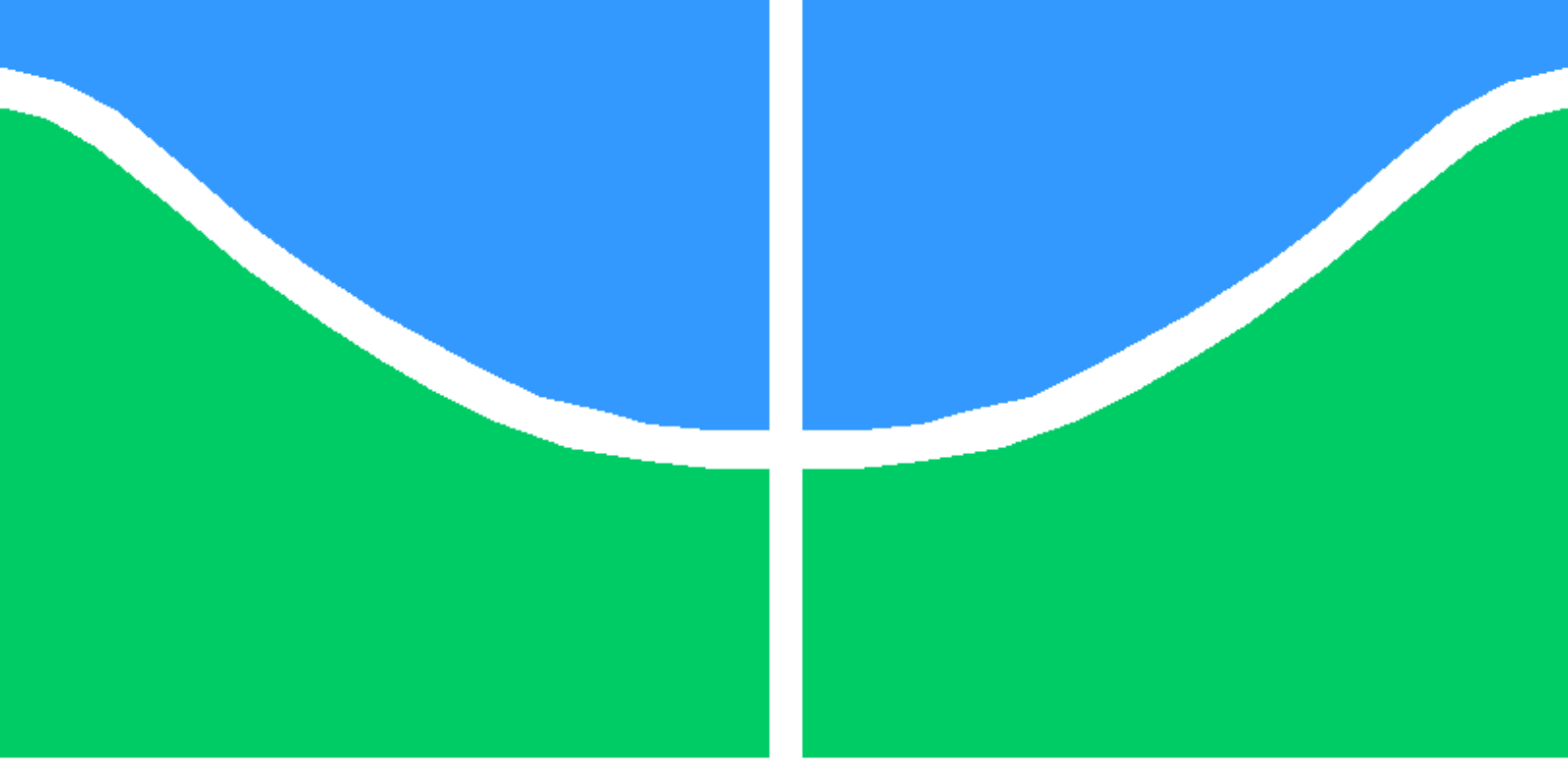

FRAMEWORK PARA A CRIAÇÃO E MANIPULAÇÃO DE REDES BAYESIANAS EM DISPOSITIVOS MÓVEIS

MAURO HENRIQUE LIMA DE BONI

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA BIOMÉDICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA BIOMÉDICA

FGA - FACULDADE GAMA

UnB - UNIVERSIDADE DE BRASÍLIA
FGA - FACULDADE GAMA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA BIOMÉDICA

FRAMEWORK PARA A CRIAÇÃO E MANIPULAÇÃO DE REDES BAYESIANAS EM DISPOSITIVOS MÓVEIS

MAURO HENRIQUE LIMA DE BONI

Orientador: PROF. DR. ADSON FERREIRA DA ROCHA, UNB - FGA GAMA

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA BIOMÉDICA

PUBLICAÇÃO FGA.DM - 007A/2012
BRASÍLIA-DF, 28 DE DEZEMBRO DE 2012.

UnB - UNIVERSIDADE DE BRASÍLIA
FGA - FACULDADE GAMA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA BIOMÉDICA

FRAMEWORK PARA A CRIAÇÃO E MANIPULAÇÃO DE REDES BAYESIANAS EM DISPOSITIVOS MÓVEIS

MAURO HENRIQUE LIMA DE BONI

DISSERTAÇÃO DE MESTRADO ACADÊMICO SUBMETIDA AO PROGRAMA DE ENGENHARIA BIOMÉDICA DA FGA - FACULDADE GAMA, DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA BIOMÉDICA.

APROVADA POR:

Prof. Dr. Adson Ferreira da Rocha, UnB - FGA Gama
Orientador

Prof. Dr. Talles Marcelo Gonçalves, PUC-GO
Avaliador Externo

Profa. Dra. Suélia de Siqueira Rodrigues Fleury Rosa- FGA Gama
Avaliador Interno

BRASÍLIA, 28 DE DEZEMBRO DE 2012.

FICHA CATALOGRÁFICA
MAURO HENRIQUE LIMA DE BONI
***Framework* para criação e manipulação de Redes Bayesianas em dispositivos móveis, [Distrito Federal] 2012.**
83p., 201x297 mm (FGA/UnB Gama, Mestre, Engenharia Biomédica, 2012)
Dissertação de Mestrado - Universidade de Brasília. Faculdade Gama - Programa de Pós-Graduação em Engenharia Biomédica

REFERÊNCIA BIBLIOGRÁFICA

MAURO HENRIQUE LIMA DE BONI (2012) *Framework* para criação e manipulação de Redes Bayesianas em dispositivos móveis. Dissertação de Mestrado em Engenharia Biomédica, Publicação 007A/2012, Programa de Pós-Graduação em Engenharia Biomédica, Universidade de Brasília, Brasília, DF, 83p.

CESSÃO DE DIREITOS

AUTOR: Mauro Henrique Lima de Boni
TÍTULO: *Framework* para criação e manipulação de Redes Bayesianas em dispositivos móveis.
GRAU: Mestre ANO: 2012

Mauro Henrique Lima de Boni
509 SUL, Alameda 16, QI 16, LT 10, Bairro: Plano Diretor Sul, Palmas-TO, CEP 77.016-612.

Dedicatória

Dedico este trabalho a minha esposa Elisangela e nossa filha Dhebora.

"Toda a educação, no momento, não parece motivo de alegria, mas de tristeza. Depois, no entanto, produz naqueles que assim foram exercitados um fruto de paz e de justiça."
Bíblia Sagrada, Hebreus 12, 11

Mauro Henrique Lima de Boni

Agradecimentos

Agradeco, primeiramente, ao Deus altíssimo por todos os dias de vida a mim concedidos e por ter abençoado e ajudado-me até esse momento. Esta conquista tão importante em minha vida é para honra e gloria do Seu santo nome;

À minha esposa Elisangela e minha filha Dhebora pelo apoio, compreensão, dedicação e amor oferecidos em todas as etapas de elaboração deste trabalho. Nós nos preocupamos, estudamos juntos, perdemos horas de sono, Esta vitória também é de vocês;

Aos meus pais Roque e Therezinha, pela educação e incentivo que me fizeram crescer. A credibilidade e confiança de vocês ajudaram-me muitíssimo e honro vocês com esta conquista;

Aos professores Adson Ferreira da Rocha e Thalles Marcelo Gonçalves A. Barbosa por toda solidariedade atribuída, conhecimento compartilhado, disponibilidade, disposição, paciência e ajuda.

Mauro Henrique Lima de Boni

RESUMO

FRAMEWORK PARA CRIAÇÃO E MANIPULAÇÃO DE REDES BAYESIANAS EM DISPOSITIVOS MÓVEIS

Autor: Mauro Henrique Lima de Boni
Orientador: Prof. Dr. Adson Ferreira da Rocha, UnB - FGA Gama
Brasília, Dezembro de 2012.
Programa de Pós-Graduação em Engenharia Biomédica

As redes bayesianas são ferramentas interessantes para construção de cenários em que seja necessário representar algum tipo de conhecimento. Elas são um modelo computacional probabilístico e podem ser utilizada como auxílio para a tomada de decisões. Dentre suas características, duas merecem destaque: a facilidade na computação das probabilidades e o fato de que permitem a visualização das variáveis aleatórias envolvidas através da utilização de um grafo direcional e acíclico. Isso faz com que seja possível o estabelecimento de relações do tipo causa e efeito entre as variáveis. Elas têm sido utilizadas em aplicações de diagnóstico e prognóstico envolvendo informações incompletas ou incertas. Esse trabalho teve como objeto de estudo a elaboração de um *software* genérico que permitisse a modelagem e manipulação de redes bayesianas. Desta forma ele apresenta o projeto e a implementação de um *framework*. São tratados os fundamentos sobre redes bayesianas e também são fornecidos alguns exemplos. Os mecanismos de propagação de evidencias são descritos, bem como algumas considerações sobre a modelagem de uma rede bayesiana. Para a elaboração desse estudo, foram usadas três formas distintas de pesquisa : exploratória, experimental e estudo de caso. Inicialmente a pesquisa exploratória foi responsável por fornecer as referências teóricas para o desenvolvimento do raciocínio. A pesquisa experimental, por sua vez, visava a reprodução dos mecanismos básicos das redes bayesianas. O estudo de caso permitiu uma análise qualitativa do *software*, o que permitiu dentre outras coisas, verificar se as estruturas propostas no *framework* estavam adequadas para a modelagem de redes. Além disso, foi possível observar se as evidências seriam propagadas na rede e se os estados do nodo seriam alterados. Como principal contribuição deste trabalho destaca-se a facilidade de reuso oferecida pela arquitetura proposta, que foi baseada no uso de padrões de projeto. O trabalho demostra o uso desta arquitetura por meio de um estudo de caso, onde uma rede, que representa um domínio inerente a engenharia biomédica, é modelada e implementada pelo *framework*. Essa aplicação é implementada em um dispositivo móvel do tipo *tablet*. Esses dispositivos oferecem mobilidade dentre outras funcionalidades que os tornam interessantes aos profissionais de área de saúde.

Palavras-chave: inteligência artificial, redes bayesianas, arquitetura de *software*, reuso.

ABSTRACT

FRAMEWORK PARA CRIAÇÃO E MANIPULAÇÃO DE REDES BAYESIANAS EM DISPOSITIVOS MÓVEIS

Author: Mauro Henrique Lima de Boni
Supervisor: Prof. Dr. Adson Ferreira da Rocha, UnB - FGA Gama
Brasília, December of 2012.
Post-Graduation Program in Biomedical Engineering

The Bayesian networks are interesting tools for the construction of scenarios that must represent some type of knowledge. They are a computational model probabilistic and can be used as an aid to decision making. Among its characteristics, two deserve special mention: the ease of computing probabilities and the fact that allow viewing of the random variables involved through the use of a directional graph and out acyclic engine frequencies on. This makes it possible the establishment of relations of type cause and effect between the variables. They have been used in applications of diagnostic and prognostic information involving incomplete or uncertain. This work has as its object of study the preparation of a generic software to allow the modeling and manipulation of Bayesian networks. This way he presents the design and implementation of a framework. Treaties are the foundations on Bayesian networks and are also provided some examples. The mechanisms of the propagation of evidence are described, as well as some considerations on the modeling of a bayesian network. For the preparation of this study were used three distinct forms of search : exploratory, experimental and case study. Initially the exploratory research was responsible for providing the references theoretical for the development of reasoning. Experimental research, in turn, was aimed at the playback of the basic mechanisms of Bayesian networks. The case study has a qualitative analysis of software, which allowed among other things, check if the structures proposed in the framework were appropriate for modeling of networks. In addition, it was possible to observe if the evidence would be propagated on the network and the states of node would be changed. The software project is discussed and its implementation is presented through examples. As the main contribution of this work there can be emphasized the reuse easiness provided by the software architecture proposed in this work, which is based on project patterns. The work demonstrates the utilization of that architecture by means of a case study. Its use was done in an application where the domain is inherent to the biomedical engineering. This application is implemented in a mobile device of the tablet type. Those devices offer mobility among other functionalities that make them interesting to the professionals of the health field.

Keywords: artificial intelligence, beysian networks, Software Architecture, reuse.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS FONTE

Lista de Abreviaturas e Siglas

GUI Graphical User Interface

HMM Hidden Markov Model

HTML Hyper Text Markup Language

IBOPE Instituto Brasileiro de Opinião Pública e Estatística

IDC International Data Corporation

MIT Massachusetts Institute of Technology

MVC Model View Controller

PDA Personal Digital Assistant

RDB redes bayesianas dinâmicas

RSSF Redes de sensores sem fio

SDK Software Development Kit

TPC Tabela de Probabilidade condicional

WIN Worldwide Independent Network of Market Research

Capítulo 1

Introdução

Este capítulo apresenta uma visão geral do trabalho. Será apresentado o contexto no qual este trabalho de pesquisa está inserido. A seguir, é descrito o problema que se deseja resolver, bem como os objetivos e finalidades da pesquisa. Por fim, é mostrada a metodologia de pesquisa utilizada, as limitações do escopo desta proposta e a maneira como ele está organizado.

1.1 O cenário atual

Em nossos dias é cada vez mais nítida a presença de diversos equipamentos cujo objetivo é permitir que nós estejamos acessíveis a outras pessoas. Para isso temos os telefones fixos, que estão ligados a um endereço e temos os telefones celulares, que permitem, inicialmente, que sejam realizadas chamadas de voz, mas que tem como característica principal a mobilidade. A partir dos anos 90 do século XX, uma tecnologia deixou o meio acadêmico e militar e tornou-se acessível ao público: a internet. Desde então ela vem transformado-se no meio cada vez mais utilizado para fazer com que as pessoas possam manter-se conectadas, via *email*, redes sociais, mensagens via texto, *voip* ou mesmo sob a forma de vídeo conferencias.

Em uma palestra chamada "The Post Pc Internet PC", proferida nos anos 1990 pelo professor do MIT David Clark, foi descrito um mundo no qual o computador pessoal é menos importante, porque qualquer dispositivo, seja uma TV, relógio de pulso, um par de óculos e até mesmo uma torradeira, poderão estar conectados à internet, e desta forma serem usados para trocar informações com outros usuários ou até mesmo com outros equipamentos, pois poderão um processador embarcado, bem como toda a infraestrutura necessária para conectar-se com uma rede. Assim, previa ele, um relógio pode temporariamente tornar-se uma pequena agenda, usando sua tela para exibir os seus compromissos para o dia, sendo que essas informações são enviadas sem fios para o relógio a partir da sua pasta de armazenamento na internet. A era pós-PC é marcada, portanto, pela heterogeneidade, uma

vez que não envolveria somente o computador pessoal.

Os *tablets* podem ser definidos como um computador para uso geral, mas surgiram sob a era da heterogeneidade. Desta forma, eles são dispositivos portáteis bastante flexíveis. Há poucos botões em seu exterior e ele contem um monitor que utiliza a tecnologia *touch screen* que é o que o torna diferente de todos os demais equipamentos. Toda a interação do usuário com o dispositivo é feita em sua tela, dispensando uso de outros periféricos tais como teclado e mouse.

Outra descrição para esta emergente categoria de dispositivos é fornecida pelo governo do Brasil, que publicou a Lei 12.507 de 11 de outubro de 2011[Planalto 2011], que alterou o art. 28 da Lei 11.196, de 21 de novembro de 2005, para que os *tablets* produzidos no Brasil tenham alguns impostos reduzidos, de forma que este incentivo possa atrair a instalação de fábricas. Eles são definidos como sendo "máquinas automáticas de processamento de dados, portáteis, sem teclado, que tenham uma unidade central de processamento com entrada e saída de dados por meio de uma tela sensível ao toque de área superior a $140cm^2$ e inferior a $600cm^2$". Eles oferecem acesso à rede sem fio (seja via *wifi* (802.11x), 3G e 4G), sendo que sua tela com tamanho significativamente maior que a de um *smartphone*, aproximando-se do tamanho da tela dos notebooks. Essa característica torna seu uso mais intuitivo do que o dos microcomputadores, sejam eles *desktops* ou portáteis. Podem ser usados os dedos do usuário, bem como as chamadas *stylus*, que são instrumentos simples, que lembram canetas.

Por terem telas maiores que as dos *smartphones*, seu uso é mais confortável para tarefas como a leitura de *emails*, acesso à internet. Além de serem mais leves que os netbooks ou notebooks, eles podem ser usados com apenas uma das mãos, o que possibilita que o usuário possa usá-lo enquanto está em pé ou mesmo deitado. Some a todas essas características a duração de sua bateria que é em torno de 7 horas.

O iPad tornou-se um grande sucesso de vendas tão logo começou a ser comercializado nos Estados Unidos no ano de 2010. Entretanto, na década de 90, surgiram os PDA's, que eram equipamentos projetados para atuarem como gerenciador de informações pessoais e que também possuíam tela sensível ao toque e eram portáteis. Porém, apesar de possuírem recursos avançados como reconhecimento de escrita, eles jamais conseguiram alcançar o sucesso que os *tablets* atingiram. Os PDA's ofereciam recursos muito mais limitados que aqueles encontrados nos *desktops* existentes na época, por exemplo. Além disso, sua capacidade de conectar-se à internet e a outros dispositivos necessitava de cabos ou de uma conexão sem fio lenta, feita por infra-vermelho. Apesar disso, esses dispositivos foram usados em vários outros trabalhos, como por exemplo em [S.Castro et al.], onde os autores usaram um desses dispositivos para realizar a tarefa de captura automática de sintomas.

A diferença entre esses dois dispositivos, que aparentemente tem o mesmo propósito, está principalmente no fato em que os *tablets* são muito mais flexíveis que os antigos PDA's, além disso, seu recursos computacionais, tais como capacidade de processamento e de armazenamento, ajudam a oferecer soluções próprias que não dependem do uso de equipamentos *desktop*. Os *tablets* foram inicialmente definidos como sendo "mecanismo de acesso à mídia" [Wong and Joussen 2011], mas atualmente eles vem tomando o lugar dos *notebooks*, *netbooks*, *smartphones* e video-games portáteis [Honeyman-Buck 2010] em diversas aplicações.

Uma forma de mensurar o quão altas são as vendas destes dispositivos, somente no ano de 2010, uma pequisa do IBOPE [IBOPE], em parceria com a WIN, apurou que 24% dos entrevistados manifestam a intenção de comprar um tablet. A pesquisa ouviu 40.557 pessoas em 44 países. No Brasil, foram 2.002 entrevistas. Outra pesquisa [Folha.com 2011], realizada pela IDC, estima que no ano de 2011 foram comercializados no Brasil mais de 400.000 destes aparelhos, sendo que em 2010 foram estimadas vendas em torno de 100.000 unidades. Outras pesquisas informam que foram importados para o Brasil mais de 60.000 unidades do iPad em 2010 [Abril].

Novos avanços tecnológicos, tais como, processadores multi-core, melhorias em relação ao consumo de energia, aumento na oferta de conexões sem fio de alta velocidade, diminuição em seu tamanho e peso e principalmente, a queda de preços que todos os equipamentos eletrônicos sofrem ao longo dos anos, criam a tendência de que eles tornem-se cada vez mais presentes, tomando muitas vez o lugar de outros equipamentos, seja em casa ou mesmo no ambiente corporativo [Geyer and Felske 2011]. Apesar de terem uma capacidade computacional menor, se comparada com os *noteboks*, por exemplo, eles são capazes de executar aplicações feitas especialmente para eles de forma rápida e ágil. Essa agilidade pode ter contribuído também para que essa popularidade pudesse ser atingida.

Os aplicativos disponíveis para estes equipamentos são muitos. Podem ser encontrados jogos, editores de texto, *web browsers* e versões digitais de livros impressos que incluem sons e animações [Siqueira 2011], há ainda um crescente número de softwares que interessam a públicos mais específicos. Os profissionais da área de saúde têm à sua disposição programas como o AO Surgery Reference [Foundation] que tem como objetivo fornecer acesso a uma base de dados sobre cirurgia. O iDoctor, cuja interface é mostrada na Figura 1.1, permite que sejam criados registros médicos completos dos pacientes, sendo possível fazer o acompanhamento de todo o histórico de peso, pressão arterial, diabetes, colesterol, creatinina. Além disso é possível que o estado atual de um paciente seja comparado com um estado anterior.

Em [Volonte et al. 2011], o autor utiliza uma aplicação, executada em um iPad, que

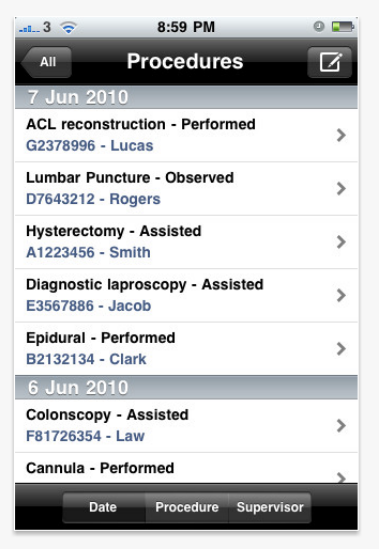

Figura 1.1: Exemplo da interface do iDoctor

exibe e manipula imagens em 3D, para auxiliar no procedimento de segmentectomia anatômica. Além de trabalhos acadêmicos há a oferta de *software*, sejam eles pagos ou gratuitos. O OsiriX é um *software* que utiliza o padrão DICOM, que é utilizado para armazenamento e transferência de imagens médicas. Além disso, permite que as imagens sejam manipuladas diretamente no *tablet* e apresenta todas as modalidades de imagens. Ele usa a rede *WiFi* ou 3G para receber as imagens de outros dispositivos compatíveis com o padrão DICOM. Como exemplo de programa livre de pagamento, há o *Mobile MIM*[Software], que é usado para visualizar, registrar, fundir imagens médicas para diagnóstico. A figura 1.2 mostra-o em execução em um *smartphone*, onde é possível ver a imagem e os dados do paciente. Um ponto importante é que as imagens podem ser acessadas a partir de um servidor próprio seja via rede local ou usando a internet, com as imagens armazenadas em um nuvem.

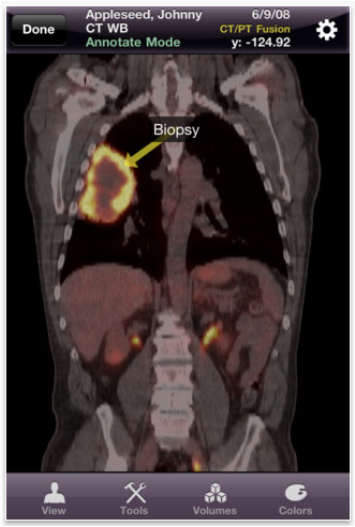

Figura 1.2: Exemplo da interface do MiM.

A rede bayesiana é um ferramenta de inteligência artificial que cria mecanismos basea-

dos em raciocínio probabilístico que utilizam o teorema de Bayes para processar a incerteza. Elas são utilizadas em áreas que incluem diagnóstico, investigação, reconhecimento de imagens, dentre outras que necessitam de conclusões que sejam baseadas em informações incompletas, ou seja, onde há algum grau de incerteza. Permitem que sejam criados modelos baseados em relação de causa-efeito entre as variáveis e essa relação permite que as probabilidades de ocorrência de um evento, sejam alteradas por eventos anteriores. Para facilitar o seu uso, abstraindo o modelo matemático e permitindo que as redes sejam modeladas de forma mais intuitiva, há softwares que disponibilizam uma interface gráfica ou GUI. Essas interfaces permitem que as redes sejam criadas de maneira visual, proporcionando uma melhoria na produtividade, uma vez que o usuário deverá ficar focado apenas no processo de modelagem e na obtenção de probabilidades. Além disso realizam inferências, executam a propagação de evidências, dentre outras atividades.

1.2 Objeto de estudo

Durante o levantamento bibliográfico inicial, notou-se que apesar de existir uma grande quantidade de aplicativos, não foi encontrado nenhum que permitisse a criação e manipulação de modelos baseados em redes bayesianas, que fosse executado em dispositivos móveis. Os aplicativos existentes que a implementam são executáveis apenas em micro computadores. Desta maneira, aproveitando o sucesso comercial dos *tablets* e capacidade de que as redes bayesianas possuem de serem úteis em várias situações, o objetivo deste trabalho é descrever a construção de uma ferramenta capaz de modelar redes bayesianas genéricas que possam ser executados em um dispositivo móvel do tipo *tablet*. Essas redes devem ser modeladas de maneira visual de forma que sejam aproveitados os recursos da interface sensível ao toque.

Foi necessário investigar como deve ser projetada a arquitetura de *software* de forma que se possa alcançar um bom nível de reuso. O reuso é uma prática bastante utilizada em áreas como a eletrônica, por exemplo. Quando um engenheiro vai projetar um novo circuito eletrônico qualquer, ele imagina o que esse circuito deverá ser capaz de fazer e seleciona componentes (circuitos integrados, capacitores, resistores, etc.) que já existem. Assim desta forma, ele garantirá que esses componentes já foram suficientemente testados e qual é o desempenho (velocidade, capacidade de operação) oferecido por eles. Desta maneira, para a criação de uma solução, há reuso de componentes. Essa mesma ideia pode ser usada na construção de *software*, onde a ferramenta usada para a solução de um problema pode ser a utilização de um outro software mais genérico. Através da reutilização é possível obter aumento da qualidade e a redução do esforço de desenvolvimento. Dessa forma, foi necessário encontrar dentro das técnicas de engenharia de *software*, alguma que pudesse ser usada para esse objetivo. Isso foi um dos principais objetivos da etapa de levantamento

bibliográfico. O reuso da ferramenta gerada deveria possibilitar que as redes bayesianas pudessem ser implementadas em qualquer ambiente.

De acordo com Pressman [Pressman 2006] um *framework*, é um arcabouço ou seja, uma estrutura básica, capaz de fornecer um comportamento genérico, por meio de funcionalidades de uso comum. A tecnologia de *frameworks* possibilita que sejam gerados produtos a partir dessa estrutura genérica. Já Gamma *et al.* [Gamma et al. 2000] afirma que um *framework* e um conjunto de objetos que colaboram com o objetivo de atender a um conjunto de responsabilidades para uma aplicação específica ou um domínio de aplicação. De acordo com Sommerville [Sommerville et al. 2007], eles oferecem reuso da arquitetura projetada. Além disso, eles se beneficiam do desenvolvimento orientado ao objeto e dão suporte ao reuso das classes que o compõe. Quando usado em conjunto com padrões de projetos, bibliotecas de classes, componentes, entre outros, *frameworks* de aplicação têm o potencial para aumentar a qualidade de software e reduzir o esforço de desenvolvimento [Sommerville et al. 2007].

As tarefas feitas para que se consiga obter um *framework* são as seguintes:

- Análise de domínio – busca-se entender qual o contexo no qual o artefato está inserido. Quais são relações com os usuários, com outros sistemas, que tarefas deve realizar, que informações ele precisa, que resultados espera-se que ele possa fornecer
- Projeto arquitetural – o projeto da arquitetura de um sistema envolvem decisões estratégicas como por exemplo, de que maneira pretende-se que as diversas partes dos sistemas estejam organizadas, como serão atendidos requisitos relacionados a mobilidade. Durante o projeto da arquitetura, são imaginados os mecanismos necessários para que se possa alcançar o reuso do *software*.
- Projeto do *framework* – tendo em vista que o projeto da arquitetura é uma descrição em alto nível, o projeto do *framework* tem como objetivo gerar descrições que possam ser implementadas em computador.
- Implementação e testes – o *framework* é codificado, através do uso de uma linguagem de programação que possa anteder aos requisitos do projeto da arquitetura. Ele deve sr testado para que sejam encontrados e solucionados problemas relativos à semânica e a sintaxe das instruções escritas na linguagem usada.

As etapas de projeto têm um uma importância grande, pois sabe-se que a manutenção de um *framework* pode tornar-se difícil, pois como ele é uma aplicação inacabada, é mais torna difícil tanto para os mantenedores de frameworks quando para os desenvolvedores de aplicação testar, respectivamente, o *framework* e a instância do *framework*, ou seja uma aplicação, isoladamente. Dessa forma, durante o projeto, devem ser feitas escolhas, como por

exemplo, com relação ao padrão arquitetural, de maneira que seja encontrado um caminho que possa facilitar a manutenção.

1.3 Justificativa

O uso da informática, como ferramenta auxiliar em diversas áreas, está em constante evolução. Como causas deste processo podemos citar: o desenvolvimento dos sistemas e o aumento crescente da capacidade operacional das máquinas disponíveis (capacidade de armazenar, tempo de processamento), simplificação da interface usuário/máquina e diminuição dos custos dos equipamentos.

A pesquisa na área de inteligência artificial possui, dentre seus objetivos principais, produzir resultados que possam ser aplicados em situações do dia-a-dia. Temos como exemplos robôs autônomos e sistemas de auxílio à tomada de decisão, também conhecidos como sistemas especialistas. Há uma grande variedade de técnicas que podem ser escolhidas. A escolha da rede bayesiana, para atuar como o mecanismo de representação do conhecimento, deu-se devido aos seguintes motivos, que foram apresentados em [Lucas 2004, Lucas et al. 2004].

A construção de um sistema baseado na opinião de especialistas pode ser feita utilizando-se redes bayesianas. Isto é necessário no caso em que não exista uma base de dados de aprendizado ou esta não forneça resultados satisfatórios. Desta maneira, elas são uma solução genérica que permite que informações vindas de profissionais da área de saúde, modelem cenários em que se possam extrair conclusões a partir de dados incompletos. Elas podem ainda possuir a capacidade de criar ou melhorar sua topologia pelo uso de técnicas de aprendizado. Essa característica, entretanto não foi avaliada no contexto desse trabalho.

Uma das maiores facilidades encontra-se na apresentação aos médicos dos métodos utilizados de forma que estes possam entender o funcionamento dos sistemas construídos e, por conseguinte, julgar com mais lucidez o nível de confiança atribuído aos sistemas. Redes bayesianas, aplicadas a problemas médicos, facilitam essa comunicação por apresentarem um modelo gráfico com relações causais que tentam aproximar os mecanismos utilizados naturalmente por médicos em sua prática cotidiana. Este é um dos motivos que fazem com que as redes bayesianas estejam entre as ferramentas de inteligência artificial que têm tido maior sucesso em aplicações práticas para a medicina.

O uso de um *framework* que possa modelar um cenário genérico é importante para a criação de novas aplicações. As redes bayesianas, assim como seus métodos associados, são especialmente bons na captura e trabalho com incertezas. Elas estão sendo aplicadas à

biomedicina e têm programas utilizados em cuidados com a saúde (*health-care*). Há mais de uma década vêm tornado-se cada vez mais popular para o controle de conhecimento incerto. Há vários cenários que possuem esta característica, dentre os quais podemos citar dois cenários:

- a triagem de pacientes, onde não se sabe *a priori* quais são os sinais e sintomas informados pelo paciente, mas essas informações são necessárias para estabelecer diagnósticos, a seleção da melhor especialidade médica e ajudar na escolha da melhor alternativa de tratamento.
- auxiliar o profissional de área de saúde a projetar uma solução que envolvam outras tecnologias, com as redes de sensores sem fio aplicadas ao corpo humano. Os nodos sensores poderiam ser imaginados como nodos bayesianos e seria utilizados para estabelecer os diagnósticos ou prognósticos que fossem necessários.

Desta forma, o uso deste *framework* facilitará a construção desses sistemas. Em trabalhos como [Gonçalves and Brunetto 2008], são utilizadas outras técnicas para representação de raciocínio. Neste artigo, o autor usou uma rede neural artificial, que pode ser um mecanismo utilizado para o reconhecimento de padrões, que usa o chamado treinamento supervisionado, onde devem ser apresentadas informações que são comparadas com as respostas obtidas pela estrutura. Caso a resposta não seja a esperada, deve haver um aprendizado, isto é, os neurônios que compõe a rede devem ser ajustados, de maneira que nas próximas vezes em que essa informação seja apresentada, a rede possa reconhecê-la corretamente. Para que se tenha uma rede suficientemente treinada é preciso tempo, pois devem ser apresentados vários exemplos. Além disso, não há uma representação gráfica que facilite a modelagem de um domínio. As redes bayesianas dispensam o treinamento supervisionado além de representarem um domínio por uma relação de causa e efeito, o que facilita a sua aplicação.

1.4 Objetivos

1.4.1 Geral

O objetivo geral deste trabalho é apresentar o projeto de solução em *software* desenvolvida com intuito de implementar uma ferramenta que seja utilizada em *tablets* que permita a modelagem e a manipulação de redes bayesianas visando representação de cenários em que seja necessário o uso de informações incompletas.

1.4.2 Específicos

- Propor e apresentar uma arquitetura para o *framework*, de maneira que a sua reutilização seja facilitada.

- Criar um mecanismo que permita a implementação de algoritmos diferentes para a propagação de evidências. É importante conhecer os tipos de algoritmos usados para esse objetivo, e que cada um destes tipos possui várias implementações diferentes.
- Desenvolver uma aplicação e um *framework* que permita a criação e uso de redes bayesianas, que devem ser executadas em dispositivos móveis do tipo *tablets* que usem o sistema operacional iOS.
- Apresentar aspectos sobre projeto e implementação do *software* proposto.
- Uso da aplicação em estudos de caso de maneira que se possa comprovar sua eficácia.

1.5 Metodologia

O presente trabalho utilizou três tipos de pesquisa. O primeiro foi a pesquisa exploratória, de forma que se pudesse conseguir uma maior familiaridade sobre o tema. A pesquisa exploratória, conforme Cervo [Cervo and BERVIAN 2007], é o passo inicial no processo de pesquisa. Ela permite que sejam definidos objetivos e que sejam buscadas novas informações sobre o assunto. O segundo tipo, para conseguir reproduzir os mecanismo que compõe a rede bayesiana, foi usada a pesquisa experimental. Nesta pesquisa, o partir da determinação do objeto de estudo, identifica-se que variáveis participam ou interferem no processo, verifica-se a as relações de dependência entre as variáveis. Por fim, um estudo de caso foi realizado, que consistiu na implementação de uma aplicação, que usava a arquitetura proposta e o *framework*. A Figura 1.3 sintetiza a metodologia utilizada.

Um levantamento bibliográfico inicial foi realizado para que se pudesse conhecer o que são as redes bayesianas e entender sua utilização, seu mecanismo de propagação de evidências, quais eram os trabalhos realizados que a utilizavam. Além disso, em face aos desafios tecnológicos, tais como qual a linguagem de programação que deveria ser utilizada, analisar possíveis estratégias para a implementação que maximize sua futura reutilização e encontrar técnicas que pudessem permitir a construção de uma solução que pudesse ser utilizada em vários cenários. Dessa forma, durante o levantamento bibliográfico, foram consultados também diversos trabalhos relacionados à engenharia de *software*. Por meio de pesquisa à base de dados buscou-se artigos científicos, dissertações ou testes. Livros também foram consultados, uma vez que são fontes de informações mais detalhadas. O projeto da arquitetura ocorreu nesta etapa.

A etapa anterior forneceu os conhecimentos iniciais. A Próxima etapa, o projeto do *framework*, valeu-se desses conhecimentos. Desta forma, o entendimento desses conhecimentos propiciou informações para a realização da primeira etapa de projeto que foi a elicitação dos requisitos. Essa fase foi responsável por gerar um conjunto inicial de funcionalidades que deviam ser fornecidas pelo *framework*. O projeto devia criar uma

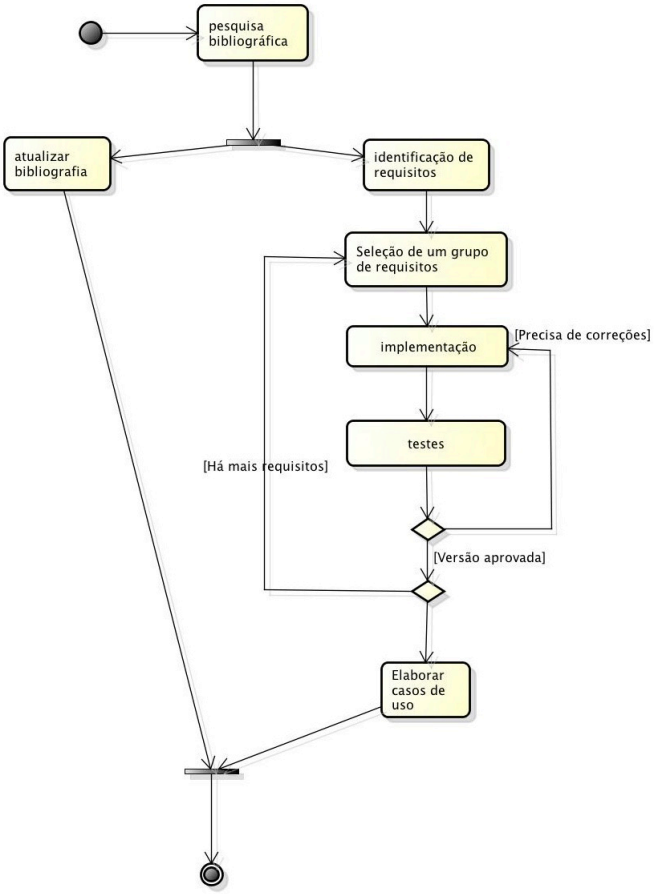

Figura 1.3: Metodologia adotada no desenvolvimento do trabalho

solução que pudesse contemplar esses requisitos, mas também devia atender a outros requisitos de ordem técnica, como por exemplo, fazer com que os artefatos produzidos tivessem o máximo de reuso. Durante esse processo novos requisitos podem surgir em decorrência de necessidades que emergem durante a implementação, como atender uma necessidade específica gerada pela tecnologia escolhida. Por exemplo, descobrir qual a linguagem de programação que deveria ser utilizada.

A implementação do *software* usou a metodologia do desenvolvimento incremental, onde foram geradas versões intermediárias do *software*, a partir de um conjunto de requisitos, escolhidos dentre os requisitos gerais. Essas versões foram testadas e avaliadas, visando procurar erros e corrigi-los.

Para que se pudesse analisar a aplicabilidade da solução proposta, a arquitetura e o *framework* foram usados em um estudo de caso. Para a realização desse estudo, uma nova pesquisa exploratória foi realizada, dessa vez para que se pudesse reunir informações sobre as redes de sensores sem fio (RSSF).

Observar que durante o tempo em que as tarefas relacionadas ao projeto, implementa-

ção e testes estavam sendo executadas, a bibliografia estava sendo atualizada. Sempre que uma nova dúvida surgia, a bibliografia era consultada e caso não fosse encontrada nenhuma informação que pudesse ajudar a solucioná-la, uma busca por mais material era realizada.

1.6 Organização do trabalho

Este trabalho está estruturado em capítulos. A seguir é apresentada uma descrição sumária de cada capítulo:

- O Capítulo 2 apresenta o levantamento bibliográfico realizado que fornece os fundamentos teóricos usados no desenvolvimento da dissertação. Traz os conceitos básicos essenciais para o entendimento das Redes Bayesianas e seu emprego para fazer inferências. Apresenta definições teóricas, bem como conceitos relativos a probabilidade usado por elas. Mostra como construí-las e presenta um exemplo. Há uma breve relação de trabalhos relacionados.

- O Capítulo 4 descreve as ferramentas utilizadas para implementação das ideias expostas no Capítulo 2. Apresenta as justificativas que levaram à escolha das ferramentas utilizadas. Os conceitos a respeito de orientação ao objeto e padrões de projeto, que são utilizados na implementação são também expostos.

- A implementação do *framework* é discutida no Capitulo 3. São apresentados aspectos inerentes ao projeto e implementação do *software*. Seus requisitos funcionais, não funcionais e sua arquitetura. Algoritmos implementados são discutidos.

- O Capítulo 5 mostra a utilização do framework em problemas relacionados com a engenharia biomédica, sendo apresentados os resultados obtidos.

- No Capítulo 6 é feita a conclusão, sendo apresentados as dificuldades encontradas, além de considerações a respeito desta dissertação e recomendações para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Introdução

Este capítulo tem o objetivo de apresentar os fundamentos sobre redes bayesianas: o que é a rede bayesiana; porque utilizá-las; expõe brevemente qual o mecanismo matemático que a representa e como construí-la. Define o que é inferência e quais as formas para propagá-la através dos nodos. A última seção faz um breve relato sobre trabalhos que utilizaram redes bayesianas em vários cenários, tais como: realizar o Interrogatório Sintomatológico, auxiliar no diagnóstico de doenças cardíacas e ajudar uma iniciativa de inclusão digital de idosos.

2.2 Redes bayesianas

As redes bayesianas são um modelo de representação do conhecimento que trabalham com o conhecimento incerto e incompleto através da teoria da probabilidade bayesiana, publicada pelo matemático Thomas Bayes em 1763 [ISBA].

As redes bayesianas oferecem uma abordagem para o raciocínio probabilístico que engloba teoria de grafos, para o estabelecimento das relações entre sentenças e ainda, teoria de probabilidades, para a atribuição de níveis de confiabilidade. A fundamentação teórica apresentada nesta seção utiliza como aplicação uma rede bayesiana modelada para o diagnostico clínico de câncer nos pulmões. Este modelo é coerente com o raciocínio empregado em clínica médica, de fácil compreensão e possibilita ilustrar diferentes aspectos conceituais. Entretanto, cumpre ressaltar que os valores de probabilidade associados às variáveis do modelo foram arbitrados. Desta forma, não representam dados reais associados às patologias destacadas.

Um paciente que tem sofrido com falta de ar, vai ao médico preocupado que isso possa

indicar que ele tenha câncer de pulmão. O médico sabe que outras doenças, tais como tuberculose e bronquite, são possíveis causas dessa falta de ar, além do câncer. Ele também sabe que outras informações relevantes incluem o fato do paciente ser fumante (o que aumenta a chance dele ter câncer ou bronquite) e qual nível de poluição ele foi exposto. Um exame de raios-X indicaria ou câncer ou bronquite. A rede que representa o domínio descrito pode ser vista na Figura 2.1. As várias setas que aparecem na figura indicam que há uma influência direta entre duas ou mais variáveis no modelo, assim, a variável "câncer"é influenciada diretamente por "fumante"e por "poluição". Os valores contidos dentro de cada uma das variáveis representam os estados que elas podem ter. Uma pessoa pode estar exposta a índices de poluição altos ou baixos, ela pode ser ou não fumante, pode apresentar ou não dispinéia.

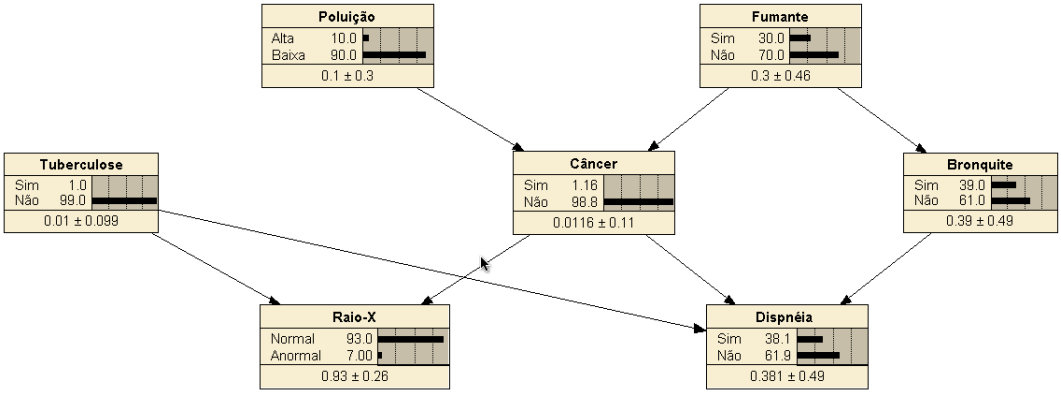

Figura 2.1: Exemplo de rede bayesiana

Dependendo do ambiente a ser modelado, uma rede bayesiana pode possuir tantos nodos que sua manipulação sem o auxilio de um *software* torna-se impraticável. Sendo assim, para facilitar a representação gráfica de tais estruturas existem ferramentas computacionais capazes de auxiliar no projeto e validação das redes bayesianas. Há vários *softwares* livres tais como: Banjo [Department of Computer Science], MSBNx [Microsoft], Netica [Norsys] e BNJ [KDD] que podem ser utilizados nesta tarefa.

2.2.1 Formalização

Uma rede bayesiana, ou rede probabilística, para um conjunto de variáveis $X = \{X_1, \ldots, X_n\}$ consiste em uma estrutura gráfica em rede G em um conjunto de distribuições de probabilidades locais associadas a cada variável Pr. Essa estrutura gráfica tem forma de um grafo direcionado acíclico. Os nós contidos em G têm uma correspondência de um-para-um com as variáveis do domínio. Desta maneira, a partir dessa definição, temos

$$B = (Pr, G). \tag{2.1}$$

onde B é a rede bayesiana, Pr e G representam, respectivamente, o conjunto de probabilidades condicionais e a estrutura gráfica. G tem a forma de um grafo direcionado acíclico,

com nós $V(G) = \{V_1, \ldots, V_n\}, n \geq 1$ e arcos $A(G) \subseteq V(G)xV(G)$. Cada nó V_i em G representa uma variável randômica que tem um conjunto finito de valores. Desta maneira temos

$$G = (V(G), A(G)) \tag{2.2}$$

Os arcos no dígrafo modelam as influências probabilísticas entre as variáveis. Podemos dizer, a grosso modo, que um arco $V_i \rightarrow V_j$ entre dois nós V_i e V_j indica que há influência entre as variáveis V_i e V_j; a ausência de um arco significa que uma variável não influencia a outra diretamente. Podemos fazer uma definição mais formal: a variável V_i é dependente de seus pais, mas é independente de qualquer nó não-descendente de seus pais. Essa propriedade é comumente chamada de Condição de Markov.

Associado a estrutura gráfica de uma rede bayesiana está uma distribuição de probabilidade conjunta Pr que é representado de forma fatorada. Para cada variável V_i no grafo está determinado um conjunto de distribuições condicionais; cada uma destas distribuições descreve o efeito de uma combinação especifica dos valores dos pais $\pi(V_i)$ de V_i, na distribuição probabilística de V_i. Esses conjuntos de distribuições, de acordo com [Lucas et al. 2004], definem uma única distribuição

$$Pr(V_1, \ldots, V_n) = \prod_{i=1}^{n} Pr(V_i|\pi(V_i)) \tag{2.3}$$

2.2.2 Construção de uma rede bayesiana

Os passos mostrados a seguir são uma compilação dos passos elencados por [Lucas et al. 2004, Korb and Nicholson 2004b]. Os autores propõem uma forma sistematizada para a construção de uma rede, sendo esse processo executado em forma espiral onde a cada iteração, um nível maior de refinamento será obtido.

1. Nodos e valores
   Em primeiro lugar, é necessário identificar quais são as variáveis de interesse. Isso envolve responder a seguinte questão: quais são os nodos relevantes e quais os valores que eles têm? Em [Lucas et al. 2004], o autor acrescenta ainda que essa seleção geralmente baseia-se em entrevistas com especialistas, descrições do domínio e em uma extensiva análise do propósito da rede a ser construída. Diz ainda que, muitas vezes, o conhecimento sobre os processos patológicos e fisiológicos envolvidos, no caso de aplicações para área médica, é usado para guiar a identificação de variáveis. Os valores assumidos são mutuamente exclusivos, o que significa que a variável terá um e somente um desses valores de cada vez. Podemos elencar como tipos comuns de nodos:

   - Nodos Booleanos - representam proposições. Tem valores binários verdadeiro

(*true*) e falso (*false*). Em um domínio de diagnóstico médico, um nodo chamado "Câncer" poderia representar se um determinado paciente tem ou não a doença. Para isso obtendo os valores verdadeiro e falso.

- Valores Ordenados - um nodo Poluição pode representar a exposição de um paciente a poluição e ter valores como baixo, médio e alto.
- Valores integrais - um nodo chamado idade pode representar a idade de um paciente e ter valores possíveis entre 1 e 120.

Mesmo em uma fase inicial, escolhas sobre a modelagem precisam ser feitas, por exemplo, ao invés de criar um nodo que represente a idade exata de um paciente, ela pode ser melhor representada se for agrupada em conjuntos diferentes, tais como bebê, criança, adolescente, jovem, adulto e idoso. O objetivo é escolher valores que representem o domínio de maneira eficiente, com um nível de detalhamento adequado requerido pelo problema a ser modelado.

Como mostrado na Figura 2.1, há sete variáveis a serem analisadas: nível de exposição à poluição, o fato de ser fumante, a presença da dispnéia, o resultado do exame de Raios X, se o paciente têm bronquite e tuberculose, além do câncer. Para cada uma dessas variáveis é necessário criar um conjunto finito de estados que cada um dos nodos podem assumir. Assim, temos a relação mostrada na tabela 2.1

Tabela 2.1: Escolha dos conjuntos de valores para os nodos

Nome do Nodo	Valores
Poluição	$\{Alta, Baixa\}$
Fumante	$\{Sim, Nao\}$
Cancer	$\{Sim, Nao\}$
Bronquite	$\{Sim, Nao\}$
Tuberculose	$\{Sim, Nao\}$
Dispnéia	$\{Sim, Nao\}$
Raio-X	$\{Normal, Anormal\}$

2. Estrutura

   A estrutura ou topologia de uma rede deve representar relacionamentos qualitativos entre as variáveis. Em particular, dois nodos de um grafo poderiam estar conectados diretamente somente se um afeta o outro, com arcos indicando a direção do efeito. Quando estamos falando sobre a estrutura da rede é útil empregarmos a seguinte notação: um nodo é pai de um outro nodo. Essa relação ocorre somente se existir um arco a partir do nodo anterior até o posterior. Ao olharmos para o exemplo fornecido na figura 2.1, veremos que o nodo "Fumante" tem um nodo filho "Câncer". Caso exista uma sequência direcionada de nodos, um nodo é ancestral de outro se ele

aparece antes na sequência, enquanto outro é chamado de descendente se ele vem depois de outro na mesma sequência. A figura 2.1 nos mostra que o nodo "Dispnéia" é descendente do nodo "Fumante" e este por sua vez é ancestral de "Dispnéia".

Uma vez que as variáveis que serão incluídas em uma rede foram definidas, os relacionamentos de dependência e independência entre elas devem ser analisados e expressos em uma estrutura gráfica. Para esse fim, a noção causalidade é usada. Questões feitas durante as entrevistas com especialistas podem ser como "o que causaria isso?" ou ainda "quais manifestações poderiam ter essa causa?". Uma vez que os relacionamentos são definidos, eles devem ser expressos de maneira gráfica isto é, através de um grafo. É por essa razão que existem dois arcos que saem de "Fumante" e vão para "Câncer" e "Bronquite". O significado é claro: o fato de ser fumante pode causar câncer e bronquite.

Em [Lucas et al. 2004] os autores dizem que nessa etapa é realizada a identificação de restrições probabilísticas qualitativas, e lógicas, que são derivadas dos relacionamentos entre as variáveis.

3. Probabilidades Condicionais
   Uma vez que a topologia da rede foi definida, o próximo passo é quantificar as relações entre os nodos conectados, e isto é feito pela a especificação de uma distribuição de probabilidade condicional para cada nodo, observando todas as possíveis combinações dos valores dos seus nodos pais. Cada combinação dessas recebe o nome de instanciação do conjunto de pais. Para cada instanciação distinta, é necessário especificar a probabilidade que o nodo filho terá. Assim, se um nodo possuir vários pais, ou se esses pais puderem ter um número elevado de valores, a tabela de probabilidade condicional será muito grande. Cada linha dessa tabela conterá os valores dos nodos pai e o valor que nodo filho terá. O tamanho dessa tabela é exponencial ao número de nodos pai. Assim em redes cujos valores são booleanos, uma variável com n pais terá uma TPC com 2^n probabilidades. A atribuição de probabilidades a cada variável pode ser obtida por meio de entrevistas com especialistas na área de estudo, bem como através de dados. Para o domínio apresentado pela figura 2.1,o médico que atendeu o paciente, com base em sua experiência e em informações probabilísticas obtidas em pesquisas, sabe que:

   - 30% da população daquele local é composta de fumantes;
   - 10% dos habitantes do local estão expostos a altas doses de poluição;
   - Um em cada cem habitantes tem tuberculose;

   Essas variáveis são representadas pelos nodos "Fumante", "Poluição" e "Tuberculose". Esse nodos representam nodos pai na rede apresentada, pois não possuem nodos antecessores. Eles possuem nodos que são influenciados por eles. As TPC dos nodos

pais são simples de serem definidas, bastando a aplicação direta das probabilidades descritas, uma vez que não há a dependência de nenhum outro evento. Por isso elas são chamadas de probabilidades incondicionais. As tabelas 2.2, 2.3 e 2.4 mostram como ficam os valores dos estados para esses nodos.

Tabela 2.2: TPC para o nodo Fumante

Valor	P(Fumante)
Sim	0.30
Não	0.70

Tabela 2.3: TPC para o nodo Poluição

Valor	P(Poluição)
Sim	0.10
Não	0.90

Tabela 2.4: TPC para o nodo Tuberculose

Valor	P(Tuberculose)
Sim	0.01
Não	0.99

As demais variáveis devem ser analisadas de uma forma um pouco diferente pois como são variáveis que possuem pais, os valores de seus estados são obtidos por meio da aplicação de probabilidades condicionais para cada caso condicional dos nós pais. O caso condicional é uma combinação possível dos valores para os nodos pais. Deve-se também lembrar que cada uma das tabelas terá $estados\_dos\_filhos^{numero\_pais}$. As tabelas 2.5, 2.6, 2.7 e 2.8, seguem essa regra. Em uma rápida análise delas podemos verificar que o exame de Raios X tem a probabilidade de 0.98, ou seja 98% de mostrar algo anormal caso o paciente tenha Câncer de pulmão ou tuberculose. Que o fato do paciente ser fumante traz a ele um risco maior de desenvolver câncer de pulmão do que se ele estivesse exposto a uma alto nível de poluição.

4. Propriedade de Markov

   O processo de modelagem com redes bayesianas requer a compreensão sobre a propriedade de Markov: não existem dependências diretas no sistema que está sendo modelado, além daquelas mostradas explicitamente através de arcos. Na figura 2.1, não há influência direta entre *Tuberculose*, *Câncer* e *Bronquite*.

Tabela 2.5: TPC para o nodo Raio_X

P(Raio_X | Tuberculose,Cancer)		Tuberculose	Cancer
Normal	Anormal		
0.02	0.98	Sim	Sim
0.02	0.98	Sim	Não
0.02	0.98	Não	Sim
0.95	0.05	Não	Não

Tabela 2.6: TPC para o nodo Câncer

P(Cancer | Poluicao,Fumante)		Poluicao	Fumante
Sim	Não		
0.05	0.95	Alta	Sim
0.02	0.98	Alta	Não
0.03	0.97	Baixa	Sim
0.001	0.999	Baixa	Não

Com a execução das etapas anteriores, uma rede bayesiana completamente especificada é obtida. Antes que a rede possa ser usada em "produção", sua qualidade e valor clínico devem ser estabelecidos. De acordo com [Lucas et al. 2004], uma das técnicas usadas para aferir a qualidade da rede é executar uma análise sensitiva com dados reais. Essa análise é útil para conferir a qualidade dos dados obtidos na saída da rede. Essa avaliação pode ser efetuada de várias maneiras. Dentre elas, estão incluídas medidas de classificação de desempenho dado conjunto de casos reais e aferir a similaridade da estrutura ou a distribuição probabilística a uma rede-padrão reconhecida ou outro modelo probabilístico. Durante o processo de construção da rede bayesiana, deve-se tentar construir a rede de forma a que o resultado seja a rede mais compacta possível. Em [Korb and Nicholson 2004b], o autor diz que se a rede obtida for mais compacta, ela apresentará uma eficiência computacional maior, pois haverá menos probabilidades condicionais a serem especificadas, o que trará uma facilidade maior para o processo de criação das TPC's que poderá ser executada com mais exatidão. Com menos dependências causais, as atualizações das probabilidades serão feitas de maneira mais rápida, pois o processo será computacionalmente mais eficiente. A ordem em que os nodos são inseridos na rede também é importante. A construção de redes em que a ordem causal não é observada vai trazer uma complexidade extra, onde mais arcos serão representados. Para que a ordem causal seja observada, deve-se encontrar primeiramente as causas origem, ou seja aquela que não dependem de outras causa. É a partir dessas variáveis que os arcos devem partir, em sentido aos seus sucessores. Esse processo deve ser repetido até que os nodos filhos (também chamados de folhas) sejam inseridos na rede.

Em face a essa necessidade, de obter a rede mais compacta possível, a estrutura apresentada na figura 2.1 passa por uma análise a fim de que se pudesse confirmar se ela representava ou não a melhor opção. A rede possui oito arcos e o total de probabilidades da rede, somando o total de linhas das TPC, incluindo as incondicionais e as condicionais é de vinte e quatro.

Tabela 2.7: TPC para o nodo Bronquite

P(Bronquite | Fumante)		Fumante
Sim	Não	
0.6	0.4	Sim
0.3	0.7	Não

Tabela 2.8: TPC para o nodo Dispeneia

P(Dispneia | Cancer,Tuberculose,Bronquite)		Cancer	Tuberculose	Bronquite
Sim	Não			
0.9	0.1	Sim	Sim	Sim
0.7	0.3	Sim	Sim	Não
0.9	0.1	Sim	Não	Sim
0.7	0.3	Sim	Não	Não
0.9	0.1	Não	Sim	Sim
0.7	0.3	Não	Sim	Não
0.8	0.2	Não	Não	Sim
0.1	0.9	Não	Não	Não

O médico que modelou a rede lembrou-se de que o exame de Raios-X mostrará algo de anormal somente se o paciente tiver Câncer e/ou Tuberculose. Que se esse mesmo paciente tiver Câncer e/ou Tuberculose, a possibilidade de que ele sofra com falta de ar (dispnéia) aumenta. A partir dessa observação, ele imaginou qual seria o impacto produzido por um novo nodo na rede: "Tuberculose_ou_Cancer", que teria arcos vindos de "Câncer" e "Tuberculose". Os arcos que inicialmente saíam desses nodos e iam até "Raio_X" e "Dispnéia" agora sairão deste novo nodo. A nova estrutura obtida é mostrada na figura 2.2. Com a inclusão desse nodo, a rede permaneceu com o mesmo número de arcos que possuía anteriormente, ou seja oito. Mesmo com um nodo a mais, o número de probabilidades caiu para vinte e duas. A nova estrutura é computacionalmente mais eficiente. Essa técnica que foi utilizada, em que os nodos pai são agrupados, é conhecida como *Divorcing*.

2.2.3 Distribuições Canônicas

A construção das TPC pode exigir muita experiência e levar bastante tempo [Russell and Norvig 2004] e no pior caso podem ser necessárias até $O(2^k)$ linhas, onde k é a quantidade de pais de um nodo, ou seja, o tamanho de uma tabela aumenta de forma exponencial em função da quantidade de seus pais. Para um nodo binário X, cujos pais Y_i também são binários, a relação $P(X|Y_i)$ que completará a TPC de X, terá 2^n linhas. Essa com certeza é uma das maiores dificuldades encontradas no uso das redes bayesianas. Porém, nem sempre o relacionamento entre os nodos se dará de forma arbitraria e seus valores podem ajustar-se a algum tipo de forma-padrão. Desta maneira, a tabela pode ser especificada nomeando-se o padrão e fornecendo alguns parâmetros. Algumas das formas mais conhecidas são chamadas de nós determinísticos e ou ruidoso.

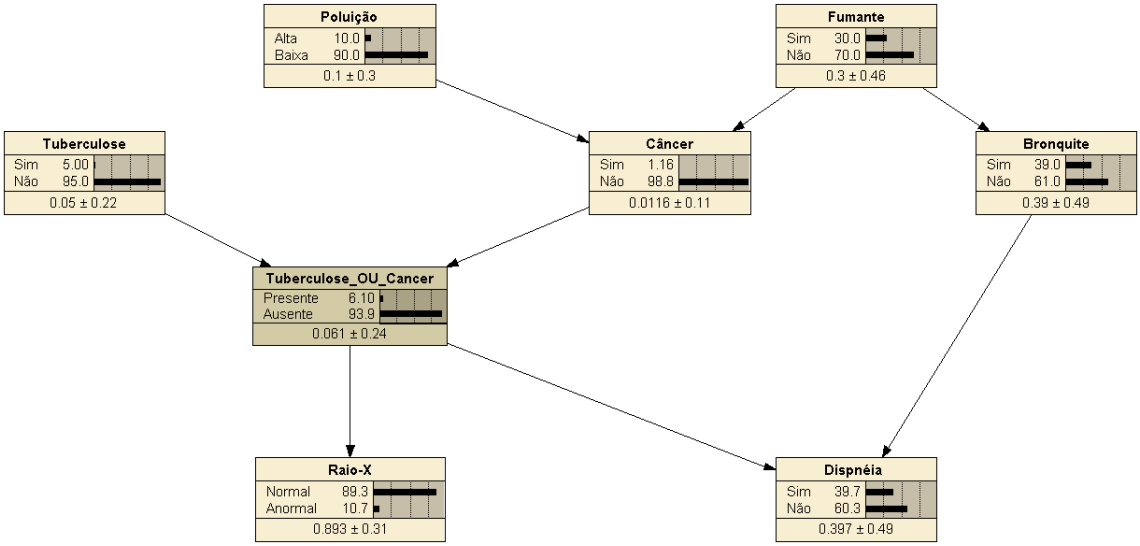

Figura 2.2: Estrutura da Rede após modificações

A forma canônica mais simples é o nodo determinístico. Ele tem seu valor especificado exatamente pelos valores de seus pais, sem qualquer incerteza, de maneira lógica. Desta forma, se imaginarmos que temos vários nodos que representam os estados que compõe o Brasil e um outro nodo que é filho destes, chamado de "brasileiro". Ele será verdadeiro se ao menos um dos seus pais é verdadeiro. Notar que a tabela será grande e terá 2^{26} linhas. Porém, seu preenchimento é fácil, pois o nodo brasileiro sempre será verdadeiro dada a disjunção dos pais. A Figura 2.3 representa esse exemplo.

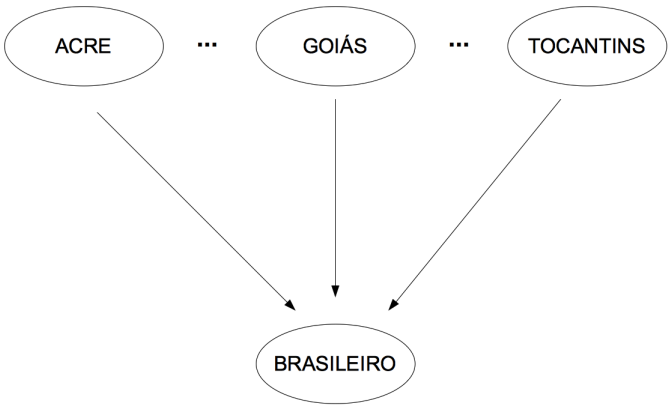

Figura 2.3: Exemplo de nodo determinístico

O ou ruidoso é uma extensão da estrutura *OR* apresentada por Pearl [Fallet et al. 2012] para reduzir um pouco o trabalho durante a construção das TPC. Esse modelo combina a probabilidade e a lógica. O ruido indica que a relação causal não é determinística, desta forma, qualquer causa pode produzir um efeito com uma probabilidade diferente. Podemos

aplica essa distribuição sempre em que uma varável tiver vários pais e seja muito difícil obter as combinações das relações entre os diversos pais para que se consiga chegar às probabilidades do filho.

Como exemplo, temos uma relação entre os nodos febre, malária, gripe e resfriado, sendo febre filho dos demais. Podemos dizer então que, febre é verdadeira se e somente se resfriado, gripe ou malária é verdadeira. Porém um paciente pode ter um resfriado, mas não apresentar febre. Em geral, relações que apresentam incertezas podem ser caraterizadas como relações que apresentam ruídos. Esse modelo, de acordo com [Russell and Norvig 2004], faz algumas suposições:

- todas as causas possíveis estão associadas ao efeito.
- A inibição de cada pai é independente da inibição de quaisquer outros pais, pois como mostrado no exemplo, o que inibe malária de causar febre é independente do que inibe gripe de causar febre.

A variável febre será falsa se e somente se todos os seus pais são falsos. A probabilidade que isso ocorra é o produto das probabilidades de inibição de cada um dos pais. Se as probabilidades de inibição para resfriado, gripe e malária são respectivamente 0,6, 0,2 e 0,1, a TPC do nodo febre será como a mostrada pela Tabela 2.9. Duas das linhas desta tabela podem ser preenchidas apenas aplicando definições: se todos os pais são verdadeiros, então o filho é verdadeiro e se todos os pais são falsos o filho será falso também. Todas as demais linhas podem ser obtidas por meio da obervação da quantidade de pais que são verdeiros: se houver um pai verdadeiro, então a probabilidade do filho ser verdadeiro é a igual ao valor da probabilidade de inibição do pai.

Caso existam mais pais verdadeiros, a probabilidade do filho ser falso é igual ao produto das probabilidades de inibição dos pais verdadeiros. Por isso, caso gripe e malária sejam verdadeiros, a probabilidade de um paciente apresentar febre é de 0,988, pois sabemos que a probabilidade de ser falso é dado por $0,2x0,1 = 0,02$. Assim ao diminuirmos 1 deste valor obtemos 0,988. Nesta caso o preenchimento da tabela depende de mais informações do que no caso de usarmos o nodo determinístico, porém bastaria conhecer apenas um dos valores das probabilidades dos pais (verdadeiro ou falso) para que se pudesse encontrar os demais valores.

As redes bayesianas podem conter *LOOPS* (laços). Um laço em uma rede bayesiana consiste em um caminho fechado no grafo obtido a partir da estrutura desta rede após a substituição dos seus arcos por arcos não direcionados. Os laços devem ser evitados pois é possível que uma informação que circule em um laço chegue a um mesmo nó por dois caminhos diferentes [Pearl and Russell 2011]. Isto pode levar a erros na avaliação das influências dos dados e também produzir instabilidade nesta avaliação.

Tabela 2.9: TPC gerada a partir de uma aplicação de ou ruidoso

Resfriado	Gripe	Malária	P(Febre)	¬P(Febre)
F	F	F	0,0	1,0
F	F	V	0,9	0,1
F	V	V	0,98	0,02 = 0,1 x 0,2
V	F	F	0,4	0,6
V	F	V	0,94	0,06 = 0,6 x 0,1
V	V	F	0,88	0,12 = 0,6 x 0,2
V	V	V	0,988	0,012 = 0,6 x 0,2 x 0,1

2.2.4 Usando a rede bayesiana

O uso de uma rede bayesiana implica a aplicação de dois conceitos:

- Evidência – Evidências são novas informações disponíveis que surgem em um dado evento. Essa nova informação faz com que uma mudança nos estados de uma variável ocorra e isso inicia o processo de propagação das probabilidades. Quando um nodo qualquer X está um estado específico e_x, onde $e_x \in E$ e E é um conjunto finito que representa os estados de X, escrevemos que $X = e_x$. Essa situação é chamada de instanciação ou também *hard evidence* [Korb and Nicholson 2004b].

- Inferência – Dada uma ou mais evidências, as probabilidades da rede deve ser calculadas. Esse processo é chamado de Inferência. o cálculo das probabilidades de um valor de uma variável, fará com que seja criado um fluxo de informação que percorrerá toda a rede. Esse fluxo não é limitado pela direção dos arcos. Essa é a tarefa mais comum que se deseja fazer com uma rede bayesiana. Também podemos chamar a inferência por propagação de probabilidades.

Se a evidência é "Dispnéia = Sim", as probabilidades dos outros nodos da rede serão afetadas em função desse evento. Neste caso poderemos efetuar um raciocínio que ocorre no sentido inverso ao dos arcos, pois a evidência foi atribuída em um nodo folha. Podemos dizer ainda que estamos efetuando neste caso um raciocínio diagnóstico, indo do sintoma para a causa.

Ao seguirmos a direção dos arcos, estamos raciocinando a partir de uma nova informação sobre as causas até novas opiniões sobre os efeitos. Por exemplo, o paciente informa ao médico que ele é fumante. Mesmo antes que todos os sintomas estejam avaliados, o médico sabe que a evidência "Fumante=Sim", aumentará as possibilidades do paciente estar com câncer. Isso muda também a expectativa do médico quanto ao paciente exibir outros sintomas, tais como a falta de ar e um resultado anormal no exame de Raio-X. Esse raciocínio tem o nome de preditivo ou prognóstico.

Uma outra forma de raciocínio envolve raciocinar sobre causas mútuas de um efeito comum, o que recebe o nome de raciocínio intercausal. Supondo que existam duas possíveis causas de um dado efeito, representados em uma estrutura em V. Essa situação é semelhante à encontrada pelos nodos Fumante e Poluição que possuem um efeito comum, Câncer. Ao olharmos para o modelo, veremos que as duas causas são independentes uma da outra, ou seja, o fato de um paciente ser fumante não muda a probabilidade de que ele esteja exposto à poluição. Se o paciente tiver câncer, isso aumentará as probabilidades para as suas causas, fazendo com que as chances de que ele seja um fumante e tenha sido exposto a uma alta taxa de poluição. Se ele é fumante, essa nova informação explica o fato de que o paciente tem câncer, o que diminuirá a probabilidade de que a doença esteja ligada à poluição.

2.2.5 Propagação de evidências – Inferência

A inferência em rede bayesiana, segundo [Russell and Norvig 2004], significa calcular a distribuição de probabilidade de um conjunto de variáveis, chamadas de variáveis de consulta, dado um evento observado, sendo esse evento um conjunto de valores atribuídos a um conjunto de variáveis de evidência. Os algoritmos usados para fazer o processo de propagação de evidências podem ser divididos em duas categorias: os que fazem inferência exata e os que fazem inferência aproximada [Marques and Dutra 2005].

Para o cálculo de inferências devemos lembrar que uma das propriedades dos processos markovianos é a falta de memória. Desta maneira, se um sistema encontra-se em um estado i em um instante t, os estados futuros não dependem dos estados anteriores, bastando apenas o estado atual. As informações passadas não são relevantes para se fazer uma inferência sobre o futuro.

2.2.6 Inferência Exata

Um algoritmo de inferência denomina-se exato se as probabilidades dos nodos são calculadas sem outro erro senão o de arredondamento, inerente a limitações de cálculo dos computadores.

A inferência exata pode ser utilizada todas as vezes em que a rede pode ser chamada de poli árvore. Uma poli árvore é a rede em que há no máximo um caminho entre um par de nodos. Essas redes são também chamadas de redes simplesmente conectadas. Em estruturas simples, a inferência pode ser feita por meio de cálculos realizados nos próprios nodos ou através da passagem de mensagens entre eles. A inferência exata pode ser computacionalmente intratável em redes muito grandes ou muito conectadas, pois trata-se de um problema NP *hard* [Korb and Nicholson 2004a]. Para as demais redes, é preferível a

utilização da inferência aproximada.

2.2.6.1 Algoritmo de eliminação de variáveis

O algoritmo de eliminação de variáveis coloca várias distribuições num vetor de distribuições. Essas distribuições são chamadas fatores. Coletam-se todos os fatores que contêm um variável $X1$, retira-os do vetor de distribuições, constrói-se uma nova distribuição não-normalizado $P(filhos(X1)|pais(X1), tios(X1))$ e adiciona essa distribuição ao vetor de distribuição. O resultado dessa operação é que X_1 foi eliminado. Depois, sobre a variável de X_2, coletam-se todos os fatores que contêm X_2, retira-os do vetor de distribuições, multiplica as distribuições e elimina-se também X_2. O resultado dessa operação é novamente um fator, que é adicionado ao vetor de distribuições. Continua-se essa operação até se eliminar todas as variáveis possíveis.

No final, restará pelo menos um fator para a variável de consulta X_q. Multiplicando esses fatores juntos e normalizando o resultado, tem-se $P(Xq|e)$. A ideia é multiplicar os membros do vetor "fatores" na sequência dada pela ordenação. É aplicada uma ordem arbitrária para as variáveis, assim optou-se pela ordenação no sentido folha raíz.

Algoritmo 1: Algoritmo de eliminação de variáveis

Entrada: X, e, rb

initialization;
$fatores \leftarrow []$;
$vars \leftarrow REVERTER(VARS[rb])$;
para cada var *em* $vars$ **faça**
  $fatores \leftarrow [CRIAR - FATOR(var, e)fatores]$
  se var *é uma variável oculta* **então**
    $fatores \leftarrow SOMAR(var, fatores)$
  fim se
fim para cada
retorna *Normalizar* $PRODUTO - PONTUAL(fatores)$;

Os parâmetros de entrada são X, que é a variável de consulta, e que um conjunto que contem as variáveis de evidência e rb que é a rede bayesiana.

Segundo [Russell and Norvig 2004] o produto pontual de dois fatores f_1 e f_2, dá origem a um outro fator f cujas variáveis representam a união das variáveis contidas em f_1 e f_2.

2.2.7 Inferência Aproximada

Os algoritmos aproximados utilizam distintas técnicas de simulação para obter valores aproximados das probabilidades

Como não se sabe *a priori* qual será o modelo de rede que usuário irá implementar, preferiu-se construir o *framework* de maneira que se possa utilizar os dois modelos de propagação. Implementou-se o algoritmo Cadeia de Markov Monte Carlo(CMMC) [Russell and Norvig 2004]. A partir de um determinado estado, especificando um valor para uma variável, é possível gerar um próximo estado, uma vez que ele estará condicionado aos valores que estão na cobertura de Markov, isto é, filhos, pais e pais dos filhos.

2.2.7.1 Cobertura de Markov

A Figura 2.4 representa a cobertura de Markov da variável X como uma área mais escura. Desta forma, é possível notar que as variáveis A, B, C, F e G estão na cobertura de Markov da variável X. É possível afirmar ainda que a variável X é condicionalmente independente de todas as outras variáveis, dados A, B, C, F e G, que representam seus pais, seus filhos e os pais de seus filhos. A cobertura de Markov, portanto, representa todos os nodos que sofrerão algum tipo de alteração caso a evidência seja a variável X.

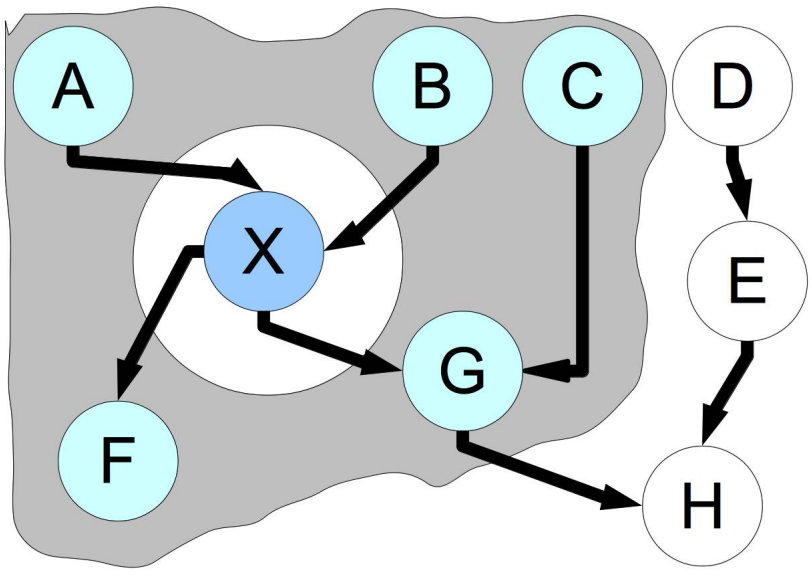

Figura 2.4: Exemplo de cobertura de Markov

2.2.7.2 Algoritmo *Gibbs Sampling*

Este algoritmo é um método usado para gerar uma sequencia de amostras da distribuição de probabilidade conjunta das variáveis aleatórias. O objetivo é conseguir uma amostra da distribuição a posteriori e calcular estimativas amostrais de características desta distribuição. Sua eficiência está no fato de que ele utiliza uma simulação iterativa baseada em cadeia de Markov, isso faz com que os valores gerados sejam dependentes e que haja uma convergência no algoritmo.

O algoritmo começa com uma configuração das variáveis consistente com as evidências, e então troca aleatoriamente o estado das outras variáveis condicionadas à sua cobertura de Markov. Depois é usada essa nova configuração gerada pra trocar os valores das outras variáveis. O Algoritmo 2, retirado de [Russell and Norvig 2004] gera cada evento a partir de uma mudança aleatória no estado precedente. O próximo estado é gerado por uma amostragem aleatória de um valor das variáveis que não são evidência X_i, condicionadas sobre os valores atuais das variáveis na cobertura de Markov de X_i.

Algoritmo 2: Algoritmo CMMC para inferência em rede bayesiana

Entrada: X, e, rb, N
Dados: $N[X]$, z, x
Saída: estimativa de $P(X|e)$
initialization;
adicionar em x valores aleatórios para as variáveis que estão em Z;
para $j \leftarrow 1$ *até* N **faça**
  $N[x] \leftarrow N[x] + 1$
  para cada Z_i *em* Z **faça**
    Fazer amostragem do valor de Z_i em x a partir de $P(Z_i|mb(Z_i))$ dados os valores de $MB(Z_i)$ em x
  fim para cada
fim para
retorna *Normalizar* $N[X]$

Assim como no algoritmo de inferência exata, este algoritmo recebe os valores X, e e rb. Além desses valores, ele usa $N[X]$ que é um vetor de contagens sobre X; z as variáveis não evidência de rb e x que é o estado atual da rede.

2.3 Trabalhos Relacionados

A pesquisa bibliográfica efetuada para o desenvolvimento deste trabalho incidiu sobre os seguintes termos: bayesian networks, redes bayesianas. A pesquisa foi realizada nas seguintes bases de dados: *Medline*, *PubMed*, *Google Scholar*, Scopus e ScienceDirect (editora Elsevier).

Foram encontrados vários trabalhos científicos que relatam o uso de redes bayesianas como mecanismo computacional para aplicação em cenários onde há incertezas. Buscou-se trabalhos que foram publicados a partir de 2005. Além disso, buscou-se também por trabalhos que tivessem utilizado algum dispositivo móvel genérico, como um PDA, Telefone celular aplicados na resolução de problemas relacionados à engenharia biomédica. Os artigos encontrados foram filtrados por tema (incluindo só os da área da saúde, bioinformática e/ou medicina). A relevância de cada artigo foi tomada a partir da análise dos títulos e os resumos.

Em [Saheki 2005] o autor construiu uma rede bayesiana cujo objetivo era o de auxiliar no diagnóstico de doenças cardíacas, especificamente em pessoas que se apresentam em postos de saúde com dispnéia. Foi implementada uma interface gráfica em linguagem Java que utilizava o Java Bayes [Cozman]. Aplicou-se o modelo *noisy-or*, juntamente com uma generalização para nos pais não-binários. A generalização proposta foi aplicada na rede construída e mostrou resultados melhores do que o modelo original, pois diminuiu o número de probabilidades necessárias e pode ser aplicada a uma variedade maior de nós, justamente por não haver a restrição de nós pais binários.

Em outro trabalho, [José et al. 2005], os autores implementaram um módulo que deveria ser executado em um telefone celular. O objetivo deste módulo era realizar o Interrogatório Sintomatológico. A rede bayesiana foi utilizada como mecanismo para que o paciente não ficasse sobrecarregado com várias perguntas. Além disso seria possível conseguir uma economia no consumo da energia da bateria. Os nodos da rede seriam percorridos utilizando um método conhecido como caminhamento em profundidade. Assim, o questionário a ser respondido deveria ser organizado de maneira hierárquica.

Uma série de artigos, dentre os quais [McCann et al. 2006], teve como objetivo promover o entendimento mais amplo sobre redes bayesianas e desta forma torná-la mais uma ferramenta disponível tanto para os pesquisadores quanto para outros profissionais que precisam trabalhar com gerenciamento de recursos. Os autores sugeriram as redes bayesianas como ferramentas úteis para aplicações como essas, pois elas têm capacidade para representar o conhecimento de especialistas de um ecossistema ou permite que sejam avaliados os efeitos de decisões de gerenciamento alternativas. Há uma série de sete passos identificados para a criação de um *framework* que tenha suporte ao aprendizado. A rede modelada utilizou um *software* já pronto, o Netica [Norsys].

Aos autores do trabalho [Sanders and Aronsky 2006] desenvolveram e avaliaram uma rede bayesiana para identificar os pacientes "elegíveis", cujos dados estavam disponíveis eletronicamente no momento da triagem, para passarem por um diretriz (*guideline*) de tratamento de asma. A rede bayesiana modelada foi capaz de detectar com boa precisão esses pacientes, o que sugere que esta técnica poderia ser usada para iniciar automaticamente

o encaminhamento e os cuidados para os pacientes elegíveis.

O artigo [Sales et al. 2010] apresenta a construção de uma rede bayesiana como uma ferramenta para ajudar uma iniciativa de inclusão digital de idosos, realizado na Universidade Federal de Santa Catarina, em seu núcleo de estudos para pessoas da terceira idade que participam de aulas de ciência da computação em um grupo homogêneo quanto à idade. A rede construída auxilia na identificação de alunos que atuarão como multiplicadores de conhecimento. Essa identificação é feita por características do perfil do próprio aluno no início das aulas. A ferramenta de apoio à construção da rede foi o Netica [Norsys]. Foi identificado que o perfil de um idoso-multiplicador como sendo alguém que: tem um lazer cognitivo (o que evidencia persistência e paciência), tem computador em casa, grau de escolaridade em sua maioria acima do fundamental e é ou tem interesse em ser voluntário.

2.4 Conclusão

Este capítulo conceitou o que são redes bayesianas. Apresentou as equações matemáticas que as representam. Por meio de exemplos tentou-se facilitar a compreensão sobre nodos, estados sobre as relações de causa e efeito que existem entre os nodos. Atenção especial foi dada ao processo de criação da rede bayesiana, onde o algoritmo proposto em [Lucas et al. 2004, Korb and Nicholson 2004b], foi apresentado.

As redes Bayesianas utilizam dos conceitos de mapas causais, para modelar domínios. Mapas causais descrevem as variáveis (nós ou nodos) e as relações de causa e efeito entre elas, na forma de um grafo acíclico direcionado. A intensidade das relações é dada pelas tabelas de probabilidade condicional de cada variável, que quantifica as probabilidades de ocorrência de um evento, dado os valores seus pais. O cálculo das probabilidades é obtido com a aplicação do teorema de Bayes, a partir das probabilidades *a priori*, adquiridas com o auxílio de um especialista ou através de um banco de dados.

A construção das Tabelas de probabilidades condicionais (TPC) foi abordada de maneira que se pudesse compreender o quão difícil pode se tornar essa tarefa quando há vários nodos com muitos estados envolvidos. Nesta atividade, é importante que se possa contar com algum auxilio para facilitar o preenchimento das tabelas. As distribuições canônicas têm esse objetivo. Foram apresentadas duas das formas mais comuns: o nodo determinístico e o ou ruidoso.

A inferência é o processo de fazer com que as evidências coletadas possam ser propagadas pela rede de forma que se possam obter conclusões. Esse é o principal mecanismo da rede bayesiana. Há duas formas de propagação de evidências: a exata e a aproximada que

são utilizadas conforme a topologia do modelo.

O trabalhos relacionados mostram como outros autores utilizaram a rede bayesiana para encontrar soluções em cenários diversos.

No capítulo seguinte serão feitas considerações sobre o projeto e implementação do *software*, onde os conceitos que foram apresentados nesse serão usados.

Capítulo 3

Projeto e implementação do *framework*

3.1 Resumo

Este capitulo tem o objetivo de apresentar e discutir os aspectos envolvidos no projeto e implementação do *framework*. A metodologia utilizada para o seu desenvolvimento é apresentada, assim com os aspectos estáticos e dinâmicos dos modelos de objeto. Os requisitos funcionais, não funcionais e de domínio serão elencados. Também será mostrada uma arquitetura genérica de *software*, projetada para que o *framework* seja utilizado em aplicações para a plataforma iOS. Os principais elementos que a compõe, bem como a maneira que estes elementos interagem entre si estão destacadas. Há ainda alguns trechos do código-fonte que serão destacados de forma que se possa mostrar como os conceitos discutidos nos capítulos anteriores foram implementados.

3.2 Metodologia usada no desenvolvimento

Na análise de requisitos orientada ao objeto, as entidades do mundo real, que estão dentro do escopo do problema a ser resolvido, são modeladas sob a forma de classes. Porém, para que se possa implementar classes é necessário que se conheça quem são essas classes, o que elas têm que fazer e como se relacionam durante a resolução de um problema. A fase de projeto é responsável por levantar essas informações. Ela permite que o processo de codificação de um software, tal qual uma planta baixa em um processo de engenharia civil, siga essas informações de maneira que se possa conseguir entregar um produto que esteja de acordo com suas especificações e com um menor risco de falhas.

Para o desenvolvimento do projeto, foi utilizada a metodologia do desenvolvimento incremental. Conforme [Sommerville et al. 2007], o desenvolvimento incremental é baseado na ideia de criar uma implementação inicial, expô-la a criticas e comentários dos usuários e continuar o processo por meio da implementação de outras versões até que o *software* esteja

de acordo com seus objetivos. Essa metodologia contém etapas de especificação, desenvolvimento e validação, que são executadas de maneira intercalada e há um rápido *feedback* entre essas etapas. A Figura 3.1 demonstra essa metodologia.

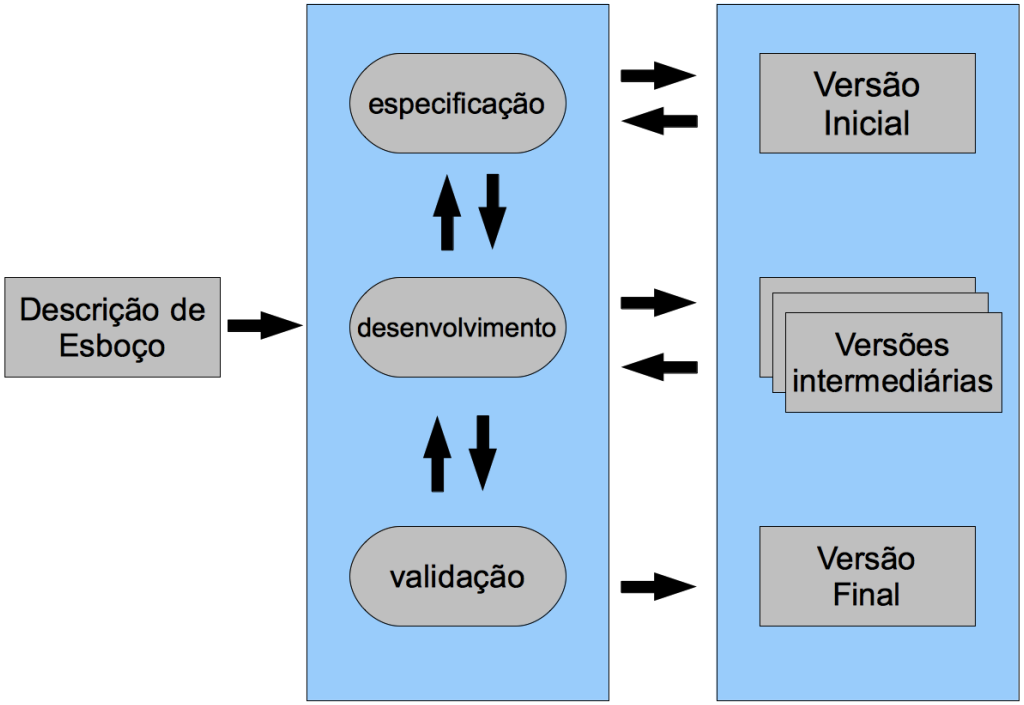

Figura 3.1: Desenvolvimento incremental

O ponto de partida é um esboço inicial do que o sistema deve realizar. Este esboço pode ser obtido por meio de descrições de alto nível colhidas junto ao usuários do *software*. Para este trabalho, usou-se como esboço as ideias apresentadas no Capítulo 2. Desta forma, o produto a ser construído deve ser capaz de realizar o processo de modelagem de uma rede bayesiana, como o mostrado na Figura 3.2 por meio de um diagrama de atividades. Estes diagramas têm o objetivo de mostrar as atividades que compõe um processo e o fluxo de controle de uma atividade para outra, representando a modelagem de aspectos dinâmicos do sistema. Eles lembram os fluxogramas, mas os diagramas de atividade suportam a representação de comportamento paralelo.

A notação gráfica utilizada é a seguinte: O início de um processo é indicado por um círculo preenchido; o fim é mostrado com um círculo preenchido dentro de outro. Os retângulos com cantos arredondados são as atividades. Eles representam processos menores específicos que devem ser realizados. O comportamento condicional é representado por losango. Sempre que ele ocorre, apenas um dos fluxos de saída poderá ser seguido. Uma intercalação, ou junção marca o fim de um comportamento condicional. Ele tem vários fluxos de entrada e apenas uma saída.

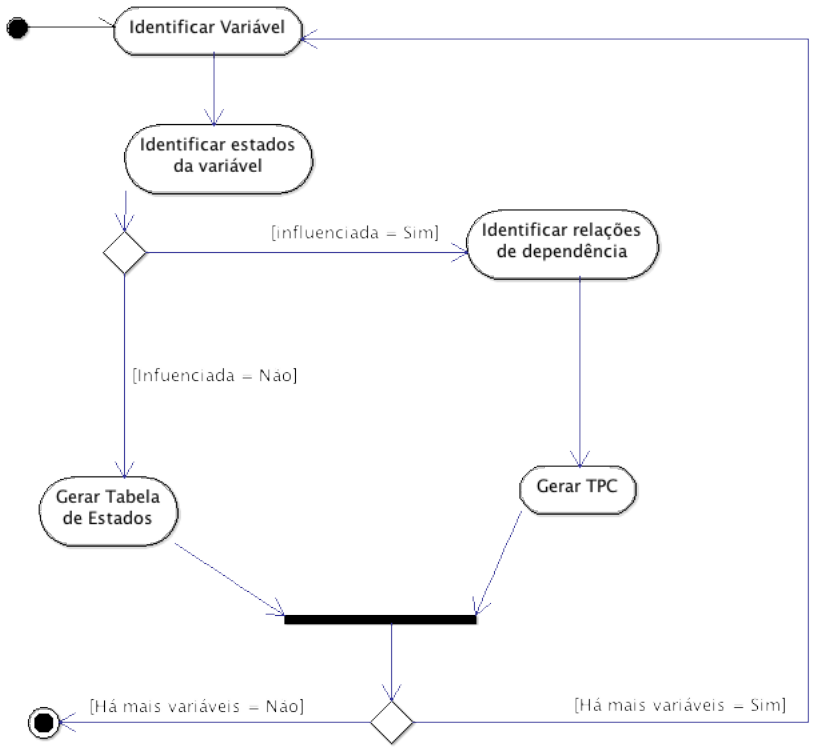

Figura 3.2: Diagrama de Atividades para a construção de uma aplicação de Rede Bayesiana

Baseado no que foi apresentado anteriormente, a Figura 3.2 mostra que o trabalho de construção de uma rede bayesiana começa com a identificação de uma variável que seja importante. Deve-se aplicar o princípio da abstração para que se consiga concentrar apenas nos valores que realmente sejam indispensáveis ao modelo. Identifica-se então os estados dessa variável. Aqui é preciso tomar uma decisão: caso essa variável seja influenciada por alguma outra, é necessário identificar as relações de dependência. Assim é necessário estabelecer qual a probabilidade de a variável que está sendo modelada tem de estar em um determinado estado, dado que a variável que a influencia está em um estado. Isso fará com que tenhamos que gerar a TPC. No caso de não haver influência, ou seja, de estarmos modelando um nodo raiz, apenas as crenças, isto é, a tabela com valores pre definidos para os estados, deve ser implementada.

3.3 Esboço Inicial

Os passos mostrados pela Figura 3.2, são seguidos para que se tenha uma rede bayesiana qualquer, independente da quantidade de nodos que ela possua. Temos assim um arcabouço simples que permite a descrição de forma genérica de como uma rede é criada. Desta forma, quando imagina-se uma ferramenta que seja capaz de implementar uma destas estruturas

deve basear-se no mesmo princípio. Ela deve ser capaz de construir várias redes bayesianas com configurações diferentes de maneira que se possa atender diversas aplicações, para tanto, é necessário que o *software* seja projetado de maneira que suas classes e seus componentes internos colaborem de maneira que sua arquitetura seja reutilizável. Essa ideia, descrita por Shmidt *et al.* conforme [Sommerville et al. 2007], é uma definição para o *framework*.

Os *frameworks* devem fornecer um conjunto básico de mecanismos genéricos que representem o domínio por meio de um projeto abstrato. Como trata-se de uma aplicação orientada ao objeto, este domínio é representado por um conjunto de classes e por definições de como as instancias dessas classes podem colaborar umas com as outras durante a realização de alguma tarefa. Desta forma, é possível observar que um dos pontos fortes no uso de *framework* é o reuso. Para que se pudesse criar o software proposto neste trabalho, foi necessário o uso de outros artefatos que ofereciam as mesmas características, tais como uma arquitetura reutilizável e a capacidade de permitir a criação de outras soluções.

O esboço inicial para o desenvolvimento do software foi o seguinte: devia-se construir um software que fosse capaz de criar e manipular redes bayesianas, sendo que por manipulação entendeu-se que era o provimento de mecanismos para a alteração da topologia da rede, remoção de nodos e realização de inferências. Para a criação da rede, as atividades mostradas na Figura 3.2 devem ser contempladas. Sendo assim, os nodo deviam manter relações de dependência entre eles. A Figura 3.3 mostra um modelo de contexto, onde está representada a fronteira do sistema, o que está dentro dele, as funções fornecidas pelo *software*, a partir da percepção do usuário, por isso não há detalhes sobre como as funcionalidades, mostradas como elipses, serão implementadas.

Uma pergunta que deve ser respondida: quem será o usuário do *software*? Essa questão é importante, pois pode fazer com que alguns requisitos, tais como a linguagem usada na interface sejam diferente de acordo com o perfil do usuário. Neste trabalho usou-se o seguinte perfil para ele: deverá possuir conhecimentos mínimos sobre o ambiente a ser modelado. Além disso, para que se possa conseguir uma melhor utilização do *software*, ele deverá ter conhecimentos básicos sobre redes bayesianas, isto é, deve ser capaz de modela-la e altera-la quando for necessário, além de saber interpretar os resultados obtidos e gerados a partir de inferências.

3.4 Especificação dos requisitos do sistema

O desenvolvimento do *software* teve inicio a partir do levantamento de requisitos. Os requisitos podem ser entendidos como descrições em alto nível, representando de maneira

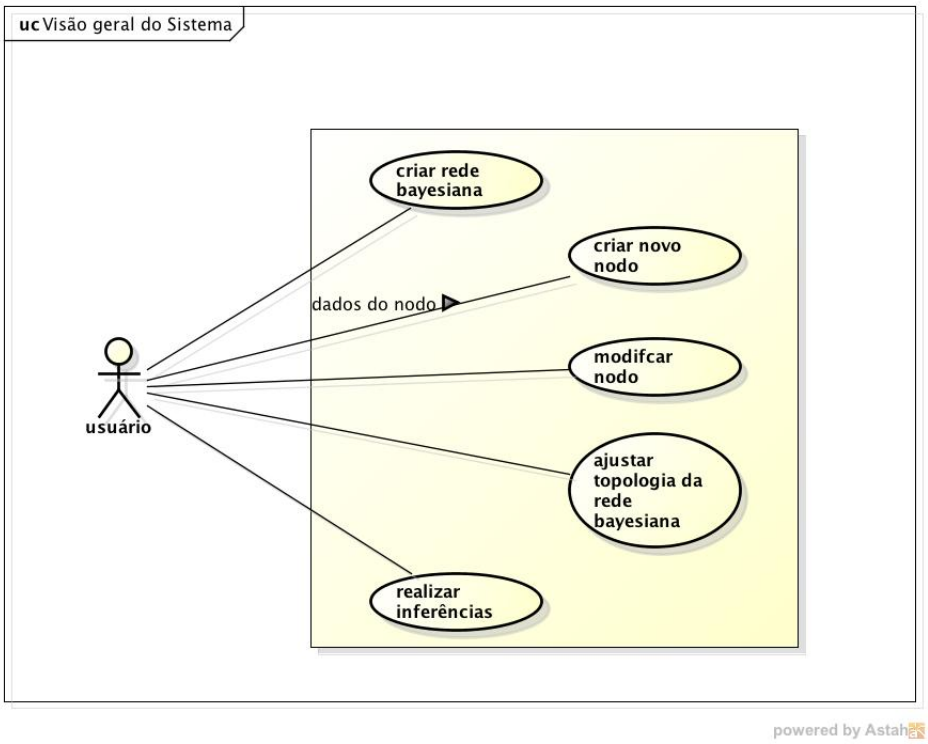

Figura 3.3: Visão geral do sistema

informal quais são as necessidades que devem ser supridas pelo novo produto, bem como o comportamento esperado para ele. De acordo com [Sommerville et al. 2007], os requisitos de um sistema são descrições dos serviços que serão executados por um *software*. Eles refletem ainda suas restrições operacionais e as necessidades dos usuários. São classificados em funcionais, não funcionais e de domínio.

- Requisitos Funcionais – São os serviços que o sistema deve fornecer, como o sistema deve reagir à entrada de dados e como deve comportar-se em determinadas situações, de maneira resumida eles representam todas as funcionalidades executadas pelo sistema.
- Requisitos não funcionais – São as restrições sob as quais as funcionalidades oferecidas pelo sistema estão sujeitas, definido as características e limitações da aplicação.
- Requisito de domínio – São derivados do domínio da aplicação, refletem suas características e restrições.

O levantamento de requisitos foi feito inicialmente, a partir da análise de outros *softwares*. Neste passo inicial, foi possível observar quais as funcionalidades que eram oferecidas por eles. Além disso, técnicas como diagramas de casos de uso, cenários e diagramas de atividades foram utilizados como ferramentas para elicitação dos requisitos. Como exemplo, a Figura 3.4, que é um diagrama de atividade, mostra as ações que devem ser executadas quando o usuário quiser criar um nodo novo. É importante frisar que a análise deste diagrama permitiu que fossem acrescentados detalhes ao nodo, como por exemplo, o um nome.

Esse nome serviria como um identificador único e assim, não seria possível que em uma mesma rede dois nodos tivessem o mesmo nome. Outro fator importante é notar a ordem em que as atividades devem ser executadas: em primeiro lugar deve ser criado um nome para o nodo; em seguida os estados do nodo devem ser inseridos, para isso cada estado deve ter um nome e o valor de crença deve ser atribuído. Esses dois passos devem ser repetidos até que não existam mais estados para serem inseridos. O terceiro passo está ligado à estrutura da rede, pois se o nodo for uma raiz da rede, isto é, não tiver nodos ancestrais o trabalho está concluído, porém, se ele for um nodo descendente, sua tabela de probabilidades condicionais (TPC) deverá ser gerada e ajustada. Cada uma das atividades desse diagrama devem ser implementadas para que se possa construir essa funcionalidade e disponibiliza-la para o usuário.

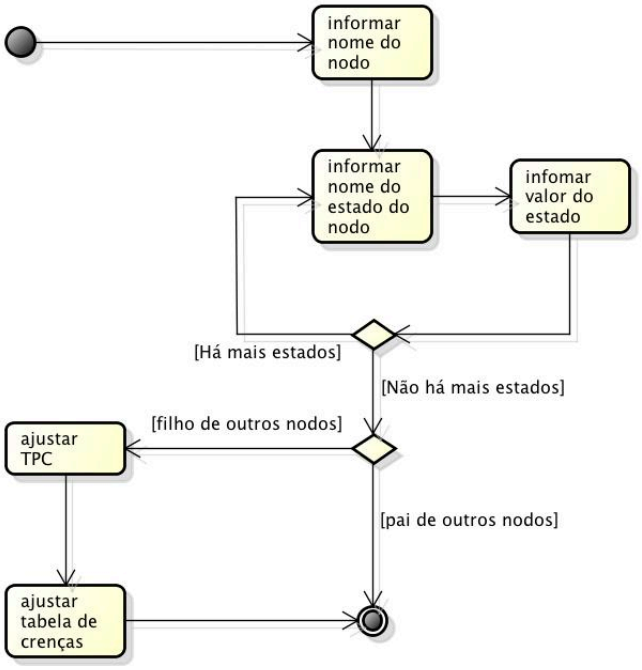

Figura 3.4: Diagrama de atividades - funcionalidade criar novo nodo

Assim, chegou-se a seguinte relação de requisitos:

- requisitos funcionais: Permitir que sejam construídas Redes Bayesianas no próprio *tablet*. Mecanismos para a criação, edição e exclusão de nodos, estados e probabilidades devem ser criados. Os nodos podem ser organizados por meio da interface *touch screen*. As redes geradas pelo usuário podem ser salvas e recuperadas para uso posterior. Deve ser possível realizar a propagação de evidências e exportar a rede modelada como uma figura.

- requisitos não funcionais: todas as funcionalidades devem consumir o mínimo de recursos possíveis do equipamento e oferecer um bom desempenho ao usuário, garantindo satisfação em seu uso. A carga da bateria deve ser maximizada. Evitar que a entrada de dados dependa do teclado virtual, para tanto, a interface deve apresentar

informações que possam ser selecionadas. As redes devem ser salvas em arquivos com formato XML para facilitar sua exportação. Usar como padrão para exportação de imagens o formato JPG.

- Requisitos de domínio: A aplicação deve ser implementada seguindo o modelo *MVC*, que é, segundo [Gamma et al. 2000], um padrão de projeto composto. Utilização da plataforma *iOS* e do ambiente *Cocoa*. É necessário que sejam seguidas as recomendações do documento *iOS Human Interface Guidelines* [Inc. 2010c] em relação à criação da interface.

Desta maneira, verificou-se que o projeto deveria ter duas partes distintas, que deveriam funcionar em conjunto :

- Uma parte responsável por controlar as regras de negócio, que corresponderia ao *Model* do padrão MVC. Seria necessária a criação de mecanismos que fossem capazes de fazer toda a manipulação das redes bayesianas, mas que pudessem oferecer o recurso do reuso, sendo que esse artefato pudesse ser usado em vários sistemas operacionais, independentemente de qual tecnologia fosse utilizada na apresentação dos dados e interação com o usuário. A linguagem utilizada, o *Objective-C* é uma linguagem que oferece este recurso. Esta parte representa o *framework* proposto. Neste trabalho, de agora em diante, sempre a chamaremos de *framework*

- Uma parte que seria responsável por disponibilizar uma interface para que o usuário pudesse utilizar os artefatos produzidos na criação das redes bayesianas. Neste trabalho, esta parte será chamada de aplicação. Ela foi implementada para que sua execução ocorresse em um dispositivo móvel do tipo *tablet*.

As próximas seções trarão mais detalhes para cada uma destas partes, onde serão apresentadas informações sobre os projeto e a implementação de cada uma delas.

3.5 Desenvolvimento do *software*

3.6 Projeto da *aplicação*

Para que o *framework* pudesse ser de fato testado em um dispositivo móvel, lançando mão de sua interface *touch screen*, imaginou-se uma aplicação que deveria ser construída de forma que o seu uso permitisse a construção de Redes Bayesianas, com isso, fez com que fosse necessário construir uma interface gráfica com o usuário. A partir de alguns requisitos encontrados, tal como a necessidade de seguir o modelo MVC, a arquitetura do sistema pode ser definida.

As regras sobre o *design* das interfaces dos aplicativos é definida no *iPhone Human Interface Guide*[Inc. 2010c], que é um extenso documento distribuído pela *Apple* que serve como um guia para o desenvolvimento de aplicativos, ou simplesmente App. As orientações deste documento devem ser seguidas à risca sob pena de não ser possível a disponibilização de um aplicativo na loja virtual. Este documento divide as aplicações em três tipos:

- Aplicativos de produtividade – Gerencia informações e ajuda a complementar tarefas. Em geral, sua interface organiza as informações de maneira hierárquica fazendo com que a navegação passe de um nível para o outro. Um exemplo são os aplicativos usados para enviar e receber *emails*.
- Aplicativos utilitários – Disponibiliza um conjunto de informações específico para o usuário.
- Aplicativos imersivos – Têm uma interface altamente customizada que facilita a interação com o aparelho. Um exemplo desta categoria são os jogos ou ainda um aplicativo que simula um nível, que usa os sensores presentes no equipamento de forma que seja possível averiguar se ele está posicionado sobre uma superfície plana.

É importante identificar o tipo de aplicativo que se pretende construir, pois para cada um dos tipos elencados anteriormente, há um conjunto de classes, fornecido pelo *Cocoa*, cuja utilização é mais adequada. A aplicação a ser implementada será do tipo utilitário.

Em face a essa necessidade, a aplicação terá uma arquitetura em camadas de forma que em cada uma dessas camadas implemente um dos subsistemas definidos no modelo MVC. Em [Sommerville et al. 2007] o modelo em camadas é descrito como sendo um modelo de máquina abstrata, onde cada camada oferece um conjunto específico de serviços. A Figura 3.5 mostra como o sistema está projetado.

Quando se está construindo um *software*, é importante garantir que as regras de negócio sejam criadas de maneira que elas atuem em conjunto com a interface com o usuário. Eles devem formar um conjunto, mas de forma que cada uma das partes tenha suas funcionalidades bem definidas, que seja possível ao desenvolvedor identificar cada uma destas partes, mas que cada uma dessas partes possam ser corrigidas ou melhoradas individualmente. Por exemplo, caso seja necessária a inclusão de um novo algoritmo de propagação de evidências, a alteração deverá ocorrer apenas no *framework*, mais especificamente na classe "implementacao", que será apresentada posteriormente. Da mesma maneira, se for necessário melhorar a usabilidade da interface, esta mudança deve ser feita na parte responsável pela exibição dos dados. Desta maneira busca-se criar um ligação mínima (baixo acoplamento) entre essas partes. Para que se alcance esse objetivo, aplicou-se o modelo MVC, que foi descrito anteriormente. Desta forma, a arquitetura de uma aplicação com interface gráfica e que reutilizará as classes do *framework* terá uma estrutura coma a

mostrada na Figura 3.5.

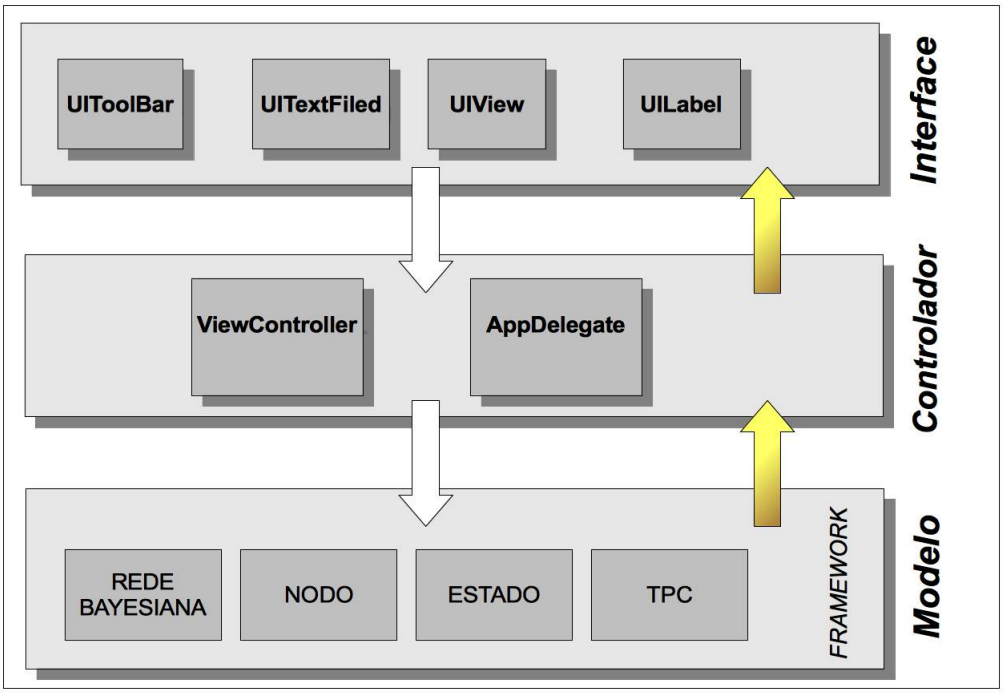

Figura 3.5: Arquitetura da aplicação

A arquitetura mostra uma visão geral do sistema, sendo usada para a especificação de componentes e compreende duas atividades:

- definir a estrutura do sistema, que é representada por um diagrama de blocos onde mostra a organização dos componentes
- definir seu comportamento, que é composto de duas atividades: casos de uso e definições a partir de uma máquina de estados.

As setas mostram a interação que existe entre as camadas da aplicação. As classes da camada de objetos, podem relacionar-se apenas com classes da camada de controle, não podendo interagir com os objetos da camada de interface diretamente. As classes *ViewController* e *AppDelegate*, por exemplo, podem interagir tanto com os objetos da camada de interface quanto com os objetos da camada de objetos. Sua função é a de possibilitar que as informações disponíveis nas classes redeBayesiana e nodo, por exemplo possam chegar até a interface gráfica da aplicação.

3.6.1 Arquitetura da aplicação

A arquitetura de software é baseada no princípio da abstração, onde abstrai-se detalhes de um sistema de forma a identificar melhor suas propriedades, desta forma, um sistema

complexo pode ter vários níveis de abstração cada um com sua própria arquitetura. Uma arquitetura se preocupa exclusivamente com o comportamento e interação entre os diversos elementos que a compõem, ignorando detalhes internos específicos de cada um destes elementos. Cada elemento, por sua vez, pode ser composto por outras arquiteturas de forma a atender ao comportamento exigido pela arquitetura do nível superior (externa a este elemento). Esta sequência de abstrações pode ser aplicada até que não seja mais possível decompor um elemento.

3.6.2 Projeto da interface da aplicação

A interface com o usuário, deverá responder a entrada de dados via *touch screen*. A interface deve ser limpa, com a menor quantidade de ícones possíveis. A Figura 3.6 mostra um exemplo de como pretende-se que a interface seja implementada, ela deve permitir que a grande maioria das operações tire proveito da interface *touch screen* do equipamento.

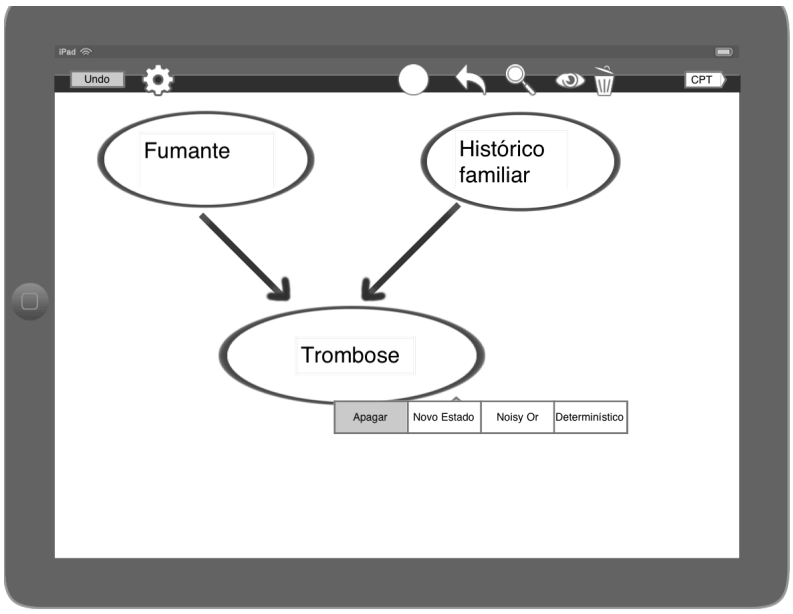

Figura 3.6: Exemplo da interface da aplicação

Para que um novo nodo seja criado, é necessário que o usuário escolha o ícone que implementa essa funcionalidade. No caso, há um circulo branco na barra de ferramentas que é esse ícone. Ao arrastá-lo para a parte branca, que é a área de modelagem, surgirá uma elipse. Quando o o usuário deixa de tocar na tela, aparecerá uma janela e o teclado virtual para que o nome do nodo seja inserido.

A interface terá ícones para que o usuário possa criar um novo nodo, criar um nova relação entre dois nodos, além disso um botão deslisante controlará o estado da rede: em edição ou em produção. O estado edição possibilita que a topologia da rede possa ser

alterada, que nomes de estados de nodos possam ser mudados, o *lay-out* da rede torne-se mais organizado, arrastando os nodos pela interface o que permitirá a alteração dos valores das tabelas de probabilidades condicionais. O comportamento do *software* quando o usuário solicitar a criação de um novo nodo é mostrado na Figura 3.7. Este exemplo de diagrama de sequencia é usado para modelar o comportamento dos objetos que criarão um resposta a um estímulo vindo do usuário.

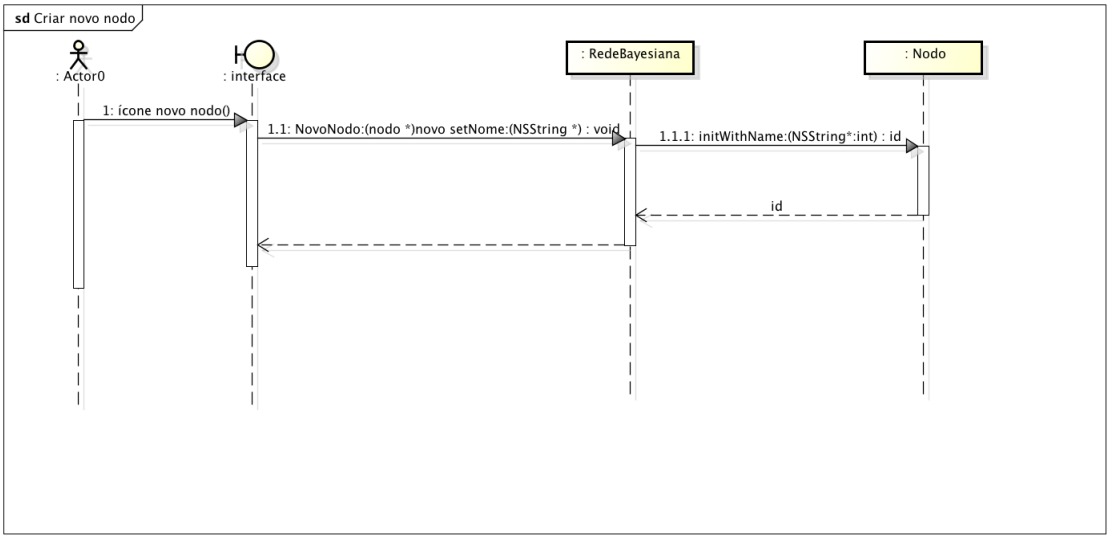

Figura 3.7: Sequência de interação entre as classes do sistema na criação de um novo nodo

A camada de interface implementa a *View*, ficando responsável por responder aos estímulos do usuário e a camada de objetos implementa o *Model*, assim o *framework* estará rodando aqui, criando instâncias de nodos, estados, calculando probabilidades. A camada de requisições fica responsável por ligar a interface e os objetos, tal como o *Controler*. Essa camada trata, por exemplo, de janelas, menus, botões, fontes e outras entidades da interface propriamente dita. Normalmente, utiliza-se algum *framework* ou biblioteca de classes para a construção de interface, como por exemplo, MFC, MacApp, ou Java Servlets. Essa camada não realiza nenhum cálculo, nem acessa diretamente as classes “redeBayesiana” ou “nodo”, por exemplo. Ela executa tão somente o processamento típico da interface, por exemplo, o preenchimento de campos obrigatórios e a navegabilidade entre os formulários (*views*).

3.6.3 Implementação da camada de objetos

A Figura 3.8 mostra o modelo de classes que será implementa na camada de objetos. A Rede Bayesiana é composta por nodos, que por sua vez possuem estados. Além disso cada nodo tem associado a ele uma tabela de probabilidade conjunta. A camada de objetos implementa o *framework* proposto. Na aplicação, o usuário não irá interagir como ela. Caso ele deseje criar um novo nodo, ele irá pressionar um ícone na interface. Essa requisição

será encaminha pela camada de requisições até a camada de objetos, onde será gerada uma solicitação para atendê-la. Alguns detalhes referentes à camada de objetos podem ser vistos na seção referente à implementação do *framework*.

A classe *UIViewControler* disponibiliza o modelo fundamental de gerenciamento de view para todas as aplicações para iOS. Normalmente, o programador não precisa instanciar objetos desta classe diretamente, ao invés disto, são instanciadas algumas de suas subclasses para a realização de tarefas específicas. Ela gerencia um conjunto de *views* que fazem parte da interface do aplicativo. Como parte da camada *controller*, ele coordena sus ações com os objetos da camada *model* e outros controladores de *view*.

3.6.4 Projeto do *framework*

O aumento tanto em tamanho e quanto em complexidade dos sistemas a serem desenvolvidos, torna as tarefas de especificação e de projeto tão importantes quanto a escolha de algoritmos e de estruturas de dados. A abordagem orientada a objetos para todo o processo de desenvolvimento de software, de acordo com Sommerville *et al.* [Sommerville et al. 2007], é de uso comum atualmente. Essa abordagem baseia-se em vários conceitos, dentre os quais podemos citar dois que foram os mais importantes: o encapsulamento, que faz com que uma mesma entidade, denominada objeto, contenha atributos, destinados ao armazenamento de valores, e operações, destinadas à manipulação destes atributos que são inacessíveis aos demais objetos.

O projeto de *software* orientado ao objeto faz com que todos os elementos que sejam necessários para a sua implementação sejam identificados como classes. Para isso devem ser considerados os aspectos mais importantes do domínio do problema e assim sendo, abstrair tudo aquilo que não é indispensável. Desta forma, foram consideradas as seguintes classes:

- Rede Bayesiana – É a principal classe do modelo. É a partir dela que se podem criar os nodos e realizar a propagação das evidências.
- Nodo – Essa classe representa as variáveis aleatórias que serão criadas para modelar um domínio qualquer.
- Tpc – A tabela de probabilidade condicional representa a relação existente entre os estados do nodo combinados com os estados dos nodos pais.
- Estado – Representam os valores que as variáveis contém.

Uma visão geral das classes que implementam o *framework* é apresentada na Figura 3.8. Ele é composto por classes que permitem a criação das redes bayesianas. Além disso disso, há classes que implementam os nodos, as tabelas de probabilidades condicionais, os estados

e os mecanismos para propagação das evidências. Por meio de um diagrama de classes, que é uma maneira para mostrar um conjunto de classes e seus relacionamentos, é possível ver como essas classes relacionam-se para a modelagem de um domínio qualquer. As classes "inferencia", "inferenciaExata" e "inferenciaAproximada" foram incluídas para que se pudesse criar formas diferentes para a implementação da propagação das evidências. Elas são uma aplicação direta dos padrões de projeto e representam uma necessidade de facilitar a codificação do *software*. A seção 3.6.4.1 trará mais informações sobre essas classes.

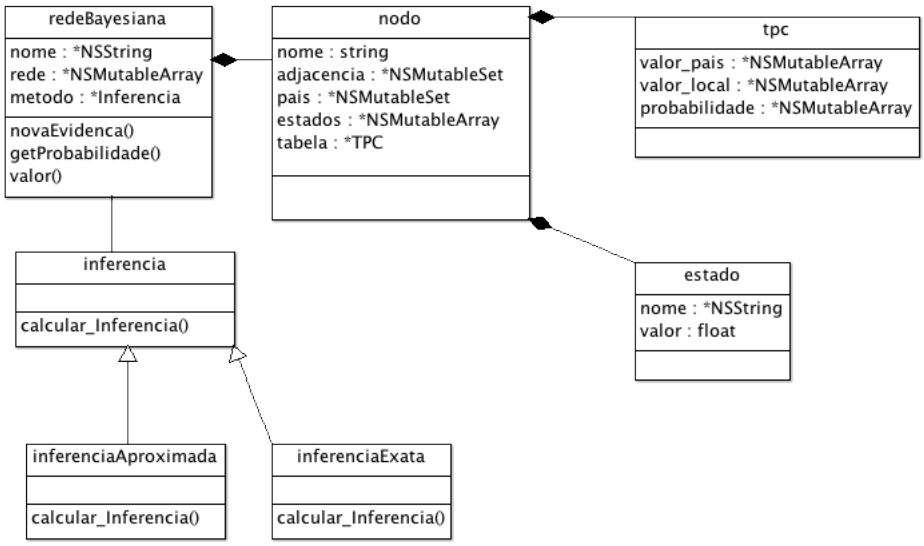

Figura 3.8: Classes implementadas no *framework*

A Figura 3.8 mostra o modelo de dados proposto. Nesta figura são destacadas a principais entidades modeladas: a rede bayesiana, o nodo, o estado a TPC (Tabela de Probabilidade Condicional) e a inferência. Podemos verificar a presença dos diferentes tipos de relacionamentos encontradas em Diagramas de Classe. São eles: a herança, a agregação, composição e a associação. Qualquer domínio que se queira representar, usará essas classes. O mecanismo da herança permitirá que todas as caraterísticas dessas classes possam ser reutilizadas posteriormente.

A entidade inicial é a própria rede bayesiana. Ela é composta por um conjunto de nodos. Os nodos, por sua vez, são compostos por um conjunto de estados e possuem, ligado a cada um deles, uma TPC. Os losangos mostrados na figura indicam essa relação "todo-parte" entre as entidades. Esta relação não permite que um nodo exista sem que ele esteja contido em uma rede bayesiana, ou que um estado não esteja associado a um nodo, a por exemplo.

Uma característica importante do modelo projetado é que ele pode ser usado para modelar vários tipos de aplicações para domínios diferentes. Omitindo-se alguns detalhes, o exemplo mostrado na Figura 2.2, seria modelado com único objeto da classe "redeBayesi-

ana" e sete objetos da classe "nodo". Cada nodo terá seus estados e sua tabela de probabilidade. Desta forma, podemos representar esse domínio com o diagrama de classes mostrado na Figura 3.8

3.6.4.1 Uso de padrões de projeto

A propagação de evidências foi implementada. Conforme discutido em [16], há dois grupos de algoritmos que são usados para executar o processo de propagação de evidências: os que fazem inferência exata e os que fazem inferência aproximada. Como a inferência pode ser implementada usando duas abordagens diferentes, sendo que a escolha sobre qual a maneira a ser utilizada ficará a cargo do usuário do *framework*, a maneira encontrada para que se pudesse facilitar a implementação desta característica, foi a utilização do padrão de projeto chamado *Strategy*[Gamma et al. 2000]. Sua utilização nesse estudo, foi particularmente interessante pois permitiu que fosse definida uma família de algoritmos, encapsular cada um deles, e faze-los intercambiaveis.

No *framework* proposto a classe de contexto é "redeBayesiana". Ela possui um atributo chamado "Metodo", cujo tipo de dados cria uma instancia da classe "inferencia". No momento em que uma nova rede é criada, é necessário escolher uma das estratégias concretas.

A linguagem *objective-c* oferece um recurso para facilitar este tipo de implementação chamado de *protocol*. Segundo a descrição disponível em [Apple 2011], os protocolos permitem que sejam criados métodos que podem ser implementados como classes. O uso dos protocolos é útil em várias situações, porém vale destacar que ele foi importante para que se pudesse declarar métodos que seriam implementados posteriormente, tendo uma funcionalidade semelhante ao da *interface* existente na linguagem java. A próxima seção mostrará trechos de código em que é implementado o padrão *strategy* usando *protocol*.

3.6.5 Implementação do *framework*

Uma abordagem orientada a objetos para todo o processo de desenvolvimento de *software*, segundo [Sommerville et al. 2007], é de uso comum atualmente, principalmente em sistemas interativos, como o proposto neste trabalho. Essa abordagem faz com que os requisitos do sistema tenham que ser expressados usando um modelo de objetos e o projeto a implementação devem ser feitos, preferencialmente, usando objetos. Como a linguagem de programação usada, *Objective C*, é orientada ao objeto, a adoção desta abordagem foi adequada. A Figura 3.8, que é um modelo de objeto, mostra como as entidades Rede Bayesiana, Nodo, Estado e Tpc foram modeladas.

O *framework* implementa várias funcionalidades dentre elas, podem ser citadas: impedir que sejam criados ciclos; construção das tabelas de probabilidade condicionais (tpc) dos nodos à medida em que as relações entre os nodos estejam estabelecidos; cálculo da probabilidade conjunta total; atualização dos valores dos estados a partir das relações estabelecidas nas tpc; permite a geração de valores para as TPC por meio das distribuições canônicas ou-ruidoso e determinístico. As próximas seções mostrarão como algumas dessas funcionalidades foram implementadas usando a linguagem *objective-c.*

3.6.5.1 Exemplo de implementação de uma classe

Quando observamos o *framework*, o uso da herança é importante já na implementação de suas classes iniciais. A linguagem *objective-C* possui uma classe genérica que é ancestral de todas outras e que cria os mecanismos mínimos para que uma classe seja construída. Seu nome é *NSobject*. Todas as classes criadas nesse trabalho são descendentes dela, uma vez que ela é a superclasse da linguagem. Como exemplo, a Figura 3.9, mostra um trecho do código fonte que é responsável pela implementação da classe "redeBayesiana". A linha 1 mostra a relação de herança que há entre a classe mais geral, isto é a superclasse *NSObject* e a classe "redeBayesiana".

```
1 @interface RedeBayesiana : NSObject
2 {
3         NSString *Nome;
4         NSMutableArray *Rede;
5         id<inferencia> Metodo;
6 }
```

Figura 3.9: Trecho do código-fonte onde há um exemplo de uso de herança entre classes.

As demais linhas criam os atributos da classe "Nome", "Rede" e "Metodo". Com relação a eles podemos observar o seguinte:

- O atributo "Nome" é responsável por armazenar uma sequencia de caracteres, que representará um nome para a rede que está sendo criada.
- "Rede" armazenará em uma estrutura de dados dinâmica, uma vez que ela poderá aumentar e diminuir de tamanho durante o tempo de execução. A classe "NSMutable-Array", definido pela biblioteca *Fountation*, implementa mecanismos para inclusão, recuperação e deleção de objetos em estruturas dinâmicas do tipo lista. Por estruturas dinâmicas, entende-se que a quantidade de dados armazenados pode aumentar ou diminuir ao logo da execução do *software*. O asterisco que aparece antes de 'Rede" indica que este atributo é do tipo ponteiro. Um tipo de dado ponteiro, armazena um endereço de memória. Assim, o atributo não terá a informação guardada dentro de sua

estrutura. Ele tem uma referência que aponta, ou indica, em que lugar na memória, os dados estão. Na Figura3.10, o atributo rede tem é um ponteiro, assim ele guarda o endereço da memória onde está o NSMutableArray. Cada uma das partições desta estrutura é capaz de guardar um endereço de memória de objeto da classe nodo.

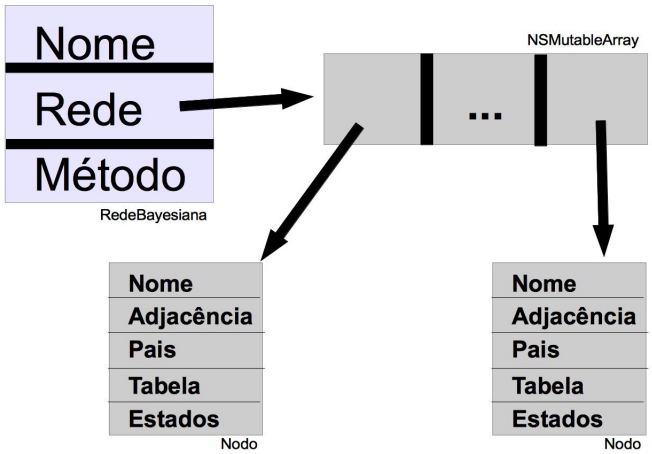

Figura 3.10: Ponteiros - exemplo de uso na implementação

- O atributo "Metodo',' é responsável por executar uma operação. Cumpre ressaltar que seu tipo de dados é diferente dos demais, *id*. Este tipo de dado armazena uma referência, um endereço de memória, a um objeto de qualquer tipo. Desta forma, ele armazenará uma única referência a um objeto que pertence à classe "inferencia". Este mecanismo é a base para a implementação do padrão *strategy*, que foi discutido anteriormente, pois quando for necessário usar os métodos definidos intercambiáveis, uma mensagem será enviada à uma das classes que implementam as estratégias concretas.

A listagem da Figura 3.11 mostra como o protocolo "inferencia" foi implementado. Sua codificação é um pouco diferente das demais classes, uma vez que começa com "@protocol". O protocolo criado herda características de "NSObject" e possui um único método chamado de "calcula_inferencia". Este método será implementado em classes que são descendentes de "inferencia". A palavra reservada "@required" indica que todas as classes descendentes terão que implementar esse método.

```
1 @protocol inferencia <NSObject>
2
3 @required
4 -(NSNumber*) calcula_inferencia;
5
6 @end
```

Figura 3.11: Código-fonte que cria o protocolo "inferencia".

3.6.5.2 Método para verificar a formação de ciclos na rede

O código mostrado na Figura 3.12 mostra como a operação "formaCiclo", da classe "nodo" foi implementada. Todas as vezes em que for preciso inserir um novo nodo na rede essa operação é executada. Desta forma é possível impedir que a rede tenha um ciclo, que contraria a definição de rede bayesiana. Essa rotina é executada de maneira recursiva, isto é, sua execução cria uma nova execução da mesma operação até que se encontre uma resposta. Na linha 3 há um teste que verifica se o nodo que pretende ser inserido na rede, chamado de nodo raiz, está contido na lista de filhos. Caso isso não ocorra, para cada um dos nodos filhos será feita a mesma busca, onde o nodo raiz será procurado na lista de filhos de cada um deles. Esse procedimento pode ser repetido nos filhos dos filhos e outros descendentes. As linhas número 4, 10 e 12 contem as possíveis respostas, que podem ser:

- *YES* – caso o novo nodo, já tenha o chamado "nodo_raiz" em sua descendência. O nodo "nodo_raiz" representa o pai do novo nodo, desta forma ele não pode ser filho do novo nodo. Portanto, a inclusão desse novo nodo formará um ciclo na rede.
- *achou* – uma nova busca será feita, agora para cada nodo existente na descendência do nodo novo. Assim poderão ser verificadas situações em que um descendente dele, seja seu ancestral. Para isso uma nova execução desta função será criada, alterando seus valores.
- *NO* – tendo sido verificadas cada uma das alternativas anteriores e não obtendo um resposta positiva, a única resposta que se pode obter é que a inclusão desse novo nodo não formará ciclos.

```
1  −(BOOL) formaCiclo:(nodo ∗)nodo_raiz grafoinicial:(NSMutableSet∗) filhos;
2  {
3          if ([filhos containsObject:nodo_raiz])
4                  return YES;
5          else
6          {
7                  for (nodo ∗ filho in filhos)
8                  {
9                          BOOL achou = [nodo_raiz formaCiclo:nodo_raiz grafoinicial:[filho adjacencia]];
10                         return achou;
11                 }
12         return NO;
13         }
14 }
```

Figura 3.12: Código-fonte da operação responsável por evitar que se formem ciclos

3.6.5.3 implementação de métodos para preenchimento de tabelas

O algoritmo da Figura 3.13, constrói a tabela de probabilidade conjunta para um nodo informado como parâmetro. Essa tabela será composta por três estruturas menores: uma

delas contem a relação de estados dos nodos pai, outra contem a referencia para os estados do nodo e a última é a probabilidade de cada estado do nodo, dados os valores dos estados dos pais.

Dado um nodo qualquer, obtem-se um dos pais desse nodo. Toma-se um dos estados dele e guarda-o em tEstadosPais. Caso existam mais pais, o algoritmo é executado novamente, pegando-se um novo pai e um novo estado. Se estados de todos os pais tiverem sido armazenados, calcula-se o valor da probabilidade padrão, dividindo-se 1 pela quantidade de dados em tEstadosPais.

Outra funcionalidade do algoritmo, é que seu uso preenche de maneira automática a probabilidade com um valor uniforme. Isso é particularmente importante, pois inicialmente as probabilidades podem não estar disponíveis. Desta forma, o preenchimento automático permite que o processo de construção da rede prossiga. Quando os valores corretos estiverem disponíveis, eles podem substituir aqueles gerados por esse algoritmo.

Há alguns tipos de distribuições de probabilidade condicionais podem ser aproximada por modelos de interação canônicos que exigem menos parâmetros e que reduzem o esforço de construção das tabelas. O framework implementou dois modelos: nodo determinístico e ou-ruidoso.

O nodo determinístico é usado quando um nodo possui vários nodos pais, isto é, há várias causas para um único efeito e cada uma dessas várias causas é suficiente para fazer com que na ausência das outras o efeito tenha um valor conhecido. A listagem mostrada na Figura 3.14 implementa uma operação que modela em uma TPC uma relação do tipo nodo determinístico. Ela recebe como parâmetros o valor de um estado, pertencente a uma variável. Além disso, o valor do estado que representa a causa comum nos pais, também é informado juntamente com o valor que deverá ser atribuído a cada uma das ocorrências de “nomeEstado”.

A linha 12 verifica se o nome de um estado que está no conjunto de estados presente na tabela do nodo é igual ao estado passado como parâmetro. Caso o teste seja verdadeiro, a tabela será atualizada com o valor do parâmetro probabilidade. Cumpre lembrar que no caso da distribuição determinística, basta apenas uma ocorrência do estado para fazer com que a linha da tabela tenha o valor determinado, indicado pelo parâmetro “valor”. Caso não seja encontrado nenhuma ocorrência, o valor atribuído será igual a um menos “valor”.

Como exemplo, imagine uma relação em que há três nodos: “câncer”; “tuberculose” são pais de “cancer ou tuberculose” que representa a ocorrência positiva de pelo menos um dos pais, como mostrado na Tabela 3.1.

```
1  −(void) geraLinhasTabela:(nodo ∗)nNodo token:(NSMutableArray ∗)tToken indice:(int ∗) i linha:(int∗) l;
2  {
3          static int estado = 0;
4          nodo∗ nPai = (nodo ∗)[nNodo pegaPainoIndice:∗i];
5          for (Estado ∗nEstado in nPai.estados)
6          {
7                  [tToken addObject:nEstado];
8                  if (((∗i)+1)!=[nNodo.pais count])
9                  {
10                         (∗i)++;
11                     [nNodo geraLinhasTabela:nNodo token:tToken indice:i linha:l];
12                         [tToken removeLastObject]; // Remove o estado anterior que havia sido inserido
13                 }
14                 else
15                 {
16                         NSNumber∗ probabilidade = [NSNumber numberWithFloat:((float)1/[nNodo.
                               estados count])];
17                     NSMutableArray ∗Copia = [[NSMutableArray alloc] initWithArray:[tToken
                           mutableCopy]];
18                         [nNodo.tabela.valor_pais insertObject:Copia atIndex:(∗l)];
19                         NSMutableArray ∗tProbabilidadeApriori = [[NSMutableArray alloc]
                               initWithCapacity:[nNodo.estados count]];
20                         int x=0;
21                         for (x=0;x<[nNodo.estados count];x++)
22                                 [tProbabilidadeApriori addObject:probabilidade];
23                         [nNodo.tabela.valor_local insertObject:nNodo.estados atIndex:(∗l)];
24                         [nNodo.tabela.probabilidade insertObject:tProbabilidadeApriori atIndex:(∗l)];
25                         estado++;
26                         (∗l)++;
27                         [tToken removeLastObject]; // remove o estado anterior
28                         [Copia release];
29                 }
30         }
31         (∗i)−−;
32         estado−−;
33 }
```

Figura 3.13: Operação responsável por gerar entradas iniciais para a TPC

Desta forma, ao invés de fazer com que o usuário tenha que digitar as quatro linhas da tabela, ele pode preenche-la usando o método da seguinte forma : [cancerTuberculose aplicarDeterministico:@"Verdadeiro"estadoPai:@"Verdadeiro"valor:1.0. Independentemente da quantidade de nodos pai, uma única instrução construirá a tabela do nodo filho e dessa forma pode ser obtida uma facilidade maior durante a modelagem da uma rede.

A listagem 3.15 ou mostra a operação que implementa a distribuição ou ruidoso (*Noisy-or*). Ela recebe dois parâmetros. O primeiro deles é um vetor que é composto por uma ou várias instâncias de um tupla que é composta pelas informações nodo, probabilidade . O segundo deles é o valor que o estado dos pais deverá ter para que o valor atribuído ao parâmetro probabilidade, informado na tupla, seja utilizado no processamento. Este valor é importante pois ele permite que se possa usar relações como “verdadeiro” / “falso”, “ligado” / “desligado”.

```
1  −(void) aplicarDeterministico:(NSString ∗) nomeEstado estadoPai:(NSString ∗) nomePai valor:(float)
       probabilidade
2  {
3          NSMutableArray∗ valorPais = nil;
4          int x,quant;
5
6          for (x=0;x<[self.tabela.valor_local count];x++)
7          {
8                  quant=0;
9                  valorPais = [self.tabela.valor_pais objectAtIndex:x];
10                 for (Estado ∗pEstado in valorPais)
11                 {
12                         if ([pEstado.nome isEqualToString:nomePai]==YES)
13                         {
14                                 [self setTpc:x estado:nomeEstado probabilidade:probabilidade];
15                                 quant++;
16                                 break;
17                         }
18                 }
19                 if (quant==0)
20                 {
21                                 [self setTpc:x estado:nomeEstado probabilidade:1−probabilidade];
22                                 quant=0;
23                 }
24         }
25 }
```

Figura 3.14: Método que aplica o nodo determinístico

Tabela 3.1: TPC para o nodo Raio_X

P(câncer ou tuberculose | Tuberculose,Cancer)		Tuberculose	Câncer
Sim	Não		
1.00	0.00	Sim	Sim
1.00	0.00	Sim	Não
1.00	0.00	Não	Sim
0.00	1.00	Não	Não

A linha 6 cria uma enumeração, isto é, agrupa os dados criando um pequeno conjunto. Neste caso, a partir dos valores contidos na estrutura tabela, agrupa-se o campo “valor_pais” no grupo “valorPais”. É possível acessar todos os elementos do grupo, o que nesse caso foi feito, utilizando uma estrutura de repetição. Há uma outra enumeração na linha 9 que agrupara os valores informados.

A linha 10 verifica se na tabela do nodo, existe o valor informado. Caso isso ocorra, a variável probabilidade será atualizada. Após todas as atualizações a tabela do nodo é atualizada (linha 11) e passa-se para o próximo valor em “valorPais”.

```
1  −(void) aplicarOuRuidoso:(NSMutableArray ∗)parametros valor:(NSString∗)valor
2  {
3          int x;
4          float probabilidade;
5          x=0;
6          for (NSMutableArray∗ valorPais in self.tabela.valor_pais)
7          {
8                  probabilidade = 1.0;
9                  for (parametro ∗dado in parametros)
10                         if ([valorPais containsObject:[[dado nodo] getEstado:valor]]==YES)
11                                 probabilidade = probabilidade ∗ [dado valor];
12                         [self setTpc:x estado:valor probabilidade:probabilidade];
13                 x++;
14         }
15 }
```

Figura 3.15: Método que aplica a distribuição ou ruidoso

3.7 Conclusão

Esta capitulo mostrou como os mecanismos fornecidos pela orientação ao objeto e as idéias vindas dos padrões de projetos contribuíram na implementação do *framework*. A linguagem *objetive-c*, que é uma linguagem orientada ao objeto, implementa, portanto, conceitos como herança e encapsulamento. Além disso disponibiliza o mecanismo *protocol*, que permitiu a aplicação do padrão de projeto *strategy*. Esse padrão foi importante no projeto do *framework*, pois facilitou a modelagem de várias formas de algoritmos de inferência.

O *framework* é composto por sete classes “redeBayesiana”, “nodo”, “estado”, “tpc”, “inferencia”, “inferenciaExata” e “inferenciaAproximada” e sua implementação visa permitir que ele possa ser utilizado em aplicações em modo interativo ou interagindo com uma interface gráfica. Neste contexto, essa aplicação possui uma arquitetura em camadas, sendo o *framework* a camada mais interna.

Foram apresentadas algumas funcionalidades implementadas de forma que se pudesse conhecer a adequação do *framework* aos requisitos funcionais que foram apresentados.

Foi apresentada a arquitetura de uma aplicação, baseada no modelo MVC, que será usada para testar o *framework*. Essa aplicação será usada nos testes mostrados no capítulo seguinte.

O capítulo seguinte dessa dissertação apresentará a tecnologia que será empregada para a resolução do problema proposto. Além disso, os conceitos de orientação ao objeto e padrões de projeto são expostos de maneira que seu uso na implementação do *framework*, seja entendido.

Capítulo 4

Ferramental Tecnológico Utilizado

4.1 Resumo

Este capitulo tem o objetivo de apresentar os aspectos tecnológicos utilizados na elaboração do projeto. São apresentadas as ferramentas utilizadas no desenvolvimento: a linguagem de programação chamada Objective-C, o *framework cocoa* sistema operacional e ambiente integrado de desenvolvimento. Além disso, são expostos brevemente conceitos sobre orientação ao objeto, padrões de projetos e modelagem de sistemas. Esses conceitos foram aplicados no capítulo 3 que trata sobre o projeto e implementação do *software*.

4.2 Materiais e Métodos

O *software* deverá ser executado em um iPad, por isso a plataforma de desenvolvimento deverá seguir um modelo estabelecido pela *Apple Computer Inc.*. O desenvolvimento precisa ser feito em um computador *Macintosh*, com o sistema operacional Mac OS X 10.5, ou superior. O *software* requerido para o desenvolvimento, está disponível gratuitamente e é composto de duas partes: o *Xcode*, que é um ambiente integrado de desenvolvimento e o iOS SDK, que contém as bibliotecas usadas para a implementação. Um simulador, chamado de *iPhone Simulator*, fornecido pelo SDK, será usado nas etapas iniciais do trabalho. O desempenho do *software* executado no simulador e no dispositivo real é diferente, conforme visto em [Pilone and Pilone 2011]. A Figura 4.2 mostra o Xcode e o simulador sendo executado.

É necessário que todo *software* escrito para um dispositivo móvel seja o mais conciso possível, fazendo com que o desenvolvedor mantenha-se focado naquilo que é indispensável, de forma que os recursos disponíveis no equipamento sejam capazes de executá-lo de forma ágil sem comprometer a carga da bateria. De acordo com [Inc. 2010a], o primeiro iPad, modelo que será usado em testes reais de uso, possui 256 Mb de memória RAM e um

processador com 1Ghz de *clock*. Em [Pilone and Pilone 2011] são elencadas algumas das principais diferenças entre o desenvolvimento para iOS e para microcomputadores:

- Os dispositivos que utilizam iOS têm tela pequena, se comparados a outros dispositivos, e são focados na tarefa que o usuário está executando.
- Memória e *CPU* têm capacidade reduzida se comparada com a dos micros.

O principal motivo que contribuiu para a escolha da plataforma proprietária fornecida pela *Apple* em detrimento à plataforma gratuita oferecida pela *Google*, o *Android* [Google], foi o fato de que, conforme [Bergström and Engvall 2011], não há uma padronização em relação a geração de interfaces. O *Android*, que vem sendo adotado como sistema operacional por vários fabricantes, é usado em dispositivos com telas cujos tamanhos variam de 2,7 até 10,1 polegadas. Além disso os diversos fabricantes usam processadores diferentes, com velocidades diferentes. Apesar de possibilitar o surgimento de vários produtos, para os desenvolvedores pode tornar-se um pouco difícil garantir uma total compatibilidade entre os aparelhos, sendo que nem todas as versões do *Android* estão disponíveis a todos os equipamentos. Este problema de compatibilidade não existe na plataforma da *Apple*.

4.2.1 Plataforma *Cocoa*

Cocoa é um ambiente que atende o desenvolvimento de aplicações tanto para o sistema operacional Mac OS X quanto para o iOS, sendo o primeiro destinado a microcomputadores pessoais (desktops e notebooks) e o segundo é usado em dispositivos móveis tais como iPad, iPhone e iPod. Ele consiste em um conjunto de bibliotecas de *software* orientadas ao objeto, um sistema de *runtime* e um ambiente integrado de desenvolvimento. Há algumas diferenças entre as bibliotecas que são usadas no desenvolvimento para dispositivos móveis e os demais, porém elas compartilham o *framework Foundation*. Não serão tratadas essas diferenças nesse trabalho.

A Figura 4.1, mostra como é a estrutura do *Cocoa Touch*, que é o *framework* responsável por suportar o desenvolvimento para o iOS. Essa estrutura é composta por camadas que cobrem as funções básicas do sistema operacional, Core OS até um conjunto de frameworks para aplicações. Há camadas intermediárias responsáveis por serviço essenciais e manipulação da interface gráfica.

- *Core OS* – Aqui estão contidos o *kernel*, o sistemas de arquivos, a infraestrutura de rede segurança, gerenciamento de energia além de *drivers* para dispositivos. Há também o suporte às especificações POSIX/BSD 4.4, por meio da biblioteca *libsystem*.
- *Core Services* – Os *frameworks* desta camada provêm serviços básicos tais como manipulação de *strings*, gerenciamento de *collections*, rede, utilitários URL, gerenciamento

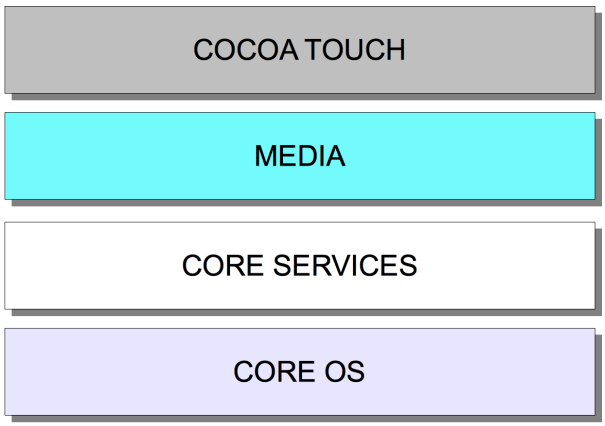

Figura 4.1: Arquitetura da plataforma *cocoa*

de contatos e as preferências do usuário. Fornece suporte ao GPS, bússola, acelerômetro e giroscópio. Nesta camada estão as bibliotecas *Foundation* e *Core Foundation* que têm abstrações para tipos comuns de dados tais como *strings*. Contem ainda o *Core Data* que é um *framework* responsável por fazer a persistência de objetos.

- *Media* – os frameworks e serviços desta camada dependem da camada *Core Services* e disponibiliza serviços gráficos e de multimídia para a camada *Cocoa Touch*. Eles incluem *Core Graphics*, *Core Text*, *OpenGL ES*, *Core Animation*, *AVFoundation*, *Core Audio*, além da reprodução de vídeos.

- *Cocoa Touch* – Os *frameworks* desta camada dão suporte a aplicações baseadas no *iOS*. Eles incluem frameowrks tais como Game Kit e Map Kit. Esta camada e a *Core Services* possuem *frameworks* que são muito importante para o desenvolvimento de aplicações para o iOS, que são o *UIKit*, que disponibiliza os objetos mostrados por uma aplicação em sua interface e define o seu comportamento , e o *foundation* que cria o comportamento básico dos objetos, estabelece os mecanismos para seu gerenciamento. Além disso, provê objetos para tipos de dados primitivos e serviços do sistema operacional. Ele é uma versão orientada ao objeto do *Core Foundation*.

4.2.2 Ambiente de desenvolvimento e linguagem de programação

Quando são utilizadas apenas ferramentas fornecidas pela *Apple*, isto é seu SDK e a linguagem de programação *Objective-C*, é gerada a chamada aplicação nativa para iOS.

Contudo, há alternativas ao uso da linguagem *Objective-C*, pois a desenvolvedores que argumentam que como apenas a Apple adota o Objective-C como linguagem de programação isso dificulta a portabilidade das aplicações. Além disso, o gerenciamento de memória feito por essa linguagem é mais próximo daquele encontrado na linguagem C, do

que em linguagens modernas, tais como C#, Ruby e Java, o que dificulta o trabalho para os desenvolvedores. Essas alternativas podem ser agrupadas em duas categorias distintas. Uma delas representam ferramentas que permitem aos desenvolvedores escrever aplicativos em outras linguagens de programação e por meio de uma compilação que serve de ponte entre a linguagem original e os *frameworks Cocoa Touch* permitem que essas aplicações sejam executadas no tablet. A segunda categoria é composta por ferramentas que permitem aos desenvolvedores usar a linguagem de marcação HTML em conjunto com folhas de estilo, CSS e a linguagem Javascript, juntas elas produzem um código-fonte parecido com os dos aplicativos para web que é preparado de forma que sua execução faça com que ele se pareça com um aplicativo nativo.

A Tabela 4.1 exibe uma breve comparação entre algumas dessas ferramentas. As características que são elencadas, além de seus nomes, são: a linguagem de programação principal usada por cada uma delas; utilizando essa ferramenta, quais são as plataformas para as quais os aplicativos podem ser gerados; o desenvolvedor deverá usar uma destas plataformas como ambiente de desenvolvimento e o preço do *software*.

Tabela 4.1: Comparação entre ferramentas de desenvolvimento para iOS

Ferramenta	Linguagem	Gera aplicativos para	Disponível nas plataformas	Preço
Cocoa Touch	Objective-C	Mac OS, iOS	Mac OS	Gratuita
RubyMotion	Ruby	iOS	Mac OS	R$ 435,37
Corona	Lua	iOS, Android	Mac OS, Windows	US$ 399,00
Mono Touch	C#, .Net	Android, iOS	Mac OS	US$ 399,00
PhoneGap	HTML5, CSS3, JavaScript	Android, Blackbarry, iOS, Symbian, Windows Phone	Windows, Mac OS, Linux	Gratuita
Titanium Mobile	HMTL5, CSS3, Javascript, Ruby, Rails, Python, PHP	Android, iOS	Windows, Mac OS, Linux	US$ 399,00

É importante destacar que o desenvolvimento de aplicativos para o iOS necessitam do uso do *Cocoa Touch*. Se um desenvolvedor usar uma outra ferramenta que faça essa interface com ele, há uma dependência adicional no processo de implementação do *software*, que é a própria interface, uma vez que podem existir alguns recursos que não sejam alcançados por essa interface. Além disso, novos recursos disponibilizados por versões mais recentes do sistema operacional podem não estar à disposição em um período breve de tempo. Desta forma, há razões para acreditar que as aplicações resultantes podem não ser tão eficientes,

tanto em performance quanto em uso de recursos computacionais. Além disso, o aproveitamento de novas funcionalidades disponibilizadas pelo sistema operacional iOS pode demorar, pois deve-se esperar que essas ferramentas forneçam mecanismos para acessá-las e isso pode demorar, prejudicando o desenvolvimento.

Outro fator que influenciou na escolha pela linguagem *Objective-C* foi o fato que ela está tornando-se uma linguagem cada vez mais popular. A popularidade de uma linguagem é um fator importante, pois quanto maior ela for, mais fácil é para encontrar outros desenvolvedores, livros, foruns ou outras fontes de pesquisa e locais onde sejam possíveis a troca de experiências. Há um mecanismo chamado de *TIOBE Programming Community index*[TIOBE 2012], que é um indicador a respeito da popularidade das linguagens de programação. Para que isso seja possível é gerado um índice baseado em informações como a quantidade de cursos oferecidos. Além disso mecanismos populares de busca da internet, como *Google*, *Bing*, *Yahoo!*, *Wikipedia*, *Amazon*, *Youtube* e *Baidu* também são consultados para gerar esse indice, que muda mensalmente. Segundo o último resultado, apresentado durante a construção deste capítulo, a linguagem *Objective-C* ocupa a terceira posição. Por mais que os resultados sejam questionáveis, é importante verificar que essa ferramenta não fica restrita a um número muito pequeno de desenvolvedores.

O Xcode possui *templates* para iniciar o desenvolvimento de uma aplicação, dependendo da aplicação, o *template* pode ser usado completamente ou apenas em parte. Além disso, ele é responsável por manter os recursos do projeto, organizando os vários arquivos necessários para a implementação em diretórios adequados. Há ainda ferramentas para controle de versão e ele oferece suporte a várias linguagens, tais como Java, Php, Python, Pascal. A Apple descreve em [Inc. 2010b] que as aplicações devem ser escritas preferencialmente em *Objective-C*, mas que as linguagens *C* e *C++* também podem ser utilizadas. Em [Aaron 2008], a linguagem *Objective-C* é definida como sendo uma extensão da linguagem C, assim como C++. Esta linguagem é fracamente tipada e orientada ao objeto. O compilador *GNU*, o gcc, pode ser usado para compilar códigos escritos em *Objective-C*, a curva de aprendizagem, para aqueles que já tem familiaridade com orientação ao objeto, não é muito ingrime, conforme [Kochan 2011]. Nesse trabalho, será utilizada a linguagem padrão, portanto o *Objective-C*.

A parte mais importante do SDK é o Simulador de aplicações iOS (*iOS Simulator application*). Essa aplicação apresenta a interface do dispositivo móvel escolhido, seja um celular ou um tablet em uma janela no computador do desenvolvedor. São disponibilizadas várias maneiras de interação, seja via *mouse* ou teclado que simulam cliques, rotação do dispositivo entre outros recursos. Ele permite que as aplicações para os dispositivos móveis sejam testadas em um computador. Esse ambiente permite que alguns erros possam ser encontrados e corrigidos. Além disso é possível testar e projetar a interface da aplicação e medir seu consumo de recursos. O iOS SDK usa bibliotecas que compõem o *Cocoa*, que é um conjunto

de *frameworks* que disponibiliza um ambiente *runtime* para aplicações que usam os sistemas Mac OS X e iOS. Apesar de estarem disponíveis várias bibliotecas, nesse trabalho duas serão usadas: *Foundation* e *UIkit*.

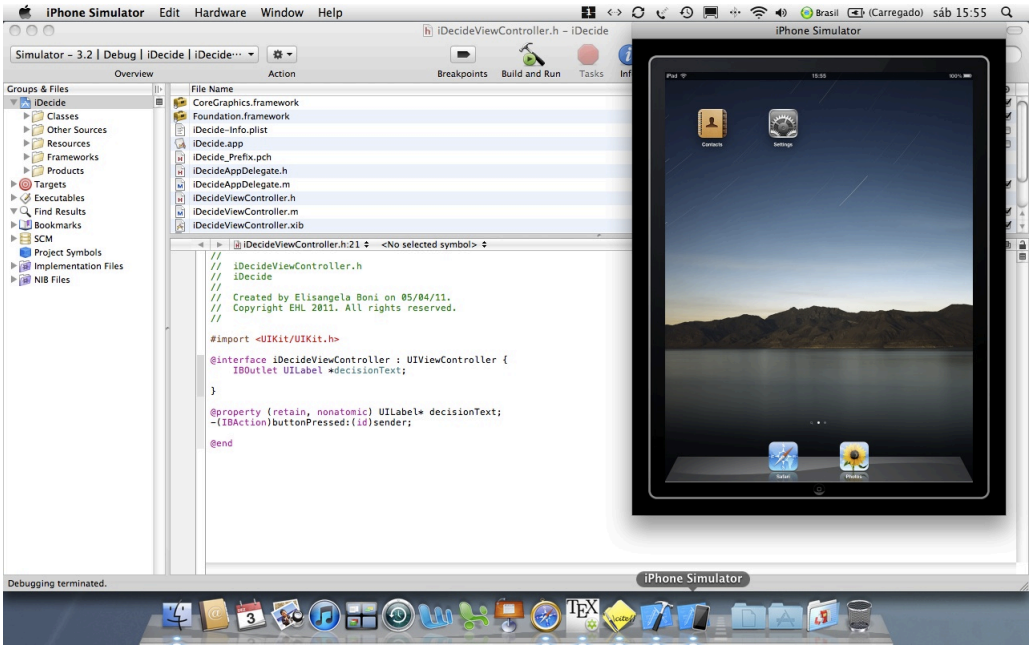

Figura 4.2: Xcode e *iOS Simulator application* em execução

4.3 Arquitetura de *software*

Conforme [Pressman 2006], um estilo arquitetural é um mecanismo usado como um gabarito para a construção de um sistema. Ele estabelece uma estrutura comum para todos os componentes do sistema. Cada estilo descreve uma categoria de sistemas que abrange um conjunto de componentes, um conjunto de conectores, restrições que definem como os componentes podem ser integrados para formar o sistema e modelos semânticos que possibilitam aos projetistas entender as propriedades gerais de um sistema pela análise das propriedades conhecidas de suas partes constitutivas. Segundo [Mendes 2002], uma arquitetura de *software* de sistema mostra a estrutura dos componentes de um sistema, apresenta seus inter-relacionamentos. Ela guia o projeto e sua evolução.

O Estilo arquitetural utilizado na implementação do sistema foi a de arquitetura em camadas. Esse modelo tem como principal característica a existência de uma hierarquia entre suas camadas, fazendo com que uma camada dependa somente de funcionalidades que estão disponíveis na camada imediatamente abaixo dela. As camadas mais altas têm um nível de abstração maior, representando especificidades da aplicação, enquanto as camadas mais baixas, que podem ser utilizáveis por camadas mais altas, têm um propósito mais generalista. Segundo [Pressman 2006], a camada mais externa serve as operações de

interface com o usuário. Na camada mais interna os componentes realizam a interface com o sistema operacional e as camadas intermediárias fornecem serviços utilitários e funções do *software*. O modelo OSI para redes de computadores é um exemplo de aplicação desta arquitetura. Cada uma das camadas é responsável por um aspecto único da comunicação, a partir dos detalhes da transmissão dos bits pelo meio físico (camada mais baixa) até o nível da aplicação (camada mais baixa).

De acordo com [Shaw and Garlan 1994], a arquitetura em camadas têm algumas propriedades importantes, dentre elas:

- suporte à implementação incremental, o que permite aos desenvolvedores particionar a resolução de um problema complexo em uma sequencia de vários passos. A implementação começa em um nível de abstração mais baixo, a partir das camadas mais internas. A cada iteração, o nível de abstração vai aumentando.
- melhor suporte a melhorias e alterações, pois uma vez que cada camada interage somente com suas camadas adjacentes, as mudanças em uma delas devem afetar no máximo outras duas.
- suporte ao reuso, uma vez que diferentes implementações de uma mesma camada podem ser usadas, desde que sejam mantidas as mesmas interfaces para as camadas adjacentes

Segundo [Barbosa 2008], este modelo arquitetural possibilita que a transparência seja aumentada de forma incremental, promovendo a portabilidade, isto é, facilitando modificações, até mesmo, do próprio *hardware*.

4.4 Orientação ao objeto

Segundo Rumbaugh et al.[Booch et al. 2006], modelar e projetor baseado em objetos é uma estratégia para que sejam estudados problemas por meio da utilização de modelos fundamentados em conceitos e ideias que são percebíveis no mundo real. A estrutura básica é o objeto, que combina a estrutura, isto é seus atributos e o comportamento, suas ações ou operações, dos dados em uma única entidade. Ainda segundo Rumbaugh et al.[Booch et al. 2006], um objeto pode ser qualquer entidade que tenha sentido ou importância dentro da representação de um domínio qualquer. Todos os objetos possuem identidade e são distinguíveis, por exemplo, duas canecas feitas com o mesmo material, com a mesma cor e tamanho são objetos diferentes. Na análise orientada a objetos, segundo [Martin and Odell 1994], o tipo de objeto é uma noção conceitual, que especifica uma família de objetos sem estipular como o tipo e o objeto são implementados. Os objetos

que possuem características em comum, isto é, atributos e operações são agrupados em classes. As classes determinam, portanto, a estrutura interna dos objetos.

Os atributos representam as características ou informações à respeito de um objeto, por exemplo, o objeto camiseta, possui como atributos: tamanho, cor, fabricante e modelo. Ainda de acordo com Rumbaugh et al.[Booch et al. 2006], diferentes instâncias podem ter valores iguais ou diferentes para um dado atributo. Com mostrado na Figura 4.3, “nome“, “rede“ e “metodo“, são atributos da classes “redeBayesiana”

Uma operação é uma função ou transformação que pode ser aplicada a objetos ou por estes a uma classe. Assim, conforme mostra a Figura 4.3, “novaEvidencia“ e “getProbabilidade“ são exemplos de operações da classe “redeBayesiana“. A mesma operação pode ser aplicada a classes diferentes. Uma operação assim é dita polimórfica, isto é, a mesma operação pode tomar formas diferentes em classes diferentes. Um método é a implementação de uma operação em uma classe.

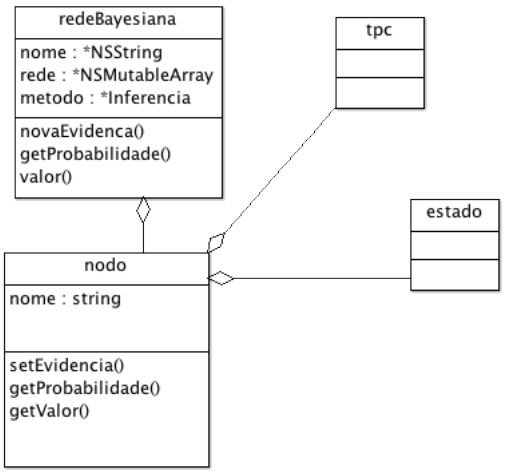

Figura 4.3: Classes implementadas no *framework*

O ato de unir ao mesmo tempo dados e métodos em uma única entidade é denominado encapsulamento. O objeto oculta seus dados de outros objetos e permite que os dados sejam acessados por intermédio de seus próprios métodos. O encapsulamento também oculta os detalhes de sua implementação interna aos usuários do objeto, aumentando assim o nível de abstração no desenvolvimento dos sistemas tempo-real embarcados. Os usuários entendem quais operações do objeto podem ser solicitadas, mas não precisam conhecer os detalhes de como a operação é executada. O encapsulamento permite que as implementações do objeto sejam modificadas sem exigir que os aplicativos que as usam sejam também modificados [Martin and Odell 1994].

Ligações ou associações são os meios para estabelecer relacionamentos entre objetos

e classes. Uma ligação é uma conexão física ou conceitual entre instâncias de objetos. Matematicamente, uma ligação é definida como uma tupla, isto é, uma lista ordenada de instâncias de objetos. Essas ligações podem estabelecer relações do tipo “parte-todo“ (agregação) ou do tipo “é-um-tipo-de“ (herança), dentre outras.

A agregação é o relacionamento “parte-todo“ ou “uma-parte-de“ no qual os objetos que representam os componentes são associados a um objeto que representa a estrutura inteira. A propriedade mais significativa da agregação é a transitividade, isto é, se A faz parte de B e B faz parte de C, então A faz parte de C. A agregação é também, anti- simétrica, isto é, se A faz parte de B então B não faz parte de A, conforme encontrado em [Booch et al. 2006].

A herança é um mecanismo que permite às classes herdarem os atributos e operações de uma classe mãe, ou super classe. Através da herança é possível definir funcionalidades comuns em classes hierarquicamente superiores, fazendo com que novas classes sejam criadas implementando somente as rotinas que forem diferentes às já existentes. Através do uso do mecanismo da herança, o paradigma da orientação ao objeto permite a obtenção de reaproveitamento ou reutilização. Rotinas embutidas em classes ancestrais são utilizadas ou adaptadas pelos objetos descendentes conforme necessário, evitando-se a duplicação de código, e facilitando manutenções futuras. Desta forma, as *IDE*’s disponibilizam para o desenvolvedor classes prontas que são utilizadas na construção de um *software*. Por isso, classes específicas para a interface, tais como botões estão prontas e torna-se necessário configurá-las, alterando atributos como sua cor de fundo, por exemplo. A reutilização permite o aumento na produtividade.

Generalização e herança são abstrações para o compartilhamento de semelhanças entre classes, ao mesmo tempo em que suas diferenças são preservadas. Generalização é o relacionamento de uma classe e uma ou mais versões refinadas dela. A classe que estiver em processo de refinamento é chamada de superclasse e cada versão refinada é denominada subclasse. A Figura 4.3 mostra este relacionamento entre as classes “inferenciaExata“ e “inferenciaEAproximada“ que são subclasses de “inferencia“. Os atributos e operações comuns a um grupo de subclasses são incluídos na superclasse e compartilhados por todas as subclasses. Diz-se que a subclasse herda as características da superclasse, conforme [Booch et al. 2006]. Portanto, tudo o que for incluído em “inferencia“ estará disponível em suas subclasses.

A Figura 4.3 mostra um diagrama de classes onde há aplicações destas ligações e associações. Há relações do tipo “parte-todo” entre as classes redeBayesiana e nodo. Isso significa que um nodo é “uma-parte-de” uma rede bayesiana e que uma rede bayesiana é composta por vários nodos. O nodo por sua vez, é composto por estados e TPC. Estado é “uma-parte-de” de um nodo. Quando olhamos para o relacionamento entre inferencia e

as classes inferenciaExata e inferenciaAproximada, notamos que há um triangulo próximo à classes inferencia. Neste caso, o que está sendo informado é que elas "são tipos-de" inferencia, ou ainda que inferencia é sua ancestral. Há uma relação de herança onde inferência especifica atributos e métodos que serão herdados pelas subclasses.

A análise de sistemas no modelo orientado a objetos é feita analisando-se os objetos e eventos que interagem com estes objetos. O projeto orientado ao objeto é feito a partir da reutilização classes de objetos existentes e, quando necessário, construindo-se novas classes . Ao modelar um sistema, os projetistas devem identificar seus tipos de objetos e as operações que fazem com que estes objetos se comportem de certas maneiras [Martin and Odell 1994]. A UML[1] é uma linguagem para modelagem orientada a objeto de sistemas que tem sido amplamente utilizada e aceita na comunidade dos desenvolvedores de *software* por propiciar diferentes visualizações para os problemas que esta sendo modelados.

4.4.1 Padrões de Projeto

Tendo em vista toda a complexidade inerente ao desenvolvimento de *software* como um *framework*, que deve ser reutilizável, buscou-se uma forma que se pudesse evitar ao máximo o reprojeto, isto é, ele deveria ser projetado de uma maneira genérica o bastante para que novos requisitos pudessem ser atingidos sem que ele tenha que ser alterado. Obter o equilíbrio correto entre ter um projeto específico para o problema a ser resolvido e genérico o bastante para atender a tanto a vários problemas quanto a requisitos futuros têm uma importância grande.

De acordo com [Gamma et al. 2000], os padrões de projeto facilitam a reutilização de projetos e arquiteturas. Eles representam técnicas testadas e aprovadas por outros desenvolvedores e sua divulgação faz com que todas essas experiências tornem-se mais acessíveis. Os padrões de projeto aplicáveis a uma determinada solução devem ser identificados durante o projeto do *software*. Gamma et al. [Gamma et al. 2000] classificam 23 padrões distintos e os agrupa dentro de três categorias: padrões de criação, estruturais e comportamentais.

4.4.1.1 Modelo MVC

De cordo com [Buck and Yacktman 2010], o *Cocoa* é organizado de acordo com o padrão de projeto MVC, que é um dos padrões de projetos mais antigos e com maior sucesso, tendo sindo introduzido juntamente com *Smalltalk*. É um padrão de alto nível em que se preocupa com a arquitetura global de um aplicativo e classifica objetos de acordo com as funções gerais que eles executam em um aplicativo. É também um padrão composto na medida em que compreende vários padrões mais elementares, em geral *Strategy*, *Observer* e *Composite*. Os *software* construídos com o paradigma da orientação ao objeto podem se

[1] *Unified Modeling Language* - linguagem de modelagem unificada

beneficiar de várias formas, adaptando o padrão de projeto MVC para seus projetos. Muitos objetos tendem a ser mais reutilizáveis e suas interfaces podem ser melhor definidas. Os programas em geral são mais adaptáveis às mudanças nos requisitos. Além disso, muitas tecnologias e arquiteturas presentes no *COCOA* são baseados em MVC. A Figura 4.4, mostra como este modelo está estruturado seguindo o *framework Cocoa*.

O padrão de projeto MVC considera a existência de três tipos de objetos: objetos de modelos, de exibição e do controle. O padrão define os papéis que estes tipos de objetos desempenham na aplicação e suas linhas de comunicação. Ao projetar uma aplicação, um passo importante é escolher ou criar classes personalizadas para os objetos, de forma que elas possam pertencer a um destes três grupos. Cada um dos três tipos de objetos é separado dos outros por fronteiras abstratas. O MVC é interessante quando é necessário o desenvolvimento de aplicações interativas com interfaces gráficas controladas pelo usuário. Para aplica-lo é necessário dividir os objetos que serão usados na construção da aplicação em três subsistemas:

- *Model* – é a camada que contém a lógica da aplicação. É composto pelos objetos que são responsáveis pelas regras de negócio e pelo processamento dos dados do aplicativo. Em sistemas que utilizam persistência de dados, o *model* representa a informação (dados) dos formulários e as regras *SQL* que manipulam os dados do banco. Ele mantém o estado persistente do negócio e fornece ao *controler* a capacidade de acessar as funcionalidades da aplicação. Atuando isoladamente, ele não tem conhecimento de quais serão as interfaces que terá de atualizar, ele somente acessa os dados e os deixa prontos para o controlador e este por sua vez, encaminha para a visão correta. No modelo estarão os objetos das classes que representam a Rede Bayesiana, o Nodo, o Estado e a Tabela de Probabilidade Condicional. Quando um usuário realizar uma mudança em um valor de um estado, haverá uma propagação de probabilidades o que irá fazer com que as Tabelas de Probabilidade Condicional de alguns nodos tenham que ser atualizadas. A propagação, e alteração nas tabelas, são tradadas pelo *model*, que encaminhará para a *view* os resultados.

- *View* – É a camada de interação com usuário. É a interface que proporcionará a entrada de dados e a visualização de respostas geradas, apresentando informações vindas do *model*. Pode haver várias *views* dentro de um mesmo aplicativo. Por exemplo, pode haver uma *View* com a interface Gráfica com usuário, outra com um relatório impresso e uma exibição linguagem de *script* que interagem com o mesmo modelo. Em aplicações web, por exemplo, o *view* é representado pelo HTML que é mostrado pelo *browser*. Em geral podem exisitir vários elementos visuais, tais como, tabelas, menus e botões para entrada e saída de dados. A visão deve garantir que sua apresentação reflita o estado do modelo, quando os dados do modelo mudam, o modelo notifica as vistas que dependem dele, cada vista tem a chance de atualizar-se. Desta maneira permite ligar muitas vistas a um modelo podendo fornecer diferentes apresentações, essa

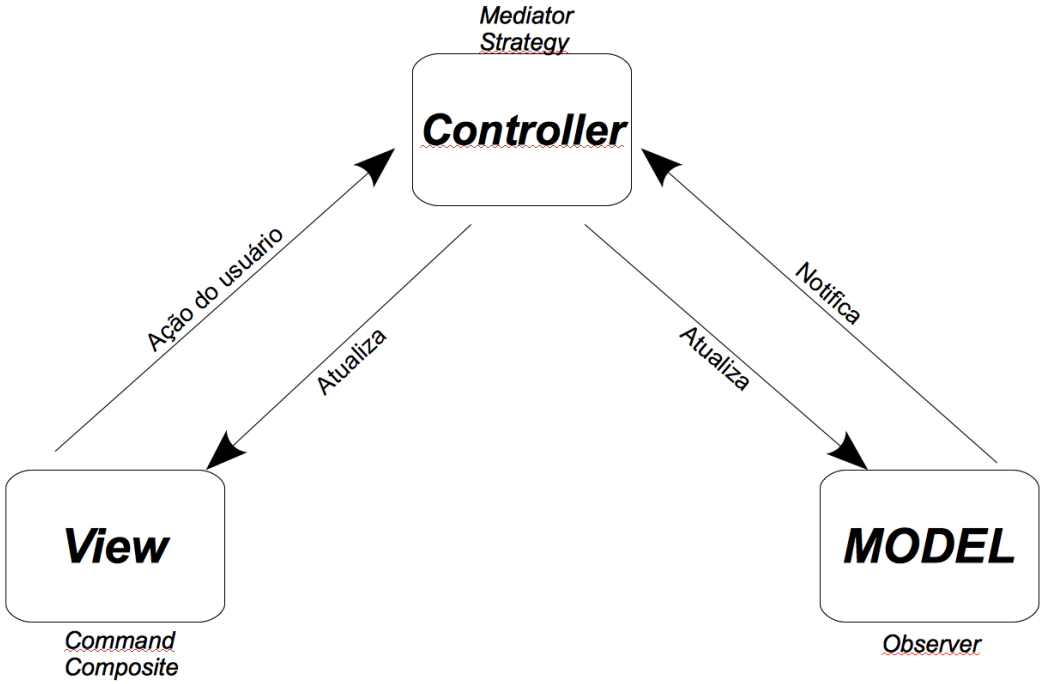

Figura 4.4: Como o MVC está implementado pelo Cocoa

camada não contém códigos relacionados á lógica de negócios, ou seja, todo o processamento é feito pelo Modelo e este repassa para a visão. Todas as classes fornecidas pelo ambiente *Cocoa*, que representam elementos de interface gráfica estão incluídas neste subsistema.

- *Controler* – É importante desacoplar os dados de sua apresentação para que a manutenção do *software* se torne mais simples. Por isso, o *Controler* é responsável por implementar mecanismos para fazer com que o *Model* e a *View* se comuniquem, de forma que cada interação do usuário com a *View* produzirá uma requisição que será encaminhada ao *Model*. O *Controler* é responsável por encaminhar as requisições e por enviar as repostas.

4.4.1.2 Padrão *Strategy*

O *Strategy* permite que os algoritmos mudem independentemente entre clientes que os utilizam. Desta forma, os algoritmos para propagação de evidências ficarão à disposição do usuário, bastando que ele escolha qual ele pretende usar. Uma das características mais importantes deste padrão é a separação dos elementos que o compõe em três participantes:

- Estratégia – é uma interface comum para todos os algoritmos a erem implementados. O contexto usará essa interface para chamar a implementação desejada por meio de uma estratégia concreta.

- Estratégia Concreta – contém a implementação dos algoritmos usando a interface definida na Estratégia.
- Contexto – é uma das classes do domínio do problema. Representa o cliente da estratégia.

Para a modelagem da estratégia, é necessária a criação de uma classe geral que declara uma interface comum para todos os algoritmos suportados. A implementação de cada estratégia é feita em sub-classes da estratégia, que podemos chamar de estratégias concretas. Para a utilização das estratégias concretas, é preciso que o contexto crie uma instancia de uma delas. Novas estratégias podem ser implementadas desde que sejam criadas novas classes concretas. A Figura 4.5 mostra, por meio de um diagrama de classes, como é a relação entre as classes.

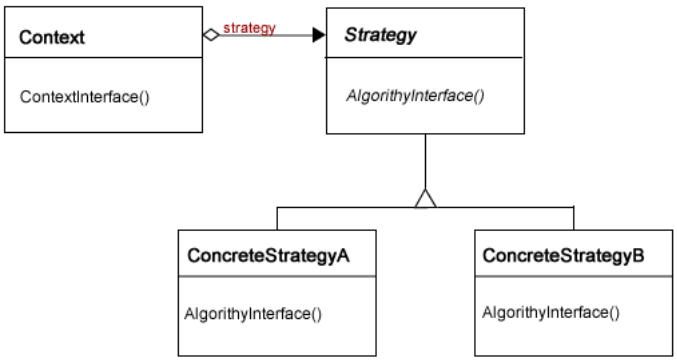

Figura 4.5: Padrão *Strategy*

A classe chamada *context* representa o contexto e ela pode usar qualquer uma das estratégias (*ConcretStrategyA*, e *ConcretStrategyB*) que estão disponíveis.

4.5 Conclusão

No processo de desenvolvimento de *software*, em decorrência de seus requisitos. é necessário que seja escolhida a infra-estrutura tecnológica responsável por realizar sua implementação. Desta maneira, Nesse capitulo foi apresentada a plataforma de *software* que foi usada na implementação do *framework*. Os conceitos básicos sobre a plataforma *Cocoa* foram listados. Informações iniciais a respeito da orientação ao objeto, tais como classes, encapsulamento, herança, que são aplicados diretamente tanto na linguagem *Objective-C* quanto nos padrões de projeto foram discutidos brevemente.

Foram comparadas algumas outras ferramentas que permitem a construção de aplicativos para plataforma iOS, que foi adotada nesse trabalho. As justificativas que fizeram com que o *Cocoa* e o *Objetctive-c* fossem as escolhidas foram apresentadas.

Informações sobre os padrões de projeto foram fornecidas. Desta maneira, busca-se permitir um maior entendimento sobre eles de maneira que sua importância para a o projeto e a implementação, assuntos que serão tratados no próximo capítulo desse trabalho, torne-se mais clara. O MVC serve para nortear as futuras aplicações que usarão o *framework*. Deve-se projeta-las de maneira que cadas uma das três partes trabalhem em conjunto.

O capitulo seguinte, apresentará algumas cenários em que o *framework* foi utilizado. Assim, pretende-se mensurar sua aplicabilidade a domínios relacionados com a engenharia biomédica.

Capítulo 5

Casos de uso

5.1 Resumo

O objetivo deste capítulo é demonstrar o uso prático de redes bayesianas, por meio do uso da aplicação, apresentada no Capítulo 3 em cenários relativos à engenharia biomédica onde são usados as Redes de Sensores Sem Fios (RSSF). O uso da aplicação tem o objetivo de modelar um cenário usando como ferramenta o *framework* proposto. Com a estrutura e as probabilidades condicionais geradas, é possível realizar inferências na rede, e também realizar análise de cenários.

5.2 Redes de sensores sem fio

Os Sensores, segundo Carvalho *et al.*[Carvalho et al. 2003], existem para medir variáveis físicas, tais como temperatura, pressão ou batimentos cardíacos. Os Nodos sensores representam dispositivos autônomos equipados com capacidades de sensoriamento, processamento e comunicação. Quando estes nodos são dispostos em rede em um modo *ad hoc*, formam as redes de sensores. Os nodos coletam dados via sensores, processam localmente ou coordenadamente entre vizinhos sendo que a informação coletada, pode ser enviada tanto para um usuário em geral quanto para um *data sink*.

Cada nodo em uma rede pode executar tarefas específicas, tais como : sensoriamento do ambiente, processamento da informação e tarefas associadas com o tráfego em um esquema de retransmissão *multi-hop*. Uma variável pode ser medida por vários sensores, o que é conhecido como sensores redundantes, ou por apenas um único sensor, chamado de sensor individual. Sensores individuais são únicos no sistema e não há razão para usar vários sensores deste tipo. Por outro lado, os sensores redundantes podem co-existir com vários sensores do mesmo tipo e isso é importante para que se possa obter tolerância a falhas, aumento da área de cobertura ou atender a outros requisitos. O sistema tem que ser apto

para diferenciar esse tipos de sensores e os dados que são obtidos por eles.

A capacidade de operação autônoma e a utilização de um meio de comunicação sem fios, são características importantes das Redes de Sensores Sem Fios aplicadas ao monitoramento do corpo humano. Em [Barbosa 2008], o autor informa que uma RSSF exige que sejam sejam aplicadas técnicas para tratamento de falhas e que estas técnicas promovam a adaptação às mais variadas condições encontradas no ambiente, de maneira que o tempo de funcionamento destes sistemas, isto é sua sobrevida, seja maximizado. Deve-se lembrar que em muitos casos, a simples substituição ou a recarga de suas baterias pode ser inviável, pois a rede pode ser composta por uma quantidade muito grande de nós sensores ou o acesso a eles é difícil, pois podem estar implantados no corpo humano.

É desejável que uma RSSF possa crescer, isto é, a adição de novos nós e/ou novas funcionalidades deve ser possível. Além dessa capacidade, ela deve suportar o aumento do volume da informação manipulada. Essas duas características não podem levar a perda da eficacia.

A topologia mais comumente usada para uma rede de sensores sem fio aplicado ao corpo humano é a topologia estrela [Lozano et al.]. Essa topologia tem com principal vantagem a otimização do consumo de energia da rede, pois há nodos chamados de escravos que tem como única é função atuar como coordenadores que transmitem as informações recebidas pelos sensores.

As principais funcionalidades das redes de sensores podem ser separadas em cinco grupos de atividades: estabelecimento da rede, manutenção, sensoriamento, processamento e comunicação.

Entre as todas as características principais de performance de uma rede de sensores, a vida útil dos sensores deve ser sempre considerada. As colisões devem ser evitadas sempre que possível pois o relay produz um consumo desnecessário de energia, além de outros atrasos. É preciso encontrar uma solução em que sejam evitados tanto a sobrecarga da rede, bem como o consumo máximo de energia.

O Serviço de estabelecimento de uma rede de sensores sem fio, tem inicio com funções de planejamento e continua com operações de instalação dos recursos necessários aos serviços providos pela rede e funções de configuração da rede. Algumas das funções envolvidas são distribuições de nodos, atribuição de valores aos parâmetros configuráveis, descoberta inicial da topologia, e descoberta da localização.

5.2.1 Aplicação das RSSF na área de saúde

Uma informação importe, encontrada em [Barbosa et al. 2003] é que uma RSSF é um tipo de sistema dependente da aplicação. Assim sendo, os fatores que podem influenciar o projeto de uma rede deste tipo advêm dos requisitos obtidos diante da serventia para a qual a ela foi projetada. As aplicações para as RSSF são muitas. Em se tratando de aplicações para residências, representa uma área promissora cujo objetivo é automatizar as funções de uma residência, como controle de temperatura, segurança, economia de energia elétrica, dentre outras, por exemplo. Na área da Medicina, elas podem permitir que sejam criados sistemas de monitoramento de pacientes, sendo que esses pacientes podem estar tanto dentro de um hospital ou unidade de saúde ou em sua casa, o que pode permitir uma grande mudança no paradigma de cuidados com a saúde. Cumpro lembrar que o paciente pode ter dificuldade para descrever os sintomas que sente, e com o intervalo existente entre as consultas alguns podem ser esquecidos. Um sistema que aplique esse novo paradigma pode permitir que os sinais possam ser reportados em em tempo real para que o profissional de saúde os possa relacionar com os dados recolhidos pelos sensores.

Como visto em [Sene Jr et al. 2006], o avanço tecnológico e a redução contínua do tamanho de componentes eletrônicos torna os sensores cada vez menores, chegando à ordem nano, com longa durabilidade e robustez No caso de aplicações na área de saúde, estas características facilitam o desenvolvimento de sistemas móveis de monitoração contínua da saúde do paciente.

Segundo Nascimento *et al.* [Nascimento et al. 2007], uma Rede de Sensores para o Corpo Humano (*Body Sensor Network BSN*) e um sistema distribuído composto por nós sensores que são usualmente interconectados por um meio de comunicação sem fios e alimentados por baterias. Em geral, o nó sensor é composto por uma unidade de processamento, memória, interfaces e transdutores, rádio transmissor e receptor, circuito de alimentação e bateria. As BSNs podem ser usadas para o monitoramento ininterrupto e não obstrutivo da saúde humana. Eles possuem um grande potencial de influir nas aplicações clínicas, pois promovem um novo paradigma para o monitoramento da saúde humana por meio de sistemas computacionais embutidos na indumentária das pessoas (*wearable systems*).

5.3 Caso de uso

O objetivo deste cenário é simular uma situação onde os sensores são aplicados no corpo de um paciente. Utilizando-se da capacidade de processamento existente nos sensores, e conhecendo as tarefas que são realizadas em uma RSSF, conforme mostrado anteriormente,

assume-se que as tarefas de sensoriamento e de processamento da informação já foram realizadas. Desta maneira, cada nó sensor é visto sob uma perspectiva de alto nível, onde são omitidos detalhes sobre a sua estrutura e também como ele realiza suas tarefas. Fica também indiferente a quantidade deles, e se estão executando o sensoriamento de forma individual ou redundante. É mais importante representá-los a por meio da função que eles executam ou qual informação esses nodos estão medindo. Desta forma, o profissional de saúde deveria focar sua atenção em criar um modelo que utilizasse estes dados e realizasse tarefas de inferência.

5.3.1 Alerta para aparecimento de trombose arterial

Como cenário para aplicação procura-se reproduzir por meio de simulações um sistema capaz de monitorar a temperatura dérmica do corpo humano obtida a partir de vários sensores de temperatura distribuídos em várias regiões na superfície do corpo. Essa distribuição deve garantir que haja sensores em cada parte do corpo. O monitoramento da temperatura é importante, pois o diagnóstico de algumas patologias, como por exemplo, a trombose arterial pode ser automatizado.

A trombose arterial costuma ser mais grave do que a venosa, pois impede a chegada do oxigênio às células provocando nelas o infarto com necrose (morte tecidual). A gravidade vai depender do local afetado e da extensão da trombose. Quando ocorre em artérias do cérebro, provoca o AVC (Acidente Vascular Cerebral) popularmente conhecido como "Derrame".

Esta doença tem como alguns dos seus sintomas dor intensa, inchaço na perna, vermelhidão, alterações na temperatura do local atingido, endurecimento da musculatura da perna e formação de nódulos dolorosos nas varizes, são freqüentes. Além disso, há alguns fatores de risco favorecem o surgimento da doença como por exemplo histórico familiar, hábito de fumar e hipertensão arterial.

A Figura 5.1 mostra como a interface da aplicação é exibida no dispositivo móvel. Na parte superior há uma barra de ferramentas onde há ícones que permitem a execução de várias funcionalidades, tais como, criar uma nova rede, criar um novo nodo ou ainda criar uma ligação entre os nodos. Além disso, como estratégia para facilitar o uso, caso o usuário toque em uma elipse, que representa o nodo, um controle aparecerá contendo algumas opções. No caso, o nodo "histórico familiar" foi clicado, o que é chamado de *tapped*, e a única opção disponível é apagar o nodo.

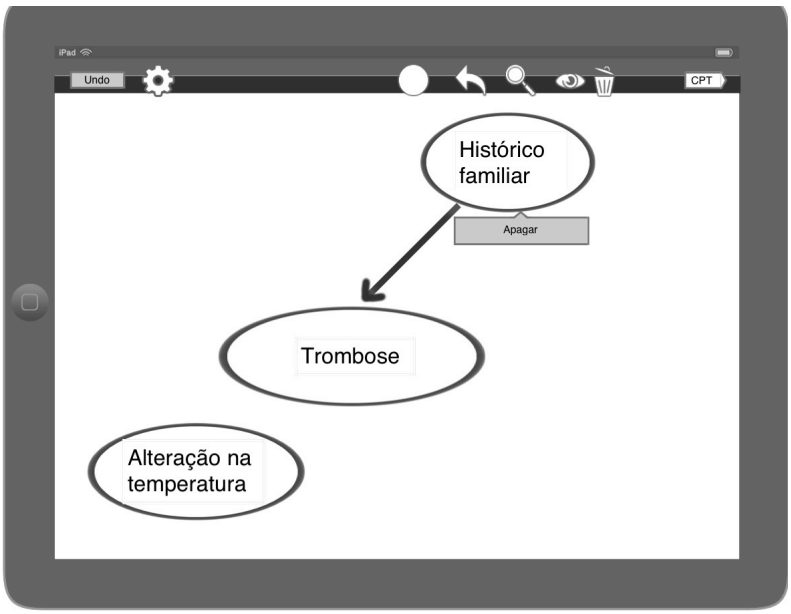

Figura 5.1: Rede bayesiana sendo modelada pela aplicação

5.3.1.1 Modelagem da rede

Para facilitar a montagem da rede, foram considerados apenas algumas das possíveis variáveis, uma vez que este estudo pretende relatar o uso do *framework* em um possível cenário. Desta forma, para que o sistema pudesse auxilar o profissional da área de saúde, criou-se a rede bayesiana mostrada na Tabela 5.2. Imaginado que cada variável possua dois estados "sim" e "não". O que está modelado é que com o surgimento da doença, a probabilidade de que os sintomas apareçam vai aumentar. Assim, a probabilidade de o estado "sim" dos nodos que representam mudanças na temperatura local, dor na região e inchaço fique maior. Os fatores de risco por sua vez, quando informados, aumentarão a probabilidade da doença.

Os sensores distribuídos coletariam os dados relativos à temperatura no membro em o paciente estivesse sentido dor e assim, os sinais seriam obtidos e processados. Deveriam ser distribuídos vários sensores de maneira que fosse possível coletar a temperatura dos membros, obter uma média desta temperatura e fazer a comparação de maneira que se pudesse constatar uma diferença significativa. O resultado deste processamento estaria representado pelo nodo "alteração na temperatura", cujos estados "sim" e "não" indicariam que há uma mudança na temperatura local. As demais informações podem ser obtidas por meio de perguntas realizadas pelo profissional da área de saúde.

Após a identificação das variáveis, foi necessário quantificar as relações de influencia que existiam entre elas. Para as variáveis "fumante" e "histórico familiar" que não pais, basta que se determine os valores de crença para cada um dos seus estados. Para estabelecer a prevalência de fumantes, usou-se dados do VIGITEL divulgados em Abril de 2012 (Vigilância de Fatores de Risco e Proteção para Doenças Crônicas por Inquérito Telefônico) revelaram o número de fumantes no Brasil, acima de 18 anos de idade, em 14,8% [INCA].

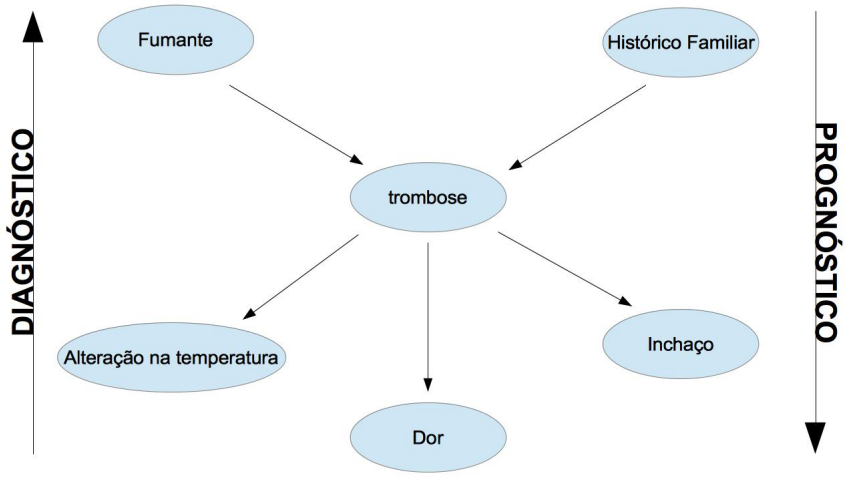

Figura 5.2: Rede bayesiana modelada para o caso de uso

Desta forma, considera-se que a probabilidade de que a pessoa, independente de que ela tenha ou não a trombose arterial, é 0.148.

Tabela 5.1: Probabilidades para o nodo Fumante

Valor	P(Fumante)
Sim	0.148
Não	0.862

Com relação ao histórico familiar, tomou-se como base os dados disponíveis em [SBACV]. A estimativa aponta, de maneira geral, que há 60 casos da doença para cada 100.000 habitantes ao ano, o que resulta em uma probabilidade de 0.0006. A tabela que contem as probabilidades priori para esta variável e mostrada na Tabela 5.2.

Tabela 5.2: Probabilidades para o nodo histórico familiar

Valor	P(histórico familiar)
Sim	0.0006
Não	0.9994

Como o nodo que representava a doença era influenciado por dois nodos pai, cada linha da TPC desta variável era dependente da relação entre os pais. Devido a dificuldade em encontrar as estatísticas que quantificassem a relação conjunta entre as causas da trombose arterial, foi usada a distribuição ou-exclusivo, uma vez que esse mecanismo depende apenas

do valor da probabilidade de inibição individual que cada pai tem. Neste caso seriam necessárias apenas duas informações e o algoritmo implementado faria o preenchimento da tabela. Usou-se os valores 0.1 e 0.2 para indicar a inibição dos nodos "fumante" e "histórico familiar". A tabela gerada é mostrada na Tabela 5.3

Tabela 5.3: TPC do nodo "trombose arterial"

Fumante	Histórico	Verdadeiro	Falso
Sim	Sim	0.98	0.02
Sim	Não	0.90	0.10
Não	Sim	0.80	0.20
Não	Não	0.00	1.00

Para os nodos que são filhos de trombose sabe-se que a trombose muitas vezes é assintomática, isto é, não apresenta sintomas. Porém, em estágios mais avançados, os sintomas tornam-se muito muito evidentes como por exemplo o inchaço nos membros afetados, onde é visualmente possível verificar a diferença entre os membros. Desta forma, usou-se o valor de 0.60 para indicar que quando uma pessoa possui trombose, os membros apresentarão inchaço.

Para os nodos os demais filhos do nodo trombose, "Alteração na temperatura" e "Dor", utilizou-se novamente a distribuição ou ruidoso e os valores de inibição 0.1 e 0.6 para eles respectivamente.

5.3.1.2 Usando a rede bayesiana

O uso da rede bayesiana permitiu que as evidências fossem informadas de maneira que se pudessem ser realizados diagnósticos e prognósticos. Desta maneira, o usuário do aplicativo pode, por exemplo, marcar o nodo "alteração na temperatura" como uma evidência e verificar qual seria a probabilidade de que uma pessoa fosse fumante. Neste caso, a RSSF que foi instalada detecta que há uma alteração na temperatura em um dos membros do paciente. Essa evidência é marcada pelo usuário, e o sistema faz a propagação da evidência, o que acarretará uma mudança nos valores dos estados do nodo "fumante", que são mostrados na Tabela 5.4. Assim, conclui-se que quando o paciente apresentar alteração na temperatura dos membros, há uma chance de 40,9% de que seja em decorrência de uma trombose e a probabilidade de que essa pessoa seja fumante é de 41.9%.

No caso em que o paciente informe ao médico que ele fuma e que em sua família há casos de trombose. Desta forma, busca-se fazer uma inferência intercausal, onde há duas causas para o mesmo efeito. Para este estudo, obteve-se 98% de chance de que esse paciente

Tabela 5.4: Novos valores em fumante após a inferência

Valor	P(Fumante)
Sim	0.419
Não	0.58.1

venha a desenvolver a trombose.

Há ainda uma terceira forma de aplicar as inferências que o raciocínio prognóstico. Imaginando que, como no exemplo anterior, o paciente tenha casos de trombose em sua familia e que ele seja fumante. Qual seria a chance de que a rede de sensores detectaria uma alteração na temperatura de suas panturrilhas ? Neste caso as evidências são "fumante" e "histórico familiar" e a variável de consulta é "alteração na temperatura". Esse raciocínio faz com que encontremos novos valores para os efeitos a partir de causas que são evidentes. Com os valores que foram usados, a probabilidade de haja alguma alteração na temperatura é de 83%.

5.4 Conclusão

Este capitulo apresentou brevemente o que são as redes de sensores sem fio e qual sua importância em aplicações para a área de saúde. Desta forma foi proposto um esto de caso em que as RSSF são utilizadas de maneira que o resultado do sensoriamento torna-se um nodo de uma rede bayesiana.

Uma rede bayesiana foi modelada para que se pudesse avaliar um quadro onde há suspeita de trombose arterial. Não há muitos dados estátisiticos disponíveis, por isso alguns valores foram sugeridos pelo autor deste trabalho. Deve-se lembrar que os que se pretendia era demostrar o funcionamento de uma aplicação que utiliza o framework e verificar como se dá o seu funcionamento e não a acurácia dos dados utilizados. Assim, procurou-se apresentar como algumas das funcionalidades implementadas no Capítulo 3 podem ser utilizadas e que resultados podem ser alcançados.

Capítulo 6

Conclusão

6.1 Considerações Finais

Este trabalho apresentou a rede bayesiana como um métodos para modelar a incerteza e com grande utilidade para representar processos probabilísticos. Elas possibilitam a tomada de decisões por meio de realiza;cão de inferências em variáveis aleatórias. Isso tem grande utilidade em engenharia biomédica, pois a modelagem dos cenários expressa uma relação de causa e efeito. Como exemplo, foram propostos a analisados dois cenários de interesse em Engenharia Biomédica.

Outro aspecto interessante se refere à capacidade de visualização global dos dados do problema, pois facilita o entendimento do problema e as possíveis modificações na instância corrente da Rede, na aplicação. Com isso é possível concluir que uma rede Bayesiana que represente corretamente um domínio, pode ser considerada um método atrativo para armazenamento e extração de conhecimento.

A utilização de *frameworks* no desenvolvimento de aplicações trouxe benefícios, originados de suas características principais: são modulares, reusáveis, extensíveis. Durante a implementação do *framework* e do estudo de caso, ficou claro que a uma ferramenta computacional que tem como principal objetivo ser uma solução computacional genérica, faz com que alguns conceitos intrínsecos da orientação a objetos sejam aplicados. Quando uma aplicação é criada a partir de uma solução genérica, observa-se que há uma mudança pois ela deixa suas características gerais e torna-se específica para um determinado proposito, mesmo que a ferramenta usada, o *framework* no qual ela foi baseada, permaneça genérico. Assim, observa-se que o mecanismo de generalização/especialização é de alguma forma utilizado, uma vez que o *framework* cria regras que definem como as tarefas devem ser feitas. Fica sob a responsabilidade do usuário entender essas regras e usá-las para a modelagem de cenários.

A principal contribuição desse trabalho foi a criação de um *framework* capaz de manipular redes bayesianas em dispositivos móveis. Ele permite que sejam modelados vários cenários. Além disso as tecnologias adotadas possibilitam utilizar a ferramenta proposta em dispositivos móveis com interface *touchscreen*, promovendo a usabilidade do sistema e a facilitando a mobilidade do usuário. Isso também pode contribuir para a potencialização do uso da tecnologia, reduzindo o tempo de resposta e melhorando a qualidade dos diagnósticos e prognósticos em Saúde.

A aplicação construída, foi utilizada para auxiliar no processo de projeto de Redes de Sensores para aplicações em Saúde probabilísticas. O nodo proposto, pensado como um nodo pertencente a uma rede bayesiana, pode ser usado sem problemas com um nodo sensor. Isso posto, permite dizer que a classe "nodo", pensada como uma entidade genérica, permite que seu reuso tanto para outros nodos bayesianos quanto para outros nodos.

Os dois *softwares* desenvolvidos nesse trabalho o *framework* e a aplicação foram de muita valia, ajudando a fixar muito bem os conceitos envolvidos principalmente os advindos da engenharia de *software* e da matemática, além de materializar a pesquisa em ferramentas de redes bayesianas.

As ferramentas tecnológicas utilizadas, o *framework Cocoa* e a linguagem *Objective-C*, possuem várias características interessantes tais como: boa documentação, grande quantidade de utilizadores e facilidade no aprendizado. É necessário ressaltar o fato de que essas ferramentas estão disponíveis apenas na plataforma MAC. Quando um desenvolvedor imaginar a implementação de aplicações nativas para iOS ou Mac OS, elas serão a principal alternativa. Mas, se esse desenvolvedor quiser utilizar sua *expertise* nestas ferramentas para desenvolver aplicações para outras plataformas, ele não conseguirá. Isso faz com que elas fiquem restritas apenas a uma parte de todo o público que usa dispositivos moveis.

6.1.1 Outros modelos de representação probabilísticos

As redes bayesianas, são um modelo para raciocínio probabilístico. Sua suas características, apresentadas ao longo desse trabalho, demonstram que elas são uma ferramenta importante para lidar com problemas onde o raciocínio incerto é necessário. Apesar disso, para que seja possível a modelagem de certos cenários em que existam dependências temporais, há outros modelos que são utilizados e essas ferramentas permitem que sejam criados e monitorados cenários que mudam ao longo do tempo. Mesmo sendo possível, através do uso de inferências, tal como mostrado nesse trabalho, verificar o comportamento futuro de seus atributos, as redes bayesianas não dispõem de meios que permitam descobrir o quão próximo ou distante esses eventos estariam de acontecer, por exemplo. Assim, elas

não permitem quantificar e apontar, no tempo, o momento da ocorrência dessas inferências. Se imaginarmos uma situação em que fosse necessário fazer o tratamento de uma pessoa diabética, imagina-se que variáveis como doses de insulina recentes, alimentação, medições de açúcar no sangue, dentre outras, serão usadas. Se esses valores forem usados como evidências, eles podem mudar ao longo do tempo, pois a taxa de açúcar variam conforme a dose da insulina ou em função da alimentação. Desta maneira, nota-se que para extrair novas informações em decorrência do tempo, é preciso introduzir o modelo em um processo Markoviano, onde os estado atual depende do estado anterior.

Há três modelos temporais que são encontrados em [Russell and Norvig 2004]: RDB, HMM e Filtros de Kalman. De acordo com [Frondana 2012] as redes bayesianas dinâmicas são um tipo de rede bayesiana que modela séries temporais onde assume-se que um evento pode implicar outro evento no futuro, não o contrário. Como encontrado em [Mihajlovic and Petkovic 2001], uma RBD é uma rede bayesiana na qual foi acrescida a dimensão temporal. Em [Ghanmi et al. 2011] encontramos outra definição, a de que as RDB são como uma sequencia várias redes bayesiana, onde cada uma dessa redes tem um passo causal que liga uma rede a outra. Com relação HMM, eles também são um modelo de probabilístico temporal. A diferença em relação à RDB é que o cenário é representado por uma única variável, que cujos estados representam os valores possíveis. Todo HMM pode, segundo [Russell and Norvig 2004], ser representado por uma RBD, desde que possua apenas uma única variável de estado e uma única variável de evidência. Já os filtros de Kalman, podem ser prepresentados por uma RBD, desde que as variáveis sejam contínuas e a distribuição dos estados é sempre uma distribuição condicional gaussiana.

Apesar da utilidade dos outros métodos probabilísticos, as redes bayesianas foram utilizadas principalmente por serem um mecanismo adequado para a modelagem e explicação de um domínio qualquer. Além disso, os métodos temporais seriam mais adequados em situações em que a rede tivesse que comunicar-se com outros dispositivos para que se pudesse fazer um monitoramento constante. Para fins de simulação, as redes bayesianas são adequadas, conforme o estudo de caso apresentado. Além disso, as redes bayesianas, são um modelo mais genérico que os demais e por isso, elas podem ser usadas como base para a implementação dos outros modelos.

6.2 Propostas para trabalhos futuros

A grande dificuldade para o uso das redes bayesiana é o fato de que *a priori* as probabilidades existentes nas relações entre as variáveis, nem sempre são conhecidas. Desta maneira, como trabalhos futuros, podem ser investigadas maneira de fazer com que a classes TPC possam receber seus dados a partir de outros softwares, usando para isso os próprios

recursos de conexão existentes no *tablet* com a troca de arquivos XML.

Seguindo nesta linha, explorando a conexão com outros sistemas, podem ser investigados a aplicação de novos algoritmos, como por exemplo algoritmos para aprendizagem probabilística, de forma que estrutura da rede possa ser criada de maneira automatizada com base em exemplos obtidos no domínio a ser modelado. Outro caminho a ser investigado refere-se ao estudo de novos algoritmos para a propagação de evidências. Estes influenciados pelos requisitos da própria aplicação. Assim, cenários mais restritivos quanto ao tempo de resposta e à resiliência podem ser representados. Um exemplo são as Redes de Sensores para aplicações em Saúde.

Conforme apresentado no capítulo 5 deste trabalho, é possível observar que a rede proposta pode ser entendida como sendo composta por dois tipos de nodo, sendo eles, o nodo bayesiano implementado pelo *framework* proposto e o nodo sensor. Cumpre ressaltar que apesar de os testes feitos mostrarem que o nodo bayesiano cumpriu o papel de representar o sensor, há na literatura modelos de nodo sensor genérico, tal como o proposto em [Carvalho et al. 2003], cujo modelo pode ser visto na figura 6.1. Para que se possa criar um nodo sensor, deve-se proceder com alterações na classe nodo proposta neste trabalho.

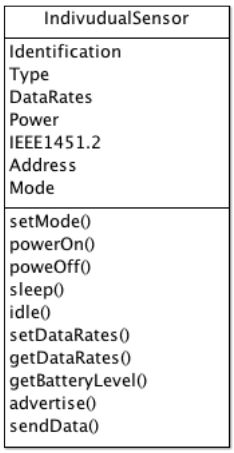

Figura 6.1: Modelo de sensor genérico

Essa alteração não representa uma mudança na função básica do *framework*, mas trata-se um complemento. Permitiria que novos algoritmos próprios das RSSF pudessem ser investigados e implementados. Além disso, alguma alterações podem ser feitas na classe "redeBayesiana". Outra classe mais específica pode ser projetada para que algumas funcionalidades que são inerentes ao uso de redes de sensores sem fio possam ser inseridas, como por exemplo a descoberta de funções.

Outro estudo envolve a conexão via *bluetooth* entre o *tablet* e uma rede sem fio. Assim,

o *framework* poderia usar os dados reais capturados dos sensores. Nesse caso, além das funcionalidades, requisitos não funcionais precisam ser considerados, como, por exemplo, o consumo de energia ao longo do tempo. Assim, pode-se também estender o modelo apresentado para que seja capaz de implementar redes bayesianas dinâmicas.

Outra sugestão para trabalhos futuros é a especialização do *framework* apresentado, de forma que seja possível a modelagem de outros métodos probabilísticos tais como as redes bayesianas dinâmicas.

Conclui-se aqui esperando ter sido possível apresentar de forma clara a construção de um sistema capaz de utilizar o raciocínio probabilístico como ferramenta para tratar incertezas e representar o conhecimento de um especialista.

Referências Bibliográficas

[Aaron 2008] Aaron, H. (2008). *Cocoa programming for Mac OS X*. Pearson Education.

[Abril] Abril, E. Mesmo sem ipad brasileiro, natal de 2011 será dos tablets. Site Web - http://veja.abril.com.br/noticia/economia/mesmo-sem-ipad-brasileiro-natal-de-2011-sera-dos-tablets.

[Apple 2011] Apple (2011). Apple inc. - the objective-c programming language. Site WEB - Disponível em https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Chapters/ocProtocols.html.

[Barbosa 2008] Barbosa, T. (2008). Uma arquitetura de redes de sensores do corpo humano. 188p. Tese de doutorado em Engenharia Elétrica - Universidade de Brasília - Departamento de Enegenhaira elétrica, Brasília. 2008.

[Barbosa et al. 2003] Barbosa, T., Sene, I., Carvalho, H., Rocha, A., and Nascimento, F. (2003). Arquitetura de software para redes de sensores sem fios: a proeminência do middleware. In *XXV Congresso da Sociedade Brasileira de Computação*.

[Bergström and Engvall 2011] Bergström, F. and Engvall, G. (2011). Development of handheld mobile applications for the public sector in android and ios using agile kanban process tool.

[Booch et al. 2006] Booch, G., Rumbaugh, J., and Jacobson, I. (2006). *UML: guia do usuário*. Elsevier.

[Buck and Yacktman 2010] Buck, E. M. and Yacktman, D. A. (2010). *Cocoa design patterns*. Addison-Wesley Professional.

[Carvalho et al. 2003] Carvalho, H., Heinzelman, W., Murphy, A., and Coelho, C. (2003). A general data fusion architecture. pages 1465–1472.

[Cervo and BERVIAN 2007] Cervo, A. L. and BERVIAN, P. A. (2007). *Metodologia científica*. Pearson Prentice Hall.

[Cozman] Cozman, F. G. Javabayes - bayesian networks in java. Site WEB - http://www-2.cs.cmu.edu/~javabayes/.

[Department of Computer Science] Department of Computer Science, D. U. Banjo: Bayesian network inference with java objects. Site WEB - http://www.cs.duke.edu/~amink/software/banjo/.

[Fallet et al. 2012] Fallet, G., Weber, P., Simon, C., Iung, B., Duval, C., et al. (2012). Evidential network-based extension of leaky noisy-or structure for supporting risks analyses. In *8th International Symposium SAFEPROCESS 2012*.

[Folha.com 2011] Folha.com (2011). Brasil importou 64 mil ipads em 2010. Site Web - http://www1.folha.uol.com.br/mercado/867821-brasil-importou-64-mil-ipads-em-2010.shtml. Acessado em 05/07/2011.

[Foundation] Foundation, A. Ao surgery reference. Site Web - http://itunes.apple.com/br/app/ao-surgery-reference/id403961165?mt=8.

[Frondana 2012] Frondana, I. (2012). Classificação de biopotenciais via cadeias de markov ocultas.

[Gamma et al. 2000] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (2000). *Padrões de projeto: Soluções reutilizáveis de software orientado a objetos*. Editora Bookman.

[Geyer and Felske 2011] Geyer, M. and Felske, F. (2011). Consumer toy or corporate tool: the ipad enters the workplace. *interactions*, 18(4):45–49.

[Ghanmi et al. 2011] Ghanmi, N., Mahjoub, M., and Amara, N. (2011). Characterization of dynamic bayesian network-the dynamic bayesian network as temporal network. *International Journal*, 2.

[Gonçalves and Brunetto 2008] Gonçalves, A. R. and Brunetto, M. A. d. O. C. (2008). Um novo modelo híbrido baseado em otimização por colônia de formigas e redes neurais para identificação de indivíduos com dpoc. *Anais do XI Congresso Brasileiro de Informática Aplicada em Saúde*.

[Google] Google. Android. Site Web - http://www.android.com.

[Honeyman-Buck 2010] Honeyman-Buck, J. (2010). The magical ipad. *Journal of Digital Imaging*, 23(4):514–515.

[IBOPE] IBOPE. Ibope. Disponível em: http://www.ibope.com.br/calandraWeb/servlet/CalandraRedirect?temp=5&proj=PortalIBOPE&pub=T&db=caldb&comp=Not%EDcias&docid=A57B524940DAA4B5832579BA0054101D. Acessado em 04/03/2012.

[Inc. 2010a] Inc., A. (2010a). ipad - technical specifications. Site Web -Disponível em iPad-TechnicalSpecifications.

[Inc. 2010b] Inc., A. (2010b). ipad programming guide general. Disponível em http://developer.apple.com/library/ios/navigation/index.html.

[Inc. 2010c] Inc., A. (2010c). iphone human interface guide. Disponível em: http://developer.apple.com/library/ios/navigation/index.html.

[INCA] INCA. Instituto nacional do cancer - observatório da política nacional de controle do tabaco. Site Web - http://www2.inca.gov.br/wps/wcm/connect/observatorio_controle_tabaco/site/home/dados_numeros/prevalencia. Acessado em 08/09/2012.

[ISBA] ISBA. International society for bayesian analysis (isba). Site WEB - http://www.bayesian.org/.

[José et al. 2005] José, A., Barbosa, T., Castro, L., Sene Jr, I., Carvalho, H., da Rocha, A., and Nascimento, F. (2005). Implementação da revisão sistemática de sintomas em sistemas móveis utilizando redes bayesianas. *Workshop de Informática Médica (WIM)*, 5.

[KDD] KDD. Kansas state university laboratory for knowledge discovery in databases - bayesian network tools in java. Site WEB - http://bnj.sourceforge.net/. Acessado em 21/03/2011.

[Kochan 2011] Kochan, S. (2011). *Programming in Objective-C*. Addison-Wesley Professional.

[Korb and Nicholson 2004a] Korb, K. and Nicholson, A. (2004a). *Bayesian artificial intelligence*, volume 1. CRC press.

[Korb and Nicholson 2004b] Korb, K. and Nicholson, A. (2004b). *Bayesian artificial intelligence Chapman & Hall/CRC computer science and data analysis*. CRC Press.

[Lozano et al.] Lozano, C., Tellez, C., and Rodríguez, O. Biosignal monitoring using wireless sensor networks.

[Lucas 2004] Lucas, P. (2004). Bayesian analysis, pattern analysis, and data mining in health care. *Current Opinion in Critical Care Disponível em : http://www.cs.ru.nl/~peterl/current-opinion.pdf Acessado em 25/04/2008*, 10(5):399.

[Lucas et al. 2004] Lucas, P., van der Gaag, L., and Abu-Hanna, A. (2004). Bayesian networks in biomedicine and health-care. *Artificial Intelligence In Medicine Disponível em http://kik.amc.uva.nl/home/aabuhanna/AIM04.pdf Acessado em Acessado em 21/04/2008*, 30(3):201–214.

[Marques and Dutra 2005] Marques, R. and Dutra, I. (2005). Redes bayesianas: o que são, para que servem, algoritmos e exemplos de aplicações. *Relatório técnico, Coppe Sistemas-UFRJ, Cidade Universitária, Centro de Tecnologia, Bloco H, Sala*, 319:21941–972.

[Martin and Odell 1994] Martin, J. and Odell, J. (1994). *Object-oriented methods*. Prentice Hall PTR.

[McCann et al. 2006] McCann, R., Marcot, B., and Ellis, R. (2006). Bayesian belief networks: applications in ecology and natural resource management. *Canadian Journal of Forest Research*, 36(12):3053–3062.

[Mendes 2002] Mendes, A. (2002). *Arquitetura de Software, Desenvolvimento orientado para arquitetura*. Editora Campus.

[Microsoft] Microsoft. Microsoft bayesian network editor. *Site WEB - http://research.microsoft.com/msbn/*.

[Mihajlovic and Petkovic 2001] Mihajlovic, V. and Petkovic, M. (2001). Dynamic bayesian networks: A state of the art.

[Nascimento et al. 2007] Nascimento, F., Rocha, A., Barbosa, T., and Carvalho, H. (2007). Uma rede de sensores para monitoração do corpo humano com suporte à programação; a body sensor netword with programming support. *Revista brasileira de engenharia biomed*, 23(3):231–244.

[Norsys] Norsys. Netica bayesian network software from norsys. Site WEB - http://www.norsys.com/ Acessado em 18/11/2008.

[Pearl and Russell 2011] Pearl, J. and Russell, S. (2011). Bayesian networks. *Department of Statistics Papers, Department of Statistics, UCLA, UC Los Angeles*, page 8.

[Pilone and Pilone 2011] Pilone, D. and Pilone, T. (2011). *Head First IPhone and IPad Development: A Learner's Guide to Creating Objective-C Applications for the IPhone and IPad*. O'Reilly Media, Inc.

[Planalto 2011] Planalto (2011). Palácio do planalto, lei 12.507. Site Web - http://www.planalto.gov.br/ccivil_03/_ato2011-2014/2011/Lei/L12507.htm Acessado em 21/08/2012.

[Pressman 2006] Pressman, R. (2006). *Engenharia de Software*. São Paulo: McGraw-Hill.

[Russell and Norvig 2004] Russell, S. J. and Norvig, P. (2004). *Inteligência artificial: tradução da Segunda Edição*. Ed. Campus.

[Saheki 2005] Saheki, A. (2005). *Construção de uma rede bayesiana aplicada ao diagnóstico de doenças cardíacas*. EPUSP
Disponível em http://www.pmr.poli.usp.br/ltd/People/asaheki/docs/defesa_full.pdf.

[Sales et al. 2010] Sales, M., Schwaab, A., and Nassar, S. (2010). Application of bayesian networks to assist the expansion of the digital inclusion of elderly people. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 8(3):275–279.

[Sanders and Aronsky 2006] Sanders, D. and Aronsky, D. (2006). Detecting asthma exacerbations in a pediatric emergency department using a bayesian network. In *AMIA Annual Symposium Proceedings*, volume 2006, page 684. American Medical Informatics Association.

[SBACV] SBACV. Sociedade brasileira de angiologia e de cirurgia vascular. Site WEB - Disponível em `http://www.sbacv.com.br/index.php/imprensa/estimativas.html`. Acessado em 22/09/2012.

[S.Castro et al.] S.Castro, L. S., Branisso, H. J. P., Figueiredo, E. C., Nascimento, F. A. O., Rocha, A. F., and Carvalho, H. S. Handmed - um sistema móvel integrado para captura automática de sintomas. *sbis.org.br Disponível em : `http://www.sbis.org.br/cbis9/arquivos/379.pdf`*.

[Sene Jr et al. 2006] Sene Jr, I., Barbosa, T., Carvalho, H., da Rocha, A., and Nascimento, F. (2006). Monitoração da temperatura corporal baseada em uma rede de sensores sem fios.

[Shaw and Garlan 1994] Shaw, M. and Garlan, D. (1994). An introduction to software architecture.

[Siqueira 2011] Siqueira, E. (2011). Vendas de tablets explodirão. Site Web - `http://blogs.estadao.com.br/ethevaldo-siqueira/2011/07/13/vendas-de-tablets-explodirao/`. Acessado em 01/08/2011.

[Software] Software, M. Mobile mim. Site WEB - Disponível em `http://www.mimsoftware.com/products/mobile/`.

[Sommerville et al. 2007] Sommerville, I., Melnikoff, S., Arakaki, R., and de Andrade Barbosa, E. (2007). *Engenharia de software - 8a. Edição*. Pearson Addison-Wesley.

[TIOBE 2012] TIOBE (2012). Tiobe programming community index. Site Web - `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`. Acessado em 10/11/2012.

[Volonte et al. 2011] Volonte, F., Robert, J., Ratib, O., and Triponez, F. (2011). A lung segmentectomy performed with 3d reconstruction images available on the operating table with an ipad. *Interactive CardioVascular and Thoracic Surgery*, 12(6):1066.

[Wong and Joussen 2011] Wong, D. and Joussen, A. (2011). Welcome to the ipad generation. *Graefe's Archive for Clinical and Experimental Ophthalmology*, 249(1):1.

ANEXOS

A. CLASSE REDE BAYESIANA

```
1  //
2  // RedeBayesiana.h
3  // tcm
4  //
5  // Created by Elisangela Boni on 16/08/11.
6  // Copyright 2011 EHL. All rights reserved.
7  //
8  #import <Cocoa/Cocoa.h>
9  #import <Foundation/NSArray.h>
10 #import <Foundation/NSString.h>
11 #import <Foundation/NSObject.h>
12 #import <Foundation/NSAutoreleasePool.h>
13 #import "stdlib.h"
14 #import "nodo.h"
15 #import "inferencia.h"
16
17 #define ARC4RANDOM_MAX 0x100000000
18
19 @interface RedeBayesiana : NSObject
20 {
21         NSString *Nome;
22         NSMutableArray *Rede;
23         id<inferencia> Metodo;
24 }
25
26 @property (assign) id<inferencia> Metodo;
27
28 -(id) initWithName:(NSString *) nome;
29 -(id) criaRede:(NSString *)nome;
30 //-(void) NovoNodo:(nodo *) novo;
31 -(void) NovoNodo:(nodo *)novo setNome:(NSString *)novoNome;
32 -(id) NovoNodo:(NSString *)novoNome;
33 -(int) Nr_Nodos;
34 -(void) printNodos;
35 -(void) removeNodo:(nodo *)rNodo;
36 -(float) calculaDistribuicaoConjuntaTotal:(NSMutableArray *) parametros;
37 -(float) getDistribuicaoConjuntaDe:(NSMutableArray *) variavelDeConsulta daddo:(NSMutableArray *)
       parametros;
38 -(NSMutableArray *) getCoberturaDeMarkov:(NSString *)nomeNodo;
39 -(void) printCoberturaDeMarkov:(NSString *) variavelDeConsulta;
40 -(void) propagaEvidencia;
41 -(void) dealloc;
42 @end
```

```
1  //
2  // RedeBayesiana.m
3  // tcm
4  //
5  // Created by Elisangela Boni on 16/08/11.
6  // Copyright 2011 EHL. All rights reserved.
7  //
8
9  #import "RedeBayesiana.h"
10
11 @implementation RedeBayesiana
12
13 @synthesize Metodo;
14
15 -(void) propagaEvidencia
16 {
17  if (self.Metodo == nil)
18          NSLog(@"É preciso escolher um metodo para propagar as evidencias
                    ...");
19  else
20          [self.Metodo calcula_inferencia];
21 }
22
23 -(id) initWithName:(NSString *) nome
24 {
25          self = [super init];
26          if (self)
27          {
28                  Nome = [[NSString alloc] initWithString:nome];
29                  Rede = [[NSMutableArray alloc] init];
30                  Metodo = nil;
31          }
32          return self;
33 }
34
35 -(id) criaRede:(NSString *)nome;
36 {
37          RedeBayesiana *novo = [[[RedeBayesiana alloc] init]autorelease];
38          if (novo)
39          {
40                  Nome = [[NSString alloc] initWithString:nome];
41                  Rede = [[NSMutableArray alloc] init];
42                  Metodo = nil;
43          }
44          return novo;
45 }
46
47 -(id) BuscaNodo:(NSString *)novoNome
48 {
```

```
49          for (nodo *nnodo in Rede)
50                  if ([nnodo.nome isEqualToString: novoNome]==YES)
51                  {
52  //  NSLog(@"%-31s",[nnodo.nome UTF8String]);
53                          return nnodo;
54                  }
55          return NULL;
56  }
57
58  -(void) NovoNodo:(nodo *)novo setNome:(NSString *)novoNome
59  {
60          if ([self BuscaNodo:novoNome]==NULL)
61          {
62                  novo = [novo initWithName:novoNome];
63                  [Rede addObject: novo];
64          }
65  }
66
67  -(id) NovoNodo:(NSString *)novoNome
68  {
69          if ([self BuscaNodo:novoNome]==NULL)
70          {
71                  nodo *novo = [[nodo alloc] initWithName:novoNome];
72                  [Rede addObject: novo];
73                  return [novo autorelease];
74          }
75          return NULL;
76  }
77
78  -(int) Nr_Nodos;
79  {
80          return [Rede count];
81  }
82
83  -(void) printNodos;
84  {
85          NSLog(@"==== Nodos da Rede : %@",Nome);
86          for (nodo *nnodo in Rede)
87                  NSLog(@"%-31s",[nnodo.nome UTF8String]);
88          NSLog(@"======================");
89  }
90
91  -(float) calculaDistribuicaoConjuntaTotal:(NSMutableArray *) parametros;
92  {
93          int indice;
94          float valor=1;
95          NSMutableArray *linha = nil;
96          NSMutableString* nomeNodo = nil;
97          NSMutableString* estadoNodo = nil;
```

```
98          nodo *nNodo = nil;
99          for (indice=0;indice<[parametros count];indice++)
100         {
101                 linha = [parametros objectAtIndex:indice];
102                 nomeNodo = [linha objectAtIndex:0];
103                 estadoNodo = [linha objectAtIndex:1];
104                 nNodo = [self BuscaNodo:nomeNodo];
105                 if (nNodo!=NULL)
106                         valor = [nNodo getProbabilidade:estadoNodo]*valor;
107                 else
108                         valor = -1;
109                 [linha removeLastObject];
110         }
111         return valor;
112 }
113
114 -(float) getDistribuicaoConjuntaDe:(NSMutableArray *) variavelDeConsulta daddo:(NSMutableArray *)
            parametros;
115 {
116         nodo *nodoConsulta =[self BuscaNodo:[variavelDeConsulta objectAtIndex:0]];
117         nodo *nodoEvidencia = nil;
118         NSString *nodos_parametros = nil;
119         Estado *estadoConsulta = [nodoConsulta getEstado:[variavelDeConsulta objectAtIndex:1]];
120         NSLog(@"Distribuicao conjunta total de %-s=%-s, dado",[[nodoConsulta
                    nome]UTF8String],[[estadoConsulta nome]UTF8String]);
121
122         int indice;
123         float valor=1.0;
124         NSMutableArray *linha = nil;
125         NSMutableString* nomeNodo = nil;
126         NSMutableString* estadoNodo = nil;
127
128         NSMutableArray *enderecosNodos = [[[NSMutableArray alloc] init] autorelease];
129         NSMutableArray *enderecosEstados = [[[NSMutableArray alloc] init] autorelease];
130
131         NSMutableArray *enderecosPais = [[[NSMutableArray alloc] init] autorelease];
132         NSMutableArray *enderecosEstadosPais = [[[NSMutableArray alloc] init] autorelease];
133
134         NSMutableArray *enderecosFilhos = [[[NSMutableArray alloc] init] autorelease];
135         NSMutableArray *enderecosEstadosFilhos = [[[NSMutableArray alloc] init] autorelease];
136
137
138         // gerar uma lista com nodos validos
139         for (indice=0;indice<[parametros count];indice++)
140         {
141                 linha = [parametros objectAtIndex:indice];
142                 nomeNodo = [linha objectAtIndex:0];
143                 estadoNodo = [linha objectAtIndex:1];
144         // Concatenar ...
```

```
145         //nodos_parametros
146                 nodoEvidencia = [self BuscaNodo:nomeNodo];
147                 if (nodoEvidencia!=NULL)
148                 {
149                         [enderecosNodos addObject:nodoEvidencia];
150                         [enderecosEstados addObject:[nodoEvidencia getEstado:estadoNodo]];
151                 }
152         }
153         // identificar os pais da variavel de consulta
154         // pois os pais determinarão qual linha da tpc deverá ser retornada
155         for (indice=0;indice<[enderecosNodos count];indice++)
156                 if ([nodoConsulta paiDoNodo:[enderecosNodos objectAtIndex:indice]]==TRUE)
157                 {
158                         [enderecosPais addObject:[enderecosNodos objectAtIndex:indice]];
159                         [enderecosEstadosPais addObject:[enderecosEstados objectAtIndex:indice]];
160                 }
161         [enderecosNodos removeObjectsInArray:enderecosPais];
162         [enderecosEstados removeObjectsInArray:enderecosEstadosPais];
163
164         // identificar os filhos da variavel de consulta
165         // pois nos filhos o valor da varivavel de consulta deverá ser procurado nas TPC
166         for (indice=0;indice<[enderecosNodos count];indice++)
167                 if ([nodoConsulta filhoDoNodo:[enderecosNodos objectAtIndex:indice]]==TRUE)
168                 {
169                         [enderecosFilhos addObject:[enderecosNodos objectAtIndex:indice]];
170                         [enderecosEstadosFilhos addObject:[enderecosEstados objectAtIndex:indice]];
171                 }
172         [enderecosNodos removeObjectsInArray:enderecosFilhos];
173         //
174         NSNumber* resultado = [NSNumber numberWithFloat:[nodoConsulta getProbabilidade:
                    estadoConsulta dadoEvidencia:enderecosEstadosPais]];
175
176         valor = [resultado floatValue] * valor;
177         int pai = 0;
178         for (nodo *nodoPai in enderecosPais)
179         {
180                 resultado = [NSNumber numberWithFloat:[nodoPai getValor:[enderecosEstadosPais
                            objectAtIndex:pai]]];
181                 NSLog(@"Resultado = %10.3f",[resultado floatValue]);
182                 valor = [resultado floatValue] * valor;
183                 NSLog(@"Valor = %1.10f",valor);
184                 pai++;
185         }
186
187         NSMutableArray *enderecoNodoConsulta = [[[NSMutableArray alloc] init] autorelease];
188         [enderecoNodoConsulta addObject:estadoConsulta];
189
190         pai=0;
191         for (nodo *nodoFilho in enderecosFilhos)
```

```
192         {
193                 resultado = [NSNumber numberWithFloat:[nodoFilho getProbabilidade:[
                            enderecosEstadosFilhos objectAtIndex:pai] dadoEvidencia:
                            enderecoNodoConsulta]];
194                 valor = [resultado floatValue] * valor;
195                 pai++;
196         }
197
198     NSLog(@"Valor Final = %1.10f",valor);
199         return 0.0;
200 }
201
202
203 -(NSMutableArray *) getCoberturaDeMarkov:(NSString *)nomeNodo;
204 {
205         NSMutableArray* paisDosFilhos = [[[NSMutableArray alloc]init]autorelease];
206         NSMutableSet* temp = [[[NSMutableSet alloc]init]autorelease];
207         NSMutableArray* resultado = [[[NSMutableArray alloc]initWithObjects:nil, nil, nil]autorelease];
208         nodo *nodoConsulta =[self BuscaNodo:nomeNodo];
209         if ([[nodoConsulta pais] count]>0)
210                 [resultado addObject:[nodoConsulta pais]];
211         if ([[nodoConsulta filhos] count]>0)
212                 [resultado addObject:[nodoConsulta filhos]];
213         for (nodo* filho in nodoConsulta.filhos)
214         {
215                 [temp addObject:[[filho pais]mutableCopy]];
216                 [temp removeObject:nodoConsulta];
217                 if ([temp count]>1)
218                         [paisDosFilhos addObject:temp];
219                 [temp removeAllObjects];
220         }
221         [resultado addObject:paisDosFilhos];
222         return resultado;
223 }
224
225 -(void) printCoberturaDeMarkov:(NSString *) variavelDeConsulta;
226 {
227     NSMutableArray* cobertura = [self getCoberturaDeMarkov:variavelDeConsulta];
228         NSLog(@"Cobertura de Markov de %-20s",[variavelDeConsulta UTF8String]);
229         NSMutableArray* nomes = [NSMutableArray arrayWithObjects:@"==Pais",@"==Filhos",@
                    "==Pais dos Filhos",nil];
230         int i;
231         for (i=0;i<[cobertura count];i++)
232         {
233         if ([[cobertura objectAtIndex:i] count]!=0)
234         {
235             NSLog(@"%-20s",[[nomes objectAtIndex:i] UTF8String]);
236                         for (nodo* pai in [cobertura objectAtIndex:i])
237                                 NSLog(@"%-20s",[pai.nome UTF8String]);
```

```
238             }
239         }
240 }
241
242 −(float) calculoInferenciaGibbs:(NSString ∗) variavelDeConsulta evidencias:(NSMutableArray ∗)
                parametros;
243 {
244             float valorAleatorio = (arc4random() % 100)/100.0f;
245             NSLog(@"%1.9f",valorAleatorio);
246             NSMutableArray∗ cobertura = [self getCoberturaDeMarkov:variavelDeConsulta];
247             return 0.0;
248 }
249
250
251 −(void) dealloc;
252 {
253             [Nome release];
254             [Rede release];
255             [Metodo release];
256             [super release];
257             [super dealloc];
258 }
259
260 −(void) removeNodo:(nodo ∗)rNodo;
261 {
262             [Rede removeObject: rNodo];
263             [rNodo release];
264 }
265
266 @end
```

B. CLASSE NODO

```
1  //
2  // nodo.h
3  // tcm
4  //
5  // Created by Elisangela Boni on 15/08/11.
6  // Copyright 2011 EHL. All rights reserved.
7  //
8
9  #import <Foundation/NSObject.h>
10 #import <Foundation/NSString.h>
11 #import <Foundation/NSArray.h>
12 #import <Foundation/NSAutoreleasePool.h>
13 #import "Estado.h"
14 #import "Tpc.h"
15
16
17
18 @interface nodo : NSObject
19 {
20
21         NSString *nome; // nome que identidica o nodo
22         NSMutableSet *filhos;
23         NSMutableSet *pais;
24 // Alterar a adjacencia: Existem filhos e pais
25         NSMutableArray *estados;
26         Tpc *tabela; // Ponteiro para a tabela de probabilidades conjunta
27 }
28
29 @property (copy, nonatomic) NSString *nome;
30 @property (retain, nonatomic) NSMutableArray *estados;
31 @property (retain, nonatomic) NSMutableSet *filhos;
32 @property (retain, nonatomic) NSMutableSet *pais;
33 @property (retain, nonatomic) Tpc *tabela;
34
35 -(id) initWithName:(NSString *) nomedonodo;
36 //-(id) BuscaNodo:(NSString *)novoNome;
37 //-(void) novoEstado:(Estado *) novo_estado setNome:(NSString *) nomedoestado andValor:(int)
       valordoestado;
38 -(void) novoEstado:(Estado *) novo_estado;
39 -(void) novoEstado:(NSString *) novo_nome valor:(float) apriori;
40 //-(void) removeEstado: (nodo *) rEstado;
41 -(BOOL) formaCiclo:(nodo *)nodo_raiz grafoinicial:(NSMutableSet*) filhos;
42 -(void) print;
43 -(void) atualizaProbabiliddes;
44 //(void) atualizarEvidencia:(NSString *)estado setValor:()valor
```

```
45 −(void) printEstados;
46 −(void) printFilhos;
47 −(void) printPais;
48 −(void) novoFilho: (nodo ∗) nFilho;
49 −(void) novoPai: (nodo ∗) nPai;
50 −(void) removeFilho: (nodo ∗) nFilho;
51 −(id) getEstado:(NSString ∗) nomeEstado;
52 //−(void) listarTPC;
53 −(void) removePai: (nodo ∗) nPai;
54 −(void) removePainoIndice:(int) indice;
55 −(id) pegaPainoIndice:(int) indice;
56 −(id) pegaPaiPorNome:(NSString∗) nomePai;
57 −(NSString ∗) pegaNome;
58 −(id) BuscaFilho:(NSString ∗)novoNome;
59 −(void) geraLinhasTabela:(nodo ∗)nNodo token:(NSMutableArray ∗)tToken indice:(int ∗) i linha:(int∗) l;
60 −(int) tabelaContagem;
61 −(void) geraTabela;
62 −(void) printTpc;
63 //−(void) setTpc:(int)indice valor:(float)valor estado:(NSString∗)estado;
64 −(void) setTpc:(int)indice estado:(NSString ∗)nome probabilidade:(float)valor;
65 −(float) getValor:(Estado ∗)enderco_estado;
66 −(float) getProbabilidade:(NSString ∗)nome_estado;
67 −(void) setEvidencia:(NSString ∗)nome_estado;
68 −(float) getProbabilidade:(Estado ∗)estado dadoEvidencia:(NSMutableArray ∗)parametros;
69 −(BOOL) paiDoNodo:(nodo ∗)pai;
70 −(BOOL) filhoDoNodo:(nodo ∗)filho;
71 −(void) aplicarDeterministico:(NSString ∗) nomeEstado estadoPai:(NSString ∗) nomePai valor:(float)
       probabilidade;
72 −(void) aplicarOuRuidoso:(NSMutableArray ∗)parametros valor:(NSString∗)valor;
73 //−(void) discretizar:(NSMutableArray ∗)
74
75
76 //−(id) BuscaPai:(NSString ∗)novoNome;
77 −(void) dealloc;
78
79 // inserir estados
80 // NSSet ∗uniqueElements = [NSSet setWithArray:myArray];
81 @end
82
83 // apagaEstado
84 // editaEstado
85 // atualizarEvidencia(estado, valor)
86 // propagarEvidencia ==> atualizar todo o modelo
```

```
//
// nodo.m
// tcm
//
// Created by Elisangela Boni on 15/08/11.
// Copyright 2011 EHL. All rights reserved.
//
#import "nodo.h"
#import "parametro.h"

@implementation nodo

@synthesize nome,estados,filhos,pais,tabela;

//-(void) novoEstado:(Estado *) novo_estado setNome:(NSString *) nomedoestado andValor:(int)
        valordoestado;
//{
// verificar
//}

-(void) novoEstado:(Estado *) novo_estado;
{
        [self.estados addObject:novo_estado];
}

-(void) novoEstado:(NSString *) novo_nome valor:(float) apriori;
{
        Estado* nEstado = [[[Estado alloc] initWithName:novo_nome] autorelease];
        [nEstado alteraValor:apriori];
        [self.estados addObject:nEstado];
}

-(void) retiraEstado:(Estado *) novo_estado;
{
        [self.estados removeObject:novo_estado];
}

-(id) initWithName:(NSString *) nomedonodo
{
        self = [[super init] autorelease];
        if (self)
        {
                nome = [[NSString alloc] initWithString:nomedonodo];
                estados = [[NSMutableArray alloc] init];
                filhos = [[NSMutableSet alloc] init];
                pais = [[[NSMutableSet alloc] init] autorelease];
                tabela = [[Tpc alloc] init];
        }
        return self;
```

```
49  }
50
51
52  -(id) getEstado:(NSString *) nomeEstado;
53  {
54          for (Estado *estado in estados)
55                  if ([estado.nome isEqualToString: nomeEstado]==YES)
56                          return estado;
57          return NULL;
58  }
59
60
61  -(void) print
62  {
63          NSLog(@"%-31@",nome);
64          for (Estado *nestado in estados)
65                  NSLog(@"%-20s %1.10f",[nestado.nome UTF8String],nestado.valor);
66  }
67
68  -(void) atualizaProbabiliddes;
69  {
70          // Para calcular as probabilidades de um estado de uma variável qualquer,
71          // é necessário calcular a probabilide de um estado X dado um estado dos pais.
72          float resultado = 0;
73          float produto = 0;
74          for (Estado *estado in estados)
75          {
76                  for (NSMutableArray* valoresPais in self.tabela.valor_pais)
77                  {
78                          produto = [self getProbabilidade:estado dadoEvidencia:valoresPais];
79                          for (Estado *estadoPai in valoresPais)
80                                  produto = [estadoPai getValor] * produto;
81                          resultado = resultado + produto;
82                  }
83                  [estado setValor:resultado];
84                  resultado = 0;
85          }
86  }
87
88  -(BOOL) formaCiclo:(nodo *)nodo_raiz grafoinicial:(NSMutableSet*) nodos_filhos;
89  {
90          if ([nodos_filhos containsObject:nodo_raiz])
91                  return YES;
92          else
93          {
94                  for (nodo * filho in nodos_filhos)
95                  {
96                          BOOL achou = [nodo_raiz formaCiclo:nodo_raiz grafoinicial:[filho filhos]];
97                          return achou;
```

```
98                  }
99          return NO;
100         }
101 }
102
103 -(void) printEstados;
104 {
105         if ([self.estados count]!=0)
106         {
107         NSLog(@"==== Estados do Nodo : %@",nome);
108                 for (Estado *nestado in estados)
109                         NSLog(@"%-20s %f",[nestado.nome UTF8String],nestado.valor);
110         }
111         else
112                 NSLog(@"sem estados para mostrar");
113 }
114
115 -(void) printFilhos;
116 {
117                 if ([filhos count]!=0)
118                 {
119                 NSLog(@"==== Filhos do Nodo : %@",nome);
120                         for (nodo *nnodo in filhos)
121                                 NSLog(@"%-20s",[nnodo.nome UTF8String]);
122                 }
123                 else
124                         NSLog(@"o nodo nao possui descendentes");
125 }
126
127 -(void) printPais;
128 {
129         if ([pais count]!=0)
130         {
131                 NSLog(@"==== Pais do Nodo : %@",nome);
132                 for (nodo *nnodo in pais)
133                         NSLog(@"%-20s",[nnodo.nome UTF8String]);
134         }
135         else
136                 NSLog(@"o nodo nao possui pais");
137 }
138
139
140 -(NSString *) pegaNome;
141 {
142         return self.nome;
143 }
144
145
146 -(BOOL) paiDoNodo:(nodo *) pai
```

```
147 {
148         if ([self.pais containsObject:pai])
149                 return TRUE;
150         else
151                 return FALSE;
152 }
153
154 -(BOOL) filhoDoNodo:(nodo *) filho
155 {
156         if ([self.filhos containsObject:filho])
157                 return TRUE;
158         else
159                 return FALSE;
160 }
161
162 -(id) BuscaFilho:(NSString *)novoNome
163 {
164         for (nodo *nnodo in filhos)
165                 if ([nnodo.nome isEqualToString: novoNome]==YES)
166                 {
167                         NSLog(@"%-31s",[nnodo.nome UTF8String]);
168                         return nnodo;
169                 }
170         return NULL;
171 }
172
173 #ifdef __llvm__
174 #pragma GCC diagnostic ignored "-Wdangling-else"
175 #endif
176
177 -(void) novoFilho:(nodo *)nFilho;
178 {
179 // Impedir que sejam inseridos estados que ja existam ...
180         NSString * nNome = [[NSString alloc] initWithString:[nFilho pegaNome]];
181         if ([self BuscaFilho:nNome]==NULL)
182         if ([self formaCiclo:self grafoinicial:[nFilho filhos]]==NO)
183         {
184                 [self.filhos addObject:nFilho];
185                 [nFilho novoPai:self];
186         }
187         else
188                 NSLog(@"A rede não pode conter ciclos");
189         [nNome release];
190 }
191
192
193 -(void) novoPai:(nodo *)nPai; // Verificar
194 {
195         // Impedir que sejam inseridos estados que ja existam ...
```

```
196 // NSString * nNome = [[NSString alloc] initWithString:[nPai pegaNome]];
197 // if ([self formaCiclo:self grafoinicial:[nPai pais]]==NO)
198 // {
199                     [self.pais addObject:nPai];
200 // }
201 // else
202 //  NSLog(@"A rede não pode conter ciclos");
203 // [nNome release];
204 }
205
206 -(void) removeFilho:(nodo *)nFilho;
207 {
208         [self.filhos removeObject:nFilho];
209
210 }
211
212 -(void) removePai:(nodo *)nPai;
213 {
214         [self.pais removeObject:nPai];
215
216 }
217
218 -(void) removePainoIndice:(int) indice;
219 {
220         int valor=0;
221         for (nodo *nNodo in self.pais)
222                 if (valor==indice)
223                 {
224                         [self removePai:nNodo];
225                         break;
226                 }
227                 else
228                         valor++;
229 }
230 -(id) pegaPainoIndice:(int) indice;
231 {
232         int valor=0;
233         for (nodo *nNodo in self.pais)
234                 if (valor==indice)
235                 {
236                         return nNodo;
237                         break;
238                 }
239                 else
240                         valor++;
241         return NULL;
242 }
243
244 -(id) pegaPaiPorNome:(NSString*) nomePai
```

```
{
        int valor=0;
        for (nodo *nNodo in self.pais)
                if ([nNodo.nome isEqualToString:nomePai]==YES)
                {
                        return nNodo;
                        break;
                }
                else
                        valor++;
        return NULL;
}


// Criando uma matriz em Objective C
//NSMutableArray *dataArray = [[NSMutableArray alloc] initWithCapacity: 3];
//[dataArray insertObject:[NSMutableArray arrayWithObjects:@"0",@"0",@"0",nil] atIndex:0];

-(void) geraLinhasTabela:(nodo *)nNodo token:(NSMutableArray *)tToken indice:(int *) i linha:(int*) l;
{
        static int estado = 0;
        //static linha = 0;
        nodo* nPai = (nodo *)[nNodo pegaPainoIndice:*i];
        for (Estado *nEstado in nPai.estados)
        {
                [tToken addObject:nEstado];
                if (((*i)+1)!=[nNodo.pais count])
                {
                        (*i)++;
                    [nNodo geraLinhasTabela:nNodo token:tToken indice:i linha:l];
                        [tToken removeLastObject]; // Remove o estado anterior que havia sido inserido
                }
                else
                {
                        NSNumber* probabilidade = [NSNumber numberWithFloat:((float)1/[nNodo.
                                estados count])];
                    NSMutableArray *Copia = [[NSMutableArray alloc] initWithArray:[tToken
                            mutableCopy]];
                        [nNodo.tabela.valor_pais insertObject:Copia atIndex:(*l)];
                        NSMutableArray *tProbabilidadeApriori = [[NSMutableArray alloc]
                                initWithCapacity:[nNodo.estados count]];
                        int x=0;
                        for (x=0;x<[nNodo.estados count];x++)
                                [tProbabilidadeApriori addObject:probabilidade];
                        [nNodo.tabela.valor_local insertObject:nNodo.estados atIndex:(*l)];
                        [nNodo.tabela.probabilidade insertObject:tProbabilidadeApriori atIndex:(*l)];
                        estado++;
                        (*l)++;
                        [tToken removeLastObject]; // remove o estado anterior
```

```
291                             [Copia release];
292                     }
293             }
294             //[nPai release];
295             (*i)--;
296             estado--;
297     }
298
299
300     -(void) geraTabela
301     {
302             int indice = 0;
303             int linha = 0;
304             NSMutableArray *tToken = [[NSMutableArray alloc] init];
305             [self geraLinhasTabela:self token:tToken indice:&indice linha:&linha];
306             [tToken release];
307     }
308
309
310     -(int) tabelaContagem
311     {
312             return (int)[self.tabela tabelaContagem];
313     }
314
315     -(void) printTpc
316     {
317             NSMutableString *linha = [[[NSMutableString alloc]init]autorelease];
318             NSLog(@"== TPC %-10s ==",[self.nome UTF8String]);
319             for (nodo *nPai in self.pais)
320                     [linha insertString:[NSString stringWithFormat:@"%-12s\t",[nPai.nome UTF8String]]
                                atIndex:[linha length]];
321             for (nodo *nEstado in estados)
322                     [linha appendString:[NSString stringWithFormat:@"%-12s\t",[nEstado.nome
                                UTF8String]]];
323
324             NSLog(@"%@",linha);
325             [self.tabela printTpc];
326             [linha setString:@""];
327     }
328
329     // [n3 setTpc:0 estado:@"Verdadeiro" probabilidade:0.95];
330
331     -(void) setTpc:(int)indice estado:(NSString *)nome_estado probabilidade:(float)valor
332     {
333
334             //refatorar
335             int x = 0;
336             int total_estados = [self.estados count];
337             NSNumber* prob = [NSNumber numberWithFloat:valor];
```

```
338             NSNumber* restante = [NSNumber numberWithFloat:((float)1-valor)/(float)(total_estados -1)];
339             Estado* tEstados =[self.tabela getValorLocalAtIndex:indice];
340             NSMutableArray *tValorLocal = [[[NSMutableArray alloc] init] autorelease];
341             for (Estado *nEstado in tEstados)
342             {
343                     if ([nEstado.nome isEqualToString:nome_estado]!=YES)
344                             [tValorLocal insertObject:restante atIndex:x];
345                     else
346                             [tValorLocal insertObject:prob atIndex:x];
347                     x++;
348
349             }
350 // if (x==total_estados)
351 //  NSLog(@"Estado %s nao encontrado",[nome_estado UTF8String]);
352 // else
353             [self.tabela.probabilidade replaceObjectAtIndex:indice withObject:tValorLocal];
354 }
355
356
357 -(void) aplicarDeterministico:(NSString *) nomeEstado estadoPai:(NSString *) nomePai valor:(float)
              probabilidade
358 {
359             NSMutableArray* valorPais = nil;
360             int x,quant;
361
362             for (x=0;x<[self.tabela.valor_local count];x++)
363             {
364                     quant=0;
365                     valorPais = [self.tabela.valor_pais objectAtIndex:x];
366                     for (Estado *pEstado in valorPais)
367                     {
368                             if ([pEstado.nome isEqualToString:nomePai]==YES)
369                             {
370                                     [self setTpc:x estado:nomeEstado probabilidade:probabilidade];
371                                     quant++;
372                                     break;
373                             }
374                     }
375                     if (quant==0)
376                     {
377                                     [self setTpc:x estado:nomeEstado probabilidade:1-probabilidade];
378                                     quant=0;
379                     }
380             }
381 }
382
383
384 -(void) aplicarOuRuidoso:(NSMutableArray *)parametros valor:(NSString*)valor
385 {
```

```
386             int x;// ,quant;
387             float probabilidade;
388             x=0;
389             for (NSMutableArray* valorPais in self.tabela.valor_pais)
390             {
391                     probabilidade = 1.0;
392 //  quant = 0;
393                     for (parametro *dado in parametros)
394                     {
395                             if ([valorPais containsObject:[[dado nodo] getEstado:valor]]==YES)
396                             {
397                                     probabilidade = probabilidade * [dado valor];
398 //    quant++;
399                             }
400                     }
401 //  if (quant==0)
402 //    [self setTpc:x estado:valor probabilidade:0];
403 //  else
404                             [self setTpc:x estado:valor probabilidade:1-probabilidade];
405                     x++;
406             }
407 }
408
409
410 -(float) getValor:(Estado *)enderco_estado
411 {
412             return [enderco_estado getValor];
413 }
414
415 -(float) getProbabilidade:(NSString *)nome_estado
416 {
417             float valor =-1.0;
418 // Estado* tEstados = [[[nodo alloc] init] autorelease];
419 // tEstados = self.estados;
420             for (Estado *nEstado in self.estados)
421             {
422                     if ([nEstado.nome isEqualToString:nome_estado]==YES)
423                             valor = [nEstado getValor];
424             }
425             return valor;
426 }
427
428 //-(float) getProbabilidadeDado:(NSMutableArray *)parametros;
429 -(float) getProbabilidade:(Estado *)estado dadoEvidencia:(NSMutableArray *)parametros;
430 {
431             if ([self.pais count]==0)
432             {
433                     // Caso seja uma variavel do tipo raiz, a probabilidade é o valor a priori
434                     NSMutableString* nomeEstado = [parametros objectAtIndex:0];
```

```
435                 NSLog(@"Valor da probabilidade %1.9f",[self getProbabilidade:nomeEstado
                        ]);
436                 return [self getProbabilidade:nomeEstado];
437         }
438         else
439                 //Caso o nodo tenha pais, parametros devera conter o estado de cada um dos pais.
440                 //Sendo assim, parametros devera trazer uma relação de enderecos de estados, sendo que
441                 //o primeiro deverá ser o estado da variável.
442                 return [[self.tabela getValor:estado evidencia:parametros] floatValue];
443 }
444
445
446 -(void) setEvidencia:(NSString *)nome_estado
447 {
448 // nada
449 }
450
451 -(void) dealloc;
452 {
453         [nome release];
454         [estados release];
455         [filhos release];
456         [pais release];
457         [tabela release];
458         nome = nil;
459         estados = nil;
460         filhos = nil;
461         pais = nil;
462         tabela = nil;
463         //self = nil;
464 // [super release];
465         [super dealloc];
466 }
467
468 // Incluir um metodo para busca dos estados cujo parametro é o nome do estado
469 // Incluir um metodo para retirada
470 // Incluir um metodo para alteração / correção dos nomes
471 // Incluir um método para permitir a entrada de evidencias
472 // Construir um metodo para a retirada do nodo
473
474
475 @end
```

C. CLASSE ESTADO

```
1  //
2  // Estado.h
3  // tcm
4  //
5  // Created by Elisangela Boni on 30/08/11.
6  // Copyright 2011 EHL. All rights reserved.
7  //
8
9  #import <Foundation/Foundation.h>
10 #import <Foundation/NSString.h>
11 #import <Foundation/NSObject.h>
12
13
14 @interface Estado : NSObject {
15         NSString *nome;
16         Float32 valor;
17 }
18
19 @property (copy, nonatomic) NSString *nome;
20 @property Float32 valor;
21
22 -(id) init;
23 -(id) initWithName:(NSString *) novoNome;
24 -(void) setNome:(NSString *) theName andValor:(Float32) theValue;
25 -(void) alteraNome:(NSString *) novoNome;
26 -(void) alteraValor:(Float32) novoValor;
27 -(float) getValor;
28 -(void) print;
29 -(void) print:(Estado *)enedereco;
30 @end
```

```
1  //
2  // Estado.m
3  // tcm
4  //
5  // Created by Elisangela Boni on 30/08/11.
6  // Copyright 2011 EHL. All rights reserved.
7  //
8
9  #import "Estado.h"
10
11
12 @implementation Estado
13
14 @synthesize nome,valor;
15
16 -(id) init
17 {
18         self = [super init];
19         return self;
20 }
21
22
23 -(id) initWithName:(NSString *) novoNome
24 {
25         self = [super init];
26         if (self)
27         {
28                 nome = [[NSString alloc] initWithString:novoNome];
29                 valor = 0.0;
30         }
31         return self;
32 }
33
34 -(void) setNome:(NSString *) theName andValor:(Float32) theValue
35 {
36         self.nome = theName;
37         self.valor = theValue;
38 }
39
40 -(void)alteraNome:(NSString *) novoNome
41 {
42         self.nome = novoNome;
43 }
44
45 -(void)alteraValor:(Float32) novoValor
46 {
47         self.valor =novoValor;
48 }
49
```

```
50 −(void) print;
51 {
52                     NSLog(@"%-31s\t",[nome UTF8String]);
53 }
54
55 −(void) print:(Estado ∗)enedereco
56 {
57         Estado∗ nEstado = enedereco;
58         NSLog(@"%-31s\t",[nEstado.nome UTF8String]);
59 }
60
61 −(float) getValor
62 {
63         return self.valor;
64 }
65
66 @end
```

D. CLASSE TPC

```
1  //
2  // Tpc.h
3  // tcm
4  //
5  // Created by Elisangela Boni on 26/10/11.
6  // Copyright 2011 EHL. All rights reserved.
7  //
8
9  #import <Foundation/Foundation.h>
10 #import <Foundation/NSArray.h>
11 #import <Foundation/NSObject.h>
12 #import "Estado.h"
13
14
15 @interface Tpc : NSObject {
16         NSMutableArray* valor_pais;
17         NSMutableArray* valor_local;
18         NSMutableArray* probabilidade;
19 }
20
21 @property (copy, nonatomic) NSMutableArray *valor_pais;
22 @property (copy, nonatomic) NSMutableArray *valor_local;
23 @property (copy, nonatomic) NSMutableArray *probabilidade;
24
25 -(id) init;
26 -(void) novoValor_pais: (NSMutableArray *) nVpais;
27 -(void) novoValor_local: (NSMutableArray *) nVlocal;
28 -(void) novoValor_probabilidade: (NSMutableArray *) nVprobabilidade;
29 //-(void) ajustaValores Estado:(NSString*) estado Valor:(NSNumber *) valor Pais(NSMutableArray *)tPais;
30 -(void) ajustaValor:(NSNumber *)valor estado:(NSString *)nEstado indice:(int)indice;
31 -(void) printTpc;
32 -(id) getEstadosPaisAtIndex:(int) indice;
33 -(id) getValorLocalAtIndex:(int) indice;
34 -(id) getProbabilidadeAtIndex:(int) indice;
35 //-(NSNumber*) getValor:(NSMutableArray *) parametros;
36 -(int) tabelaContagem;
37 -(void) setValor:(float) valor Estado:(NSString*)estado;
38 -(NSNumber*) getValor:(Estado*)valor evidencia:(NSMutableArray *)parametros;
39
40
41 @end
```

```
1  //
2  // Tpc.m
3  // tcm
4  //
5  // Created by Elisangela Boni on 26/10/11.
6  // Copyright 2011 EHL. All rights reserved.
7  //
8
9  #import "Tpc.h"
10
11 @implementation Tpc
12
13 @synthesize valor_pais,valor_local,probabilidade;
14
15 -(id) init
16 {
17         self = [super init];
18         if (self)
19         {
20                 valor_pais = [[NSMutableArray alloc] init];
21                 valor_local = [[NSMutableArray alloc] init];
22                 probabilidade = [[NSMutableArray alloc] init];
23
24         }
25         return self;
26 }
27
28 -(void) novoValor_pais: (NSMutableArray *) nVpais
29 {
30         [self.valor_pais addObject:nVpais];
31 }
32
33 -(void) novoValor_local: (NSMutableArray *) nVlocal
34 {
35         [self.valor_local addObject:nVlocal];
36 }
37
38 -(void) novoValor_probabilidade: (NSMutableArray *) nVprobabilidade
39 {
40         [self.probabilidade addObject:nVprobabilidade];
41 }
42
43 -(void) printTpc
44 {
45         int linhas;
46         NSMutableString *linha = [[[NSMutableString alloc]init]autorelease];
47         linhas=[self tabelaContagem];
48         int i;
49         NSMutableSet *tEstadoPais = nil;
```

```
50          NSMutableSet *tValorLocal = nil;
51          for (i=0;i<linhas;i++)
52          {
53                  tEstadoPais=[self getEstadosPaisAtIndex:i];
54                  for (Estado *nEstado in tEstadoPais)
55                          [linha insertString:[NSString stringWithFormat:@"%-12s\t",[nEstado.nome
                                    UTF8String]] atIndex:[linha length]];
56
57                  tValorLocal = [self getProbabilidadeAtIndex:i];
58                  for (NSNumber *nValor in tValorLocal)
59                          [linha insertString:[NSString stringWithFormat:@"%2.5f\t",[nValor floatValue
                                    ]] atIndex:[linha length]];
60
61                  NSLog(@"%@",linha);
62                  [linha setString:@""];
63          }
64  }
65
66  -(void) ajustaValor:(NSNumber *)valor estado:(NSString *)nEstado indice:(int)indice;
67  {
68  //
69  }
70
71  -(int) tabelaContagem
72  {
73          return [self.valor_pais count];
74  }
75
76
77  -(id) getEstadosPaisAtIndex:(int) indice
78  {
79          return [self.valor_pais objectAtIndex:indice];
80  }
81
82  -(id) getValorLocalAtIndex:(int) indice
83  {
84          return [self.valor_local objectAtIndex:indice];
85  }
86
87  -(id) getProbabilidadeAtIndex:(int) indice
88  {
89          return [self.probabilidade objectAtIndex:indice];
90  }
91
92  -(NSNumber*) getValor:(Estado*)valor evidencia:(NSMutableArray *)parametros
93  {
94          NSMutableArray *linha = nil;
95          NSMutableArray *valores = nil;
96          NSMutableArray *probabilidades = nil;
```

```
97          Estado* estado = valor;
98          // verificar os parametros. Linha tem estados. Parametros tem nodos
99          int indice,indice2,i;
100         if ([[self.valor_pais objectAtIndex:0] count]==[parametros count])
101                 for (indice=0;indice<[self.valor_pais count];indice++)
102                 {
103                         BOOL igual = YES;
104                         linha = [self.valor_pais objectAtIndex:indice];
105                         for (i=0;i<[parametros count];i++)
106                             if ([linha containsObject:[parametros objectAtIndex:i]]==NO)
107                                 {
108                                         igual = NO;
109                                         break;
110                                 }
111                         if (igual==YES)
112                         {
113                                 valores = [self getValorLocalAtIndex:indice];
114                                 for (indice2=0; indice2<[valores count];indice2++)
115                                         if (estado==[valores objectAtIndex:indice2])
116                                         {
117                                                 probabilidades = [self getProbabilidadeAtIndex:indice];
118                                                 NSLog(@"%10.3f",[[probabilidades objectAtIndex:
                                                        indice2] floatValue]);
119                                                 return [probabilidades objectAtIndex:indice2];
120                                         }
121                         }
122                 }
123         NSNumber* zero = [NSNumber numberWithFloat:0.0];
124         return zero;
125 }
126
127
128 -(void) setValor:(float) valor Estado:(NSString*)estado
129 {
130         // nada
131 }
132 @end
```

E. CLASSE INFERENCIA

```
1  //
2  // inferencia.h
3  // tcm
4  //
5  // Created by Elisangela Boni on 23/08/12.
6  // Copyright 2012 EHL. All rights reserved.
7  //
8
9  #import <Cocoa/Cocoa.h>
10
11
12 @protocol inferencia <NSObject>
13
14 @required
15 -(NSNumber*) calcula_inferencia:(NSMutableArray*) evidencias;
16
17 @end
18
19
20 //
```

F. CLASSE INFERENCIAEXATA

```
1  //
2  // inferenciaExata.h
3  // tcm
4  //
5  // Created by Elisangela Boni on 14/07/12.
6  // Copyright 2012 EHL. All rights reserved.
7  //
8
9  #import <Foundation/Foundation.h>
10 #import "inferencia.h"
11
12 @interface inferenciaExata : NSObject <inferencia>
13 {
14 }
15 -(NSNumber*) calcula_inferencia:(NSMutableArray *) evidencias;
16 @end
```

```
1  //
2  // inferenciaExata.m
3  // tcm
4  //
5  // Created by Elisangela Boni on 14/07/12.
6  // Copyright 2012 EHL. All rights reserved.
7  //
8
9  #import "inferenciaExata.h"
10
11
12 @implementation inferenciaExata
13
14 -(NSNumber*) calcula_inferencia:(NSMutableArray *) evidencias;
15 {
16         NSLog(@"Inferencia Exata");
17         return nil;
18 }
19
20 @end
```

G. CLASSE INFERENCIAAPROXIMADA

```
1  //
2  // inferenciaAproximada.h
3  // tcm
4  //
5  // Created by Elisangela Boni on 14/07/12.
6  // Copyright 2012 EHL. All rights reserved.
7  //
8
9  #import <Foundation/Foundation.h>
10 #import "inferencia.h"
11
12 @interface inferenciaAproximada : NSObject <inferencia>
13 {
14 }
15 -(NSNumber*) calcula_inferencia:(NSMutableArray*) evidencias;
16 @end
```

```
1  //
2  // inferenciaAproximada.m
3  // tcm
4  //
5  // Created by Elisangela Boni on 14/07/12.
6  // Copyright 2012 EHL. All rights reserved.
7  //
8
9  #import "inferenciaAproximada.h"
10
11 @implementation inferenciaAproximada
12
13 -(NSNumber*) calcula_inferencia:(NSMutableArray*) evidencias;
14 {
15         NSLog(@"Inferencia Aproximada");
16         return nil;
17 }
18
19 @end
```