UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DESENVOLVIMENTO DE UM APLICATIVO PARA IPHONE INTEGRADO AO GOOGLEHEALTH COMO FERRAMENTA DE AUXÍLIO AO CLINICSPACE

TRABALHO DE GRADUAÇÃO

Bruno Gallina Apel

Santa Maria, RS, Brasil

2010

DESENVOLVIMENTO DE UM APLICATIVO PARA IPHONE INTEGRADO AO GOOGLEHEALTH COMO FERRAMENTA DE AUXÍLIO AO CLINICSPACE

por

Bruno Gallina Apel

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Prof. Giovani Rubert Librelotto

Trabalho de Graduação N. 286

Santa Maria, RS, Brasil

2010

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

DESENVOLVIMENTO DE UM APLICATIVO PARA IPHONE INTEGRADO AO GOOGLEHEALTH COMO FERRAMENTA DE AUXÍLIO AO CLINICSPACE

elaborado por
Bruno Gallina Apel

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Prof. Giovani Rubert Librelotto
(Presidente/Orientador)

Profª. Iara Augustin (UFSM)

Profª. Márcia Pasin (UFSM)

Santa Maria, 04 de Janeiro de 2010.

DEDICATÓRIA

Dedico este trabalho à minha família, que nunca deixou de acreditar em mim e aos meus amigos, que sempre estiveram ao meu lado.

AGRADECIMENTOS

Gostaria de agradecer, em primeiro lugar, à minha família por me incentivar e aguentar meu "gênio difícil". A minhã mãe, por sempre querer ajudar a sua maneira, ao meu pai por confiar cegamente em mim e em meu potencial, a minha irmã, por, na maioria das vezes, compartilhar do meu jeito de pensar. Agradeço também ao meu avô, por ser o modelo a quem me espelho e a minha avó, por todo o carinho que recebo.

Não posso deixar de agradecer aos meus colegas e amigos, que me acompanharam ao longe desta jornada, seja em épocas conturbadas de muito estudo e noite em claro, o Vielmo sabe bem o que é isso, como nos momentos de descontração, festas e "eventos *nerds*", onde posso citar o Vinicius, o Cristo, o Adler, o Breno, o Ceretta... ah e o Magoo e futebolzinho constante das sextas-feiras.

Tenho que reservar um espaço nesta sessão de agradecimentos para o PET, pois, foi através dele que conheci muitas pessoas e cultivei amizades as quais levarei por toda minha vida, então Mártin, Felipe, Luiz e Maurício, Silvana, Greice e Camila esse espaço é de vocês. E sem esquecer o pessoal da Computação, o Fred e o Garcia (quando não esta "enxendo o saco"da gurizada) e em especial a professora Andrea, e agora ao Giovani, por me aconselhar e indicar a direção certa quando o caminho estava meio obscuro. E a Carol, pelas discuções e frequentes tira-dúvidas que ela se submetia durante a criação deste trabalho. A todos aqueles que deveriam estar neste parágrafo, mas por esquecimento meu ou outro motivo não estão, meu mais sincero muito obrigado.

"Se o conhecimento pode criar problemas, não é através da ignorância que podemos solucioná-los."

— Isaac Asimov

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

DESENVOLVIMENTO DE UM APLICATIVO PARA IPHONE INTEGRADO AO GOOGLEHEALTH COMO FERRAMENTA DE AUXÍLIO AO CLINICSPACE

Autor: Bruno Gallina Apel
Orientador: Prof. Giovani Rubert Librelotto
Local e data da defesa: Santa Maria, 04 de Janeiro de 2010.

O objetivo desse trabalho consiste em criar um aplicativo na área de saúde que sirva de modelo para uma futura análise na criação de interfaces de usuário para dispositivos móveis, a serem utilizadas no projeto ClinicSpace.

Esse aplicativo deverá permitir o acesso a informações referentes a pacientes cadastrados em um sistema. Para o armazenamento dessas informações utilizou-se o serviço Google Health, que organiza seus dados de acordo com o padrão *Continuity of Care Record* (CCR), definido por especialistas como um modelo a ser seguido para possibilitar uma continuidade no tratamento de seus pacientes

O dispositivo móvel escolhido para ser utilizado na implementação foi o iPhone, telefone celular da Apple. Aplicativos para esta plataforma são criados utilizando-se a linguagem de programação Objective-C em conjunto com as ferramentas disponibilizadas no SDK do próprio iPhone.

Palavras-chave: Aplicações móveis; iPhone; GoogleHealth; objetive C; ambiente hospitalar; acessibilidade.

ABSTRACT

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

DEVELOPMENT OF AN APPLICATION FOR IPHONE INTEGRATED WITH GOOGLEHEALTH AS A TOOL TO COOPERATION WITH THE CLINICSPACE

Author: Bruno Gallina Apel
Advisor: Prof. Giovani Rubert Librelotto

The purpose of this work is to create a medical applicative that way serve as a model for a future analysis on the creation of user interfaces for mobile devices, which will be used on ClinicSpace project.

This applicative should allow access to information of patients registered in a system. For the data storage, the Google Health service has been used. It organizes its information according to CCR standard, which is defined by experts to allow the continuity os patient treatment.

The mobile device chosen to store the applicative developed is the Apple's iPhone. Applications for this platform are created using the Objective-C programming language and the iPhone SDK tools.

Keywords: Mobile Applications, iPhone, GoogleHealth, Objetive C, hospital environment, accessibility.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CCR	Continuity of Care Record
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
ID	Identification
J2ME	Java 2 Micro Edition
MacOS	Mac Operating System
MIDP	Mobile Information Device Profile
MVC	Model-View-Controller
PDA	Personal Digital Assistants
SDK	Software Development Kit
UIKit	User Interface Kit
URL	Uniform Resource Locator
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

A computação Pervasiva (WEISER, 1991) tem por objetivo tornar a computação invisível ao usuário, fazendo com que este foque apenas na realização de suas atividades e não na utilização dos recursos e aplicações computacionais. Para isso, ela prevê a comunicação entre dispositivos, recursos e aplicações do ambiente de forma pró-ativa para auxílio ao usuário, sem que este tenha que se preocupar com instalação ou configuração.

A proposta de computação pervasiva vem ao encontro das necessidades da área da saúde, onde os profissionais não podem perder tempo de atendimento ao paciente na utilização de sistemas convencionais, pois necessitam de informações relevantes em tempo real para tomada de decisões.

A maioria dos atuais sistemas de registro eletrônico de informações do paciente não são desenvolvidos tendo como foco a visão do profissional da saúde (médicos e clínicos) e sim na abordagem de gerência e administração (VICENTINI; MACHADO; AUGUSTIN, 2009). Por outro lado o Grupo de Sistemas de Computação Móvel da UFSM, através do projeto *ClinicSpace: auxílio às tarefas clínicas em um ambiente hospitalar do futuro baseado em tecnologias da Computação Ubíqua/Pervasiva*, vem prototipando ferramentas que visam auxiliar a gerencia de informações dentro de hospitais (FERREIRA et al., 2009). Estas ferramentas devem ser inseridas no contexto móvel para uma maior aceitação dos usuários finais e, nesse sentido, este trabalho propõe o estudo de métodos que contribuam para a criação destas interfaces e aplicativos.

Dentre os dispositivos móveis existentes atualmente, um dos mais atrativos tanto a desenvolvedores quanto a usuários é o telefone celular multi-funcional da Apple, o iPhone. Devido a suas características e funcionalidades inovadoras ele será utilizado como objeto-alvo do aplicativo implementado neste trabalho.

1.1 Objetivos

Com a crescente popularização dos dispositivos móveis e a evidente miniaturização dos mesmos, fazem-se, cada vez mais, necessários estudos para a imersão de qualquer tipo de informação atual para dentro destas máquinas. Porém, uma simples migração não é possível, pois os requisitos e paradigmas encontrados nestes aparelhos são diferentes dos tradicionalmente utilizados para armazenamento de informação, seja no fato de possuírem uma área de exibição menor ou no de alterarem seu contexto a todo momento, como, por exemplo, estando conectados a *Internet* ou não.

O objetivo geral deste trabalho é a construção de uma ferramenta que possibilite ao portador de um iPhone executar consultas diretamente do seu dispositivo em um perfil específico do serviço GoogleHealth (GOOGLE HEALTH, 2009) da Google (GOOGLE, 2009) e, com isso, manipular e atualizar informações de caráter médico.

Dentre os objetivos específicos do trabalho, tem-se (*i*) criar uma interface de visualização adequada para o dispositivo da Apple para informações referentes a saúde de pacientes e (*ii*) auxiliar o projeto ClinicSpace ao providenciar aos clínicos um método de consulta a dados de paciente de forma móvel, sem necessitar que o usuário do sistema tenha que interromper suas atividades e procurar um terminal estático para conseguir acesso a estes dados.

1.2 Estrutura do Texto

Este trabalho está estruturado da seguinte forma: o capítulo 2 revisa sobre conceitos e tecnologias úteis para o entendimento do trabalho em geral; o capítulo 3 trata do desenvolvimento da ferramenta que é o objetivo deste trabalho e o capítulo 4 realiza a avaliação dos resultados. Por fim, o capítulo 5 conclui o trabalho, realizando uma análise sobre o que foi desenvolvido.

2 REVISÃO BIBLIOGRÁFICA

2.1 O Projeto ClinicSpace

Partindo do princípio que as atividades médicas diferem das atividades realizadas em outras áreas, seja por sua possível urgência ou por lidarem com vidas humanas, a necessidade de colaboração entre profissionais, além da obtenção de uma certa mobilidade dentro de um hospital, e de outras características dinâmicas, faz com que os sistemas desenvolvidos atualmente não se incorporem ao estilo de trabalho dos profissionais. Hoje em dia, o profissional deve interromper sua atividade para utilizar o sistema, gastando parte do tempo de atendimento na utilização de recursos computacionais.

Tendo como motivação auxiliar a associação da área médica com a computação ubíqua, o projeto ClinicSpace propõe a utilização de tecnologias, que procuram tornar a computação invisível para o usuário e o auxiliam na realização de suas atividades, através da construçãos de uma ferramenta intuitiva e personalizada que auxilie o profissional de saúde na realização de suas atividades.

Por ser focado diretamente nos clínicos, e não na administração de um hospital, a idéia principal do ClinicSpace é que o profissional de saúde possa configurar a forma com que ele realiza suas atividades. Para isso foram selecionadas 11 tarefas mínimas que os próprios profissionais consultados selecionaram como sendo as mais realizadas no dia-a-dia de um ambiente hospitalar (SILVA; AUGUSTIN, 2009). Estas tarefas são listadas abaixo:

1. Revisar os problemas do Paciente.
2. Procurar informações específicas no registro do paciente.
3. Obter o resultado de novos testes e investigações.
4. Adicionar notas diárias sobre condições do paciente.

5. Requisitar análises laboratoriais.
6. Requisitar exames de vídeo/imagem (raio-x, tomografia, etc).
7. Obter resultados laboratoriais.
8. Obter resultados de exames de vídeo/imagem.
9. Requisitar tratamento.
10. Escrever prescrições médicas.
11. Registrar códigos para diagnóstico de procedimentos executados.

Estas tarefas são compostas por sub-tarefas, como, por exemplo, identificação do profissional e depois do paciente, necessárias para uma execução confiável de uma atividade e também para melhor modular o sistema. Com o conjunto de 11 tarefas, o profissional pode reorganizá-las de acordo com sua rotina, criando atividades maiores baseadas na junção de tarefas mínimas, para melhor desempenhar seu trabalho. Um exemplo dessa composição pode ser visto na figura 2.1, que também exibe as sub-tarefas presentes em cada uma das tarefas mínimas selecionadas.

O ClinicSpace está atualmente sendo construído utilizando o EXEHDA (YAMIN et al., 2005), que consiste em um *middleware* de gerenciamento do ambiente pervasivo e de suporte à execução de aplicações pervasivas.

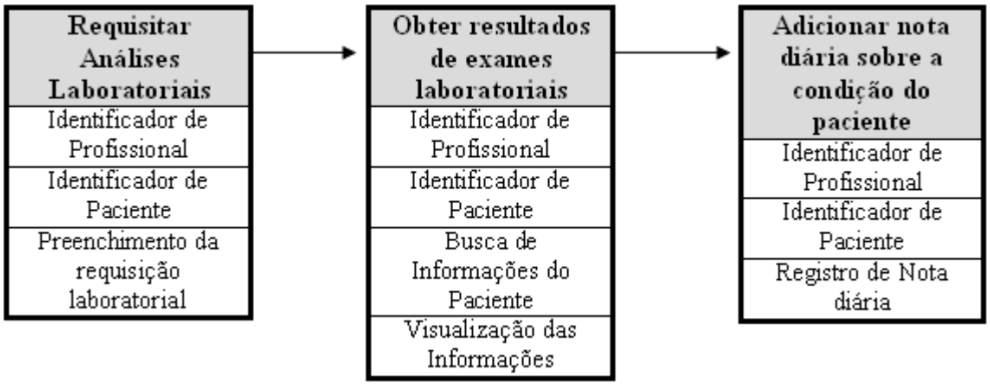

Figura 2.1: Composição de uma atividade através da junção das tarefas mínimas 5, 7 e 4.

O ClinicSpace é dividido em diversos módulos, alguns já implementados e outros em implementação. Um desses módulos refere-se à Interface do Usuário, destacado na figura 2.2. A construção desta interface deve ser planejada de acordo com os princípios definidos

nos estudos de Interação Homem-Computador (IHC), facilitando assim sua aceitação e, consequentemente, sua utilização, pelo usuário final do sistema.

Portanto, de acordo com um dos objetivos deste trabalho, o aplicativo construido poderá ser utilizado em uma futura análise de viabilidade da utilização do iPhone para acesso a dados médicos. E, com isso, facilitar o estudo na construção de interfaces de usuário adaptáveis ao dispositivo, um dos módulos presentes na arquitetura ClinicSpace.

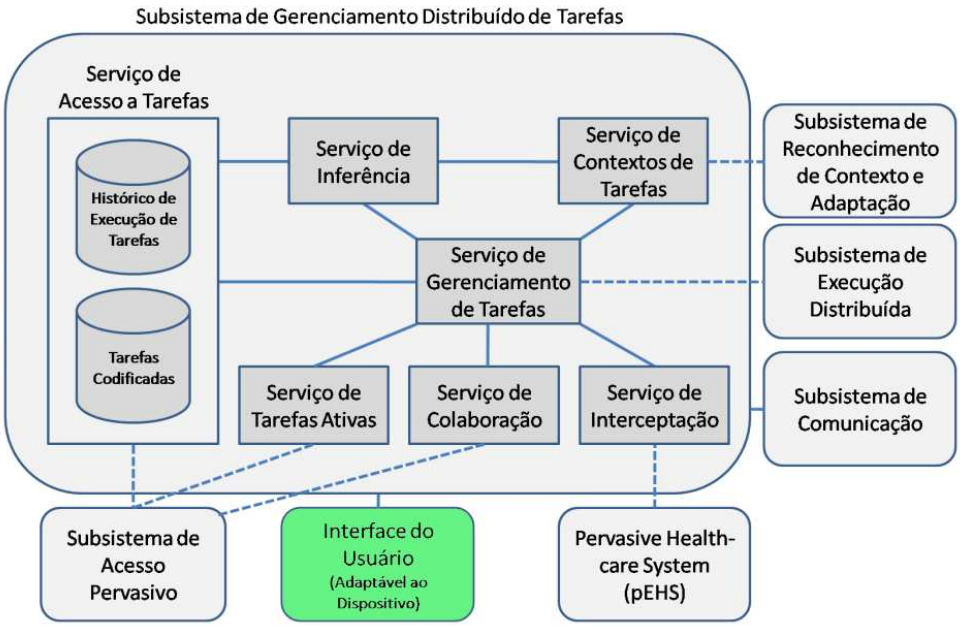

Figura 2.2: Módulos existentes na arquitetura ClinicSpace. (Silva, F.L.)

2.2 Programação para Dispositivos Móveis

Antes de iniciar a programação de dispositivos móveis é necessário ter em mente alguns conceitos como mobilidade, portabilidade, usabilidade e funcionalidade.

Para avaliar a utilidade de um dispositivo móvel, deve-se balancear suas características, tais como ser portável, funcional ou se permite fácil conectividade. Todas essas características são importantes, mas sua importância varia de usuário para usuário, pois alguns podem preferir um acessório um pouco menos portável, porém mais funcional ou vice-versa. O detalhamento destas características será abordado a seguir.

A mobilidade, dentro da computação móvel (LEE; SCHNEIDER; SCHELL, 2005), pode ser definida como a habilidade de utilizar tecnologia enquanto em movimento. Este conceito se aplica na utilização de dispositivos móveis e portáteis, como telefones celulares e PDAs.

Portabilidade consiste na capacidade de ser facilmente transportável. Em suma, um objeto pode ser quantificado quanto a sua portabilidade baseado em seu tamanho e peso. Atualmente, frente à tendência de miniaturização, é importante que exista uma preocupação com o quanto diminuir determinados objetos pode ser proveitoso, pois certas funcionalidades tornam-se difíceis de executar com dispositivos menores do que aqueles que podem ser carregados com apenas uma mão.

Quanto à usabilidade é recomendado notar que um bom dispositivo móvel deve ser utilizável por tipos de pessoas diferentes, em diversos ambientes. O tamanho e força variam de pessoa para pessoa e este fator não deve atrapalhar, bem como deve ser possível a manipulação destes dispositivos em ambiente com condições normais de funcionamento, seja no escritório, em casa, ou então em condições extremas, como excesso ou falta de luminosidade.

Por último, mas não menos importante, há o conceito de funcionalidade associado a dispositivos móveis que servem a múltiplos propósitos e têm diversos tipos de funções. Em termos de computação móvel, as funcionalidades são implementadas como aplicações móveis, e estas podem se enquadrar em duas categorias: aquelas que podem operar de modo independente, sem contato com o usuário ou outro sistema, como um relógio; aquelas que são dependentes das interações do usuário e/ou outro sistema, como um GPS.

Além destes conceitos deve-se destacar que uma das principais funções de um dispositivo móvel é conectar as pessoas, transmitir e receber informações; a conectividade com o mundo é crucial para determinadas tarefas.

Para finalizar, uma aplicação móvel não é projetada, desenvolvida e implantada fora de um contexto (LEE; SCHNEIDER; SCHELL, 2005). Geralmente, são implementadas por razões de negócio, tendo como foco facilitar a vida do profissional a fim de melhorar sua produtividade. Em geral, também precisam ser integradas às aplicações existentes.

2.2.1 Sistemas Embarcados

Dispositivos móveis, como telefones celulares ou PDAs, possuem sistemas embarcados responsáveis por gerenciar os recursos de hardware disponíveis e tratar as interações que o usuário irá efetuar.

Diferentemente de um computador tradicional, que é responsável por realizar as mais variadas funções de acordo com a vontade do usuário, um sistema embarcado é definido

como um circuito integrado dotado de capacidade computacional, porém preparado para desempenhar apenas determinadas funções. São encontrados sistemas encapsulados nos mais diversos dispositivos, como aparelhos celulares, automóveis, geladeiras, microondas, etc.. No exemplo do automóvel, um dos sistemas embutidos em questão localiza-se nos medidores de combustível, que efetuam a leitura no tanque de diesel ou gasolina através de certos sensores e retornam ao motorista a quantidade de quilômetros que podem ser percorridos até uma nova parada para reabastecimento.

Por definição, tradicionalmente, esses sistemas devem ser confiáveis, eficientes e responder a certas restrições em tempo real (GUERROUAT; RICHTER, 2005). Por confiáveis, entende-se que o sistema deve ser seguro e estar disponível sempre que necessário. A eficiência diz respeito principalmente a características como gerência de energia, peso e custo do dispositivo. Por fim, um bom sistema embarcado deve satisfazer restrições em tempo real, ou seja, deve reagir a estímulos provenientes do objeto controlado (ou do operador) no intervalo de tempo disponibilizado pelo ambiente.

2.2.2 Plataforma Java Para Dispositivos Móveis

Algum tempo atrás, devido a maioria dos dispositivos oferecerem suporte e por ser a plataforma com maior número de dispositivos no mercado, consequentemente com o maior número de usuários (NOGUEIRA; LOUREIRO; ALMEIDA, 2005), programar para dispositivos móveis era associado quase que exclusivamente a *Java 2 Micro Edition* (J2ME), portanto essa tecnologia merece uma pequena explanação neste trabalho.

A edição J2ME da plataforma Java foca diretamente no desenvolvimento de aplicações voltadas para dispositivos com memória, vídeo e poder de processamento limitados (MUCHOW, 2006), como muitos modelos de telefones celulares, PDAs e pagers. A utilização da plataforma J2ME permitiu a estes dispositivos fugir da sua natureza "estática", onde possuiam somente as funcionalidades de seus sistemas embutidos, vindas de fábrica. Após a inserção de uma Máquina Virtual Java (JVM) nestes acessórios, a dinamicidade de suas funcionalidades foi aumentada, pois o usuário poderia utilizar qualquer aplicativo que havia sido desenvolvido utilizando a plataforma J2ME.

Como os recursos dos dispositivos que utilizam a J2ME variam bastante, por exemplo o tamanho e resolução de suas telas, uma única plataforma Java não abrangeria todos os casos de desenvolvimento. Para isso, foram introduzidos os conceitos de *Configurações*

e *Perfis*.

Uma *Configuração* está intimamente vinculada a uma JVM e define os recursos da linguagem e as bibliotecas que poderão ser utilizadas. Dentro do paradigma proposto pelo J2ME existem dois tipos de configuração: Configuração de Dispositivo Conectado (CDC) e Configuração de Dispositivo Conectado Limitado (CLDC). A tabela 2.1 apresenta as diferenças entre estas duas configurações existentes. Dispositivos como telefones celulares pertencem à segunda categoria, CLDC.

Tabela 2.1: Comparativo entre Configuração de Dispositivo Conectado e Dispositivo Conectado Limitado

	Memória para Executar Java	Alocação de Memória em Tempo de Execução	Outros Detalhes
CDC	512 kilobytes (no mínimo)	256 kilobytes (no mínimo)	Conectividade de rede, largura de banda possivelmente persistente e alta
CLDC	128 kilobytes	32 kilobytes	Interface Restrita com o Usuário. Alimentação por bateria. Conectividade de rede, normalmente, dispositivos sem fio com largura de banda baixa e acesso intermitente.

Apenas uma *Configuração* não é o suficiente para a diversidade de dispositivos existente. Por isso a criação de *Perfis* se fez necessária. Um *Perfil* é a extensão de uma *Configuração*, fornecendo as diretrizes para o desenvolvimento de aplicações para um tipo específico de dispositivo. O *Perfil* tradicionalmente utilizado para o desenvolvimento de aplicações para dispositivos móveis é o *Mobile Information Device Profile* (MIDP) que define APIs para componentes, entrada e tratamento de eventos de interface com o usuário, armazenamento persistente, interligação em rede e cronômetros, levando em consideração as limitações de tela e memória dos dispositivos móveis. Na figura 2.3 é exibida a arquitetura do perfil para dispositivos de informação móvel.

2.3 Programação para iPhone

Antes mesmo de ser lançado o dispositivo móvel nomeado iPhone (APPLE IPHONE, 2009) já era considerado revolucionário e, após o seu lançamento, as expectativas vieram a se confirmar, ameaçando a aparente hegemonia da programação com J2ME. Mesclando os telefones celulares comuns com tocadores de MP3, além de oferecer suporte à rede sem fio, bússola e GPS integrado, acelerômetros embutidos, telas multi toque, entre ou-

Figura 2.3: Arquitetura do Perfil MID.

tras características, o dispositivo possui atualmente, em sua melhor configuração, um processador de 600MHz e 256MB de memória RAM e até 32GB de armazenamento, algo considerável frente às configurações da maioria dos telefones celulares existentes no mercado.

A criação de aplicações para este dispositivo mostra-se um desafio, pois além da expectativa na possibilidade de explorar todos os recursos disponíveis no mesmo e criar aplicações cada vez mais complexas, não há uma cultura muito difundida de utilização de aparelhos da Apple no Brasil, tornando assim escassos os materiais necessários para tal, pois o Kit de Desenvolvimento de Software (SDK) para o iPhone funciona somente em sistemas operacionais Mac OS X (APPLE, 2009a).

2.3.1 Linguagem Objective-C

A programação de aplicativos para o iPhone é feita utilizando-se a linguagem de programação Objective-C. Atualmente, na versão 2.0 (adicionando o recurso de coletor de lixo), esta linguagem é considerada uma fina camada acima da linguagem ANSI C (ANSI C, 2009), ou seja, incorpora ao C os conceitos de orientação a objetos e reflexão além de troca de mensagens.

O modelo de programação orientada a objetos desta linguagem é baseado na troca de mensagens entre as instâncias de objetos. Em Objective-C não são chamados métodos e sim realizadas mensagens. A diferença entre estes dois conceitos está em como o código referido pelo nome do método ou mensagem é executado. No primeiro caso, o nome do método é ligado a uma sessão do código da classe de destino pelo compilador. Já com o conceito de mensagens, o nome da mesma é apenas um nome, e o alvo deste é resolvido somente em tempo de execução, fazendo com que o objeto receptor da mensagem se pre-

ocupe em como tratá-la. Como conseqüência, o sistema de troca de mensagens não possui verificação de tipos e caso o receptor não possua alguma maneira para tratar a mensagem recebida, ele simplesmente a ignora retornando um ponteiro nulo. Uma comparação entre a sintaxe destes conceitos, escrita em C e Objective-C pode ser vista na figura 2.4.

Função em C

```
NSTimer.scheduledTimerWithTimeInterval(0.01, self, fetchVideos, null, TRUE);
```

Método em Objetive-C

```
[NSTimer scheduledTimerWithTimeInterval:0.01
                                 target: self
                               selector:@selector(fetchVideos:)
                               userInfo:nil
                                repeats:YES];
```

Figura 2.4: Exemplo da diferença de sintaxe entre C e Objective-C.

Por ser um *superset* de C, qualquer código-fonte escrito em C pode ser compilado em um compilador de Objective-C, porém o contrário não é válido. Isto deve-se ao fato de que sua sintaxe baseia-se em SmallTalk (SMALLTALK, 2009), uma linguagem criada nos anos 70 com o intuito de sustentar o "novo mundo"da computação exemplificado pela "simbiose homem-computador"(KAY, 1993). Um exemplo de programa escrito em Objective-C pode ser analisado na figura 2.5.

Outra característica da linguagem é que ela é inteiramente baseada em objetos. Todos os tipos primitivos como *Strings* e inteiros, até estruturas mais elaboradas como vetores são representados como objetos, NSString, NSNumber e NSArray, respectivamente nos casos citados. Ambos herdando suas características do tipo padrão da linguagem NSObject. São adicionados os tipos:

- id: ponteiro para um objeto qualquer. Permite a criação de objetos dinamicamente tipados.
- nil: ponteiro para objetos nulos.
- SEL: *selector*, pode ser considerado similar a um ponteiro para função.
- YES e NO: os tipos booleanos, representando verdadeiro e falso.

Suas classes são referidas como interfaces e estas são declaradas em um arquivo de extensão ".h"abaixo do *token* @interface. Uma classe em Objective-C possui métodos de

```
#import "Button_FunViewController.h"

@implementation Button_FunViewController
@synthesize statusText;

- (IBAction)buttonPressed:(id)sender
{
    NSString *title = [sender titleForState:UIControlStateNormal];
    NSString *newText = [[NSString alloc] initWithFormat:
        @"%@ button pressed.", title];
    statusText.text = newText;
    [newText release];
}
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)➥
interfaceOrientation {
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}


- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning]; // Releases the view if it doesn't➥
have a superview
    // Release anything that's not essential, such as cached data
}
```

Figura 2.5: Parte de código-fonte escrito em Objective-C.

classe (iniciados com um sinal de "+"durante sua declaração na interface) equivalentes a métodos estáticos em C++ e métodos de instância (iniciados com o sinal de -").

A função da interface é informar o protótipo do que será criado. Os métodos são verdadeiramente implementados em arquivos com a extensão ".m"logo após o *token* @implementation. Uma das vantagens presentes em Objective-C é o conceito de @synthesize que ao ser utilizado durante a implementação, localizado antes a uma variável da classe, automaticamente cria os métodos de requisição e alteração de valor desta variável.

2.3.2 iPhone SDK

O SDK do iPhone disponibiliza ferramentas e recursos necessários para a criação de aplicações para o iPhone que aparecem como ícones na tela principal do usuário. Essas aplicações executam de forma autônoma e possuem acesso a todas as funcionalidades que tornam o iPhone interessante, como os acelerômetros, serviço de localização e interface multi toque. Aplicações também podem salvar dados no sistema de arquivos local e até

se comunicar com outras aplicações instaladas no dispositivo.

Para a criação de aplicativos, deve-se utilizar a API Cocoa Touch que consiste de uma camada de abstração do sistema operacional presente em um iPhone. Esta é dividida em duas partes *UIKit* e *Foundation*, cuja arquitetura pode ser visualizada na figura 2.6.

Figura 2.6: Arquitetura da API Cocoa Touch.

Todo aplicativo desenvolvido para esta arquitetura é construído utilizando-se o *UIKit Framework* e, portanto, possui a mesma arquitetura núcleo. Este *framework* fornece objetos necessários para se executar o aplicativo e coordenar as eventuais interrupções das entradas do usuário e a exibição de conteúdo na tela.

Um aplicativo em execução recebe interrupções continuamente do sistema e precisa responder a estas interrupções. Receber estas interrupções é papel do objeto *UIApplication*, porém respondê-las é de responsabilidade do código a ser desenvolvido. Para uma melhor compreensão de onde se deve responder a certas interrupções, é necessário entender um pouco sobre o ciclo de vida de um aplicativo para iPhone. A figura 2.7 ilustra o ciclo de vida simplificado de um aplicativo. Este diagrama mostra a seqüência de eventos que ocorrem a partir do momento que o aplicativo inicia até o seu término. Na inicialização e término, o *framework UIKit* envia mensagens específicas para a o objeto *Delegation* do aplicativo, responsável por gerenciar o funcionamento geral do mesmo, por informá-lo sobre qualquer acontecimento ocorrido durante o ciclo de vida do aplicativo. Durante o ciclo de eventos, o *framework UIKit* despacha as interrupções para o manipulador das mesmas, presente no aplicativo criado.

As ferramentas que compõem o SDK do iPhone são o Xcode, onde todos os arqui-

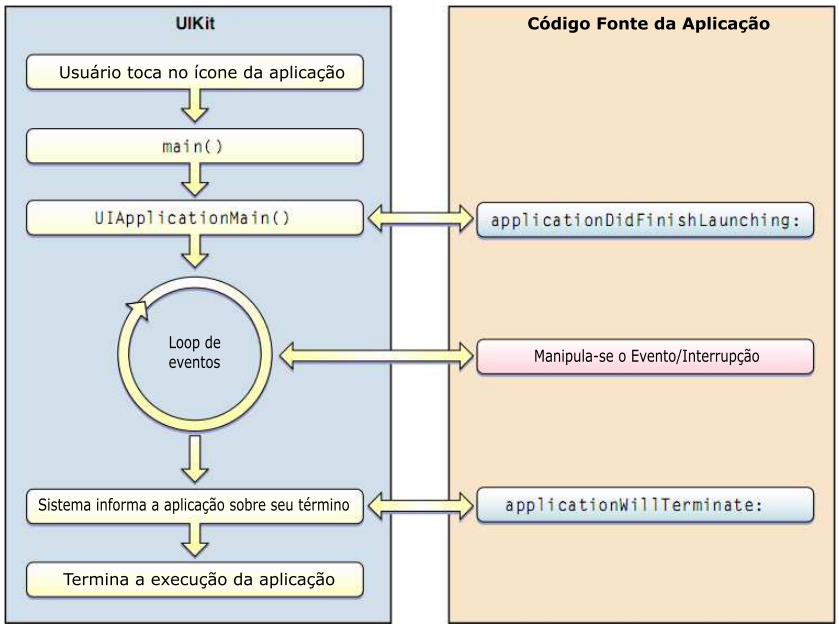

Figura 2.7: Ciclo de vida de uma aplicação para iPhone. (Apple)

vos necessários a criação de um aplicativo são manipulados, e também onde o código-fonte é escrito, sua interface é visualizada na figura 2.8. Na figura 2.9 tem-se também a ferramenta *Interface Builder* responsável pela manipulação dos arquivos com extensão ".nib"onde são definidas as interfaces visuais a serem manipuladas pelo usuário. E, por fim, ao término da criação de um novo programa é necessário testá-lo, e para isso a última aplicação presente no SDK é o *iPhone Simulator*, aplicação que emula um iPhone dentro do computador do desenvolvedor, sua interface pode ser vista na figura 2.10.

2.4 Google Health

Tendo como missão "organizar as informações do mundo todo e torná-las acessíveis e úteis em caráter universal", a empresa Google destaca-se ,cada vez mais, no cenário mundial através dos diversos serviços que disponibiliza ao público, como as já tradicionais ferramentas de busca, email (GMAIL, 2009), documentos (GOOGLE DOCS, 2009), entre outras.

Lançado em janeiro de 2008, o Google Health (GOOGLE HEALTH, 2009) foi desenvolvido para se juntar aos outros serviços já oferecidos; porém seu foco está na gerência e armazenamento de informações médicas. Possibilita que o usuário compartilhe suas informações com seus médicos, facilitando assim o acesso do mesmo a informações antigas e, consequentemente, na realização de um diagnóstico mais rápido e preciso.

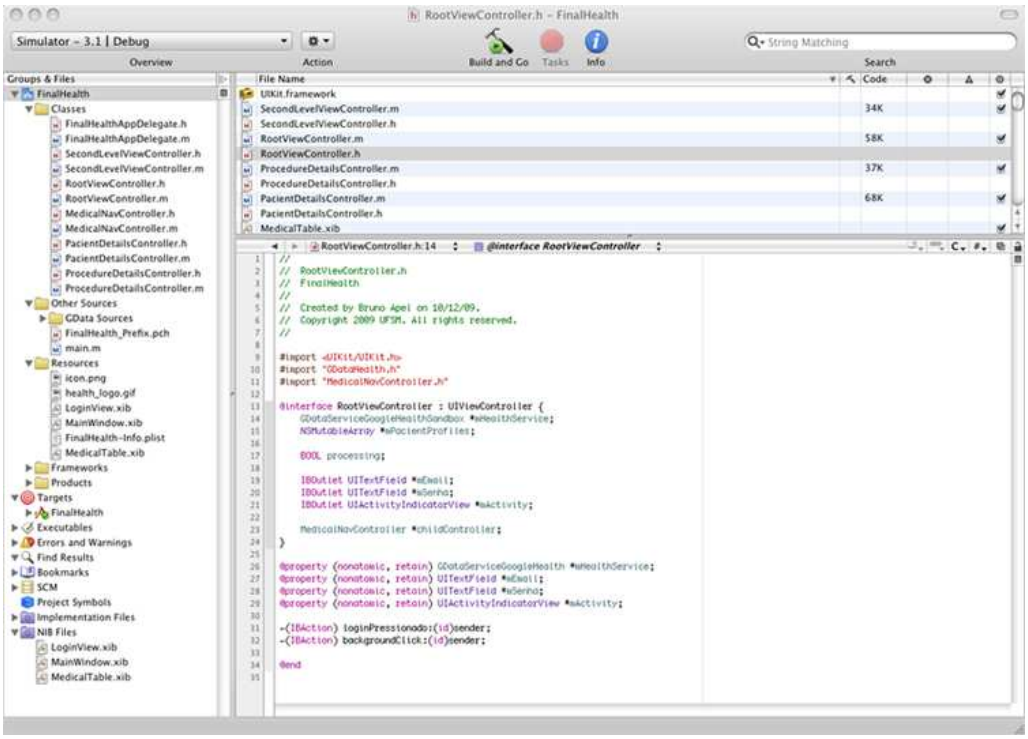

Figura 2.8: Janela padrão do xCode.

Figura 2.9: Janelas do Interface Builder.

Figura 2.10: Janelas do iPhone Simulator.

Nesta ferramenta da Google é possível armazenar dados como histórico médico completo e dados clínicos gerais, como alergias, medicamentos em uso, cirurgias já realizadas, resultados de exames, vacinas, etc.. Para auxiliar o usuário, já se encontram cadastrados no sistema diversas informações referentes às citadas acima, como por exemplo, ao inserir a condição de diabetes a um perfil, o sistema retorna uma breve descrição do que é a diabetes, seus sintomas, tratamento, causas, diagnóstico, prevenção, complicações e quando contatar um médico, como pode ser visto na figura 2.11.

Atualmente, o serviço firma parcerias com diversos hospitais, laboratórios e farmácias possibilitando ao usuário importar seus dados diretamente do sistema onde se encontram, porém este serviço ainda não está disponível ao Brasil. O sistema também verifica como é a interação entre medicamentos em uso, alergias cadastradas e patologias existentes, caso exista alguma contra indicação, a mesma é exibida em detalhes. Por fim, permite ao usuário o *upload* de arquivos com imagens digitais, facilitando a inserção do resultado de exames como raios-x ou qualquer tipo de exame gráfico.

Sua interface divide-se em três colunas, como é destacado na figura 2.12, demonstrando a tela inicial, após a autenticação do usuário. A coluna localizada a esquerda da tela, definida na figura 2.12 como coluna 1, possui os *links* necessários para navegar entre as funcionalidades do sistema, a coluna central, ou coluna 2, exibe as informações referenciadas pelo *link* atualmente selecionado e por fim a coluna da direita, ou coluna 3, mantêm um sumário de todas as informações cadastradas ao usuário selecionado. Grande parte das telas foram concebidas a partir de um padrão de armazenamento de dados mé-

dicos chamado de *Continuity of Care Record* (CCR), o qual será detalhado na próxima seção.

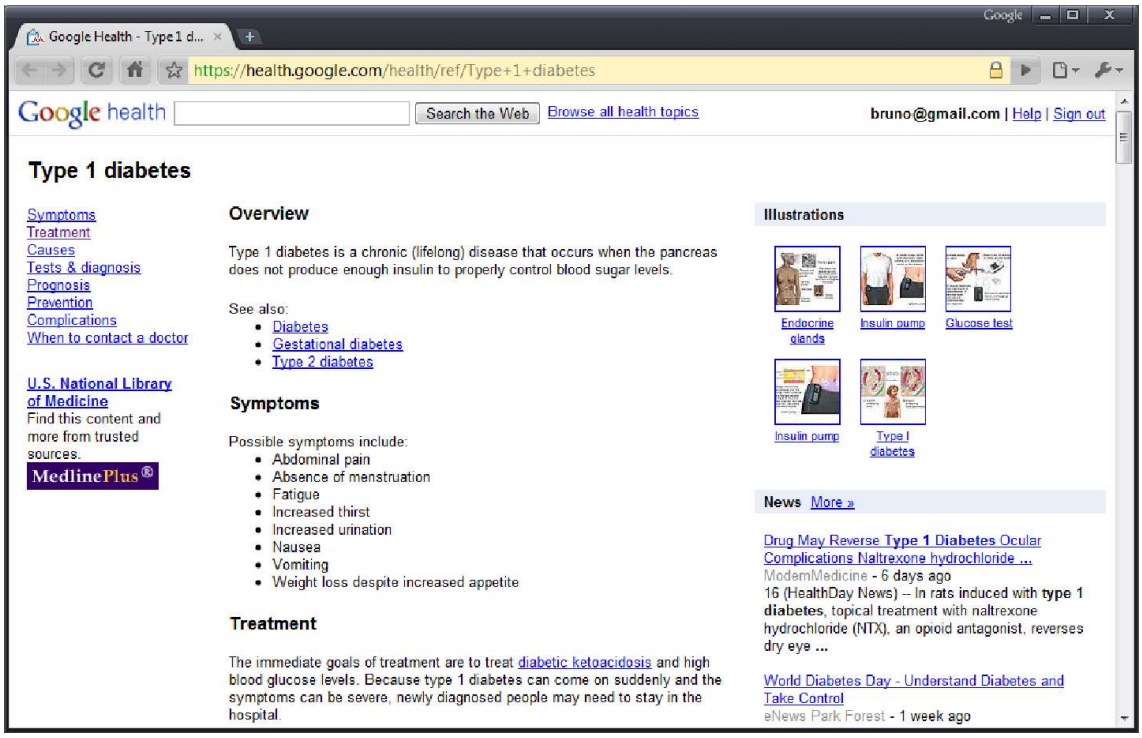

Figura 2.11: Interface do Google Health responsável pela descrição de patologias.

2.4.1 Padrão CCR

A Continuidade do Registro de Atendimento (ASTM, 2003), do inglês *Continuity of Care Record* (CCR), é um conjunto dos mais relevantes dados administrativos, demográficos e informações clínicas sobre a saúde do paciente, cobrindo um ou mais encontros médicos.

Criado em 2003 por um grupo de médicos, enfermeiras, tecnólogos e leigos com um interesse em melhorar o sistema de saúde, este padrão fornece a um profissional da saúde ou sistema um meio de agregar todos os dados pertinentes a um paciente e transmiti-los para qualquer outro profissional da saúde ou sistema para garantir a continuidade do tratamento.

O padrão define um formato XML que oferece uma interface comum de comunicação entre os sistemas. Ele possui três componentes, que são:

- *Header*: identifica o formato do registro e o tempo em que ele foi criado. Possui os campos <CCRDocumentObjectID>, <Language>, <Version>, <DateTime>

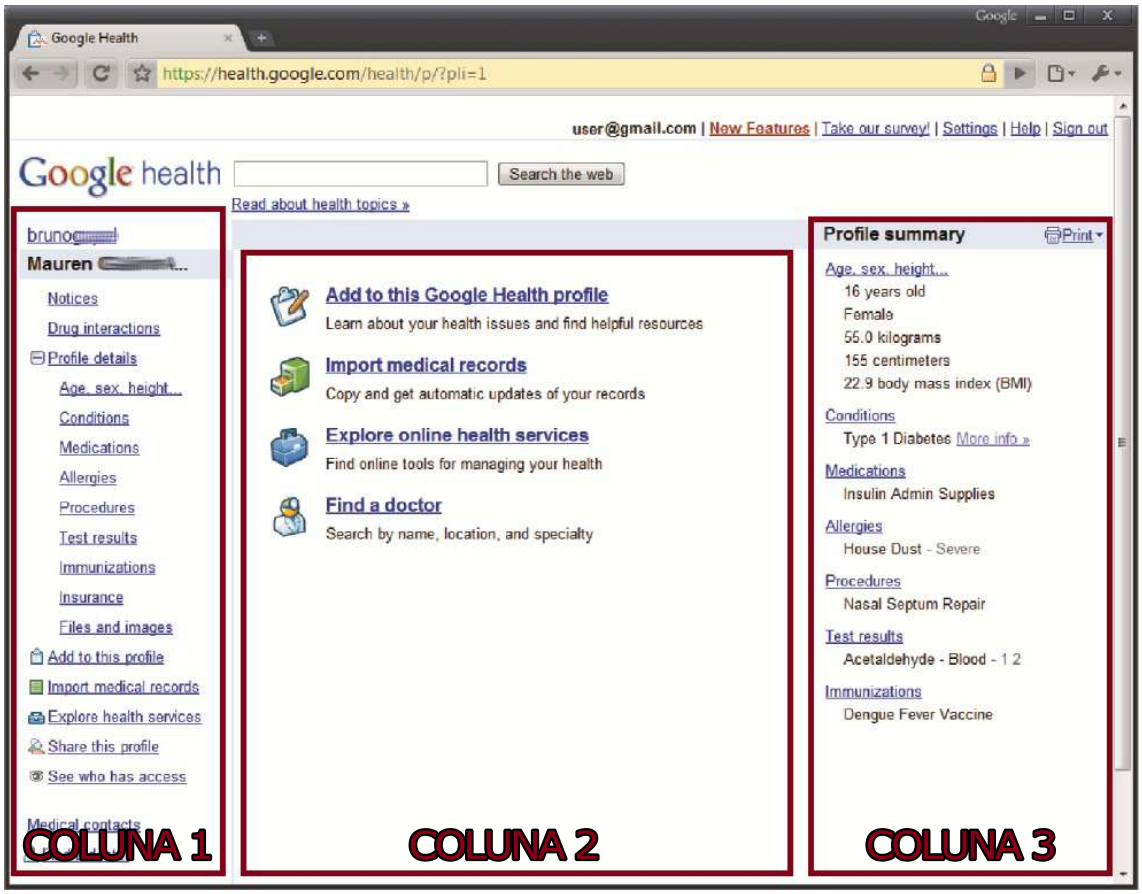

Figura 2.12: Interface inicial do Google Health.

e <Patient>.

- *Body*: contem os detalhes do histórico médico. É composto por <FunctionalStatus>, <Problems>, <SocialHistory>, <Alerts>, <Medications>, <Immunizations>, <VitalSigns>, <Results> e <Procedures>.

- *Footer*: consiste das informações demográficas sobre o paciente, como nome, gênero e data de nascimento. É representado por um único campo <Actor> que possui subcampos para informar os dados demográficos citados acima.

Mais detalhes sobre o formato podem ser observados na figura 2.13.

2.4.2 API Google Health

A Google disponibiliza uma API para desenvolvimento de possíveis aplicações clientes, interessadas em acessar e modificar o conteúdo encontrado no serviço Google Health. Estas APIs de dados oferecem um protocolo simples e padrão para ler e gravar dados na web.

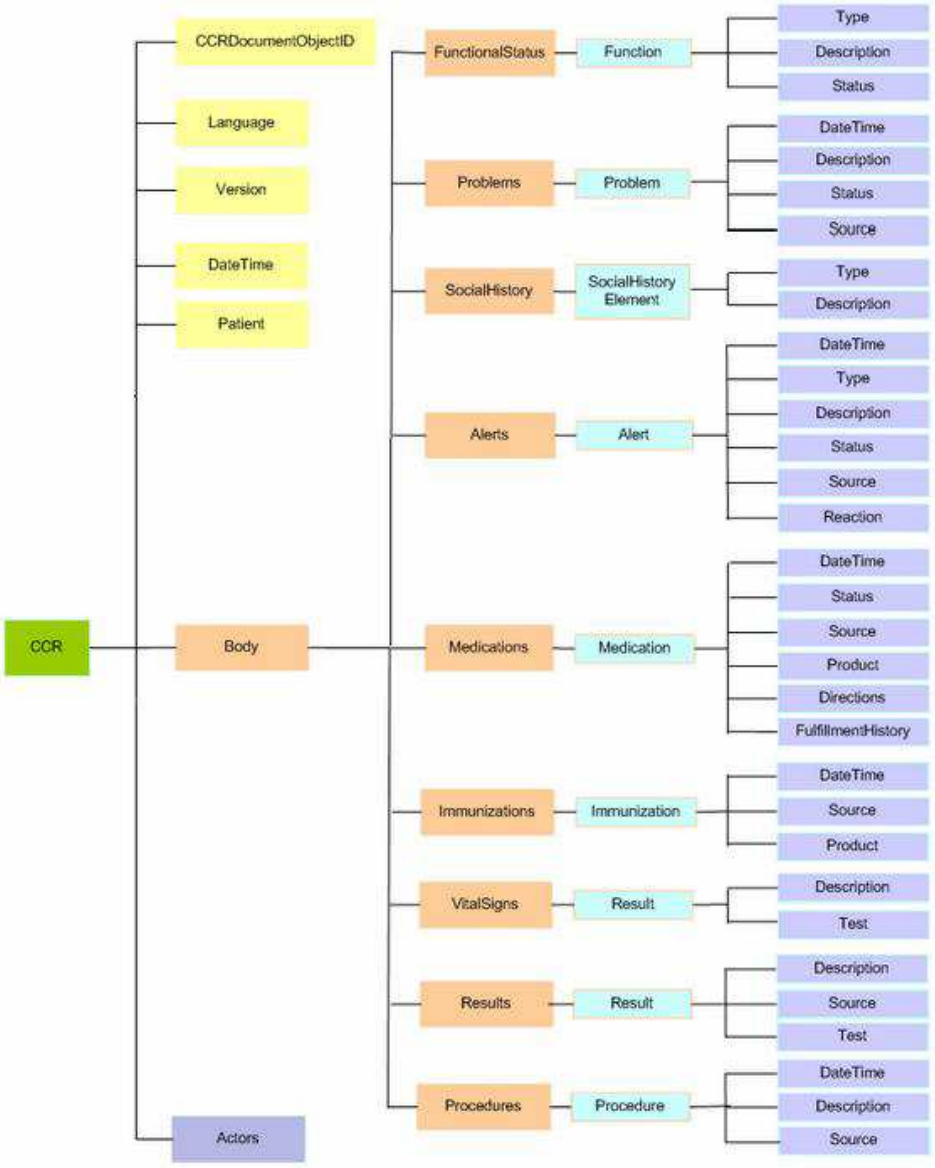

Figura 2.13: Detalhamento dos componentes do padrão CCR. (Google Health)

Ao contrário das outras APIs do Google, por manipular informações de caráter médico, a API do Google Health obriga que o desenvolvimento seja realizado em uma *sandbox*, ou seja, um ambiente de testes, chamado H9 (GOOGLE HEALTH - H9, 2009). Esta *sandbox* não possui tantas restrições para a realização de qualquer operação quanto a versão oficial. A diferença entre as duas plataformas podem ser visualizadas na tabela 2.2. As aplicações desenvolvidas não poderão interagir com a plataforma real do Google Health antes de serem testadas dentro do H9.

Tanto no H9 quanto diretamente no Google Health, o acesso a dados requer autorização. Existem dois modos de autenticação:

Tabela 2.2: Comparativo entre os recursos presentes no serviço GoogleHealth e a Sandbox H9

	H9 Sandbox https://www.google.com/h9	Google Health https://www.google.com/health
As requisições da API precisam ser assinadas digitalmente?	Não	Sim
Onde deve ser requisitado um token AuthSub?	https://www.google.com /h9/authsub	https://www.google.com /health/authsub
Posso utilizar http://localhost como o parametro **next** do AuthStub?	Sim	Não
Qual o valor do parametro **scope** do AuthStub?	https://www.google.com /h9/feeds/	https://www.google.com /health/feeds/
Qual o nome do **service** para o login do cliente?	weaver	health

- AuthStub: permite aos usuários conectar suas contas do Google Health entre sites externos e o Google sem a necessidade de compartilhadar o *login* e a senha do serviço com o site externo. Ao invés disto, um token é utilizado como credencial de autenticação com o Google Health;

- OAuth: é uma alternativa de padrão aberto para autenticação e troca de dados ao AuthStub.

A troca de informações entre o aplicativo em desenvolvimento e o servidor de dados do Google é realizada através de requisições HTTP. Existem quatro tipos de requisição disponíveis para comunicação, (*i*) uma consulta simples com busca por determinado tipo de informação, (*ii*) uma criação de um novo registro e (*iii*) a edição ou (*iv*) remoção de um registro já existente.

2.4.2.1 Biblioteca para Objective-C

Para facilitar o acesso a dados, o Google oferece bibliotecas para clientes em diversas linguagens de programação (entre elas Objective-C), e também códigos-fonte contendo exemplos de aplicações desenvolvidas. A biblioteca para Objective-C possibilita ao usuário de Mac ou iPhone armazenar, acessar ou compartilhar as suas informações presentes nos servidores do Google através da sintaxe própria da linguagem, como no exemplo

abaixo:

```
[calendar fetchThisFeed:url callWhenDone:myMethod];
```

Para a utilização desta biblioteca é necessário conhecer a terminologia utilizada, que consiste de:

- *Element*: qualquer item de dados, tais como um tempo de início, o nome de uma foto, uma fórmula;
- *Entry*: um registro feito de *Elements*. Por exemplo, um compromisso no calendário, uma foto, uma célula de uma planilha;
- *Feed*: uma lista de *Entrys*, tais como uma lista de compromissos, um álbum de fotos, uma planilha;
- *Service*: um objeto que se comunica com o servidor, como um Calendário, Fotos, Planilhas;
- *Ticket*: uma transação realizada em um serviço. Por exemplo, o agendamento de um compromisso, a submissão de uma foto.

3 ESTUDO DE CASO - *SOFTWARE* PARA ACESSO MÓVEL AO GOOGLE HEALTH, UTILIZANDO O IPHONE

Neste capítulo é detalhado o processo de desenvolvimento do aplicativo para acesso às informações contidas no sistema Google Health, utilizando o dispositivo móvel iPhone (APPLE, 2009a). O aplicativo desenvolvido provê uma ferramenta para que o usuário (um médico), tenha acesso às informações relacionadas à saúde de seus pacientes, bem como possa adicionar novas informações, referente a medicamentos e patologias.

Primeiramente, é descrito o objetivo do aplicativo desenvolvido além de seus requisitos. A seguir, é detalhado o projeto do mesmo e, para concluir o capítulo, é descrita a implementação em si, além de uma exibição das interfaces finais obtidas.

3.1 Objetivos e Requisitos do apicativo

Um programa sem um objetivo definido equivale a um programa que não será utilizado. Todo aplicativo a ser desenvolvido precisa cumprir com, no mínimo, um objetivo e possuir alguns requisitos previamente definidos. O caso em estudo não poderia ser diferente; portanto, possui como objetivo geral prover acesso móvel às informações de caráter médico hospedadas no serviço Google Health. A escolha do Google Health se justifica por possibilitar a manipulação destas informações através da "visão do usuário/paciente"fugindo das tradicionais implementações de programas focados na área médica, onde a maioria possui a "visão do gerenciamento do hospital".

Para cumprir com este objetivo geral, alguns objetivos mais específicos foram criados, e estes consistem de:

1. Permitir ao usuário entrar no sistema utilizando sua própria conta do Google;

2. Prover ao usuário uma lista de seus pacientes;

3. Possibilitar a visualização dos dados definidos no padrão CCR para cada um dos seus pacientes;

4. Possibilitar a inserção de novas informações no sistema.

Tendo estes objetivos definidos, os seguintes requisitos precisam ser observados, que são:

- O usuário deverá ter a possibilidade de acessar o sistema de qualquer lugar através de seu dispositivo móvel;
- O usuário deverá ter acesso à internet através do seu dispositivo móvel;
- O usuário deverá possuir uma conta do Google para acessar os dados presentes no aplicativo.
- As informações inseridas via dispositivo ou através de um computador qualquer deverão estar sincronizadas entre si.

Após a definição do objetivo principal e dos objetivos mais específicos, bem como dos requisitos a serem implementados, partiu-se para a criação do projeto do aplicativo.

3.2 Projeto e Modelagem de Dados

Para o projeto deste aplicativo, convêm citar que um programa semelhante já foi criado pelos integrantes do GMob, porém utilizando uma tecnologia diferente, a anteriormente citada J2ME. A diferença entre este programa e o aplicativo desenvolvido neste trabalho também é explicitada na localização das informações, enquanto um utiliza um repositório local, o outro se conecta aos servidores do Google para adquirir os dados a serem exibidos. As demais características dos programas são similares, pois ambos baseiam-se na utilização do padrão CCR para manipulação de informações médicas.

A figura 3.1 exibe interfaces deste programa, como por exemplo, a tela de *login*, a lista de pacientes e a interface principal, com ícones para cada tipo de informação encontrada. Este programa serviu para uma análise anterior ao desenvolvimento do aplicativo deste trabalho.

O desenvolvimento de aplicativos para iPhone baseia-se no padrão de arquitetura de *software Model-View-Controller* (MVC), que especifica que, com o aumento da complexidade dos programas criados, torna-se fundamental a separação entre os dados (*Model*)

Figura 3.1: Telas extraidas de um programa J2ME com o mesmo objetivo do desenvolvido neste trabalho.

e o leiaute (*View*). A adoção desse padrão traz ao programador uma maior segurança, seja na manipulação dos dados, garantindo que isto não afetará o leiaute, ou seja na eventual reorganização do leiaute, garantindo que não resulte em um efeito colateral aos dados.

Para realizar a ligação entre dados e leiaute é criada a figura do controlador (*Controller*), que processa e responde a ações do usuário, através das interfaces e também pode realizar modificações diretas nos dados. Esta relação é exibida na figura 3.2 onde é apresentada a figura do controlador, baseada em um objeto modelo, e somente ela troca

informações com o que é exibido na tela do aplicativo.

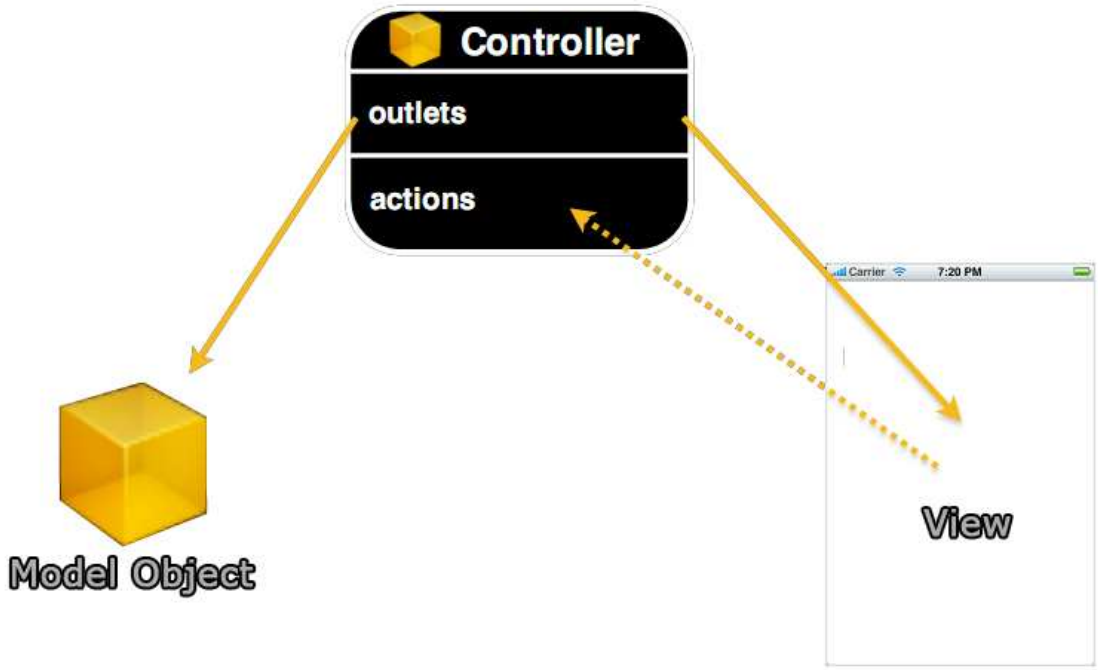

Figura 3.2: Relação entre Model-View-Controller. (Apple Iphone App Programming Guide)

No caso específico do iPhone, a utilização de MVC representa, na grande maioria dos casos, que cada tela de exibição existente possuirá uma classe controladora responsável pela lógica da mesma, bem como por indicar todos os dados a serem exibidos.

Devido à variada gama de elementos de interface já inclusos no SDK, muitos destes também trazem consigo um controlador "cru", ou seja, disponibiliza-se o esqueleto de uma classe controladora para o tipo de elemento escolhido. Tendo acesso a essa classe, necessita-se acrescentar o código para tratamento de interrupções, inserção de dados, entre outras funcionalidades, nos seus respectivos locais.

Dentre os elementos com classes controladoras disponíveis convêm citar:

- *View Controller*: controlador responsável por gerenciar uma tela genérica, com qualquer tipo de elemento inserido;
- *Navigation Controller*: controlador que gerencia a navegação hierárquica entre diversas telas;
- *Tab Bar Controller*: controlador que gerencia um conjunto de telas que representam

itens/ícones numa barra guia, normalmente, localizada na parte inferior do aplicativo;

- *Table View Controller*: controlador que administra uma tela contendo uma exibição em forma de tabela;
- *Image Picker Controller*: controlador que administra uma tela para escolha e navegação entre imagens.

O aplicativo desenvolvido baseia-se em uma sequência de passos definida e exibida na figura 3.3.

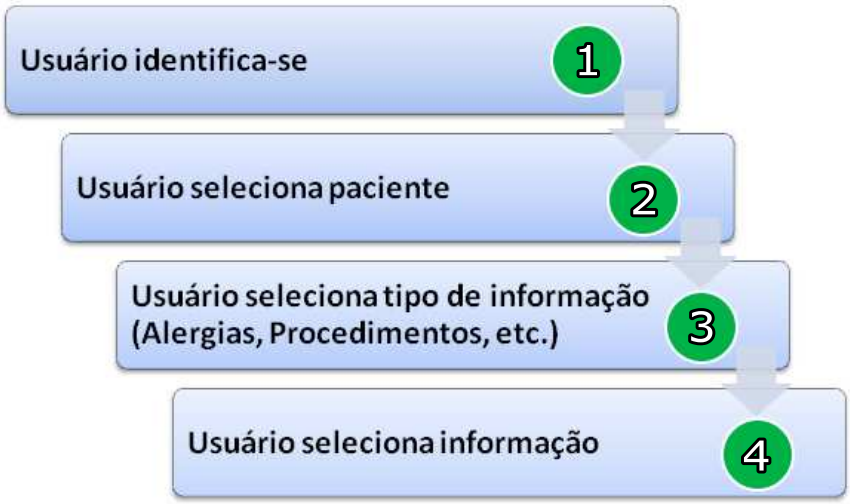

Figura 3.3: Sequência de passos do algoritmo definido para a utilização do aplicativo.

Primeiramente, o usuário deverá realizar o *login* no sistema (1). Com esta tarefa cumprida, lhe será exibido uma lista com seus pacientes. Ao selecionar um paciente (2), uma nova tela será exibida, com os tipos de informações disponíveis no sistema (Avisos, Dados Demográficos, Alergias, Condições ou Patologias, Imunizações, Medicamentos, Procedimentos e Resultados de Testes). Ao clicar em qualquer um destes termos (3), o sistema retornará para o usuário todos os dados do paciente em questão referentes ao termo escolhido. Clicando-se em um dado em exposição (4), um alerta será exibido com o detalhamento do que foi selecionado.

Na tela de exibição dos dados referentes a um dos tipos definidos no padrão CCR, e citados no parágrafo anterior, existirá um ícone com o símbolo de adição (+) que, caso pressionado, permitirá ao usuário inserir uma nova informação do mesmo tipo de dado em exibição.

Frente a estas especificações, os controladores escolhidos para a implementação do aplicativo foram um *Navigation Controller*, para gerenciar a hierarquia entre as telas de exibição, um *View Controller*, para administrar as informações da tela inicial de *login*, e por fim, alguns *Table View Controllers*, um para a tela de listagem de pacientes, outro para a listagem dos tipos de informações presentes e, por último, outro controlador para a listagem dos dados cadastrados do paciente, para cada tipo de informação.

Tanto o *View Controller* quanto os *Table View Controllers* são submissos às ações tomadas pelo *Navigation Controller*, estando neste último toda a lógica de controle de qual tela será exibida em determinado momento. As responsabilidades do *View Controller* e dos *Table View Controllers* consistem em gerenciar as informações presentes nas telas associadas a cada um deles. A estrutura de um *Navigation Controller* pode ser observada na figura 3.4 onde se percebe que este é composto de um vetor de controladores (1), bem como ponteiros para o primeiro controlador do vetor (2) e para o controlador em exibição (3).

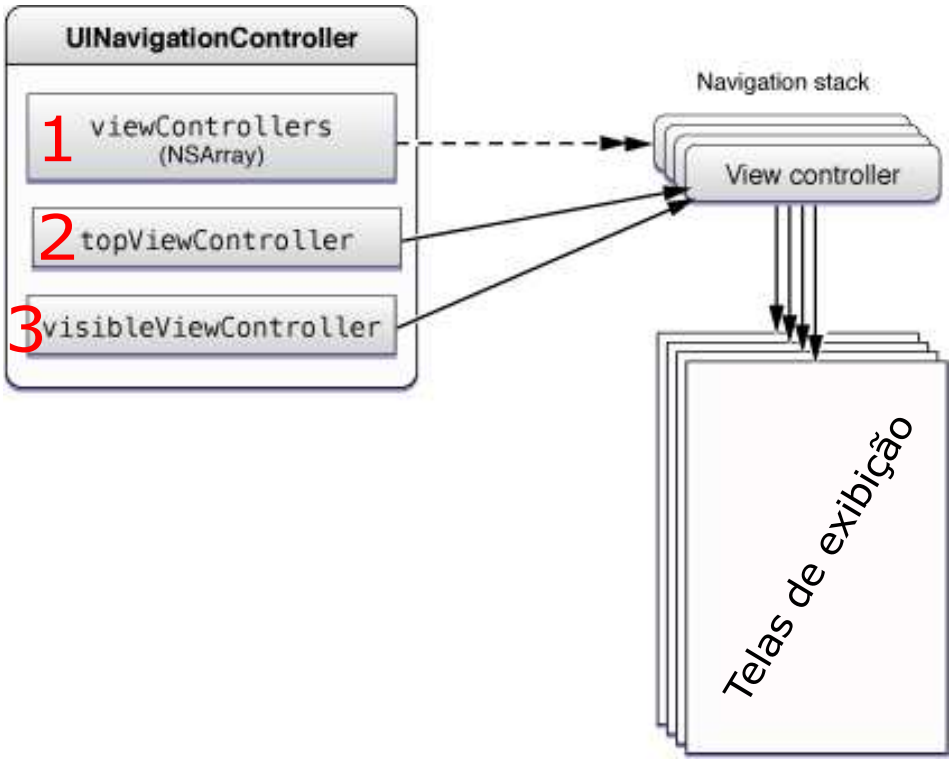

Figura 3.4: Estrutura de um Navigation Controller. (Apple Iphone App Programming Guide)

As classes necessárias para a implementação do aplicativo desenvolvido nesse trabalho, bem como sua estrutura, são descritas abaixo:

- *Pacient*: armazena os dados referentes ao paciente. Esta possui um identificador (ID) obtido diretamente dos servidores do Google, bem como uma variável para registrar o nome do paciente e uma estrutura para conter todos os dados no formato CCR referentes ao enfermo.

- *RootViewController*: deriva de um *Navigation Controller*, portanto, ela possuirá os métodos responsáveis pela troca de telas do aplicativo. Para isso é utilizado uma estrutura no formato de pilha de *Controllers*, onde os controladores que gerenciam as telas do aplicativo serão armazenadas ou retirados através dos comandos *push* e *pop*, respectivamente.

- *LoginController*: herda as propriedades de um *View Controller*, portanto, não possui nenhuma característica específica já definida. Ela controla uma tela onde são vistos dois campos de texto, que são "Email"e "Senha"e um botão para *Login*. Quando os campos de textos estão selecionados, um teclado auxiliar é exibido para que estes possam ser preenchidos. Cada campo possui um leiaute de teclado característico a sua finalidade, onde o teclado referente ao campo "Email"apresenta um destaque aos caracteres "@"e ".", presentes em todos os endereços de email. Já o leiaute do teclado utilizado para o preenchimento do campo "Senha"é o leiaute padrão; porém, durante o preenchimento deste campo os caracteres da senha do usuário não são exibidos, e sim substituídos por um caractere genérico.

- *PacientsController*: herda suas características de um *Table View Controller*, esta é responsável por exibir a lista de pacientes cadastrados no sistema em formato de uma tabela do iPhone. Também é papel desta classe instanciar cada um dos "Pacient"necessários e carregar suas informações referentes à ID e nome localmente.

- *InformationController*: carrega do servidor remoto todos os dados no formato CCR do paciente selecionado. Cria uma tela com os tipos de dados existentes e aguarda uma interação do usuário. Deriva de um *Table View Controller*.

- *SpecificInformationController*: exibe os dados detalhados do tipo selecionado. Ou seja, é esta classe que listará as alergias do paciente, seus medicamentos, ou qualquer um dos tipos previstos pelo sistema, além de retornar ao usuário os dados mais detalhados, caso algum clique em determinada informação seja realizado. Como a

classe "InformationController"também deriva de um *Table View Controller*.

O termo a ser utilizado para efetuar o *login* no sistema do Google Health deve ser o email completo do usuário. Deviso a este fato, a classe "LoginController"possui uma funcionalidade extra implementada, permitindo que seja utilizado somente o prefixo do email, omitindo qualquer coisa após o arroba, ficando a cargo do sistema acrescentar estes dados antes de uma possível consulta aos servidores do Google.

3.3 Implementação

Com as classes e suas tarefas descritas, partiu-se para a implementação de fato. Inicialmente, foi planejada uma única classe responsável por realizar os contatos com os servidores do Google. Porém, em tempo de codificação isso se mostrou inviável, pois ao se iniciar uma nova consulta a alguma informação, uma nova *thread* era disparada.

A manipulação dos resultados era feita em um método apontado por um seletor indicado durante a execução da consulta. Consequentemente, a classe controladora, ao solicitar as informações a esta classe responsável pela comunicação, não obtinha uma resposta de forma síncrona e seguia sua atualização da tela de exibição ainda sem receber os dados necessários. Para resolver este problema, a classe responsável pela comunicação foi desconsiderada e cada comunicação em específico é codificada e realizada na própria classe controladora responsável pela gerência dos dados da tela em exibição.

Neste ponto, definiu-se que o contato com o repositório de dados, necessitando da conexão com a internet, seria realizado somente em três ocasiões (*i*) no momento em que o usuário tenta realizar o *login* no sistema; (*ii*) ao clicar no nome de algum paciente e (*iii*) ao inserir novos dados no repositório de dados.

No primeiro caso, o contato com o servidor realiza-se para a verificação do *login* e também para obtenção da lista de pacientes cadastrados. Na segunda ocasião, ao clicar no nome de um determinado paciente, o servidor é contatado para retornar todos os dados relativos ao paciente em questão. Caso esta operação seja feita mais de uma vez para o mesmo paciente e presumindo-se que os dados do paciente não foram alterados, o servidor não retornará novamente todos os dados do solicitados, mas uma mensagem informando que os dados antes adquiridos ainda podem continuar sendo utilizados, pois não houve nenhuma modificação nas informações do paciente no sistema. Isto evita um tráfego de banda desnecessário.

No último caso, ao se adicionar novas informações, o sistema deve ser atualizado para corresponder as mudanças incorporadas, portanto, o servidor é novamente contatado.

A comunicação realiza-se através da utilização de um tipo de dado responsável pela abstração de um serviço oferecido pelo Google, nomeado *GDataServiceGoogleHealth* para acesso a base de dados oficial do Google Health ou *GDataServiceGoogleHealth-Sandbox* para acesso ao H9. Todas as classes controladoras possuem um ponteiro para este dado, podendo assim realizar suas consultas.

Uma consulta ao servidor é nomeada *Ticket* e todas são feitas utilizando parâmetros do tipo *Feed*. Este *feed* nada mais é do que a URL representativa ao que se espera obter do servidor, como por exemplo *https://www.google.com/health/feeds/profile/list*. Esta *feed* em específico retorna uma lista de nomes, no caso pacientes, associados ao usuário, desde que o cliente já esteja autenticado no sistema.

Qualquer erro encontrado entre as consultas ao serviço do Google é exibido ao usuário através de um alerta, sobrepondo-se ao que é exibido na tela e solicitando uma tomada de decisão do usuário quanto ao problema.

Alguns imprevistos foram identificados durante a implementação. Entre eles, a biblioteca responsável pela manipulação de XML, o formato utilizado pelo padrão CCR, não estava presente no SDK do iPhone e sim, somente no SDK para MacOS. Para contornar este problema foi utilizada uma biblioteca equivalente, disponibilizada pelo próprio Google.

Após a finalização da codificação, obteve-se diversas telas como resultado do aplicativo criado e estas serão exibidas e detalhadas nesta sessão.

A figura 3.5 mostra a a tela com os aplicativos instalados em um iPhone, com ênfase no ícone do aplicativo desenvolvido neste trabalho. A tela de *login* do aplicativo é analisada na figura 3.6, contendo sua configuração inicial além de outra tela com a exibição do teclado para inserção dos dados. Feita a autenticação do usuário, obtem-se uma das telas exibidas na figura 3.7, cobrindo as possibilidades de "Erro de *Login*"e "*Login* Efetuado com Sucesso"exibidas.

Após a tela de *login*, todas as outras apresentam um botão na barra superior possibilitando a volta para a tela anterior. Estando o usuário autenticado, uma lista de seus pacientes é exibida. Clicando-se no nome de qualquer paciente, uma nova tela é carregada contendo os tipos de dados presentes no formato CCR. Tanto a tela com a lista de

Figura 3.5: Tela de um iPhone com o aplicativo desenvolvido instalado. Com ênfase no ícone do aplicativo.

pacientes como aquela contendo as informações do formato CCR podem ser visualizadas na figura 3.8. Como exemplo de uma informação específica exibida, tem-se as telas referentes a alergias e iminuzações exibidas na figura 3.9.

Prevendo-se que algumas informações seriam demasiadamente extensas para uma total exibição na tela, em conjunto com a possibilidade de detecção de rotação do iPhone, indicando qual aresta do dispositivo está direcionada para cima, criou-se um método para que a disposição da informação acompanhe a diposição do próprio iPhone. Assim, informações extensas são melhor visualizadas com o dispositivo horizontalmente posicionado, como visto na figura 3.10, com o resultado de testes associados ao paciente selecionado.

Figura 3.6: Telas de login do aplicativo desenvolvido. Com e sem a exibição do teclado auxiliar.

Figura 3.7: Telas exibindo o resultado da operação Login. Com e sem erro de login.

Figura 3.8: Tela com a lista de pacientes e tela com as informações do padrão CCR.

Figura 3.9: Telas com o detalhamento de Alergias e Imunizações de determinado paciente.

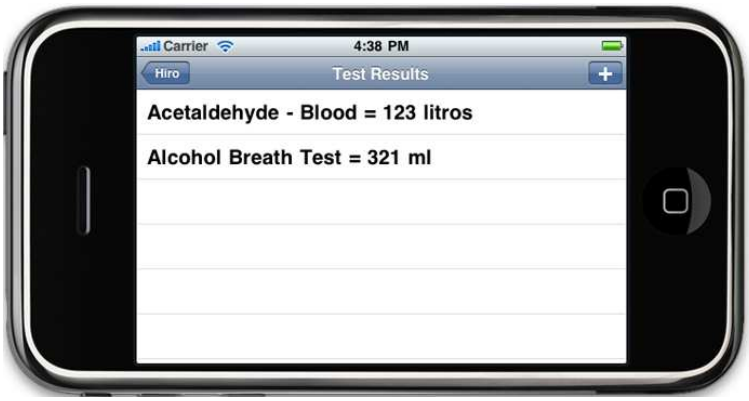

Figura 3.10: Tela no formato horizontal, para uma melhor exibição de informações extensas.

4 CONCLUSÃO

Neste trabalho foram apresentados conceitos sobre o desenvolvimento de aplicativos para dispositivos móveis, mais especificamente para o telefone celular multi-funcional da Apple, o iPhone. Acompanhou-se a criação e implementação de um estudo de caso focado na manipulação de informações médicas baseadas no padrão CCR.

Frente aos desafios encontrados, como a dificuldade em obter uma plataforma que disponibiliza-se a programação para iPhone, e apesar de certos imprevistos no momento de implementação, tendo em vista os objetivos primários deste trabalho, seu resultado pode ser considerado satisfatório, pois o aplicativo criado possibilita ao seu usuário o acesso e manipulação de seus dados encontrados no serviço Health do Google.

A programação para iPhone mostra-se desafiadora e instigante, pois as possibilidades que essa arquitetura disponibiliza são consideráveis. Idéias das mais diversas tem surgido e, com isso, a gama de aplicativos para o dispositivo da Apple tem crescido exponencialmente.

Espera-se que o aplicativo construído sirva para uma futura análise na criação de interfaces de usuário que lidam com informações de caráter médico a serem desenvolvidas pelo projeto ClinicSpace. Além disto, em um futuro próximo, pretende-se refinar a maneira como os dados são exibidos pelo aplicativo, bem como adicionar novas funcionalidades. Dentre essas novas funcionalidades, destaca-se aquela onde, durante a inserção de um novo dado no sistema, o aplicativo indicará ao usuário uma série de alergias, medicamentos, patologias, entre outras informações pré-cadastradas no sistema, facilitando a padronização de dados pertencentes a mais de um usuário.

REFERÊNCIAS

ANSI C. Disponível em http://en.wikipedia.org/wiki/ANSIC.

APPLE. **The Objective-C 2.0 Programming Language**. [S.l.: s.n.], 2009. Disponível em: http://developer.apple.com/mac/library/documentation/cocoa/Conceptual/ ObjectiveC/ObjC.pdf.

APPLE. **iPhone Application Programming Guide**. [S.l.: s.n.], 2009. Disponível em: http://developer.apple.com/iphone/library/documentation/iPhone/Conceptual/ iPhoneOS-ProgrammingGuide/iPhoneAppProgrammingGuide.pdf.

APPLE iPhone. Disponível em http://www.apple.com/iphone/.

APPLE. Disponível em http://www.apple.com/.

ASTM. **ASTM E2369 - 05e1 Standard Specification for Continuity of Care Record (CCR)**. West Conshohocken, PA: ASTM International, 2003. Disponível em: http://www.astm.org/Standards/E2369.htm.

FERREIRA, G.; AUGUSTIN, I.; LIBRELOTTO, G. R.; SILVA, F. L. da; YAMIN, A. C. Middleware for management of end-user programming of clinical activities in a pervasive environment. **Workshop on Middleware for Ubiquitous and Pervasive Systems**, [S.l.], 2009.

GMAIL. Disponível em http://www.google.com/mail.

GOOGLE Docs. Disponível em http://www.google.com/docs.

GOOGLE Health - H9. Disponível em http://www.google.com/h9.

GOOGLE Health. Disponível em http://www.google.com/health.

GOOGLE. Disponível em http://www.google.com.

GUERROUAT, A.; RICHTER, H. A Formal Approach for Analysis and Testing of Reliable Embedded Systems. **Electronic Notes in Theoretical Computer Science, Volume 141, Issue 3**, [S.l.], p.91–106, Dezembro 2005.

KAY, A. C. **The Early History of Smalltalk**. Disponível em http://gagne.homedns.org/ tgagne/contrib/EarlyHistoryST.html.

LEE, V.; SCHNEIDER, H.; SCHELL, R. **Aplicações Móveis - Arquitetura, projeto e desenvolvimento**. [S.l.]: Pearson Education do Brasil, 2005. p.1–9.

MARK, D.; LAMARCHE, J. **Beginning iPhone Development - Exploring the iPhone SDK**. [S.l.]: Apress, 2009. p.1–329.

MUCHOW, J. W. **Core J2ME - Tecnologia e MIDP**. [S.l.]: Pearson Education do Brasil, 2006. p.1–8.

NOGUEIRA, W. F.; LOUREIRO, E. C.; ALMEIDA, H. O. de. Plataformas para Desenvolvimento de Jogos para Celulares. **Journal of Computer Science - Volume 1**, [S.l.], 2005.

SILVA, F. L. da; AUGUSTIN, I. **ClinicSpace**: modelagem de uma ferramenta piloto para definição de tarefas clínicas em um ambiente de computação pervasiva baseado em tarefas e direcionado ao usuário-final. 2009.

SMALLTALK. Disponível em http://www.smalltalk.org.

VICENTINI, C. F.; MACHADO, A.; AUGUSTIN, I. Requisitos de um Registro Eletrônico de Saúde Ubíquo. **SIRC**, [S.l.], 2009.

WEISER, M. The Computer for the 21st Century. **Scientific American**, [S.l.], Setembro 1991.

YAMIN, A.; AUGUSTIN, I.; SILVA, L. C. da; REAL, R. A.; FILHO, A. E. S.; GEYER, C. F. R. EXEHDA: adaptive middleware for building a pervasive grid environment. **Frontiers in Artificial Intelligence and Applications - Vol. 135**, [S.l.], 2005.