UNIVERSIDADE FEDERAL DE ITAJUBÁ

PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Um *framework* para sistemas eletrocardiográficos baseado em Android e HL7

Hícaro da Silva Santos

Itajubá, Novembro de 2017

UNIVERSIDADE FEDERAL DE ITAJUBÁ

PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Hícaro da Silva Santos

Um *framework* para sistemas eletrocardiográficos baseado em Android e HL7

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do título de Mestre em Ciência e Tecnologia da Computação.

Área de Concentração: Sistemas de Computação

Orientador: Prof. Dr. Edmilson Marmo Moreira

Coorientador: Prof. Dr. Odilon de Oliveira Dutra

Novembro de 2017
Itajubá — MG

Agradecimentos

À minha família, amigos e colegas pelo apoio emocional e por estarem do meu lado em vários momentos que precisei.

Agradeço aos meus orientadores, Edmilson e Odilon, pela oportunidade que me ofereceram para trabalhar no projeto, por me guiarem durante esta jornada e por sempre serem atenciosos ao me atender.

Agradeço aos colaboradores, aos técnicos, à seção administrativa e à fundação que liberou verba para minha bolsa de pesquisa, CAPES.

Muito obrigado à todos.

The main activity of programming is not the origination of new independent programs, but in the integration, modification, and explanation of existing ones.

— Terry Winograd (1991) em *Artificial intelligence & software engineering*

Resumo

Esta dissertação apresenta a proposta de um modelo de *framework* para sistemas eletrocardiográficos baseado na plataforma Android e com o padrão de comunicação de serviços de saúde para interoperabilidade de sistemas *Health Level* 7 (HL7). O *framework* providencia funcionalidade para calcular a taxa de ritmo cardíaco, identificar complexos QRS e armazenamento dos dados processados no formato HL7. A sua estrutura de *software* pode ser expandida com a criação de classes filhas especializadas em funções diversas, assim como uma função para o diagnóstico automático de doenças cardíacas. Além disso, o *framework* consegue interoperar com quaisquer sistemas de eletrocardiograma que consigam lidar com dados no padrão HL7, mais especificamente, o *Annotated* ECG (aECG) HL7 versão 3. O modelo vem sendo testado com a plataforma *OpenEinthoven* juntamente com a aplicação Android *BlueHeart*. Foi projetado com o intuito de servir como referência para o desenvolvimento de algoritmos para diagnóstico automático de cardiopatias.

Palavras-chave: Framework. Android. Eletrocardiograma. Deteção de complexo QRS. HL7. Diagnóstico de cardiopatias.

Abstract

This thesis presents a proposal of the modeling of a framework based on Android platform for eletrocardiographic systems capable to interoperate with Health Level 7, which is a healthcare communication standard. The framework provides funcitonality to calculate the cardiac rhythm rate, identify QRS complexes and storage of the processed data in the HL7 format. Its software structure can be expanded with the creation of children classes specialized on other functions, such as automated heart diseases diagnosis algorithms. In addition, the framework can interoperate with any electrocardiogram systems that can handle data in the HL7 format, more specifically, the Annotated ECG (aECG) HL7 version 3. The model has been tested with the OpenEinthoven platform together with the BlueHeart application. It was designed to be a benchmark for the development of automatic diagnosis algorithms of heart diseases.

Keywords: Framework. Android. Electrocardiogram. QRS complex detection. HL7. Heart diseases diagnosis.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Lista de Abreviaturas e Siglas

aECG *Annotated Electrocardiogram*

API *Application Programming Interface*

DICOM-ECG *Digital Imaging and Communications in Medicine*

DOM *Difference Operation Method*

ECG Eletrocardiograma

EMR *Electronic Medical Record*

FDA *Food and Drug Administration*

HL7 *Health Level 7*

JAR *Java ARchive*

RCRIM *Regulated Clinical Research Information Management*

ROI *Region of Interest*

SCP-ECG *Standard Communications Protocol for Computer Assisted Electrocardiography*

SO Sistema Operacional

SWOT *Strengths, Weaknesses, Opportunities and Threats*

UI *User Interface*

XML *Extensible Markup Language*

Capítulo 1

Introdução

Com o advento da tecnologia *mobile*, diversas áreas do conhecimento passaram a se beneficiar e sofrer transformações. Em medicina, no que diz respeito à sistemas que providenciam diagnóstico de doenças relacionadas ao coração, vários dispositivos de eletrocardiograma (ECG) e outras tecnologias começaram a emergir. Para exemplificar as tecnologias envolvendo ECGs, podem ser mencionadas: computação nas nuvens, como o sistema proposto por Xia et al. (2013) para clientes com dispositivos móveis ou *web browsers* voltado para monitoramente e análise de ECG; Redes sociais, como a proposta de Trigo et al. (2013) para acompanhamento cardiovascular de pacientes utilizando a rede social *Twitter* e o padrão de comunicação de saúde HL7 (*Health Level 7*) versão 2.x, onde utilizou-se duas aplicações, uma ferramenta baseada em Android para coletar e "tuitar" sinais cardiovasculares vitais e uma outra ferramenta baseada em Java *desktop* para receber e analisar as informações; *Smartphones*, como o sistema proposto em Depari et al. (2014) para aquisição de rastreamento de eletrocardiograma de condutor único.

Esta dissertação traz uma proposta de um *framework* que é parte da iniciativa *OpenEinthoven*, que tem o objetivo de servir como uma plataforma de código aberto baseada em microcontroladores que adquire e envia dados para um dispositivo inteligente (*smart device*) como um *smartphone*, por exemplo, através de uma interface *bluetooth*. Uma vez que os dados são transferidos para o *smartphone* via *bluetooth*, existem diversas possibilidades no que diz respeito ao processamento de dados para se fazer o diagnóstico automático de doenças cardíacas, considerando a razoável capacidade de processamento que tais dispositivos possuem atualmente.

Para o desenvolvimento de um *framework* capaz de executar algoritmos de diagnóstico em um *smartphone*, é necessário considerar aspectos de usabilidade e desempenho. Os desafios encontrados ao criar um *framework* capaz de executar vários algoritmos de diagnóstico em um mesmo *hardware* são restrições de tempo e controle do desempenho geral. Tais características são geralmente negligenciadas quando se testam algoritmos com dados pré-gravados que não são processados em tempo real.

Embora Android não seja um sistema operacional (SO) de tempo real, ele oferece formas para se lidar com estas questões ao prover serviços e tarefas assíncronas que executam em segundo plano. Outra característica favorável do Android é que ele é um sistema escalável, ou seja, permite fácil integração de novos componentes e métodos, além de oferecer facilidade para desenvolvimento e distribuição. Um *framework* executando em um sistema operacional escalável é desejável para que se possa verificar o desempenho de algoritmos de diagnóstico em ambientes de tempo real.

Para se alcançar tal tarefa, propõe-se nesta dissertação o modelo de uma plataforma baseada em Java Android, projetado para ser usado por qualquer pesquisador interessado no desenvolvimento de sistemas para o diagnóstico automático de cardiopatias. Os requisitos para um *hardware* executar o *framework* proposto é ser um *hardware* com Android embarcado na faixa de versões de 4 a 6, que é a faixa na qual os testes foram conduzidos.

O modelo proposto também pode interoperar com outros sistemas diferentes de ECG. Um dos requisitos para um sistema operar com o modelo é ser capaz de interoperar com o padrão HL7, pois os dados processados são armazenados nesse formato. Em outras palavras, o *framework* possui um método nativo para automaticamente armazenar os dados processados no formato aECG HL7.

Além disso, existem outras características implementadas como o reconhecimento do complexo QRS (despolarização ventricular) e a possibilidade de salvar dados brutos e processados como regiões conhecidas como padrões de doenças no sinal ECG e pela identificação do complexo QRS. Dentre as várias razões para se ter HL7 como o formato para armazenamento dos dados, é válido mencionar: legibilidade (é baseado em XML a partir da versão 3, portanto herda as vantagens do XML), provê facilidade para mineração de dados e tem o apoio de agências influentes, como a FDA (*Food and Drug Administration*) e *Health Level Seven International*.

O *framework* providencia uma estrutura escalável, o que significa que mais funcionalidades (como algoritmos para reconhecimento de doenças cardíacas) sejam desen-

volvidas, integradas e testadas com facilidade. Ele será distribuído sob o contrato de licença GNU[1], portanto qualquer pesquisador será capaz de usá-lo.

1.1 Motivação e justificativa

Na literatura técnica, ainda há carência de *frameworks* computacionais baseados em desenvolvimento *mobile* para dar suporte à aquisição de sinais ECG, que ainda sejam capazes de lidar com as questões de usabilidade e desempenho e que ofereçam suporte para a execução de algoritmos de diagnóstico de doenças cardíacas em tempo real. A necessidade para um *framework* escalável para ECGs vem crescendo, portanto há a oportunidade para projetar um *framework mobile* modular que permitirá que pesquisadores desenvolvam, integrem e testem seus próprios algoritmos, executando suas aplicações em tempo real.

Devido ao crescimento da fatia de mercado do Android como a plataforma *mobile* mais popular (Silva et al., 2015), é de se esperar que, no decorrer dos anos, sistemas de saúde comecem a adotar cada vez mais tecnologias *mobile* para complementar as atividades de monitoramento de pacientes e diagnósticos preventivos, e que o Android ainda será relevante servindo como base para *frameworks* voltados à aplicações de saúde.

Além disso, o trabalho acarretará contribuições para as seguintes áreas:

Contribuições para a computação: O *framework* poderá ser integrado com outros projetos que necessitem parcial ou integralmente dos métodos implementados, pois será distribuído através de um pacote JAVA (.jar) para a plataforma Android, sob o acordo de licença GNU.

Contribuições para a medicina: Por meio de um modelo que pode ser estendido para permitir a detecção de diversas cardiopatias através do sinal eletrocardiográfico adquirido por um eletrocardiógrafo compatível com o *framework*, possibilitando a criação de novas classes para análise e diagnóstico de outras doenças através de sinais biológicos adquiridos por outros tipos de sistemas eletrônicos que podem interoperar com o *framework*.

Destaca-se que o *framework* em questão será distribuído como código aberto, através do sítio da iniciativa *OpenEinthoven*, juntamente com o aplicativo para Android

[1] https://www.gnu.org/licenses/gpl-3.0.en.html

BlueHeart. Além disso, o *framework* será utilizado dentro da universidade, fomentando pesquisas na área de processamento digital de sinais, nos âmbitos de iniciação científica, mestrado e doutorado.

1.2 Objetivos

Os principais objetivos do trabalho são:

- Modelar o código do *framework* de tal forma que seja possível estendê-lo e incorporá-lo em outros projetos, como por exemplo, aplicações com foco em detecção de cardiopatias;
- Tornar o *framework* compatível com a maior parte dos eletrocardiógrafos, utilizando o padrão HL7 para armazenamento dos dados de tal forma a assegurar interoperabilidade.

Os objetivos secundários do trabalho são:

- Desenvolver uma aplicação para dispositivo móvel com o sistema operacional Android utilizando o *framework* proposto, com o intuito de demonstrar a utilização e funcionalidades do *framework*;
- Organizar o código do sistema em um pacote Java (.jar) para permitir a sua distribuição.

1.3 Organização da Dissertação

O texto segue a seguinte estrutura:

- **Capítulo 1** — contém a introdução, a justificativa e os objetivos do trabalho;
- **Capítulo 2** — apresenta a fundamentação teórica relacionada aos elementos, padrões e algoritmos utilizados no *framework* desenvolvido;
- **Capítulo 3** — detalha a modelagem do *framework* e os seus principais componentes;
- **Capítulo 4** — descreve a metodologia adotada para avaliar o *framework* proposto;

- **Capítulo 5** — descreve os experimentos executados e os resultados obtidos a partir da modelagem do *framework*;
- **Capítulo 6** — contém as conclusões da pesquisa e propostas para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo traz uma breve apresentação dos conceitos de eletrocardiografia que são pertinentes no escopo do *framework* e descreve as tecnologias que foram utilizadas para desenvolver o mesmo. Não é objetivo desta dissertação se aprofundar em conceitos e definições estudadas em medicina, por isso elas serão mencionadas brevemente apenas para contextualizar o propósito geral das tecnologias empregadas no *framework* proposto.

O *framework* é baseado em Java para Android e tem aECG HL7 como o formato para armazenamento dos dados (Brown e Badilini, 2005). O *framework* também pode ser integrado com ECGs externos que sejam capazes de interoperar com HL7. A proposta é torná-lo disponível para a comunidade através de um arquivo .jar, de tal forma que pesquisadores possam ser capazes de estendê-lo ou integrá-lo com os seus próprios projetos.

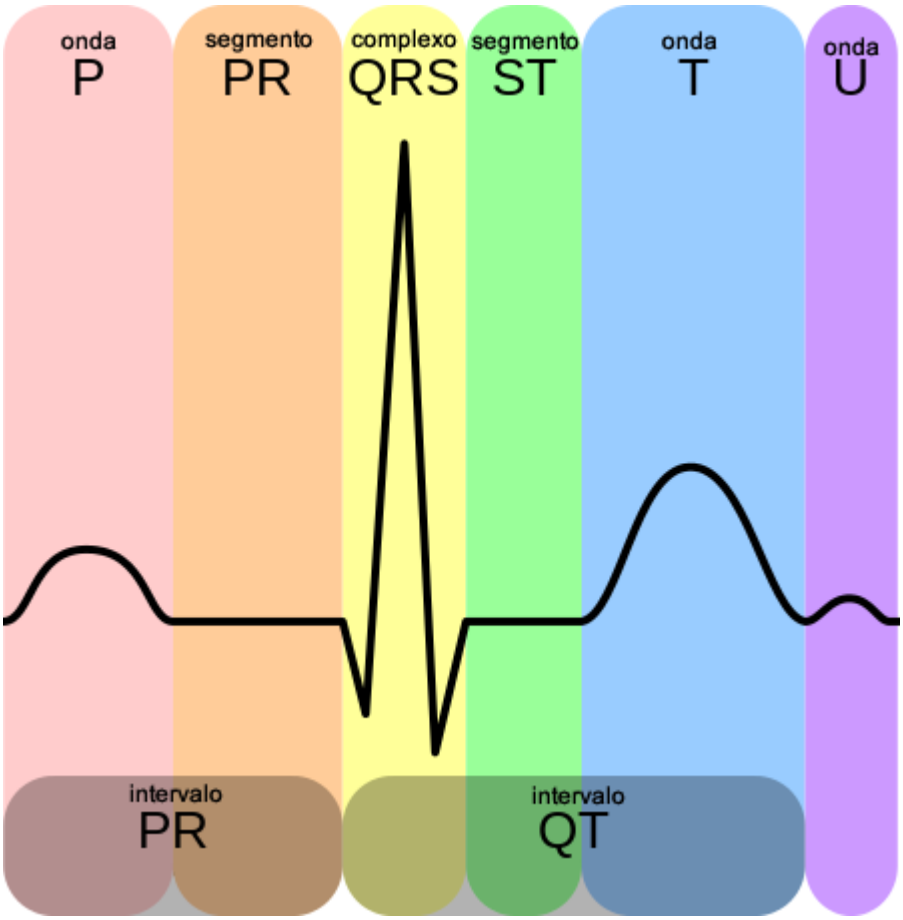
2.1 Definição de eletrocardiografia

Em Morris e Brady (2002), eletrocardiografia é definida como uma ferramenta essencial para avaliação cardiovascular, utilizada para investigar arritmias e fazer diagnóstico de cardiopatias. O conhecimento dos padrões de eletrocardiograma observados em pessoas normais, assim como a compreensão das causas de doenças não cardíacas no traçado do sinal, são fundamentais para se interpretar corretamente o sinal.

2.1.1 Eletrocardiograma

Eletrocardiograma ou ECG é o gráfico cujas transcrições descrevem as mudanças elétricas captadas por meio de eletrodos (colocados nos membros e na região peitoral de pacientes). Tais mudanças elétricas refletem a contração e relaxamento dos músculos do coração, que resultam da despolarização e repolarização das células do miocárdio (músculo cardíaco). É o exame utilizado para avaliação da atividade elétrica do coração, contendo informações sobre o seu estado (Aires, 2012).

Figura 2.1: Sinal de eletrocardiograma padrão



FONTE: https://pt.wikipedia.org/wiki/Eletrocardiograma#/media/File:EKG_Complex_pt.png

Em um sinal ECG padrão é possível reconhecer ondas específicas, como mostradas na Figura 2.1, que são: onda P, gerada pela ativação dos átrios; complexo QRS, que é um conjunto de ondas caracterizada pela excitação ventricular; onda T, que coincide com

a repolarização do músculo ventricular e a onda U, que surge dos potenciais tardios da repolarização ventricular, geralmente com valores baixos (Aires, 2012). O *framework* proposto neste trabalho implementa o algoritmo DOM (Yeh e Wang, 2008), que faz a identificação do complexo QRS, que é o intervalo utilizado para diagnosticar uma grande gama de doenças cardíacas. O algoritmo DOM será explicado na Subseção 2.4.

2.2 Sistema Operacional Android como plataforma

Android é um sistema operacional desenvolvido pela *Google* para dispositivos móveis que inclui um *middleware*, composto por bibliotecas e uma máquina virtual para permitir que desenvolvedores escrevam e executem aplicações para Android. Seu modelo de aplicação é dividido em componentes, denominados *Activities*, *Services*, *Receivers* e *Content Providers*. *Activities* são interativas, permitindo que os usuários interajam com a tela. *Services* executam operações em segundo plano e não providenciam uma interface de usuário. Os *Receivers* atendem à chamadas de transmissão no escopo do sistema. Os *Content Providers* encapsulam os dados e providenciam gerenciamento para a segurança dos dados.

Android não é um sistema operacional de tempo real, porém, providencia componentes e métodos para lidar com computações paralelas, tais como *Background Services* para executar uma operação em uma única *thread* em segundo plano. Portanto, é possível executar operações longas sem afetar a responsividade da interface. Além disso, Android suporta a execução de tarefas assíncronas com a classe *AsyncTask*. Essa classe é ideal para testar e executar algoritmos que podem levar alguns segundos para terminar, sem manter a interface de usuário ociosa.

No caso de operações que requerem longos períodos, o Android ainda providencia um conjunto de várias APIs em um pacote chamado java.util.concurrent, que são a *Executor interface*[1], a classe *ThreadPoolExecutor*[2] e a classe *FutureTask*[3].

Essas alternativas são úteis para a implementação de algoritmos de diagnóstico que irão possuir diferentes implementações com intervalos de execução distintos. Além disso, dispositivos diferentes vão variar em poder de processamento. No trabalho proposto, essas características são elementos chave para permitir a integração de uma grande faixa de algoritmos de diagnósticos para doenças, que poderão ser executados por serviços em

[1] http://developer.android.com/reference/java/util/concurrent/Executor.html
[2] http://developer.android.com/reference/java/util/concurrent/ThreadPoolExecutor.html
[3] http://developer.android.com/reference/java/util/concurrent/FutureTask.html

segundo plano (*background services*) ou através de tarefas assíncronas (*asynchronous tasks*) sobre o *framework*, sem interferir com a interface de usuário.

2.3 Padrão de Comunicação *Health Level 7*

HL7 (*Health Level 7*) é um conjunto de padrões de comunicação internacional para a troca de dados clínicos e administrativos entre aplicações de *software* utilizadas por vários fornecedores de serviços de saúde. Os padrões HL7 são produzidos pela organização *Health Level Seven International*.

Dentre os padrões definidos como os padrões primários da *HL7 International*, existem dois que tratam de especificações de interoperabilidade para transações médicas e de saúde: HL7 versão 2.x e versão 3.

2.3.1 HL7 versão 2.x

O HL7 versão 2 foi criado em 1989, com o objetivo de dar suporte ao fluxo de trabalho em serviços de saúde. O padrão define uma série de mensagens eletrônicas para dar suporte aos processos clínicos, financeiros e administrativos. As mensagens usam uma sintaxe baseada em segmentos e delimitadores de um caracter. Os segmentos possuem compostos separados por delimitadores. Ainda, os compostos podem ter subcomponentes separados por subcomponentes delimitadores.

A listagem 2.1 mostra um exemplo (Spronk, 1999) de mensagem no formato HL7 versão 2.

Listagem 2.1: Exemplo de mensagem no formato HL7 V2

```
1 MSH|^~\&|GHH LAB|ELAB-3|GHH OE|BLDG4|200202150930||ORU^R01|CNTRL
      -3456|P|2.4<cr>
2 PID|||555-44-4444||EVERYWOMAN^EVE^E^^^^L|JONES|19620320|F|||153
      FERNWOOD DR.^
3 ^STATESVILLE^OH^35292||(206)3345232|(206)752-121||||AC555444444||67-
      A4335^OH^20030520<cr>
4 OBR|1|845439^GHH OE|1045813^GHH LAB|15545^GLUCOSE
      |||200202150730|||||||||
5 555-55-5555^PRIMARY^PATRICIA P^^^^MD^^|||||||||F||||||444-44-4444^
      HIPPOCRATES^HOWARD H^^^^MD<cr>
6 OBX|1|SN|1554-5^GLUCOSE^POST 12H CFST:MCNC:PT:SER/PLAS:QN||^182|mg/dl
      |70_105|H|||F<cr>
```

Na listagem 2.1, o segmento do cabeçalho da mensagem (MSH, do inglês *Message Header*) contém o tipo da mensagem, neste caso, ORU^R01, que identifica o tipo da mensagem e o evento que a ativa. O remetente é o GHH Lab em ELAB-3. A aplicação que a recebe é o sistema GHH OE localizado em BLDG4. A mensagem foi enviada no dia 15 de Fevereiro de 2002 (2002-02-15) às 09:30. O cabeçalho da mensagem é o segmento inicial da estrutura da mensagem. O segmento PID (*Patient Identification*) contém as informações demográficas do paciente, que são: Eve E. Everywoman nasceu em 20 de março de 1962 (1962-03-20) e mora em *Statesville OH*. Seu número de ID de paciente é 555-44-4444. O segmento OBR (*Observation Request*) identifica a observação que foi originalmente ordenada: 15545^GLUCOSE. A ordem da observação partiu de *Patricia Primary MD* e foi realizada por *Howard Hippocrates MD*. E, finalmente, o seguimento OBX (*Observation*) contém os resultados da observação: 182 mg/dl.

A tabela 2.1 apresenta uma visão geral sobre os benefícios e desafios de se adotar o formato HL7 versão 2.

Tabela 2.1: Avaliação do padrão HL7 V2

Padrão	Benefícios	Desafios
HL7 V2	• Permite customização para o complexo mundo do sistema de saúde • Muito mais barato para construir interfaces HL7 comparando-se com interfaces customizadas • Providencia 80% da interface e um *framework* para negociar os restantes 20% em uma base de interface para interface • Historicamente construído de uma forma *ad hoc*, permitindo que as áreas mais críticas sejam definidas primeiro • geralmente providencia compatibilidade entre as versões 2.X	• Definições opcionais e vagas no HL7 levam a discrepâncias nas interfaces HL7 • Não inclui necessidades internacionais • Sem compatibilidade com HL7 V3 • Define vários requisitos a serem negociados antes da construção da interface • Fornecedores de aplicações não apoiam as últimas versões lançadas do HL7

FONTE: *Corepoint Health (2010)*

2.3.2 HL7 versão 3

O desenvolvimento do HL7 versão 3 começou por volta de 1995, resultando em um padrão de publicação em 2005. Diferentemente da versão 2, esse padrão é baseado em uma metodologia formal e tem princípios orientados a objeto. O sistema de mensagens define uma série de mensagens de texto, chamadas de interações, para suportar assistência de fluxo de trabalho médico. Além disso, o sistema de mensagem é baseado na sintaxe XML.

A listagem 2.2 mostra um exemplo parcial (Spronk, 1999) de mensagem no formato HL7 versão 3, referenciando o conteúdo de domínio (*Domain Content*). O exemplo completo pode ser visto no anexo I.1.

Listagem 2.2: Exemplo parcial de mensagem no formato HL7 V3

```
<observationEvent>
   <id root="2.16.840.1.113883.19.1122.4" extension="1045813"
       assigningAuthorityName="GHH LAB Filler Orders"/>
   <code code="1554-5" codeSystemName="LN"
         codeSystem="2.16.840.1.113883.6.1"
         displayName="GLUCOSE^POST 12H CFST:MCNC:PT:SER/PLAS:QN"/>
   <statusCode code="completed"/>
   <effectiveTime value="200202150730"/>
   <priorityCode code="R"/>
   <confidentialityCode code="N"
      codeSystem="2.16.840.1.113883.5.25"/>
   <value xsi:type="PQ" value="182" unit="mg/dL"/>
   <interpretationCode code="H"/>
   <referenceRange>
      <interpretationRange>
         <value xsi:type="IVL_PQ">
            <low value="70" unit="mg/dL"/>
            <high value="105" unit="mg/dL"/>
         </value>
         <interpretationCode code="N"/>
      </interpretationRange>
   </referenceRange>
   <!-- Exemplo parcial, código abreviado -->
</observationEvent>
```

Na listagem 2.2, o conteúdo de domínio inicia com o seu próprio elemento raiz, neste caso *observationEvent*. Os elementos dentro deste campo especificam o tipo de observação, o ID, o tempo de observação, o *statusCode* e os resultados. O valor

para o resultado é mostrado no elemento *value*. O elemento *interpretationCode* mostra que o elemento foi interpretado como alto (*high* (H)), enquanto que o *referenceRange* providencia os valores normais para esta observação particular.

Uma avaliação do padrão HL7 V3 é mostrada na tabela 2.2

Tabela 2.2: Avaliação do padrão HL7 V3

Padrão	Benefícios	Desafios
HL7 V3	• Não é um *framework* de negociação, mas um padrão • Padrão baseado em modelo que providencia consistência • O papel da aplicação é bem definido • Menos opcionalidade com as mensagens • Mais barato para construir e manter interfaces de médio prazo • Reflete a filosofia da comunidade que busca melhorar o padrão HL7	• Sem compatibilidade com HL7 V2 • Ciclo de adoção longa, a menos que haja mudanças nos requerimentos regulatórios • É necessário retreinamento e reequipamento de pessoal para adoção do padrão • Aplicações terão que dar suporte para ambas as versões V2 e V3 futuramente

FONTE: *Corepoint Health (2010)*

Comparando-se as tabelas 2.1 e 2.2, verifica-se que o formato HL7 V3 surgiu para tornar o padrão mais consistente, em detrimento da customização oferecida pela versão 2. Embora não haja retrocompatiblidade da versão 3 com a versão 2, HL7 V3 vem sendo adotado em diversas regiões do mundo, onde agências do governo conseguem focar seus esforços. Exemplos incluem o Servico de Saúde Nacional do Reino Unido, Instituto de Informação de Saúde do Canadá, entre outros (*Corepoint Health*, 2010).

2.3.3 aECG HL7

Neste trabalho, uma variação do padrão HL7 versão 3 é utilizada, chamado de aECG (*annotated Electrocardiogram)* HL7, que é um padrão voltado especificamente

para ECGs e também baseado na sintaxe XML. Foi criado por volta de 2004 para atender as necessidades da FDA como avaliação sistemática de formas de onda ECG e localizações de medidas (*annotations*), as quais devem estar disponíveis com a aplicação. O *framework* proposto providencia um método para gerar automaticamente o arquivo aECG HL7 correspondente logo após as medições. Esse método foi implementado seguindo as normas do guia de implementação do aECG HL7 (Brown e Badilini, 2005). O formato aECG HL7 foi escolhido por ser um padrão voltado especificamente para ECG e por adotar as convenções do HL7 versão 3. Desta forma, é possível assegurar a interoperabilidade do *framework* com quaisquer eletrocardiógrafos ou sistemas, de forma geral, que sejam compatíveis com HL7 V3.

Com a adoção do Padrão de Formas de Onda XML aECG pela *Health Level Seven International*, tornou-se possível a submissão de aECGs digitais para estabelecer a segurança cardíaca de novas drogas. O Padrão de Formas de Onda aECG final foi autorizado por meio de um grupo consistindo de representantes de organizações patrocinadoras, laboratórios principais para ECG, instituições acadêmicas e companhias de dispositivos médicos.

De acordo com a análise SWOT (*Strengths, Weaknesses, Opportunities and Threats*) realizada por Bond et al. (2011), que pode ser visto na tabela 2.3, o HL7 é um forte candidato a se tornar o formato dominante para armazenar dados ECG. Entre seus pontos fortes estão legibilidade, grande comunidade, apoio das organizações FDA e *HL7 International*, integração com as tecnologias XML e facilitação para mineração de dados. Dentre suas fraquezas estão o uso extensivo de código, tamanho relativamente grande de arquivos (quando comparado com o binário, que não é legível), tempo e data não utilizam convenções XML e tem pouco suporte direto de eletrocardiógrafos.

A comparação entre os pontos fortes e fracos do HL7 mostram que a comunidade dos serviços de saúde tem muito mais a se beneficiar por adotar e considerar o padrão HL7 do que não adotá-lo, portanto esse trabalho apresenta um *framework* que gera aECG HL7 como o formato de armazenamento.

Tabela 2.3: Análise SWOT do padrão aECG HL7

Forças	• Herda as vantagens do XML • Grande comunidade • Tem o apoio de agências influentes, como a FDA e HL7 • Legibilidade • Pode ser aberto por quase qualquer editor de texto • Pode facilmente integrar tecnologias relacionadas com XML • Auxilia aplicações de mineração de dados • Integra bem com a Arquitetura de Documento Clínico do HL7
Fraquezas	• Herda as desvantagens do XML • Poucos eletrocardiógrafos suportam diretamente o formato • Uso extensivo de código • Complexo • Produz arquivos relativamente grandes • Pode vir a ser útil apenas em testes de drogas clínicas • Atributos como a data e o tempo não utilizam convenções XML • Arquivos aECG não são necessariamente válidos para a FDA
Oportunidades	• Se tornar o padrão dominante para armazenar dados ECG • Ser inerentemente adotado e suportado por eletrocardiógrafos
Ameaças	• A comunidade pode concluir que o formato é muito complexo e não pode submeter seus dados ECG para a FDA • Um consenso que este formato é apenas útil para testes de drogas clínicas e não para propósito geral

FONTE: Adaptado de Bond et al. (2011)

A listagem 2.3 apresenta um exemplo didático de arquivo aECG HL7 extraído e adaptado do guia de implementação do formato (Brown e Badilini, 2005).

Listagem 2.3: Exemplo de mensagem no formato aECG HL7

```
<component>
   <sequenceSet>
      <component>
         <sequence>
            <code code="TIME_ABSOLUTE" codeSystem="
                2.16.840.1.113883.5.4"/>
            <value xsi:type="GLIST_TS">
               <head value="20021122091000.000"/>
               <increment value="0.002" unit="s"/>
            </value>
         </sequence>
      </component>
      <component>
         <sequence>
            <code code="MDC_ECG_LEAD_I" codeSystem="
                2.16.840.1.113883.6.24"/>
            <value xsi:type="SLIST_PQ">
               <origin value="0" unit="uV"/>
               <scale value="5" unit="uV"/>
               <digits>0 1 2 3 2 1 0 0 0 -1</digits>
            </value>
         </sequence>
      </component>
   </sequenceSet>
</component>
```

É possível observar na listagem 2.3 que a segunda *tag code* possui o parâmetro de mesmo nome com valor de atributo *MDC_ECG_LEAD_I*, que diz respeito ao condutor utilizado para captar as medidas ECG. Além disso, dentro da mesma *tag sequence*, tem-se a *tag digits*, que armazena o traçado do sinal. Neste exemplo, a sequência de dígitos caracteriza a deteção de um segmento PQ (de acordo com a *tag value xsi:type*), que é o intervalo entre as ondas P e Q mostradas na figura 2.1. Dentro da *tag digits*, o intervalo com os valores 0, 1, 2, 3, 2, 1 caracteriza a onda P, e o valor -1 caracteriza o ponto Q do complexo QRS.

O método de armazenagem dos dados do *framework* proposto gera o arquivo XML aECG HL7 das medidas coletadas do sinal ECG, que é similar ao exemplo mostrado na listagem 2.3. Os elementos do código XML mostrados no exemplo serão detalhados a

seguir. De acordo com o guia de implementação do padrão (Brown e Badilini, 2005):

- *component* (opcional): diz respeito às partes componentes do aECG. Corresponde às formas de ondas e anotações. Apesar de ser um elemento opcional, uma mensagem nesse formato não é muito útil sem ter pelo menos uma forma de onda.
- *sequenceSet* (obrigatório): agrupa duas ou mais sequências. Toda sequência deve ter o mesmo comprimento. O primeiro valor de cada sequência é relacionado com o primeiro valor de todas as outras sequências. O segundo valor de cada sequência é relacionado ao segundo valor de todas as outras sequências e assim por diante.
- *sequence* (obrigatório): é uma lista ordenada de valores que possui um código ou dimensão em comum. Os valores de uma sequência são relacionados com os valores de outras sequências dentro de um conjunto de sequências (*sequenceSet*).
- *code* (obrigatório): dá nome à dimensão ou ao tipo dos valores na sequência.
- *TIME_ABSOLUTE*: caracteriza uma sequência de valores no domínio de tempo "absoluto". É o tempo medido pelo calendário Gregoriano.
- *value* (obrigatório): contém (representa) a lista de valores em uma sequência. A lista de valores pode ser gerada de um algoritmo ou explicitamente enumerada.

O campo *increment* carrega a informação do valor do período de tempo e da unidade de tempo para cada medida do sinal ECG. As medidas do sinal bruto dos condutores do ECG são armazenadas no campo *digits*, com os respectivos valores de escala e unidade. Outras ondas ou intervalos específicos são armazenados em sequências, com seus respectivos códigos que podem ser consultados no guia (Brown e Badilini, 2005). Um exemplo de arquivo XML aECG HL7 gerado pelo *framework* pode ser verificado no Apêndice A.

2.4 O Algoritmo DOM

Um sistema para diagnóstico automático de doenças cardíacas depende essencialmente da correta identificação do complexo QRS, pois esse complexo carrega as informações necessárias para analisar o sinal ECG.

Em Yeh e Wang (2008), foi proposto um algoritmo baseado em diferenciação para identificar o pico R e depois as ondas Q e S em um primeiro estágio. Em seu segundo

estágio, a partir da obtenção do complexo QRS, é possível obter a onda P e a onda T aplicando métodos existentes. Esse algoritmo é chamado de Método da Operação de Diferença (DOM, do inglês *Difference Operation Method*). É demonstrado que esse algoritmo é simples, rápido e confiável quando comparado com algoritmos mais complexos como o algoritmo baseado em *wavelet* proposto por Li e Wang (2013) e o algoritmo de Pan e Tompkins (1985).

Apesar de sua simplicidade em termos de *software*, o método DOM depende de um sinal ECG de qualidade para ser aplicado. Por isso, geralmente é necessário a aplicação de um filtro para remoção de ruído randômico e ruídos provenientes da rede elétrica em 60 Hz. Estas etapas de filtragem estão compreendidas na plataforma *OpenEinthoven*.

As etapas subsequentes são implementadas em *software* e por isso foram implementadas no *framework* afim de se detectar o complexo QRS. O método DOM e suas etapas podem ser visualizados na Fig. 2.2 a seguir.

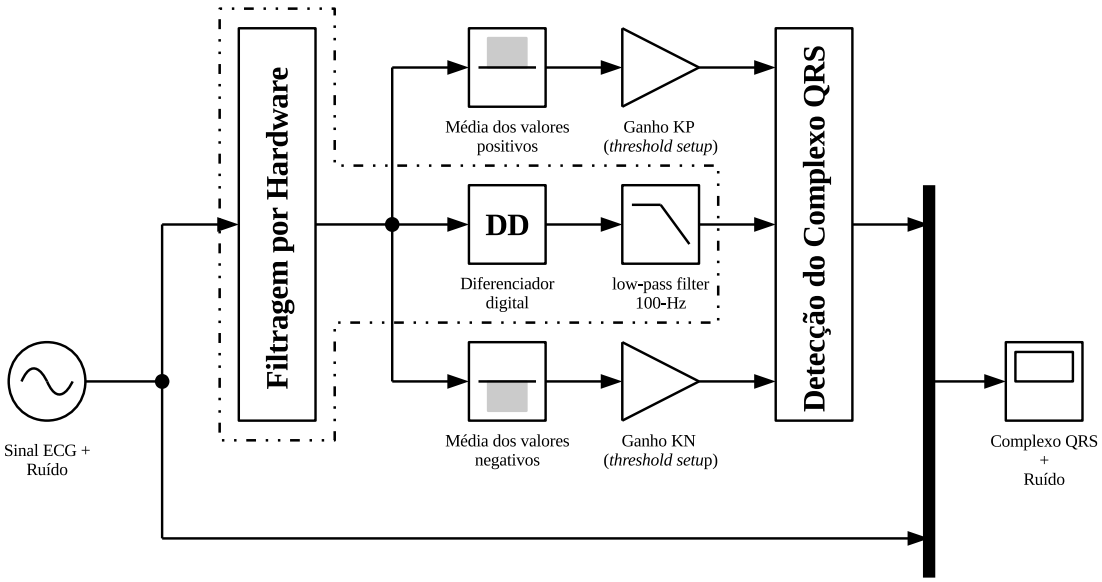


Figura 2.2: Algoritmo DOM

A etapa de detecção dos picos R e, consequentemente, a identificação do complexo QRS, pode ser visualizada no fluxograma apresentado na Fig. 2.3.

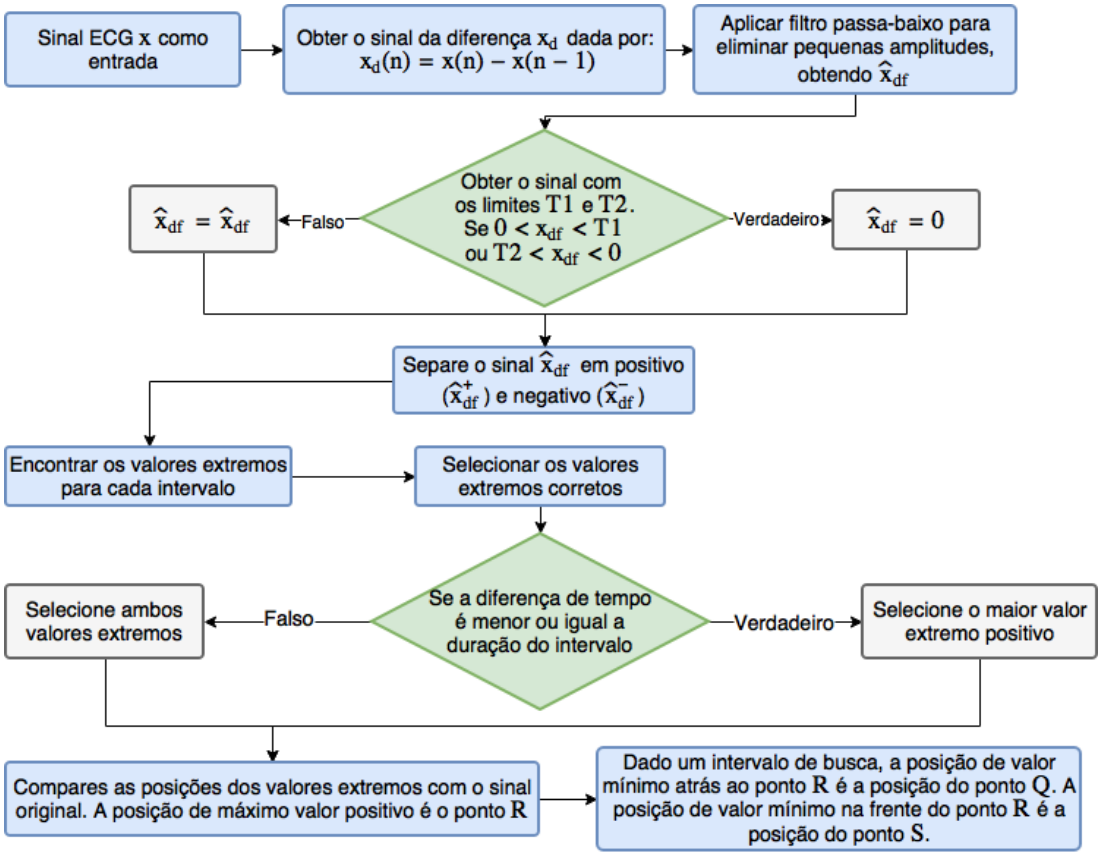


Figura 2.3: Fluxograma do algoritmo DOM

2.5 Trabalhos Relacionados

Na área de saúde, diversos trabalhos começaram a adotar novas tecnologias como dispositivos *mobile*, computação nas nuvens e mídias sociais. Além disso, novos padrões para interoperabilidade de sistemas de informação começaram a emergir.

Um exemplo para ilustrar a mudança gradual no cenário de saúde que emprega a maioria das tecnologias mencionadas pode ser visto em Trigo et al. (2013), que apresenta o desenvolvimento de um sistema para acompanhamento cardiovascular de pacientes ao integrar mídia social e padrões de comunicação de saúde (HL7 versão 2.x). A aplicação para coletar sinais cardiovasculares vitais é baseada em Android e troca informações com outra aplicação baseada em Java para *desktop*.

Em relação a proposta de *middlewares* para integração entre diferentes aplicações de

saúde, há o projeto de Alenazi e Alhamed (2015). O projeto implementa um *middleware plug-in* para processar diferentes versões do HL7 (versões 2 e 3 inclusas) enquanto resolve incompatibilidades, prevenindo brechas de segurança e aumentando a confiabilidade na transferência de mensagens. Outro projeto é um *middleware* para integrar aplicações que providenciam serviços de saúde na Colômbia, proposto por Torres e Ricaurte (2015), que também possui HL7 como o padrão de comunicação para interoperabilidade. Em Liu e Huang (2012), também há uma proposta de *middleware* que utiliza uma estrutura semântica de mapeamento para a conversão entre informações de saúde hetereogêneas.

Sistemas *mobile* são amplamente usados como tecnologias para providenciar serviços para a troca de registros médicos eletrônicos (EMR, do inglês *electronic medical records*), como dispositivos para monitorar pacientes ao permitir gerenciamento remoto da saúde. Dentre os trabalhos que ilustram esses casos, podem ser mencionados Sierra et al. (2014), que construiu um protótipo de sistema no qual registros de saúde eletrônicos podem ser seguramente transferidos entre um dispositivo *mobile* e um sistema de informação médico. Isso é alcançado através do uso de algoritmos de criptografia em ambos o servidor e cliente (dispositivo *mobile*). Outro trabalho é o de Deng e Chen (2016), que implementou um sistema de saúde baseado em Android para ajudar usuários *mobile* a adotarem hábitos de vida mais saudáveis e conscientizá-los sobre a importância da saúde. Kang et al. (2013) propõe um *framework* colaborativo para *lifelog*, com o intuito de monitorar e coletar informações de saúde de pacientes, também baseado em Android, para providenciar os serviços de saúde.

O desenvolvimento de tecnologias para monitoramento de condições cardíacas se beneficiou significativamente de sistemas *mobile*, como pode ser visto no trabalho de Aristomenopoulos et al. (2014), que propõe um dispositivo de assistência ventricular para pacientes, com arquitetura integrada de ponta a ponta de uma aplicação para monitoramente especialista baseada em *web*, que também é baseado em Android/HL7 e tem características de um sistema EMR. Outro trabalho relacionado a aquisição de sinais de eletrocardiograma baseado em *smartphone* é o de Depari et al. (2014), que descreve o desenvolvimento de um sistema de aquisição de sinal de eletrocardiograma de condutor único. Ainda há a proposta para detectar emoções humanas usando o sinal ECG em Covello et al. (2013), que aplica algoritmos para detectar o complexo QRS no sinal ECG. Esses trabalhos mencionados não utilizam *frameworks*, o que significa que as aplicações propostas não podem ser utilizadas para propósitos gerais. A vantagem de se recorrer a um *framework* é usufruir de uma estrutura escalável que permita que os projetos cresçam sem que seja necessário reestruturar suas arquiteturas.

Ao considerar todos os trabalhos mencionados, fica claro que *smartphones* estão tendo um grande papel por permitir o desenvolvimento dos mais variados tipos de aplicações relacionadas à soluções para a saúde. Nesse aspecto, é também observável que o HL7 é um padrão estabelecido e que permite interoperabilidade nos sistemas descritos. Não obstante, na literatura ainda há lacunas em termos de *frameworks* computacionais para suportar a aquisição do sinal ECG, que automaticamente gerem e utilizem o padrão aECG HL7. Além disso, não existem *frameworks* que facilitam o desenvolvimento e acoplamento de novos algoritmos para utilizar o sinal ECG com a finalidade de detectar doenças cardíacas. Tendo isto em vista, um *framework* estruturado para permitir que vários algoritmos para diagnóstico sejam executados concorrentemente em um *smartphone* de forma eficiente encontra-se em necessidade. A próxima seção descreve o *framework* tamanho.

Capítulo 3

Modelo do *Framework*

Este capítulo apresenta a modelagem do *framework*, descrevendo suas características e organização estrutural da hierarquia de seus elementos. Primeiramente, antes de descrever o modelo, será abordado o conceito de *framework*.

3.1 Definição de *framework*

Segundo Riehle (2000), a terminologia *framework* refere-se à abstração que modela um domínio específico ou um aspecto importante do mesmo. *Frameworks* representam um domínio como um projeto abstrato, consistindo de classes abstratas ou interfaces. O projeto abstrato é mais do que um conjunto de classes, uma vez que define como as instâncias das classes irão colaborar umas com as outras em tempo de execução. Em outras palavras, um *framework* serve como um esqueleto que determina como os objetos do mesmo irão se relacionar.

Um *framework* é composto por implementações reutilizáveis na forma de classes abstratas e concretas. Implementações abstratas são classes abstratas ou interfaces que implementam partes de uma abstração do *framework*, mas deixam decisões de implementação crucial para as classes filhas. Esse princípio é denominado de Projeto por Primitivas (Riehle e Dubach, 1999), que define a implementação de uma classe em um pequeno conjunto de operações primitivas que são deixadas em aberto para serem implementadas por meio de classes filhas.

3.2 Propósitos do *framework* proposto

O propósito geral do *framework* é servir como base para os processos de captura, análise, visualização, diagnóstico e armazenamento de sinais biológicos, tais como atividade elétrica do coração (eletrocardiografia ou ECG), atividade elétrica do cérebro (eletroencefalografia ou EEG), atividade elétrica das células musculares (eletromiografia ou EMG), entre outros. O *framework* providencia estrutura para um conjunto de métodos voltados para o desenvolvimento de algoritmos de diagnóstico de doenças. Neste trabalho, há o enfoque em apresentar a integração do *framework* com uma aplicação voltada para análise de eletrocardiograma, denominada *BlueHeart*.

O *framework* proposto não providencia diagnóstico, ao invés disso, o que o *framework* oferece são os meios para o desenvolvimento de algoritmos de diagnóstico automático. A estrutura do *framework* define como irá ocorrer a coleta do sinal ECG, a análise, o armazenamento e o diagnóstico. Portanto, qualquer método para diagnóstico de cardiopatias pode ser implementado sobre o *framework*.

Em relação aos elementos de *software* da aplicação *BlueHeart*, a mesma implementa o cálculo da taxa de batimento cardíaco, detecção do complexo QRS e armazenamento dos dados processados no padrão aECG HL7. Desta forma, vários algoritmos de diagnóstico de cardiopatias podem reutilizar os cálculos feitos previamente pelos métodos da classe de análise, poupando tempo e processamento.

3.3 Modelagem do *framework*

Figura 3.1 mostra o diagrama de classes do *framework*.

A arquitetura do *framework* consiste de seis classes abstratas, que são:

- `Sensor`: define o comportamento da classe responsável por fazer a captura de sinais biológicos (através do método `captureRawData`) e armazenamento dos dados brutos (com o parâmetro `rawData`).
- `Filter`: descreve a estrutura de uma classe filtro, que recebe os dados brutos a partir de uma instância que herde da classe `Sensor`. Além disso, deve conter um método para aplicar seu respectivo filtro (método `applyFilter`) para reduzir o ruído do sinal biológico captado, e armazenar os dados filtrados (com o parâmetro `filteredData`).

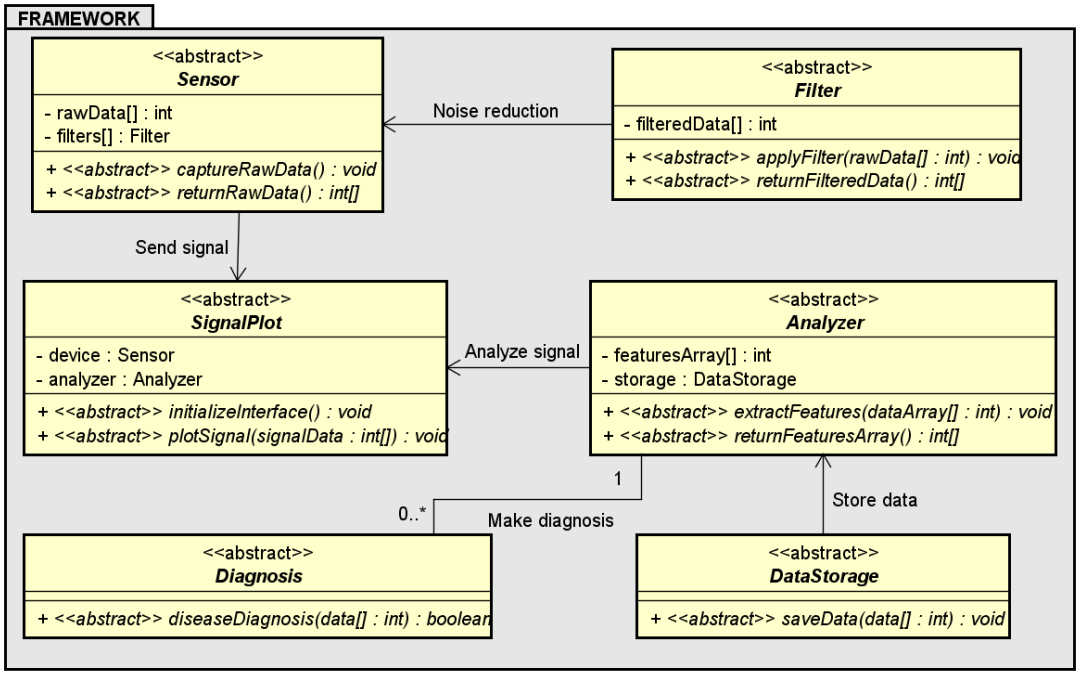


Figura 3.1: Diagrama de classes do *framework*

- `Analyzer`: define o comportamento da classe que tem por objetivo extrair características do sinal biológico medido (com o método `extractFeatures`), para que seja possível fazer diagnóstico de doenças a partir dessas características. Além disso, deve estar associado a uma classe para interface gráfica e uma classe para armazenar o vetor de características em disco (a partir do parâmetro `featuresArray`), através de um formato clínico específico.

- `SignalPlot`: define a estrutura da classe responsável por prover a interface gráfica para as aplicações (através do método `initializeInterface`) e pela plotagem gráfica do sinal biológico (pelo método `plotSignal`).

- `Diagnosis`: descreve o corpo da classe que tem por objetivo detectar alguma doença específica, analisando um vetor de características oriundo da análise de um vetor contendo dados de sinais biológicos (através do método `diseaseDiagnosis`).

- `DataStorage`: define o corpo da classe responsável por fazer o armazenamento dos dados analisados (através do método `saveData`), salvando os dados em disco.

A classe `SignalPlot` possui os métodos `initializeInterface`, que é responsável

pela construção da interface gráfica, e o `plotSignal`, que faz a plotagem do sinal adquirido. O método `plotSignal` recebe como parâmetro o respectivo vetor (`signalData`) contendo os dados adquiridos por uma instância que herde da classe `Sensor`. Além disso, possui dois atributos, `device` e `analyzer`, que são resultados dos relacionamentos de agregação, respectivamente, com as classes `Sensor` e `Analyzer`. Ambos relacionamentos possuem cardinalidade 1. A classe `SignalPlot` tem a função de fazer a plotagem de um sinal específico adquirido por um `Sensor`.

A classe `Sensor` possui dois métodos: `captureRawData` e `returnRawData`. O método `captureRawData` tem a função de fazer a aquisição de um sinal biológico e armazená-lo no vetor `rawData`. O acesso ao vetor `rawData` é feito através do método `returnRawData`. A classe `Sensor` deve estar associada com pelo menos um `Filter`, que é uma classe responsável por aplicar filtros no sinal adquirido para reduzir o ruído.

No modelo do *framework* também há a classe `Filter`, que também possui dois métodos: `applyFilter` e `returnFilteredData`. O método `applyFilter` tem a função de executar algum algoritmo de filtro para o sinal adquirido pela classe `Sensor`, que pode estar associada a mais de um `Filter`. O método `applyFilter` recebe como parâmetro um vetor com os dados brutos (`rawData`). O novo sinal com ruído reduzido é armazenado no atributo `filteredData`. O acesso ao `filteredData` é feito por meio do método `returnFilteredData`.

Para o tratamento e extração de características do sinal adquirido, é proposto a classe `Analyzer`, que possui os métodos `extractFeatures`, responsável por executar algum algoritmo que extraia características conhecidas de um sinal biológico específico e armazená-lo em um vetor (`featuresArray`), e o `returnFeaturesArray`, método que devolve o vetor de características. O método `extractFeatures` recebe como parâmetro o sinal filtrado adquirido (`dataArray`). A classe `Analyzer` também possui um atributo do tipo `DataStorage` (`storage`) por meio de relacionamento de agregação, para poder fazer armazenagem em disco do vetor de características. Além disso, pode estar associada a nenhuma ou a várias classes do tipo `Diagnosis`.

Com o intuito de armazenar os dados processados pela classe `Analyzer` em algum formato de comunicação clínico, é proposto a classe `DataStorage`, que possui o método `saveData`. Este método recebe como parâmetro um vetor de características extraídas de um sinal biológico adquirido (`data`), e o armazena em algum formato padrão que assegure interoperabilidade entre sistemas de informação clínicos e hospitalares.

No modelo também é proposto a classe `Diagnosis`, que possui um método para

fazer diagnóstico de doenças a partir do sinal coletado e analisado por uma instância que herde da classe `Analyzer`. Este método é denominado de `diseaseDiagnosis`, que é um método booleano que recebe como parâmetro um vetor de características (`data`) gerado por um `Analyzer`. Retorna verdadeiro, caso o algoritmo retorne resultado positivo para o diagnóstico de uma doença, e falso, caso contrário.

Figura 3.2 apresenta o diagrama de sequência que ilustra um cenário de aquisição e análise de sinais biológicos para uma aplicação que seja desenvolvida sobre o *framework* proposto.

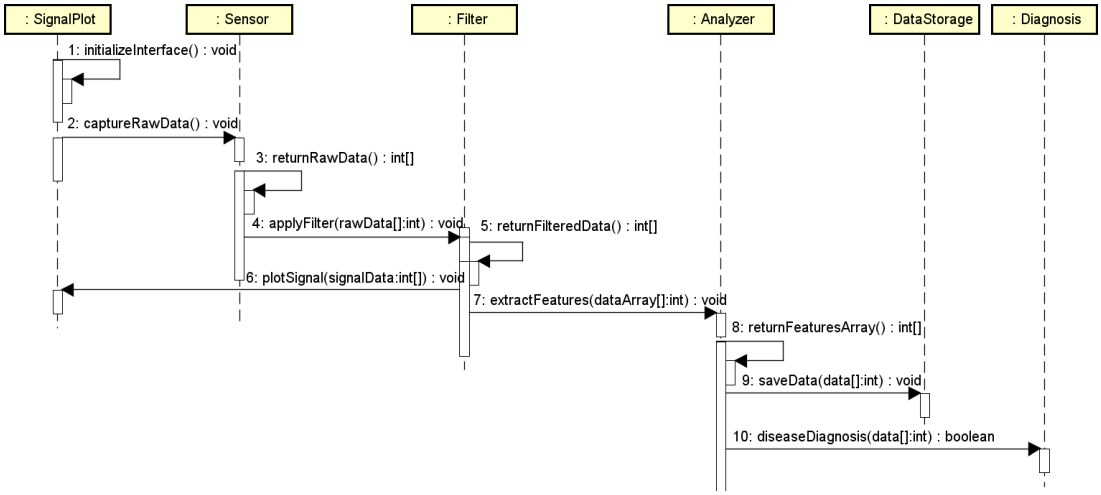


Figura 3.2: Diagrama de sequência para uma possível aplicação que implemente a estrutura do *framework* em um cenário de aquisição e análise de sinais biológicos

No diagrama de sequência da Fig. 3.2, é possível observar a ordem dos métodos e mensagens, que são: (1)`initializeInterface`, que é o método da classe `SignalPlot` que cria a interface gráfica e a grade para plotagem do sinal biológico adquirido. Após a criação da interface, já é possível adquirir o sinal biológico e armazená-lo. Isto é feito pelo método da classe `Sensor`, denominado (2)`captureRawData`. Em seguida, o sinal bruto precisa ser filtrado para reduzir o ruído. Com essa finalidade, há o método (3)`returnRawData` da classe `Sensor`, que retorna os dados brutos para uma instância de classe que herda da classe `Filter`. Toda instância que herde da classe `Filter` tem o método (4)`applyFilter`, que aplica determinado filtro a fim de reduzir o ruído no sinal, antes da extração de características. Após a aplicação dos filtros, é possível plotar o sinal e extrair características do mesmo. Todo o filtro possui um método para transferir o sinal para plotagem e análise, que é o método (5)`returnFilteredData`. Toda instância que herda da classe `SignalPlot` possui o método (6)`plotSignal`, que

faz a contínua plotagem do sinal biológico adquirido em tempo de execução. E toda instância que herda da classe `Analyzer` possui o método (7)`extractFeatures`, que extrai características dos dados de acordo com o tipo de sinal biológico a ser analisado. Após a análise, armazena-se o vetor de características gerado. Com o término da extração de características do sinal, é possível transferir o vetor de características para a instância da classe responsável pela armazenagem dos dados processados e para a instância da classe que realiza diagnósticos a partir do mesmo. Para isso, o método (8)`returnFeaturesArray` da classe `Analyzer` é utilizado. Com o intuito de armazenar o vetor de características em algum formato de comunicação para sistemas de saúde especificado, o método (9)`saveData` de uma instância que herde da classe `DataStorage` é utilizado. E para a realização de diagnóstico de doenças específicas a partir do vetor de características, o método (10)`diseaseDiagnosis` é utilizado em uma instância que herde da classe `Diagnosis`.

Em suma, é possível definir os métodos e mensagens da seguinte forma:

1. `initializeInterface`: método da classe `SignalPlot` que cria a interface gráfica e a grade para plotagem do sinal biológico adquirido.
2. `captureRawData`: método da classe `Sensor` que adquire o sinal biológico e o armazena.
3. `returnRawData`: método da classe `Sensor` que retorna os dados brutos para uma instância de classe que herda da classe `Filter`.
4. `applyFilter`: método da classe `Filter` que aplica determinado filtro a fim de reduzir o ruído no sinal, antes da extração de características.
5. `returnFilteredData`: método da classe `Filter` que retorna o sinal filtrado para instâncias que herdem das classes `SignalPlot` e `Analyzer`.
6. `plotSignal`: método da classe `SignalPlot` que faz a contínua plotagem do sinal biológico adquirido em tempo de execução.
7. `extractFeatures`: método da classe `Analyzer` que extrai características dos dados de acordo com o tipo de sinal biológico a ser analisado. Em seguida, armazena o vetor de características gerado.
8. `returnFeaturesArray`: método da classe `Analyzer` que retorna o vetor de características para instâncias que herdem das classes `DataStorage` e `Diagnosis`.

9. `saveData`: método da classe `DataStorage` que salva o vetor de características do `Analyzer` em algum formato de comunicação para sistemas de sáude especificado.

10. `diseaseDiagnosis`: método que recebe um vetor de características de um `Analyzer` e implementa um diagnóstico para alguma doença específica.

3.4 A aplicação *BlueHeart*

O *framework* será utilizado com a aplicação *BlueHeart*, que é uma interface de usuário para *hardware* de aquisição de ECG voltada para Android. A aplicação *BlueHeart* é a aplicação padrão executando sobre o *framework*, implementando parcialmente sua estrutura.

Figura 3.3 mostra o diagrama de classes parcial do *framework* e da aplicação *BlueHeart*. A arquitetura do *BlueHeart* consiste de três classes principais: `ECG_Analysis`, para processamento dos dados ECG, `Save_Data_HL7`, para gerar o arquivo aECG HL7 e `Tracking`, para gerar a interface gráfica e fazer a plotagem do sinal. As classes `ECG_Analysis`, `Save_Data_HL7` e `Tracking` herdam, respectivamente, das classes `Analyzer`, `DataStorage` e `SignalPlot` do *framework*.

Na aplicação *BlueHeart*, o sinal ECG bruto é adquirido dos sensores via *bluetooth*. Os dados brutos são transferidos para uma instância da classe `ECG_Analysis` do *BlueHeart*, através de um *buffer*.

No escopo da aplicação *BlueHeart*, a classe `ECG_Analysis` é a base para o processamento de dados ECG. Para cada iteração, obtendo os dados da aplicação via *bluetooth*, a `ECG_Analysis` acrescenta os novos dados para um vetor que é processado logo após a aquisição. Esta classe ainda é responsável por calcular a taxa de batimentos cardíacos usando os picos R identificados pelo algoritmo DOM. Além disso, a classe `ECG_Analysis` tem um método para identificar o complexo QRS e outro para buscar pelos picos R nos dados. Logo após os dados serem processados e plotados na tela por um objeto da classe `Tracking`, a `ECG_Analysis` chama o método de salvamento da classe `Save_Data_HL7` para iniciar a geração do arquivo XML aECG HL7.

Como pode ser visto na Fig. 3.1, o *BlueHeart* implementa um método para calcular a taxa de ritmo cardíaco, que é o `calculateHeartRate()`. Ele também possui outro método para obter a média dos batimentos cardíacos, nomeado de `getMeanHeartRate` e o algoritmo DOM que inicia com a chamada do método `searchPeaksR()`. Além disso,

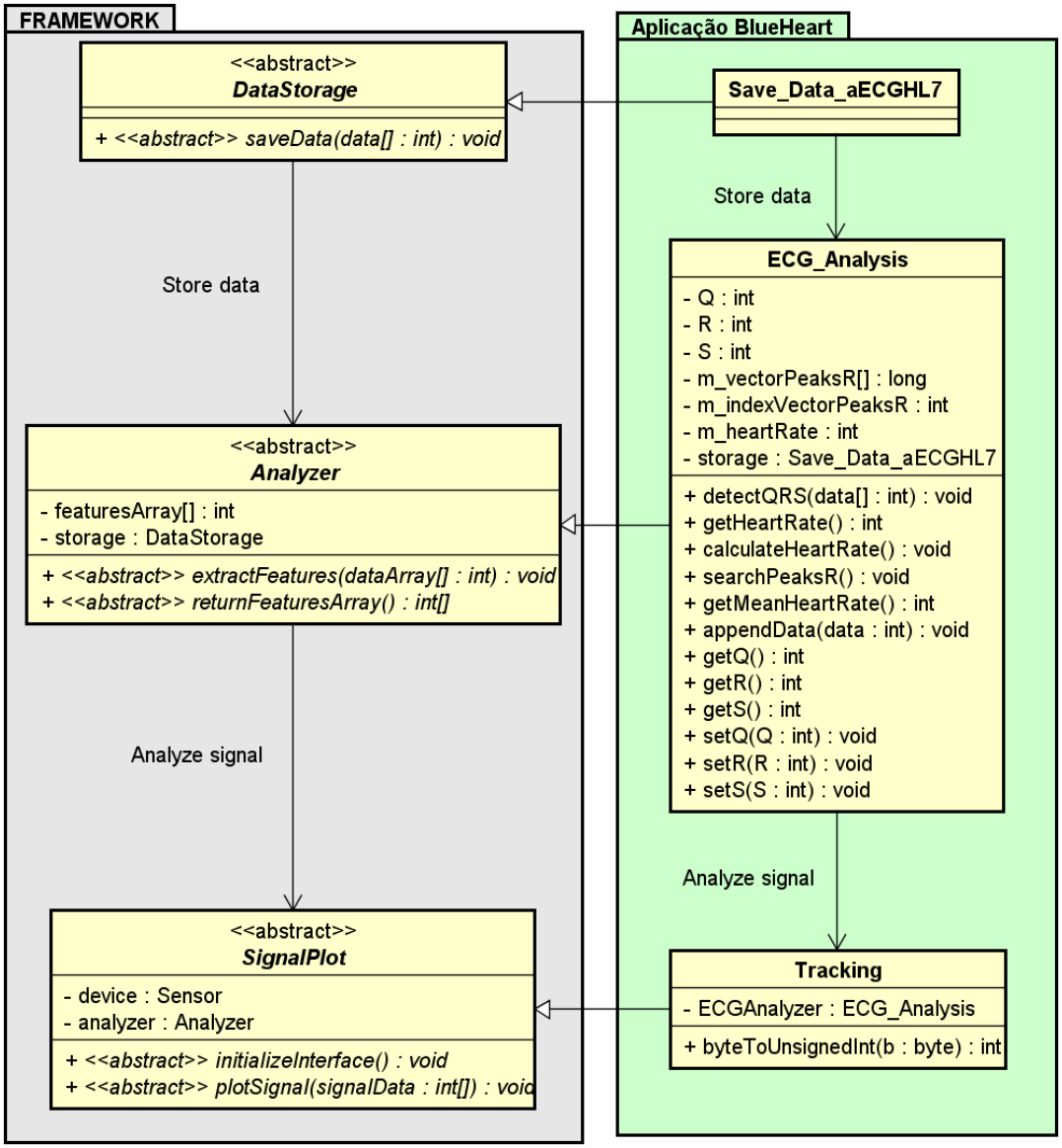


Figura 3.3: Diagrama de classes parcial do *framework* e da aplicação *BlueHeart*

o *BlueHeart* disponibiliza métodos para recuperar a taxa de batimentos cardíacos, o vetor de picos R, os índices dos picos R e o complexo QRS. Também é possível recuperar os dados brutos e os picos individuais que compõem o complexo QRS.

3.4.1 Fluxo de aplicação provido pelo *framework*

O que torna a estrutura de classes do *framework* flexível é o fluxo de aplicação que ela oferece, que pode ser verificado no fluxograma da aplicação *BlueHeart*, mostrada na Figura 3.4. O diagrama está em inglês com o intuito de favorecer a universalidade do uso.

No diagrama de fluxo da Figura 3.4, existem cinco processos descrevendo o comportamento esperado da aplicação *BlueHeart*. Qualquer aplicação criada a partir do *framework* proposto será implementada em uma forma similar para ser capaz de utilizar os métodos descritos no *framework*, da forma pela qual o mesmo foi projetado. Esses processos são detalhados a seguir:

1. *Initialize UI process* (Processo de inicializar a Interface de Usuário): Processo responsável pela criação da interface. A aplicação *BlueHeart* faz uso de uma versão modificada da biblioteca *GraphView*[1], a partir da qual os recursos de interface são carregados para construir a grade para plotar o sinal ECG (as modificações na *GraphView* foram feitas para lidar com as escalas milimétricas do padrão de ECG, para qualquer tamanho de tela).

2. *Stipulate the measurement time for ECG process* (Processo de estipular o tempo de medida para coletar o sinal ECG): Exibe um menu de configuração para determinar o tempo para as medidas do sinal ECG. Entre as opções, é possível optar por um tempo indeterminado para manter a ocorrência da medição até que o usuário decida pará-la manualmente. Outra opção é escolher um tempo determinado para as medidas, assim a parada é automática após o tempo expirar.

3. *Read data from the sensors process* (Processo de leitura de dados dos sensores): É o processo de aquisição em tempo real do sinal ECG coletado pelos sensores, além dos cálculos da taxa de batimentos cardíacos.

4. *Show the ECG trace signal process* (Processo de exibir o sinal ECG): Esse processo implica na plotagem do sinal ECG na tela em tempo real, enquanto os sensores continuamente adquirem o sinal ECG.

5. *Generate aECG HL7 file process*: Finalmente, este é o processo que descreve a construção do arquivo de armazenagem baseado nas especificações definidas pela

[1] http://www.android-graphview.org/simple-graph

Gestão de Informação de Pesquisa Clínica Regulada (RCRIM, do inglês *Regulated Clinical Research Information Management*) da *Health Level Seven International*, descritas no guia de implementação do padrão (Brown e Badilini, 2005). O arquivo de armazenagem gerado preenche todas as *tags* mínimas requeridas no XML para garantir a conformidade com o padrão. Após cada iteração através de todo o sistema de informação, o usuário pode sempre realizar uma outra medida ou escolher encerrar a aplicação.

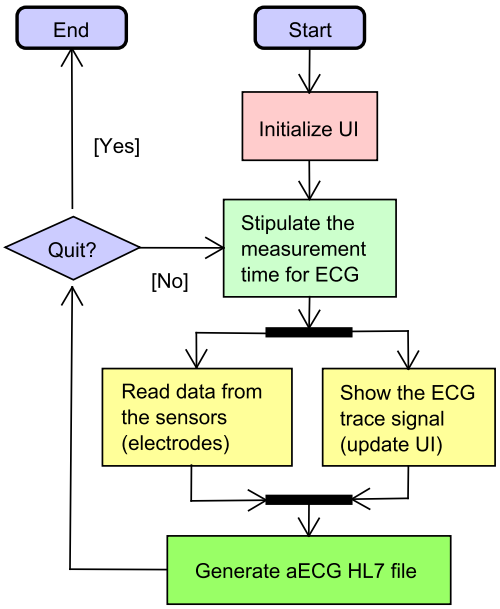


Figura 3.4: Diagrama de fluxo da aplicação *BlueHeart*

3.4.2 Integração do *framework* com algoritmos de diagnóstico

A escalabilidade do *framework* pode ser observada no exemplo da Figura 3.5, com a criação de novas classes para fazer diagnóstico de doenças específicas do coração.

Figura 3.5 mostra como o *framework* foi projetado para ser integrado com outros algoritmos de diagnóstico de cardiopatias. O conjunto de classes com os seus respectivos métodos para análise e diagnóstico de doenças cardíacas herdam, respectivamente, das classes `Analyzer` e `Diagnosis`. A partir deste ponto, é possível definir o comportamento das classes concretas que herdaram de `Analyzer` para análise dos sinais ECG coletados e da extração das características a partir destes sinais.

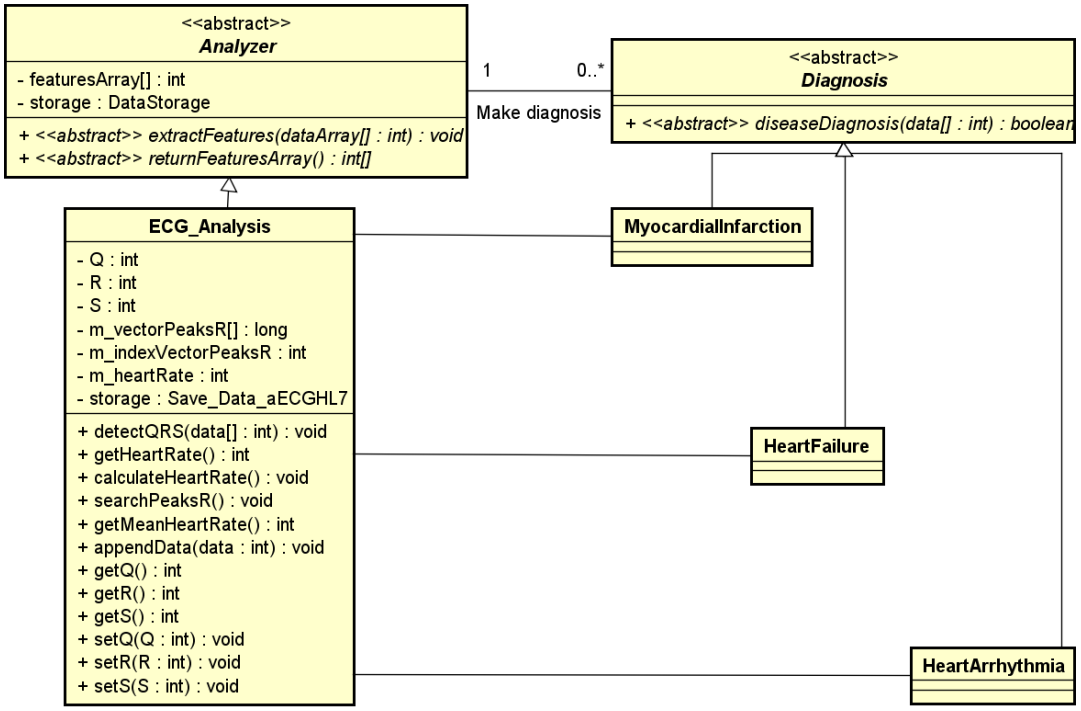


Figura 3.5: Exemplo de integração do *framework* com algoritmos de diagnóstico voltados para ECG

Figura 3.5 exibe a classe `ECG_Analysis` da aplicação *Blueheart*, integrado com exemplos de classes para diagnosticar cardiopatias. Além disso, mostra que as classes `MyocardialInfarction`, `HeartFailure` e `HeartArrhythmia` herdam da classe abstrata para diagnóstico de doenças denominada `Diagnosis`. Desta forma, o método booleano `diseaseDiagnosis` apresentará uma implementação diferente em cada uma das classes, que possuem o objetivo de diagnosticar as respectivas doenças: ataque cardíaco, insuficiência cardíaca e arritmia cardíaca. Essa estrutura permanecerá a mesma independentemente do número de classes que herdem da classe `Diagnosis`. Estritamente falando, o *framework* permite que vários algoritmos de diagnóstico obtenham as mesmas medidas e dados processados como entrada a partir de um mesmo `Analyzer`, sem a necessidade de fazer novas medidas para cada algoritmo a ser executado.

Não obstante, algoritmos de diagnóstico podem ser integrados com os seus respectivos analisadores de sinais biológicos, sem a necessidade de ter que fazer todos os cálculos do zero. Ou seja, os algoritmos simplesmente vão reutilizar os cálculos feitos previamente pela instância de uma classe que herdou da classe `Analyzer`, para entregar

todos os diagnósticos de acordo.

Similarmente, para a integração do *framework* com aplicações que analisem outros tipos de sinais biológicos, é necessário seguir o mesmo procedimento. Este procedimento consiste em importar a biblioteca do *framework*, que contém o conjunto de classes abstratas definidas de acordo com a Figura 3.1. Desta forma, a aplicação a ser desenvolvida pode definir classes concretas que implementem cada uma de suas versões dos métodos abstratos definidos no *framework*, de acordo com o tipo de sinal biológico a ser tratado. Também é possível fazer uso parcial do *framework*, como é o caso da aplicação *BlueHeart*, que apenas utiliza as classes `DataStorage`, `Analyzer` e `SignalPlot`. O modelo do *framework* também leva em consideração que, da mesma forma como é possível ter vários tipos de diagnóstico de doenças relacionadas a um tipo de sinal biológico, também é possível ter vários tipos de filtros diferentes para tratar o sinal bruto adquirido.

Para cada tipo de sinal biológico, existe uma faixa diferente de características e particularidades que podem ser analisadas. Sendo assim, não é possível tratar diferentes sinais biológicos da mesma forma. Por exemplo, para a detecção de doenças cardíacas, é relevante saber informações das ondas Q, R e S (complexo QRS), que ocorrem continuamente em um sinal ECG. Mas isso não ocorre com outro sinal biológico que não seja o ECG. Por isso, a classe `Analyzer` tem um método abstrato genérico denominado `extractFeatures` que, no caso da aplicação *BlueHeart*, faz chamada a outros métodos específicos que extraem características de sinal ECG, alimentando o vetor de características. O mesmo ocorre com o método `initializeInterface` da classe `SignalPlot` que, na aplicação *BlueHeart*, é a classe `Tracking`. A forma de plotar um sinal ECG é diferente da forma de plotar um sinal EEG (eletroencefalografia). Esse padrão pode ser visto no modelo do *framework*: os métodos e atributos generalizados são aqueles que tem um comportamento comum, independentemente do sinal biológico a ser analisado. Se forem inseridos atributos específicos apenas de um ou alguns sinais biológicos no modelo do *framework*, o mesmo deixará de atender a proposta de servir como *framework* para quaisquer sinais biológicos.

3.5 Considerações finais

Dentre as vantagens para a utilização de *frameworks*, estão economia de tempo, escalabilidade e reutilização de recursos. A economia de tempo ocorre porque o uso de *frameworks* evita o dispêndio financeiro e intelectual, uma vez que providenciam

estruturas e recursos definidos de antemão para um domínio específico do conhecimento. A escalabilidade também é uma característica intrínseca do uso de *frameworks*, pois a estrutura de um *framework* irá permanecer a mesma, independentemente do quanto um projeto que o implemente cresça. O fato da projeção de *frameworks* permitir implementações abstratas (como classes abstratas e interfaces), é possível reutilizar sua estrutura para desenvolver diversas implementações concretas diferentes entre si, a partir de um mesmo conceito.

O modelo de *framework* proposto neste trabalho tem o objetivo de prover uma arquitetura que facilite o desenvolvimento de aplicações com foco em aquisição e tratamento de sinais biológicos. O modelo representa a abstração das operações básicas de tratamento de sinais biológicos que estão presentes, por exemplo, em eletrocardiografia, eletroencefalografia, eletromiografia, entre outros. Eletrocardiografia foi a área escolhida para testar o modelo. Desta forma, utiliza-se a aplicação *BlueHeart*, que implementa uma parte do *framework*, para realizar os testes que são descritos no próximo capítulo.

Capítulo 4

Metodologia

Para assegurar que o *framework* funciona da forma para a qual ele foi projetado, é necessário levar em consideração os seguintes quesitos:

- A possibilidade de integrar o *framework* com algoritmos de diagnóstico de cardiopatias, o que implica no devido processamento dos sinais de ECGs captados, como o complexo QRS;
- A correta saída dos dados processados, ou seja, o armazenamento no formato que assegura interoperabilidade com outros eletrocardiógrafos e sistemas compatíveis com o formato aECG HL7.

Desta forma, de acordo com os quesitos mencionados, é possível recorrer aos testes que façam essas verificações. Para isso, dois testes foram feitos com o uso de duas aplicações terceiras, uma para avaliar os sinais ECGs captados e a outra para verificar a integridade do arquivo aECG HL7 gerado. Os testes e as aplicações utilizadas serão detalhados nas próximas seções.

4.1 A ferramenta *C# ECG Toolkit*

Em Nelwan et al. (2005), foi proposto um *framework* para integrar múltiplas bases de dados para ECG. Para isso, foi desenvolvido uma ferramenta em C# e *DOT NET* para criar um visualizador capaz de exibir sinais ECG e fazer conversões entre diferentes formatos de dados. Esta proposta gerou um outro trabalho (van Ettinger et al.,

2008), no qual outros formatos foram incorporados, entre eles o SCP-ECG (*Standard Communications Protocol for Computer Assisted Electrocardiography*), o aECG HL7 e o DICOM-ECG (*Digital Imaging and Communications in Medicine*). A partir desses trabalhos, surgiu a ferramenta *C# ECG Toolkit*[1], que é um *software* de código aberto para converter, visualizar e imprimir eletrocardiogramas. Possui suporte para os formatos SCP-ECG, aECG HL7, DICOM, entre outros.

A Figura 4.1 exibe a interface do *C# ECG Toolkit*, plotando um sinal ECG lido de um arquivo no formato aECG HL7. É válido ressaltar que caso o arquivo esteja inconsistente com o formato de entrada escolhido, um erro é gerado e o sinal não é plotado.

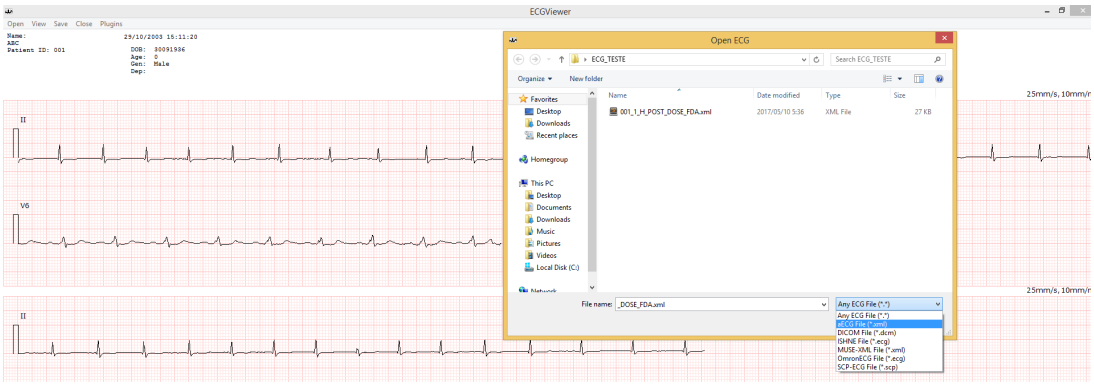


Figura 4.1: Interface do visualizador do *ECG Toolkit*, plotando um sinal ECG a partir de um arquivo de entrada

O teste que utiliza o *ECG Toolkit* consiste em gerar um sinal ECG, processá-lo com a aplicação padrão do *framework* proposto, *BlueHeart*, extraindo as características do sinal e gerando o respectivo arquivo no formato aECG HL7. Em seguida, utilizar este arquivo como entrada para o *ECG Toolkit*. Desta forma, é possível observar se erros de formatação do arquivo serão gerados e se o sinal característico plotado pela ferramenta condiz com o sinal da aplicação.

[1] https://sourceforge.net/projects/ecgtoolkit-cs/

4.2 A Aplicação *FDAECGSuite*

O *FDAECGSuite*[2] é um visualizador para ECGs no formato XML aECG HL7 (Brown e Badilini, 2005), desenvolvido pela AMPS llc (*Analyzing Medical Parameters for Solutions*), que é uma companhia que desenvolve soluções para análises de eletrocardiogramas. O *FDAECGSuite* é um *software* que providencia uma forma para mostrar, revisar, pontuar e validar arquivos ECG no formato apoiado pela *Food and Drug Administration* dos Estados Unidos. Nesta dissertação, utilizou-se a versão de demonstração do *software*, que é disponível gratuitamente.

A Figura 4.2 mostra uma captura de tela da interface do *FDAECGSuite* versão demonstração, onde é possível ver a plotagem de um sinal ECG de exemplo que vem anexado com a aplicação.

Figura 4.2: Interface do *FDAECGSuite* demo com plotagem de sinal ECG do exemplo anexo com a aplicação

4.2.1 Sistema de Validação do *FDAECGSuite*

O *FDAECGSuite* providencia visualização do ECG para dados brutos e médias de formas de onda, além de incluir a validação e módulos de pontuação. O módulo

[2] http://www.amps-llc.com/website/index.php?option=com_rokdownloads&view=file&Itemid=60&id=17:fdaecgsuite-demo

de validação verifica se a estrutura e o conteúdo dos arquivos XML HL7 atendem aos requisitos descritos no Guia de Implementação HL7 e Esquema HL7 (Brown e Badilini, 2005) endorsado pela FDA.

A Figura 4.3 mostra a validação do *FDAECGSuite* demo para o exemplo anexo com a aplicação.

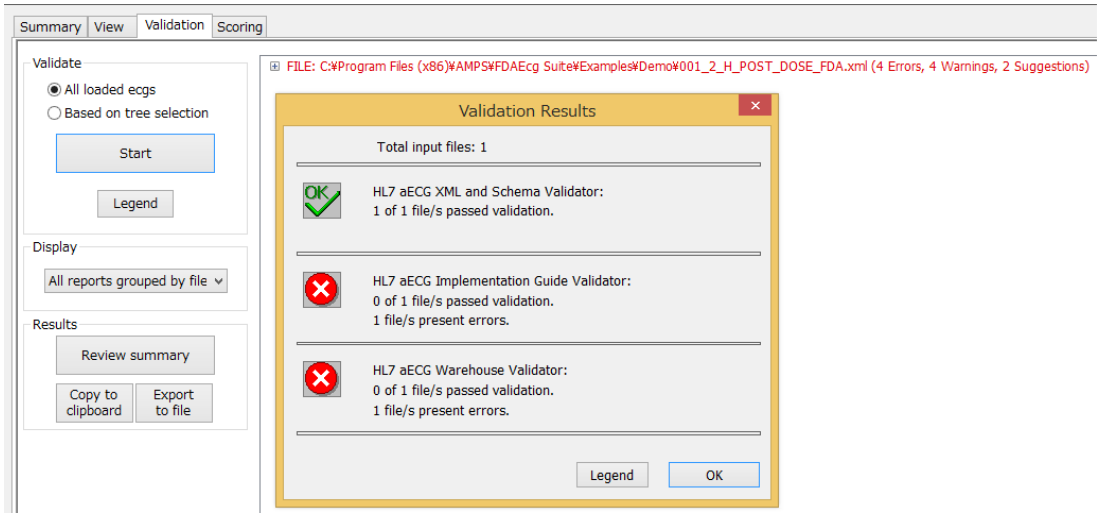


Figura 4.3: Sistema de avaliação de arquivo aECG HL7 do *FDAECGSuite* demo do exemplo anexo da aplicação

A validação de arquivo realizada pelo *FDAECGSuite* possui três critérios. O primeiro é o *HL7 aECG XML and Scheme Validator*, que é a validação que checa se todos os elementos utilizados na construção do arquivo estão em conformidade com as regras de sintaxe do XML e com o esquema do HL7. O segundo é o *HL7 aECG Implementation Guide Validator*, que também faz a checagem dos elementos utilizados na construção do arquivo, verificando se os mesmos estão em conformidade com as regras de sintaxe do XML e do esquema HL7, levando em conta o guia de implementação do padrão aECG HL7. O terceiro é o *HL7 aECG Warehouse Validator*, que é a validação que lida com questões relacionadas com a submissão de arquivos aECG para o depósito da FDA (*Food and Drug Administration*), mas que não são especificados no nível de esquema (ou até mesmo no nível do guia de implementação do padrão aECG HL7), tais como o preenchimento de informações clínicas no arquivo (número identificador de paciente ou ID, por exemplo).

Verifica-se que a validação de arquivo do *FDAECGSuite* retorna resultado positivo

para o *HL7 aECG XML and Scheme Validator* e resultado negativo para o *HL7 aECG Implementation Guide Validator* e o *HL7 aECG Warehouse Validator*. A causa dos resultados negativos são que algumas *tags* do arquivo estão vazias, como a *codeSystem*, por exemplo. É válido constatar que esses resultados condizem com os diversos exemplos fornecidos com a aplicação. Sendo assim, considera-se neste trabalho que o sistema de validação a ser utilizado como critério para avaliar o arquivo gerado pelo *framework* seja apenas o *HL7 aECG Implementation Guide Validator*, que faz a checagem das *tags* que são essenciais (não opcionais) para a validação do arquivo.

4.3 Considerações finais

Através da metodologia apresentada, é possível analisar aplicações para sistemas eletrocardiográficos que sejam desenvolvidas a partir do *framework* proposto. Com o intuito de assegurar interoperabilidade com grande parte dos eletrocardiógrafos, é necessário garantir que o formato de arquivo de dados processados esteja consistente com o padrão aECG HL7. As ferramentas *C# ECG Toolkit* e *FDAECGSuite* foram selecionadas para testar o arquivo de dados gerado pela aplicação *BlueHeart*. Os experimentos e resultados serão apresentados no próximo capítulo.

Capítulo 5

Experimentos e Resultados

Os testes experimentais foram conduzidos utilizando a aplicação *BlueHeart*, que é a aplicação padrão do *framework*. Os critérios a serem analisados são os mencionados no Capítulo 4. Primeiramente, verificar se o processamento dos sinais de ECGs captados estão sendo processados devidamente, utilizando-se a aplicação *C# ECG Toolkit* para receber como entrada o arquivo aECG HL7 gerado pela aplicação *BlueHeart*. Desta forma, é possível verificar se os dados processados geram a plotagem característica do sinal. O outro teste consiste em analisar sintaticamente o arquivo aECG HL7 gerado pela aplicação, certificando-se de que o mesmo atende aos requisitos descritos no guia de implementação do padrão (Brown e Badilini, 2005). Com estes propósitos, as aplicações *C# ECG Toolkit* e *FDAECGSuite* serão utilizadas para as análises.

5.1 Teste e resultado com o *C# ECG Toolkit*

Neste teste, o sinal ECG foi gerado com *ProSim 8 Vital Signs Patient Monitor Simulator*[1], que é um equipamento eletrônico utilizado para simular sinais vitais. A taxa de amostragem do sinal foi de 192 Hz, captado pela aplicação *BlueHeart* instalada em um *smartphone Samsung Galaxy J7 Duos*, com 1.5 GB de RAM, 16 GB de memória interna e processador *Exynos* de 1.5 GHz.

Com o sinal adquirido e seu respectivo arquivo aECG HL7 gerado, utilizou-se o *ECG Toolkit* para receber como entrada o arquivo gerado. A Figura 5.1 exibe o arquivo

[1] http://www.flukebiomedical.com/biomedical/usen/Patient-Simulators/ProSim-8-vital-signs-patient-simulator.htm?PID=72624

aECG HL7 gerado e sua respectiva plotagem do sinal adquirido.

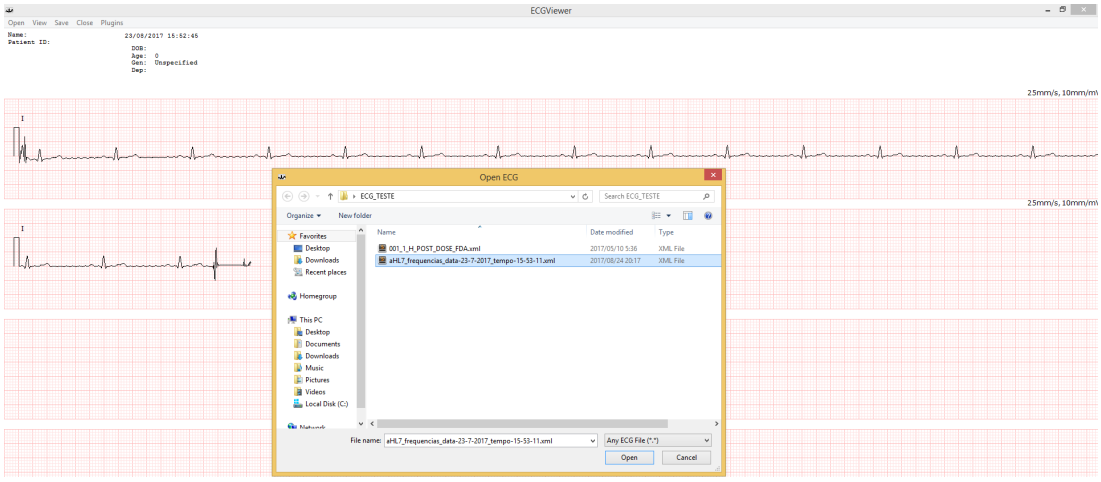


Figura 5.1: Captura de tela da aplicação *C# ECG Toolkit* plotando o sinal do arquivo gerado pelo *BlueHeart*

A Figura 5.2 exibe o mesmo sinal da Figura 5.1, mas com *zoom* na plotagem do sinal para facilitar a visualização.

Figura 5.2: Captura de tela com *zoom* da aplicação *C# ECG Toolkit* plotando o sinal do arquivo gerado pelo *BlueHeart*

O objetivo deste teste foi observar se o *framework* proporciona a devida estrutura para aplicações que o implementem, de tal forma que as mesmas possam ser integradas com aplicações que interoperem em um formato comum. Desta forma, a aplicação

BlueHeart poderá ser integrada com algoritmos de detecção automática de cardiopatias futuramente. Caso houvesse comprometimento dos dados ou falhas na devida construção do arquivo aECG HL7, esses problemas seriam detectados na plotagem do sinal (dados comprometidos) ou erro na leitura do arquivo a partir do *ECG Toolkit* (falha na construção adequada do arquivo). Nenhum desses problemas ocorreu.

5.2 Teste e resultado com o *FDAECGSuite*

Um outro teste foi conduzido para verificar se o arquivo aECG HL7 gerado pela aplicação *BlueHeart* atende ao formato de troca de sinais ECG endorsado pela FDA (*Food and Drug Administration*).

Neste teste, o sinal ECG foi gerado com *ECGSyn* (McSharry et al., 2003), que é um *software* que simula os sinais vitais do coração, rodando no MATLAB com taxa de amostragem de 192 Hz, plotado e o arquivo armazenado na memória do *smartphone* no formato aECG HL7 para análises futuras. A aplicação e o sinal são mostrados na Figura 5.3. É uma amostragem de 4 segundos, onde o incremento para cada medida (ponto) no sinal é de 0.005208333 segundos.

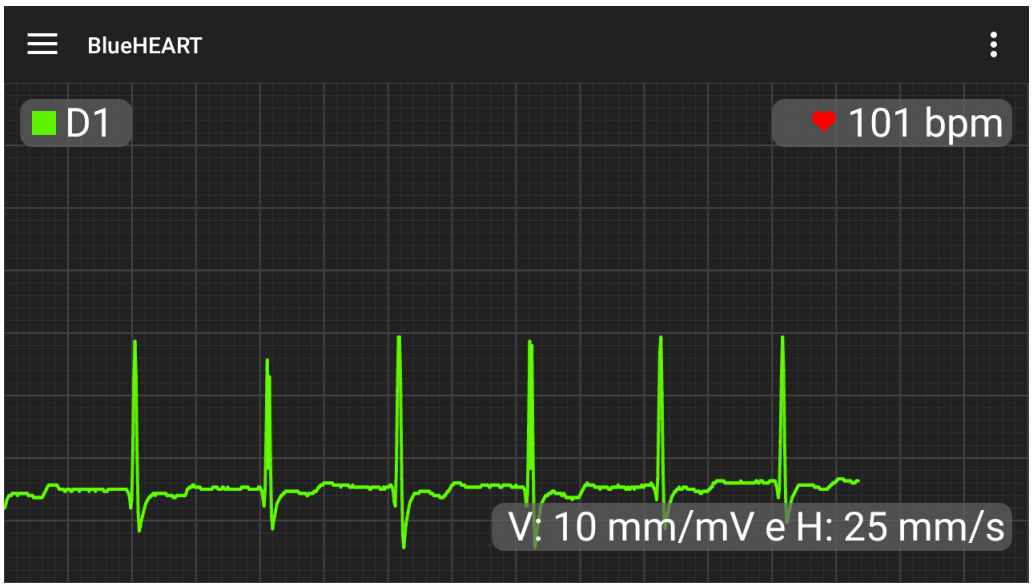


Figura 5.3: Captura de tela da aplicação *BlueHeart*

Assim como especificado no Capítulo 3, a aplicação *BlueHeart* possui três classes que herdam das classes do *framework*: `Tracking` (herda de `SignalPlot`), `ECG_-`

`Analysis` (`Analyzer`) e `Save_Data_aECGHL7` (`DataStorage`). A grade e os elementos de interface da aplicação, como mostrados na Figura 5.3, foram gerados pelo método `InitializeInterface` da classe `Tracking`. A plotagem do sinal ECG (em verde) foi gerada pelo método `plotSignal` da mesma classe. O arquivo gerado da extração de características do sinal, realizada pelo método `extractFeatures` da classe `ECG_Analysis`, pode ser visto no Apêndice A. O método utilizado para gerar esse arquivo no formato aECG HL7 foi o `saveData` da classe `Save_Data_aECGHL7`.

O arquivo XML aECG HL7 é armazenado no diretório *Documents* do *smartphone*. O arquivo aECG HL7 gerado é compatível com a aplicação *FDAECGSuite* demo da AMPS llc. O resultado da validação pode ser visto na Figura 5.4. O *FDAECGSuite* demo, que é uma aplicação *desktop* que serve como um visualizador para arquivos no formato XML HL7, foi desenvolvido para validar e providenciar módulos de pontuação para os sinais ECG no formato XML HL7, certificando que a estrutura e o conteúdo dos arquivos XML HL7 atendam aos requisitos descritos no guia de implementação e esquema do aECG HL7 (Brown e Badilini, 2005), endorsado pela FDA. O *smartphone* utilizado para realizar os testes foi um *Samsung Galaxy J7 Duos*, com 1.5 GB de RAM, 16 GB de memória interna e processador *Exynos* de 1.5 GHz.

5.2.1 Resultado da Validação do *FDAECGSuite* do arquivo aECG HL7 gerado

Figura 5.4 exibe a saída do sistema de validação da aplicação *FDAECGSuite* demo, mostrando a análise da sintaxe do arquivo XML gerado pelo *framework*. Os mesmos resultados de validação foram realizados para os arquivos anexos distribuídos juntamente com a aplicação. O ícone verde com a mensagem de OK refere-se à aprovação vinda do validador de esquema XML aECG HL7 sobre o arquivo XML gerado, ou seja, a análise da sintaxe do arquivo aECG HL7 gerado. Os ícones vermelho são respectivamente o validador do guia de implementação do aECG HL7 e o validador *Warehouse*. Eles apontam que algumas *tags* no arquivo XML estão vazias e que outras utilizam um código desconhecido de sistema. Esses erros de sintaxe são consistentes com os arquivos anexados de exemplo distribuídos com a aplicação. Enquanto os resultados forem consistentemente aprovados pelo validador de esquema, apresentando a mesma saída de validação tanto para os arquivos gerados pelo *framework* como para os exemplos anexados com a aplicação, conclui-se que o sistema de validação retorna resultado positivo para o arquivo gerado pelo *framework*. Ou seja, significa que os arquivos ge-

rados pelo *framework* passam na validação. Com exceção do conteúdo na *tag* dígitos, todo o conteúdo do arquivo XML foi gerado e preenchido pelo método automático do *framework* para salvar os dados no formato aECG HL7. Essa restrição foi aplicada, pois a versão demonstração do *FDAECGSuite* não permite customizar o conteúdo da *tag* dígitos. De outro modo, todos os requisitos foram atendidos de forma bem sucedida.

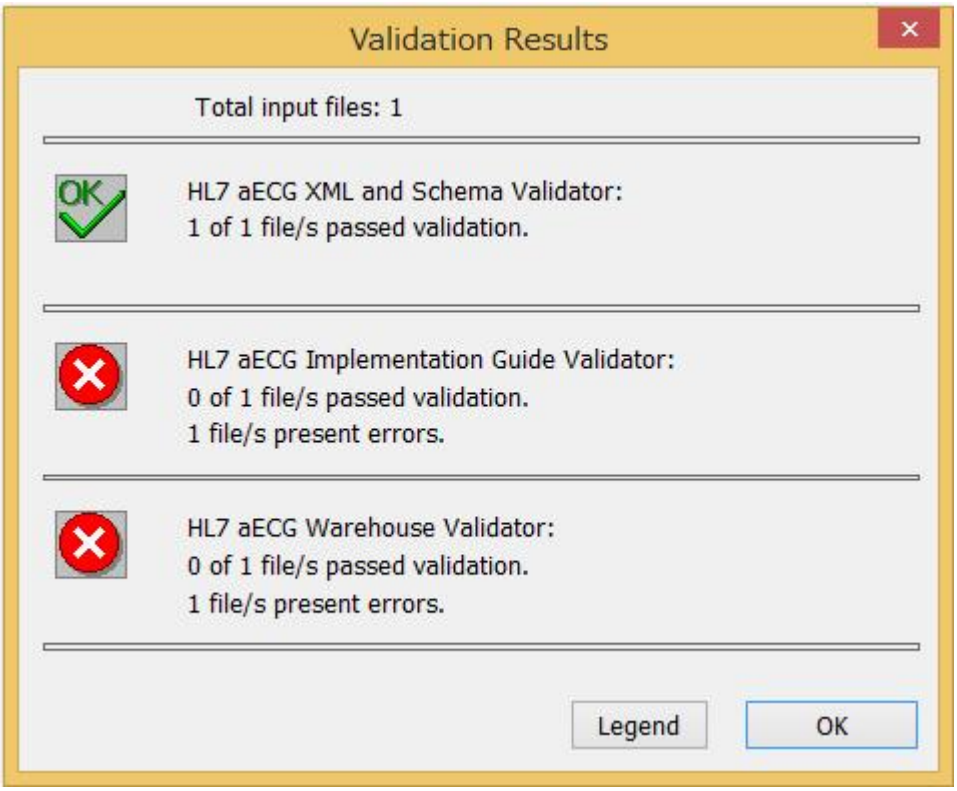


Figura 5.4: Validação do *FDAECGSuite* sobre o arquivo XML aECG HL7 gerado pela aplicação *BlueHeart*

O arquivo XML aECG HL7 gerado pelo *framework* com os dados brutos e complexos QRS detectados, cuja plotagem do sinal ECG é mostrada na Figura 5.3, pode ser visto no Apêndice A. A forma de armazenagem dos complexos QRS no arquivo pode ser vista no Anexo II.1.

5.3 Considerações finais

Os resultados obtidos com os testes feitos nas ferramentas *C# ECG Toolkit* e *FDAECGSuite* a partir do arquivo gerado pela aplicação *BlueHeart* são satisfatórios,

pois demonstram que uma aplicação desenvolvida sobre o *framework* proposto consegue prover interoperabilidade para os demais eletrocardiógrafos que sejam capazes de lidar com o padrão aECG HL7. Além disso, o arquivo gerado pela aplicação *BlueHeart* contém os complexos QRS detectados, que são essenciais para o desenvolvimento de aplicações com foco em detecção de cardiopatias.

Capítulo 6

Conclusão

A integração de sistemas *mobile* com soluções de saúde é uma tendência que vem sendo largamente explorada atualmente. Existem muitas pesquisas focando em como tirar o maior proveito de *hardware* customizável e de baixo custo, integrando-o com serviços na nuvem e procurando por padronização para interoperabilidade entre os vários tipos de sistemas de informação de saúde. A faixa de tecnologias para adquirir e processar sinal ECG engloba vários tipos de tecnologias. Não obstante, a vantagem de recorrer aos *smartphones* é compreensível do ponto de vista de custo e benefício, já que esses dispositivos são relativamente fáceis de se programar, testar e distribuir.

Levando isso em consideração, um *framework* baseado na plataforma Android para análise de sinais ECG com o objetivo de prover diagnóstico automático de doenças cardíacas foi proposto neste trabalho. O *framework* apresentado mostrou-se flexível e escalável. Em outras palavras, sua estrutura não é rígida e pode ser facilmente estendida para incorporar novos algoritmos para detectar cardiopatias. A modelagem proposta foi integrada e testada com o desenvolvimento da aplicação *BlueHeart*. A forma de armazenagem dos dados da aplicação *BlueHeart*, com o intuito de assegurar interoperabilidade com demais eletrocardiógrafos, também foi testada e comprovada com os métodos que geram os respectivos arquivos no formato aECG HL7, após o tratamento dos sinais.

A arquitetura do *framework* também suporta as características do Android para fazer operações em segundo plano, como os recursos das classes para operações em segundo plano (*background services*) e tarefas assíncronas (*asynchronous task*). Essas características definem o *framework* como uma ferramenta para servir de referência para o desenvolvimento, integração e teste de uma grande gama de algoritmos, em

um sistema de tempo real, diferentemente de aplicações executadas em MATLAB com dados pré-gravados. Portanto, é possível analizar o desempenho e o comportamento de novos algoritmos diretamente do sistema em tempo real, verificando se eles podem entregar uma resposta em um espaço de tempo aceitável, dentro de um sistema escalável, para que seja possível suportar diagnósticos em tempo real.

Por fim, os *middlewares* desenvolvidos e propostos para aquisição de sinal ECG ainda não providenciam um *framework* que combine Android com a geração automática dos respectivos arquivos aECG HL7 das análises oriundas de sinais ECG. Portanto, o *framework* proposto ajuda a preencher esta lacuna e irá servir também como uma alternativa para pesquisadores mais interessados em desenvolvimento de algoritmos em alto nível ao invés de se preocuparem com desenvolvimento de baixo nível. Em outras palavras, desenvolvedores podem focar no desenvolvimento de algoritmos de diagnóstico de doenças, ao invés de trabalhar em operações básicas de aquisição de sinal ECG.

6.1 para o *framework* com foco em eletrocardiografia

Em relação à utilização do *framework* para construção de aplicações voltadas ao domínio da análise de sinais ECG, como no caso da aplicação *BlueHeart*, as sugestões para trabalhos futuros são:

- Adicionar métodos para detectar as ondas P, pois é possível detectar doenças a partir do diagnóstico da mesma. Um exemplo de diagnóstico de doença envolvendo a onda P é o alargamento do átrio direito, causado por hipertensão pulmonar (Cadogan, 2017);
- Integrar os dados das ondas P no método do *framework* que gera o arquivo aECG HL7, com seus respectivos *timestamps*. O anexo II.2, extraído do guia de implementação do padrão de Brown e Badilini (2005), demonstra como construir o arquivo para o correto armazenamento das anotações da onda P no formato aECG HL7;
- Adicionar métodos para detectar as ondas T, pois é possível detectar doenças a partir do diagnóstico da mesma. Isquemia e doença da artéria coronária (que fornece sangue para o coração) estão entre os exemplos de diagnósticos de doenças que podem ser feitas a partir da onda T (Burns, 2017);

- Integrar os dados das ondas T no método do *framework* que gera o arquivo aECG HL7, com seus respectivos *timestamps*. O anexo II.3, extraído do guia de implementação do padrão Brown e Badilini (2005), demonstra como construir o arquivo para o correto armazenamento das anotações da onda T no formato aECG HL7;

- Adicionar métodos como padrões para detecção das cardiopatias mais comuns, como arritmia cardíaca, por exemplo;

- Testar e tornar o *framework* compatível com versões anteriores (a 4) e futuras (acima da 6) do Android;

- Adicionar modificação ao algoritmo DOM, proposto em Nishida et al. (2017) (publicado em IEEE MEMEA 2017 mas ainda não indexado), para maior robustez do processamento e identificação do complexo QRS em sinais poluídos com ruído de rede (50 ou 60 Hz) e randômico.

Referências Bibliográficas

AIRES. *Fisiologia*. Grupo Gen - Guanabara Koogan, Rio de Janeiro, 2012. ISBN 9788527721004. (acesso em: Setembro/2016).

ALENAZI, T. M. e ALHAMED, A. A. A middleware to support hl7 standards for the integration between healthcare applications. *2015 International Conference on Healthcare Informatics*, pages 509–512, Oct 2015. DOI: 10.1109/ICHI.2015.93. URL http://ieeexplore.ieee.org/document/7349755/. (acesso em: Novembro/2016).

ARISTOMENOPOULOS, G., FONTANA, R., VATTERONI, M., e TORTORA, G. Remote management of left ventricular device assisted patients. *2014 4th International Conference on Wireless Mobile Communication and Healthcare - Transforming Healthcare Through Innovations in Mobile and Wireless Technologies (MOBIHEALTH)*, pages 59–62, Nov 2014. DOI: 10.1109/MOBIHEALTH.2014.7015909. URL http://ieeexplore.ieee.org/document/7015909/. (acesso em: Novembro/2016).

BOND, R. R., FINLAY, D. D., NUGENT, C. D., e MOORE, G. A review of {ECG} storage formats. *International Journal of Medical Informatics*, 80(10):681 – 697, 2011. ISSN 1386-5056. DOI: 10.1016/j.ijmedinf.2011.06.008. URL http://www.sciencedirect.com/science/article/pii/S1386505611001304. (acesso em: Outubro/2016).

BROWN, B. D. e BADILINI, F. HL7 aECG implementation guide, 2005. URL http://www.amps-llc.com/website/documents/UsefulDocs/aECG_Implementation_Guide.pdf. (acesso em: Setembro/2016).

BURNS, E. T wave, 2017. URL https://lifeinthefastlane.com/ecg-library/basics/t-wave/. (acesso em: Junho/2017).

CADOGAN, M. P wave, 2017. URL https://lifeinthefastlane.com/ecg-library/basics/p-wave/. (acesso em: Junho/2017).

COREPOINT HEALTH. The hl7 evolution - comparing hl7 version 2 to version 3, including a history of version 2, 2010. URL https://corepointhealth.com/resource-center/white-papers/evolution-hl7. (acesso em: Janeiro/2017).

COVELLO, R., FORTINO, G., GRAVINA, R., AGUILAR, A., e BRESLIN, J. G. Novel method and real-time system for detecting the cardiac defense response based on the ecg. *2013 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, pages 53–57, May 2013. DOI: 10.1109/MeMeA.2013.6549705. URL http://ieeexplore.ieee.org/document/6549705/. (acesso em: Novembro/2016).

DENG, H. e CHEN, S. Design and implementation of android-based health and healthcare system. *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 164–169, Sept 2016. DOI: 10.1109/IWCMC.2016.7577051. URL http://ieeexplore.ieee.org/document/7577051/. (acesso em: Novembro/2016).

DEPARI, A., FLAMMINI, A., SISINNI, E., e VEZZOLI, A. A wearable smartphone-based system for electrocardiogram acquisition. *2014 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, pages 1–6, June 2014. DOI: 10.1109/MeMeA.2014.6860030. URL http://ieeexplore.ieee.org/document/6860030/. (acesso em: Novembro/2016).

KANG, K., KWON, Y., KIM, Y., LEE, J., e BAE, C. Lifelog collaboration framework for healthcare service on android platform. *International Conference on ICT for Smart Society*, pages 1–4, June 2013. DOI: 10.1109/ICTSS.2013.6588078. URL http://ieeexplore.ieee.org/document/6588078/. (acesso em: Novembro/2016).

LI, H. e WANG, X. Detection of electrocardiogram characteristic points using lifting wavelet transform and hilbert transform. *Transactions of the Institute of Measurement and Control*, 35(5):574–582, 2013. DOI: 10.1177/0142331212460720. URL http://dx.doi.org/10.1177/0142331212460720. (acesso em: Novembro/2016).

LIU, L. e HUANG, Q. An extensible hl7 middleware for heterogeneous healthcare information exchange. *2012 5th International Conference on BioMedical Engineering and Informatics*, pages 1045–1048, Oct 2012. DOI: 10.1109/BMEI.2012.6513196. URL http://ieeexplore.ieee.org/document/6513196/. (acesso em: Novembro/2016).

MCSHARRY, P., CLIFFORD, G., TARASSENKO, L., e SMITH, L. A dynamical model for generating synthetic electrocardiogram signals. *IEEE Transactions on Biomedical*

Engineering, 50(3):289–294, 2003. DOI: 10.1109/TBME.2003.808805. URL http://ieeexplore.ieee.org/document/1186732/.

MORRIS, F. e BRADY, W. J. Abc of clinical electrocardiography: Acute myocardial infarction—part i. *BMJ*, 324(7341):831–834, 2002. ISSN 0959-8138. DOI: 10.1136/bmj.324.7341.831. URL http://www.bmj.com/content/324/7341/831. (acesso em: Setembro/2016).

NELWAN, S. P., VAN ETTINGER, M., DE WIJS, M. C. J., e MEIJ, S. H. Integration of multiple ecg databases into a unified framework. In *Computers in Cardiology, 2005*, pages 447–450, Sept 2005. DOI: 10.1109/CIC.2005.1588133. (acesso em: Agosto/2017).

NISHIDA, E. N., DUTRA, O. O., FERREIRA, L. H. C., e COLLETTA, G. D. Application of Savitzky-Golay digital differentiatorfor QRS complex detection in anElectrocardiographic monitoring system. 2017.

OPENEINTHOVEN. OpenEinthoven platform. URL http://einthoven.unifei.edu.br. (acesso em: Setembro/2016).

PAN, J. e TOMPKINS, W. J. A real-time QRS detection algorithm. *IEEE Trans Biomed Eng*, 32(3):230–236, March 1985. ISSN 0018-9294. DOI: 10.1109/TBME.1985.325532. URL http://view.ncbi.nlm.nih.gov/pubmed/3997178. (acesso em: Novembro/2016).

RIEHLE, D. Framework design a role modeling approach. Master's thesis, Swiss Federal Institute of Tecnology Zurich, 2000. (acesso em: Setembro/2017).

RIEHLE, D. e DUBACH, E. Working with java interfaces and classes. 1999. (acesso em: Setembro/2017).

SIERRA, D. F., TORRES, Y. F., e CAMARGO, J. E. A secure mobile system to interchange electronic medical records in hl7. *2014 9th Computing Colombian Conference (9CCC)*, pages 213–221, Sept 2014. DOI: 10.1109/ColumbianCC.2014.6955356. URL http://ieeexplore.ieee.org/document/6955356/. (acesso em: Novembro/2016).

SILVA, B. M., RODRIGUES, J. J., DE LA TORRE DÍEZ, I., LÓPEZ-CORONADO, M., e SALEEM, K. Mobile-health: A review of current state in 2015. *Journal of Biomedical Informatics*, 56:265 – 272, 2015. ISSN 1532-0464. DOI: 10.1016/j.jbi.2015.06.003. URL http://www.sciencedirect.com/science/article/pii/S1532046415001136. (acesso em: Outubro/2016).

SPRONK, R. Hl7 message examples: version 2 and version 3, 1999. URL http://www.ringholm.com/docs/04300_en.htm. (acesso em: Janeiro/2017).

TORRES, O. H. M. e RICAURTE, J. A. B. Application middleware for management of medical applications based on hl7 standards. *2015 Asia-Pacific Conference on Computer Aided System Engineering*, pages 19–23, July 2015. DOI: 10.1109/APCASE.2015.11. URL http://ieeexplore.ieee.org/document/7286987/. (acesso em: Novembro/2016).

TRIGO, J. D., EGUZKIZA, A., MARTÍNEZ-ESPRONCEDA, M., e SERRANO, L. A cardiovascular patient follow-up system using twitter and hl7. *Computing in Cardiology 2013*, pages 33–36, Sept 2013. ISSN 0276-6574. URL http://ieeexplore.ieee.org/abstract/document/6712404/. (acesso em: Novembro/2016).

VAN ETTINGER, M. J. B., LIPTON, J. A., DE WIJS, M. C. J., VAN DER PUTTEN, N., e NELWAN, S. P. An open source ecg toolkit with dicom. In *2008 Computers in Cardiology*, pages 441–444, Sept 2008. DOI: 10.1109/CIC.2008.4749073. (acesso em: Agosto/2017).

XIA, H., ASIF, I., e ZHAO, X. Cloud-ecg for real time {ECG} monitoring and analysis. *Computer Methods and Programs in Biomedicine*, 110(3):253 – 259, 2013. ISSN 0169-2607. DOI: 10.1016/j.cmpb.2012.11.008. URL http://www.sciencedirect.com/science/article/pii/S0169260712003069. (acesso em: Outubro/2016).

YEH, Y.-C. e WANG, W.-J. {QRS} complexes detection for {ECG} signal: The difference operation method. *Computer Methods and Programs in Biomedicine*, 91 (3):245 – 254, 2008. ISSN 0169-2607. DOI: 10.1016/j.cmpb.2008.04.006. URL http://www.sciencedirect.com/science/article/pii/S0169260708001004. (acesso em: Novembro/2016).

APÊNDICES

Apêndice A

Arquivo XML aECG HL7 gerado pelo framework

Listagem A.1: Arquivo gerado pelo framework no formato aECG HL7

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<AnnotatedECG xmlns="urn:hl7-org:v3" xmlns:voc="urn:hl7-org:v3/voc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:hl7-org:v3 /HL7/aECG/2003-12/schema/
    PORT_MT020001.xsd" type="Observation" classCode="OBS">
  <id root="2.16.840.1.114396.1.110673.5.2014131" />
  <code code="93000" codeSystem="2.16.840.1.113883.6.12"
      codeSystemName="CPT-4" displayName="Electrocardiogram" />
  <effectiveTime>
      <low value="20170611041002.013" inclusive="true" />
      <high value="20170611041007.015" inclusive="false" />
  </effectiveTime>
  <component>
    <series>
      <id root="2.16.840.1.114396.1.110673.5.2014131" />
      <code code="RHYTHM" codeSystem="2.16.840.1.113883.5.4"
          codeSystemName="ActCode" displayName="ECG Rhythm Waveforms"
          />
      <effectiveTime>
        <low value="20170611041002.013" inclusive="true" />
        <high value="20170611041007.015" inclusive="false" />
      </effectiveTime>
      <component>
        <sequenceSet>
```

```
19              <component>
20                <sequence>
21                  <code code="TIME_ABSOLUTE" codeSystem="
                        2.16.840.1.113883.5.4" codeSystemName="ActCode"
                        displayName="Absolute Time" />
22                  <value xsi:type="GLIST_TS">
23                    <head value="20170611041002.013" />
24                    <increment value="0.0055556" unit="s" />
25                  </value>
26                </sequence>
27              </component>
28              <component>
29                <sequence>
30                  <code code="MDC_ECG_LEAD_I" codeSystem="
                        2.16.840.1.113883.6.24" codeSystemName="MDC" />
31                  <value xsi:type="SLIST_PQ">
32                    <origin value="0.000" unit="uV" />
33                    <scale value="6.250" unit="uV" />
34                    <digits>44 46 48 49 50 50 50 51 51 51 51 51 50 51 51
                          51 51 51 51 51 51 51 50 50 50 50 50 49 49 49 49 49
                           49 49 49 49 50 51 52 53 54 55 55 55 55 55 55 55
                          55 55 54 53 53 53 53 53 52 52 53 53 53 53 52 52 53
                           53 53 53 53 53 52 53 53 53 53 53 53 53 53 52 53
                          53 53 53 53 52 53 53 53 53 53 53 53 52 52 52 52 53
                           53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53
                          53 52 51 49 44 48 72 105 124 107 69 42 33 35 38 40
                           42 44 46 47 48 49 49 50 50 51 51 50 51 51 51 51
                          51 51 51 51 51 51 50 50 50 50 50 50 49 49 49 50 50
                           50 50 50 50 50 50 50 51 53 53 54 54 54 55 55 54
                          54 54 54 54 54 53 53 53 53 53 53 53 53 53 53 53 54
                           54 54 54 54 53 53 53 53 54 54 53 53 53 54 53 53
                          53 53 53 53 53 53 53 53 53 53 54 54 54 54 54 54 54
                           53 53 53 53 53 53 53 53 53 54 54 53 52 49 45 53
                          80 115 63 107 67 40 34 38 40 43 46 47 48 49 50 50
                          51 51 52 52 52 52 52 52 52 52 52 52 52 52 52 52 51 51
                           51 51 51 51 51 50 50 51 51 51 51 51 52 52 52 53
                          54 54 55 55 55 56 56 56 56 56 56 56 56 56 55 55 55
                           55 55 55 55 54 55 55 55 55 55 55 55 55 54 54 54 55
                          54 54 54 54 54 54 54 54 54 54 54 54 54 54 54 53 53
                           54 54 54 54 54 54 54 54 53 53 53 54 53 54 54 54
                          54 55 55 54 52 47 45 62 100 126 126 104 58 31 25
                          31 36 39 42 45 47 48 49 50 51 51 51 51 51 52 52 52
                           52 52 52 52 52 52 52 51 51 51 50 50 50 50 50 50
                          49 49 49 49 50 50 51 52 52 53 54 55 55 56 56 56 56
```

```
                    56 56 56 55 55 55 55 54 54 54 54 54 54 54 54 54
                   54 54 54 54 54 54 55 55 55 55 55 55 55 54 54 55 55
                    55 55 55 55 55 55 54 54 54 55 55 55 55 55 55 55
                   55 54 54 54 54 55 54 54 54 54 54 54 54 53 50 45 52
                    83 124 63 122 75 38 25 28 34 37 40 44 46 47 48 49
                    50 50 50 51 51 51 52 52 52 51 51 51 51 51 51 50
                   50 51 50 49 49 49 49 49 49 48 49 49 49 49 49 49 50
                    51 52 53 54 55 55 56 55 55 55 55 55 55 54 54 54
                   54 54 53 54 54 54 54 54 54 54 53 53 53 54 54 54 54
                    54 54 55 54 54 54 54 54 54 53 53 54 53 53 54 54
                   54 53 54 54 54 54 54 54 55 54 54 55 55 55 55 55 55
                    56 55 54 53 51 48 58 87 119 126 99 61 39 34 38 41
                    43 46 47 49 50 51 52 52 53 53 54 53 53 53 54 54
                   54 54 53 53 53 53 53 53 53 52 52 52 52 52 51 51 52
                    52 52 52 52 52 53 53 54 54 56 56 56 57 57 57 57
                   57 57 57 57 56 56 56 56 56 56 56 56 56 56 56 56 56
                    56 56 57 56 56 56 56 56 56 56 56 56 56 56 56 56 56
                   57 57 56 56 57 57 57 56 56 56 56 56 56 57 57 57 57
                    57 57 55 53 49 53 76 108 126 110 74 46 37 39 42
                   44 47 49 50 51 52 53 54 55 55 55 55 55 55 55 55 55
                    55 55 55 55 55 55 54 54 54 54 54 53 53 53 53 53
                   53 53 53 54 54 55 55 56 56 56 57 57 58 58 58 58 58
                    58 57 57 57 57 57 57 57 56 56 56 56 57 57 57 </
                   digits>
35              </value>
36            </sequence>
37          </component>
38        </sequenceSet>
39      </component>
40      <subjectOf>
41        <annotationSet>
42          <component>
43            <annotation>
44              <code code="MDC_ECG_TIME_PD_QRS" codeSystem="
                    2.16.840.1.113883.6.24" codeSystemName="MDC" />
45              <value xsi:type="PQ" value="" unit="ms" />
46              <component>
47                <annotation>
48                  <code code="MDC_ECG_WAVC_TYPE" codeSystem="
                        2.16.840.1.113883.6.24" codeSystemName="MDC" />
49                  <value xsi:type="CE" code="MDC_ECG_WAVC_QRSWAVE"
                        codeSystem="2.16.840.1.113883.6.24"
                        codeSystemName="MDC" />
50                  <support>
```

```
51                          <supportingROI>
52                            <code code="ROIPS" codeSystem="
                                  2.16.840.1.113883.5.4" codeSystemName="
                                  ActCode" />
53                            <component>
54                              <boundary>
55                                <code code="TIME_RELATIVE" codeSystem="
                                      2.16.840.1.113883.5.4" codeSystemName="
                                      ActCode" displayName="Relative Time" />
56                                <value xsi:type="IVL_TS">
57                                  <low value="540" unit="ms" />
58                                  <high value="580" unit="ms" />
59                                </value>
60                              </boundary>
61                            </component>
62                            <component>
63                              <boundary>
64                                <code code="MDC_ECG_LEAD_I" codeSystem="
                                      2.16.840.1.113883.6.24" codeSystemName="
                                      MDC" />
65                              </boundary>
66                            </component>
67                          </supportingROI>
68                        </support>
69                      </annotation>
70                    </component>
71                  </annotation>
72                </component>
73                <component>
74                  <annotation>
75                    <code code="MDC_ECG_TIME_PD_QRS" codeSystem="
                          2.16.840.1.113883.6.24" codeSystemName="MDC" />
76                    <value xsi:type="PQ" value="" unit="ms" />
77                    <component>
78                      <annotation>
79                        <code code="MDC_ECG_WAVC_TYPE" codeSystem="
                              2.16.840.1.113883.6.24" codeSystemName="MDC" />
80                        <value xsi:type="CE" code="MDC_ECG_WAVC_QRSWAVE"
                              codeSystem="2.16.840.1.113883.6.24"
                              codeSystemName="MDC" />
81                        <support>
82                          <supportingROI>
83                            <code code="ROIPS" codeSystem="
                                  2.16.840.1.113883.5.4" codeSystemName="
```

```
                    ActCode" />
84                <component>
85                  <boundary>
86                    <code code="TIME_RELATIVE" codeSystem="
                          2.16.840.1.113883.5.4" codeSystemName="
                          ActCode" displayName="Relative Time" />
87                    <value xsi:type="IVL_TS">
88                      <low value="1145" unit="ms" />
89                      <high value="1185" unit="ms" />
90                    </value>
91                  </boundary>
92                </component>
93                <component>
94                  <boundary>
95                    <code code="MDC_ECG_LEAD_I" codeSystem="
                          2.16.840.1.113883.6.24" codeSystemName="
                          MDC" />
96                  </boundary>
97                </component>
98              </supportingROI>
99            </support>
100         </annotation>
101       </component>
102     </annotation>
103   </component>
104   <component>
105     <annotation>
106       <code code="MDC_ECG_TIME_PD_QRS" codeSystem="
              2.16.840.1.113883.6.24" codeSystemName="MDC" />
107       <value xsi:type="PQ" value="" unit="ms" />
108       <component>
109         <annotation>
110           <code code="MDC_ECG_WAVC_TYPE" codeSystem="
                  2.16.840.1.113883.6.24" codeSystemName="MDC" />
111           <value xsi:type="CE" code="MDC_ECG_WAVC_QRSWAVE"
                  codeSystem="2.16.840.1.113883.6.24"
                  codeSystemName="MDC" />
112           <support>
113             <supportingROI>
114               <code code="ROIPS" codeSystem="
                      2.16.840.1.113883.5.4" codeSystemName="
                      ActCode" />
115               <component>
116                 <boundary>
```

```
117                         <code code="TIME_RELATIVE" codeSystem="
                                2.16.840.1.113883.5.4" codeSystemName="
                                ActCode" displayName="Relative Time" />
118                         <value xsi:type="IVL_TS">
119                           <low value="1740" unit="ms" />
120                           <high value="1775" unit="ms" />
121                         </value>
122                       </boundary>
123                     </component>
124                     <component>
125                       <boundary>
126                         <code code="MDC_ECG_LEAD_I" codeSystem="
                                2.16.840.1.113883.6.24" codeSystemName="
                                MDC" />
127                       </boundary>
128                     </component>
129                   </supportingROI>
130                 </support>
131               </annotation>
132             </component>
133           </annotation>
134         </component>
135         <component>
136           <annotation>
137             <code code="MDC_ECG_TIME_PD_QRS" codeSystem="
                    2.16.840.1.113883.6.24" codeSystemName="MDC" />
138             <value xsi:type="PQ" value="" unit="ms" />
139             <component>
140               <annotation>
141                 <code code="MDC_ECG_WAVC_TYPE" codeSystem="
                        2.16.840.1.113883.6.24" codeSystemName="MDC" />
142                 <value xsi:type="CE" code="MDC_ECG_WAVC_QRSWAVE"
                        codeSystem="2.16.840.1.113883.6.24"
                        codeSystemName="MDC" />
143                 <support>
144                   <supportingROI>
145                     <code code="ROIPS" codeSystem="
                            2.16.840.1.113883.5.4" codeSystemName="
                            ActCode" />
146                     <component>
147                       <boundary>
148                         <code code="TIME_RELATIVE" codeSystem="
                                2.16.840.1.113883.5.4" codeSystemName="
                                ActCode" displayName="Relative Time" />
```

```
149                         <value xsi:type="IVL_TS">
150                           <low value="2330" unit="ms" />
151                           <high value="2370" unit="ms" />
152                         </value>
153                       </boundary>
154                     </component>
155                     <component>
156                       <boundary>
157                         <code code="MDC_ECG_LEAD_I" codeSystem="
                                2.16.840.1.113883.6.24" codeSystemName="
                                MDC" />
158                       </boundary>
159                     </component>
160                   </supportingROI>
161                 </support>
162               </annotation>
163             </component>
164           </annotation>
165         </component>
166         <component>
167           <annotation>
168             <code code="MDC_ECG_TIME_PD_QRS" codeSystem="
                    2.16.840.1.113883.6.24" codeSystemName="MDC" />
169             <value xsi:type="PQ" value="" unit="ms" />
170             <component>
171               <annotation>
172                 <code code="MDC_ECG_WAVC_TYPE" codeSystem="
                        2.16.840.1.113883.6.24" codeSystemName="MDC" />
173                 <value xsi:type="CE" code="MDC_ECG_WAVC_QRSWAVE"
                        codeSystem="2.16.840.1.113883.6.24"
                        codeSystemName="MDC" />
174                 <support>
175                   <supportingROI>
176                     <code code="ROIPS" codeSystem="
                            2.16.840.1.113883.5.4" codeSystemName="
                            ActCode" />
177                     <component>
178                       <boundary>
179                         <code code="TIME_RELATIVE" codeSystem="
                                2.16.840.1.113883.5.4" codeSystemName="
                                ActCode" displayName="Relative Time" />
180                         <value xsi:type="IVL_TS">
181                           <low value="2920" unit="ms" />
182                           <high value="2960" unit="ms" />
```

```
183                                    </value>
184                                  </boundary>
185                                </component>
186                                <component>
187                                  <boundary>
188                                    <code code="MDC_ECG_LEAD_I" codeSystem="
                                           2.16.840.1.113883.6.24" codeSystemName="
                                           MDC" />
189                                  </boundary>
190                                </component>
191                              </supportingROI>
192                            </support>
193                          </annotation>
194                        </component>
195                      </annotation>
196                    </component>
197                    <component>
198                      <annotation>
199                        <code code="MDC_ECG_TIME_PD_QRS" codeSystem="
                               2.16.840.1.113883.6.24" codeSystemName="MDC" />
200                        <value xsi:type="PQ" value="" unit="ms" />
201                        <component>
202                          <annotation>
203                            <code code="MDC_ECG_WAVC_TYPE" codeSystem="
                                   2.16.840.1.113883.6.24" codeSystemName="MDC" />
204                            <value xsi:type="CE" code="MDC_ECG_WAVC_QRSWAVE"
                                 codeSystem="2.16.840.1.113883.6.24"
                                 codeSystemName="MDC" />
205                            <support>
206                              <supportingROI>
207                                <code code="ROIPS" codeSystem="
                                       2.16.840.1.113883.5.4" codeSystemName="
                                       ActCode" />
208                                <component>
209                                  <boundary>
210                                    <code code="TIME_RELATIVE" codeSystem="
                                           2.16.840.1.113883.5.4" codeSystemName="
                                           ActCode" displayName="Relative Time" />
211                                    <value xsi:type="IVL_TS">
212                                      <low value="3470" unit="ms" />
213                                      <high value="3505" unit="ms" />
214                                    </value>
215                                  </boundary>
216                                </component>
```

```
217                         <component>
218                           <boundary>
219                             <code code="MDC_ECG_LEAD_I" codeSystem="
                                    2.16.840.1.113883.6.24" codeSystemName="
                                    MDC" />
220                           </boundary>
221                         </component>
222                       </supportingROI>
223                     </support>
224                   </annotation>
225                 </component>
226               </annotation>
227             </component>
228           </annotationSet>
229         </subjectOf>
230       </series>
231     </component>
232   </AnnotatedECG>
```

ANEXOS

ANEXO I

Mensagem no formato HL7 V3

I.1 Exemplo de mensagem de Conteúdo de Domínio

Listagem I.1: Exemplo de mensagem de Conteúdo de Domínio em HL7 V3

```
<observationEvent>
   <id root="2.16.840.1.113883.19.1122.4" extension="1045813"
       assigningAuthorityName="GHH LAB Filler Orders"/>
   <code code="1554-5" codeSystemName="LN"
         codeSystem="2.16.840.1.113883.6.1"
         displayName="GLUCOSE^POST 12H CFST:MCNC:PT:SER/PLAS:QN"/>
   <statusCode code="completed"/>
   <effectiveTime value="200202150730"/>
   <priorityCode code="R"/>
   <confidentialityCode code="N"
      codeSystem="2.16.840.1.113883.5.25"/>
   <value xsi:type="PQ" value="182" unit="mg/dL"/>
   <interpretationCode code="H"/>
   <referenceRange>
      <interpretationRange>
         <value xsi:type="IVL_PQ">
            <low value="70" unit="mg/dL"/>
            <high value="105" unit="mg/dL"/>
         </value>
         <interpretationCode code="N"/>
      </interpretationRange>
   </referenceRange>
   <author>
      <time value="200202150730"/>
```

```
25          <modeCode code="WRITTEN"/>
26          <signatureCode code="S"/>
27          <assignedEntity>
28              <id root="2.16.840.1.113883.19.1122.3" extension="
                    444-444-4444"/>
29              <assignedPerson>
30                  <name>
31                      <given>Harold</given>
32                      <given>H</given>
33                      <family>Hippocrates</family>
34                      <suffix qualifier="AC">MD</suffix>
35                  </name>
36              </assignedPerson>
37          </assignedEntity>
38      </author>
39      <recordTarget>
40          <patientClinical>
41              <id root="2.16.840.1.113883.19.1122.5" extension="444-22-2222
                    "
42              assigningAuthorityName="GHH Lab Patient IDs"/>
43              <statusCode code="active"/>
44              <patientPerson>
45                  <name use="L">
46                      <given>Eve</given>
47                      <given>E</given>
48                      <family>Everywoman</family>
49                  </name>
50                  <asOtherIDs>
51                      <id extension="AC555444444" assigningAuthorityName="SSN
                            "
52                      root="2.16.840.1.113883.4.1"/>
53                  </asOtherIDs>
54              </patientPerson>
55          </patientClinical>
56      </recordTarget>
57      <inFulfillmentOf>
58          <placerOrder>
59              <id root="2.16.840.1.113883.19.1122.14" extension="845439"
60                assigningAuthorityName="GHH OE Placer orders"/>
61          </placerOrder>
62      </inFulfillmentOf>
63  </observationEvent>
```

Neste exemplo de anotação, é incluído o prestador de serviços, *Harold H. Hippocrates*,

com seu ID e nome. Existem dois níveis de informação presentes, o nível do praticante e o nível de pessoa. O ID está no nível de praticante, enquanto que o nome está no nível de pessoa. A paciente (*Eve E. Everywoman*) também é representada por dois níveis - o paciente e o pessoa. O ID está disponível assim como o nome, providenciando alguma forma para checagem de erro.

O pedido original para a observação do laboratório é referenciado pela relação de ato FLFS, denominada *inFulfillmentOf*, que identifica o pedido pelo número do operador. Esse número deve ser usado pelo receptor para combinar os resultados para o pedido.

ANEXO II

Anotações do formato aECG HL7

II.1 Exemplo de anotação do complexo QRS

Listagem II.1: Exemplo de anotação do complexo QRS no padrão aECG HL7

```
<annotation>
   <code code="MDC_ECG_WAVC" codeSystem="2.16.840.1.113883.6.24"/>
   <value xsi:type="CE" code="MDC_ECG_WAVC_QRSWAVE" codeSystem="
       2.16.840.1.113883.6.24"/>
   <support>
      <supportingROI>
         <code code="ROIPS" codeSystem="2.16.840.1.113883.5.4"/>
         <component>
            <boundary>
               <code code="TIME_RELATIVE" codeSystem="
                   2.16.840.1.113883.5.4"/>
               <value xsi:type="IVL_PQ">
                  <low value="434" unit="ms"/>
                  <high value="554" unit="ms"/>
               </value>
            </boundary>
         </component>
      </supportingROI>
   </support>
</annotation>
```

O código “MDC_ECG_WAVC” denota que a anotação II.1 é um componente de forma de onda ECG. O valor “MDC_ECG_WAVC_QRSWAVE” denota que o tipo do

componente da forma de onda é o complexo QRS. O código ROI (Região de Interesse) “ROIPS” denota que esse ROI está parcialmente especificado, ou seja, aplica-se à todos os condutores. O código de fronteira “TIME_RELATIVE” significa que a fronteira está especificada em tempo relativo, ou seja, relativo ao começo da série. O valor da fronteira é o intervalo de quantidade física que iniciou em 434 e terminou em 554, em unidade de millissegundos.

II.2 Exemplo de anotação da onda P

Listagem II.2: Exemplo de anotação da onda P no padrão aECG HL7

```
<annotation>
   <code code="MDC_ECG_WAVC" codeSystem="2.16.840.1.113883.6.24"/>
   <value xsi:type="CE" code= "MDC_ECG_WAVC_PWAVE" codeSystem="
       2.16.840.1.113883.6.24"/>
   <support>
      <supportingROI>
         <code code="ROIFS" codeSystem="2.16.840.1.113883.5.4"/>
         <component>
            <boundary>
               <code code="TIME_RELATIVE" codeSystem="
                   2.16.840.1.113883.5.4"/>
               <value xsi:type="IVL_PQ">
                  <low value="286" unit="ms"/>
                  <high value="388" unit="ms"/>
               </value>
            </boundary>
         </component>
         <component>
            <boundary>
               <code code="MDC_ECG_LEAD_II" codeSystem="
                   2.16.840.1.113883.6.24"/>
            </boundary>
         </component>
      </supportingROI>
   </support>
</annotation>
```

Este exemplo é similar ao II.1, exceto que o código para o componente da forma de onda indica P-wave ao invés de QRS. Além disso, essa anotação utiliza um ROI (código “ROIFS”) especificado plenamente, que significa que todas as fronteiras são

especificadas. Esse ROI está no condutor (*lead*) II apenas. Note que a fronteira do condutor II não tem nenhum valor porque todas as voltagens do condutor II estão inclusas no ROI.

II.3 Exemplo de anotação da onda T

Listagem II.3: Exemplo de anotação da onda T no padrão aECG HL7

```
<annotation>
   <code code="MDC_ECG_WAVC" codeSystem="2.16.840.1.113883.6.24"/>
   <value xsi:type="CE" code="MDC_ECG_WAVC_TWAVE" codeSystem="
       2.16.840.1.113883.6.24"/>
   <support>
      <supportingROI>
         <code code="ROIPS" codeSystem="2.16.840.1.113883.5.4"/>
         <component>
            <boundary>
               <code code="TIME_RELATIVE" codeSystem="
                   2.16.840.1.113883.5.4"/>
               <value xsi:type="IVL_PQ">
                  <high value="854" unit="ms"/>
               </value>
            </boundary>
         </component>
      </supportingROI>
   </support>
</annotation>
```

Nesta anotação, a fronteira ainda é um intervalo de quantidade física. Porém, apenas a parte alta (*high*) do intervalo é especificada. Essa anotação denota que a onda T tem um começo e fim, mas apenas o fim é dado em 854 ms no início da série.