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

LEANDRO FANTINEL
WAGNER PARISOTO

SISTEMA DE ACOMPANHAMENTO DE ATIVIDADES FÍSICAS - DROIDFITNESS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2012

LEANDRO FANTINEL

WAGNER PARISOTO

SISTEMA DE ACOMPANHAMENTO DE ATIVIDADES FÍSICAS - DROIDFITNESS

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 – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Luis C. F. Bueno, M.Sc..

PATO BRANCO

2012

ATA Nº: **203**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DOS ALUNOS **WAGNER PARISOTO e LEANDRO FANTINEL.**

Às 13:40 hrs do dia 26 de outubro de 2012, Bloco S da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Luís Carlos Ferreira Bueno (Orientador), Eliane Maria de Bortoli Fávero (Convidada) e Wendel Goes Pedrozo (Convidado), para avaliar o Trabalho de Diplomação do aluno Wagner Parisoto, matrícula 1066978 e do aluno Leandro Fantinel, matrícula 659525, sob o título **Implementação de um Sistema para Acompanhamento de Atividades Físicas**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. 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 15:05 hrs foi encerrada a sessão.

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

Profa. Eliane Maria de Bortoli Fávero, M.Sc.
Convidada

Prof. Wendel Goes Pedrozo, M.Sc.
Convidado

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

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

Dedico esse trabalho a meus pais,
que pacientemente acompanharam todas as dificuldades
para concluir esta graduação. Principalmente a meu Pai
que infelizmente devido a fragilidade da vida
não pude lhe dar o orgulho de ver em vida um filho formado.

Dedico também a todos os colegas que sei também
sentirem-se remando sozinhos a deriva do conhecimento.

Agradeço a todos os amigos que nunca duvidaram
e nos quais sempre busquei forças para não desistir.

Em bora nem todas as experiências tenham sido agradáveis
nesta trajetória acadêmica, confesso que fiquei por muito tempo
olhando para os caminhos e acabei escolhendo o menos percorrido;
porém, hoje vejo que isso fez toda a diferença.

Aos professores desejo que a esperança deixe se ser o não
que se insiste em maquiar de sim.

Aos Mestres Omero, Geri e Gustavo meu eterno respeito e agradecimento
pela inefável dedicação a licenciatura, pelo profissionalismo
e pela maneira impar com que transformam o ensinar em arte.
Obrigado por dividirem comigo conhecimentos
que carregarei por toda a vida.
(Leandro Fantinel)

A todos os professores que contribuíram significamente na
obtenção do conhecimento em todo o período de convivência
na universidade. Especialmente ao professor orientador Luis Bueno, por
proporcionar este trabalho.
(Wagner Parisoto)

… “Quero dizer às pessoas que nada foi em vão.
Que o amor existe, que vale a pena se doar às amizades e às pessoas,
que a vida é bela sim, que eu sempre dei o melhor de mim...
e que valeu a pena.”
(Mário Quinta)

RESUMO

FANTINEL, Leandro e PARISOTO, Wagner. **Implementação do Sistema de Acompanhamento de Atividades Físicas - DROIDFITNESS**. Pato Branco, 2012. 94 f 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, 2012.

Este Trabalho de Conclusão de Curso, apresenta a abordagem do desenvolvimento de um sistema cliente-servidor para oferecer suporte no acompanhamento de atividades físicas em academias de ginástica. O sistema abstrai a tradicional ficha de papel, para uma aplicação móvel (A*ndroid*) que é alimentado através de umaii aplicação de gerenciamento (*desktop*) e a troca de dados ocorre por meio de uma conexão de rede. Ambas as aplicações utilizam as tecnologias *Java* como linguagem de programação e bancos de dados orientados a objetos para persistência dos dados.

Palavras-chave: Android. Java. Neodatis. Musculação. *Fitness*.

ABSTRACT

FANTINEL, Leandro and PARISOTO, Wagner. **System Implementation Monitoring Physical Activities - DROIDFITNESS.** Pato Branco, 2012, 94 p. Completion of course work. Degree in Technology Analysis and Systems Development. Federal Technological University of Paraná, Campus Pato Branco. Pato Branco, 2012.

This Labor Completion of course, presents the approach of developing a client-server system to support monitoring of physical activities in gyms. The system abstracts the traditional paper form, an application for mobile (Android) that is fed through a management application (desktop) and data exchange occurs via a network connection. Both applications use technologies like Java programming language and databases to object-oriented data persistence.

Keywords: Android. Java. Neodatis. Bodybuilding. Fitness.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta uma descrição do estudo do negócio para o qual o software foi desenvolvido, bem como funcionalidades e características gerais, modeladas no estágio supervisionado anteriormente realizado (FANTINEL, 2012).

1.1 Considerações Iniciais

O sistema tem por objetivo suprir a necessidade encontrada pelos praticantes de atividades físicas, academias de ginástica e clinicas de *fitness*, sendo todo e qualquer tipo de atividade que por meio de equipamentos adequados e acompanhamento, realizado por meio de um profissional, possa oferecer bem estar e saúde ao praticante.

A proposta visa oferecer uma solução eficiente no que diz respeito ao suporte e acompanhamento à prática de atividades físicas.

Cada vez mais pessoas procuram manter uma boa forma e saúde corporal através da prática de exercícios em centros de musculação e ginástica, com instrutores e equipamentos especializados. O Brasil já é o segundo país do mundo em número de academias, conta com aproximadamente 16.952 mil academias sendo que a maior parte se concentra nos estados de São Paulo e Santa Catarina e metade desse público pertence a classe C, que hoje conta com um poder aquisitivo cada vez maior (ESTADÃO, 2012).

Com a informatização atingindo todos os setores comerciais, os centros de ginástica também são tomados pela necessidade competitiva de tornar a tarefa de prescrição dos exercícios ao aluno mais ágil e flexível. Se torna ágil pois através de um dispositivo móvel, permite que o aluno visualize seu treinamento de forma rápida e flexível, pois ele poderá acessar de qualquer lugar e ter seus dados atualizados a qualquer momento.

O *DroidFitness* permite o instrutor de uma academia realizar a prescrição dos exercícios e atividades para seus alunos através de um dispositivo móvel, de modo que o aluno tenha em mãos e em tempo real a sua lista de atividades, tanto *in loco* como em viagens. Ainda permitirá ao instrutor acompanhar a evolução de cada aluno de acordo com o treinamento.

1.2 Objetivos

A seguir são apresentados os objetivos gerais e específicos abordados neste trabalho.

1.2.1 Objetivo Geral

Desenvolver um Sistema para fornecer suporte de acompanhamento para a realização de atividades físicas, principalmente em academias de ginástica ou nos mais diversificados tipos de estabelecimentos que desenvolvem algum tipo de atividade física.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Permitir ao instrutor criar e manter séries de exercícios personalizadas para cada aluno;
- Possibilitar ao Instrutor acompanhar de forma específica cada aluno do estabelecimento;
- Permitir ao aluno consultar e acompanhar suas séries de exercícios;
- Oferecer ao aluno a portabilidade de sua ficha de treino, para realizar seus exercícios diários em algum clube, hotel, ou em uma academia fora de sua cidade.
- Permitir ao aluno visualizar estatísticas referentes a avaliações físicas realizadas anteriormente pelo seu instrutor.

1.3 Justificativa

As fichas de acompanhamento convencionais estão obsoletas. A metodologia de uso das atuais fichas de papel, não é mais adequada, pois com o manuseio, as fichas se desgastam, sujam, rasgam e as informações ficam prejudicadas, isso somado ao próprio inconveniente de carregá-las, leva ao seu desuso. Devido a isso os alunos acabam por realizar séries aleatórias por conta própria, normalmente com cargas fora do especificado pelo instrutor. Ainda, a utilização das fichas de papel dependem totalmente de o aluno solicitar que seja elaborada uma nova lista de exercícios para tê-la atualizada, o que também prejudica a qualidade do exercício e muitas vezes a própria saúde do praticante, bem como a finalidade da atividade física como instrumento de manutenção da saúde.

Recentes matérias veiculadas em rede nacional já mostram a informatização das academias: esteiras com TV, reprodutores de música e acesso a Internet, além da customização e individualização do treinamento são as mais fortes tendências desse segmento.

1.4 Estrutura do Trabalho

Este é o primeiro capítulo e apresenta as considerações iniciais, objetivos e justificativa deste trabalho.

No Capítulo 2 está a revisão bibliográfica dos autores e obras que dão fundamento aos métodos e conceitos utilizados no estudo.

No capítulo 3 – Materiais e métodos – são apresentadas as tecnologias utilizadas, ferramentas, modelos e metodologias aplicadas.

O Capítulo 4 contêm a descrição do cenário para o qual o sistema foi modelado, bem como as funcionalidades identificadas para que o sistema atenda as necessidades para as quais está sendo proposto.

No Capítulo 5 é apresentada a análise dos resultados, comentando cada módulo do sistema, suas funcionalidades, características, descrição do processo de codificação, as dificuldades encontradas e as adequações que foram necessárias

para superá-las.

O Capítulo 6 contem a conclusão e parecer final do desenvolvimento.

Por fim no Capítulo 7 estão as referências bibliográficas utilizadas para obter embasamento nas técnicas e soluções implementadas.

2 REFERENCIAL TEÓRICO

Neste capítulo é apresentado o referencial teórico sobre atividades físicas e suas técnicas, modelagem (UML), tecnologias de desenvolvimento como Java e Android e bancos de dados orientado a objetos, também é descrito os principais conceitos a respeito de sistemas distribuídos.

2.1 Atividade Física

A atividade física é consensualmente definida como todo e qualquer movimento corporal produzido pela contração músculo esquelético resultando num gasto energético. Já o conceito de exercício físico definido por Caspersen *et al*. é mais especificamente, uma atividade repetida e estruturada que tem em vista a manutenção ou melhoria da aptidão física (CASPERSEN; POWELL; CHRISTENSON, 1985).

A atividade física, nas suas respectivas vertentes, como a utilitária (andar, subir e descer escadas, jardinagem), a educação física e o desporto, executados de uma forma moderada, é favorável à manutenção da saúde e ajuda também na prevenção das doenças (NUNES, 1999).

Segundo ACSM (1997) a prática de atividades física, em todas as idades, é essencial para uma boa saúde, sendo indispensável para o controle do peso e do balanço energético.

Esta prática, durante a infância, apresenta uma série de benefícios, possibilitando um crescimento saudável, bem-estar psicológico e diminuição de alguns fatores de risco, tais como a hipertensão, o colesterol elevado e obesidade (CLARK, 1988).

A atividade física quando for controlada por profissionais da Educação Física e elabora de forma correta, está diretamente associada a melhora da saúde e do condicionamento físico do praticante, contribuindo para a redução do stress da ansiedade, ajuda o organismo se tornar mais tolerável a infecções e diminuição de de dores provocadas pelo sedentarismo e postura. A atividade física quando realizada de forma contínua e programada resulta no ganho de força física, aumento

da capacidade cardiorrespiratória, da flexibilidade.

2.2 Prescrição de Atividades Físicas

Segundo ACSM (2002), entende-se por prescrição, “todo o processo através do qual o estabelecimento de recomendações para um regime de atividade física é concebido de forma sistemática e individualizada”. Conforme expõe Moreira (2012), a prescrição do exercício é caracterizada por um processo de preparação sistemática do organismo, onde resultam modificações morfológicas e funcionais, caracterizadoras do estado de condição física do indivíduo. O objetivo de uma prescrição de exercício depende das características do indivíduo e do seu objetivo.

Na generalidade pretende-se a promoção da saúde através da prevenção dos fatores de risco de doenças de causa hipocinética, melhoria da condição física e a ação pedagógica, informando sobre os benefícios da atividade física e sobre a forma correta e segura de realizar exercício. A prescrição deverá ser individualizada para cada sujeito, embora existam elementos básicos comuns a todas as prescrições do exercício. Esses elementos básicos são: a intensidade, o volume e a frequência do treino.

2.3 Academia Desportiva

Segundo o dicionário da língua portuguesa Houaiss (2012), academia tem por significado “Estabelecimento de instrução onde se ministram artes, ofícios ou práticas esportivas” (HOUAISS, 2012). Uma academia desportiva, ginásio ou apenas academia é um local (que pode ser fechado ou aberto) destinado ao ensino e à prática de esportes (natação, musculação e/ou ginástica com exercícios aeróbicos ou anaeróbios), e dotados de equipamento específico para o trabalho do corpo humano.

2.4 Orientação a Objetos

A orientação a objetos é o paradigma de programação predominante atualmente, e vem substituindo a programação procedural, que foi criada no inicio da década de 1960. O termo orientação a objetos pressupõe uma organização de software em termos de coleção de objetos discretos incorporando estrutura e comportamento próprios. Esta abordagem de organização é essencialmente diferente do desenvolvimento tradicional de software, onde estruturas de dados e rotinas são desenvolvidas de forma fracamente acopladas (MATTOS, 2007 ; p.11).

A Orientação a Objetos consiste em conhecer um Sistema Computacional como um todo, formado por objetos que se relacionam entre si. Esse enfoque pode ser tanto aplicado na análise de sistemas quanto à programação, e essa é umas das principais vantagens da Orientação a Objetos: a mesma metodologia serve tanto para definição lógica do sistema quanto para a sua implementação.

A principal vantagem da Orientação a Objetos consiste em reunir em uma mesma estrutura os dados e os processos que são executados sobre esses dados, permitindo assim um grau maior de organização e simplicidade do programa. Quando uma estrutura dos objetos é organizada de forma eficiente, permite que seja incluído módulos adicionais no sistema que reutilizem as funcionalidades anteriormente codificadas sem necessidade de reprogramação, essa ideia é mais conhecida como a reutilização de código-fonte.

2.4.1 Princípios Básicos da Orientação a Objetos

A Orientação a Objetos compreendem os princípios conforme descritos a seguir.

2.4.1.1 Classes

Uma classe é um gabarito para a definição de objetos. Através da de definição de uma classe, descreve-se que propriedades, ou atributos o objeto terá (RICARTE, 2001 ; p.3).

Ainda conforme explica Ricarte, além da especificação de atributos, a definição de uma classe descreve também o comportamento de objetos da classe, ou seja, que funcionalidades podem ser aplicadas a objetos da classe. Essas funcionalidades são descritas através de métodos. Um método é o equivalente a um procedimento ou função, com a restrição que ele manipula apenas as suas variáveis locais e os atributos que foram definidos para a classe.

Uma classe define as características de um grupo de objetos, isto é, define como serão as instâncias pertencentes a ela. Assim, uma classe especifica quais serão os atributos que a ela pertence, bem como os serviços que ela pode executar. Um serviço também pode ser denominado Operação. Uma classe consiste em uma entidade que descreve diversos atributos comuns aquela representação (BORATTI, 2007 ; p.30).

Outros possíveis membros de uma classe são:

- **Construtores:** definem o comportamento no momento da criação de um objeto de uma classe.
- **Destrutor:** define o comportamento no momento da destruição do objeto de uma classe.
- **Propriedades:** define o acesso a um estado do objeto.
- **Eventos:** define um ponto em que o objeto pode chamar outros procedimentos de acordo com seu comportamento e estado interno.

2.4.1.2 Objetos

Um objeto, na vida real, é qualquer coisa a qual se pode dar um nome.

Um objeto, em programação orientada a objetos, é uma instância (ou seja, um exemplar) de uma classe (DEBONI, 2003 ; p.34). Tomando como exemplo: carro é um exemplo de veículo, ou pode-se dizer que carro é uma instancia de veículo, isto pode ser representado na Orientação a Objetos como uma classe chamada Veículo e um objeto do tipo Veículo chamado de Carro.

Um objeto é capaz de armazenar estados através de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos. O conjunto de valores dos atributos de um determinado objeto é chamado de estado.

2.4.1.3 Mensagens

A comunicação entre os objetos é feita por meio de mensagens. Um objeto que desejar uma informação de outros objetos deve solicitar às funções desses objetos, na forma de mensagens, que responda a seu pedido. Como um sistema é formado por um conjunto de objetos, o processamento do sistema é obtido mediante troca de mensagens entre dois objetos (DEBONI, 2003 ; p.32).

Uma mensagem é a chamada de uma função de um objeto. A informação é transmitida para a função por meio da passagem de parâmetros no momento da sua invocação e em seguida a função responde por meio de parâmetros de retorno, também chamada de resposta.

2.4.1.4 Abstração

Segundo o dicionário da língua portuguesa, a palavra abstração tem por definição, “Ato de separar mentalmente um ou mais elementos de uma totalidade

complexa (coisa, representação ou fato)". Na Orientação a Objetos, a abstração visa dividir um problema maior em problemas menores e menos complexos, com o objetivo de resolver o problema inteiro por resolução em partes (COAD, 1991; p.11).

Em desenvolvimento de sistemas a abstração, de um modo geral tenta representar na aplicação somente aquilo que será usado. Isola-se os objetos que do ambiente complexo se quer representar e nesses objetos são representados somente as características que são relevantes para o problema em questão. Por exemplo: Todo o veículo possui placa, no entanto para um sistema de controle de caixa, essa informação é irrelevante, por tanto ela não será representada.

2.4.1.5 Encapsulamento

O Encapsulamento, segundo Deboni (2003, p.29) é a principal característica para a identificação de um objeto. O princípio por trás dessa ideia é que o objeto possua todos os atributos e funções necessários para a sua existência. Ainda conforme explica Deboni, o encapsulamento protege os dados de um objeto ao acesso descontrolado por outros objetos. O acesso é realizado por intermédio de mensagens trocadas entre objetos, como as mensagens passam por uma função do objeto, antes de acessar os dados, esse acesso é controlado e os dados protegidos.

O Encapsulamento disponibiliza toda as funcionalidades que um objeto possui sem que seja necessário saber como ele funciona internamente e também como são armazenados os dados e como são recuperados. Pode-se exemplificar da seguinte forma: Quando uma pessoa faz uma ligação usando um telefone, o usuário não precisa ter em mente como a empresa telefônica realiza o procedimento, ou por onde os fios do telefone estão passando, é preciso apenas que ele saiba o número do telefone em que se deseja contatar e assim a ligação será estabelecida.

> **Encapsulamento, em programação orientada a objetos significa separar o programa em partes, o mais isoladas possível. A ideia é tornar o software mais flexível, fácil de modificar e de criar novas implementações (SCOTT, 2006 ; p.481) .**

Trazendo o exemplo do telefone, para a Orientação a Objetos e associando aos objetos, pode-se entender que, o encapsulamento protege o acesso direto (referência) aos atributos de uma instância fora da classe onde estes foram declarados. Esta proteção consiste em se usar modificadores de acesso mais restritivos sobre os atributos definidos na classe. Depois devem ser criados métodos para manipular de forma indireta os atributos da classe.

Pode-se dizer também que dessa forma, os objetos agem como uma “caixa preta”, ou seja, não é preciso saber como eles funcionam internamente, o que realmente importa para haver a interação entre dois objetos, é que um conheça o conjunto de operações contidos no outro objeto.

2.4.1.6 Herança

A herança é uma das principais características da orientação a objetos e está intimamente associada ao conceito de agrupamento dos objetos, segundo um conjunto de propriedades comuns. A orientação a objetos permite criar uma relação entre classes de objetos, de modo que a classe pode ser gera a partir de outra classe, herdando dela as suas propriedades (DEBONI, 2003 ; p.36).

Tomando como exemplo uma classe chamada Veículo, pode-se dizer que é uma generalização, já uma classe Caminhão pode ser uma especialização de Veículo sendo capaz de herdar os Atributos e Serviços de “Veículos”. De tal forma “Caminhão” pode conter Atributos incluídos como “Número de eixos” ou “Número de rodas”, e Serviços que como “Limite de peso”, que por exemplo, para um veículo “Carro de passeio” se torna irrelevante.

2.4.1.7 Polimorfismo

Segundo Deboni (2003, p.37), Polimorfismo é a propriedade de uma ou mais classes responderem a mesma mensagem, cada uma de uma forma diferente. Numa linguagem orientada a objetos, uma referência polimórfica é tal que, no

decorrer do desenvolvimento do software, refere-se a mais de uma classe. Desta forma é possível explorar similaridades entre diferentes classes de objetos. Este conceito é útil para distinguir mensagens de um método. Um objeto emissor envia uma mensagem, se o objeto receptor implementa um método com a mesma assinatura, ele poderá respondê-la. Diferentes respostas serão possíveis, dependendo de como os métodos dos receptores estão implementados.

Tomando como exemplo uma classe chamada “OperacaoMatematica” e imaginando que em matemática existem diversas operações (somar, dividir, subtrair, multiplicar). Utilizando o polimorfismo, cada operação pode ser dividida em subclasses, dessa forma, a assinatura do método será sempre o mesma, a única parte que muda será a estrutura do código, que equivale a cada operação implementada devolvendo como parâmetro de resposta o resultado equivalente.

2.4.2 Análise Orientada a Objetos

Baseado na ideia da Orientação a Objetos, o processo de análise consiste em identificar quais são os objetos existentes na situação que se pretende automatizar e quais serão os atributos e as ações (mensagens) que permitem que eles trabalhem juntos (CORREIA; TAFNER, 2006).

A Análise Orientada a Objetos parte do paradigma de que o mundo é formado de objetos e de que desenvolver um sistema nada mais é que criar uma simulação dos objetos e de seu comportamento. Durante a análise orientada o objetos, há uma enfase em encontrar e descrever os objetos, ou conceitos, no domínio do problema. Por exemplo, no caso de um sistema de informação de voo, alguns dos conceitos incluem avião, voo e piloto. Durante o projeto Orientado a Objetos, há uma ênfase na definição dos objetos de software e como eles colaboram para a satisfação dos requisitos. Por exemplo, um objeto de software avião pode ter um atributo número de capacidade de passageiros, e um método obter histórico do voo.

2.4.3 Programação Orientada a Objetos

O conceito de Programação Orientada a Objetos (POO), tem suas raízes na linguagem SIMULA 67 e *Smaltalk*, desenvolvidas por volta de 1980. Muitos consideram a *Smaltalk* como sendo a única linguagem de programação puramente orientada a objetos. Uma Linguagem com essas características deve oferecer três recursos chaves: tipos de dados abstratos, herança e um tipo particular de vinculação dinâmica (SEBESTA, 2003 ; p.443).

Ainda conforme explica Sebesta (2003), Programação Orientada a Objetos é essencialmente uma aplicação do princípio da abstração de tipos de dados. Na Programação Orientada a Objetos, os atributos comuns de uma coleção de tipos de dados abstratos similares são fatorados e colocados em um novo tipo. Os membros da coleção herdam as partes comuns deste.

A programação Orientada a Objetos consiste em utilizar a estrutura de dados que simulem o comportamento dos objetos, facilitando a programação pelo uso de uma metodologia mais unificada para a análise e a programação. Os mesmos objetos que foram identificados pelo analista no seu diagrama podem ser implementados exatamente como foram projetados, sem a necessidade de fazer uma tradução dos diagramas da análise para estruturas de dados e programação. A Programação Orientada a Objetos impede que o programador precise escrever um código similar infinitas vezes, gastando tempo e recursos programando rotinas que já foram criadas anteriormente. Resumindo, as principais vantagens da POO são o reuso de código fonte e a capacidade de expansão do mesmo.

2.5 *Unified Modeling Language* (UML)

Segundo Booch *et al.* (2005, p.13), a *Unified Modeling Language* (UML) é uma linguagem-padrão para a elaboração da estrutura de projetos de software Orientados a objeto. Ela poderá ser empregada para a visualização, a especificação, a construção e a documentação de artefatos que façam uso de sistemas complexos de software.

> **A UML é uma linguagem visual para modelar sistemas orientados a objetos, ou seja, é uma linguagem que define elementos gráficos (visuais) que podem ser utilizados para a modelagem de sistemas (BEZERRA, 2007 ; p.15).**

A UML pode fazer parte de um todos os processos do desenvolvimento de software. É independente do processo, apesar de ser perfeitamente utilizadas em processo orientado a casos de usos, centrado na arquitetura, interativo e incremental.

A UML suporta o desenvolvimento iterativo e incremental. Desenvolvimento iterativo e incremental é o processo de desenvolvimento de sistemas em pequenos passos. Uma iteração é um laço de desenvolvimento que resulta na liberação de um subconjunto de produtos que evolui até o produto final percorrendo as atividades de: Análise de requisitos, Análise, Projeto, Implementação, Teste.

Os objetivos da UML são (BOOCK, 2006 ; p.14):

- **Visualizar:** a UML permite criar modelos visuais, como diagramas e protótipos. Sua proposta facilita a comunicação e faz com que os membros de um grupo tenham a mesma ideia do sistema.

- **Especificar:** a UML é considerada uma ferramenta poderosa para a especificação de diferentes aspectos arquiteturais e de uso de um sistema. Em particular, a UML atende todas as decisões importantes em termos de análise, projeto e implementação, que devem ser tomadas para o desenvolvimento e implantação de sistemas complexos de software.

- **Construir:** a UML não é considerada uma linguagem de programação, no entanto, seus modelos criados podem ser conectados nos mais variados tipos de linguagens ou mesmo em banco de dados, dessa forma a representação dos modelos do sistema podem ser convertidos em código fonte. De forma simplificada, a UML pode representar de forma muito mais precisa o que geralmente se representa de forma gráfica, enquanto a linguagem de programação melhor expressa os termos que são representados de formas textuais.

- **Documentar:** a UML é uma linguagem completa, ou seja, pode ser usada desde os marcos iniciais de um projeto até os testes finais e integração com o cliente. Dessa forma toda a produção desenvolvida na análise irá compreender uma documentação de crucial importância para a vida de um projeto, sendo utilizada nas etapas de codificação, dos testes e na implantação e para extrair os dados estatísticos relacionados ao projeto, como por exemplo, o tempo gasto em cada atividade e o mais importante: os custos do projeto.

2.5.1 Diagramas

A UML compõe diversos diagramas. Segundo Boock (2005, p.241), “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”. De acordo com Furlan (1998), os principais diagramas da UML são:

- **Diagrama de Caso de Uso:** casos de uso descrevem as funcionalidades do sistema percebida por atores externos. Um ator interage com o sistema podendo ser usuário, dispositivo ou outro sistema. Este diagrama permite que sejam representados os relacionamentos entre os casos de uso do sistema.

- **Diagrama de Classes:** mostra o conjunto de classes, e seus relacionamentos. Os Diagramas de classes da UML, são um superconjunto dos diagramas de entidade-relacionamento, uma ferramenta básica de modelagem para o projeto lógico de bancos de dados.

- **Diagrama de Objetos:** este diagrama é uma variação do diagrama de classes e utiliza quase a mesma notação. A diferença é que o diagrama de objetos mostra os objetos que foram instanciados das classes. Os

diagramas de objetos não são tão importantes como os diagramas de classes, mas eles são muito úteis para exemplificar diagramas complexos de classes ajudando muito em sua compreensão. Diagramas de objetos também são usados como parte dos diagramas de colaboração, onde a colaboração dinâmica entre os objetos do sistema são mostrados.

- **Diagrama de Sequência:** mostra a colaboração dinâmica entre os vários objetos de um sistema. O mais importante aspecto deste diagrama é que a partir dele percebe-se a sequência de mensagens enviadas entre os objetos. Ele mostra a interação entre os objetos, alguma coisa que acontecerá em um ponto específico da execução do sistema.

- **Diagrama de Colaboração:** mostra de maneira semelhante ao diagrama de sequencia, a colaboração dinâmica entre os objetos. Normalmente pode-se escolher entre utilizar o diagrama de colaboração ou o diagrama de sequência.
No diagrama de colaboração, além de mostrar a troca de mensagens entre os objetos, percebe-se também os objetos com os seus relacionamentos.

- **Diagramas de Estados:** o diagrama de estado é tipicamente um complemento para a descrição das classes. Este diagrama mostra todos os estados possíveis que objetos de uma certa classe podem se encontrar e mostra também quais são os eventos do sistemas que provocam tais mudanças. Os diagramas de estado não são escritos para todas as classes de um sistema, mas apenas para aquelas que possuem um número definido de estados conhecidos e onde o comportamento das classes é afetado e modificado pelos diferentes estados.

- **Diagrama de Atividades:** é considerado um diagrama de estado especial onde a maioria dos estados é estado de ações e a maioria das transições é ativada por conclusão das ações nos estados de origem. O diagrama de atividade é uma variação do diagrama de estado e possui um

propósito um pouco diferente do diagrama de estado, que é o de capturar ações (trabalho e atividades que serão executados) e seus resultados em termos das mudanças de estados dos objetos.

- **Diagramas de Componentes:** O diagrama de componente e o de execução são diagramas que mostram o sistema por um lado funcional, expondo as relações entre seus componentes e a organização de seus módulos durante sua execução. O diagrama de componente descreve os componentes de software e suas dependências entre si, representando a estrutura do código gerado. Os componentes são a implementação na arquitetura física dos conceitos e da funcionalidade definidos na arquitetura lógica (classes, objetos e seus relacionamentos). Eles são tipicamente os arquivos implementados no ambiente de desenvolvimento.

- **Diagrama de Implantação:** procura demonstrar a arquitetura física do hardware e do software no sistema. Pode mostrar os atuais computadores e periféricos, juntamente com as conexões que eles estabelecem entre si e pode mostrar também os tipos de conexões entre esses computadores e periféricos. Especifica-se também os componentes executáveis e objetos que são alocados para mostrar quais unidades de software são executados e em que destes computadores são executados.

- **Diagrama de Módulos:** o diagrama de módulos, definido pela UML descreve pacotes, ou pedaços do sistema divididos em agrupamentos lógicos, mostrando as dependências entre estes, ou seja, pacotes podem depender de outros pacotes.

Como visto, cada diagrama da UML permite obter uma visão diferente do sistema de maneira geral, permitindo realizar uma análise consistente.

2.6 Banco de Dados Orientado a Objetos

Os Bancos de Dados Orientados a Objetos (BDOO) surgiram na década de 80, visando aumentar a produtividade do desenvolvimento de software (NASSU; SETZER, 1999).

Um BDOO é um banco em que cada informação é armazenada na forma de objetos, que só podem ser manipuladas através de métodos definidos pela classe que está o objeto. De acordo com Elmasri e Navathe (2005), existem pelo menos dois fatores que levam a adoção desse modelo, são eles:

- Nos Banco de Dados relacionais se torna difícil trabalhar com dados complexos, já que todas as informações são armazenadas por meio de tabelas relacionadas e a junção das informações é considerada relativamente complexa;

- Aplicações são construídas em linguagens orientadas a objetos e o código precisa ser traduzido para uma linguagem que o modelo de banco de dados relacional entenda, o que torna essa tarefa muito tediosa. Essa tarefa também é conhecida como “perda por resistência”.

O modelo Orientado a Objetos (OO) ganhou espaço nas áreas como banco de dados espaciais, telecomunicações, e nas áreas científicas. Isso porque essa tecnologia oferece aumento de produtividade, segurança e facilidade de manutenção. Como objetos são modulares, mudanças podem ser feitas internamente, sem afetar outras partes do programa.

Segundo Khoshafian (1999, p.335), “devido a seus atributos orientados a objetos, os bancos de dados orientados a objetos oferecem modelos diretos e intuitivos para o desenvolvimento de aplicações.”

Conforme explica Nassu e Setzer (1999, p.75), o modelo utilizado para representação da estrutura de um banco orientado a objetos é *Object Entity-Relationship* (OER). “Esse modelo de dados é uma extensão do Modelo de Entidades e Relacionamentos”.

Todos os desenvolvedores das linguagens orientadas a objetos sabem das dificuldades de passar um modelo orientado a objetos para uma persistência relacional. Grande parte do tempo de desenvolvimento de um software é perdida na fase de mapeamento das classes. Utilizando um banco de dados orientado a objetos é possível eliminar ferramentas e códigos para o mapeamento objeto relacional e aproveitar os benefícios do paradigma orientado a objetos sem estar preso pelo banco de dados. Nesse cenário, um banco de dados relacional[1] não traz nenhuma vantagem. Talvez a escolha de um modelo Orientado a Objeto seja uma evolução natural.

Mapear o esquema de um banco de dados lógico para uma banco de dados orientado a objetos é algo simples, pois até heranças complexas podem ser convertidas em persistentes diretamente (BOOCK, 2005). Basicamente o modelo relacional deixa de existir e os objetos do sistema são persistidos da forma que foram criados, sem a necessidade de uma conversão.

2.6.1 Características

Umas das características dos sistemas Orientados a Objetos (OO) é ocultar informação e tipos abstratos de dados, sendo que é muito complicado aplicar esse modelo na prática. Por exemplo, nos sistemas atuais para uma consulta em uma determinada tabela é necessário saber todos os atributos da tabela, para formar a consulta. Em um sistema OO, que preza pelo encapsulamento nem toda tabela pode enxergar a outra, o que dificultaria muito as consultas (NASSU, 1999).

A ideia do encapsulamento em um Banco de Dados Orientado a Objetos (BD OO), já que não dá para ser aplicado a rigor, é pelo menos tratar o comportamento do objeto com funções pré-definidas. Por exemplo, inserir, excluir, atualizar etc. Ou seja, a estrutura interna do objeto é escondida, e os usuários externos só conhecem a interface do tipo de objeto como os argumentos (parâmetros) de cada operação. Então a implementação é oculta para usuários externos que está incluído a definição

1 Banco de dados Relacionais, são banco de dados onde sua estrutura é baseada em registros relacionados e organizados em tabelas. Essas relações tornam os registros integrados (FERRARI, 2007);

da estrutura interna, de dados do objeto e a implementação das operações que acessam essas estruturas. Enfim, o Banco de Dados Orientado a Objetos propõe o seguinte dividir a estrutura do objeto em partes visíveis e ocultas. Então, para operações que exigem atualização da base de dados, torna se oculta, e para operações que exige consultas, torna-se visível. (ELMASRI; NAVATHE, 2005).

Grande parte do modelo OO separa claramente o que é objeto persistente, e objeto transiente. Por exemplo, quando é realizada uma consulta, é carregada uma lista de objetos numa classe transiente (temporária), o sistema pode manipular os dados nessa classe e assim que forem feitas as manipulações necessárias elas deixam de existir.

Uma das grandes vantagens de um BDOO é que ele permite salvar objetos grandes e depois obter a recuperação facilmente desses grandes objetos como texto longos, imagens etc. Eles são considerados não estruturados porque o BD não conhece a sua estrutura. A aplicação pode utilizar várias funções para manipular esses objetos. E o mais importante é que o BD não conhece essas funções, mas através de técnicas oferecidas por ele é capaz de reconhecer esses objetos e buscá-los no banco de dados.

É importante frisar que BDOO não é capaz de processar diretamente condições de seleções e outras operações desses objetos. É necessário que esses dados sejam passados para o BD para que ele possa saber tratar os objetos corretamente. Por exemplo, considerando objetos que são imagens. Supondo que a aplicação precise selecionar a partir de uma coleção de tais objetos somente aqueles que incluem certo padrão. Nesse caso, o usuário deve fornecer o programa de reconhecimento do padrão, como um método em objetos do tipo imagens. O BDOO recupera, então, um objeto do banco de dados e aplica nele o método para o reconhecimento do padrão para determinar se o objeto adere ao padrão desejado.

2.7 Sistemas Distribuídos

Sistemas Distribuídos consistem em uma coleção de computadores independentes que se apresentam ao usuário como um sistema único e coerente.

Esta definição tem dois aspectos. Primeiro trata do *hardware*: as máquinas são autônomas. Segundo trata do software: os usuários pensam do sistema como sendo um único computador, porém ambos são essenciais (TANENBAUM; STEEN, 2006, p.2).

Pode-se dizer também que um Sistema Distribuído é um sistema onde os componentes de hardware e software estão localizados em computadores interligados por uma rede, comunicam e coordenam suas ações somente através da troca de mensagens (COULOURIS; DOLLIMORE; KINDBERG, 2007, p.1).

O motivo que se leva a construir sistemas distribuídos vem da necessidade de compartilhar informações ou recursos de hardware e software pela rede, permitindo assim que esses recursos estejam disponíveis para qualquer lugar em qualquer dispositivo, desde que conectados por meio de uma rede física e através de procedimentos definidos de comunicação.

Conforme explica Coulouris *et al.* (2007, p.28) para a construção de sistemas distribuídos podem ser encontrados muitos desafios, como:

- **Heterogeneidade:** um mesmo sistema distribuído pode ser construído para funcionar nos mais variados tipos de redes de computadores, sistemas operacionais, hardware e linguagens de programação. Os protocolos de comunicação da Internet mascaram as diferenças existentes nas redes e o *middleware*[2] pode cuidar das outras diferenças.

- **Sistemas abertos:** os sistemas distribuídos devem ser extensíveis, ou seja, permitir que outros componentes de software sejam acoplados ao sistema, no entanto como os componentes podem ser escritos por diversos programadores, a integração se torna um desafio real.

2 ***Middleware*** ou mediador, é uma camada de software cujo objetivo é mascarar a Heterogeneidade visa fornecer um modelo de programação conveniente para os programadores de aplicativos. É composto por um conjunto de processos ou objetos, em um grupo de computadores, que interagem entre si de forma a implementar a comunicação e oferecer suporte para compartilhamento de recursos a aplicativos distribuídos (COULOURIS; DOLLIMORE; KINDBERG, 2007);

- **Segurança:** a criptografia pode ser usada para proporcionar proteção adequada para os recursos compartilhados e para manter informações sigilosas em segredo, para quando são transmitidas por uma rede.

- **Escalabilidade:** um sistema distribuído é considerado escalável, quando consegue permanecer ativo de forma eficiente, possuindo um número considerável de recursos e usuários.

- **Concorrência:** a presença de múltiplos usuários em um sistema distribuído, possibilita que sejam requisitados acessos aos recursos de forma simultânea.

- **Transparência:** tem por objetivo tornar certos aspectos da distribuição abstratas para o programador de sistemas, para que ele se preocupe somente com os procedimentos que sua aplicação irá implementar. Por exemplo, ele não precisa estar preocupado com a sua localização, ou como as mensagens serão trocadas pela rede, se uma falha por alguma razão possa vir a ocorrer, ela poderá ser tratada como forma de exceção.

2.7.1 Soquetes

Soquete ou "*Socket*" no termo Inglês, é o modelo mais utilizado para o desenvolvimento de aplicações distribuídas, que utiliza uma rede de computadores para a comunicação de vários dispositivos que estão interligados por ela. A comunicação costuma se dar através de uma mensagem de solicitação do cliente enviada para o servidor, pedindo para que alguma tarefa seja executada (COULOURIS; DOLLIMORE; KINDBERG, 2007).

Um Soquete nada mais é do que uma conveniente abstração. Ele representa

uma ponto de conexão em uma rede TCP/IP[3], muito parecida com os soquetes elétricos em uma casa que provêm uma conexão ponto para os eletrodomésticos.

Quando dois computadores desejam “conversar”, isto é, trocar mensagens, cada usuário usa um Soquete. Um computador é denominado o servidor - ele abre Soquetes e espera por conexões. O outro computador é denominado o cliente - ele chama o servidor de Soquetes para iniciar a conexão. Para estabelecer uma conexão, tudo o que é necessário é um endereço do servidor de destino e um número de porta.

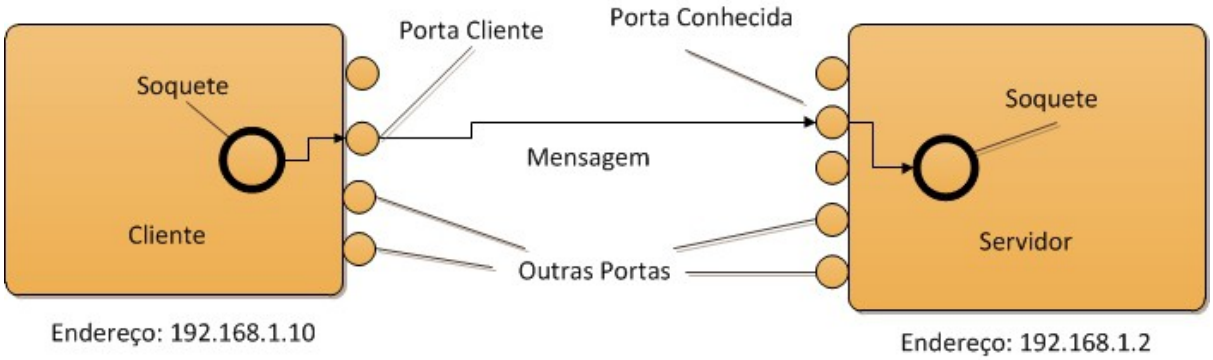

Figura 1 - Conexão cliente-servidor via Soquete.
Fonte: Adaptado de Coulouris *et al.* (2007)

Conforme visto na Figura 1, cada computador em uma rede possui um único endereço. Portas representam conexões individuais dentro daquele endereço.

Tomando como exemplo o serviço de correio de cartas, para uma carta chegar ate o seu destino (Casa) o carteiro irá percorrer toda a rua até encontrar a casa que contenha determinado número (Equivale a porta), como a rua possui o mesmo nome do início ao fim, a única forma de individualizar o seu destino é pelo número da casa, dessa forma o carteiro encontrará o destino correto e entregará a carta. Cada porta dentro de um computador compartilha o mesmo endereço, mas os dados são encaminhados dentro de cada computador pelo número da porta.

Quando um soquete é criado, deve ser associado com uma porta específica -

3 O **TCP/IP** é um conjunto de protocolos de comunicação entre computadores em rede (também chamado de pilha de protocolos TCP/IP). Seu nome vem de dois protocolos: o TCP (*Transmission Control Protoco*l - Protocolo de Controlo de Transmissão) e o IP (*Internet Protocol* - Protocolo de Interconexão). Os protocolos TCP/IP possibilitam que qualquer par de *hosts* se comuniquem, apesar das diferenças na arquitetura física (COMER; DROMS, 2007 ; p.268).

este processo é conhecido como ligando a uma porta.

Soquetes possuem dois modos principais de operação (COULOURIS; DOLLIMORE; KINDBERG, 2007):

- **Modo orientado à conexão:** Soquetes orientados à conexão operam como um telefone: eles devem estabelecer uma conexão e ao final desligá-la. Tudo o que fluir entre estes dois eventos chega na mesma ordem em que foi enviado.

- **Modo não orientado à conexão:** Soquetes não orientados à conexão operam como o correio: a entrega não é garantida, e múltiplos pedaços da correspondência podem chegar em uma ordem diferente daquela em que foram enviados.

3 MATERIAIS E MÉTODOS

Neste capítulo são descritos os materiais, métodos e tecnologias utilizadas para o desenvolvimento deste trabalho.

3.1 Materiais

Na implementação do sistema *DroidFitness* foram utilizados uma série de materiais, como podem ser vistos a seguir.

3.1.1 Java

Java segundo Oracle (2012), é uma linguagem computacional completa, adequada para o desenvolvimento de aplicações baseadas na rede Internet, redes fechadas ou ainda aplicações locais e para equipamentos portáteis, como telefones celulares. Atualmente está licenciada pela *General Public Licence* (GNU), sendo uma linguagem de programação de código livre e gratuita.

Foi desenvolvida por volta de 1995 nos laboratórios da empresa *Sun Microsystems* com o objetivo de ser mais simples e eficiente do que suas linguagens predecessoras. O alvo inicial era a produção de software para produtos eletrônicos de consumo (fornos de micro-ondas, agendas eletrônicas, etc.). Um dos requisitos para esse tipo de software é ter código compacto e de arquitetura neutra, ou seja, uma aplicação que funcione em qualquer plataforma.

3.1.1.1 Plataforma Java

O universo Java é um vasto conjunto de tecnologias, composto por três plataformas principais que foram criadas para segmentos específicos de aplicações (ORACLE, 2012):

- **Java SE** (*Java Platform, Standard Edition*). É a base da plataforma; inclui o ambiente de execução e as bibliotecas comuns.
- **Java EE (Java Platform, Enterprise Edition):** A edição voltada para o desenvolvimento de aplicações corporativas e para internet.
- **Java Micro Edition (ME):** A edição para o desenvolvimento de aplicações para dispositivos móveis e embarcados.

Além disso, pode-se destacar outras duas plataformas Java mais específicas:

- **Java *Card:*** Voltada para dispositivos embarcados com limitações de processamento e armazenamento, como *Smart Cards* e o Java *Ring*.
- **JavaFX:** Plataforma para desenvolvimento de aplicações multimídia para aplicações locais, para Internet (JavaFX *Script*) e dispositivos móveis (JavaFX *Mobile*).

A plataforma Java é constituída de um grande número de tecnologias, cada uma provê uma porção distinta de todo o ambiente de desenvolvimento e execução de software. Os usuários finais, tipicamente, interagem com a máquina virtual *Java Virtual Machine* (JVM) e um conjunto padrão de bibliotecas de classe. Os desenvolvedores de aplicações em Java utilizam um conjunto de ferramentas de desenvolvimento, denominado *Java Developer Kit* (JDK).

3.1.1.2 Características

As plataforma Java implementa o conceito híbrido, que tem como objetivo a segurança das verificações existentes em um processo de compilação e a portabilidade dos ambientes interpretados. O processo adotado para a implementação do modelo híbrido baseia-se na utilização da representação de um modelo intermediário denominada *bytecode*, que é gerada pelo compilador e interpretada no momento da execução (MATTOS, 2007).

Ainda conforme explica Mattos (2007), a plataforma Java utiliza essa

abordagem, contendo os seguintes passos em seu processo de desenvolvimento:

- arquivos com código fonte são armazenados em forma de um arquivo texto comum (.java);
- após a compilação dos arquivos de código fonte são gerados os *bytecodes* (.class);
- os *bytecodes* por final são interpretados pela Máquina Virtual Java, que por conseguinte transforma-os em linguagem de baixo nível (Linguagem de máquina – binária).

Um recurso fundamental do Java é o "*garbage collector*", que se responsabiliza por liberar parte da memória alocada que não é mais referenciada. Graças ao coletor de lixo do Java, os programadores não necessitam, e nem podem, liberar explicitamente memória anteriormente alocada.

O nome coleta de lixo é associado ao fato de que, se objetos não estão mais sendo utilizados, então pode-se "jogá-los fora". Outra metáfora para esse recurso é a "reciclagem" de memória. Quando um programa não mais referencia um objeto, esta parte da memória pode ser liberada para que outros objetos subsequentes, possam ali serem alocados.

A coleta de lixo alivia os programadores da questão de liberar a memória alocada, uma vez que esse processo é considerado complicado e perigoso para o nível de aplicação. A coleta de lixo é parte importante da estratégia de segurança da plataforma, pois se ela for feita de maneira incorreta pode causar uma falha crítica no funcionamento da JVM, e assim comprometer a aplicação que está sendo executada.

3.1.2 *Swing* API

Uma interface gráfica apresenta um mecanismo amigável para interagir com uma aplicação. A interface dá ao programa uma "aparência" e um "comportamento" distintos. Há dois conjuntos de componentes no Java (SERSON, 2007 ; p.373).

Antes do *Swing* ter sido introduzido no J2SE 1.2, os componentes gráficos do Java eram construídos em *Abstract Window Toolkit* (AWT) (ORACLE, 2012). Quando um programa elaborado com AWT é executado em diferentes plataformas, os componentes são desenhados na tela de forma diferente, de acordo com o sistema operacional utilizado.

A *Application Programming Interface* (API) *Swing* procura renderizar/desenhar por conta própria todos os componentes, ao invés de delegar essa tarefa ao sistema operacional, como a maioria das outras API`s de interface gráfica trabalham.

Por ser uma API de mais alto nível, ou seja, mais abstração, menor aproximação das API`s do sistema operacional, ela tem bem menos performance que outras API`s gráficas e consome mais memória de trabalho em geral. Porém, ela é bem mais completa, e os programas que usam *Swing* têm uma aparência muito parecida e independente do Sistema Operacional utilizado.

Os componentes *Swing* são implementados no Java; desse modo, eles são mais portáveis e flexíveis do que os componentes originais do AWT, que os foram baseados na plataforma subjacente. A maior parte dos componentes *Swing* não está vinculada a componentes reais suportados pela plataforma subjacente, onde um programa é executado. Esses componentes independentes da plataforma são conhecidos como componentes leves. Os componentes AWT, de forma contrária, por estarem vinculados à plataforma subjacente são chamados de pesados. Os componentes pesados contam com o sistema de janela da plataforma local para determinar s sua funcionalidade, aparência e comportamento (SERSON, 2007 ; p.373). Os componentes leves fornecem mais controle sobre a sua aparência e funcionalidade, por essa razão foram escolhidos para montar a composição da interface gráfica da aplicação de retaguarda do *DroidFitness*.

3.1.3 NeoDATIS ODB

NeoDatis Object Data Base (NeoDatis ODB) é um banco de dados muito simples (Para projetos de pequeno porte) orientado a objetos que pode ser

executado em Java (Ver Seção 3.1.1), .Net (MICROSOFT, 2012), Google Android (GOOGLE, 2012), Projeto *Groovy* (GROOVY, 2012) e Projeto *Scala* (SCALA, 2012). Disponível sob licença *General Public License* (GNU) , portanto, pode ser utilizado livremente em aplicações comerciais.

Orientado a Objetos porque a unidade básica persistente é um objeto e não uma tabela. Um objeto pode ser entendido como sendo uma entidade no sistema, (Ver Seção 5.2). Nativo e transparente, porque persiste objetos diretamente na forma como eles existem na linguagem de programação nativa, sem qualquer conversão.

> **Usando NeoDatis ODB como sua camada de persistência é possível se concentrar em sua regra de negócio permitindo armazenar e recuperar objetos nativos em uma única linha de código (NEODATIS, 2012) .**

O Quadro 1, representa o processo pelo qual um objeto no DroidFitness, passa até ser persistido na base de dados.

```
//Cria uma instancia do objeto a ser gravado
Aluno aluno = new Aluno();
Traino treino = new Treino();
treino.setNome(“Treino Aluno X”);
//...
aluno.setNome(“Pedro”);
aluno.setTreino(treino);

//Abre a conexão com a base de dados
ODB odb = NeoDatis.open("academia.db");

//Grava o objeto na base de dados
odb.store(aluno);

//Fecha a conexão com a base de dados
odb.close();
```

Quadro 1: Persistindo objeto no NeoDatis.

Como visto no Quadro 1, é inicialmente criado um objeto do tipo Aluno, em

seguida pode ser valorizado cada um de seus atributos. Na sequencia é necessário abrir a conexão com a base de dados e posteriormente o objeto pode ser persistido na base de dados. Caso a transação tenha sido efetuada com sucesso a conexão com a base de dados é fechada.

O Quadro 2 representa o momento de recuperar um objeto da base dedados.

```
//Abre a conexão com a base de dados
ODB odb = NeoDatis.open("academia.db");


//Recupera todos os objetos do tipo Aluno, da base de dados
Objects<Aluno> listaAlunos = odb.getObjects(Aluno.class);


//Fecha a conexão com a base de dados
odb.close();
```

Quadro 2: Recuperando objetos no NeoDatis

No momento de recuperar objetos do NeoDatis é aberta a conexão com a base de dados, logo após, por meio do método "*getObjects()"* é possível recuperar uma lista com todos os objetos que estão na base de dados ("tipado" com objeto do tipo Aluno). Por final é fechada a conexão através do método "*close()*".

O Quadro 3 representa o momento de alterar um objeto previamente cadastrado na base de dados.

```
//Abre a conexão com a base de dados
ODB odb = NeoDatis.open("academia.db");


//Recupera o objeto aluno no qual se deseja alterar
Aluno alunoBanco = (Aluno) odb.getObjectFromId(aluno.getCodigo());
//É definido o novo conteúdo dos atributos da entidade Aluno
alunoBanco.setNome(aluno.getNome());
alunoBanco.setTreino(aluno.getTreino());
//Grava na base de dados o objeto alterado
odb.store(alunoBanco);
//Fecha a conexão com a base de dados
odb.close();
```

Quadro 3: Alterando objetos no NeoDatis

No processo de alteração de um registro, é aberta a conexão com a base de dados, em seguida é recuperado o objeto desejado por meio do atributo “OID”, que é nativo do banco de dados NeoDatis. Tendo recuperado o objeto, já é possível alterar os dados de seus atributos e posteriormente gravá-lo novamente na base de dados e por fim fechar a conexão.

Para a última operação de manutenção de dados no NeoDatis, a exclusão de objetos, é realizada conforme a representação do Quadro 4.

```
//Abre a conexão com a base de dados
ODB odb = NeoDatis.open("academia.db");


//Recupera o objeto aluno no qual se deseja excluir
Aluno alunoBanco = (Aluno) odb.getObjectFromId(aluno.getCodigo());


//Remove da base de dados o objeto
odb.delete(alunoBanco);


//Fecha a conexão com a base de dados
odb.close();
```

Quadro 4: Excluindo objetos no NeoDatis

O procedimento de exclusão de objetos é semelhante a de alteração (Quadro 3), é necessário primeiramente abrir a conexão e recuperar o objeto a ser excluído por meio do atributo “OID”, em seguida através do método “*delete()*” é possível remover o objeto da base de dados.

As quatro operações básicas de manutenção de dados (Incluir, Recuperar, Alterar, Excluir), como visto são consideradas relativamente simples, uma vez que o NeoDatis possui comandos de fácil implementação por essa razão o tempo de aprendizagem pode ser muito curto.

O NeoDatis possui uma ferramenta chamada, *ODB Explorer* (Figura 1): uma ferramenta gráfica de navegação e consulta, possibilita criar, atualizar e excluir objetos, bem como a importação / exportação do banco de dados para outros tipos de arquivos de dados.

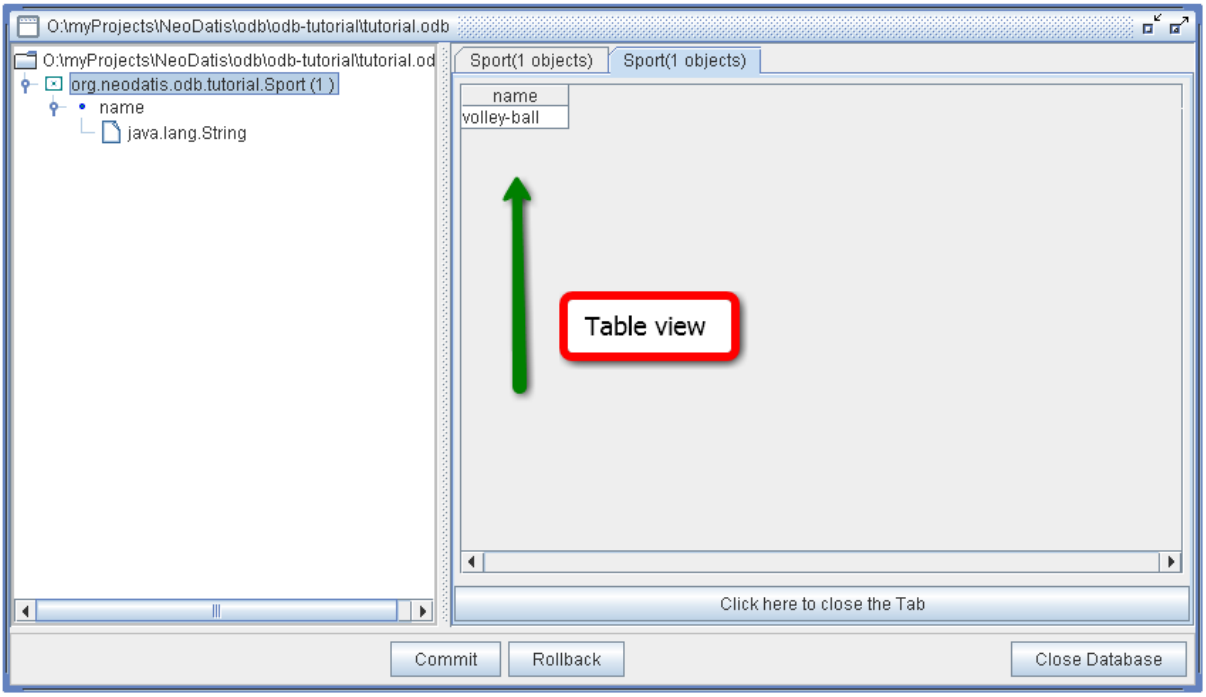

Figura 2 - ODB Explorer
Fonte: NeoDatis (2012)

3.1.4 Arquitetura de Servidores Multi-threads

Linha ou Encadeamento de execução (em inglês: *thread*), é uma forma de um

processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrentemente. O suporte à *thread* é fornecido pelo próprio sistema operacional ou implementada através de uma biblioteca de uma determinada linguagem, como por exemplo o Java. Uma *thread* permite, por exemplo, que o usuário de um programa utilize uma funcionalidade do ambiente enquanto outras linhas de execução realizam outros cálculos e operações (SILBERSCHATZ, 2008).

No *DroidFitness* a utilização de *threads* é fundamental, pois por se tratar de uma aplicação distribuída, onde mais de uma conexão de clientes podem acontecer ao mesmo tempo. Na aplicação de retaguarda, existe uma *thread* chamada de *thread* de Entrada e Saída E/S, que é responsável por manter o módulo servidor ativo, e quando um processo cliente se conectar para recuperar suas informações, a aplicação servidora continuará ativa para receber novas conexões, sem que o novo pedido fique numa fila de espera e o servidor fique travado enquanto processa a solicitação.

> **Com o uso de várias *threads* permite-se aos servidores maximizarem sua taxa de rendimento, medido como o número de pedidos processados por segundo (COULOURIS; DOLLIMORE; KINDBERG, 2007).**

O *DroidFitness* implementa uma arquitetura *Multi-thread* chamada arquitetura de *threads* por pedido (Figura 3), *thread* de E/S gera uma nova *thread* trabalhador para cada pedido e esse trabalhador se autodeterminará quando tiver processado o pedido. Essa arquitetura tem a vantagem de que as *threads* não disputam uma fila compartilhada e a taxa de rendimento é potencialmente maximizada, pois a *thread* de E/S pode criar tantos trabalhadores quantos forem os pedidos pendentes. Sua desvantagem é a sobrecarga das operações de criação e destruição de *threads*.

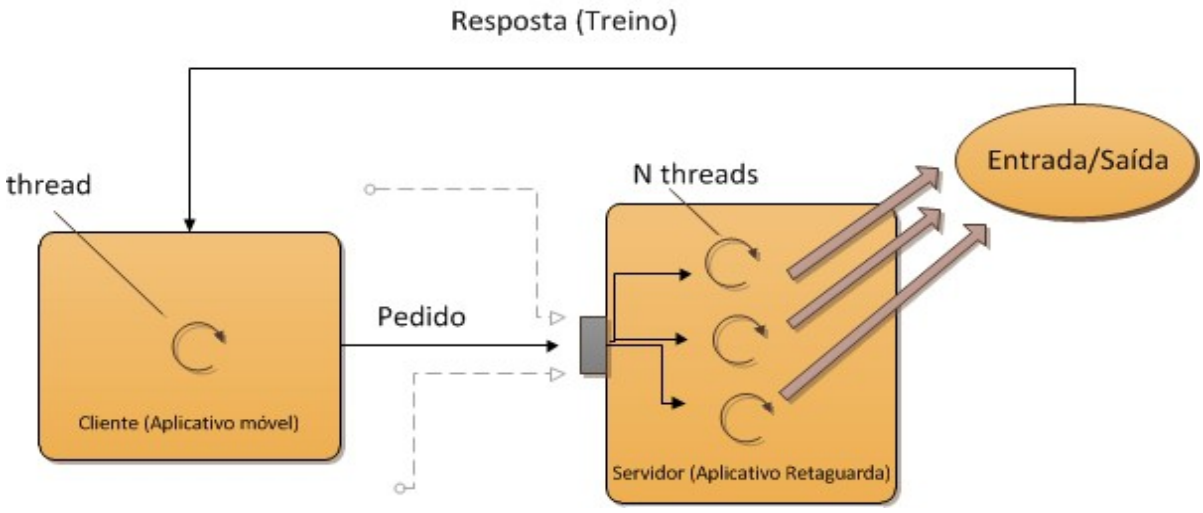

Figura 3 - Soquetes multi-threads
Fonte: Adaptado de Coulouris *et al.* (2007)

O procedimento de criação e inicialização de uma *thread* pode ser visualizado no Quadro 5.

```
//Cria uma nova thread para cada conexão recebida pelo soquete
public void run() {
      try {
          ServerSocket servidor = new ServerSocket(26900);
          while (true) {
              System.out.println("Aguardando Conexões.");
              Socket cliente = servidor.accept();
              new ConexaoCliente(cliente).start();
          }
      } catch (IOException ex) {
      }
  }

  //Inicializa a nova thread
  class ConexaoCliente extends Thread {
      ObjectOutputStream out;
      ObjectInputStream in;
      Aluno aluno;
      Socket socket;

      public ConexaoCliente(Socket socket) {
          this.socket = socket;
      }
      public void run() {
       //Rotina de processamento
     }
```

Quadro 5: Criação de uma nova thread.

Uma nova conexão é iniciada pela aplicação móvel, em seguida é recebida pela aplicação servidora por meio de um soquete cliente, cada conexão por sua vez é tratada de maneira independente por meio de uma *thread* individual.

Conforme visto na Figura 3, no momento do recebimento da solicitação de conexão, a *thread* principal instancia uma nova *thread* trabalhadora que será responsável por tratar a solicitação daquela conexão, para em seguida devolver a resposta para a aplicação cliente. Como cada conexão é tratada de forma individual,

inúmeras requisições podem ser processadas simultaneamente, dessa forma é descartada a possibilidade de concorrência entre várias conexões.

3.1.5 NetBeans IDE

O NetBeans IDE, segundo Oracle (2012), é um *Integrated Development Environment* (IDE), gratuito e de código aberto para desenvolvedores de software para diversas linguagens de programação, inclusive o Java e funciona mas maioria dos sistemas operacionais existentes. O NetBeans IDE oferece aos desenvolvedores, ferramentas necessárias para criar aplicativos profissionais de *Desktop*, empresariais, para redes de computadores e Internet e móveis multiplataformas.

O NetBeans foi iniciado em 1996 por dois estudantes tchecos na Universidade de Charles, em Praga, quando a linguagem de programação Java ainda não era tão popular como atualmente. Primeiramente o nome do projeto era Xelfi, em alusão ao Delphi[4], pois a pretensão deste projeto era ter funcionalidades semelhantes aos IDE's então populares do Delphi que eram mais atrativas por serem ferramentas visuais e mais fáceis de usar, porém com o intuito de ser totalmente desenvolvido em Java.

Em 1999 o projeto já havia evoluído para uma IDE proprietário, com o nome de NetBeans DeveloperX2, nome que veio da ideia de reutilização de componentes que era a base do Java. Nessa época a empresa Sun *Microsystems* havia desistido de sua IDE Java Workshop e, procurando por novas iniciativas, adquiriu o projeto NetBeans DeveloperX2 incorporando-o a sua linha de softwares.

A IDE NetBeans auxilia programadores a escrever, compilar, depurar e instalar aplicações, e foi arquitetada em forma de uma estrutura reutilizável que visa simplificar o desenvolvimento e aumentar a produtividade, pois reúne em uma única aplicação todas estas funcionalidades. Totalmente escrita em Java, mas que pode suportar qualquer outra linguagem de programação que desenvolva com *Swing*

4 **Delphi**, é um compilador, uma IDE e uma linguagem de programação muito utilizado no desenvolvimento de aplicações locais, aplicações multicamadas e cliente/servidor, compatível com os bancos de dados mais conhecidos do mercado (EMBARCADERO, 2012).

(ORACLE, 2012), como *C* (FEOFILOFF, 2009), *C++* (HUBBARD, 2000), *Ruby* (FITZGERALD, 2007) e PHP (LERDORF, 2002). Também suporta linguagens de marcação como XML e HTML.

3.1.6 Eclipse IDE

Eclipse segundo sítio do projeto Eclipse (2012), é um IDE desenvolvido em Java, seguindo o modelo código livre de desenvolvimento de software. O projeto Eclipse foi iniciado na empresa IBM no final da década de 90 para utilização interna. Devido a falta de investimento no projeto doou-o como software livre para a comunidade. Em 2001 juntou-se com outras empresas como: Borland, QNX, Rational, Hed Hat, Suse; e formaram a *Eclipse Fundation* para manter o projeto. Hoje, o Eclipse é o IDE Java mais utilizado no mundo. Possui como características o uso da SWT (ECLIPSE, 2012) e não do *Swing* como biblioteca gráfica, a forte orientação ao desenvolvimento baseado em *plug-ins* e o amplo suporte ao desenvolvedor com centenas de *plug-ins* que procuram atender as diferentes necessidades de diferentes programadores. *Plug-ins* atualizados de novas tecnologias como GWT[5], Android, ExtJS[6] quase sempre estão disponíveis primeiro para o eclipse.

3.1.7 Android

O Android é uma plataforma para tecnologia móvel, envolvendo um pacote com programas para celulares, já com um sistema operacional *middleware,* aplicativos e interface com o usuário (PEREIRA; SILVA, 2009).

Android foi construído com a intenção de permitir aos desenvolvedores criar aplicações móveis que possam tirar total proveito do que um aparelho portátil possa oferecer. Foi construído para ser verdadeiramente aberto. Por exemplo, uma

5 **Google Web Toolkit** – Kit de ferramentas da google pra desenvolvimento web.
6 **ExtJS** – *Framework* baseado em Css e JavaScript que se aproxima muito da Orientação a Objetos para desenvolvimento de interfaces ricas para o ambiente web.

aplicação pode apelar a qualquer uma das funcionalidades de núcleo do telefone, tais como efetuar chamadas, enviar mensagens de texto ou utilizar a câmera, que permite aos desenvolvedores adaptarem e evoluírem cada vez mais estas funcionalidades.

Por ser de código livre, pode ser sempre adaptado a fim de incorporar novas tecnologias, conforme estas forem surgindo. A plataforma vai estar sempre em evolução, já que as comunidades de desenvolvedores estarão trabalhando em conjunto para construir aplicações móveis inovadoras.

O sistema Android, é o primeiro projeto de uma plataforma de código livre para dispositivos móveis. O SDK é o kit de desenvolvimento que disponibiliza as ferramentas e API's necessárias para desenvolver aplicações para a plataforma Android, utilizando a linguagem Java.

A plataforma Android foi desenvolvida com base no sistema operacional Linux, no entanto não é considerado como sendo um Linux, pois não possui sui *windowing system* nativo (componente de interface gráfica), não suporta *glibc* (GNU, 2012) e não possui alguns dos conjuntos de padrões apresentados em algumas distribuições Linux.

O Android é considerado muito seguro. Como ele é executado em um *Kernel* (núcleo) Linux, toda a vez que um aplicativo for instalado no dispositivo, é criado um novo usuário Linux para aquele programa, com diretórios que serão usados pelo aplicativo, mas somente para aquele usuário Linux. Como cada aplicação instalada fica completamente isolada umas das outras, qualquer tentativa de acessar informações de outro aplicativo precisa ser explicitamente autorizada pelo usuário, podendo ser negada a sua instalação ou autorizada, mas controlando as permissões que este aplicativo poderá ter através de um mecanismo de permissões (PEREIRA; SILVA, 2009).

3.1.7.1 Arquitetura Android

Na Figura 4 é possível visualizar a estrutura da arquitetura do Android, com cada uma das camadas disponíveis e funcionalidades (GOOGLE, 2012).

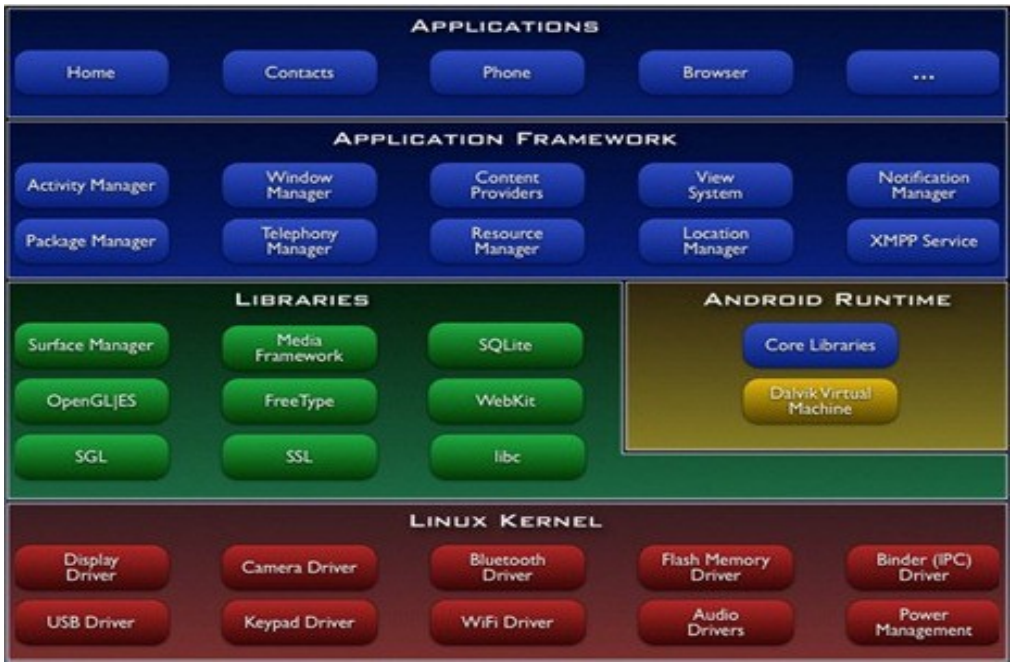

Figura 4 - Arquitetura Android
Fonte: Google (2012).

• **Camada de Aplicações:** na camada de aplicações podem existir todos os aplicativos fundamentais (escritos em Java) do Android, como por exemplo, navegadores web, clientes de e-mail, calendários, mapas, gerenciadores de contatos, aplicações de voz e todas as demais que são em nível de usuário.

• **Camada de *Framework*:** nesta camada são encontradas as API`s e os recursos utilizados pelos aplicativos, como classes visuais (componentes de interface gráfica), gerenciadores de localização (para aplicações de mapa e posicionamento global), gerenciadores de notificações (fornecem informações sobre o que acontece no dispositivo).

• **Camada de Bibliotecas:** carrega consigo um conjunto de bibliotecas C/C++ utilizadas pelo sistema e também bibliotecas multimídia, visualização de camadas 2D e 3D, funções para navegadores para Internet, funções para

gráficos e aceleradores de hardware, renderização 3D, funções de acesso a banco de dados, e muitos outros recursos estão disponíveis no *framework* para o desenvolvimento de aplicativos.

- **Camada Android *Runtime*:** é uma instância da máquina virtual *Dalvik*, criada para cada aplicação ser executada no Android. A *Dalvik* é uma máquina virtual onde cada aplicação é executada dentro do sistema Android, e possui bom desempenho, boa integração com a nova geração de hardware e projetada para executar várias máquinas virtuais paralelamente. Foi projetada para funcionar em sistemas com baixa frequência de processamento, pouca memória de trabalho e sem área de troca de memória e além de tudo para ter maior rendimento do consumo da bateria.

- **Linux *Kernel*:** Utiliza a versão 2.6 do Kernel (núcleo) do Linux para serviços centrais do sistema, tais como segurança, controle de memória, de processos, protocolos de rede e de modelos de drives. Também se encontra um módulo de software capaz de gerenciar energia, ele identifica quais recursos do aparelho não estão em uso e os desliga.

3.1.7.2 Android SDK

O Android SKD[7] é uma API[8] que disponibiliza todo o ambiente para construir e testar aplicações Android. Atualmente suporta as versões 1.5, 1.6, 2.1, 2.2, 2.3.3, 3.0, 3.1, 3.2, 4.0, 4.0.3, e 4.1 do Android (GOOGLE, 2012).

O *SDK* pode ser livremente utilizado para o desenvolvimento de aplicativos. Para publicá-lo na loja oficial de aplicativos da Google o *Google Play* é necessário o pagamento de ma taxa de inscrição como desenvolvedor, atualmente no valor de U$ 23,00 (vinte e três Dólares americanos).

---

7 **Software Development Kit**: Kit para desenvolvimento de Softwares.

8 **Application Programming Interface** (ou Interface de Programação de Aplicativos) é um conjuntode rotinas e padrões para a utilização das funcionalidades de algum sistema.

3.1.8 AfreeChart

AFreeChart é uma biblioteca gratuita de gráficos do tipo *chart*[9] para a plataforma Android, desenvolvida a partir da JfreeChart[10] (AFREECHART, 2012).

Usando AFreeChart é possível construir gráficos para aplicações Android muito facilmente. AfreeChart possui licença pública *General Public Licence* (GPL), por tanto seu código fonte pode ser editado acordo com as necessidade do desenvolvedor.

3.1.9 Astah

Os diagramas apresentados foram confeccionados com o apoio da ferramenta Astah.

O Astah é uma ferramenta para modelagem UML muito fácil de usar e muito útil. Ela está disponível na versão *Professional* – Paga com todas as funcionalidades e a versão *Community Edition* – Gratuita com funcionalidades reduzidas, possuindo apenas, modelagem de classes, entidades e relacionamentos, casos de uso, diagramas de sequência, componentes e pacotes (ASTAH, 2012).

9 **Chart**: Em inglês, se refere a gráficos analíticos geralmente apresentados em relatórios gerenciais.

10 **JFreeChart** é uma biblioteca gráfica gratuita para Java (JFREECHART, 2012).

4 RESULTADOS E DISCUSSÕES

Este capítulo apresenta as principais características do ramo de negócio para o qual o sistema está sendo desenvolvido bem como a descrição do processo manual que é executado atualmente.

4.1 Descrição Geral

Nesta Seção é apresentada uma abordagem geral da descrição do cenário de Funcionamento do *DroidFitness* e da solução proposta.

4.1.1 Cenário

Ao se inscrever em uma academia de ginástica, é realizada uma avaliação física onde, através de métricas próprias do negócio são determinados os tipos, quantidade, intensidade e frequência dos exercícios e/ou atividades que o indivíduo deve executar para atingir os objetivos esperados.

Com base nos resultados da avaliação física é elaborada uma ficha de avaliação[11] e acompanhamento. Periodicamente é realizada uma nova avaliação para medir a evolução e a eficácia do treinamento e também identificar se é necessário alterar a carga de exercícios e consequentemente atualizar a ficha de acompanhamento, conforme pode ser visto na Figura 5.

11 **Ficha de avaliação**: Ficha contendo dados do aluno e dos grupos musculares com cada exercício equivalente.

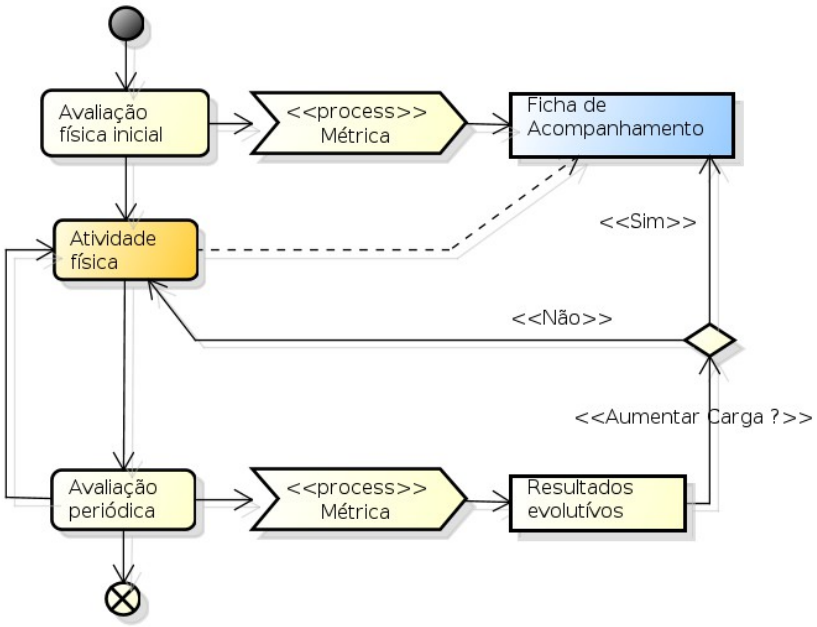

Figura 5 - Processo Atual

Todo esse processo (Figura 5) necessita de um acompanhamento dinâmico tanto para o instrutor acompanhar a evolução individual de cada aluno, bem como para que os alunos tenham acesso a informações atualizadas sobre seus exercícios, além de indicadores que mostrem a evolução de seus treinos e resultados.

4.1.2 Solução Proposta

Com base no diagrama da Figura 5 da seção 4.1.1 e nos requisitos estabelecidos na modelagem realizada durante o estágio supervisionado foi implementado um sistema cliente-servidor (Figura 6) com os seguintes módulos:

i. **Móvel (cliente):** Aplicativo móvel cliente para acompanhamento de exercícios;
ii. **Retaguarda (servidor):** Aplicativo *desktop* para cadastro e manutenção das séries e exercícios.
iii. **Integrador:** Módulo responsável por realizar a integração com sistemas de terceiros.

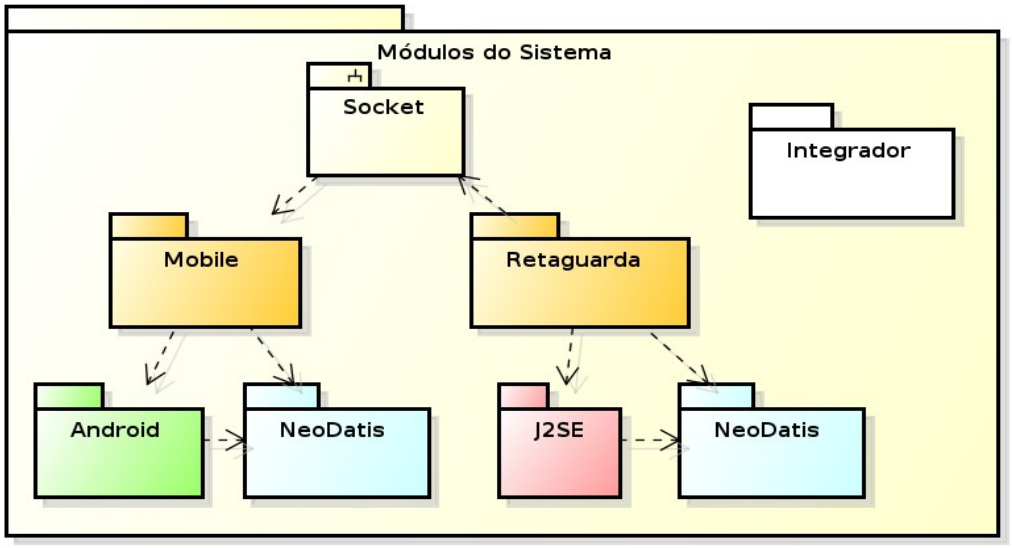

Figura 6: Módulos do sistema.

Por fim deve ficar claro que o produto a ser desenvolvido neste projeto consiste nos módulos i e ii descritos na modelagem desenvolvida num trabalho anterior (ver Figura 2 da modelagem) conforme descrições de funcionalidade estabelecidas na seção 3 do mesmo documento, uma vez que o integrador (módulo iii) depende de estudos e diálogos com o proprietário desenvolvedor do(s) seu(s) sistema(s). Por esse motivo não foi identificada a necessidade de conexões com outros sistemas e as descrições e aplicações estabelecidas neste documento referem-se apenas aos módulos móvel (cliente) e Retaguarda (servidor).

4.2 Especificação do Negócio

Nesta Seção é apresentada uma abordagem das principais regras de negócio que constituem a solução proposta.

4.2.1 Regras de Negócio

As regras de negócio representam a forma de trabalho, o que o processo deve respeitar para ser realizado corretamente. Indicadores, limites, taxas e condições são visivelmente observadas. Para este módulo foram identificadas as seguintes regras:

1. Todos os campos determinados como obrigatórios devem ser validados.
2. Podem ser criados diferentes treinos (sequências de exercícios predefinidos).
3. Cada exercício deve estar vinculado a um aparelho e deve ser personalizável para cada aluno.
4. Cada aluno deve estar vinculado a um treino específico.
5. Cada treino transferido para o dispositivo móvel deve possuir um cabeçalho contendo nome da academia e/ou instrutor e telefone para contato.
6. O sistema deve informar ao aluno sua evolução (ganho de massa, perda de peso, IMC).
7. Somente o instrutor, e somente através da aplicação retaguarda, pode alterar o treino, as cargas e a frequência.
8. O sistema deve permitir oferecer ao aluno substitutos para cada exercício/aparelho.
9. Não foi identificada a necessidade de controle de acesso/bloqueio por diferentes níveis de usuário.

4.3 Glossário

Esta seção apresenta alguns termos específicos do domínio ou cenário de negócio na qual o sistema será inserido. Estas informações serão úteis para que os *stakholders*[12] entendam o propósito do software a qualquer estado do projeto.

12 ***Stakeholder***: (parte interessada): é alguém ou algo que tem um interesse legal no comportamento do caso de uso (COCKBURN, 2001, p.65). *Stakeholder* são os Atores que irão interagir com o sistema.

- **Treino**: conjunto de exercícios pré-definidos tipicamente de a cordo com cada objetivo. Ex.: Treino de força, Treino de hipertrofia, Treino A/B, Pirâmide, etc.
- **Série**: é a repetição de algum treino, por algum tempo com cargas e repetições personalizadas de a cordo com a capacidade e objetivo de cada aluno.
- **Aparelho**: Equipamento mecânico ou eletromecânico que através de roldanas e/ou alavancas permite controlar a intensidade, direção e ergonomia do exercício.
- **Carga**: É o peso ou a resistência que cada aparelho oferece para a execução do exercício.
- **IMC**: Índice de Massa Corporal. Fator muito utilizado para avaliações físicas cujo valor representa com bastante aproximação o percentual de gordura corporal (IMC = peso/altura²).
- **Instrutor**: Profissional da área de educação física. Habilitado para avaliar e definir as condições para realização de atividade física.
- **CREF**: Conselho Regional de Educação Física.
- **Aluno**: Indivíduo que procura academias de ginástica ou musculação com o objetivo de realizar atividades físicas para melhorar seu estado corporal.

4.4 Especificação e Requisitos do Sistema

Esta seção contém os principais requisitos modelados para o sistema, e as regras de negócio que necessariamente precisam ser obedecidas para que o sistema atenda a finalidade a qual se propõe.

As pessoas que usam o sistema são identificadas como atores (que executam um papel no sistema). Como os usuários possuem algumas funções específicas no sistema, a descrição realizada na Tabela 1 identifica, resumidamente

quem são estes usuários e na Tabela 2, a vinculação de suas funções.

Nesta seção também estão os diagramas de classes e entidade-relacionamento. Também estão documentados a representação da arquitetura do sistema e dos padrões de projeto utilizados.

Nº	Nome do Ator	Descrição
1	Administrador (A)	Pessoa que pode realizar todas as funções no sistema.
2	Instrutor (I)	Pessoa que pode realizar todas as manutenções no sistema exceto cadastrar instrutores
3	Aluno (U)	Pessoa que pode visualizar sua série através de um dispositivo móvel.

Tabela 1 - Atores e seus papéis.

Nº	Nome do Caso de Uso	Atores		
		A	I	U
1	Manter Instrutores	X		
2	Manter Alunos	X	X	
3	Manter Aparelhos	X	X	
4	Manter Treinos	X	X	
5	Manter Exercícios	X	X	
6	Manter Séries	X	X	
7	Manter Avaliações	X	X	
8	Visualizar série específica			X
9	Receber atualização de Dados			X

Tabela 2 - Atores e suas funcionalidades

O Administrador é uma diferenciação do usuário instrutor, pois pode realizar as manutenções básicas do sistema como: cadastro do estabelecimento e manutenção de instrutores.

4.5 Listagem de Atores e Casos de Uso

Para esse sistema foram identificados os seguintes atores:

- **Administrador:** é o usuário que acesso total ao sistema, em resumo é o usuário que poderá acessar pela primeira o sistema terá a permissão para cadastrar novos instrutores ou as demais informações;

- **Instrutor:** é o indivíduo responsável pela academia, ou seja, será ele quem poderá cadastrar alunos, treinos ou mesmo realizar o acompanhamento do alunos;

- **Aluno:** é o indivíduo que a partir do acompanhamento do Instrutor, poderá acessar a lista de treino no dispositivo móvel, ou mesmo os dados estatísticos referentes as avaliações físicas realizadas pelo instrutor.

Como funcionalidades do módulo *Desktop* (servidor) têm-se: controle de acesso, manutenção de instrutores, manutenção de aparelhos, manutenção de exercícios, manutenção de alunos, manutenção de treinos, manutenção de ficha de acompanhamento e cadastro de avaliações. No módulo Móvel (cliente): visualização e acompanhamento da série, pesquisa de exercícios alternativos, detalhes do exercício, quadro evolutivo e atualização de dados.

As funcionalidades do sistema sistema foram modeladas em casos de usos organizados em diagramas. Esta modelagem permitirá um acompanhamento do requisito nas diversas etapas do processo de desenvolvimento.

Para representar as funcionalidades de manutenções de dados do sistema proposto, o diagrama de casos de uso foi modelado e está representado na Figura 7.

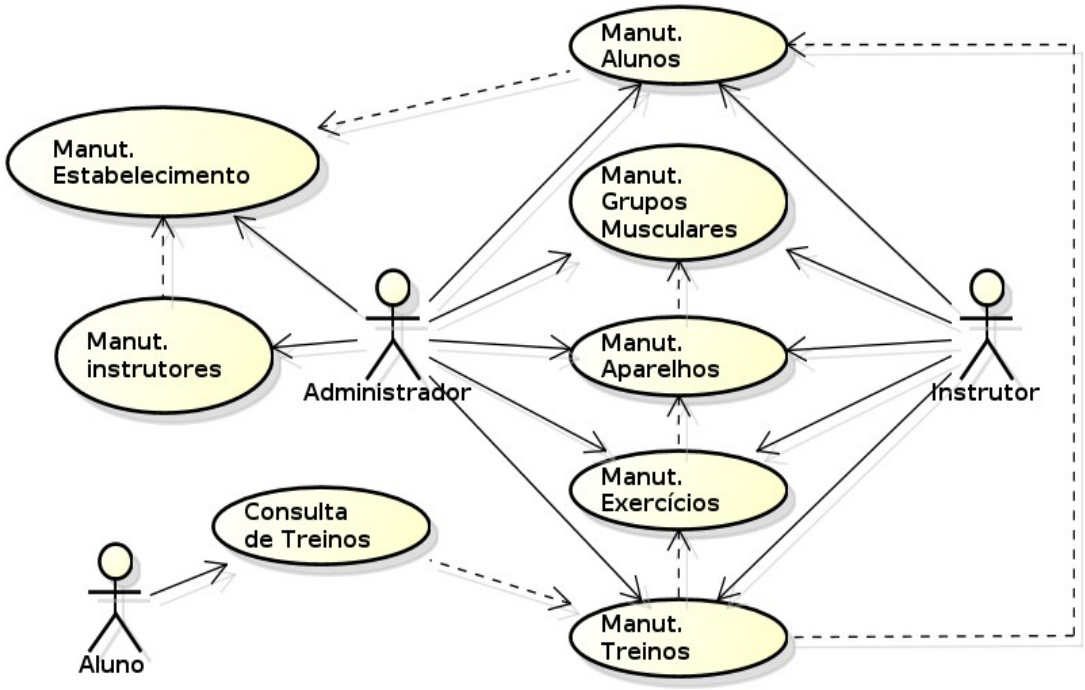

Figura 7 - Diagrama de Caso e Uso

De acordo com a Figura 7, o Instrutor interage com todos os módulos de manutenção de dados, exceto o formulário de manutenção de estabelecimento e manutenção de instrutores, sendo que somente o administrador poderá acessá-los.

4.6 Arquitetura Padrão de Projeto

A arquitetura de projeto utilizada (Figura 9) é o modelo de três camadas MVC (Modelo-Visualização-Controle) por fornecer uma maneira de dividir a funcionalidade envolvida na manutenção e apresentação dos dados da aplicação.

MVC é um modelo de desenvolvimento de Software, atualmente considerado uma "arquitetura padrão" utilizada na Engenharia de Software. O modelo isola a "lógica" (A lógica da aplicação) da interface do usuário (Inserir e exibir dados), permitindo desenvolver, editar e testar separadamente cada parte. Basicamente o padrão MVC implementa um caso de uso interativo em três componentes (SAMPAIO, 2007 ; p.79):

• **Modelo:** mantém o estado atual do módulo de Visualização. É responsável por alterar o estado e fornecer à Visualização os valores atuais ;

• **Visualização:** captura ações do usuário e comunica ao Controle para que sejam executadas. Também atualiza as informações exibidas sempre que o Controle avisa que devem ser atualizadas;

• **Controlador:** recebe solicitações da Visualização, invoca as regras de negócio necessárias, comanda a alteração do estado do Modelo e a atualização das informações exibidas na Visualização.

No *DroidFitness* a arquitetura padrão (MVC), utilizada é representada conforme a Figura 8.

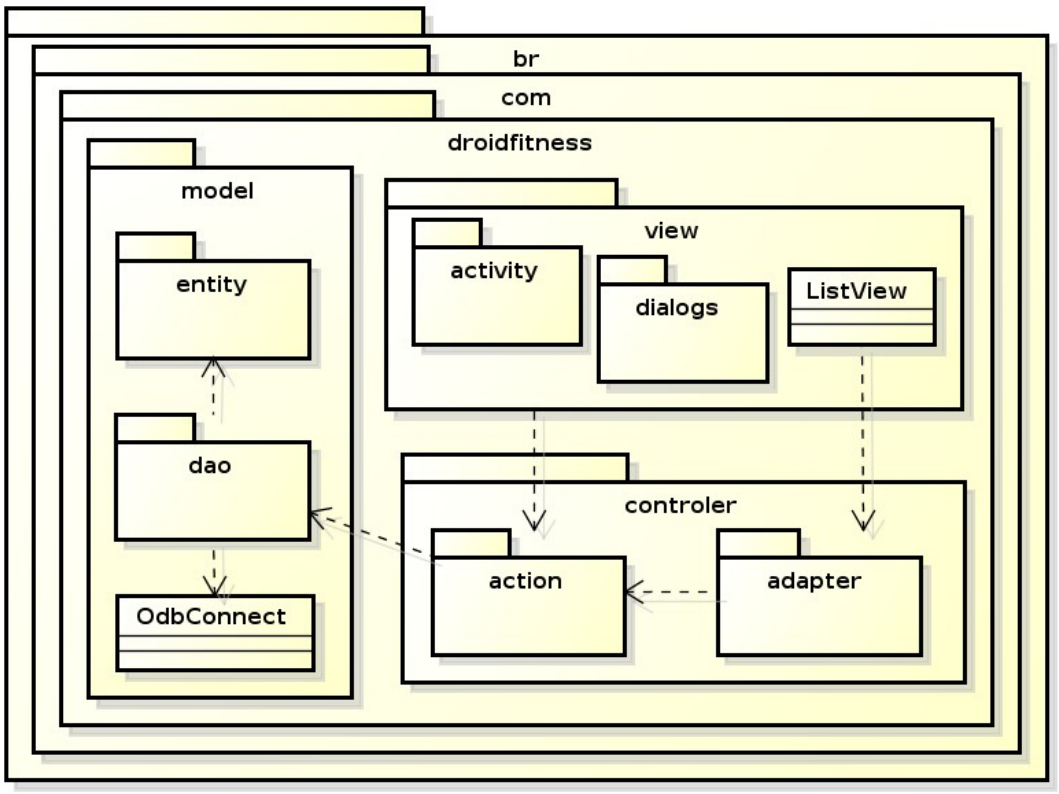

Figura 8 - Arquitetura e padrão de projeto

Conforme pode ser visualizado na Figura 8, a camada de Visualização (*View*), é responsável por manter o funcionamento das janelas (interface com o usuário). As informações após serem processadas pela camada de Visualização são enviadas para a camada de Controle (*Controller*), onde é realizado o processamento intermediário, ou seja, a preparação dos dados para serem persistidos ou recuperados da base de dados. Após essa etapa, os dados são enviados ou recuperados da base de dados, por meio da camada de Modelo (*Model*), que é inteiramente responsável por realizar as transações com a base NeoDatis.

4.7 Diagrama de Classes

O diagrama de Classes apresenta as entidades que representam os objetos abstraídos do modelo real, bem como os relacionamentos entre eles.

A classe abstrata *Entity* possui o atributo *OID* que é o identificador único utilizado pelo *NeoDatis* e ainda implementa a “serialização” de objetos através da classe Java *Serializable.* Ela transfere por herança essas características para as demais classes que compões as entidades persistentes do sistema compondo assim uma única família de entidades.

Este diagrama também representa a modelagem do banco de dados, uma vez que é utilizada a tecnologia de banco orientado a objetos para persistência de dados na aplicação cliente e de gerenciamento.

O diagrama de Classes e relacionamentos é representado de acordo com a Figura 9.

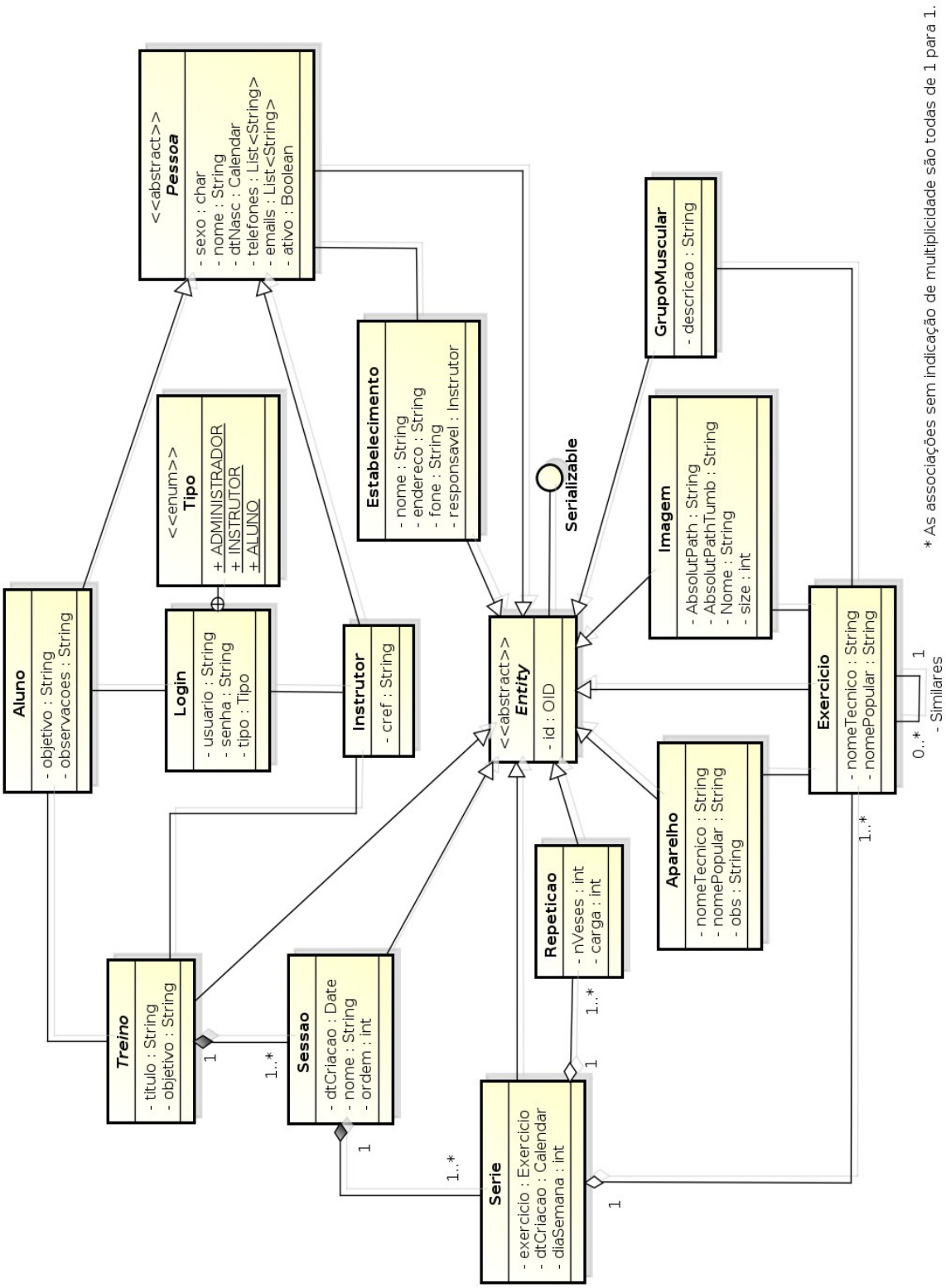

Figura 9: Diagrama de Classes e Relacionamentos

A partir do Diagrama visto anteriormente na Figura 9, percebeu-se durante o desenvolvimento, a necessidade de mais duas classes pra melhor representar a abstração desejada. São elas:

- **Sessão:** Esta classe se fez necessária para que fosse possível abstrair os diferentes dias de um treino que são verdadeiramente sessões de uma quantidade se séries de exercícios.

- **Imagem:** inicialmente pretendia-se “serializar” as imagens em *bytes* para transmiti-las e posteriormente persisti-las na base de dados da aplicação móvel. Porém durante os testes ao recuperar essas informações do banco de dados ocorrem problemas para o Android “desserializar” os objetos que contem classes do tipo **byte[]** e **File**. Para superar isso implementou-se uma classe contendo apenas os atributos de tamanho e localização do arquivo e passou-se a salvar as imagens diretamente no cartão de memória do dispositivo.

5 ANÁLISE DOS RESULTADOS

Neste capítulo é apresentado os resultados obtidos no o desenvolvimento da solução, incluindo o módulo de retaguarda e o módulo móvel.

5.1 Módulo de Retaguarda

A estrutura do sistema está dividida em módulos, cada módulo por sua vez é representado por meio de uma entidade, formando assim, as telas de cadastro e funcionalidades do sistema.

No momento de cadastro das informações o usuário deve seguir um fluxo

O fluxo de uso do sistema segue uma ordem conforme pode ser visualizado na Figura 10.

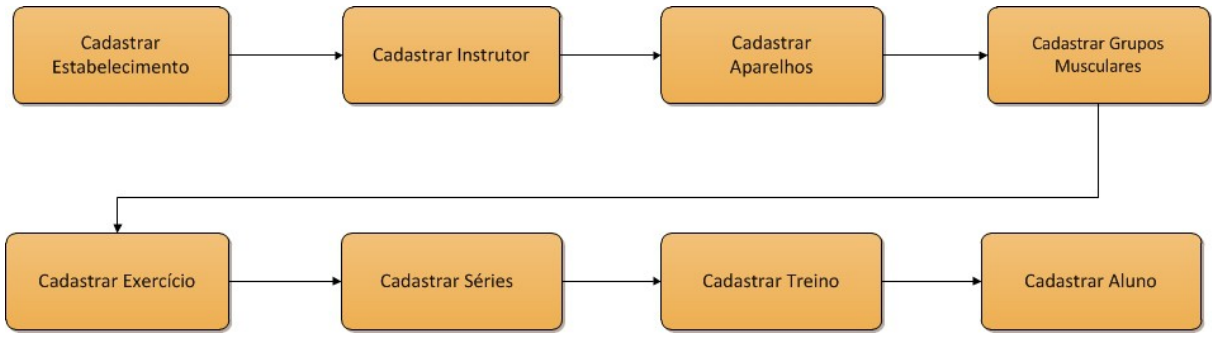

Figura 10 - Fluxo de uso do sistema

5.1.1 Manutenção de Estabelecimento

O módulo Estabelecimento representa o ambiente de cadastro de estabelecimentos (Academia de Ginástica), é representado pela entidade *EstabelecimentoEntity*. Por meio deste módulo é possível realizar o cadastro de estabelecimento, ou seja, as informações referentes de onde o aluno realiza as suas

atividade físicas. A Figura 11 representa a estrutura da tela de cadastro.

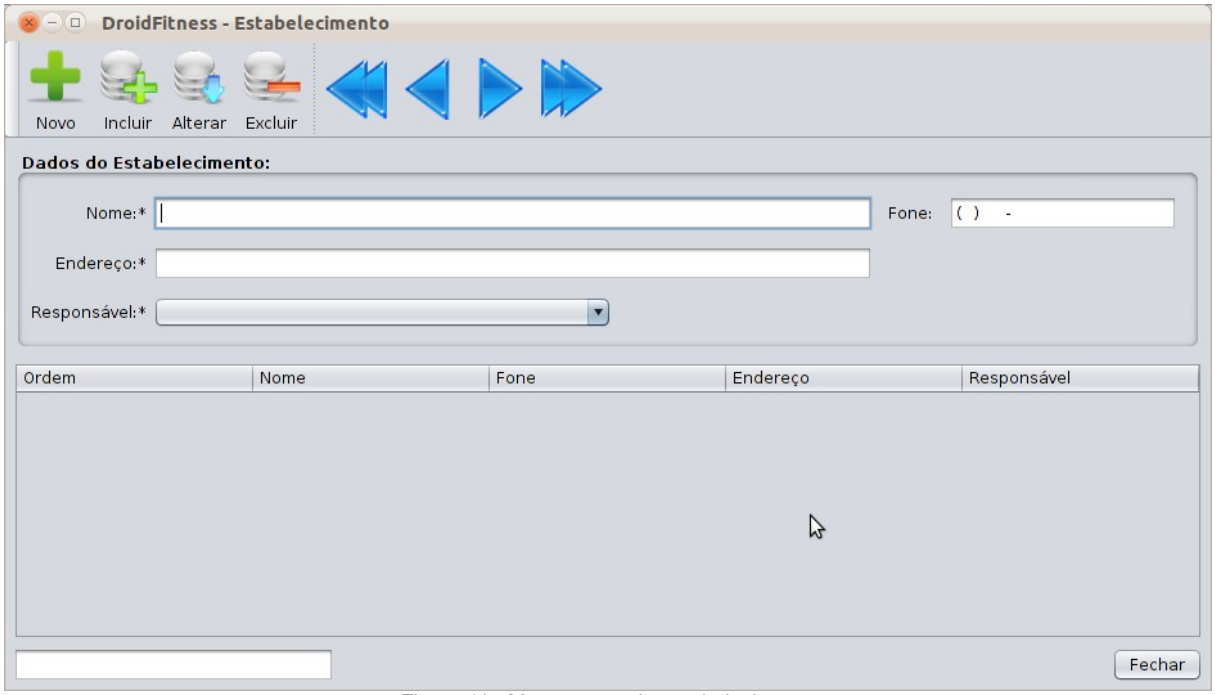

Figura 11 - Manutenção do estabelecimento

5.1.2 Manutenção de Instrutores

O módulo de cadastro de instrutores, possibilita a ação de cadastro de instrutores que farão parte do estabelecimento de ginástica. É representada pela entidade *InstrutorEntity* e está diretamente relacionada com os registros de estabelecimento.A Figura 12 representa a estrutura da tela de cadastro.

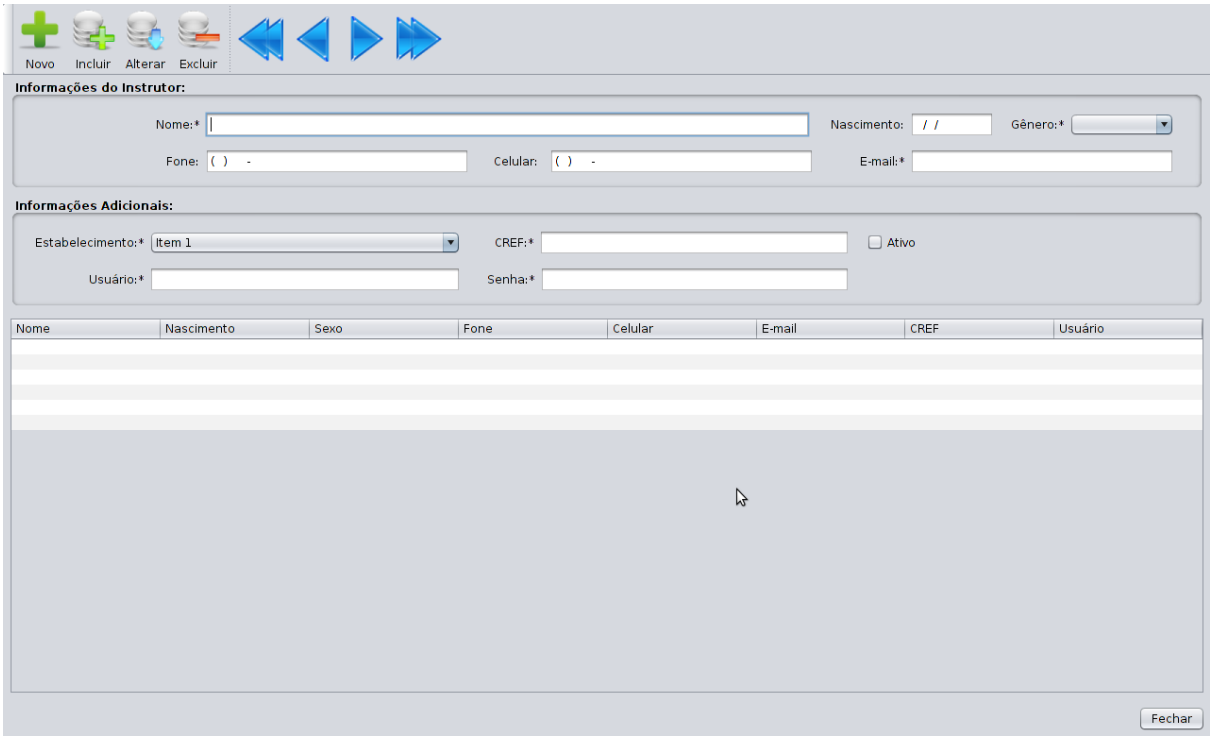

Figura 12 - Manutenção de instrutores

5.1.3 Manutenção de Aparelhos

Neste módulo é possível realizar o cadastro de aparelhos de ginástica que posteriormente irão compor o cadastro de exercícios. É representado pela entidade *AparelhoEntity* e contem as principais informações referentes aos dados dos aparelhos contidos no estabelecimento. A Figura 13 representa a estrutura da tela de cadastro.

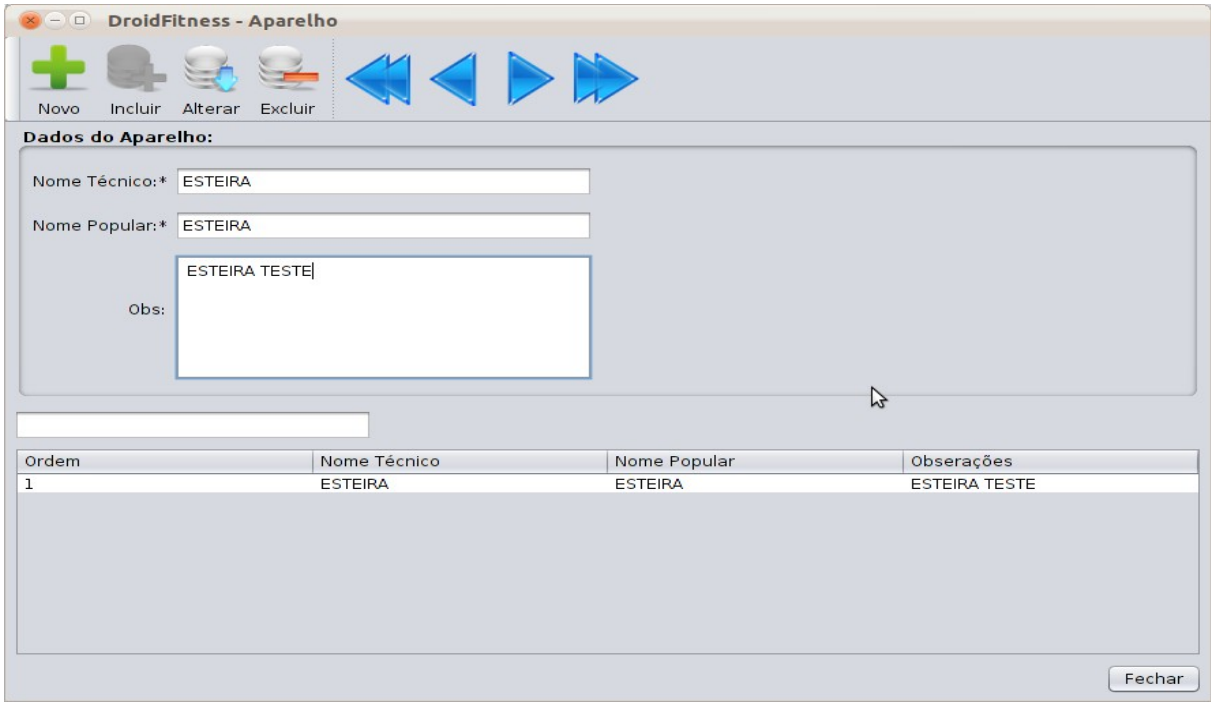

Figura 13 - Manutenção de aparelhos

5.1.4 Manutenção de Grupos Musculares

Módulo responsável por permitir o cadastro de grupos musculares, é representado pela entidade *GrupoMuscularEntity*. Cada região do corpo possui um grupo muscular. A Figura 14 representa a estrutura da tela de cadastro.

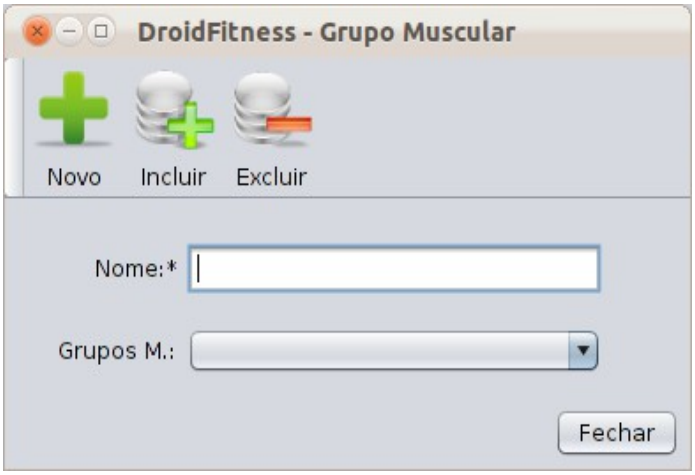

Figura 14 - Manutenção de grupos musculares

5.1.5 Manutenção de Exercícios

No módulo de cadastro de exercícios, é possível criar um novo exercício com base nas informações referentes ao cadastro de Aparelho (Seção 5.1.3) e pelo cadastro de grupos musculares (Seção 5.1.4). É representado pela entidade *ExercícioEntity* sendo que um exercício corresponde a associação de um tipo aparelho para um determinado grupo muscular. A Figura 15 representa a estrutura da tela de cadastro.

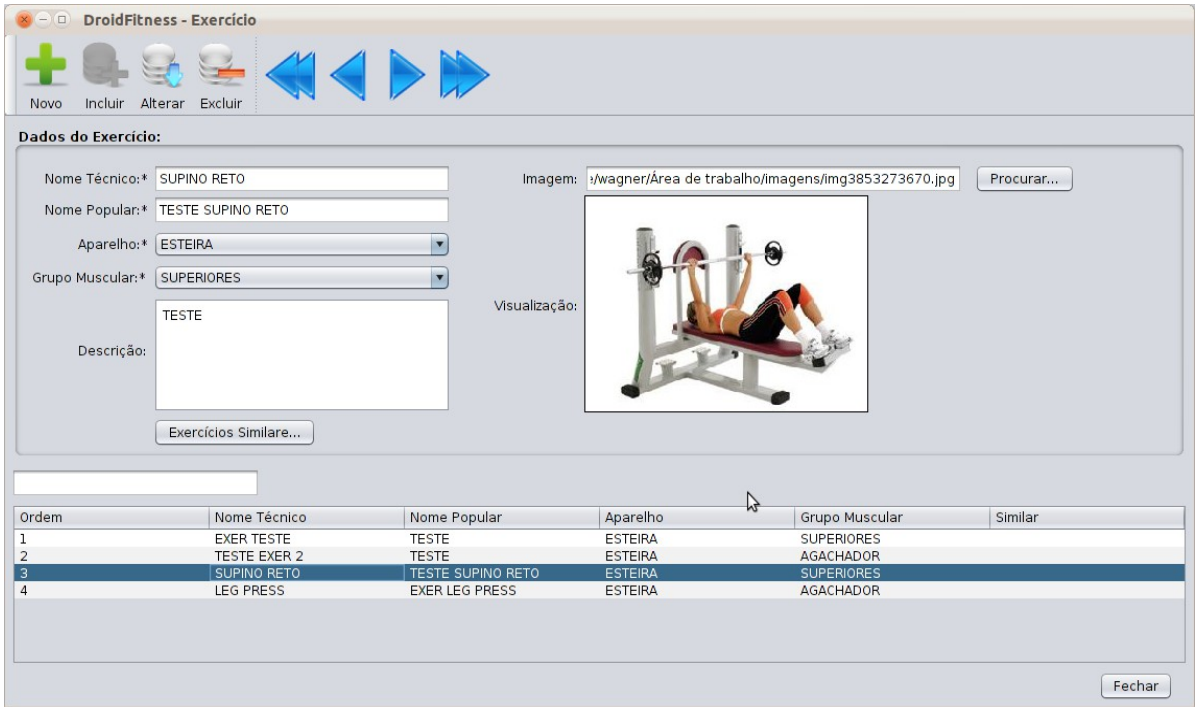

Figura 15 - Manutenção de Exercícios

5.1.6 Manutenção de Séries

Com o módulo de cadastro de séries, permite que o Instrutor do Estabelecimento, crie Séries a partir dos exercícios criados na tela de Cadastro de Exercícios. Séries são constituídas pela quantidade de repetições daquele determinado exercício, e claro, a carga equivalente em quilogramas. Uma imagem

da tela de cadastro de séries pode ser visualizada conforme a Figura 16.

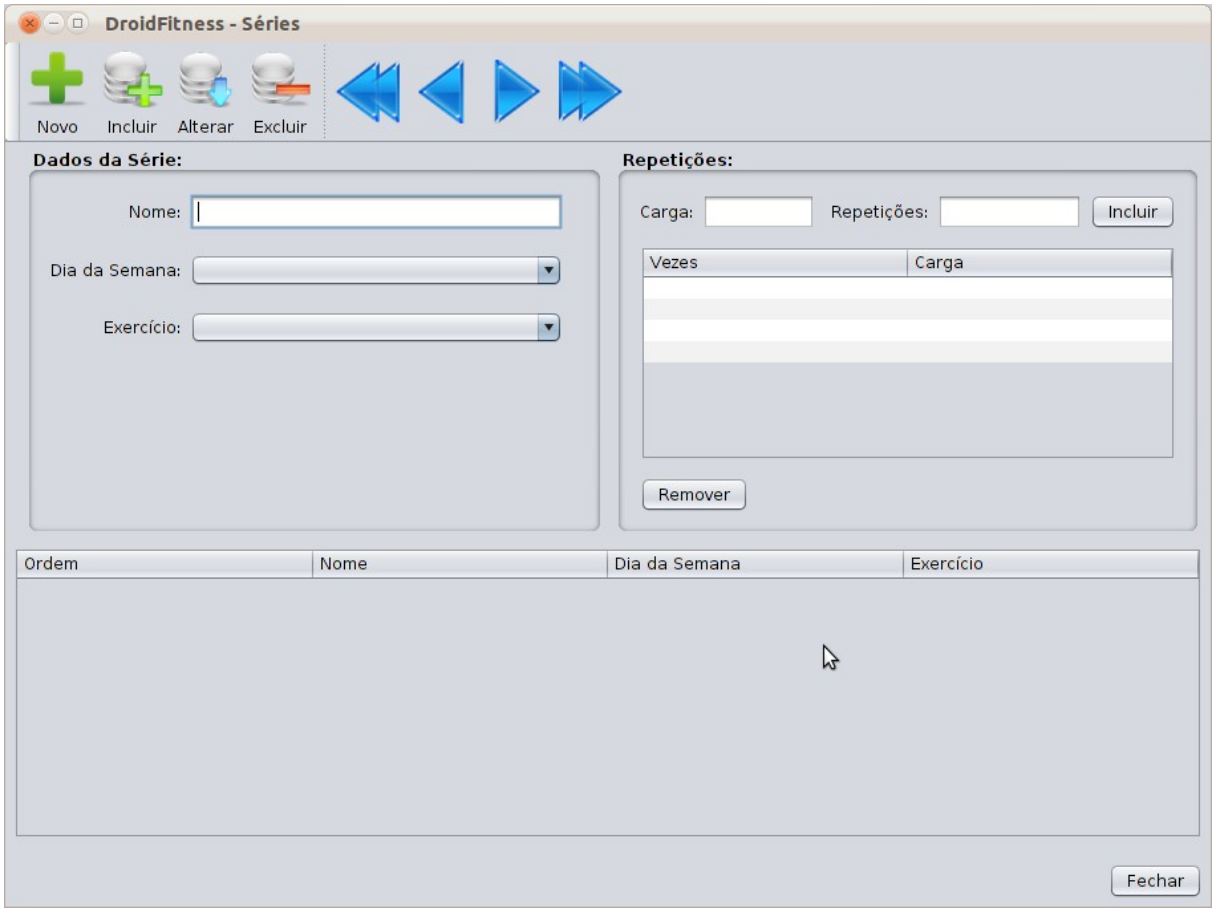

Figura 16 - Manutenção de séries

5.1.7 Manutenção de Treinos

O módulo de cadastro de treinos, permite que sejam cadastrados diferentes tipos de treinos, permitindo associar exercícios com séries personalizadas criadas através da tela de cadastro de séries (Ver seção 5.1.6). Um treino é composto pelo instrutor que irá conduzir o treinamento do aluno, também por sessões de treinamento, ou seja, sessões de treinamento equivalem os dias em que o aluno irá praticar o seu treinamento, por exemplo, treino de manhã, noite, segunda-feira, terça-feira etc. A sessão é composta por um nome e pela escolha das séries de exercícios criadas anteriormente na tela de cadastro de séries. A representação da

tela de cadastro de treino pode ser visto na Figura 17.

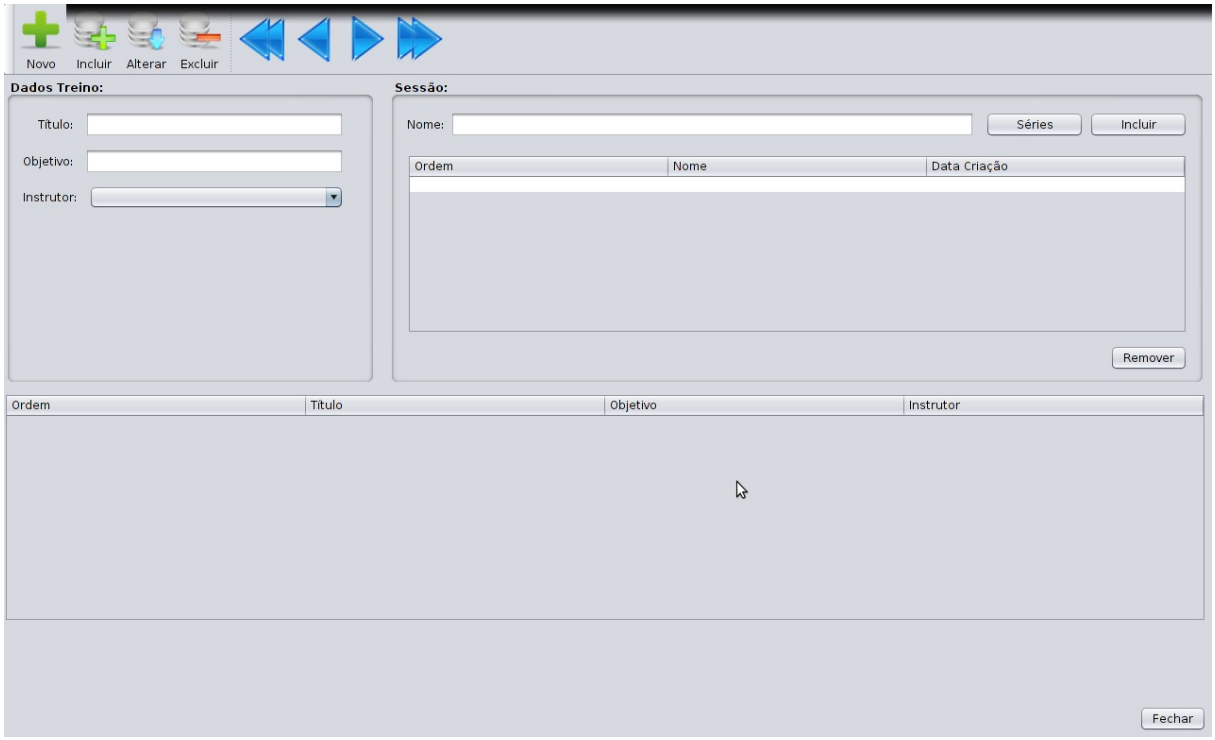

Figura 17 - Manutenção de Treinos

5.1.8 Manutenção de Alunos

O Cadastro de alunos é o módulo responsável por realizar o cadastro e gerenciamento de alunos do estabelecimento. Este módulo permite que sejam cadastrados diversos dados referentes ao aluno, como por exemplo, nome, endereço, e-mail, telefone e um treino que o respectivo aluno irá praticar. No momento do cadastro do aluno é necessário informar o estabelecimento em que o aluno está realizando suas atividades e um treino personalizado criado a partir da tela de cadastro de treinos, a partir deste momento o aluno com sua aplicação móvel, poderá se conectar ao módulo e receber a sua lista de exercícios. Uma representação da tela de cadastro de aluno pode ser vista conforme a Figura 18.

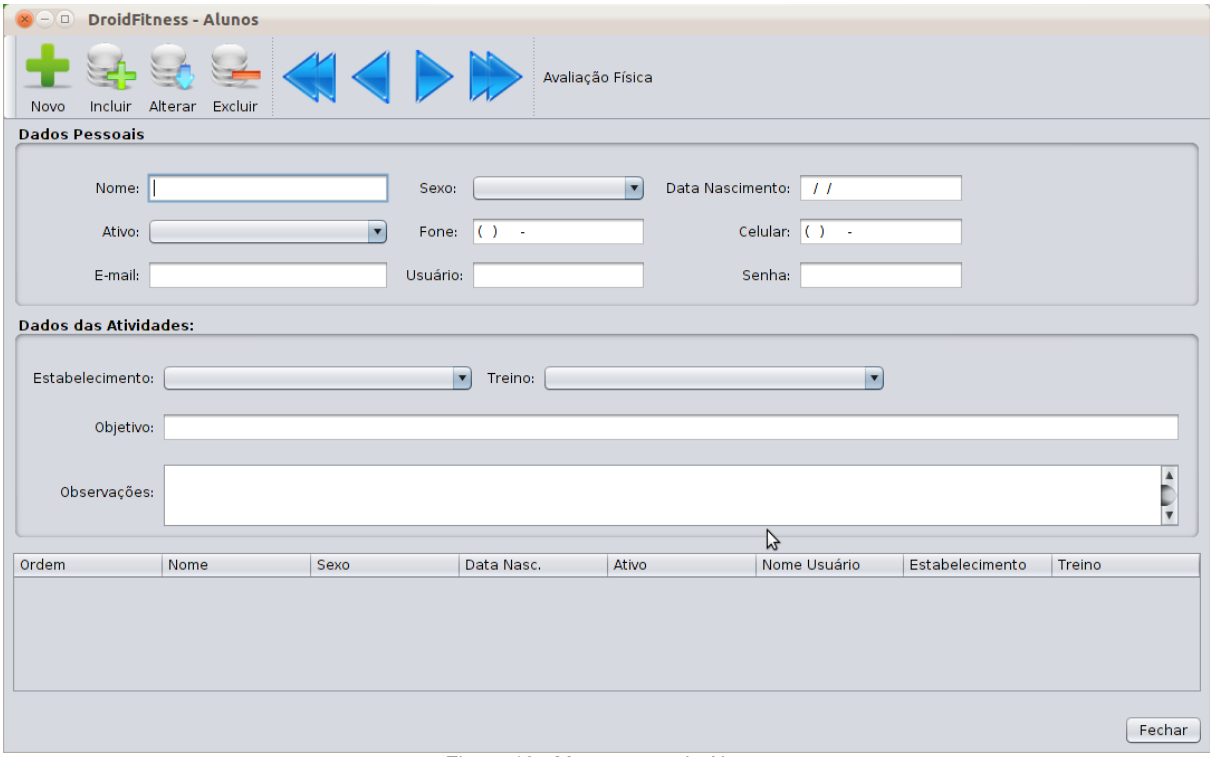

Figura 18 - Manutenção de Alunos

5.1.9 Manutenção de Avaliações Física

O cadastro de avaliações física é um módulo opcional criado especialmente para o aluno possa obter um *feedback*, por parte do seu instrutor, referentes a informações do seu desenvolvimento ao longo do treinamento. É representado pela entidade *AvaliaçãoEntity* e permite que sejam cadastradas diversas informações relacionada à uma avaliação física, como por exemplo, peso, altura, circunferência do tronco, braços e pernas. Dessa forma é possível que o aluno acompanhe os acompanhe através da aplicação cliente, bem como a visualização por meio de gráficos estatísticos. A Figura 19 representa a estrutura da tela de cadastro.

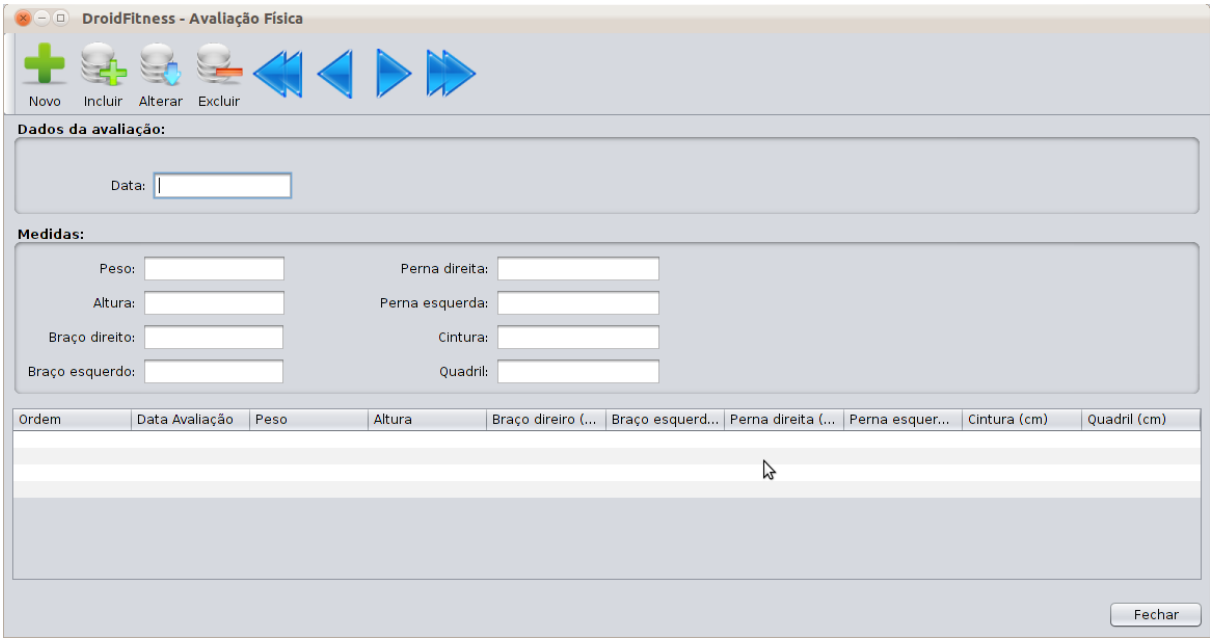

Figura 19 - Manutenção de avaliações físicas

5.2 Módulo Móvel

O modulo móvel foi desenvolvido para Android versão 2.1 ou superior que representa 80% dos dispositivos móveis que utilizam esta tecnologia.

Este módulo tem por finalidade ser um cliente para unicamente visualizar as informações obtidas do módulo retaguarda.

A estrutura do sistema está dividida da seguinte forma:

- **Atualização de dados do treino:** permite o aluno atualizar a sua lista de treinamento em qualquer momento;

- **Atualização de Imagens:** permite ao aluno atualizar as imagens dos exercícios, essa atualização não é necessariamente obrigatória, sendo que o exercício poderá ser normalmente visualizado;

- **Visualização de do Treino:** permite ao aluno visualizar o seu treinamento, com a descrição de dos exercícios, bem como sua imagem e

exercícios similares;

- **Visualização de Avaliações físicas:** permite o aluno visualizar dados por meio de gráficos, principalmente informações a respeito de suas medidas e sua evolução no treinamento.

5.2.1 Permissões do Dispositivo

Para acessar recursos do dispositivo é preciso deixar explicito no arquivo "AndoridManifest.xml" as permissões desejadas. Para as funcionalidades implementadas no *DoidFitness* é necessário permissão para escrever na memória externa e acesso a rede externa. As configurações podem ser vistas na Quadro 6.

```
10 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
11 <uses-permission android:name="android.permission.INTERNET"/>
```

Quadro 6: Arquivo de permissões "AndroidManifest.xml".

5.2.2 Conexão com Banco de Dados

A conexão com com banco é feita utilizando um *singleton*[13] através de uma variável estática que representa uma instancia da própria classe de conexão (Quadro 7).

13 ***Singleton***, é um padrão de projeto de software (do inglês *Design Pattern*). Este padrão garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto (SHALLOWAY; TROTT, 2002 ; p.259).

```
18⊖    private OdbConnect() {
19         File dir = new File(path);
20         dir.mkdirs();
21         odb = ODBFactory.open(path+"/"+dbName,user,pass);
22     }
23
24⊖    public static OdbConnect getInstance(){
25         if (instance == null){
26             instance = new OdbConnect();
27         }
28         return instance;
29     }
```

Quadro 7: Módulo de conexão com a base de dados.

5.2.3 Configurações

A tela de configuração permite que o usuário configure o endereço IP e a porta de comunicação para acessar o servidor (Figura 20).

Figura 20 - Configurações

A janela de configuração de conexão (Figura 20), é montada conforme o código representado pela Quadro 8.

```
65    @Override
66    public boolean onMenuItemSelected(int featureId, MenuItem item) {
67        switch (item.getItemId()) {
68        case id.mConfigurar:
69            new ConfigDialog(this).show();
70            break;
71 //         ...
```

Quadro 8: Montagem de janela de configuração.

Após o usuário informar as configurações necessárias, elas serão armazenadas na base de dados do aplicativo móvel, conforme demostrado na Quadro 9.

```
51    protected void btnSaveOnClick() {
52        if (validaFields()) {
53            config = new Configuration();
54            config.setIp(edtUrl.getText().toString().toLowerCase());
55            config.setPort(Integer.parseInt(edtPort.getText().toString()));
56            new ConfigAction().store(config);
57 //         ...
```

Quadro 9: Persistindo configurações.

5.2.4 Identificação

Toda vez que o usuário solicita uma atualização ele precisa se identificar através de um nome de usuário e uma senha cadastrados na aplicação de gerenciamento. A Figura 21 representa a interface onde o usuário pode realizar o processo de autenticação.

Figura 21 - Autenticação

O Quadro 10 representa o método que realiza o envio das credencias do usuário aluno para o servidor.

```
86      protected void btnConectOnClick() {
87          if (validaCampos()) {
88              String login = edtLogin.getText().toString().toLowerCase();
89              String pass = edtPass.getText().toString();
90              doConnect(hdl,config,new Login(login, pass, Tipo.ALUNO));
91  //          ...
```

Quadro 10: Método de ação do botão de envio de credenciais.

O método "*doConnect()*" é um método abstrato de "*LoginDialog*" que é implementado onde uma janela do tipo "diálogo" é instanciada, conforme visto no Quadro 11.

```
105 LoginDialog loginDlg = new LoginDialog(this){
106     @Override
107     protected void doConnect(LoginHandler hdl, Configuration config,
108             Login login) {
109         login.setRequisisao(Requisicao.EXERCICIOS);
110         new ConnectionMananger(hdl, config, login).start();
111     }
112 };
```

Quadro 11: Método de envio de credenciais.

5.2.5 Atualização de Treino

Na Figura 22 pode ser visualizado o fluxo, obedecendo o protocolo implementado para que as mensagens sejam transferidas através da rede. Esse procedimento permite que o usuário recupere as suas informações do treinamento, através do servidor.

Os dados são atualizados separadamente das imagens permitindo ao usuário escolher o momento que deseja recuperar as imagens, uma vez que esse procedimento demanda maior transferência de dados e pode representar um problema em redes de operadoras de telefonia móvel, gerando demora e custos para o usuário.

Para receber a atualização, o endereço *Internet Protocol* (IP) e porta de aplicação do servidor devem estar corretamente configurados no dispositivo e o usuário precisa realizar sua autenticação, informando nome de usuário e senha que foram cadastrados na aplicação de gerenciamento.

No momento de realizar a atualização dos dados, o aplicativo cliente envia uma mensagem para o servidor com os parâmetros necessários, como nome de usuário e senha, solicitando assim a conexão (de acordo com a Figura 22, passo 1), em seguida no passo 1.1, após o servidor ter realizada a autenticação do cliente, uma mensagem é retornada contendo os dados referentes ao treino.

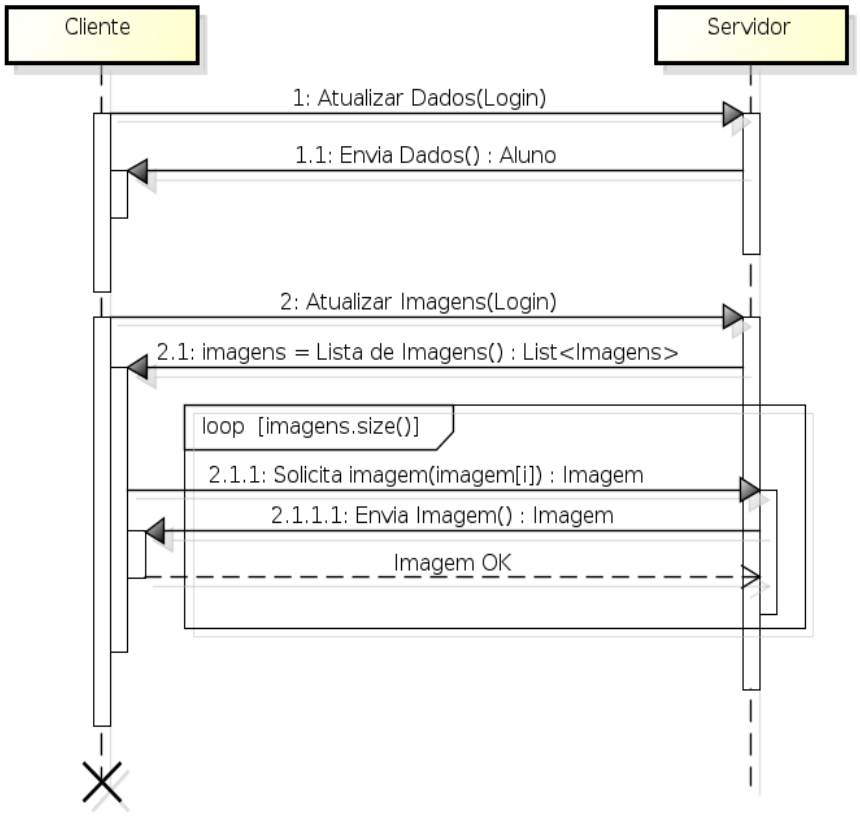

Figura 22 - Sequencia de comunicação Cliente-Servidor

O processo de atualização das imagens do treino, ocorre de maneira separada. No passo 2 o cliente envia uma mensagem contendo as credenciais do usuário, na sequencia o servidor valida essas informações e devolve uma resposta ao cliente (passo 2.1). O processo transferência ocorre através de um *loop*, isto é, somente uma imagem é trafegada por vez, sendo que o procedimento se repete até todas as imagens terem sido recebidas pelo aparelho móvel.

O quadro 12, é demonstrado através de linhas de código fonte, o módulo do aplicativo cliente, utilizado para realizar a comunicação com o servidor.

```
56      @Override
57      public void run() {
58  //      ...
59
60          //Envia login ao servidor
61          out.writeObject(login);
62          //Agurada resposta
63          Object ret = in.readObject();
64
65          if (ret instanceof Aluno) {
66              //Se a resposta for um Aluno
67              OdbConnect.deleteDB();
68              AlunoAction act = new AlunoAction();
69              act.store((Aluno) ret);
70
71  //          ...
```

Quadro 12: Recuperando dados do treino e persistindo na base de dados.

No Quadro 13 é mostrada a estrutura do código fonte do módulo servidor, responsável por processar as requisições dos dispositivos clientes.

```
public void run() {
    try {
        out = new ObjectOutputStream(socket.getOutputStream());
        in = new ObjectInputStream(socket.getInputStream());

        Login login = (Login) in.readObject();
        System.out.println(login.getUsuario());

        if (DEBUG) {
            aluno = getAluno(login.getUsuario(), login.getSenha());
        } else {
            try {
                aluno = new PreparaObjetoCliente().getObjetoAluno(login.getUsuario(), login.getSenha());
            } catch (Exception ex) {
                Logger.getLogger(ServidorSocket.class.getName()).log(Level.SEVERE, null, ex);
            }
        }

        if (aluno == null) {
            out.writeObject(new Exception("Usuário ou senha inválido(s)"));
        } else {
            switch (login.getRequisisao()) {
                case EXERCICIOS:
                    out.writeObject(aluno);
                    break;
                case IMAGENS:
                    enviaImagens(extractImageList(aluno));
                    break;
            }
```

Quadro 13: Módulo servidor de conexões cliente.

No quadro 14, é apresentada a estrutura do código do método responsável por fazer o processamento e envio das imagens dos exercícios.

```
private void enviaImagens(List<Imagem> imgs) throws IOException {
    out.writeObject(imgs);
    int n = imgs.size();
    int idx = 0;
    FileInputStream fIs;

    byte[] buffer = new byte[socket.getSendBufferSize()];
    while (idx < n) {
        Imagem img = imgs.get(idx);


        if (img.getFile().exists()) {
            File arq = img.getTumbFile();
            int offset = 0, len1 = 0;
            img.setTumbSize(arq.length());
            out.writeLong(arq.length());
            out.writeObject(img);
            out.flush();
            fIs = new FileInputStream(arq);
            while (offset < arq.length() && (len1 = fIs.read(buffer)) != -1) {
                out.write(buffer, 0, len1);
                offset += len1;
            }
            out.flush();
            boolean ok = in.readBoolean();
            if (ok) {
                idx++;
            }
        }
    }
    out.writeObject(null);
    out.flush();
```

Quadro 14: Módulo de envio de imagens.

No Quadro 15, pode ser visualizada a implementação do módulo cliente responsável por receber as imagens do servidor.

```
76 @Override
77 protected Boolean doInBackground(Login login) {
78             //Login
79             out.writeObject(login);
80             out.flush();
81             Object ret = in.readObject();
82             if (ret instanceof List) {
83                 //Se o retorno for uma lista
84                 List<Imagem> list = (List<Imagem>) ret;
85                 Imagem img = (Imagem) in.readObject();
86                 while (img != null && !"FIM.eof".equals(img.getName())) {
87                     boolean ok = receberArquivo(img);
88                     out.writeBoolean(ok);
89                     out.flush();
90                     img = (Imagem) in.readObject();
91                 }
92 //          ...
```

Quadro 15: Método de recebimento de imagens.

Como visto, os módulos apresentam claramente a implementação dos objetos de conexão através de Soquetes, que são utilizados para a transferência das informações entre cliente-servidor.

5.2.6 Visualização de Séries

Esta é a atividade foco de todo o sistema onde o usuário pode navegar e acompanhar o treino, contendo os exercícios de cada dia e visualizar informações sobre a execução dos mesmos (Figura 23).

Figura 23 - Visualização de séries

A Quadro 16 representa o código necessário para recuperar dados da base de dados para posteriormente apresentar na tela da aplicação móvel.

```
62 // ...
63 List<Aluno> list = new AlunoAction().getAll();
64 // ...
```

Quadro 16: Recuperando objetos da base de dados.

O método "*getAll*" é genérico, ou seja, o método de recuperação das informações sempre será o mesmo, independente do tipo de dado recuperado da base de dados (Quadro 17).

```
18    public List<T> getAll() {
19        Objects<T> objs = getDao().getAll();
20        List<T> list = new ArrayList<T>();
21        while (objs.hasNext()) {
22            list.add(objs.next());
23        }
24        return list;
25    }
```

Quadro 17: Método genérico de recuperação de dados da base de dados.

5.2.6.1 Visualização de Informações e Exercícios Similares

Permite ao usuário consultar exercícios alternativos que tem a mesma finalidade motora (Figura 24). Bem como visualizar informações detalhadas sobre o exercício e a forma correta de se executá-lo (Figura 25).

Figura 24 - Exercícios similares

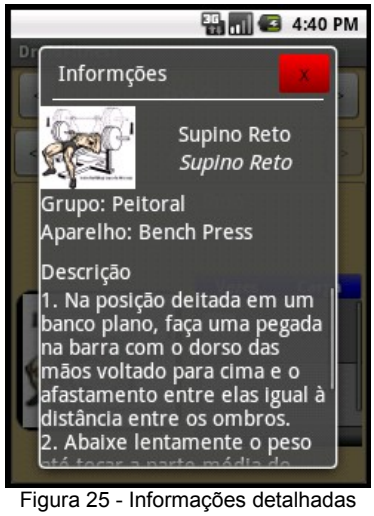

Figura 25 - Informações detalhadas

5.2.6.2 Visualização de Avaliações

Na visualização de avaliações o usuário pode acompanhar a evolução de suas medidas e resultados.

Como pode ser visto na Figura 26, mostra as informações referentes a última avaliação física a qual o aluno se submeteu. O cálculo do IMC é realizado automaticamente através da equação: IMC = peso / altura².

Nesta atividade também são disponibilizados cinco gráficos para visualizar a evolução dos resultados avaliados.

Figura 26 - Visualização das avaliações

5.2.6.3 Variação da Massa Corporal (Peso) e IMC

O sistema mostra graficamente a variação da massa corporal (Figuras 27 e 28). O peso ideal é calculado a partir do IMC ideal que é tabelado entre (18,5 e 24,9 Kg/m²)[14]

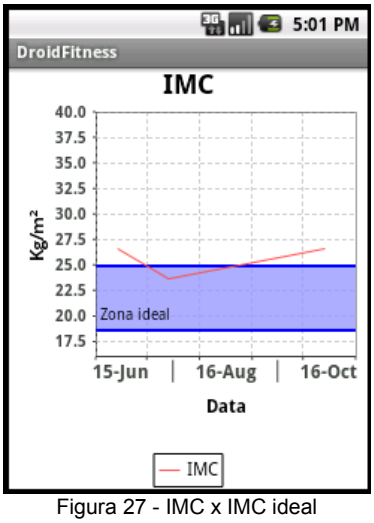

Figura 27 - IMC x IMC ideal

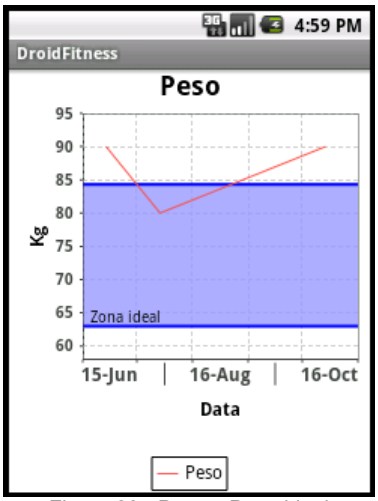

Figura 28 - Peso x Peso ideal

14 O Índice de Massa Corporal (IMC) é uma medida internacional usada para calcular se uma pessoa está no peso ideal (OMS, 2012);

5.2.6.4 Variação das Medidas Corporais (Tronco e Membros)

Tem por objetivo mostrar graficamente a evolução das medidas corporais avaliadas: braços, coxas, cintura e quadril, conforme pode ser visualizado na Figura 29.

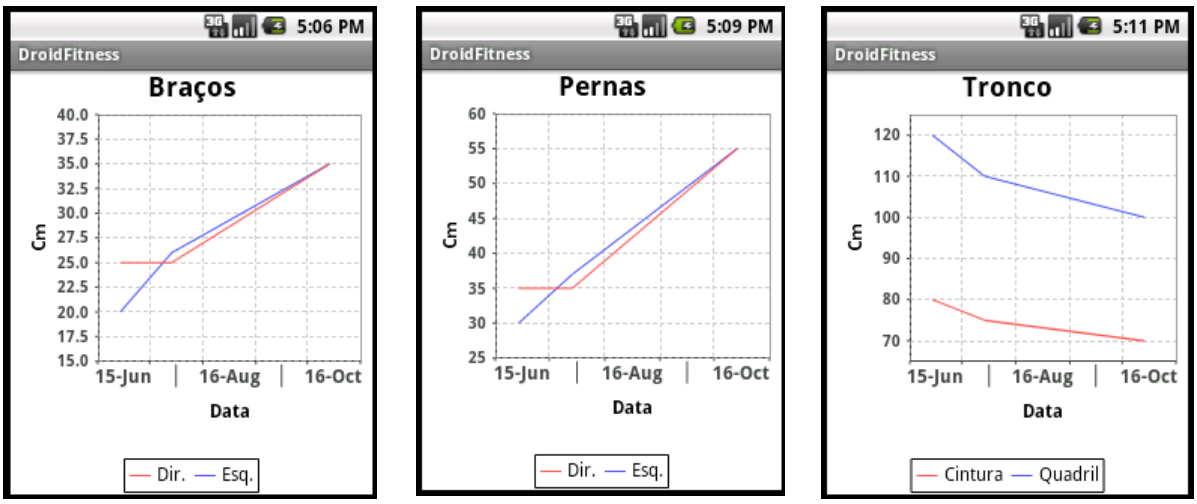

Figura 29 - Variação de medidas corporais

Com base nas avaliações realizadas é possível fazer uma comparação das medidas corporais. Os gráficos representados na Figura 39, representam essa comparação, onde no eixo x é representada a data das avaliações físicas realizadas, logo no eixo y, as medidas. Com esses dados, várias linhas demostram a variação da evolução.

O Quadro 18 apresenta por meio de código a montagem de parâmetros dos gráficos de comparação das medidas corporais.

```
47      AFreeChart chart = ChartFactory.createXYLineChart(
48              "Braços",                   // Titulo do gráfico
49              "Data",                     // Eixo "x"
50              "Cm",                       // Eixo "Y"
51              getDataSet(Tipo.BRACO),     // Dados
52              PlotOrientation.VERTICAL, // Orientação
53              true,                       // Legenda
54              true,                       // ToolTip
55              false);                     // url
56      XYPlot plot = (XYPlot) chart.getPlot();
57  //    ...
```

Quadro 18: Mongem de parâmetros dos gráfico de comparação de medidas corporais.

O Quadro 19 representa a recuperação de dados para montagem dos dados dos gráficos de medidas corporais.

```
192 private XYDataset getDataSet(Tipo tipo) {
193     switch (tipo) {
194     case BRACO:
195         xyS1 = new TimeSeries(getContext().getString(string.R));
196         xyS2 = new TimeSeries(getContext().getString(string.L));
197         for (int i = 0; i < avaliacoes.size(); i++) {
198             float d = avaliacoes.get(i).getBracoD();
199             float e = avaliacoes.get(i).getBracoE();
200             Day data = new Day(avaliacoes.get(i).getData(i-(avaliacoes.size()-1)));
201             if (d > 0 && e > 0) {
202                 xyS1.add(data, d);
203                 xyS2.add(data, e);
204             }
205         }
206         break;
207 //      ...
```

Quadro 19: Recuperação dos dados para montagem de parâmetros dos gráficos de medidas corporais.

6 CONCLUSÕES

Este trabalho teve como objetivo, apresentar uma solução, de forma simplificada e ágil para a questão da falta de acompanhamento técnico que os alunos de centros de atividades desportivas enfrentam atualmente. No decorrer do desenvolvimento, foi mantido o foco em diversos quesitos como, elaborar uma solução em que os instrutores mantivessem de forma simplificada uma rotina personalizada de treinamento para seus alunos, tendo prioridade a boa usabilidade do sistema, eficiente disponibilidade das informações e utilizando tecnologias atuais e consolidadas no mercado para seu desenvolvimento.

Foram identificados diversos requisitos e casos de uso nas regras de negócio em que o processo de funcionamento de um estabelecimento desportivo desenvolve, e a melhor maneira de implementá-los, levando em conta as regras de negócio existentes no processo analisado.

A partir dos estudos foi possível perceber que, os alunos ao manter uma base com todas as informações necessárias, podem tornar as suas atividades de treinamento muito mais eficientes e com resultados muito mais satisfatórios, já que o acompanhamento por parte do instrutor é de fundamental importância para o sucesso do treinamento, melhorando dessa forma, a saúde e o bem estar do praticante.

6.1 Vantagens

Como vantagens em relação as tradicionais fichas de papel, o *DroidFitness* apresenta os seguintes itens:

- **Disponibilidade**: Onde quer que o aluno esteja, as informações do seu treino estarão sempre disponíveis e atualizadas, diferente das fichas de papel, em que o aluno precisa ir até a academia para buscar uma nova e atualizada.

- **Concentração das informações:** as informações referentes ao

treino do aluno estarão somente concentradas em seu aparelho móvel, podendo dessa forma consultar séries antigas, logo nas fichas de papel, pode ser mais complicado de localizar.

- **Praticidade**: Não existe o inconveniente de sujar ou rasgar as tradicionais fichas de papel, já que seus dados estarão somente disponíveis em meio eletrônico.

- **Descrição detalhada**: Pelo dispositivo móvel é possível além de visualizar informações em modo texto, também é possível ver imagens de exercícios, o que permite maior compreensão por parte do aluno no momento da realização do exercício.

6.2 Desvantagens

Como desvantagens podem ser observados os seguintes itens:

- **Compatibilidade**: Para que o aluno consiga praticar seu treinamento é necessário que ele possua um dispositivo móvel, compatível com a tecnologia utilizada (Android) no desenvolvimento do sistema.

- **Médio custo**: Equipamentos com Android ainda possuem valores um pouco elevados no mercado.

- **Conectividade**: É necessário que o aluno possua uma conexão com a Internet para recuperar as informações do servidor, e o serviço de conexão, fornecido pela maioria das operadoras de telefonia móvel ainda é de custo elevado.

6.3 Perspectivas Futuras

Futuramente numa possível comercialização do sistema, pode ser desenvolvidos módulos e funcionalidades que fazem com que seja agregado valor ao sistema, como por exemplo:

- **Avaliação Física:** Atualmente o sistema possibilita apenas cadastrar dados referentes à avaliações físicas geradas por sistemas de terceiros, ou mesmo ainda obtidas através cálculos manuais. Pode ser desenvolvido um módulo que realize automaticamente um resultado de avaliação física a partir de dados coletados, e assim, disponibilizado para o aluno.

- **Relatórios estatísticos**: pode ser desenvolvido relatórios estatísticos, que mostrem para o aluno a sua evolução no treinamento, por meio de gráficos ou imagens, geradas a partir de dados coletados.

- **Cadastro de treino no dispositivo móvel**: Esta funcionalidade permite ao aluno, que ele cadastre seu próprio treinamento pelo aparelho, uma vez que o instrutor da academia pode estar ausente no momento.

7 REFERÊNCIAS

ACSM, **American College of Sports Medicine; American Heart Association**; British Medical Association, Themudo e col., 1997.

ACSM. **Position Stand on Progression Models in Resistance Training For Healthy Adults** / American College of Sports Medicine. Med Sci Sports Exerc: 2002.

AFREECHART. **Bibliotega de Gráficos do Tipo *Chart* Para Android**. Disponível em <http://code.google.com/p.afreechart/> Acesso em 14 Outubro. 2012.

ASTAH. **Ferramenta de modelagem UML**. Disponível em <http://astah.net/> Acesso em: 25 de Setembro. 2012.

BEZERRA, Eduardo. **Princípios de Análise e Projeto de Sistemas com UML** / Eduardo Bezerra. - Rio de Janeiro : Elsevier, 2007.

BOOCH, G; RUMBAUGH, J.; JACOBSON, I. **UML – Guia do Usuário** / Grady Booch, James Rumbaugh, Ivar Jacobson; Rio de Janeiro - RJ: Elsevier, 2005.

BORATTI, Isaias C. **Programação Orientada a Objetos Usando Delphi** / Isaias Camilo Boratti. Florianópolis: Visual Books, 2007.

CASPERSEN, C. J.; POWELL, J. E.; CHRISTENSON, G. M. **Physical activity, exercise, and physical fitness**: definitions and distinctions for health-related research. Public Health Re- ports. 1985.

COAD, Peter. **Análise Baseada em Objetos** / Peter Coad; Rio de Janeiro - RJ: Editora Campus, 1991.

COCKBURN, Alistair. **Escrevendo Casos de Uso Eficazes** / Alistair Cockburn; Porto Alegre – RS: Artmed, 2001.

COMER, D. E.; DROMS, R. E. **Redes de Computadores e Internet** / Douglas Earl Comer, Ralph Droms; Porto Alegre – RS: Artmed, 2007.

CORREIA, C. H.; TAFNER, M. A. **Análise Orientada a Objetos.** / Carlos Herique Correia, Malcon Anderson Tafner. Florianópolis: Visual Books, 2006.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas Distribuídos: conceitos e projeto** / George Coulouris, Jean Dollimore, Tim Kindberg ; tradução João Tortello. - 4 ed. - Porto Alegre : Bookman, 2007.

DEBONI, José. E. Z. **Modelagem Orientada a Objetos com a UML** / José Eduardo Zindel Deboni. São Paulo: Futura, 2003.

ECLIPSE. **Comunidade Eclipse**. Diponível em: <http://www.eclipse.org/>. Acesso em 16 Outubro 2012.

ELMASRI, R.; NAVATHE, S. B. **Sistema de Banco de Dados** / Ramez Elmasri, Sham Navathe. São Paulo: Pearson Addison Wesley. 2005.

EMBARCADERO. **Linguagem Delphi**. Disponível em: <http://www.embarcadero.com.br/>. Acesso em: 25 Setembro. 2012.

ESTADÃO, Jornal**. Brasil só perde para EUA em número de academias**. Disponível em: <http://www.estadao.com.br/noticias/impresso,brasil-so-perde-para-eua-em-numero-de-academias,585706,0.htm>. Acesso em 06 de Outubro. 2012.

FANTINEL, Leandro. **Especificação de Casos de Uso do Sistema de Acompanhamento de Atividade Físicas – DroidFitness**. Leandro Fantinel. Pato Branco, 2012.

FEOFILOFF, Paulo. **Algoritmos** / Paulo Feofiloff. Rio de Janeiro - RJ: Elsevier. 2009.

FERRARI, Fabrício A. **Crie Banco de Dados em MySQL** / Fabrício Augusto Ferrari. São Paulo - SP: Digerati Books, 2007.

FITZGERALD, Michael. **Learning Ruby** / Michael Fitzgerald; Gravenstein highway North – EUA: O' Reilly, 2007.

GNU. **The GNU C Library (glibc)**. Disponível em: <http://docs.oracle.com/javase/1.5.0/docs/guide/awt/index.html>. Acesso em: 25 Setembro. 2012.

GOOGLE, Developers. **Página oficial de desenvolvedores Google**. Disponível em <https://developers.google.com/?hl=pt-br>. Acesso em 16 Outubro 2012.

GOOGLE. **Sistema Operacional Android**. Disponível em: <http://www.android.com/>. Acesso em: 25 Setembro. 2012.

GOOVY. **Projeto Groovy**. Disponível em: <http://groovy.codehaus.org/>. Acesso em: 25 Setembro. 2012.

HOUAISS, Antonio. **Grande Dicionário Houaiss da Língua Portuguesa** / Antonio Houaiss. São Paulo - SP : Objetiva, 2012.

HUBBARD, JOHN R. **Programação em C++** / John Randolph Hubbard; Porto Alegre – RS: Artmed. 2000.

JFREECHART. Biblioteca para geração de de gráficos. Disponível em: <http://www.jfree.org/jfreechart/>. Acesso em 06/11/2012.

KHOSHAFIAN, Setrag. **Banco de Dados Orientado a Objetos** / Setrag Khoshafian; Rio de Janeiro - RJ: Infobooks, 1999.

LERDORF, Rasmus. **Programing PHP** / Rasmus Lerdorf; Gravenstein highway North – EUA: O' Reilly, 2002.

MATTOS, Érico C. T. de. **Programação de Software em Java /** Érico Casella Tavares de Mattos; São Paulo: Digerati Books, 2007.

MICROSOFT. **Microsoft .Net Framework.** Disponível em: <http://www.microsoft.com/net>. Acesso em: 25 Setembro. 2012.

MOREIRA, Carla. **Prescrição do Exercício Físico**. Disponível em: <html://obesidade.info/prescricaoaf.htm>. Acesso em 06 de Outubro. 2012.

NASSU, E. A.; SETZER, V. W. **Bancos de Dados Orientados a Objetos** / Eugenio Akihiro Nassu, Valdemar Waingort Setzer; São Paulo - SP: Edegard Blücher, 1999.

NEODATIS. **NeoDatis Object Database**. Disponível em: <http://neodatis.wikidot.com/documentation>. Acesso em: 29 mar. 2012.

NETBEANS. **Net Beans IDE**. Disponível em: <http://netbeans.org/>. Acesso em: 25 Setembro. 2012.

NUNES, Jhom. **Prevalência de sedentarismo e prontidão para atividade física de lazer em funcionários públicos** - Anais do Congresso Centro-Oeste de Educação Física, Esporte e Lazer / Jhom Nunes; Brasília - DF, 1999.

OMS. **Organização Mundial da Saúde**. Disponível em: <http://www.who.int/en/index.html>. Acesso em 12 Novembro. 2012.

ORACLE. **Abstract Window Toolkit** (AWT). Disponível em: <http://docs.oracle.com/javase/1.5.0/docs/guide/awt/index.html>. Acesso em: 25 Setembro. 2012.

ORACLE. **Plataforma Java**. Disponível em: <http://www.oracle.com/br/technologies/java>. Acesso em: 25 Setembro. 2012.

PEREIRA, L. C.; SILVA, M. L. **Android para Desenvolvedores** / Lúcio Camilo Pereira, Michel Lourenço da Silva. Rio de Janeiro - RJ: Brasport, 2009.

RICARTE, Ivan. L. M. **Programação Orientada a Objetos: Uma Abordagem com Java** / Ivan Luiz Marques Ricarte; Unicamp: Campinas - SP, 2001.

SAMPAIO, Cleuton. **Guia do Java: Enterprise Edition 5: desenvolvendo aplicações corporativas** / Cleuton Sampaio; Rio de Janeiro : Brasport, 2007.

SCALA. **Projeto Scala**. Disponível em: <http://www.scala-lang.org/>. Acesso em: 25 Setembro. 2012.

SCOTT, Michael L. **Programming Language Pragmatics** / Michael Lee Scott; Nova York - EUA: Elsevier, 2006.

SEBESTA, Robert W. **Conceitos de linguagens de programação** / Robert W Sebesta; Porto Alegre: Bokman, 2003.

SERSON, Roberto. R. **Programação Orientada a Objetos com Java** / Roberto Rubstein Serson; Rio de Janeiro : Brasport, 2007.

SHALLOWAY, A.; TROTT, J. R. **Explicando Padrões de Projeto. Uma nova perspectiva em Projeto Orientado a Objeto** / Alan Shalloway, James Trott; São Paulo – SP: Bookman, 2002.

SILBERSCHATZ, Abrahan. **Sistemas Operacionais com Java** / Abrahan Silberschatz, Peter Baer Galvin, Greg Gagne; Rio de Janeiro : Elsevier, 2008.

TANENBAUM, A. S; STEEN, M. V. **Sistemas Distribuídos: Princípios e**

Paradigmas / Andrew Stuart Tanenbaum, Maarten Van Steen. 2 ed.; Pearson, 2006.