruturada, e vice-versa. (DAVID, 2007)

Objetos, na POO, são representações de um objeto que existe no mundo real. São entidades que possuem uma identidade. Essa identidade quer dizer que podem existir vários objetos, mas cada objeto será único para o sistema e para a função que está executando. Os objetos podem ser distribuídos ou não, e podem ser executados em paralelo ou simultaneamente.

Os objetos se comunicam com outros objetos através de mensagens. Cada um deles irá responder às mensagens conhecidas por este, e assim poderá enviar mensagens a outros, para que sejam atendidas, de maneira que ao final do programa, seja atingido o objetivo. Nesse tipo de programação, o papel principal do programador é apontar quais serão as mensagens que cada objeto receberá, e também qual a ação que aquele objeto deve realizar ao receber aquela mensagem.

Segundo definição do Dicionário Aurélio da Língua Portuguesa (1986):
“Objeto. 1. Tudo que é apreendido pelo conhecimento, que não é o sujeito do conhecimento. 2. Tudo que é manipulável e/ou manufaturável. 3. Tudo que é perceptível por qualquer dos sentidos [...]”

Portanto, é uma definição que se encaixa tanto para um objeto palpável, do mundo real, como para uma representação de objeto dentro do sistema.

Os objetos geralmente possuem três grandes características:

- Estado: É uma das possíveis condições em que esse objeto existe, exemplo, o valor de uma variável.
- Comportamento: Como esse objeto responderá a mensagens e eventos, ou seja, que valores ele poderá assumir.
- Identidade: Alguma coisa que o torne único nesse sistema, como um nome, número ou outro identificador único.

Além do objeto, outro elemento importantíssimo na Programação Orientada a Objetos é a classe. Classe é o conjunto de dados estruturados que são caracterizados por propriedades comuns. Segundo o Dicionário Aurélio da Língua Portuguesa (1986), classe é: “Classe. 1. Numa série ou num conjunto, grupo ou divisão que apresenta características semelhantes; categoria, ordem. [...]”. Ou seja, classe é um conjunto de objetos que compartilham as mesmas operações, ou

objetos parecidos, ou que possuem a mesma funcionalidade, agrupados para proporcionar uma melhor visualização, entendimento e funcionamento do programa.

2.3 ARQUITETURA CLIENTE-SERVIDOR

A arquitetura cliente-servidor é o modelo computacional no qual duas ou mais máquinas se comunicam em rede compartilhando funções, métodos e informações através de mensagens. Nesse modelo sempre existe uma máquina servidor, máquina geralmente mais potente em relação aos recursos, e que provê o serviço, e a máquina cliente, a qual faz a requisição desse serviço, buscando uma determinada informação. (CREATIVE COMMONS..., 2009)

O cliente solicita uma tarefa ao servidor, que processa a requisição, e baseado no seu código, devolve a informação para o cliente, que utiliza essa informação do jeito para o qual foi programado. Um exemplo muito utilizado de aplicação cliente/servidor são os aplicativos de transferência de arquivos por FTP. O cliente manda uma requisição para o servidor de arquivos com seu IP e porta, o qual retorna uma mensagem permitindo ou não a conexão e *download* do arquivo para o aplicativo cliente.

Algumas das vantagens desse modelo:

- Centralização de recursos: Os métodos, funções e informações a serem necessárias se encontram todos (ou quase todos) no servidor, o que concentra os processamentos no servidor, que geralmente é uma máquina com mais recursos.
- Segurança e rapidez no acesso aos dados: Como o banco de dados é centralizado, não acontece redundância de dados e as informações estão disponíveis de forma on-line no momento de sua alteração. Além disso, geralmente um servidor tem controles de segurança maiores do que uma máquina cliente comum.
- Portabilidade: Aplicativos cliente programados de forma diferente podem acessar os mesmos dados, pois os métodos do servidor são acessados através de mensagens curtas e independentes.

Ele também possui algumas desvantagens:

- Dependência do servidor: Como as funcionalidades do aplicativo devem ser acessadas no servidor, caso ele não estiver disponível, os dados não poderão ser acessados.

- Custo e manutenção do servidor: O fato de o funcionamento do aplicativo ser tão dependente do servidor acarreta num custo maior, para manter o servidor funcionando e disponível 100% do tempo.

2.4 ANÁLISE ORIENTADA A OBJETOS

Diferentemente do enfoque tradicional de análise, no qual um sistema é formado por um conjunto de programas que executam processos sobre dados, na análise orientada a objetos, o sistema é considerado como uma coletânea de objetos que interagem entre si, com características próprias e atributos, como já explicado anteriormente.

2.4.1 Concepção

A fase de concepção é a primeira fase da análise, é nela que serão obtidas as primeiras informações do sistema, ou seja, é criada uma visão geral do sistema. O início da concepção é a ideia do sistema e o final é o enunciado do problema. (BLAHA e RUMBAUGH, 2006).

Para a solução proposta pelo *software* desenvolvido ser uma solução satisfatória para a pessoa ou empresa que for usá-lo, se faz necessária uma boa comunicação entre todos os envolvidos nesse projeto. O contratante do *software* já tem uma ideia pronta de como quer que tudo funcione, portanto parte do trabalho da *software house* é capturar a essência dessa solução que o cliente deseja e transformá-la, se possível, em um projeto concreto.

Neste ponto é que a concepção de *software* é importante. A equipe que projetará o *software*, através de reuniões iniciais, listas de eventos, diagramas e o levantamento de requisitos, deve levar em conta todas as exigências do cliente, e dentro do necessário, colocá-las no projeto do *software*. Após todos esses passos, pode ser visualizada uma estrutura interna, e como os componentes do sistema se interligarão pra fazer funcionar o sistema como um todo.

Além disso, os desenvolvedores terão que interpretar de forma satisfatória o que for definido pelos analistas de requisitos e projetistas, para trazer a solução do jeito mais parecido possível com aquilo que o cliente necessitava.

2.4.2 Linguagem UML

A Linguagem de Modelagem Unificada, a UML (*Unified Modeling Language*) é uma linguagem visual para especificação, construção e documentação de artefatos de *software*. Ela não é um método ou processo de desenvolvimento, portanto ela precisa ser utilizada em conjunto com um processo coordenado de desenvolvimento para ter sucesso. Segundo a OMG (*Object Management Group*), um consórcio internacional de empresas que define e ratifica padrões na área de Orientação a Objetos, a UML pode ser considerada como um padrão para modelagem de aplicações orientadas a objetos.

A UML tem como função facilitar a visualização de todos os processos e objetos que comporão esse *software*, permitindo-se dividir em várias partes e com isso, visualizar os relacionamentos entre os componentes de forma a antever o produto final. Por ser uma linguagem independente de qualquer linguagem de programação e método, com a UML pode-se fazer um esboço livre do projeto, sem os vícios de alguma linguagem específica e deixando de forma apresentável a estrutura do projeto para qualquer um dos envolvidos entender. (BOOCH; HARBAUGH; JACOBSON, 2006)

Existem vários tipos de diagramas que podem ser utilizados quando trabalhado com a UML. Alguns dos principais são:

- Diagrama de Classes: É o diagrama onde se listam todas as informações sobre os métodos, atributos e funções das classes do sistema que será desenvolvido.
- Diagrama de Contexto: Nesse diagrama são relacionados os eventos identificados no modelo anterior dentro do contexto completo do sistema, definindo os limites de escopo do mesmo e dizendo o que ele fará e com quem essa ação será relacionada. Esse diagrama não mostra de maneira ampla os recursos, mas sim de forma sucinta, apenas para uma visualização do todo do sistema.
- Diagrama de Caso de Uso: O diagrama de casos de uso é um dos diagramas disponíveis na linguagem UML para a modelagem de aspectos dinâmicos de sistemas. Deve ser um diagrama sem muitos dados técnicos, fácil de ler e ser entendido por todos envolvidos no projeto e mesmo pelos não envolvidos.

> “(os diagramas de caso de uso)... têm um papel central para a modelagem de comportamento de um sistema, de um subsistema ou de uma classe. São importantes para visualizar, especificar e documentar o comportamento de um elemento. Esses diagramas fazem com que sistemas, subsistemas e classes fiquem acessíveis e compreensíveis, por apresentarem uma visão externa sobre como esses elementos podem ser utilizados no contexto” (BOOCH;HARBAUGH; JACOBSON, 2006, p. 127).

- Diagrama de Fluxo de Dados (DFD): O DFD (Diagrama de Fluxo de Dados) é uma das ferramentas mais utilizadas na modelagem de sistemas, que permite ao analista visionar o sistema como uma rede de processos funcionais, interligados por dutos e tanques de armazenamento de dados. Os principais componentes de um DFD são os processos, fluxos de dados, depósito e o terminador (entidade externa). Geralmente é um diagrama muito simples, com indicações claras de para onde os dados estão indo e quem são os agentes e terminadores desses processos. (ROBERTO, [s.a.])
- Diagrama de Entidades e Relacionamentos (DER): O DER é um modelo utilizado para descrever a estruturação dos dados que serão armazenados e manipulados pelo sistema. É um modelo a nível conceitual que não traz nenhuma informação sobre a implementação, nem sobre as funções em si, mas sim apenas sobre os dados que serão utilizados. (PRESSMAN, 2005)

2.5 IMPLEMENTAÇÃO

Assim como em qualquer desenvolvimento de *software*, algumas boas práticas são necessárias na implementação desse projeto. Foram-se os dias em que um *software* que ganhe o mercado ou apresente uma solução razoável era desenvolvido sem um projeto adequado e definições dentro das metodologias. O projeto desse *software* deve ser utilizado como um norte para o desenvolvimento.

Muito provavelmente, durante seu processo de implementação e testes esse *software* irá passar pela mão de vários profissionais, tanto pra agilizar os testes, como pra deixá-los mais fidedignos. E mesmo se for um projeto de um homem só, é necessário uma boa documentação, estabelecimento de padrões de variáveis, criação organizada de classes e objetos, escrita de código adequada para melhor leitura do mesmo e uso de nomes significativos para todas as entidades. Tudo isso

ajudará tanto na implementação desse código quanto numa possível futura reutilização. (SCHACH, 2009)

2.6 TESTES

Parte importantíssima de qualquer projeto e não necessariamente apenas de um projeto de desenvolvimento de *software* é a fase de testes. Uma vez implementado o código dessa aplicação, o mesmo deve ser testado para descobrir possíveis problemas antes da entrega do produto de *software* ao seu cliente. Pela própria natureza do projeto de *software* permitir múltiplas tentativas e erros, ao contrário de outros tipos de projetos, os testes devem ser executados à exaustão para evitar erros quando chegar à compilação definitiva do produto. (SOMMERVILLE, 2007)

Os testes geralmente são intercalados com a fase de implementação, para, caso for encontrado algum erro, o desenvolvedor entrar em ação, e corrigir as falhas. Além de erros que impedem a completa utilização desse *software*, os testes podem apontar também alguma falha no projeto ou nas análises, posteriormente tornando esse programa lento, ou de utilização difícil pelo usuário final.

Segundo Sommerville (2007, p. 53), existem três estágios no processo de teste de *software*:

- Teste de componente: são testados os componentes (ou módulos) do sistema independentemente uns dos outros, apenas para garantir o funcionamento de cada um;
- Teste de sistema: nessa fase os componentes são interligados e o sistema é testado como um todo, para encontrar falhas tanto em interações previstas como interações não previstas entre cada parte do programa;
- Teste de aceitação: é o estágio final, aonde o *software* é utilizado em condições similares a como será usado pelo cliente final. Inclusive são utilizados dados fornecidos pelo próprio cliente em vez de dados simulados. A utilização de dados reais pode acarretar na descoberta de alguma falha no projeto, pois situações não previstas na análise de requisitos ou alguma fase posterior poderão aparecer;

Os testes devem ser bem planejados antes de serem realizados. Todas as fases dos testes devem ser guiadas por um plano de teste e incluir os objetivos de

cada fase, como eles serão executados e quais serão os critérios utilizados para determinar se o resultado foi encontrado, e se o *software* está de fato pronto para ser distribuído e, ou, utilizado pelo seu contratante. Esse plano de testes pode ser posteriormente alterado, visando a adicionar novas situações e especificar alterações que possam ter sido feitas pelos desenvolvedores.

Para esse sistema, especificamente, são utilizados dados reais para os testes, ou seja, dados coletados sobre um time real de futebol americano. A utilização desses dados traz todas as situações que esse *o* usuário desse *software* irá encontrar e permite uma simulação convincente, pois pode acarretar nos mesmos problemas que o usuário encontrará quando utilizar o sistema.

2.7 FUTEBOL AMERICANO

O futebol americano é um esporte coletivo derivado do rugby, que teve origem nos Estados Unidos no início do século XX, e foi trazido e adaptado por ingleses, que o diferenciaram em alguns aspectos e o chamaram de *football*, pois no início se chutava mais a bola do que se carregava ela nas mãos.

É hoje o esporte mais popular dentre os esportes profissionais praticados nos Estados Unidos, mas não tão popular fora desse país. A sua final, o *Superbowl,* que acontece na primeira semana de fevereiro, é transmitida para mais de 150 países e tem a maior audiência de um evento único da TV mundial ano a ano.

No Brasil, o esporte vem crescendo rapidamente. Começou a ser praticado no Brasil em meados da década de 90, principalmente no centro-sul, regiões mais desenvolvidas e nas quais a cultura americana está um pouco mais presente. Alguns dos times das capitais inclusive, são patrocinados por times de futebol, como o Coritiba Crocodiles, Fluminense Imperadores e Corinthians Steamrollers. Hoje, no país, são praticadas três modalidades do esporte:

- *Tackle*: a mais disputada; nela os jogadores utilizam todo o equipamento necessário, como as ombreiras, capacetes e protetores para as pernas. É jogado conforme as regras da liga profissional americana.

- *Flag*: é o futebol americano sem contato, em vez de pancadas e empurrões, deve ser puxada uma fita que fica na cintura de cada jogador para acabar a jogada.
- *No-pads*: semelhante ao *tackle***,** mas sem equipamentos; portanto é mais restrita e as regras são mais protetoras. O contato físico não é o mesmo que o profissional, mas é maior que o da modalidade *flag.*

Hoje são disputados dois campeonatos nacionais anuais, o Torneio Touchdown e a Liga Brasileira, os dois organizados pela AFAB (Associação de Futebol Americano do Brasil), a entidade máxima do esporte no país. Além disso, já existem 8 campeonatos estaduais, campeonato de seleções estaduais e campeonato de futebol americano feminino.

Figura 1. Estádio da liga colegial de futebol americano
Fonte: Landonhowell

O campo oficial, como demonstrado na Figura 01, tem 100 jardas de comprimento (em torno de 91 metros) mais as duas zonas finais (*end zones*) que possuem 10 jardas cada (9,1 metros). (ALMEIDA, 2006)

Ao final de cada lado do campo, se encontram as traves em formato de Y usadas como alvo para os chutes, como pode se ver na Figura 02.

Figura 2. Trave em formato de Y

Fonte: Dreamstime: Goal posts on American football field

Os times de futebol americano são formados normalmente por 53 jogadores, sendo 11 titulares de defesa, 11 de ataque e mais os especialistas, que entram para os chutes de precisão e de longa distância. Em cada momento do jogo, apenas 11 de cada equipe estão em campo, pois quando o ataque de um time está tentando conduzir a bola, a defesa do adversário os tenta parar, e vice-versa.

Daí vem a necessidade de se ter muitos jogadores no elenco, pois além de cada jogador apenas cumprir a função específica de sua posição, o futebol americano é um esporte de explosão, e como os jogos duram as vezes mais de três horas, a maioria dos jogadores tem pelo menos alguns momento de descanso.

Ao contrário da visão popular de quem não conhece o esporte, o futebol americano é um esporte muito inteligente, que exige muita estratégia, táticas e esquemas e grande conhecimento das regras pelos seus participantes. O objetivo principal desse esporte é a conquista de território, pois quanto mais perto da zona final do campo do adversário, maior a probabilidade de pontuar. Existem dois meios de pontuar: chutando a bola por entre a trave em forma de Y que se localiza no fim do campo, o chamado *field goal,* ou entrar na zona final do adversário com a bola nas mãos, o famoso *touchdown.* (ALMEIDA, 2006)

Além de toda a estratégia envolvida, outro aspecto importante do futebol americano é o confronto individual. Cada atleta, do time de ataque tem uma posição que o bloqueia ou que ele deve enfrentar durante o jogo do time da defesa, e vice-

versa, portanto, apesar de os times terem 11 jogadores de cada lado, no fim tudo se resume a confrontos individuais que acontecem paralelamente no campo.

Implícito ao confronto individual está o condicionamento físico dos atletas. Como a batalha dentro de campo é de um contra um, o atleta mais preparado física e psicologicamente tem a maior chance de vencer e criar uma boa jogada para seu time. Isso se torna ainda mais importante, pois o elenco de um time de futebol americano tem às vezes até 50 jogadores e como são apenas 22 que são os titulares – 11 no ataque e 11 na defesa - a tendência é sempre ter os melhores atletas dentro de campo.

Para reiterar a importância do aspecto físico, periodicamente são realizadas avaliações físicas dos atletas, tanto nos times profissionais como nos amadores, para avaliar a melhora – ou piora – no desempenho desses jogadores. Essas avaliações consistem de alguns exercícios simples, que trazem uma medição mais coerente e parecida com as situações encontradas no jogo. Alguns das medições utilizadas são:

40 Yard Dash: um percurso reto de 40 jardas (36 metros) em que são avaliadas a aceleração e velocidade máxima final do atleta.

20 Yard Shuttle: percurso lateral, na qual o atleta percorre 5 jardas para um lado, 10 para outro e 5 para o meio novamente, como demonstrado na Figura 03.

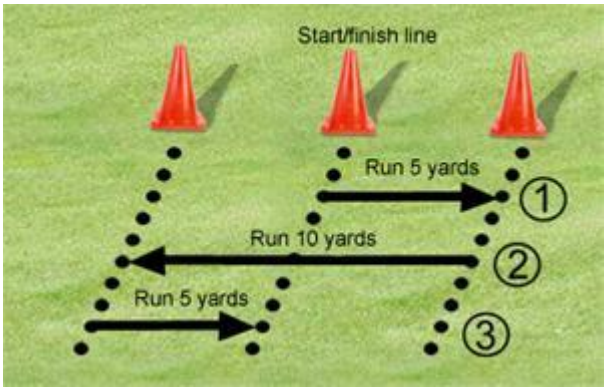

Figura 3. Percurso do 20 *yard shuttle*

Fonte: Fit For Football

3 cones: testa agilidade e mudança de direção dos jogadores com cortes de 90 e 180 graus, como demonstrado na Figura 04.

Figura 4. Percurso do exercício 3 cones
Fonte: The Three Cone Shuffle

Salto Horizontal: assim como representado na Figura 05, os atletas tentam saltar o máximo possível para frente com os pés parados no chão, sem propulsão anterior nenhuma. Mede a explosão física do atleta.

Figura 5. Atleta efetuando o salto horizontal
Fonte: Horizontal Jump

Todos esses exercícios são muito importantes na avaliação do condicionamento físico, explosão, agilidade e velocidade final do atleta, e num

esporte em que cada milésimo de segundo conta como o futebol americano, esses dados são muito importantes. (THE NFL SCOUTING COMBINE..., 2007)

3. MATERIAIS E MÉTODOS

Nesta seção são apresentados os materiais e métodos utilizados no presente trabalho.

3.1 MATERIAIS

Nos subitens sequentes são descritos os materiais utilizados no desenvolvimento do software.

3.1.1 Java

Java é a linguagem de programação, criada no ano de 1995, que tem como principal característica sua orientação a objetos. Com a utilização de classes e objetos, possibilita a divisão do programa em várias pequenas partes, o que facilita o desenvolvimento e posteriormente a manutenção e entendimento do código-fonte. É a tecnologia que permite o desenvolvimento de aplicações, em Java, que rodam em um sistema de propósito específico, como celulares, PDAs e outros dispositivos móveis (DEITEL, 2005).

A sua facilidade e praticidade para trabalhar, e a compatibilidade entre diferentes dispositivos, de todos os tipos, funcionalidades e tamanhos, são alguns dos motivos que mostram porque essa linguagem de programação é uma das mais usadas em todo mundo, e porque está alcançando praticamente todo tipo de equipamento hoje em dia e não só os microcomputadores. Hoje vários aparelhos de *Blu-Ray*, televisões e até eletrodomésticos tem funcionalidades que se utilizam dessas vantagens e dessa portabilidade. Desenvolvida pela empresa *Sun Microsystems*, foi originalmente utilizada para integrar circuitos de eletrodomésticos, mas acabou ganhando a Internet na década de 90, possibilitando a interatividade em páginas web que até então eram estáticas.

Pouco tempo depois, no final da década de 90, descobriu-se a facilidade dessa plataforma para trabalhar com dispositivos móveis, com isso, a linguagem Java tornou-se uma excelente plataforma para o desenvolvimento de aplicativos para celulares. Qualquer dispositivo móvel, computador ou eletroeletrônico que

execute a Máquina Virtual Java (*Java Virtual Machine*, ou JVM) está apto a rodar programas desenvolvidos em Java, portanto, não importando a configuração ou especificação desse aparelho.

Existem duas versões da máquina virtual Java, que são necessárias para rodar aplicativos desenvolvidos nessa linguagem:

- JRE, ou *Java Runtime Environment*, que contém apenas as bibliotecas para executar as aplicações.
- JDK, ou *Java Development Kit*, é o conjunto de utilitários e bibliotecas que permite desenvolver aplicativos para essa linguagem.

3.1.2 Java ME

Uma das mais importantes ramificações da linguagem de programação Java é a plataforma Java Micro Edition (Java ME, ou J2ME). Teve seu início no fim da década de 90 com o grande aumento no número de dispositivos móveis que vem havendo nos últimos anos. A plataforma Java ME é um conjunto de especificações e tecnologias que têm o foco em dispositivos pessoais, como: celulares, PDAs, controles remotos, aplicações domésticas e uma outra gama de dispositivos.

Estes dispositivos têm uma quantidade limitada de memória, menor poder de processamento, pequenas telas e geralmente baixa velocidade de conexão, mesmo com as rápidas melhorias conseguidas nesse ramo da indústria nos últimos tempos. Ao contrário dos computadores pessoais, que tem a possibilidade de upgrade e maior flexibilidade, os dispositivos móveis se aproveitam dessa plataforma, que possui uma máquina virtual, chamada de KVM (*Kilobyte Virtual Machine*), capaz de compactar as instruções o suficiente para as aplicações desenvolvidas em Java serem suportadas pelas restrições de memória destes dispositivos. (FRARI, [s.a.])

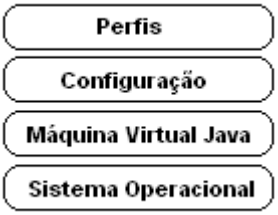

Figura 6. As quatro camadas do ambiente J2ME
Fonte: Introdução Java ME

A Figura 06 representa as quatro camadas de que consiste o ambiente J2ME, que são:

- Sistema Operacional: sistema operacional nativo;
- Maquina Virtual JAVA: que interpreta os *bytecodes* (códigos fontes da linguagem JAVA) e os transforma em códigos de máquina que o sistema operacional nativo consegue entender;
- Configuração: que define uma plataforma mínima para cada categoria de dispositivo;
- Perfis: as APIs que complementam e trazem novas funcionalidades às configurações.

3.1.3 NetBeans

Netbeans é uma das IDEs (*Integrated Development Environment* ou Ambiente Integrado de Desenvolvimento) mais utilizadas para desenvolvimento de aplicativos em ambiente Java, e mais especificamente, aplicativos para dispositivos portáteis. O Netbeans, desenvolvido totalmente em Java pela empresa Oracle Corporation, é gratuito, possui seu código aberto, e trabalha também com as linguagens: C, C++, PHP., Groovy, Ruby, além de XML e HTML, entre outras.

Disponível em várias plataformas e sistemas operacionais, como o Windows, Linux, MacOS e Solaris, ele possibilita a criação de uma grande variedade de aplicativos *desktop*, *Web* e móveis, entre outras plataformas, com controles de fácil acesso e propriedades ao mesmo tempo completas para o usuário avançado e simples para o usuário iniciante, como demonstrado na Figura 07.

Possui um grande conjunto de bibliotecas, módulos e APIs(*Application Program Interface*, um conjunto de rotinas, protocolos e ferramentas para a construção de aplicativos de *software*), além de muita documentação, inclusive em português, que pode ser encontrada na seção de documentação da página do Netbeans na internet: http://netbeans.org/kb/index_pt_BR.html.

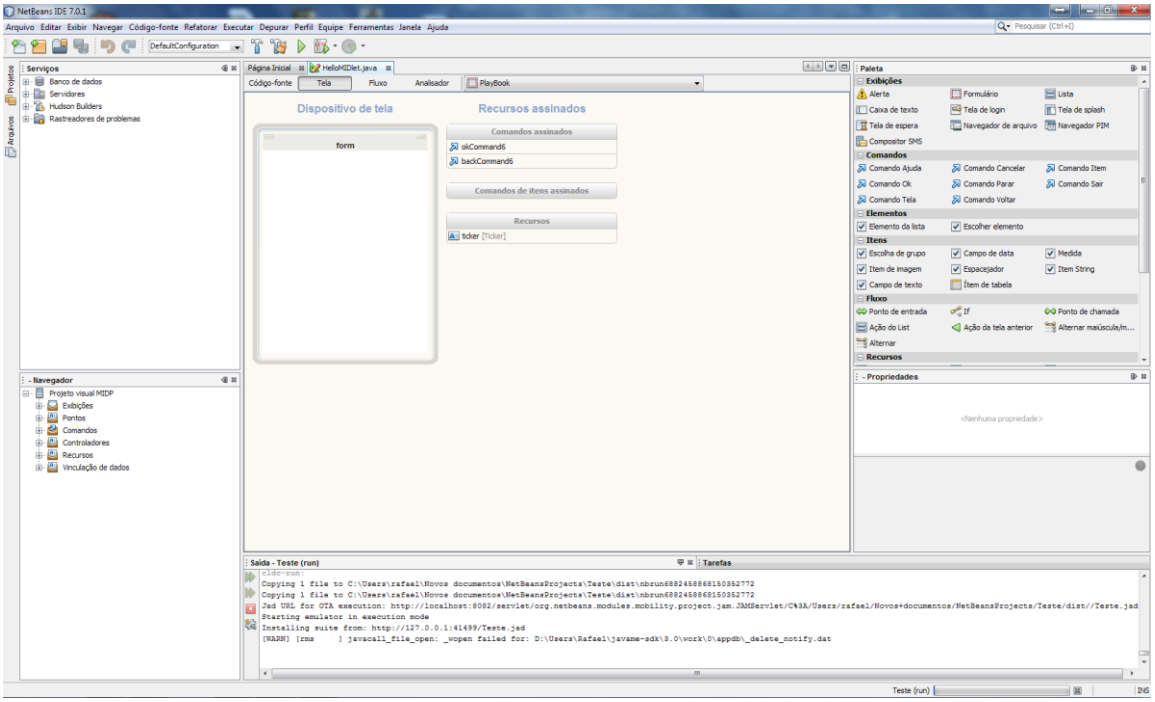

Figura 7. Interface do NetBeans, versão 7.0.1

Além das funcionalidades já mencionadas, o Netbeans, para facilitar o desenvolvimento de aplicativos móveis, possui um emulador, que executa o programa recém-desenvolvido numa tela muito parecida com a de um celular comum, que roda Java, inclusive com botões clicáveis simulando as teclas do celular, para trazer melhor a realidade do dispositivo móvel ao desenvolvedor e como ele será exibido e rodado simulando as limitações de hardware que esse tipo de dispositivo possui.

Figura 8: Emulador de dispositivo móvel do Netbeans

Com esse emulador, representado na Figura 08, o programa pode ser executado em condições mais reais e mais perto do que será exibido realmente na tela do aparelho móvel e assim, melhor testado.

3.1.4 DIA

O DIA é um aplicativo grátis e de código-aberto utilizado para design e criação de diagramas. Qualquer tipo de diagrama pode ser criado, pois ele não tem um foco específico em análise de *software*, mas mesmo assim pelas suas funcionalidades e por ser um programa aberto, foi o escolhido para a análise desse *software*.

Como pode-se verificar na Figura 09, ele é um programa que funciona de forma parecida com o Visio, da Microsoft. Como algumas de suas vantagens, ele possui uma interface simples e fácil de mexer e tem suporte a várias linguagens, inclusive o português. Além disso, permite desenhar diagramas de relacionamento, fluxo de dados, diagramas de rede e outros diagramas da linguagem UML.

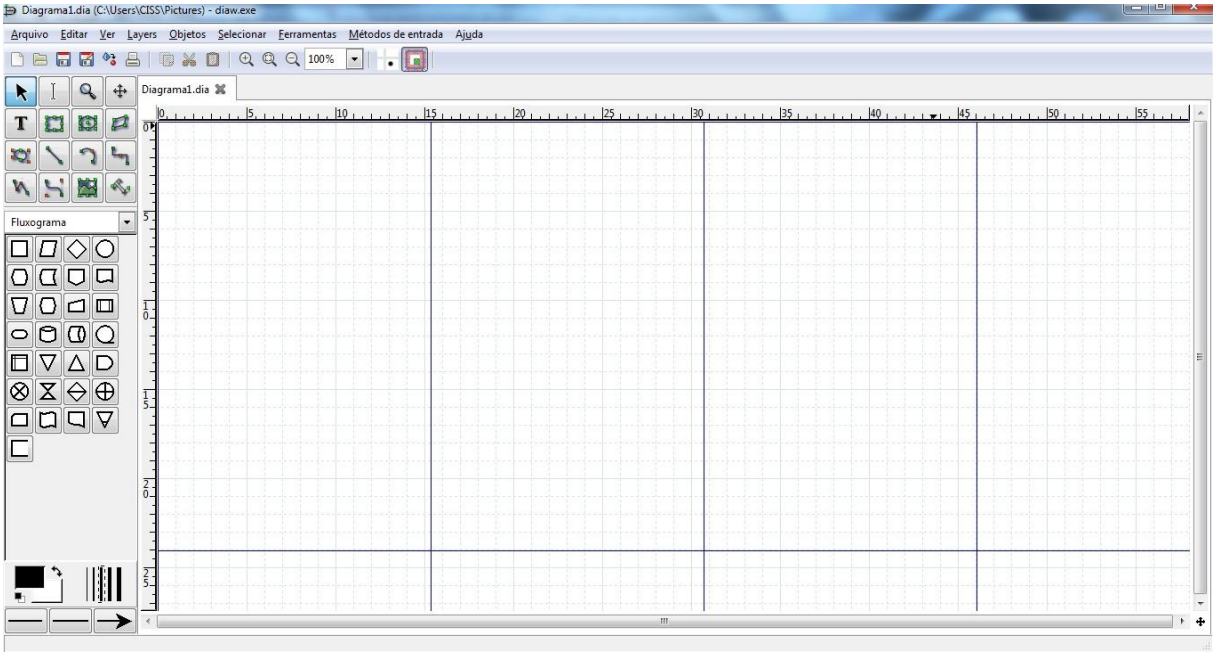

Figura 9: Interface do programa de análise DIA

3.1.5 Apache Tomcat 6

O Tomcat é um servidor de aplicações Java, que serve basicamente como um *container* de *servlets* (componentes que cuidam da transmissão e recepção de dados), e permite a publicação dessas aplicações em ambientes *web*, funcionando como um servidor, ou integrado a um servidor *web* Apache. (TOMCAT..., [s.a.])

A partir da publicação dessa aplicação no servidor Tomcat, pode ocorrer a comunicação envio e recepção de dados entre o aplicativo cliente e o servidor, possibilitando acesso a dados do banco de dados remoto de usuários, atletas, e avaliações, que é um dos propósitos do sistema.

3.1.6 Banco de dados MySQL

O MySQL é um sistema gerenciador de banco de dados (SGBD) desenvolvido pela empresa Oracle. É um dos tipos mais robustos de bancos de dados. Sendo utilizado por várias grandes empresas, tanto da área de informática, como de outras áreas. (DEITEL, 2005, p.910)

Ele é famoso por sua portabilidade e compatibilidade, pois pode ser utilizado com várias plataformas, tipos de equipamentos, sistemas operacionais ou linguagens de programação.

3.1.7 Record Management System

No caso específico dessa aplicação, para um aproveitamento maior dos recursos e para garantir menor utilização de banda, os dados provenientes do banco de dados MySQL são armazenados primeiramente em registros *Record Store,* para depois serem utilizados.

Os registros *Record Store* não são um banco de dados relacional, e sim apenas um conjunto de registros organizados de forma a permitir o máximo de integridade e persistência dos dados durante o uso da aplicação. Se faz necessário o uso desses registros pela própria natureza das aplicações móveis, que pela sua portabilidade, podem ser usadas em qualquer lugar, mesmo aonde o sinal de rede ou internet é pobre ou inexistente. (PATTA, [s.a.])

Para trabalhar com *Record Store* é usada a API RMS, que suporta a criação e manipulação de vários registros para armazenamento de dados.

3.1.8 Persistência de dados com o plugin Floggy

Floggy é um framework de persistência de dados, que pode ser utilizado com várias IDEs de desenvolvimento Java, como o NetBeans, Eclipse, Ant, Maven ou até mesmo linha de comando. Ele é configurado separadamente na IDE, como um plugin, e tem métodos especialmente desenvolvidos para auxiliar na manipulação e persistência de dados do aplicativo móvel. Segundo Lugon e Russato(2005, p.16) “O processo de permanência pode ser definido como o armazenamento e manutenção do estado de objetos em algum meio não-volátil, como um banco de dados.”

O Floggy age como uma camada de persistência de dados, que torna o software independente e centraliza os métodos de consulta e manipulação de dados. Ainda segundo Lugon e Russato(2005, p.26) “A vantagem no uso da camada de persistência é o encapsulamento do código de manipulação de dados, proporcionando total abstração ao engenheiro de software.”.

3.1.9 SQLYog

Para a conexão com o banco de dados, inclusão, alteração e manutenção de dados externamente ao sistema, foi utilizado o programa SQLYog, da empresa

Webyog. É uma interface gráfica para gerenciamento de banco de dados leve, mas robusta, que permite completo controle de bancos de dados MySQL, consulta, criação de scripts, criação e alteração de tabelas e análise e exportação de dados. A Figura 10 mostra um pouco da interface do programa. (ABOUT US..., [s.a.])

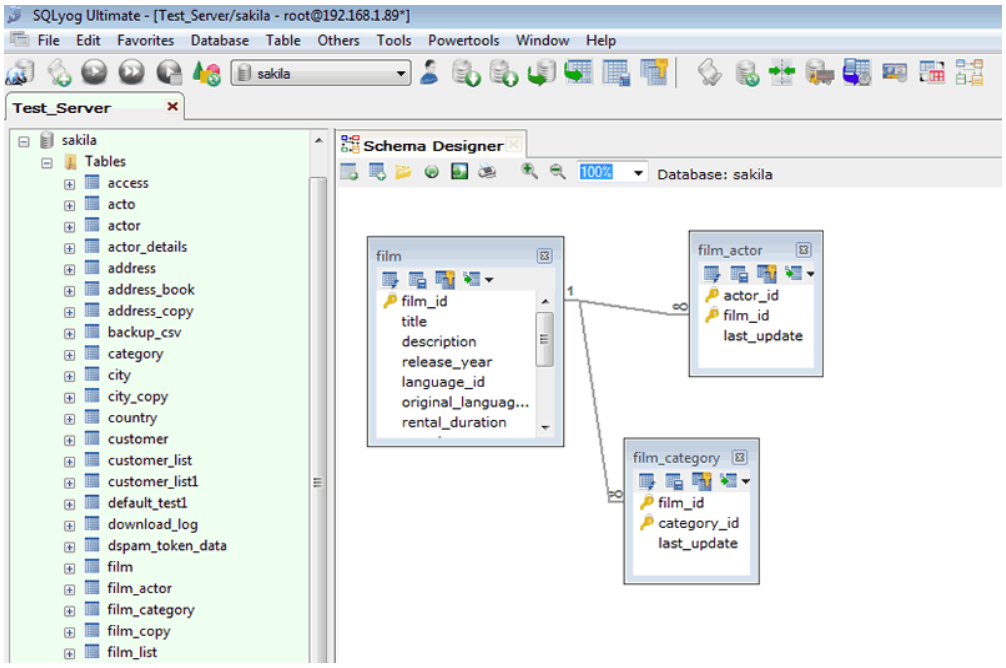

Figura 10: Interface do SQLYog

3.2 MÉTODOS

O método utilizado para desenvolvimento desse software é o método de desenvolvimento em Cascata. Ele segue uma série de passos ordenados, e ao final de cada fase, é gerada uma versão. Como o desenvolvimento é linear, gasta-se mais tempo e esforços na fase de planejamento do que as outras metodologias.

Várias das metodologias novas e mais avançadas são derivadas do método em Cascata.

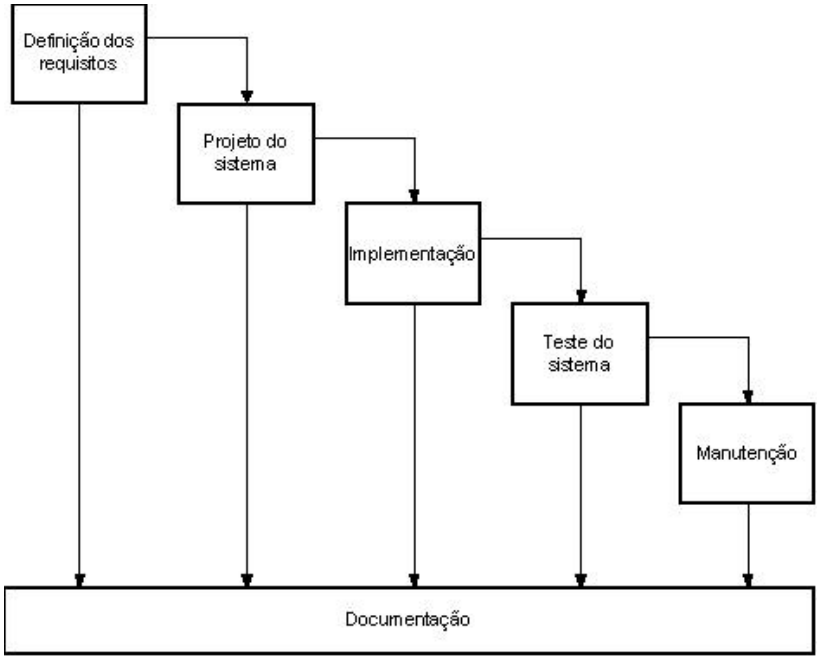

Figura 11: Modelo em cascata modificado

Fonte: Adaptado de Sommerville(2007, p. 44)

Como representado na Figura 11, esse modelo possui apenas cinco grandes fases, que são:

- Análise e definição de requisitos: objetivos, funções e restrições são definidos, com ajuda de clientes e usuários, e servem como uma especificação do sistema, indicando o que deve ser implementado;
- Design de sistemas e *software: e*nvolve a descrição do sistema e do *software* em termos de unidades abstratas e de suas relações, indicando como o *software* deve ser implementado;
- Implementação e testes de unidade: as unidades do *software* devem ser codificadas e testadas individualmente;
- Integração e testes de sistema: as unidades são integradas e testadas;
- Entrega, operação e manutenção: o sistema é instalado e colocado em operação. A manutenção envolve a correção de erros e evolução do sistema para atender a novos requisitos;

Cada uma dessas fases possui uma relação com a documentação, que em todos os *softwares* deve ser suficiente para possibilitar futuras alterações de código sem uma re-análise total dos dados. E, isso é importante mesmo num *software* simples como esse proposto.

4. RESULTADOS E DISCUSSÕES

Esse sistema tem como função principal a organização e cadastro de informações referentes a um time de futebol americano. Para chegar a esse objetivo, como em qualquer projeto, se encontraram algumas dificuldades, desde a análise dos requisitos até a implementação do código.

Apesar de não se aprofundar muito na parte de análise dos dados coletados, a funcionalidade desse *software* é muito clara. Ele traz módulos de cadastro e apresentação dos dados mais importantes para um time de futebol americano – que são: os dados dos atletas; as avaliações físicas dos mesmos; os treinos; e, os jogos desse time, permitindo assim uma total coleta e controle desses dados.

Posteriormente isso irá ajudar o técnico e seus auxiliares a fazer verificações que não eram possíveis antes, sem o controle dos dados, como: saber qual é a velocidade de um jogador; o quão ágil é um jogador; quando e aonde foram realizados os treinos e jogos do time; e o placar e adversário dos jogos que foram disputados. Esse tipo de verificação vai auxiliar os coordenadores do time a decidir qual será o atleta mais preparado para um jogo e num esporte tão competitivo, isso pode fazer a diferença entre perder e ganhar.

Para transpassar as dificuldades foram utilizados todos os elementos de análise, as ferramentas e métodos apresentados anteriormente, e logo abaixo, cada tópico desses será apresentado e descrito mais especificamente.

4.1 ANÁLISE

Nesse item serão apresentados os artefatos utilizados durante a análise de requisitos feita para o desenvolvimento desse sistema. Para diagramação foi usado o programa grátis DIA.

4.1.1 Declaração de Objetivos

Desenvolver um sistema para aplicativos móveis que permita o cadastro, controle e análise de dados referentes a atletas amadores de futebol americano.

4.1.2 Lista de Eventos

É uma lista com as funcionalidades que existem no sistema, cadastrados seguindo várias diretrizes de nomeação e classificação. Cada evento é classificado pelo seu tipo (fluxo de dados, temporal ou de controle), e são listados: os estímulos que apontam para a o sistema a ocorrência do evento, a ação a ser feita pelo sistema e a resposta produzida por essa ação.

Cada um os eventos listados no Quadro 1, representa uma função do sistema bem como especificações sobre as ações tomadas por essas funções.

Quadro 1: Lista de Eventos

N	Evento	T	Estímulo	Ações	Resposta
1	Administrador faz cadastro de usuários	F	Cadastro de Usuários	Registrar cadastro de usuários	Cadastro de usuários registrado
2	Usuário faz cadastro de atletas	F	Cadastro de Atletas	Registrar cadastro de atletas	Atleta cadastrado
3	Usuário altera dados de atletas	F	Alteração de cadastro dos atletas	Registrar alteração no cadastro de atletas	Alteração no cadastro do atleta registrado
4	Usuário exclui atletas	F	Exclusão de atletas	Registrar exclusão de atletas	Atleta excluído
5	Usuário insere dados de avaliações físicas dos atletas	F	Inserção de dados físicos dos atletas	Inserir informação física dos atletas	Informação física do atleta inserida
6	Usuário altera avaliações físicas	F	Alteração de cadastro de avaliações	Registrar alteração no cadastro de avaliações	Alteração de avaliação realizada
7	Usuário exclui avaliações	F	Exclusão de avaliações	Registrar exclusão de avaliações	Avaliação excluída
8	Usuário faz cadastro de Treinos	F	Cadastro de Treinos	Registrar cadastro de treinos	Treino cadastrado
9	Usuário altera dados de treinos	F	Alteração de cadastro dos treinos	Registrar alteração no cadastro de treinos	Alteração no cadastro do treino registrado
10	Usuário exclui treinos	F	Exclusão de treinos	Registrar exclusão de treinos	Treino excluído
11	Usuário faz cadastro de Jogos	F	Cadastro de Jogos	Registrar cadastro de jogos	Jogo cadastrado
12	Usuário altera dados de jogos	F	Alteração de cadastro dos jogos	Registrar alteração no cadastro de jogos	Alteração no cadastro do jogo registrado
13	Usuário exclui jogos	F	Exclusão de jogos	Registrar exclusão de jogos	Jogo excluído
14	Usuário solicita exibição de análise de dados dos atletas	F	Exibição de dados de atletas	Listar dados de atletas	Relatório de dados de atletas

4.1.3 Diagrama de Entidade e Relacionamento (DER)

Utilizando a linguagem UML e os procedimentos de análise, foi feito também o DER, que mostra as tabelas e campos das tabelas que serão utilizadas no banco de dados desse sistema, como mostrado na Figura 12.

A tabela de atletas possui os campos para armazenar as informações principais dos atletas, como o número, nome, posição, e uma possível observação.

A tabela de avaliações possui os campos referentes às informações físicas dos atletas, que são as informações coletadas no dia de avaliação física dos mesmos. Cada campo representa o tempo - tempo40, tempo20, tempo3cones, - ou distância percorrida – saltohorizontal - pelo atleta em cada uma das provas de avaliação já explicadas anteriormente. Essa tabela, através do campo codatleta, possui relação com a tabela de atletas, utilizando o campo *id*.

Na tabela de treinos ficam gravadas as informações referentes aos treinos, com a data, local e observação, além de dois que irão mostrar se esse treino teve *redzone* – a parte no final do treino que se assemelha a um jogo e os times de ataque e defesa treinam suas jogadas – e se teve exercícios físicos (pelo campo físico).

Quanto às informações sobre os jogos, ficam armazenadas na tabela de mesmo nome. Também são gravadas informações básicas sobre essas partidas, como a data, local, observação e os placares dos times.

A tabela de usuários é onde ficam gravadas as informações sobre os usuários, para utilização para o *login* do sistema.

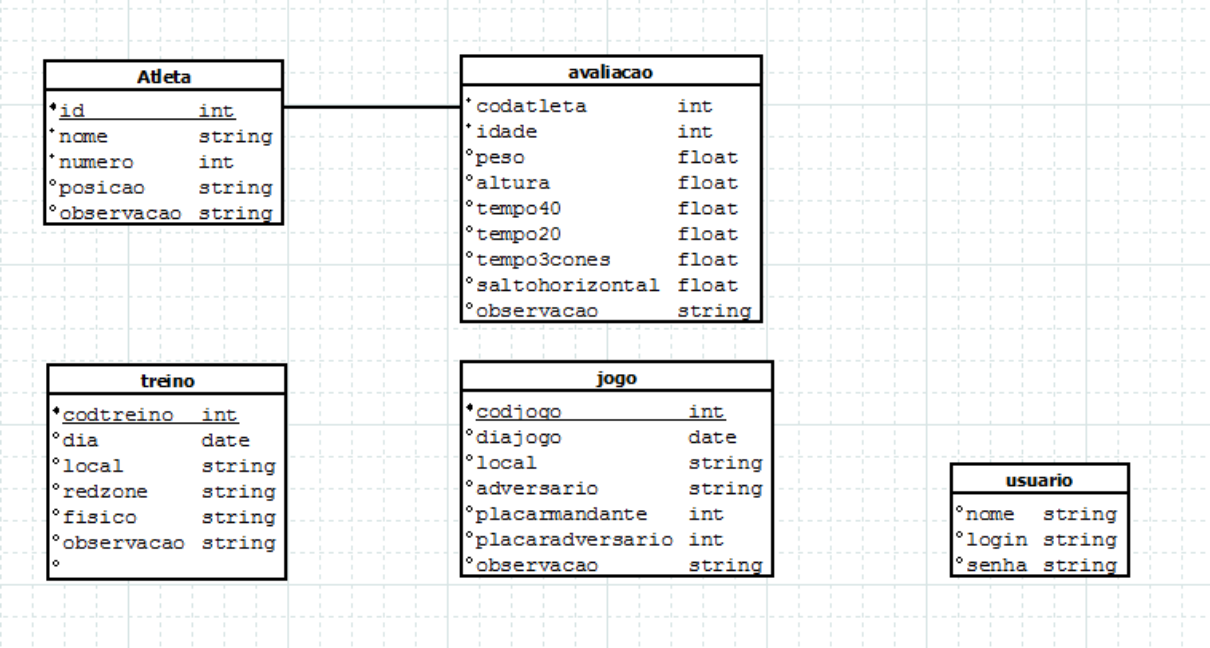

Figura 12: Diagrama de Entidade e Relacionamento do sistema

4.1.4 Diagrama de Fluxo de Dados(DFD)

A Figura 13 mostra através do diagrama de fluxo de dados os processos realizados no sistema, com o seu devido agente, aquele que os realizam, e com o resultado, a gravação de dados em um armazenamento.

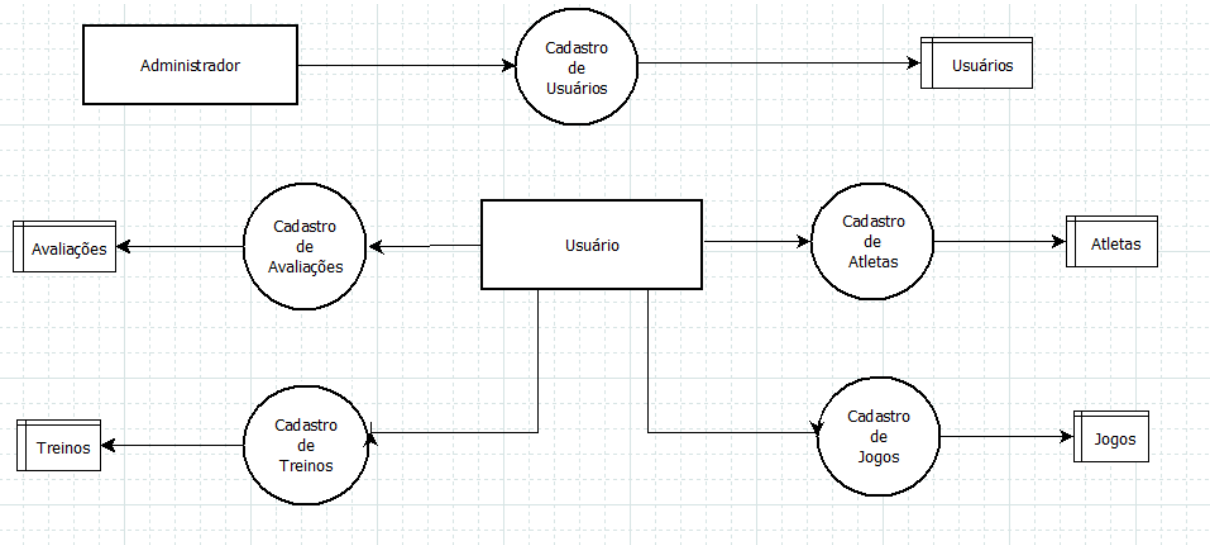

Figura 13: Diagrama de Fluxo de Dados do sistema

4.1.5 Diagrama de Casos de Uso

O diagrama de caso de uso, como mostrado na Figura 14, representa de forma mais visual como será o funcionamento do sistema, sem preocupações com dados técnicos, métodos e ferramentas.

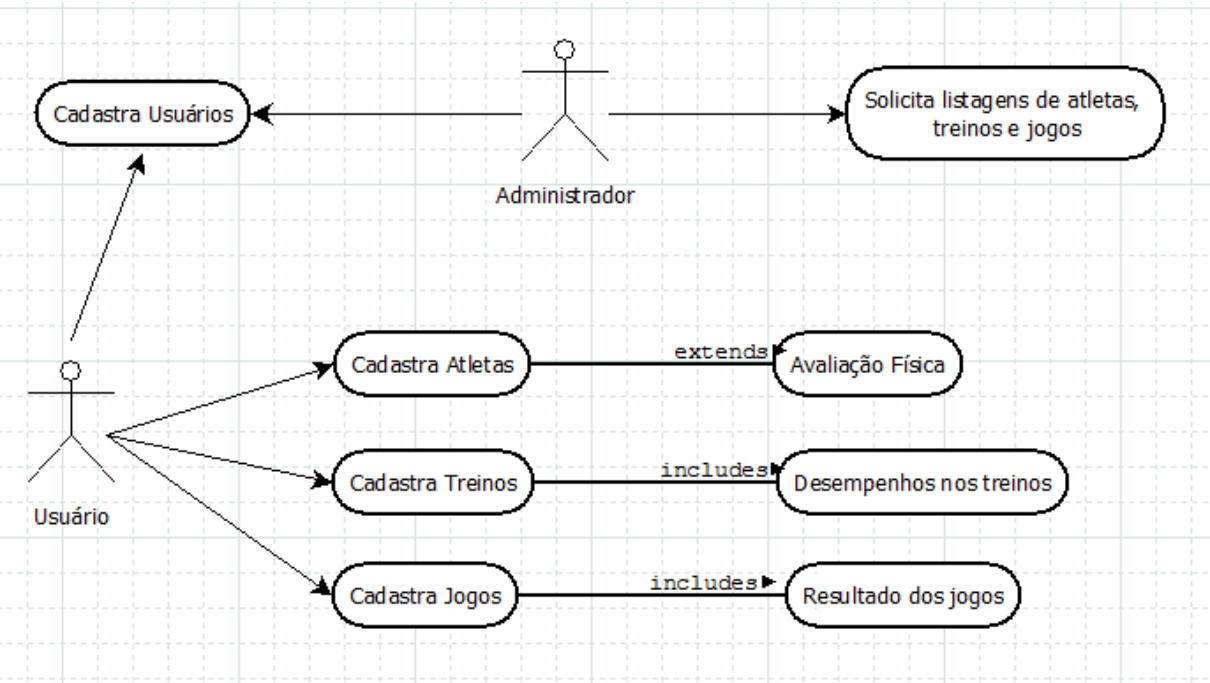

Figura 14: Diagrama de Caso de Uso do sistema

4.2 IMPLEMENTAÇÃO

Nessa fase foram postos em prática todos os processos comentados e demonstrados nas fases anteriores.

Como a maioria das informações necessárias para funcionamento desse aplicativo se encontram em um banco de dados MySQL, que fica no servidor, é necessário um meio de comunicação entre o aplicativo cliente, que é instalado no dispositivo móvel e um aplicativo servidor que fica numa máquina remota, na web ou na mesma rede de dados. Esse aplicativo servidor serve como uma ponte de comunicação entre aplicação Java e o servidor de banco de dados, que é o caso da aplicação desenvolvida para esse trabalho.

Assim, quando o aplicativo cliente necessita de algum dado do banco de dados remoto, ele aciona a comunicação com o servidor através da interface de

programação JDBC (*Java Database Connectivity*), que é um conjunto de APIs escritas em Java, com as quais é possível estabelecer conexão com o banco de dados, enviar instruções em linguagem SQL e receber os resultados dessas instruções.

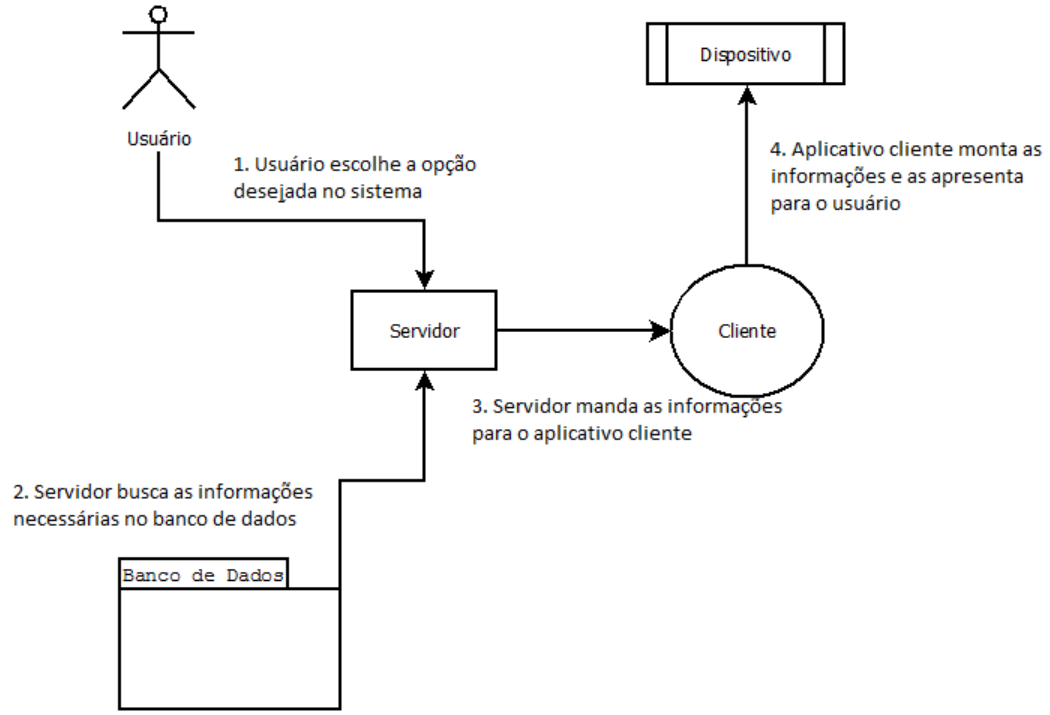

Figura 15 Processo de comunicação cliente-servidor

A Figura 15 demonstra o processo que e feito em todos os métodos que utilizam essa comunicação com o servidor.

Algumas considerações gerais sobre o desenvolvimento, implementação e testes desse sistema:

- Para o completo funcionamento e melhor aproveitamento de recursos desse aplicativo, foram desenvolvidos dois projetos, um cliente e um servidor. No servidor ficam apenas os, métodos que utilizam conexão no banco de dados MySQL, enquanto no aplicativo cliente ficam as telas, validações e métodos que montam as informações para listagem ou exportação.
- Em todas as telas que fazem consulta no banco de dados, primeiramente é feita a leitura dos dados e inseridos os valores em registros RMS, para evitar acesso constante ao banco de dados principal. Cada funcionalidade tem seu método de leitura e importação dos dados no aplicativo servidor e o método de

listagem ou cadastro no aplicativo cliente. Esses dois aplicativos se comunicam por meio de conexões e vetores de dados.

- Em toda situação que acontece uma busca de dados no servidor, é mostrada uma mensagem de erro ou sucesso concatenada com o retorno que veio do servidor. Para isso foi usado um componente *Alert*, que é valorizado com a mensagem dependendo do retorno.
- Cada campo de dados utilizado nos formulários possui um nome único e num padrão definido anteriormente, para facilitar a documentação e visualização do código.
- A tela de *login* é um componente *LoginScreen*, o qual já vem por padrão com os campos de *login* e senha e seu tratamento próprio.
- Foi usado um padrão para as telas, no qual cada item compreendido pelo sistema terá uma tela de cadastro e alteração de dados e uma tela de listagem.
- Todas as telas de cadastro possuem três botões, como demonstrados na Figura 16 – e mais o botão de voltar para a tela anterior - e funcionam da mesma forma. Caso o usuário deseje incluir um atleta, treino, jogo ou avaliação, deve informar seus dados e pressionar o botão Incluir. Caso o usuário deseje alterar qualquer informação, deve selecionar no *ChoiceGroup* correspondente, alterar o que deseja e pressionar o botão Alterar. E caso o usuário deseje excluir algum registro, deve selecioná-lo no *ChoiceGroup* e pressionar o botão Excluir.

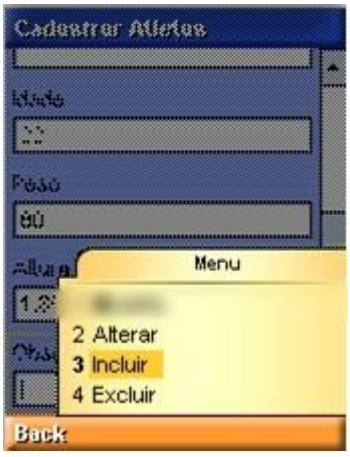

Figura 16 Comandos disponíveis nas telas de cadastros

- Ainda, dentro do servidor estão os métodos que possuem as instruções em SQL que realmente serão executadas no banco de dados. Para cada tipo de instrução (*select, update, insert, delete*) existe um método, pois as informações são

passadas de formas diferentes para cada um desses comandos, apesar de a comunicação com o banco de dados se dar da mesma forma.

- Foi criado um método chamado *itemStateChanged*, que juntamente com um comando verifica se o conteúdo de um dos campos *choiceGroup* foi mudado, e transfere automaticamente os dados daquele atleta,treino,jogo ou avaliação para os respectivos campos de texto da tela de cadastro. Isso facilitará a alteração e visualização dos dados cadastrados.

4.2.1 Tela de login

Figura 17 Tela inicial do sistema

Quando iniciado o aplicativo, é aberta a tela de *login* mostrada na Figura 17. Ela possui duas opções, que podem ser acessadas pelos comandos de tela Cadastrar e *Login*.

Caso acionado o comando de login, vão ser feitas duas verificações no banco de dados, primeiro para validar se foram inseridos dados de login e senha nos respectivos campos, e depois para validar se o usuário e senha realmente estão cadastrados no banco de dados e se as duas informações estão corretas.

No aplicativo servidor foi criado um método que verifica se o usuário e senha digitados estão corretos, e caso o resultado for positivo, ele retorna um vetor de dados com o valor *Sucess*. Caso usuário ou senha estiverem incorretos, ele retorna uma mensagem de erro, que aparece para o usuário na hora do login. No quadro 2 é apresentado o código fonte do método de validação do aplicativo servidor:

Quadro 2 Método de validação do usuário

```
01 private Vector validUser(Vector data) {
02
03     Vector ret = new Vector();
04     try{
05
06     String fields[] = (String[]) data.elementAt(0);
07
08     PreparedStatement ps =  (PreparedStatement) con.prepareStatement(
09           "select * from usuarios where login=? and senha=?");
10
11     ps.setString(1, fields[0]);
12     ps.setString(2, fields[1]);
13
14     ResultSet rs = ps.executeQuery();
15
16     if( rs.next()){
17        fields =  new String[1];
18        fields[0] = "Sucess";
19     } else{
20        fields =  new String[2];
21        fields[0] = "Error";
22        fields[1] = "Usuário não encontrado";
23     }
24     ret.add(fields);
25
26     }catch (SQLException ex){
27        String fields[] =  new String[2];
28        fields[0] = "Error";
29        fields[1] = ex.getMessage();
30        ret.add(fields);
31     }
32     ret.trimToSize();
33     return ret;
34  }
```

Das linhas 3 a 6 são criados o vetor de dados que será usado pra retorno e vetor que armazenará os dados vindos do aplicativo cliente. Da linha 8 a 14 é montado a consulta SQL que será executada no banco de dados, aonde será buscado o usuário com o *login* e senha que foram digitados na tela de *login* do aplicativo cliente. Da linha 16 a 23 é feita uma verificação, e caso existir um usuário com esses dados, é retornada a *string* de sucesso para o aplicativo cliente, permitindo assim a entrada no menu principal do aplicativo.

Caso o usuário não for encontrado, ou usuário e senha não forem corretos, é retornada uma *string* passando essa mensagem para o usuário. Na linha 24 é adicionada a mensagem de sucesso ou de insucesso no vetor e posteriormente, na linha 33, ela é retornada para o aplicativo cliente. Das linhas 26 a 31, é feito o tratamento de uma possível exceção, que pode ocorrer num erro ou falha de comunicação com o banco de dados.

No aplicativo cliente ocorre a validação desse retorno do servidor. Caso o retorno foi de sucesso, o menu principal do programa será exibido. Caso contrário,

apresentará uma mensagem de *Login* e, ou, Senha incorretos, e voltará para a tela de login.

4.2.2 Tela Principal

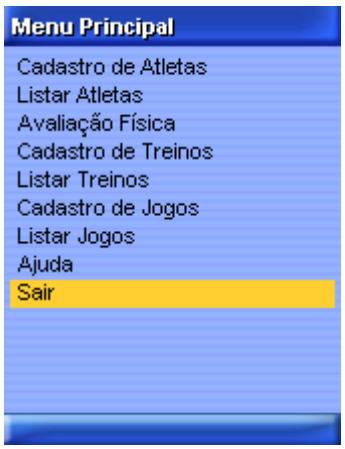

Figura 18 Menu principal, com as opções disponíveis ao usuário

Depois de logado no sistema com o usuário e senha, aparece a tela principal, com as opções de cadastro e listagem disponíveis, como representado na Figura 18. É possível fazer cadastro e controle de atletas, treinos, jogos e avaliações físicas de atletas, além de poder listar cada um desses.

4.2.3 Telas de Cadastro

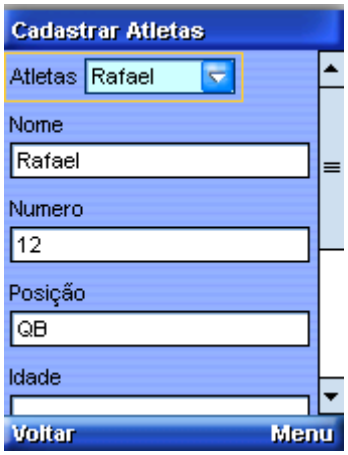

Figura 19 Tela de cadastro de atletas

As telas de cadastros são compostas de um componente do tipo *Choice Group* permitindo a seleção do atleta, treino, jogo ou avaliação que deve ser alterado

ou excluído, e dos campos de texto com as informações que compõem seu cadastro. Esse padrão pode ser visto na Figura 19.

4.2.4 Comunicação cliente-servidor

No aplicativo servidor foi criado um método para importação dos dados dos registros, e esse método será utilizado cada vez que os dados forem solicitados para a tela no aplicativo cliente.

Esse método roda um comando de SQL *select* no banco de dados principal, mais especificamente na tabela de atletas, e traz todos os dados do atleta, por exemplo. Ele é rodado quantas vezes for necessário utilizando-se da rotina *while* e no final uma variável de retorno é enviada para o aplicativo cliente com todos os registros e seus dados.

Caso algum erro na comunicação com o banco de dados ou importação dos dados ocorrer, é retornado um vetor valorizado com uma string com a mensagem de erro, para permitir a verificação pelo usuário.

Quadro 3 Método de importação de dados de atletas

```
01 private Vector importarAtletas() {
02     Vector ret = new Vector();
03     try {
04         Statement stmt = (Statement) con.createStatement();
05         ResultSet rs = stmt.executeQuery("select * from atletas");
06         while (rs.next()){
07             Integer id = rs.getInt("id");
08             String nome = rs.getString("nome");
09             Integer numero = rs.getInt("numero");
10             String posicao = rs.getString("posicao");
11             String observacao = rs.getString("observacao");
12             String fields[] = new String[5];
13
14             fields[0] = String.valueOf(id);
15             fields[1] = nome;
16             fields[2] = String.valueOf(numero);
17             fields[3] = posicao;
18             fields[4] = observacao;
19
20             ret.add(fields);
21         }
22     } catch (SQLException ex) {
23         ret = new Vector();
24         String fields[] = new String[2];
25         fields[0] = "Error";
26         fields[1] = ex.getMessage();
27
28         ret.add(fields);
29     }
30    return ret;
31 }
```

Na linha 02 e 12 são criados os vetores que armazenarão os dados que serão mandados ao aplicativo cliente – que fez a requisição da lista de atletas, com seus dados. Na linha 04 e 05 é montada a consulta na tabela de atletas do banco de dados, para trazer todas as informações referentes a eles. Da linha 06 a 11 é feita uma rotina para buscar os dados resultantes dessa consulta enquanto existir um próximo registro, ou seja, todas as informações de atletas serão recuperadas. Os dados de cada campo dessa tabela são passados para a variável correspondente, sempre respeitando o tipo de dados correspondente (*Int* ou *String*, nesse caso).

Nas linhas 14 a 20 os dados selecionados são incorporados ao vetor previamente definido, e depois esse vetor é adicionado à variável de retorno, que é devolvida ao aplicativo cliente na linha 30. Da linha 22 a 29 é feito o tratamento de exceções, para possíveis erros de conexão ao banco de dados que venham a ocorrer.

Para os treinos, jogos e avaliações, a lógica é a mesma, portanto são usados também métodos que possuem o mesmo funcionamento, apenas sendo alterados para trazer os campos correspondentes a cada tabela do banco de dados.

No aplicativo cliente, são lidos esses dados que foram importados do banco de dados principal pelo método de importação do servidor, e com a variável de retorno são montados os registros RMS, a partir dos campos definidos na classe. Por exemplo, a classe atleta possui todos os campos para armazenamento dos dados de atletas, e também seus métodos de encapsulamento (métodos *Get* e *Set*), que são usados para leitura e armazenamento de dados nos registros do banco de dados, respectivamente.

Para as funções de inclusão, alteração e exclusão, esse mesmo método é usado para montar os dados na tela antes de o processo definido ocorrer, para os registros sempre estarem gravados em RMS e haver consistência de dados.

Assim que é pressionado o botão para incluir, alterar ou excluir as informações que estão nos campos da tela de cadastro são jogadas para variáveis compatíveis com seu tipo, e gravadas nos registros *RecordStore*.

Posteriormente, é montada o vetor de retorno, com os dados que serão atualizados no banco de dados principal.

Enquanto isso, para a exclusão de registros, assim que é pressionado o botão de excluir, é chamada a rotina de exclusão do servidor. Ela recebe o vetor com o código do atleta que deve ser excluído. Esse código foi buscado a partir do atleta

que estava selecionado no *choiceGroup* de atletas da respectiva tela, portanto só é deletado o atleta que estava realmente selecionado.

Quadro 4 Método de alteração de atletas do aplicativo cliente

```
01 if (getTfnomeatleta().getString() == "" || getTfnumeroatleta().getString() == "") {
02         getAlstatus().setString("Insira os dados obrigatórios para alteração do atleta");
03         switchDisplayable(getAlstatus(), getFmcadastroatletas());
04     } else {
05
06         String nome = tfnomeatleta.getString();
07         int numero = Integer.parseInt(tfnumeroatleta.getString());
08         String posicao = tfposicao.getString();
09         String observacao = tfobservacaoatleta.getString();
10
11         Atleta atletas = (Atleta) listaAtletas.elementAt(cgatleta.getSelectedIndex());
12
13         atletas.setNome(nome);
14         atletas.setNumero(numero);
15         atletas.setPosicao(posicao);
16         atletas.setObservacao(observacao);
17
18         HttpConnectionClient client = new HttpConnectionClient(URL, "POST");
19         Vector data = new Vector();
20
21         String fields[] = new String[5];
22         fields[0] = String.valueOf(atletas.getId());
23         fields[1] = atletas.getNome();
24         fields[2] = String.valueOf(atletas.getNumero());
25         fields[3] = atletas.getPosicao();
26         fields[4] = atletas.getObservacao();
27
28         data.addElement(fields);
29
30         client.add("exportarAtletas", data);
31         Vector ret = client.enviarReceberDados();
```

O código-fonte demonstrado no Quadro 4 é o código utilizado no método de alteração de atletas, do aplicativo cliente. Das linhas 1 a 4 é feita uma validação, para verificar se existem dados nos campos de texto referentes ao nome e número dos atletas. Caso for validada negativamente, é apresentada uma mensagem para inserir os campos obrigatórios. Caso passar dessa condição, continua a alteração do atleta.

Das linhas 06 a 09 são definidas as variáveis referentes a cada campo retornado do banco de dados e valorizadas com os dados vindos dos campos de texto da tela. Na linha 11 é montada uma lista de atletas e definido que o atleta selecionado no *choiceGroup* será o atleta a ser alterado. Nas linhas 13 a 16, são chamados os métodos de encapsulamento *Set*, para serem armazenados os dados referentes ao atleta e depois, montados os dados no vetor *fields* a partir desses

dados gravados, nas linhas 21 a 26. Da 28 a 30 são enviados os dados para o aplicativo servidor, para aí ser utilizado o método ExportarAtletas, que é o método que realmente irá rodar o comando SQL no banco de dados para atualizar o cadastro desse atleta.

Quadro 5 Método de alteração de atletas do aplicativo servidor

```
01  private Vector exportarAtletas(Vector data)  {
02      Vector ret = new Vector();
03      try {
04          for(int i = 0; i<data.size();i++){
05              String fields[] = (String[]) data.elementAt(i);
06              PreparedStatement ps;
07
08                  ps = (PreparedStatement) con.prepareStatement("update atletas set nome = 09?,numero = ?,posicao =
?,observacao = ? where id = ?");
10
11              ps.setString(1,fields[0]);
12              ps.setString(2,fields[1]);
13              ps.setString(3,fields[2]);
14              ps.setString(4,fields[3]);
15              ps.setString(5,fields[4]);
16
17              ps.executeUpdate();
18          }
19
20          Statement stmt = (Statement) con.createStatement();
21          ResultSet rs = stmt.executeQuery("select * from atletas");
22          while (rs.next()){
23              String fields[] = new String[4];
24              fields[0] = rs.getString("nome");
25              fields[1] = rs.getString("numero");
26              fields[2] = rs.getString("posicao");
27              fields[3] = rs.getString("observacao");
28              ret.add(fields);
29          }
30      } catch (SQLException ex) {
31                ret = new Vector();
32                String fields[] = new String[2];
33                fields[0] = "Error";
34                fields[1] = ex.getMessage();
35              }
36       ret.trimToSize();
37       return ret;
38  }
```

No Quadro 5 está representado o método de alteração de dados de atletas utilizado no aplicativo servidor. Esse método recebe as informações vindas do cliente e executa o comando SQL de update no banco de dados a partir dessas informações.

Ele é composto de uma rotina for (linha 04), que passa lendo todo o conteúdo da variável data – que é a variável retornada pelo aplicativo cliente, com os dados para serem alterados. Das linhas 06 a 09 é preparado o script para atualização dos

dados, enquanto das linhas 11 a 15 são lidos os dados de cada campo e passados ao campo respectivo na hora de atualizar.

Depois desse comando de *update*, nas linhas 21 a 28 é feito novamente uma consulta no banco, para já retornar os registros atualizados para o aplicativo cliente. No aplicativo cliente, posteriormente são montados os dados na tela já com esses dados atualizados. Na linha 28 é adicionado o conteúdo dos dados selecionados para a variável de retorno e na linha 37 ela é passada novamente para o cliente.

Caso ocorrer uma exceção – problema na conexão com o banco de dados – esse erro será tratado, com o código entre as linhas 30 e 35.

4.2.5 Telas de Listagem

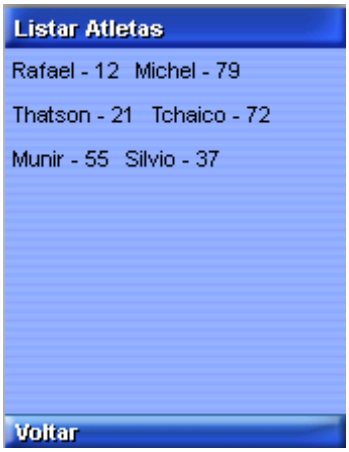

Figura 20 Tela de listagem de atletas

Para as listagens de dados, são utilizados novamente os métodos de importação do servidor - por exemplo, o método importarAtletas - que recupera as informações de cada atleta e as grava nos registros RMS, permitindo assim, a busca das informações e as trazendo na tela utilizando um componente *Form*, como demonstrado na Figura 20.

CONCLUSÃO

Através desse trabalho, pôde-se ver que mesmo um time amador de futebol americano não sendo uma empresa ou uma instituição e não visando lucros, necessita-se de um controle das informações pertinentes a ele.

Deve se manter um cadastro atualizado dos atletas e manter um controle de quando serão realizados os treinos e jogos do time, além de ter informações sobre as avaliações físicas dos atletas, permitindo um melhor conhecimento da equipe por seus gestores.

Foram encontradas várias dificuldades no desenvolvimento desse *software*, principalmente com relação ao desenvolvimento do mesmo e a pouca experiência com desenvolvimento em Java ME foi um fator importante.

O fato de essa comunicação entre cliente e servidor requerer alguns *plug-ins* e *frameworks* de conexão e persistência também dificultou, pois apesar de tudo relacionado ao desenvolvimento estar centralizado no Netbeans, trabalhar com várias ferramentas auxiliares torna mais complexo o desenvolvimento, especialmente para funções de envio de dados.

Com esse *software*, o técnico do time - e seus assistentes – tem mais uma ferramenta disponível para controlar seu elenco com informações atualizadas, utilizando o cadastro de atletas; analisar a condição física de seus atletas, a partir do cadastro de avaliação física e ter uma listagem organizada de seus treinos realizados e jogos disputados. Com essas informações, o técnico pode estar um passo a frente dos adversários e saberá sempre colocar em campo os jogadores que estão melhores preparados, e gerenciar corretamente e pontualmente a equipe.

E tudo isso num aplicativo móvel, que propicia mobilidade e flexibilidade aos seus usuários, permitindo o uso mesmo em situações que outras tecnologias não alcançariam. O fato de ele ter sido desenvolvido com a linguagem de programação Java torna ainda maior suas vantagens e sua portabilidade, devido à grande popularização da tecnologia Java, que possui muitos adeptos no mundo inteiro.

Esse é um *software* muito específico, portanto sua utilização comercial seria limitada aos cerca de 150 times existentes agora no país e mais os que estão apenas iniciando e ainda não estão completamente formados. Mas com algum investimento e algumas adaptações à realidade dos times maiores – que hoje em dia são quase profissionais - esse *software* poderia ser distribuído ou

comercializado. A funcionalidade prioritária a ser adicionada seria uma tela, ou várias telas, para análise dos dados, como por exemplo, para mostrar qual o jogador mais rápido, qual o mais pesado, a média de velocidade por posição, entre outras estatísticas.

Além disso, poderiam ser adicionadas algumas funções de controle de *depth chart* – o elenco dos times separando titulares e reservas – e também controle e organização das jogadas dos times (o *playbook*), assim proporcionando um controle maior dos dados dos times.

REFERÊNCIAS

ALMEIDA, Luis Fernando S. **Regras do Futebol Americano**. 2006. Disponível em: <http://www.nflbrasil.com.br/rules.htm>. Acesso em: 29 nov. 2011.

BLAHA, Michael. BUMBAUGH, James. Tradução Vieira, Daniel. **Modelagem e Projetos Baseados em Objetos com UML 2.** Rio de Janeiro: Editora Elsevier, 2006.

BOOCH, Gary; HARBAUGH, James; JACOBSON, Ivar. **UML, Guia do Usuário**. 2. Ed. São Paulo: Editora Campus, 2006

DAVID, Marcio Frayze. **Programação Orientada a Objetos: uma introdução**. 2007. Disponível em <http://www.hardware.com.br/artigos/programacao-orientada-objetos/>. Acesso em: 27 out. 2011.

CREATIVE COMMONS. **Ambiente Cliente/Servidor**. 2009. Disponível em <http://pt.kioskea.net/contents/cs/csintro.php3>. Acesso em: 14 nov. 2012.

DEITEL, Harvey. **Java, Como Programar**. 8. Ed. São Paulo: Editora Pearson, 2009

DIA. Disponível em: <http://live.gnome.org/Dia>. Acesso em: 09 dez. 2011.

DREAMSTIME: GOAL POSTS ON AMERICAN FOOTBALL FIELD. Disponível em: <http://www.dreamstime.com/stock-image-goal-posts-on-american-football-field-image14428401>. Aceso em: 09 dez.2011.

FERREIRA, Aurélio B. de Hollanda. **Novo Dicionário da Língua Portuguesa**. 2. ed. Rio de Janeiro: Nova Fronteira, 1986. 1838 p.

FIT FOR FOOTBALL. Disponível em: <http://www.yofootball.co.uk/football/fit-for-football.php>. Acesso em: 09 dez. 2011.

FRARI, Douglas. **Introdução a programação para celulares com suporte a Java ME**. 2007. Disponível em: <http://profdouglas.blogspot.com/2007/02/introduo-programao-para-celulares-com.html>. Acesso em: 30 nov.2011.

FURLAN, José Davi. **Modelagem de Objetos através da UML – Unified Modeling Language**. São Paulo: Makron Books, 1998.

HORIZONTAL JUMP. Disponível em: <http://mediagallery.usatoday.com/NFL-combine-2010/G1465,A6421>. Acesso em: 09 dez. 2011.

LANDONHOWELL. Disponível em: <http://landonhowell.com/index.php/category/southern-miss/page/2/landonhowell.com>. Acesso em: 09 dez. 2011.

LUGON, Priscila T.; ROSSATO, Thiago. **Floggy: Framework de persistência para J2ME/MIDP**. 2005. 75f. Trabalho de Conclusão(Bacharelado em Sistemas de Informação), Universidade Federal de Santa Catarina, 2005. Disponível em: <http://projetos.inf.ufsc.br/arquivos_projetos/projeto_284/Monografia_Floggy_Priscila_Thiago.pdf>. Acesso em: 09 dez.2011.

MACORATTI, José Carlos. **Diagramação de Software – DFD**. 2010. Disponível em <http://www.macoratti.net/vb_dfd1.htm>. Acessado em 07 set. 2011.

MUNCHOW, John W. **Core J2ME Tecnologia & MIDP**. 1 Ed. São Paulo: Editora Pearson, 2004

PATTA, Andrea Pivoto. **Armazenamento de dados através do RMS(Record Management System)**. Disponível em: <http://www.devmedia.com.br/post-155-Armazenamento-de-dados-atraves-do-RMS--Record-Management-System.html>. Acesso em 09 dez.2011.

PRESSMAN, Roger. **Engenharia de Software: Uma abordagem prática**. 1 Ed. Rio de Janeiro: Editora McGraw-Hill, 2006.

ROBERTO, Rogério Fernandes. **Diagrama de Fluxo de Dados**. Disponível em <http://www.professorgersonborges.com.br/site/cursos/desenvolvimento_sistemas/DFD.pdf>. Acessado em 21 out. 2011.

SCHACH, Stephen R.. **Engenharia de software: os paradigmas clássico e orientado a objeto**. 1 Ed. Rio de Janeiro: Editora McGraw-Hill, 2009.

SOMMERVILLE, Ian. **Engenharia de Software**. 8. Ed. São Paulo: Editora Pearson, 2007

SYSTEMSCAPE. **Introdução Java J2ME**. 2009. Disponível em <http://systemscape.wordpress.com/2010/01/17/introducao-java-mej2me/>. Acessado em: 14 set. 2011.

THE THREE CONE SHUFFLE. Disponível em: <http://www.theclevelandfan.com/cleveland-browns/1-browns-archive/7756-the-three-cone-shuffle>. Acesso em: 09 dez. 2011.

THE NFL SCOUTING COMBINE. 2007. Disponível em: <http://www.sportznutz.com/nfl/draft/combine.htm>. Acesso em: 09 dez. 2011.

TOMCAT. Disponível em: http://tomcat.apache.org Acesso em: 30 nov. 2011.

WEBYOG. **About Us**. 2011. Disponível em <www.webyog.com/en/about_webyog.php>. Acesso em: 12 nov. 2011.