UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA

**VISUALIZADOR CONTEXTUAL DE IMAGENS
MÉDICAS**

RAMON ALFREDO MORENO

Tese apresentada à Escola Politécnica da
Universidade de São Paulo, para concorrer
ao título de Doutor, pelo curso de Pós-
Graduação em Engenharia Elétrica – Área
de concentração: Sistemas Eletrônicos

São Paulo
2005

UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA

**VISUALIZADOR CONTEXTUAL DE IMAGENS
MÉDICAS**

RAMON ALFREDO MORENO

Tese apresentada à Escola Politécnica da
Universidade de São Paulo, para concorrer
ao título de Doutor, pelo curso de Pós-
Graduação em Engenharia Elétrica – Área
de concentração: Sistemas Eletrônicos

Orientador: Sérgio Shiguemi Furuie

São Paulo
2005

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha Catalográfica

Escola Politécnica da Universidade de São Paulo

Moreno, Ramon Alfredo

Visualizador Contextual de Imagens Médicas. / Ramon Alfredo Moreno; orientador Sérgio Shiguemi Furuie - São Paulo, 2005.

173p. : fig.

Tese (Doutorado – Programa em Pós Graduação em Engenharia Elétrica. Área de Concentração: Sistemas Eletrônicos) - Escola Politécnica da Universidade de São Paulo.

1. Contexto. 2. Imagens Médicas. 3. Visualização. 4. Interfaces Inteligentes I. Título.

“No fim, tudo é uma piada”

Charles Chaplin

Data da Defesa: ____/____/____

Banca Examinadora

Prof. Dr. Sérgio Shiguemi Furuie

Julgamento _____ Assinatura: _____

Prof. Dr. Marco Antônio Gutierrez

Julgamento _____ Assinatura: _____

Prof. Dr. Marcelo Knörich Zuffo

Julgamento _____ Assinatura: _____

Prof. Dr. Agma Juci Machado Traina

Julgamento _____ Assinatura: _____

Prof. Dr. Paulo Mazzoncini de Azevedo Marques

Julgamento _____ Assinatura: _____

*A meus pais, René e Margarita, pelo seu
contínuo amor e atenção, e a meus irmãos,
Ignácio, Patrícia e Mariano por sempre
estarem presentes quando necessário.*

AGRADECIMENTOS

Em primeiro lugar, agradeço a meu orientador, Sérgio Shiguemi Furuie, pela confiança, apoio contínuo e por sempre me dar um prazo extra.

Aos amigos e família, pelos fins-de-semana e horas nas quais não pude dar a devida atenção a vocês.

Esta tese não teria sido possível sem a ajuda de Marina Rebelo, que tem sido uma amiga, confidente, revisora, e uma fonte interminável de apoio moral.

Agradeço também aos colegas de trabalho Nivaldo Bertozzo e Marcelo Santos pela amizade, diversão e por ficarem com o trabalho duro enquanto eu escrevia a tese.

A Júlio Figueiredo, por mostrar que a pesquisa científica pode ser divertida.

À professora Cecil Robilotta pela minuciosa revisão do trabalho, amizade e sabedoria compartilhada.

Ao professores Marco Antonio Gutierrez, Marcelo Zuffo e Tereza Carvalho pelas sugestões e contribuições para a tese.

À Natalia Rosa e Agma Traina, por gentilmente cederem o algoritmo de histograma métrico.

A Giuliana Orlandi e Marcos Yamaguti pela amizade, paciência e por sempre me auxiliarem quando eu precisei.

Aos amigos das unidades de Suporte e Sistemas do Serviço de Informática do InCor que direta ou indiretamente auxiliaram no desenvolvimento desta tese.

Às secretárias do Serviço de Informática, Sílvia Marina e Lúcia, pela disposição em atender as minhas solicitações.

Ao Dr. Umberto Tachinardi, pela determinação e por mostrar que nunca é tarde para começar algo novo.

Aos amigos Luiz Kobayashi, Fabiane Nardon, Wietske Meyering, Paulo Pilon, Keylla Urtiga, Ângela Vilhena, Vivian Fiales, Almir Velho, Alice Bacic e Alexandre Murakami pelas conversas e companhia.

Ao 'Dr' Rodrigo Lacerda, pelo incentivo a distância.

Ao amigo Valci Balbino Jr, por mostrar que sempre existe uma maneira mais inteligente de ver as coisas.

Ao Serviço de Informática do InCor, pelas inúmeras oportunidades oferecidas para crescer pessoalmente e profissionalmente.

Às secretárias da Pós-Graduação da Escola Politécnica da USP, pela atenção, eficiência e educação dispensadas.

Ao apoio financeiro concedido pela FAPESP inicialmente com bolsa de mestrado, processo 1997/14206-5, posteriormente convertida em doutorado direto, processo 2001/05987-0 e cancelada por motivo de contratação no InCor. Agradeço também ao CNPq, pelo apoio ao projeto MiniWEBPACS, processo 680121/2003-0. Sem o suporte dessas instituições o desenvolvimento da tese não teria sido possível.

A todos aqueles que de forma direta ou indireta ajudaram na elaboração deste trabalho.

SUMÁRIO

1.Introdução.....	1
1.1 Visualização contextual de Imagens	4
1.2 Objetivo	5
1.3 Justificativa.....	5
1.4 Estrutura e Organização	7
2. Conceitos Básicos	9
2.1 Contexto.....	9
2.1.1 Definição.....	10
2.1.2 Representação e uso do contexto.....	14
2.1.3 Ontologias	25
2.1.3.1 Protégé 2000	26
2.2 Representação de informação	29
2.3 Sistemas de Informação em Hospitais.....	33
2.4 PACS-HC	39
2.5 Visualização e processamento de imagens	43
3. Contexto e Regras	47
3.1 Modelo de Contexto	47
3.1.1 Infra-estrutura.....	49
3.1.2 Fluxo de Trabalho.....	51
3.1.3 Sistema Informatizado	52
3.1.4 Recursos humanos	55
3.1.5 Raciocínio Clínico	57
3.2 Sistema de Inferência.....	59
3.3 Exemplos de Aplicação do Modelo de Contexto	61
3.3.1 CASO 1: Recuperação de imagem semelhante	61
3.3.2 CASO 2: Notificação de exames do HIS.....	66
4. Visualizador Contextual: Arquitetura e Implementação	71
4.1 Arquitetura de software	71
4.2 Fontes de dados desenvolvidas para o projeto.....	72
4.2.1 Servidor DICOM	73

4.2.2.1 C-STORE.....	75
4.3.2.2 C-MOVE.....	76
4.3.2.3 C-FIND.....	77
4.2.2 CIAS	79
4.2.3 BIRAM	82
4.3 Implementação do visualizador contextual	88
5. Discussão.....	105
5.1 Contribuição do contexto para visualização	105
5.1.1 Visualização de Imagens	106
5.1.2 Adaptação de acordo com o contexto.....	108
5.2 Vantagens e desvantagens	113
5.3 Trabalhos futuros.....	113
6. Conclusões	115
Apêndice A – Conceitos Básicos de OO	117
A.1 Orientação a Objeto	117
A.2 Diagramas UML.....	121
A.2.1 Diagrama de classe	122
A.2.2 Diagrama de caso de uso	126
A.2.3 Diagrama de seqüência	127
A.3 Design Patterns	128
Apêndice B – Java	135
B.1 Introdução	135
B.2 Java Advanced Imaging (JAI).....	137
B.3 Java3D.....	137
Referências Bibliográficas	139
Produção bibliográfica e técnica.....	149
Trabalhos em revista.....	149
Trabalhos em congressos internacionais	149
Trabalhos em congressos nacionais	150
Apresentações e palestras	152

LISTA DE FIGURAS

Figura 1 - Esquema geral do projeto PACS-HC.....	3
Figura 2 – Contexto é tudo exceto a entrada e saída explícita de um sistema, segundo Lieberman e Selker (2000).	11
Figura 3 – Uma Stick-e note. Extraído de (Brown, 1996).....	14
Figura 4 – Arquitetura do Serviço de Informação Contextual. Extraído de (Judd e Steenkiste, 2003).....	15
Figura 5 – Solicitação de serviço ao <i>middleware</i> . Extraído de Capra (2001).	16
Figura 6 – Representação do contexto e ação correspondente. Os nós à esquerda representam uma condição a ser satisfeita e o nó à direita a ação correspondente quando a condição for verdadeira. Extraído de Perich (2002).	17
Figura 7 – Regras do ATT-Meta com nível de incerteza. Extraído de Barnden e Lee (1999).....	18
Figura 8 – Esquema do sistema ORIMUHS. Extraído de Encarnação e Stoev (1999). ..	19
Figura 9 –Componentes da Arquitetura de Gerenciamento de Contexto do HL7. Extraído de CCOW (2003).	21
Figura 10 – Diagrama geral da arquitetura do CoBrA (Extraído de Chen et al, 2004). ..	22
Figura 11 – Exemplo de uma descrição de uma ontologia e instâncias.	26
Figura 12 – Ontologia com elementos básicos.....	27
Figura 13 – Definição de uma classe no Protégé.....	27
Figura 14 – Conjunto de classes definidas em um projeto.....	28
Figura 15 – Definição de um <i>slot</i> , ou propriedade de uma classe.	28
Figura 16 – Definição de uma instância de uma classe.	29
Figura 17 – Laudo de laboratório codificado como texto.....	30
Figura 18 – Resultado laboratorial codificado utilizando XML.....	31
Figura 19 – Lista de modalidades segundo a especificação DICOM.....	32
Figura 20 – Modelo de uma observação segundo a especificação COAS (Extraído de OMG 2001b).....	38
Figura 21 – Exemplo de diferentes representações da informação utilizando observações do COAS.....	38

Figura 22 – Solução proposta para interoperabilidade semântica através da conversão de DICOM SR para CORBAMED.....	39
Figura 23 – Visão geral do sistema de informações.	40
Figura 24 – Modelo geral de imagens médicas.	41
Figura 25 – Imagens médicas. a) Eletrocardiograma b) Radiografia c) Angiografia d) por Ressonância Magnética (Extraído do <i>site</i> do programa MRIcro).....	44
Figura 26 – Passos fundamentais do processamento de imagens (Gonzalez, 2000).....	45
Figura 27 – Modelo de representação do hospital.	48
Figura 28 – Especialização da infra-estrutura, representando elementos físicos, como computadores, leitos e redes.....	48
Figura 29 – Modelo de um computador como componente de infra-estrutura.	50
Figura 30 – Associação da Infra-estrutura com uma localização.	50
Figura 31 – Exemplo da representação de um fluxo de trabalho para uma tomografia de tórax.	51
Figura 32 – Representação diagramática do fluxo de trabalho (simplificado) para aquisição de um CT.	52
Figura 33 – Modelo de um sistema informatizado.	53
Figura 34 – Modelo de um Documento Eletrônico.	54
Figura 35 – Extensão do modelo para incluir a representação de um servidor DICOM como uma fonte de dados. Ressaltado em cinza, aparecem os modelos apresentados nas Figura 33 e Figura 34.....	55
Figura 36 – Modelo dos Recursos Humanos.....	56
Figura 37 – Papéis de um Profissional de Saúde.	57
Figura 38 – Modelo de fluxogramas para procedimentos médicos e <i>guidelines</i>	58
Figura 39 – Hipertensão pulmonar e <i>cor pulmonale</i> na DPOC (Doença Pulmonar Obstrutiva Crônica) (Extraído de Lopes, 1995).	58
Figura 40 – Exemplo de informação do usuário contida no modelo. À esquerda se encontra o modelo, no centro estão as instâncias e à direita estão as informações de cada instância.	62
Figura 41 – Ilustração didática para a linguagem de pesquisa dos dados.	63
Figura 42 – Papeis do médico.....	64
Figura 43 – Regra para busca automática por imagens.	64

Figura 44 – Regra que determina o algoritmo mais apropriado para pesquisa no BIRAM.....	65
Figura 45 – Regras para busca de exames no HIS do InCor.	67
Figura 46 – Resultado de pesquisa no HIS do InCor.....	67
Figura 47 – Unidade Clínica de Emergência.	68
Figura 48 – Regra que permite a busca por localização.	69
Figura 49 – Regra modificada. Mudanças em tom mais escuro.....	70
Figura 50 – Gráfico de quantidade de creatinofosfoquinase (CK-MB) de um paciente ao longo do tempo.	70
Figura 51 – Esquema geral do Visualizador Contextual.	72
Figura 52 – Diagrama de tabelas utilizadas na implementação do servidor DICOM.....	74
Figura 53 – Tabela para mapeamento de parâmetros DICOM para colunas do banco de dados.....	77
Figura 54 – Esquema geral de funcionamento do CORBA.....	79
Figura 55 – Relacionamento entre a interface do CIAS (ImageAccess) e a interface do COAS (Query Access).....	80
Figura 56 – Interface de pesquisa de imagens (esquerda) e interface de busca de observações clínicas (direita).....	81
Figura 57 – Representação de uma imagem segundo o modelo do CIAS.....	81
Figura 58- Tabelas do CIAS.	82
Figura 59 – Diagrama das tabelas do BIRAM.....	83
Figura 60 – Diagrama de sequência exemplificando o funcionamento do BIRAM.....	84
Figura 61 – Modelo de um algoritmo no BIRAM. A classe <i>Algoritmo</i> descreve informações sobre o tipo de imagens a que ele se aplica (modalidade, número de bits e parte do corpo). O método <i>process(imagem)</i> da classe <i>Algoritmo</i> gera um objeto do tipo <i>Resultado</i> , o qual é armazenado no banco de forma serializada, isto é, convertido em um <i>array</i> de bytes.....	85
Figura 62 – Cliente para digitação de laudos. À esquerda, texto livre e, à direita, laudo codificado com CID-10.	87
Figura 63 – Abstração de um documento.....	88
Figura 64 – Esquema de classes dos <i>plugins</i>	90
Figura 65 – Componente reutilizável, “ <i>widget</i> ”.	92

Figura 66 – Gerenciador de Interface e Gerenciador de Plugins.....	93
Figura 67 – Tela inicial do visualizador contextual, requerindo <i>login</i>	94
Figura 68 – Diagrama de seqüência, mostrando a inicialização do visualizador contextual.....	95
Figura 69 - Visualização de imagem 1D.	95
Figura 70 – À esquerda, visualização de imagem bidimensional DICOM, com medida de intensidade. À direita, visualização de imagem 2D dinâmica (somente um frame).....	96
Figura 71 – Visualização de imagem de <i>Gated Blood Pool</i> em modo cine.	97
Figura 72 – Cálculo de fração de ejeção sobre imagens de <i>Gated Blood Pool</i> , com o uso de regiões de interesse.	97
Figura 73 – Visualização de imagem 3D utilizando a biblioteca Visad. À esquerda, cabeça humana e à direita, <i>phantom</i> MCAT (http://www.bme.unc.edu/mirg/mcat/).	98
Figura 74 – Imagem de radiográfica de tórax processada com brilho e contraste modificados, rotacionada e com dimensão reduzida.	99
Figura 75 – Configurações dos <i>plugins</i> . À esquerda, tela inicial de configuração. À direita, configuração do servidor DICOM.....	99
Figura 76 – Interface de pesquisa do BIRAM com lista de resultados.	100
Figura 77 – Interface de pesquisa do CIAS para chamada pelo método <i>get_observation_by_qualified_code()</i>	101
Figura 78 – Interface de pesquisa do Servidor DICOM com lista de resultados.	101
Figura 79 – Busca no <i>site PubMed</i>	102
Figura 80 – Visualização de texto recuperado do <i>site PubMed</i>	102
Figura 81 – Representação do modelo de contexto no programa Protégé (2004).....	103
Figura 82 – Tela de entrada de dados do Protégé.	103
Figura 83 – Visualizador genérico de imagens ImageJ.	107
Figura 84 – Visualizador específico MVE.	107
Figura 85 – Modelo de um ECG.....	111
Figura 86 – Enfoque baseado em sistema versus enfoque baseado em objeto (Furlan, 1998).....	117
Figura 87 – Representação de um objeto.....	118

Figura 88 – Troca de mensagem entre objetos.....	119
Figura 89 – Classes e instâncias.	120
Figura 90 – Herança.....	121
Figura 91 - Representação de uma Classe segundo a UML (Furlan, 1998).....	123
Figura 92 – Papéis em associação (Furlan, 1998).	123
Figura 93 - Exemplo de multiplicidade um para muitos.	124
Figura 94 – Exemplo de associação qualificada (Furlan, 1998).....	124
Figura 95 – Relação de agregação.	125
Figura 96 - Relação de composição.....	125
Figura 97 – Navegabilidade.....	125
Figura 98 – Especialização / generalização.	126
Figura 99 – Diagrama de casos de uso.	126
Figura 100 – Exemplo de Caso de Uso.	127
Figura 101 – Diagrama de sequência.....	127
Figura 102 – Estrutura de um <i>Singleton</i>	129
Figura 103 – Exemplo de uma <i>Abstract Factory</i>	130
Figura 104 – Exemplo do padrão <i>Factory Method</i>	131
Figura 105 – Exemplo do padrão <i>Mediator</i>	132
Figura 106 – Estrutura do padrão <i>Observer</i>	132
Figura 107 – Exemplo do padrão <i>Strategy</i>	133
Figura 108 – Exemplo do padrão <i>Visitor</i>	134
Figura 109 – Funcionamento da linguagem Java (Extraído de http://java.sun.com/docs/books/tutorial/getStarted/cupojava/win32.html).	136
Figura 110 – Organização dos grafos de cenas do Java 3D.	138

LISTA DE TABELAS

Tabela 1 – Resumo das diferentes abordagens para representação e utilização de contexto.....	23
Tabela 2 – Representação do diagrama da Figura 39 utilizando-se o modelo de raciocínio clínico.	59
Tabela 3 – Resumo dos padrões de projeto e sua classificação (Extraído de Gamma, 2000).....	128

LISTA DE ABREVIATURAS E SIGLAS

1D	Uma dimensão.
2D	Duas dimensões.
3D	Três dimensões.
ACR-NEMA	American College of Radiology / National Electrical Manufacturers Association.
AI	Artificial Intelligence.
API	Application Programming Interface
BIRAM	Base de Imagens Relacional para Algoritmos e Métricas
CBIR	Content Based Image Retrieval
CBR	Case Based Reasoning
CCOW	Clinical Context Object Workgroup.
CFM	Conselho Federal de Medicina.
CIAS	Clinical Image Access Service.
CID-10	Classificação Estatística Internacional de Doenças e Problemas Relacionados à Saúde, 10ª Revisão.
CK-MB	Creatinofosfoquinase
COAS	Clinical Observation Access Service.
CORBA	Common Object Request Broker Architecture.
CORBAMed	CORBA in Medicine.
CSL	Context Specification Language
CT	Computed Tomography.
CTN	Central Test Node
DICOM	Digital Imaging and Communications in Medicine.
DICOM SR	DICOM Structured Reporting
ECG	Eletrocardiograma.
EPR	Electronic Patient Record.
FITS	Flexible Image Transport System.
FMUSP	Faculdade de Medicina da Universidade de São Paulo
GPS	Global Positioning System
HIS	Hospital Information System.

HL7	Health Level 7.
HTTP	HyperText Transfer Protocol
ICU	Intensive Care Unit.
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
INCOR	Instituto do Coração do Hospital das Clínicas
INRAD	Instituto de Radiologia do Hospital das Clínicas
JADE	Java Agent DEvelopment Framework
JVM	Java Virtual Machine
MACA	Middleware de Autenticação e Controle de Acesso.
MBR	Memory Based Reasoning
MR	Magnetic Ressonance.
NLM	National Library of Medicine
ODIN	Observador Dinâmico de Imagens N-dimensionais
OLAP	OnLine Analytical Processing
OMG	Object Management Group
PACS	Picture, Archiving and Communication System.
PACS-HC	Projeto PACS para o Hospital das Clínicas de São Paulo
PEP	Prontuário Eletrônico do Paciente.
PET	Positron Emission Computed Tomography.
RAD	Resource Access Decision.
RAM	Random Access Memory
RDF	Resource Description Framework
RGB	Red, Green and Blue.
RIS	Radiological Information System.
RMI	Remote Method Invocation.
RSNA	Radiological Society of North America
SCAR	Society for Computer Applications in radiology.
SGML	Standard Generalized Markup Language
SNOMED	Systematized Nomenclature of Human and Veterinary Medicine.
SPECT	Single Photon Emission Computed Tomography.
SQL	Structured Query Language

UML	Unified Modeling Language.
UMLS	Unified Medical Language.
UTI	Unidade de Tratamento Intensivo
VISAD	Visualization for Algorithm Development
WWW	World Wide Web.
XA	Angiografia
XML	Extended Markup Language.
XY	Coordenadas cartesianas

RESUMO

MORENO, R. A. **Visualizador Contextual de Imagens Médicas**. 2005. 173 p. Tese (Doutorado) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2005.

O Prontuário Eletrônico do Paciente deve, idealmente, agregar todas as possíveis fontes de informação de um paciente, independentemente de sua localização e forma de armazenamento, e permitir às pessoas autorizadas acessar esses dados. Porém, na prática, os sistemas de informação médica são heterogêneos e de difícil integração, e as informações médicas são altamente complexas e pouco estruturadas (Klein, 2002). No sentido de integrar as informações médicas de sistemas heterogêneos e distribuídos desenvolveu-se, no Instituto do Coração, o projeto PACS-HC, com foco em imagens médicas. Neste trabalho, propõe-se o componente final de acesso às informações do PACS-HC, o Visualizador Contextual de Imagens Médicas. O visualizador proposto é capaz de realizar, além da visualização, processamento e pesquisa de imagens médicas, tarefas como a recuperação de imagens semelhantes em bases de dados de patologias e a recuperação de informações bibliográficas, utilizando o contexto dos dados. Para isso, criou-se um modelo de contexto, implementado na forma de uma ontologia utilizando-se o software Protégé. Também se desenvolveu uma arquitetura extensível e robusta para processamento e visualização de imagens médicas. Como resultado, apresenta-se uma implementação, denominada ODIN, capaz de visualizar imagens de diferentes modalidades e dimensões e utilizar dados contextuais através de regras. Conclui-se que é possível e desejável a utilização de informação contextual para melhorar a interação com o usuário e percebe-se que há um potencial ainda maior para o uso de informações clínicas, como laudo e *guidelines*, no auxílio do Profissional de Saúde. Como trabalhos futuros pode-se citar a pesquisa de novas formas de utilização do contexto, por exemplo, com o uso de agentes inteligentes (DeLoack, 1999), e o refinamento do modelo do contexto a partir de casos de uso que evidenciem as necessidades clínicas do profissional de saúde em diferentes ambientes e situações.

Palavras-chave: contexto, imagens médicas, visualização, arquitetura.

SUMMARY/ABSTRACT

MORENO, R. A. **Contextual Medical Image Viewer**. 2005. 173 p. Thesis (Doctoral) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2005.

The Electronic Health Record should, theoretically, aggregate all possible sources of information from a patient, no matter their location or storage format, and allow authorized people to access this information. However, in practice, the medical information systems are heterogeneous and hard to integrate, besides the fact that medical information is complex and poorly structured. To handle this problem the Heart Institute of São Paulo leaded a project named PACS-HC with the aim of integrating medical information from heterogeneous systems, focused mainly on medical images. This work proposes the visualization component of the PACS-HC, called Contextual Medical Image Viewer. The proposed viewer is capable of performing automatic actions using contextual information, like user role, location and image dimension. Some examples of actions are the retrieval of similar images from a case image database and reference information from the WEB. The contextual viewer uses a context model, implemented as an ontology, to perform those actions. It also relies on an extensible architecture for supporting the processing and visualization of medical images. The result is the software ODIN, which is capable of displaying images from different modalities and dimensions and using contextual information to perform actions using rules. It can be concluded that the use of contextual information is possible and desirable as a way to improve user interaction and it must be added that there is an extra potential for the use of medical information, not only image-related, but to improve the healthcare attention. As future work there are investigation of new ways of using contextual information, such as through intelligent agents, and the refinement of the context model from use cases that reveal the needs of physicians and healthcare professionals in different environments and situations.

Key words: context, medical images, visualization, architecture

Introdução

*“Todo o meu saber consiste
em saber que pouco sei”*

Sócrates

O Prontuário Eletrônico do Paciente (PEP) é, segundo o *Institute of Medicine* (IOM 1997), “um registro eletrônico que reside em um sistema especificamente projetado para apoiar os usuários, fornecendo acesso a um completo conjunto de dados corretos, alertas, sistemas de apoio à decisão e outros recursos, como *links* para bases de conhecimento médico” (tradução de Massad et al (2003)).

A busca por um PEP eficiente e completo tem estimulado o trabalho de diversos grupos de pesquisa como Grimson (2001), Berg (2001), Katehakis (2000), Cimino (2000), Adelhard (1999), Tachinardi (1995). As vantagens potenciais do PEP sobre o prontuário em papel envolvem o acesso, maior disponibilidade, maior rapidez na localização de informações, melhor qualidade e confiabilidade. O uso simultâneo e distribuído deste conjunto de informações por diversos profissionais é um atrativo adicional. A integração de dados de diferentes sistemas torna possível uma redução na necessidade de transcrições manuais, que apresentam riscos de erros significativos. A capacidade de se implantarem mecanismos de controle de acesso, sistemas de auditoria e assinaturas digitais assegura níveis de segurança maiores do que os equivalentes em papel.

Idealmente, o Prontuário Eletrônico deve agregar todas as possíveis fontes de informação de um paciente, mesmo se a informação estiver distribuída em lugares distintos ou mesmo em países diferentes, e torná-las acessíveis às pessoas autorizadas.

Existe, no entanto, uma enorme lacuna entre a definição ideal e a existência real de um PEP padronizado. Sob o ponto de vista tecnológico, os sistemas de informação médica atuais são baseados em diferentes plataformas de hardware e software. Além disso, ainda não existem modelos de dados, vocabulários e conjuntos de cenários comuns entre instituições. Espera-se que o valor e a funcionalidade do PEP evoluam e acompanhem tanto as exigências, em constante modificação pelas Instituições de Saúde, quanto a evolução tecnológica das plataformas, sobre as quais o sistema é construído.

Os avanços tecnológicos atuais contribuíram para que algumas das barreiras presentes nas primeiras experiências descritas de prontuário eletrônico fossem vencidas, como por exemplo, a universalização do uso do *World Wide Web* (Tachinardi, 1995). Por outro lado, outras questões extremamente importantes vieram à tona como, por exemplo, a confidencialidade de dados de Saúde em meios eletrônicos e legislação específica (Liu 2001).

O Serviço de Informática do Instituto do Coração (InCor) do Hospital das Clínicas da Universidade de São Paulo iniciou o desenvolvimento de seu prontuário eletrônico em meados da década de 90 utilizando, de forma pioneira, a Web como interface universal para o prontuário (Tachinardi 1995), pois um dos objetivos era fornecer um sistema integrado de acesso a todos os dados do paciente, com uma interface simples e padronizada. A implantação do prontuário envolveu o desenvolvimento de interfaces entre diferentes sistemas, como o Sistema de Informações Hospitalares (HIS) e o Sistema de Armazenamento e Transmissão de Imagens (PACS). Ao longo dos anos, novas funcionalidades foram acrescentadas ao prontuário, como a incorporação de documentos digitalizados (Furuie, 2001), substituindo os equivalentes em papel, e a possibilidade de acompanhamento de sinais de monitoração de beira de leito em tempo real (Oliveira, 2002).

O InCor é parte do complexo Hospital das Clínicas da FMUSP (HC-FMUSP), que reúne ainda em seu conjunto os Institutos Central (IC), da Criança (ICr), Instituto de Radiologia (InRad), Ortopedia e Traumatologia (IOT), de Psiquiatria (IPq) e hospitais-satélites. A troca de informações eletrônicas entre os diferentes institutos é um dos requisitos básicos para uma otimização de processos e custos. Nesse contexto, a questão de integração de informações hospitalares passou a representar um grande desafio. Com o objetivo de propor uma arquitetura de integração de informações entre sistemas

heterogêneos e distribuídos, o grupo de Informática do InCor uniu esforços com o InRad no desenvolvimento do projeto chamado “*Desenvolvimento de um modelo de ambiente distribuído para transmissão, arquivamento, recuperação, processamento e visualização de imagens médicas*”. O projeto, denominado PACS-HC, modelou as informações relacionadas às imagens. Entre as funções projetadas, incluem-se operações como buscas complexas a partir de qualquer atributo ou informação. Na Figura 1 apresenta-se uma visão geral do projeto PACS-HC.

Figura 1 - Esquema geral do projeto PACS-HC.

A recuperação e a manipulação eficientes das imagens médicas exigem que as informações associadas, como dados do paciente, procedimento realizado, laudo do exame, entre outros, estejam disponíveis e caracterizadas de forma a permitir consultas complexas, contextualizando-se, assim, a imagem estudada.

Existem dois problemas básicos que devem ser tratados na integração da informação em Saúde: o primeiro é um problema tecnológico e diz respeito a como integrar sistemas distribuídos que utilizam plataformas de hardware e software heterogêneas; o segundo está em um nível mais semântico: como integrar informação que está representada de maneiras completamente diferentes nas diversas bases de dados envolvidas. Além destes dois problemas, deve-se considerar, ainda, que a informação em Saúde é altamente complexa e pouco estruturada.

Para se atingir um grau de integração mais complexo, optou-se pela utilização de padrões. Dentre os padrões adotados no projeto PACS-HC, alguns ocupam papel de destaque: UML (*Unified Modeling Language*) (OMG, 1996), linguagem orientada a objetos para modelagem de sistemas; DICOM (ACR-NEMA, 2000), padrão para armazenamento e transmissão de imagens médicas; Middleware CORBA (*Common Object Request Broker Architecture*) (OMG, 1996), padrão aberto para desenvolvimento de sistemas que utilizem objetos distribuídos integrados em plataformas diferentes; PIDS (*Person Identification Service*) (OMG, 1998), COAS (*Clinical Observation Access Service*) (OMG, 2001b) e CIAS (*Clinical Images Access Service*) (OMG, 2001a) especificações aprovadas pelo comitê CORBAMED (OMG 1999), responsável pela padronização de serviços de software na área da Saúde.

Uma parte do projeto PACS-HC era o desenvolvimento de visualizadores contextuais, que fossem capazes de se adaptar a partir do contexto da imagem considerando, por exemplo, dados como o tipo de usuário, imagem e dimensionalidade.

1.1 Visualização contextual de Imagens

Como foi dito anteriormente, o visualizador contextual nasceu a partir do projeto do sistema PACS-HC e tem como função exibir imagens de acordo com o contexto das mesmas.

Resumidamente, o visualizador contextual apresenta os seguintes requisitos:

- (i) Ele deve ser capaz de exibir imagens médicas de diversas dimensões, como raios X (2D) e angiografias (2D dinâmicas);
- (ii) Deve ser capaz de realizar manipulações sobre as imagens, como zoom, ajuste de brilho e contraste e aplicação de filtros digitais;
- (iii) Deve ser extensível, suportando novas ferramentas e novas formas de visualização;
- (iv) Deve ser capaz de recuperar informações a partir de diversas fontes de dados distribuídos e tratá-las uniformemente (DICOM, CORBA, etc);
- (v) Deve ser o único componente visualizado pelo usuário final e permitir ao mesmo realizar tarefas de revisão, análise e pesquisa;
- (vi) Deve exibir as imagens de acordo com o contexto das mesmas.

Os cinco primeiros requisitos já constituem um grande desafio, pois o acesso e visualização a dados distribuídos e heterogêneos não é uma tarefa simples. Foi necessário desenvolver uma arquitetura flexível e robusta para atender a esses requisitos. Porém, o sexto item é o maior desafio, uma vez que o contexto é difícil de ser definido, representado e utilizado de forma eficaz (Lieberman e Selker, 2000; Dey, 2000). Para resolver o problema do contexto foi preciso criar um modelo que representasse as informações digitais presentes em um hospital e desenvolver uma forma de utilizá-lo em benefício do usuário.

1.2 Objetivo

O objetivo deste trabalho é estudar a utilização de informação contextual para a visualização e processamento de imagens médicas no auxílio do profissional de saúde para a visualização e processamento de imagens médicas.

Como objetivos secundários têm-se:

- Permitir o acesso a dados heterogêneos e distribuídos dentro de um ambiente hospitalar;
- Ser capaz de exibir imagens médicas de diversas modalidades e dimensões;
- Criar um visualizador extensível, que suporte a incorporação de novas ferramentas, novas fontes de dados e formas de visualização;
- Implementar parte da arquitetura do projeto PACS-HC, especificamente, servidores de imagens DICOM, CIAS e BIRAM.

1.3 Justificativa

As técnicas de aquisição e processamento de imagens desenvolvidas nas últimas duas décadas mudaram sensivelmente a forma de atuação da Medicina. A Medicina do novo milênio é impensável sem a presença cada vez mais marcante das imagens no diagnóstico, tratamento e acompanhamento médicos. Houve, nos últimos anos, uma enorme expansão dos métodos de obtenção de imagens e de sua análise, resultando em exames mais completos e complexos, e com maior capacidade de diagnóstico e terapia. A criação de unidades como a Divisão de Diagnóstico por Imagens do Instituto do Coração do Hospital das Clínicas - FMUSP, por exemplo, ilustra uma tendência de

organizar parte do atendimento médico em torno destes equipamentos, sobretudo pela sua capacidade resolutiva.

Se, no campo da tecnologia de aquisição e processamento, as imagens médicas vêm tendo um desenvolvimento marcante, o mesmo não se pode dizer sobre a disponibilização destas imagens para o ambiente hospitalar. Em outras palavras, apesar da existência de ambientes distribuídos de cobertura mundial, como a Internet, e dos grandes hospitais terem a maior parte de seus equipamentos interligados em redes internas, a atividade de visualização de imagens ainda está muito organizada, física e logicamente, em torno do equipamento de aquisição e análise. Exemplos claros disto são as unidades de hemodinâmica e ressonância magnética do InCor, nas quais se localizam equipamentos específicos para a visualização e o pós-processamento dos exames ali realizados.

Contudo, o que poderia ser uma atividade descentralizada, em que médicos e outros profissionais de saúde tivessem acesso a imagens e dados médicos a partir de suas mesas de trabalho, ou de qualquer parte do hospital, ou mesmo de sua residência, encontra-se, ainda, praticamente restrita a salas e equipamentos especialmente desenhados para este fim.

O subsistema de visualização é o componente final de um sistema distribuído de transmissão, armazenamento e processamento de imagens médicas, e constitui a interface com o usuário e deve, portanto, ser o único componente visualizado pelo usuário final. É a interface almejada para o sistema descentralizado citado acima, que permitirá ao médico, ou profissional de saúde autorizado, ter acesso aos dados de imagem e realizar tanto tarefas de revisão (visualização de imagens integrantes do laudo) como tarefas de análise (extração de parâmetros qualitativos ou quantitativos que permitam a elaboração de laudos) e pesquisa.

Por outro lado, um visualizador contextual e inteligente permitirá um avanço na implementação do Prontuário Médico Eletrônico ou Prontuário Eletrônico do Paciente. Atualmente, a maioria dos Prontuários Médicos é baseada em textos e documentos, nos quais se incluem as impressões de dados de sinais (como o eletrocardiograma) e filmes de radiologia e são objeto de pesquisa de diversos centros universitários em todo o mundo. A próxima geração de Prontuários, que está sendo modelada a partir do

conceito de Prontuário Eletrônico, deverá conter as informações relevantes sobre o paciente em todos os formatos disponíveis, incluindo texto, imagem, vídeo e áudio.

1.4 Estrutura e Organização

Na metodologia proposta utilizou-se a notação UML (*Unified Modeling Language*), por ser a especificação mais adotada para desenvolvimento de software. Também se teve o cuidado de utilizar *Design Patterns* (Apêndice A) na arquitetura de classes, de forma a tornar a solução mais extensível e reconfigurável possível.

Para complementar a arquitetura do visualizador contextual, foram desenvolvidos três software: um servidor de imagens **DICOM**, o qual é o padrão para transmissão e armazenamento de imagens médicas; um servidor de imagens **CIAS**, este baseado na especificação da OMG (*Object Management Group*) e parte do CORBA *Healthcare* (antigo CORBAmed); e uma base de imagens e algoritmos chamada **BIRAM** (Base de Imagens Relacional para Algoritmos e Métricas), a qual permite a busca por conteúdo (*Content Based Image Retrieval* - CBIR), por exemplo, utilizando o histograma. O BIRAM oferece uma fonte de informações contextuais importante além de demonstrar que é possível a integração do visualizador com outros sistemas que compõem o prontuário eletrônico do paciente.

A implementação do visualizador contextual, chamada de ODIN (Observador Dinâmico de Imagens N-dimensionais), foi realizada utilizando-se a linguagem de programação Java, por ser multi-plataforma, menos suscetível a erros e com uma API bem desenvolvida, mais adequada às necessidades do projeto. O processo adotado para a implementação do ODIN foi o desenvolvimento evolucionário (Sommerville, 1996). Nesse processo, as atividades de especificação, projeto, implementação e testes de protótipo ocorrem simultaneamente. Com base em uma especificação preliminar, um protótipo inicial é projetado e implementado, para depois ser validado. O objetivo dos testes é observar as deficiências do software em atender os requisitos previamente estabelecidos, de modo que a especificação preliminar seja refinada e o processo se repita. Ao final de vários ciclos, espera-se obter um produto que atenda aos requisitos (Motta, 2003).

O trabalho está organizado da forma mostrada a seguir:

- Capítulo 2: demonstra em detalhes a arquitetura do projeto PACS-HC além de fazer uma revisão do significado do contexto e suas formas de representação;
- Capítulo 3: propõe um modelo para representação do contexto;
- Capítulo 4: mostra a arquitetura e implementação do servidor DICOM, servidor CIAS e BIRAM; Define a arquitetura de software idealizada para o desenvolvimento do visualizador contextual; Apresenta o programa ODIN e os resultados dessa implementação;
- Capítulo 5: discute as contribuições da tese, suas limitações e apresenta as possíveis extensões, visando seu aperfeiçoamento;
- Capítulo 6: aponta as conclusões do trabalho, ressaltando os objetivos alcançados e as contribuições para a comunidade científica.

Conceitos Básicos

“If we knew what it was we were doing, it would not be called research, would it?”

Albert Einstein

A visualização contextual de imagens médicas consiste em exibir as imagens de acordo com o seu contexto. Porém, o contexto para imagens médicas é variável de acordo com as informações disponíveis e da forma como elas estão representadas. Neste capítulo se apresenta, na seção 2.1, o conceito de contexto, formas para sua representação e utilização. Na seção 2.2 são discutidos os problemas da representação de informações clínicas e o uso de padrões. Em 2.3 se apresentam os sistemas de informação comumente presentes em Hospitais, os quais são fontes de informação contextual. Em 2.4 é revisto o projeto PACS-HC e, finalmente, em 2.5, apresentam-se brevemente considerações sobre a visualização e processamento de imagens.

2.1 Contexto

O uso do contexto tem como objetivo permitir “fazer a coisa certa” em uma determinada situação na interação do usuário com o programa. A coisa certa a ser feita pode ser a exibição uma mensagem de alerta, o realce de uma informação relevante ou a sugestão de um curso de ação. O que determina o que deve ser feito é o conjunto de informações disponível e o conhecimento existente para transformar a informação em uma ação. Assim, o problema existente para a utilização do contexto é o acesso e

manipulação da informação e a definição de maneiras para utilizar a informação em benefício do usuário.

A informação utilizada como contexto pode ser extremamente variada, por exemplo, o histórico de ações do usuário, sua localização, hora, tarefa sendo realizada, entre outros. Também é preciso considerar que a informação pode ser variante no tempo, por exemplo, a quantidade de memória disponível no computador, ou estática, como a localização física de um *desktop*. Dessa forma, o modelo de contexto, deve ser capaz de representar informações heterogêneas e complexas.

No item 2.1.1 é apresentada a definição de contexto, e, em 2.1.2, são apresentadas diferentes alternativas para representação e uso do contexto. Em 2.1.3 é introduzido o conceito de ontologia, solução adotada neste trabalho para representação do contexto, e demonstra-se o editor de ontologias Protégé 2000.

2.1.1 Definição

A palavra contexto vem do latim *contextus*, que significa conectar ou conexão, assim, o contexto é a conexão entre as palavras e o texto em uma sentença. Segundo o dicionário Houaiss, contexto é a “inter-relação de circunstâncias que acompanham um fato ou uma situação”, ou então “o conjunto de palavras, frases, ou o texto que precede ou se segue a determinada palavra, frase ou texto, e que contribuem para o seu significado”.

Porém, no nosso dia-a-dia, contexto pode ser considerado como qualquer informação relevante para uma atividade que estejamos realizando. Por ser um termo vago, muitos pesquisadores tentaram definir contexto de forma mais restrita, especialmente aqueles da área de *context-aware computing* (Chen e Kotz, 2000).

Schilit et al (1994) dividem-no em categorias e o definem por exemplos:

- Contexto computacional: conexão de rede, custos de comunicação, largura de banda, recursos próximos, como impressoras e estações de trabalho;
- Contexto de usuário: perfil, localização, pessoas próximas, situação social corrente;
- Contexto físico: iluminação, níveis de ruído, condições de tráfego e temperatura.

Lieberman e Selker (2000) definem o contexto como sendo tudo, exceto as entradas e saídas explícitas de um sistema (Figura 2). Essa definição é apresentada em referência aos programas “caixa-preta”, nos quais uma mesma entrada aplicada gera

sempre a mesma resposta de saída, isto é, a saída é determinada unicamente pela entrada. Segundo essa definição, o contexto é determinado pelo tamanho da caixa-preta, consistindo em tudo aquilo que não é englobado pela mesma.

Figura 2 – Contexto é tudo exceto a entrada e saída explícita de um sistema, segundo Lieberman e Selker (2000).

Chen e Kotz (2000), por sua vez, definem o contexto como sendo o conjunto de estados ambientais e configurações que determinam o comportamento de um programa e/ou como sendo as informações geradas por um programa que são interessantes para o usuário.

Já Dey (2000) define contexto como sendo “qualquer informação relevante que caracterize a situação de uma entidade, na qual uma entidade é uma pessoa, local ou objeto que seja considerado relevante para a interação entre um usuário e um programa, inclusive o próprio programa e o usuário”.

O que estas definições tentam fazer é limitar o escopo da palavra contexto, para que ela seja utilizada e compreendida univocamente por todos os pesquisadores. Porém, qual a razão para utilização do contexto? Segundo Dey (2000), a razão é que, até agora, os computadores não podem ir além do que foram programados para fazer e possuem limites para entender nossa linguagem e nossas atividades. Atualmente, somos obrigados a entrar dados no computador de maneira bastante explícita para que ele possa lidar com a informação e consiga entender o que deve ser feito. Uma solução para melhorar a interação homem-computador é, então, segundo Dey, aumentar a riqueza da entrada de dados trocada com o computador, através do uso do contexto, isto é, fornecer uma maior qualidade de informações para o computador, para que ele seja capaz de

captar informações implícitas, de maneira semelhante à que fazemos na comunicação entre seres humanos. Porém, essa argumentação é contestada por Winograd (2001), o qual diz que a interação atual com o computador já está envolta em contexto. O contexto existente pode ser observado, por exemplo, pelo efeito de se pressionar a tecla F5 em um programa. Dependendo do programa utilizado e da janela em foco, obtém-se uma ação diferente. Uma vez que o resultado dessa ação pode variar (por exemplo, salvar um arquivo ou então abrir uma janela de pesquisa), percebe-se que já existe um contexto associado à interação com o computador. Winograd afirma, ainda, que essa dependência do contexto não é uma propriedade especial de interfaces nem algo novo em computação. Ela é uma consequência do fato de que foi criada uma situação de comunicação (entre o usuário e o programa) e, nesta situação, a interpretação da intenção depende de um contexto mutuamente disponível (por exemplo, o usuário deve saber reconhecer qual janela está selecionada para saber o resultado de sua ação). Porém, Winograd não deixa claro com o exemplo o fato de que, apesar de existir um contexto entre o usuário e o computador nos programas tradicionais, o mesmo é **implícito**, isto é, está embutido na aplicação e no conhecimento do usuário, não sendo facilmente manipulável. O desenvolvimento e uso de um contexto explícito podem beneficiar a interação do usuário e do programa.

Sendo assim, achamos mais conveniente definir o contexto de forma análoga a Dey, porém, aplicando a definição ao domínio da Medicina.

Contexto é qualquer informação que seja relevante para caracterizar as atividades de um profissional de saúde. As atividades do profissional podem ser desde a simples visualização de uma imagem até a realização de pesquisas complexas, passando pelo processamento de imagens e produção de laudos.

Desta maneira, estabelecemos um entendimento comum para a utilização do termo.

Uma vez definido o conceito de contexto, restam dois problemas: como representá-lo e como utilizá-lo de forma significativa, em benefício do usuário.

A representação do contexto deve levar em consideração as seguintes preocupações:

- Uma vez que o contexto não é estático e sim dinâmico, deve ser possível disponibilizar novos tipos de dados contextuais sem muito impacto no modelo utilizado;
- Se o contexto representado for acessível por diversos clientes, então deve ser possível para um cliente acessar novos contextos. Por exemplo, descobrir dinamicamente que existe uma nova fonte de contexto e passar a utilizá-la;
- Agregar dados contextuais (Mostéfaoui, 2003). Podemos desejar unir duas fontes de dados contextuais para extrair uma nova informação. Por exemplo, determinar a atividade de uma pessoa a partir de sua localização e horário;
- Permitir trabalhar no nível de abstração desejado. O contexto fornecido pode ser de muito baixo nível, por exemplo, coordenadas x e y, quando somente necessitamos saber se a pessoa está no prédio ou não;
- Lidar com mudanças lentas e rápidas do contexto. Existem contextos que não variam e outros que mudam frequentemente. Por exemplo, a memória livre do computador está continuamente flutuando, enquanto que a resolução do monitor raramente é alterada;
- Garantir segurança ao acesso do contexto. Informações pessoais do usuário devem ser mantidas sigilosas para pessoas ou dispositivos não autorizados. Isto se aplica se o contexto for acessível remotamente;
- Poder definir contextos complexos. Informações como o nível de atenção de uma pessoa, podem ser relevantes para um programa (por exemplo, evitar preenchimento errôneo de dados) e devem ser passíveis de serem representadas;
- Fazer pesquisas complexas sobre o contexto. O contexto deve poder ser pesquisado e, mais do que isso, fornecer informações complexas como, por exemplo, dizer qual o horário do dia que oferece a maior largura de banda para um determinado ponto de rede;
- Definir a confiabilidade de uma informação (Dey et al, 2002). Se existirem várias fontes de contexto, é preciso determinar a qualidade da informação fornecida por cada fonte. Por exemplo, a informação de que o usuário está realizando a tarefa de produção de um laudo, pode ser inacurada se ela foi inferida a partir do histórico de ações do usuário. Então é preciso um nível de confiabilidade.

A seguir são apresentadas diferentes maneiras de representar e utilizar o contexto.

2.1.2 Representação e uso do contexto

A computação ciente de contexto (*context-aware computing*) é definida, segundo Chen e Kotz (2000), como um paradigma de computação móvel no qual os programas podem descobrir e tirar vantagem de informação contextual (como localização do usuário, horário, pessoas e dispositivos próximos, e a atividade do usuário).

Nesta área, o problema de representação do contexto é particularmente difícil (Goularte, 2003), pois é necessário lidar com mudanças rápidas de contexto (por exemplo, quando uma pessoa, carregando um dispositivo móvel, transita entre dois ambientes), lidar com as limitações de hardware dos dispositivos (memória e tamanho do *display*) além de mudanças na largura de banda e conexão de rede (por exemplo, o programa em um *Palm* deve continuar funcionando quando o usuário passa para uma região sem cobertura de rede).

O Stick-e Document (Brown, 1996), um dos primeiros programas de computação ciente do contexto, representa o contexto de forma implícita através de um programa chamado SETRIGGER, o qual percebe as mudanças do contexto e as utiliza através de condições simples representadas por um documento em SGML (*Standard Generalized Markup Language*, um superset do XML), como mostrado na Figura 3. O objetivo do Stick-e Document era associar às páginas WEB uma nota, similar a um *post-it*, para ser exibida quando uma determinada condição fosse atingida.

```
<note some attributes>
<required>
<with> J.D. Bovey <or> X. Chen
<at> co-ordinates of location
<content>
This is the content.
</note>
```

Figura 3 – Uma Stick-e note. Extraído de (Brown, 1996).

Posteriormente, desenvolveram-se programas mais complexos para explorar o contexto e diferentes arquiteturas para acessar e utilizar dados contextuais.

Judd e Steenkiste (2003) representam o contexto através de uma interface de acesso comum para a informação, similar a um comando SQL, na qual são passados os dados desejados (SELECT), os nomes das fontes de dados (FROM), condições de pesquisa (WHERE), parâmetros extras e tempo de resposta (Figura 4).

Figura 4 – Arquitetura do Serviço de Informação Contextual. Extraído de (Judd e Steenkiste, 2003).

Essa abstração de interface permite que a maneira como os dados contextuais são armazenados fique oculta. A vantagem desta solução é que o acesso é feito de maneira similar a um banco de dados, minimiza a curva de aprendizado para os clientes do serviço. Também é interessante, pois permite fazer pesquisas complexas sobre os dados do contexto (com o equivalente aos JOINS do banco de dados). Porém, apresenta os problemas de tipagem de dados (Hong 2002) e não tem suporte a exploração dos serviços oferecidos (*Discovery Service*), como o feito por (Bisdikian, 2001), (Dey et al 2001) e Bardram e Christensen (2001).

Hong (2002) propõe construir uma infra-estrutura de contexto, chamada de *Context Fabric*, que oferece três serviços: um centro de armazenamento de dados contextuais (*context data store*) para facilitar a modelagem, armazenamento e distribuição do contexto; uma linguagem declarativa de especificação de contexto; e mecanismos de proteção de dados do usuário. A representação do contexto é feita através de quatro conceitos: entidades, atributos, relacionamentos e agregações. O acesso ao contexto seria feito por uma linguagem específica chamada CSL (*Context Specification Language*).

Capra et al (2001) apresentam um *middleware* para contexto. O *middleware* é responsável por oferecer serviços de *binding* (conectar-se a um serviço remoto) e *trading* (localizar um serviço remoto) e o contexto é representado como fontes locais ou

fontes externas e controladas pelo *middleware*. O contexto é utilizado através de condições codificadas em XML, de maneira similar ao Stick-e Document. Um exemplo é mostrado na Figura 5, no qual um comportamento é disparado quando uma condição específica é atingida.

```
<SERVICE name="accessData">
  <BEHAVIOUR policy="copy">
    <RESOURCE name="memory">
      <STATUS operator="greaterEqual" value=x/>
    </RESOURCE>
  </BEHAVIOUR>
  <BEHAVIOUR policy="link">
    <RESOURCE name="bandwidth">
      <STATUS operator="greaterEqual" value=y/>
    </RESOURCE>
    <RESOURCE name="memory">
      <STATUS operator="less" value=x/>
    </RESOURCE>
  </BEHAVIOUR>
</SERVICE>
```

Figura 5 – Solicitação de serviço ao *middleware*. Extraído de Capra (2001).

Perich (2002) apresenta o software *Me-Centric*, no qual o contexto do usuário é modelado em termos de entidades e domínios. Cada entidade participa de um ou mais domínios (chamado de associação ou *membership*). Os domínios são divididos em quatro categorias: domínio físico, do conhecimento, conceitual e colaborativo. A representação dos relacionamentos entre domínios e entidades é feita através de uma ontologia especificada em RDF (*Resource Description Framework*), a qual armazena também as ações a serem tomadas quando uma condição é satisfeita (Figura 6). A ontologia determina os relacionamentos e as instâncias determinam o comportamento.

Figura 6 – Representação do contexto e ação correspondente. Os nós à esquerda representam uma condição a ser satisfeita e o nó à direita a ação correspondente quando a condição for verdadeira. Extraído de Perich (2002).

Dey et al (2001), com o *Context Toolkit*, representa o contexto como um *Widget* ou como um *Aggregator*, os quais são interfaces que ocultam a representação do contexto (no caso, de sensores). O *Aggregator* une diferentes *widgets* para gerar um novo tipo de saída. Define também um *Interpreter* que transforma uma representação de dados em outra (por exemplo, transformar um sinal de GPS em uma posição em um mapa). O cliente se conecta com os *widgets* utilizando *sockets*. A arquitetura também oferece um *Discoverer*, para permitir que os programas encontrem os recursos pela rede. Não é definida uma forma explícita de como usar a informação (como as regras de Perich). Os problemas dessa abordagem são (segundo Winograd, 2001), a necessidade de configuração de cada aplicação para acessar os *widgets*, e a ineficiência de uso da rede, uma vez que cada *widget* requer uma conexão de rede diferenciada.

Barnden e Lee (1999) implementaram o ATT-Meta, o qual é uma ferramenta para processamento de linguagem natural, mas que representa uma outra possibilidade de representação do contexto. O ATT-Meta é um sistema de *rule-based reasoning* (que pode ser traduzido como ‘raciocínio baseado em regras’), isso significa que é composto de um conjunto de regras relacionadas entre si por predicados comuns. As regras são

disparadas quando os predicados são verdadeiros e podem, por sua vez, disparar outras regras se mudarem o estado do sistema. No caso, o ATT-Meta incorpora em suas regras um nível de certeza qualitativa (*qualitative certainty level*). As regras são escritas da forma mostrada na Figura 7.

```
IF <antecedent component 1> AND <antecedent component 2> .... AND <antecedent  
component N>  
THEN [<certainty level>] <consequent>
```

Figura 7 – Regras do ATT-Meta com nível de incerteza. Extraído de Barnden e Lee (1999).

Os valores possíveis de incerteza são: *suggested*, *presumed*, *certain* (sugerido, presumido e correto). As regras são disparadas utilizando-se *backward-chaining* (isto é, a partir de uma hipótese claramente determinada - por exemplo, saber se John gosta de chá - disparam-se outras regras tentando encontrar informações que suportem essa hipótese – por exemplo, que todos os ingleses gostam de chá). Com a medida da incerteza é possível saber a confiabilidade de uma hipótese, o que é relacionado com duas das questões de representação do contexto: a da confiabilidade da informação e da agregação de informações do contexto. Assim, uma possível representação do contexto é através de regras que descrevem as relações entre os elementos associadas com seus níveis de incerteza. O problema com esta abordagem é a dificuldade em estabelecer as regras a serem utilizadas.

Encarnação e Stoev (1999), com o sistema de ajuda ORIMUHS, representam o contexto através de grafos (Figura 8). Para isso, o sistema utiliza um conjunto de ações de referência, extraídas de um usuário com nível de expertise conhecido, o qual é comparado com as ações do usuário corrente, permitindo a previsão das ações do usuário. A vantagem desse sistema é o fato de poder ser conectado a qualquer tipo de interface, pois a ação é genérica. Porém, a desvantagem é que a previsão de ações é limitada às ações selecionadas pelo usuário experiente, não sendo possível fazer inferências mais complexas sobre as ações tomadas, pois falta um modelo explícito do usuário.

Figura 8 – Esquema do sistema ORIMUHS. Extraído de Encarnação e Stoev (1999).

Adomavicius et al (2004) representam o contexto de forma multidimensional, a qual supõe que cada característica do contexto pertence a uma dimensão, que é posta no eixo de um cubo multidimensional, sendo possível encontrar situações contextualmente parecidas por distância entre pontos no espaço N-dimensional. Essa abordagem utiliza conceitos de bases OLAP (*OnLine Analytical Processing*).

Outra possibilidade para representar o contexto é através do uso de exemplos. A partir de exemplos, é possível derivar regras, como na técnica de classificação de Hong (1997). O uso de exemplos é útil para sistemas que aprendem a partir das ações do usuário, como em agentes inteligentes (Fleming e Cohen, 1999) (Wooldrige e Jennings, 1995). Outras técnicas incluem *Memory Based Reasoning* (MBR) (Sheppard e Salzberg, 1995), (Stanfill e Waltz, 1986), (Kasif et al 1998) e *Case Based Reasoning* (CBR) (Russel, 1995).

Maes (1994) apresenta a idéia de um assistente pessoal que aprende a partir das ações do usuário, guardando exemplos de sua atuação (usando MBR) e apresenta dois exemplos de implementação: um assistente de e-mails e um assistente de reuniões.

Kozierok e Maes (1993) utilizam também um agente de interface, porém com outra técnica de Inteligência Artificial (AI), chamada de *reinforcement learning* para melhorar os resultados do agente. Já Dent et al (1992) desenvolvem um assistente

pessoal que, a partir da observação das ações do usuário, infere um conjunto de regras as quais são generalizadas pela remoção de pré-condições que não aumentem o número de falsos positivos do algoritmo de treinamento. Laskari et al (1994) utiliza agentes de interface colaborativos, que consultam agentes de outros usuários para melhorar sua taxa de aprendizado.

Horvitz (1999), por sua vez, apresenta o sistema LookOut para agendamento e gerenciamento de reuniões a partir de e-mails. O LookOut processa o cabeçalho, assunto e corpo da mensagem e, baseado nesta informação, assinala uma possibilidade de o usuário estar interessado em ver um calendário e marcar uma reunião. A possibilidade é calculada através de um sistema de classificação probabilística que é treinado a partir da observação do usuário lidando com o e-mail.

O CCOW (*Clinical Context Object Workgroup*) é parte do HL7 (*Health Level 7*, padrão para transmissão de dados gerenciais e de laboratório) e tem como missão “...definir padrões que permitam a integração visual de programas da área de saúde. Programas são visualmente integrados quando trabalham conjuntamente, de uma maneira percebida pelo usuário, de forma a melhorar a habilidade do usuário para incorporar tecnologia da informação como parte do processo de atendimento ao paciente” (CCOW, 2003).

De uma maneira simples, o objetivo do CCOW é a integração de informações do paciente entre programas de vendedores diferentes (CCOW, 2004). Por exemplo, quando o usuário seleciona um paciente no programa A, automaticamente, o programa B mostra as informações do mesmo paciente. Até o momento, o CCOW foca em dois tipos de dados contextuais: o contexto de paciente, que permite ao usuário, a partir de qualquer programa, selecionar um paciente de interesse, sincronizando automaticamente os programas restantes para o mesmo paciente (CCOW, 2003); e o contexto de usuário, que permite ao usuário ‘logar’ de forma segura uma única vez e obter acesso a todos os programas sem necessidade de repetidos *logins*, sincronizando todos os programas com o usuário.

A arquitetura do CCOW utiliza uma variação do *pattern Observer* (Gamma, 2000), utilizando um Gerenciador de Contexto, o qual contém todos os dados comuns de contexto e os vários Participantes do Contexto, que são aqueles interessados em mudanças no contexto. O Gerenciador de Contexto possui algumas facilidades (ou

serviços) como o mapeamento de identificadores de paciente e acesso a dados demográficos do mesmo (Figura 9).

Figura 9 –Componentes da Arquitetura de Gerenciamento de Contexto do HL7. Extraído de CCOW (2003).

Uma das deficiências do CCOW é possuir uma especificação muito complexa para o fim proposto. Por outro lado, por ser parte do HL7, é possível que ela possa ser adotada amplamente, no futuro, sendo necessário acompanhar a sua evolução.

Chen et al (2004) defendem a representação do contexto na forma de ontologias, dizendo que elas são um requisito chave para sistemas cientes de contexto, pois: (i) uma ontologia comum permite o compartilhamento de conhecimento em um sistema distribuído, dinâmico e aberto; (ii) ontologias com semânticas declarativas bem definidas fornecem maneiras de agentes inteligentes ponderarem sobre a informação contextual e (iii) ontologias representadas explicitamente permitem que dispositivos e agentes não designados explicitamente para trabalhar conjuntamente consigam fazê-lo. Neste sentido, propõem o sistema CoBrA (*Context Broker Architecture* - Figura 10), o qual assume as seguintes responsabilidades: adquirir informação contextual a partir de fontes heterogêneas e manter a consistência do contexto como um todo através de *reasoning*; ajudar agentes distribuídos a compartilharem conhecimento sobre o contexto através do uso de ontologias e linguagens de comunicação entre agentes; e proteger a privacidade dos usuários pelo estabelecimento de políticas bem definidas, quando os dados pessoais estiverem sendo compartilhados entre os agentes da comunidade.

Figura 10 – Diagrama geral da arquitetura do CoBrA (Extraído de Chen et al, 2004).

Na Tabela 1, a seguir, são mostradas, resumidamente, as diferentes abordagens para representação do contexto com seus pontos positivos e negativos.

Tabela 1 – Resumo das diferentes abordagens para representação e utilização de contexto.

<i>Trabalho</i>	<i>Modelo do contexto</i>	<i>Uso do contexto</i>	<i>Prós</i>	<i>Contras</i>
Stick-e Document (Brown, 1996)	SETRIGGER .	Regras SGML.	Suporta codificação de regras flexíveis.	Muito simples.
Judd e Steenkiste (2003)	O cliente deve conhecer o contexto a ser acessado.	Interface similar ao SQL.	Permite pesquisas complexas.	Não lida com dados complexos; não oferece <i>Discovery Service</i> .
<i>Context Fabric</i> (Hong e Landay, 2001)	Não é definido claramente.	Usando linguagem de pesquisa (CSL).	Simplificação do acesso ao contexto, por um único servidor.	Falta detalhamento da infra-estrutura proposta.
<i>Reflective Middleware</i> (Capra et al, 2001)	Fontes locais e fontes externas. Modelo representado por meta-dados.	Regras codificadas em XML.	Simplifica o acesso ao contexto. Oferece serviços auxiliares de <i>trading</i> e <i>binding</i> .	Implementação complexa do serviço.
<i>Me-Centric</i> (Perich, 2002)	Ontologia (RDF).	Regras armazenadas também na ontologia.	Permite representação de modelos mais complexos.	Não considera segurança dos dados do usuário.
Context Toolkit (Dey, 2001)	Representados por <i>widgets</i> .	Uso de <i>Discovery Service</i> .	Abstrai o acesso a fontes de informação contextual.	Mau uso de largura de rede. Configuração complexa.
ATT-Meta (Barnden e Lee, 1999)	Base de conhecimento (<i>knowledge based system</i>).	<i>Backward chaining</i> .	Trata da confiabilidade da informação e inferência de dados.	Manutenção das regras. Foi desenvolvido para aplicação em linguagem natural.
ORIMUHS (Encarnação e Stoev, 1999)	Modelo do usuário empírico.	Estatístico e Probabilístico.	Pode prever ações do usuário.	Falta um modelo explícito do usuário para inferências mais abstratas.

Adomavicius et al (2004)	Eixos de um cubo multidimensional.	Cálculo de distância entre pontos no espaço multidimensional.	Utiliza métodos OLAP (<i>On-Line Analytical Processing</i>).	Dificuldade em estender o modelo (acrescentar mais itens para o contexto). Foi proposto para sistemas de recomendação (<i>Recommender Systems</i>).
Maes (1994)	Exemplos do usuário.	Probabilidade (<i>Memory-Based Reasoning</i>).	É adaptado a cada usuário.	Demora a aprender os hábitos do usuário.
Kozierok e Mães(1993)	Exemplos do usuário.	Probabilidade e <i>Reinforcement Learning</i> .	É adaptado a cada usuário.	Não permite inferência sobre as ações do usuário, pois não possui um modelo explícito do mesmo.
Dent et al (1992)	Exemplos do usuário.	Generaliza regras a partir dos exemplos.	Adaptado ao usuário.	Não permite inferência sobre as ações do usuário.
Horvitz (1999)	Exemplos do usuário.	Probabilístico.	Adaptado ao usuário.	Não permite inferência sobre as ações do usuário.
CCOW (2003)	Modelo de identificação do paciente, modelo de autenticação.	Participante do contexto – conhece o modelo do contexto.	Pode se tornar padrão do HL7.	Especificação complexa. Contexto limitado.
CoBrA (Chen, 2004)	Ontologia.	Linguagem de comunicação de agentes.	Conhecimento compartilhado entre participantes. Proteção dos dados do usuário. Permite <i>reasoning</i> sobre o contexto.	Segundo Henricksen et al (2004), é limitado por possuir reasoning baseado puramente em OWL sem suporte adicional a regras.

2.1.3 Ontologias

Uma ontologia é uma especificação explícita de uma conceituação. Uma conceituação é uma visão abstrata e simplificada do domínio que se deseja representar para algum propósito (Gruber, 1993; Crapo et al, 2003).

Uma ontologia pode ser considerada, em AI, como uma representação de um domínio a partir de seus conceitos abstratos e a forma como esses conceitos se relacionam entre si. É um modelo consensual do domínio de discurso, no sentido de que é reconhecido da mesma forma por aqueles que o utilizam e, por ser um modelo, não faz referência às suas instâncias. Uma ontologia associada a um conjunto de instâncias individuais constitui uma base de conhecimento (*knowledge base*).

A representação do domínio é feita através de conceitos (também chamados de classes). As propriedades dos conceitos (*slots* ou *roles*) descrevem os atributos e características dos conceitos. As propriedades podem possuir restrições (*facets*) em seus possíveis valores.

As classes são o foco principal da maioria das ontologias. Por exemplo, uma classe de vinhos representa todos os vinhos¹. Vinhos específicos são instâncias desta classe. Uma classe pode ter subclasses que representam conceitos que são mais específicos que a superclasse. Por exemplo, é possível dividir a classe de todos os vinhos nas subclasses de vinho tinto, rose e branco.

As propriedades descrevem características das classes. Por exemplo, um vinho tem associado os atributos de região de origem, sabor e quantidade de açúcar.

Em termos práticos, o desenvolvimento de uma ontologia inclui:

- Definição das classes em uma ontologia;
- Ordenamento das classes em uma hierarquia taxonômica (classe-superclasse);
- Definição de propriedades e suas restrições.

Um exemplo de ontologia pode ser visto na Figura 11, na qual é mostrada a relação de classes (Chardonnay é um tipo de vinho tinto, o qual por sua vez é um tipo de vinho) e instâncias (Bancroft é um produtor de vinhos) e atributos (o vinho Bancroft chardonnay é produzido por Bancroft).

¹ http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

Figura 11 – Exemplo de uma descrição de uma ontologia e instâncias.

As ontologias permitem o desenvolvimento de programas que os tornam mais capazes de lidar com informação complexa e heterogênea. Elas são úteis quando as distinções semânticas utilizadas normalmente por humanos são cruciais para o propósito do programa.

As ontologias não são construídas da mesma forma. Existe um grande número de linguagens possíveis para sua descrição. Alguns exemplos são *Common Logic (CL)*², *DAM+OIL*³ e *OWL*⁴.

Neste trabalho, a implementação do modelo de contexto, apresentado no Capítulo 3, foi implementada na forma de uma ontologia, com o uso de software Protégé, descrito a seguir.

2.1.3.1 Protégé 2000

Protégé-2000 é um editor de ontologias, com código-fonte aberto, desenvolvido pelo grupo de informática médica de Stanford⁵, o qual permita um usuário: (i) construir uma ontologia de um domínio; (ii) customizar formulários de entrada de

² <http://cl.tamu.edu/>

³ <http://www.daml.org/>

⁴ <http://www.w3.org/2001/sw/WebOnt/>

⁵ <http://www.smi.stanford.edu/>

dados; (iii) entrar dados. Pode ser usado como uma biblioteca em outros programas para acessar e exibir bases de conhecimento⁶.

Quando um projeto é iniciado, é definida uma superclasse denominada **THING** (Figura 12), a partir da qual as entidades da ontologia sendo definida são derivadas (Figura 13).

Figura 12 – Ontologia com elementos básicos.

Figura 13 – Definição de uma classe no Protégé.

Uma vez definido um conjunto de entidades ou classes que representam o domínio (Figura 14) é possível definir atributos para cada um deles (Figura 15). Os atributos são definidos de acordo com o seu tipo (inteiro - int, flutuante - float, texto - string, booleano - boolean e outra classe - object).

⁶ <http://protege.stanford.edu/whatis.html>

Com o estabelecimento das propriedades da entidade é possível criar instâncias da mesma (Figura 16), com o preenchimento dos valores das propriedades.

Figura 14 – Conjunto de classes definidas em um projeto.

Figura 15 – Definição de um *slot*, ou propriedade de uma classe.

Figura 16 – Definição de uma instância de uma classe.

O Protégé permite o rápido desenvolvimento de uma ontologia e preenchimento de dados de instâncias.

2.2 Representação de informação

A forma de representação da informação digital é importante, pois determina o que é possível realizar, em termos computacionais, com a informação. Por exemplo, um laudo laboratorial que exibe o resultado de uma medida de glicose/soro, como mostrado na Figura 17, que seja codificado na forma de texto, possui todas as informações necessárias para o médico. Porém, as mesmas informações não são facilmente acessíveis computacionalmente, pois não existe uma marcação clara que determine quais os elementos relevantes do texto. Ou seja, o conhecimento que deve ser embutido no computador, na forma de um programa, para extrair a informação relevante do texto é de difícil elaboração, uma vez que é preciso considerar variações de posição no texto, o uso de variantes para uma mesma palavra e decidir o que fazer na falta de uma informação esperada (por exemplo, um valor sem unidade).

GLICOSE / SORO				
Valores de Referencia				
		Valores (mg/dl)	Categorias	
		-----	-----	
Resultado:	101	mg/dl	70 a 99	Normal
			100 a 125	Inapropriada
			>=126	Elevada
Valores de referencia recomendados pela Associacao Americana de Diabetes-ADA				
Metodo: Enzimatico, automatizado				

Figura 17 – Laudo de laboratório codificado como texto.

Por outro lado, se o laudo de glicose/soro for armazenado com representação em XML, como mostrado na Figura 18, o programa necessário para extrair as informações relevantes torna-se bastante simples, uma vez que todos os itens estão explicitamente marcados. Então, o laudo em XML é contextualmente mais relevante, pois permite extração da informação e seu uso de forma computacionalmente simples.

O laudo codificado também tem a vantagem de permitir o desenvolvimento de sistemas que realizam buscas complexas sobre os dados, por exemplo, buscar todos os pacientes cuja medida de glicose/soro foi acima de 100 mg/dl.

```
<exame tipo="GLICOSE / SORO">
  <resultado>
    <medida-numerica>
      <valor type="inteiro">101</valor>
      <unidade>mg/dl</unidade>
    </medida-numerica>
    <valor-referencia type="normal" dic="ADA">
      <medida-range>
        <valor-min type="inteiro">70</valor-min>
        <valor-max type="inteiro">99</valor-max>
        <unidade>mg/dl</unidade>
      </medida-range>
    </valor-referencia>
    <valor-referencia type="inapropriado" dic="ADA">
```

```
<medida-range>
  <valor-min type="inteiro">100</valor-min>
  <valor-max type="inteiro">125</valor-max>
  <unidade>mg/dl</unidade>
</medida-range>
</valor-referencia>
<valor-referencia type="elevada" dic="ADA">
  <medida-range>
    <valor-min type="inteiro">126</valor-min>
    <valor-max type="inteiro">*</valor-max>
    <unidade>mg/dl</unidade>
  </medida-range>
</valor-referencia>
<metodo>
  <tipo automatizado="sim">Enzimatico</tipo>
</metodo>
</resultado>
```

Figura 18 – Resultado laboratorial codificado utilizando XML.

Mas, ainda que codificado na forma de XML, o laudo da Figura 18 apresenta o problema da interoperabilidade de informações. O significado da marcação “medida-numérica” pode ser conhecido em um programa na Instituição A, mas sem significado para outro programa na Instituição B. Assim torna-se necessário o uso de padrões.

Os padrões determinam um conhecimento comum entre diferentes sistemas de informação, retirando a ambigüidade de uma informação, ou seja, contextualizando-a. No padrão DICOM, a modalidade de uma série de imagens é determinada pelo campo (0008,0060) e tem um número finito de valores possíveis (mostrados na Figura 19) que são claramente definidos.

C.7.3.1.1 General Series Attribute Descriptions	
C.7.3.1.1.1 Modality	
Defined Terms for the Modality (0008,0060) are:	
CR = Computed Radiography	CT = Computed Tomography
MR = Magnetic Resonance	NM = Nuclear Medicine
US = Ultrasound	OT = Other
AS = Angioscopy	BI = Biomagnetic imaging
CD = Color flow Doppler	CP = Culposcopy
CS = Cystoscopy	DD = Duplex Doppler
DG = Diaphanography	DM = Digital microscopy
EC = Echocardiography	ES = Endoscopy
FA = Fluorescein angiography	FS = Fundoscopy
LP = Laparoscopy	LS = Laser surface scan
MA = Magnetic resonance angiography	MS = Magnetic resonance spectroscopy
PT = Positron emission tomography (PET)	RG = Radiographic imaging (conventional film/screen)
ST = Single-photon emission computed tomography (SPECT)	TG = Thermography
XA = X-Ray Angiography	RF = Radio Fluoroscopy
RTIMAGE = Radiotherapy Image	RTDOSE = Radiotherapy Dose
RTSTRUCT = Radiotherapy Structure Set	RTPLAN = Radiotherapy Plan
HC = Hard Copy	<u>PR = Presentation State</u>

Figura 19 – Lista de modalidades segundo a especificação DICOM.

Um dos problemas para a interoperabilidade, em especial do HL7, é que os fabricantes não seguem exatamente o padrão, implementando partes da especificação de formas ligeiramente diferentes (por exemplo, representar uma data no HL7 como 19991231^100000.000 ao invés de 19991231100000.000, dividindo data e hora em dois campos⁷).

Um outro problema são omissões do padrão em campos contextualmente significativos, como *Study Description* (0008,1030), do DICOM, que é um campo texto que suporta qualquer valor e descreve o tipo de estudo sendo realizado⁸ (por exemplo, Ecocardiograma com Doppler ou Exame Renal Dinâmico).

⁷ Exemplo extraído de <http://www.interfaceware.com/manual/ch-2-1-12.html>

⁸ Evidentemente seria difícil atingir um consenso para limitar os nomes de todos os estudos possíveis, mas é um exemplo ilustrativo da dificuldade computacional em se utilizar informações, mesmo que padronizadas.

2.3 Sistemas de Informação em Hospitais

Um Sistema de Informação Hospitalar (*Hospital Information System* - HIS) contém um grande conjunto de informações digitais, as quais incluem: (i) dados financeiros e gerenciais, como o custo de um cateter, o convênio do paciente, a folha de pagamento dos funcionários, controle de estoque, etc; e (ii) informações do paciente, como laudos, data de admissão, alta, anamnese, etc (Stasui, 1999). Devido a razões históricas e ao tipo de tecnologia empregada, as imagens médicas são consideradas como um sistema à parte, e são organizadas em um sistema de transmissão e arquivamento de imagens médicas chamado PACS (*Picture, Archiving and Communication Systems*) que visa o desenvolvimento de um hospital sem filmes (Bick e Lenzen, 1999; Binkhuysen, 1992; Huang, 2000).

O PACS e o HIS têm intersecção com o Prontuário Eletrônico do Paciente (PEP ou EHR, em inglês). O PEP tem diversas definições, nas quais se varia o espectro de atribuições do mesmo. Segundo Massad et al (2001), o prontuário eletrônico “é um meio físico, um repositório onde todas as informações de saúde, clínicas e administrativas, ao longo da vida de um indivíduo estão armazenadas”. Segundo o IOM (1997), é “um registro eletrônico que reside em um sistema especificamente projetado para apoiar os usuários fornecendo acesso a um completo conjunto de dados corretos, alertas, sistemas de apoio à decisão e outros recursos, como *links* para bases de conhecimento médico”. Essas definições são genéricas e não constituem uma definição exata de quais informações devem ou não estar presentes no PEP, e também não determina como as informações devem ser armazenadas e disponibilizadas. Porém, todas as definições vislumbram a possibilidade de utilizar os recursos computacionais como uma maneira de unir e disponibilizar todas as informações do paciente através de uma interface comum. As vantagens esperadas com o desenvolvimento do PEP são listadas por Massad et al (2001) como:

- *Acesso remoto e simultâneo.* Vários profissionais podem acessar um mesmo prontuário simultaneamente e remotamente;
- *Legibilidade.* Registros à mão são na maioria das vezes difíceis de ler;
- *Segurança e confidencialidade de dados.* Garante a persistência dos dados do paciente independentemente de falhas de hardware e desastres e acesso somente por pessoas autorizadas;

- *Flexibilidade de layout.* O usuário pode usufruir formas diferentes de apresentação dos dados;
- *Integração entre sistemas de informação diferenciados.* Uma vez em formato eletrônico, os dados do paciente podem ser integrados a outros sistemas de informação;
- *Captura automática de dados.* evita erros de transcrição;
- *Processamento contínuo de dados.* os programas podem checar continuamente consistência e erros de dados, emitindo alertas e aviso aos profissionais;
- *Assistência à pesquisa.* os dados podem facilitar os estudos epidemiológicos;
- *Relatórios.* dados podem ser reunidos de diversas fontes de acordo com o objetivo da apresentação;
- *Dados atualizados.* um dado que entra no sistema em um ponto, automaticamente atualiza e compartilha a informação nos outros pontos do sistema.

Apesar de não ter sido encontrada nenhuma implementação de PEP que tenha todas as características citadas anteriormente, deve-se notar que o desenvolvimento de um PEP é um processo evolutivo, no qual as informações são gradativamente convertidas em formato eletrônico e integradas em níveis mais complexos.

Podem ser definidos vários objetivos desejáveis durante o desenvolvimento do prontuário eletrônico, entre eles podemos citar (extraído de Massad et al, 2001):

- Oferecer uma lista de problemas que indique os problemas atuais e progressos do paciente: uma lista de problemas deve denotar o número de ocorrências associadas com o passado e o problema corrente, assim como o estado (ativo, inativo, resolvido, indeterminado, etc.) atual de cada problema;
- Poder documentar o raciocínio clínico em diagnósticos, conclusões e na seleção de intervenções terapêuticas: permitir compartilhar o raciocínio clínico com outros profissionais, desenvolver meios automáticos para acompanhar os caminhos no processo de tomada de decisão;
- Ser um registro longitudinal abrangendo toda a vida do paciente, ligando todos os dados de consultas e atendimentos anteriores;

- Permitir visualização simultânea e customizada dos dados do paciente pelos profissionais, departamentos e empresas. Esta capacidade melhora a eficiência do trabalho técnico de usuários específicos permitindo que o dado seja apresentado no formato que é mais usado por estes usuários. A flexibilidade em permitir diferentes e simultâneas visualizações dos dados é uma característica que a maioria dos fabricantes tem dificuldade em conseguir atender;
- Apoiar o acesso em linha a recursos de informação locais e remotos: bases de dados em texto, correio eletrônico, CDROM. O acesso a fontes externas deve garantir ao profissional obtenção da informação necessária para apoiar o cuidado ao cliente;
- Facilitar a solução de problemas clínicos fornecendo instrumentos de análise e de decisão. Exemplos destes instrumentos são os alertas e os sistemas de apoio à decisão clínica e administrativa;
- Apoiar a entrada de dados diretamente pelo médico: a questão é fornecer mecanismos e interfaces simples e diretas para a entrada de dados;
- Ter flexibilidade para apoiar a incorporação de existentes e futuras necessidades das especialidades clínicas: deve ser flexível para ser expandido.

Para que seja possível desenvolver um PEP mais próximo de sua visão ideal é preciso que haja o desenvolvimento de modelos que representem e estruturem a informação médica para usos complexos e que seja interoperável com outros sistemas (Bidgood et al, 1999). Também se faz necessário o uso de padrões maduros que permitam a troca de informações médica e hospitalar. No caso de imagens médicas, o padrão mais adotado é o DICOM 3.0 (*Digital Imaging and Communications in Medicine*). O padrão DICOM determina formas específicas de armazenamento e transmissão de imagens médicas. O DICOM trata também da codificação de laudos integrados a imagens, com a especificação do DICOM *Structured Reporting* (DICOM SR) (Clunie, 2000), apresentação de imagens (DICOM *Grayscale*) (ACR-NEMA, 2003a) e impressão de imagens (ACR-NEMA, 2003b). Para as informações gerenciais e dados de laboratório, o padrão mais adotado para troca de informações é o HL7 (*Health Level 7*), versão 2.3. O HL7 define mensagens para dados que são trocados entre

aplicações, baseados em um evento particular (por exemplo, admissão de um paciente). O padrão HL7 também habilita um sistema a questionar outro para obter informações relevantes. Por exemplo, um cliente pode questionar o sistema de laboratório para obter resultados de testes para um paciente em particular.

Porém, a representação de informações médicas como laudo, prescrição e anamnese não possui uma estrutura definida. Este tipo de informação pode ser representado como sendo um texto simples ou até como um texto estruturado com dados codificados por dicionários médicos (Cimino, 1998). Os diagnósticos, por exigências legais, são normalmente codificados com o dicionário CID-10 (Classificação Estatística Internacional de Doenças e Problemas Relacionados à Saúde 10ª Revisão).

Uma tentativa de permitir a interoperabilidade entre sistemas de informação de saúde é o CORBAMED. O CORBAMED é um conjunto de interfaces que definem uma forma padronizada de acesso a dados clínicos entre hospitais e Instituições de Saúde. É composto de um conjunto de especificações, sendo as principais, descritas a seguir.

- PIDS (*Person Identification Service*) (OMG, 1998). É um serviço para unificar as representações do paciente entre instituições diferentes. Pelo fato de que uma pessoa pode ser atendida em dezenas de hospitais e instituições de saúde ao longo da vida, surge o problema de não ser possível correlacionar os pacientes com os diferentes registros eletrônicos. Para unificar essas representações desenvolveu-se a especificação do PIDS, a qual tem, como principais funções, a atribuição de identificadores e a correlação dos mesmos entre instituições;
- CIAS (*Clinical Image Access Service*) (OMG, 2001a). O CIAS tem a função de fornecer um mecanismo claro, não-ambíguo, para acesso a imagens médicas e meta-dados relacionados, que possa ser utilizado por clínicos gerais e especializados quando a qualidade diagnóstica não é uma consideração principal. Apesar da ressalva, a interface permite a recuperação de imagens em seu formato original;
- COAS (*Clinical Observation Access Service*) (OMG, 2001b). O COAS tem a função de prover o acesso a observações clínicas, que são definidas como sendo “qualquer medida, gravação, ou descrição de um estado anatômico, fisiológico, patológico, ou psicológico ou o histórico de um ser humano ou qualquer amostra proveniente de um ser humano, e quaisquer impressões, conclusões ou

julgamentos feitos em relação àquele indivíduo dentro do contexto do processo corrente de atendimento a saúde”⁹. Essa definição é similar à definição de contexto, discutida mais adiante, pois ela tenta englobar toda informação relativa ao paciente que possa auxiliar o médico em sua tarefa de atendê-lo, enquanto que no contexto, se tenta englobar essa informação mais o ambiente no qual ela é acessada.

- Serviço de Controle de Acesso (RAD – *Resource Access Decision*) (OMG, 2000). É um mecanismo para obter decisões de autorização e para administração de políticas de decisão de acesso. Ele fornece uma maneira uniforme para um programa requerer e receber uma decisão de autorização.

O problema da especificação do CORBAMED, especialmente para o COAS e CIAS, é falhar em garantir a interoperabilidade semântica, pois somente é definido **como** a informação será passada e não **qual** informação será transmitida. A definição de uma estrutura de uma observação segundo o COAS é mostrada na Figura 20. Nela, uma observação é dividida em dois tipos: observações atômicas (*AtomicObservation*) e observações compostas (*CompositeObservation*), sendo que cada observação atômica possui um valor associado (*ObservationValue*) e a observação composta contém outras observações (*+composed of*). Esta representação é equivalente a uma estrutura em forma de árvore, como a do sistema de arquivos, com pastas (observações compostas) e arquivos (observações atômicas) ou um documento XML. Este tipo de representação, porém, não indica a maneira como a informação é estruturada. Por exemplo, a representação do resultado de laboratório mostrado anteriormente poderia ser codificado das duas formas mostradas na Figura 21, sem que haja quebra do padrão. Assim, é necessário um modelo explícito da informação, de forma que ela possa ser compartilhada entre duas instituições quando se utiliza o COAS e o CIAS (Moreno, 2002a). Uma solução proposta por Moreno (2002b) foi a combinação do CORBAMED com outros padrões, especificamente, com o DICOM SR, realizando uma conversão entre uma representação e outra (Figura 22).

⁹ “any measurement, recording, or description of the anatomical, physiological, pathological, or psychological state or history of a human being or any sample from a human being, and any impressions, conclusions, or judgments made regarding that individual within the context of the current delivery of healthcare.”

Figura 20 – Modelo de uma observação segundo a especificação COAS (Extraído de OMG 2001b).

Figura 21 – Exemplo de diferentes representações da informação utilizando observações do COAS.

Figura 22 – Solução proposta para interoperabilidade semântica através da conversão de DICOM SR para CORBA Med.

2.4 PACS-HC

O Instituto do Coração desenvolveu um projeto para criar um ambiente distribuído para transmissão, arquivamento, recuperação e visualização de imagens médicas e suas informações relacionadas, chamado PACS-HC (Nardon et al, 2000; Furuie et al, 2003).

Esse projeto envolve a integração das diferentes instituições do Hospital das Clínicas da Faculdade de Medicina - Universidade de São Paulo, que formam um ambiente heterogêneo e distribuído.

O PACS-HC é composto basicamente de quatro módulos:

1. *Serviços com acesso padronizado segundo as especificações do CORBA Med¹⁰.*

O propósito da adoção do CORBA Med é permitir a interoperabilidade entre diferentes instituições de saúde.

2. *Criação de um sistema de informações flexível, capaz de realizar buscas complexas sobre as bases de dados.*

¹⁰ Common Object Request Broker Architecture in Medicine

Foi realizada uma modelagem orientada a objeto (Nardon et al, 2000; Furuie et al, 2003) para representar as informações clínicas e imagens médicas.

A Figura 23 mostra as classes básicas da modelagem do prontuário do paciente. A classe *MedicalRecord* representa o conjunto dos documentos que compõe o prontuário do paciente. Cada instância da classe *MedicalRecord* possui uma associação com um e apenas um paciente. Cada instância da classe *MedicalRecord* está também associada a zero ou mais documentos médicos, que são representados pela classe *Document*. A classe *Document* foi criada para representar cada documento que faz parte do prontuário do paciente. Subclasses da classe *Document* representam um documento específico, como um laudo de raios X, um laudo de ECG, etc. Cada documento possui uma lista de outros documentos que auxiliaram a sua criação. A classe *PatientEvents* representa a lista de eventos que ocorreram na vida de um paciente, tais como admissões, exames, cirurgias, etc. Cada instância desta classe possui uma associação com uma e apenas uma instância da classe *Patient* e com uma ou mais instâncias da classe *Event*. Através da classe *PatientEvents* é possível recuperar todo o histórico do paciente. Instâncias da classe *Event* representam eventos que podem ocorrer na vida de um paciente.

Figura 23 – Visão geral do sistema de informações.

Na Figura 24 é apresentado o modelo geral de imagens médicas. A classe *Study* da Figura 24 representa um conceito global de estudo clínico. Um estudo envolvendo imagens constitui uma especialização dessa classe e suas características e atributos específicos são contemplados na classe *ImageStudy* (que no modelo DICOM 3 é a classe *GeneralStudy*).

A classe *SeriesOfImages* (que possui toda a funcionalidade da classe *GeneralSeries* do modelo DICOM 3), por sua vez, é uma especialização da classe *PerformedProcedure*, a qual descreve os diversos procedimentos aos quais um paciente pode se submeter dentro da Instituição de Saúde.

Figura 24 – Modelo geral de imagens médicas.

A classe *ImageDicom* (Figura 24) é uma classe abstrata que contém atributos da classe *ImageData* (não mostrada). Como o principal objetivo do modelo proposto é

permitir a manipulação, visualização e processamento de imagens médicas, optou-se por uma descrição mais semântica do objeto imagem. Do ponto de vista de estruturação de imagens médicas, uma grande divisão pode ser efetuada entre dados de aquisição e dados processados. Esses dois tipos estão descritos nas classes *AcquisitionImage* e *ProcessedData*. A esses dois tipos de dados são associadas permissões de acesso distintas para visualização e processamento. A informação de cada elemento do conjunto de dados está contida na classe *DataND*, que descreve sinais N-dimensionais. Nesta abordagem, *Overlays* e *Curvas* - compostas por elementos descritos pela classe *DataND* - são considerados como dados resultantes de processamento, e constituem especializações da classe *ProcessedData*.

A classe *DataND* descreve as informações intrínsecas dos dados de imagem, ou seja, sua dimensão, a quantidade de elementos de cada dimensão, calibração, unidades e o *array* de dados. Os objetos desta classe serão instanciados de acordo com a sua dimensão. No modelo atual, estão descritas as classes para sinais 1D, 2D, 3D e 4D. Novas subclasses representando outras dimensões podem, entretanto, ser incluídas no modelo. Uma vez definida a dimensão do volume de dados, o próximo passo foi classificá-las funcionalmente. Assim, por exemplo, um volume de dados 3D pode representar um conjunto de imagens 2D dinâmicas, um conjunto de imagens 2D de projeções ou ainda um único volume 3D.

O modelo aqui mostrado serviu de base para a implementação do modelo de contexto, apresentado no Capítulo 3.

3. Desenvolvimento de uma rede capaz de suportar o tráfego exigido para a troca de informações e imagens entre instituições.

A capacidade de troca de informações em uma Instituição de Saúde ou entre diferentes Instituições está limitada pela capacidade disponível, ou largura de banda, dos meios físicos dos quais essas Instituições dependem (par metálico, cabo coaxial e fibra óptica). Em geral, a complexidade e os custos envolvidos para instalação e manutenção dos meios de comunicação são proporcionais à largura de banda disponível. O custo e a capacidade de transporte de informação envolvidos nas diferentes tecnologias de comunicação são importantes, pois afetam a disponibilidade e qualidade da informação necessárias para o diagnóstico de doenças. Se a informação a ser distribuída contiver imagens, que necessitam de alta resolução temporal, espacial e de

contraste, então a aplicação necessariamente requer alta largura de banda e infraestrutura eficiente de comunicação.

O modelo de rede de comunicação adotado no InCor consiste de uma plataforma heterogênea, porém escalonável, que emprega tecnologias *Ethernet* e suas extensões (*Fast* e *Gigabit Ethernet*). Sobre esta plataforma, adotou-se como padrão para protocolo de rede o TCP/IP. Para atender à demanda de alta disponibilidade de informações nas estações clientes, consideramos a utilização do padrão *Fast Ethernet* (100 Mbps) como enlace típico, com o *backbone* apresentando capacidade para operar em *Gigabit Ethernet* (2 Gbps). Para permitir a troca de informações à distância de forma eficiente, foi utilizado o protocolo ATM (*Asynchronous Transfer Mode*), que oferece recursos como baixa latência e possibilidade de envio de prioridades no envio de pacotes.

4. Visualizador contextual de informações médicas.

Permite o acesso a toda informação disponibilizada pela infra-estrutura montada pelo projeto PACS-HC.

2.5 Visualização e processamento de imagens

As imagens médicas podem ser exibidas de diferentes formas para o profissional de Saúde, de acordo com as diferentes informações diagnósticas de cada imagem e com a natureza das mesmas. Algumas características globais que devem ser consideradas para a visualização de imagens médicas são:

- *Contraste*: é a variação de intensidade da imagem, que permite às pessoas diferenciarem regiões distintas de uma imagem;
- *Resolução de imagem*, que é o grau de detalhes discerníveis de uma imagem. Depende da frequência de amostragem da imagem;
- *Resolução de intensidade*, que é o número de bits por elemento da imagem;
- *Cor*, se a imagem é colorida ou em tons de cinza;
- *Espaço ocupado em disco / memória*: devido às limitações de hardware e software, podem ser necessárias técnicas de otimização de memória e processamento, para exibir imagens muito grandes.

De maneira geral, podemos dividir as imagens (e, conseqüentemente, a sua visualização) de acordo com a sua dimensão:

- *Imagens 1D*: sinais biológicos/fisiológicos, como ECG. São exibidas na forma de gráficos, com pares de pontos XY;
- *Imagens 2D*: imagens estáticas, como radiografias, de ultra-som, microscópicas. Dependendo do tamanho, pode ser necessário redimensioná-las para exibição, com o uso de algoritmos de interpolação;
- *Imagens 2D Dinâmicas*: imagens dinâmicas, como angiografias e de ultra-som. Necessitam redimensionamento e controle da taxa de exibição;

Figura 25 – Imagens médicas. a) Eletrocardiograma b) Radiografia
c) Angiografia d) por Ressonância Magnética (Extraído do *site* do programa MRICro¹¹).

- *Imagens 3D*: imagens volumétricas, como empilhamentos de cortes tomográficos (CT, PET e SPECT), de ultra-som, por ressonância magnética, de microscopia. Para a visualização da imagem na tela do computador a mesma é

¹¹ <http://people.cas.sc.edu/rorden/render.html>

projetada em 2D. A imagem 3D é comumente reprojeta em 2D para ser visualizada. A forma como a imagem é projetada difere de acordo com objetivo proposto (cirurgia virtual, segmentação) e capacidade de processamento. Para obterem-se as imagens 2D, podem ser aplicadas várias classes de algoritmos, como *Ray-Tracing*, *Volume Rendering* e Extração de Superfície;

- *Imagens 3D Dinâmicas*: imagens volumétricas adquiridas ao longo do tempo, como *Spiral CT*, *Gated SPECT*, ultra-som e *Gated MR*. São tratadas visualmente como imagens 3D projetadas ao longo do tempo.

O processamento de imagens médicas permite oferecer ao médico informações quantitativas, objetivas e reproduzíveis para suportar as avaliações sobre os pacientes, ao invés de se basear nas avaliações subjetivas realizadas pelos especialistas (Sonka, 2000).

Os passos fundamentais do processamento de imagens são mostrados na Figura 26 (adaptado de Gonzalez (2000)) e descritos a seguir.

Figura 26 – Passos fundamentais do processamento de imagens (Gonzalez, 2000).

- *Aquisição de Imagens*: passo no qual ocorre a digitalização do sinal obtido por sensores;
- *Pré-processamento*: responsável pela melhora da imagem de maneira a aumentar as chances de sucesso dos processos seguintes. Inclui filtros e variação de contraste;
- *Segmentação*: Neste passo, divide-se uma imagem de entrada em seus objetos constituintes, como por exemplo, com *thresholding*;

- *Representação e descrição*: converte os dados da segmentação, em regiões ou fronteiras, e extrai características fundamentais para identificação de classes de objetos.
- *Reconhecimento e interpretação*: Atribui-se um rótulo a um objeto, baseado nas informações do descritor. Para a interpretação, dá-se um significado a um conjunto de objetos reconhecidos.

Alguns tipos de processamento importantes para imagens médicas são:

- *Segmentação*: realizado com diferentes algoritmos como *thresholding*, *snakes*, *region-growing*, *split-merge* (*octree*, *quadtree*), *fuzzy connectedness*, redes neurais e métodos estatísticos;
- *Fusão*: Reunião de informações anatômicas e funcionais para obter maior capacidade diagnóstica;
- *Quantificação*: cálculo área, perímetro, volume, fração de ejeção, etc.

Contexto e Regras

“Imagination is more important than knowledge.”

Albert Einstein

Conforme colocado no Capítulo 2, uma vez definido o conceito de contexto, é necessário representá-lo e utilizá-lo de forma significativa. A representação do contexto adotada neste trabalho foi através da descrição de uma ontologia, enquanto que o contexto foi ativado por meio de um conjunto de regras, com o uso de *forward-chaining*.

Neste capítulo serão apresentados o modelo do contexto e de regras propostos para o visualizador contextual, descritos na forma de diagramas UML e XML.

3.1 Modelo de Contexto

Dentre as diversas alternativas para representar o contexto optou-se por utilizar uma descrição em diagramas UML (notação descrita no Apêndice A), que pode ser implementada de diferentes formas, inclusive como uma ontologia (Cranefield e Purvis, 1999).

Uma ontologia é uma especificação de uma conceitualização. Em outras palavras, ela é uma especificação explícita de algum tópico, ou uma representação formal e declarativa de alguma área do conhecimento. Ela especifica conceitos para serem usados para expressar conhecimento a respeito de uma determinada área. Este conhecimento compreende tipos de entidades, atributos e propriedades, relações e funções, assim como restrições. O propósito de uma ontologia é o compartilhamento e

reuso do conhecimento entre sistemas baseados em conhecimento e agentes. (Devedzic, 1999).

O modelo de contexto proposto é centralizado no Hospital. O hospital é definido como sendo composto de quatro elementos (Figura 27, ver notação UML no Apêndice A): Infra-estrutura, Sistema Informatizado, Fluxo de Trabalho (*Workflow*) e Recursos Humanos. A Infra-estrutura representa a estrutura física do hospital, indicando presença de computadores, *notebooks*, equipamentos médicos, pontos de rede (Figura 28).

Figura 27 – Modelo de representação do hospital.

Figura 28 – Especialização da infra-estrutura, representando elementos físicos, como computadores, leitos e redes.

O Fluxo de Trabalho (*Workflow*) representa os processos existentes em um hospital. O *workflow* indica, por exemplo, o processo realizado para obtenção de uma imagem de ressonância do tórax (admissão, agendamento, aquisição de imagem, realização de laudo e disponibilização dos resultados). O Sistema Informatizado caracteriza os sistemas de informação hospitalar disponíveis para acesso e consulta. Representa, por exemplo, o HIS (*Hospital Information System*) e o PACS (*Picture Archiving and Communication System*) de um hospital. Finalmente, os Recursos Humanos representam os funcionários de um hospital, com seus papéis e autorizações de acesso.

Nos itens a seguir, são apresentados com mais detalhes cada um dos elementos que compõem o modelo do contexto.

3.1.1 Infra-estrutura

Como apresentado no item anterior, a infra-estrutura representa elementos físicos do hospital e foi modelada como sendo composta de diversos componentes (Figura 27). O computador é um exemplo de um componente da infra-estrutura e é mostrado na Figura 29. Ele é composto de dois itens: *Software* e *Hardware*. Os itens de software representam programas de computador, como por exemplo, um aplicativo de escritório ou instâncias do próprio Visualizador Contextual ou mesmo configurações de rede. Os itens de hardware representam peças do computador, como placas de vídeo, memória RAM e monitores.

Esta modelagem permite que, por exemplo, sejam obtidas as máquinas no hospital que possuem instalado um visualizador contextual e se estabeleça a comunicação com os mesmos. Outra possibilidade é saber a capacidade do hardware de um equipamento (quantidade de memória, tipo de placa de vídeo), informação necessária quando se deseja, por exemplo, realizar um processamento tridimensional intensivo.

Figura 29 – Modelo de um computador como componente de infra-estrutura.

A Infra-estrutura tem associado um local, o qual pode representar um andar do edifício, uma sala específica, um prédio, ou qualquer outra localização física, como mostrado na Figura 30.

Figura 30 – Associação da Infra-estrutura com uma localização.

A associação de um local com a infra-estrutura permite determinar de onde o usuário está acessando o sistema. Essa informação é relevante quando se acessa o sistema de um local crítico, por exemplo, a partir de uma UTI (Unidade de Tratamento Intensivo), na qual as regras para acesso a informação são diferenciadas.

3.1.2 Fluxo de Trabalho

O Fluxo de Trabalho representa os processos realizados dentro de um hospital, como atendimento, marcação de consultas e agendamento de exames.

Para tornar o fluxo de trabalho uma representação mais genérica, definiu-se uma estrutura de classes que suporta a construção de grafos, mostrada na Figura 31. O *Workflow* é composto de conexões que ligam dois passos do *workflow*.

Figura 31 – Exemplo da representação de um fluxo de trabalho para uma tomografia de tórax.

Seja um fluxo para um exame de tomografia de tórax como mostrado na Figura 32. Os passos a serem seguidos são: **admissão** (ou cadastramento) no qual o paciente é registrado no sistema; **agendamento**, no qual é marcada uma data para realização do exame; **aquisição**, no qual o exame é realizado e a imagem é adquirida; **realização do**

laudo, aonde um especialista emite laudo a partir das imagens; e **liberação do laudo**, no qual o médico que solicitou o exame passa a ter acesso ao resultado do laudo.

Neste caso, cada etapa do processo é uma subclasse de *PassoWorkflow* e está conectada com a seguinte através de uma conexão (classe *Conexoes*), que indica o sentido do fluxo através dos atributos 'de' e 'para'. O *Workflow* de tomografia de tórax contém as conexões, englobando todas as etapas do processo.

Cada passo do fluxo de trabalho é associado a um (ou mais) profissional(ais) de saúde e um local de realização. A utilização deste contexto permite que, por exemplo, se saiba qual a próxima etapa de uma tarefa que está sendo realizada pelo profissional de saúde e seu local de realização.

Figura 32 – Representação diagramática do fluxo de trabalho (simplificado) para aquisição de um CT.

3.1.3 Sistema Informatizado

Um sistema informatizado representa as informações digitais disponíveis para utilização no hospital e as meta-informações associadas a essas fontes de dados. Alguns exemplos de sistemas são o HIS e PACS.

No modelo (Figura 33), um Sistema Informatizado contém Recursos, Protocolo de Acesso, Entrada e Saída de Dados. Um Recurso está relacionado com o controle de acesso e determina quais são os módulos do sistema que podem ser gerenciados por um

controle de acesso. Por exemplo, um HIS pode ter módulos para prescrição e consulta, que são acessados por profissionais distintos, com tarefas diferenciadas. Assim, poderiam ser determinados os recursos “leitura de prescrição”, “criação de prescrição”, “marcação de consulta”, “cancelamento de consulta” e assim por diante. A utilização dos recursos é discutida no item seguinte, quando for descrito o Profissional de Saúde.

Figura 33 – Modelo de um sistema informatizado.

O Protocolo de Acesso é uma meta-informação para determinar como o sistema pode ser acessado. São exemplos de Protocolo de Acesso HTTP (*Hypertext Transfer Protocol*) e RMI (*Remote Method Invocation*). Um exemplo de utilização do contexto de protocolo de acesso seria abrir automaticamente um navegador de Internet apontando para uma página que acesse o sistema. Outra possibilidade é desabilitar todos os menus que acessam um sistema com protocolo HTTP, uma vez que se percebe que não existe conexão de rede.

A Entrada de Dados descreve a forma pela qual os dados do sistema especificado são submetidos para consulta. Um exemplo de entrada de dados é um formulário WEB, cuja descrição pode ser utilizada, contextualmente, para acionar um recurso de autopreenchimento, completando dados como nome, RG, identificador do paciente entre outros.

A Saída de Dados descreve qual o tipo de informação é obtida a partir do Sistema Informatizado. A Saída de Dados está associada a um ou mais Documentos Eletrônicos.

O Documento Eletrônico (Figura 34) descreve um tipo de informação médica e sua representação digital. Por exemplo, uma imagem no padrão DICOM (*Digital Imaging and Communications in Medicine*) é um tipo de documento eletrônico (mais especificamente, é uma subclasse de *DocImagens*).

Figura 34 – Modelo de um Documento Eletrônico.

Um Documento Eletrônico tem associados uma forma de armazenamento e um tipo de representação. O *Armazenamento* indica, por exemplo, se o dado gerado pela Saída de Dados é um arquivo ou está em memória. A *Representação* refere-se à forma como o documento é disponível para o visualizador. No caso de uma imagem DICOM, pode-se utilizar várias bibliotecas para ler o arquivo, sendo que cada uma possui uma API (*Application Programming Interface*) específica para acessar o arquivo. Essa é uma meta-informação que facilita o acesso ao dado se a API utilizada for conhecida.

Para deixar mais claro o modelo de um Sistema de Informação, é mostrado na Figura 35, a representação para um servidor DICOM. Em cinza, destacam-se os modelos mostrados anteriormente na Figura 33 e Figura 34.

Figura 35 – Extensão do modelo para incluir a representação de um servidor DICOM como uma fonte de dados. Ressaltado em cinza, aparecem os modelos apresentados nas Figura 33 e Figura 34.

A representação do servidor DICOM é feita com a definição do protocolo de acesso (*ProtDICOM*), a forma de entrada (*C-STORE*) e saída (*C-GET*, *C-FIND* e *C-MOVE*) dos dados. A saída de dados gera documentos eletrônicos (*ImagemDicom*) que são armazenados na forma de arquivos (*SistemaDeArquivos*) e a biblioteca que representa a imagem DICOM é a *dcm4che* (2004).

3.1.4 Recursos humanos

Os recursos humanos representam os funcionários que trabalham em um hospital. O *RecursoHumano* possui N profissionais de saúde, sendo que cada profissional de saúde tem uma profissão (por exemplo, médico ou paramédico) e um ou

mais papéis associados (por exemplo, médico e chefe de equipe). Também, cada profissional possui um cadastro de informações pessoais (endereço, RG, número de matrícula, etc).

Figura 36 – Modelo dos Recursos Humanos.

A partir do *ProfissionalSaude* é possível recuperar informações relativas ao usuário, como o seu papel, as quais podem ser utilizadas contextualmente para, por exemplo, determinar qual o visualizador a ser utilizado pelo usuário (um cardiologista vê a imagem com uma interface e o cirurgião com outra).

Cada profissional de saúde tem um papel associado, que indica qual a sua função dentro do hospital (Figura 37). Os conceitos de papel e profissional de saúde foram determinados como entidades separadas, pois um representa a profissão da pessoa, enquanto o outro (o papel) representa as responsabilidades que a pessoa assume dentro do hospital. Os papéis são relacionados com a capacidade de acesso aos sistemas informatizados, mais especificamente, aos recursos de cada sistema informatizado (*Recurso*, mostrado na Figura 33). Assim, um médico com papel de imagenologista pode, por exemplo, processar imagens enquanto que um médico com papel de residente pode somente visualizar as imagens.

Figura 37 – Papéis de um Profissional de Saúde.

3.1.5 Raciocínio Clínico

O modelo de contexto demonstrado até o momento tem o ponto de vista do hospital como motivação principal. Porém, também é possível representar informações mais significativas para a atividade do médico, como o raciocínio clínico utilizado para diagnosticar uma doença. Para demonstrar essa possibilidade, optou-se por incluir uma representação de fluxogramas de diagnóstico ou *guidelines*, aproveitando-se do conceito apresentado no modelo dos fluxos de trabalho, isto é, a idéia de diagramas de fluxo. Com esta abordagem é possível representar procedimento médicos e *guidelines*. A razão para a inclusão deste modelo é para que seja possível utilizar conhecimentos *a priori* de procedimentos, de forma a poder auxiliar o médico na tomada de decisões.

Na Figura 38 é mostrado o diagrama de classes representando um raciocínio clínico. A classe *RaciocínioClinico* é composta de várias *Conexões*, as quais possuem 2 nós. Os nós podem ser de vários tipos. São representados na figura os nós de *Sintoma*, *Doença* e *Procedimento*.

Figura 38 – Modelo de fluxogramas para procedimentos médicos e *guidelines*.

Com o modelo de classes da Figura 38 é possível representar diagramas como o mostrado na Figura 39. Basta representar cada nó como uma das subclasses de Node (Figura 38) e associá-los através de uma conexão, como mostrado na Tabela 2. O conjunto de conexões forma uma instância de *RaciocinioMedico*.

Figura 39 – Hipertensão pulmonar e *cor pulmonale* na DPOC (Doença Pulmonar Obstrutiva Crônica)
(Extraído de Lopes, 1995).

Tabela 2 – Representação do diagrama da Figura 39 utilizando-se o modelo de raciocínio clínico.

<i>ID</i>	<i>De</i>	<i>Para</i>
1	Área da vasculatura pulmonar	Pressão Arterial Pulmonar
2	Debito Cardíaco	Pressão Arterial Pulmonar
3	Volume Sangüíneo Pulmonar	Pressão Arterial Pulmonar
4	Viscosidade sangüínea	Pressão Arterial Pulmonar
5	Pressão venosa pulmonar	Pressão Arterial Pulmonar
6	Pressão Arterial Pulmonar	Pós-carga para o VD
7	Pós-carga para o VD	Dilatação/Hipertrofia VD
8	Dilatação/Hipertrofia VD	Desempenho VD
9	Disfunção Primária, outros fatores	Desempenho VD

3.2 Sistema de Inferência

O sistema de inferência adotado é proveniente de um projeto *open source* chamado DROOLS¹², o qual implementa o algoritmo Rete¹³, adaptado para a linguagem Java. O DROOLS utiliza *forward-chaining*, o que significa que é capaz de encadear regras para chegar a uma ou várias conclusões. A partir de um conjunto de fatos geram-se novas conclusões, as quais, por sua vez, podem gerar outras inferências através do acionamento de regras (Russell, 1995).

As regras no DROOLS são representadas por um XML, como mostrado na Listagem 1. No exemplo, a regra é equivalente à condição: SE (param1=='teste') E (param2>12) ENTAO imprima 'Regra foi ativada!'.

```
<rule name="nome da regra">
  <parameter identifier="param1">
    <class>java.lang.String</class>
  </parameter>
  <parameter identifier="param2">
    <class>java.lang.Integer</class>
  </parameter>
```

¹² <http://drools.org/>

¹³ Rete vem do Latin e quer dizer rede.


```
<java:condition>param1.equals("teste")</java:condition>
<java:condition>param2.intValue()>12</java:condition>

<java:consequence>
    System.out.println( "Regra foi ativada!" );
</java:consequence>
</rule>
```

Listagem 1 – Exemplo de regra. Equivalente à condição SE (param1=='teste') E (param2>12) ENTÃO imprima 'Regra foi ativada!'.

Para exemplificar a interação de regras com uso de *forward-chaining*, considere a representação da situação em que se decide a necessidade ou não de se pegar um guarda-chuva do armário antes de sair de casa¹⁴. Nesse caso, podem-se estabelecer as seguintes regras:

- a) SE está chovendo ENTÃO você precisa de um guarda-chuva;
- b) SE você precisa de um guarda-chuva ENTÃO pegue um do armário.

Essas regras podem ser representadas como mostrado na Listagem 2. Caso o objeto tempo indique chuva (*tempo.chuvoso()* igual a verdadeiro) então há a necessidade de um guarda-chuva (indicado pelo objeto do tipo *AcessorioNecessario*). A notificação (*assertObject*) de uma mudança no estado do sistema faz com que seja disparada a segunda regra. Na segunda regra verifica-se se o acessório necessário é um guarda-chuva. Em caso afirmativo, é impressa a sentença “Pegue um guarda-chuva do armário!”.

```
<rule name="determinação do tempo">
    <parameter identifier="tempo">
        <class>minhaclasse.Tempo</class>
    </parameter>
    <java:condition>tempo.chuvoso()</java:condition>
    <java:consequence>
        drools.assertObject(new minhaclasse.AcessorioNecessario("guarda-chuva"));
    </java:consequence>
</rule>
<rule name="o que fazer">
    <parameter identifier="acessorio">
```

¹⁴ Adaptado de <http://drools.org/Declarative+Programming>

```
<class> minhaclasse.AcessorioNecessario</class>

</parameter>
<java:condition>acessorio.igual("guarda-chuva")</java:condition>
<java:consequence>
    System.out.println( "Pegue um guarda-chuva do armário!" );
</java:consequence>
</rule>
```

Listagem 2 – Exemplo de caso em que duas regras são encadeadas.

Apesar de ser possível reproduzir este exemplo proceduralmente, quando o número de regras aumenta, a programação procedural torna-se bastante complexa, gerando código incompreensível e difícil de manter (IGDA, 2004). Assim, o uso de regras passa a ser vantajoso.

3.3 Exemplos de Aplicação do Modelo de Contexto

A seguir são mostrados exemplos que demonstram o uso das regras combinadas com o modelo proposto para realizar ações sobre o visualizador contextual. Os exemplos mostram o modelo implementado com a ferramenta Protégé, descrita no Capítulo 2. Os resultados das ações são mostrados na implementação do visualizador contextual (ODIN), o qual é mostrado em detalhes no Capítulo 4.

3.3.1 CASO 1: Recuperação de imagem semelhante

Neste caso, desejamos mostrar uma lista de imagens semelhantes quando um usuário for um residente e abrir uma imagem qualquer.

Para o usuário conseguir acessar o visualizador, é necessário que ele forneça um *login* e uma senha de acesso. A partir do *login*, é possível recuperar dados cadastrais do usuário. Na Figura 40, é mostrado, na implementação, como os dados do usuário estão cadastrados (correspondente ao modelo da Figura 36), contendo informações de sexo, identificação no hospital (matrícula), ramal, local de trabalho e *login*. É importante notar que esse modelo deve ser adaptado para cada hospital no qual o visualizador é utilizado, pois os campos refletem a forma de organização do mesmo. Por exemplo, o campo matrícula pode ser irrelevante para outro hospital, ou mesmo pode ser referido internamente de outra forma e, portanto, esse campo deveria ser renomeado. Isto é relevante, pois existe uma política de trabalho dentro de cada instituição com um

vocabulário característico e, para o sistema poder ser compreendido e utilizado facilmente, ele deve utilizar expressões familiares localmente.

Figura 40 – Exemplo de informação do usuário contida no modelo. À esquerda se encontra o modelo, no centro estão as instâncias e à direita estão as informações de cada instância.

A recuperação de informações do modelo pode ser feita de diversas formas, porém, na implementação desenvolvida para este trabalho (descrita em detalhes no Capítulo 4 – Visualizador Contextual) utiliza-se uma linguagem parecida com Lisp (e parte do Protégé), a qual será utilizada para demonstrar a recuperação da informação contextual.

Por exemplo, para a recuperação da informação da unidade de trabalho, é possível executar o comando descrito na Listagem 3.

```
( (:instance CadastroInCor ?cads)
  (login ?cads "ramon.moreno")
  (lotacao ?cads ?lot) )
```

Listagem 3 – Comandos em linguagem similar a Lisp (*plugin* Algernon do Protégé).

Este comando pode ser interpretado como “coloque todas as instâncias de CadastroInCor na variável *cads*. Em seguida, do conjunto de instâncias armazenadas em *cads*, separe aquelas cujo login seja igual a ‘ramon.moreno’. Finalmente, coloque o conteúdo da variável *lotação* de cada instância restante na variável *lot*. Outra interpretação pode ser dada graficamente, utilizando-se conjuntos. Cada elemento entre parênteses pode ser imaginado como uma condição que reduz o universo possível de respostas (algumas mantêm o conjunto do mesmo tamanho). Quando todas as condições são satisfeitas tem-se a resposta final (Figura 41).

Figura 41 – Ilustração didática para a linguagem de pesquisa dos dados.

A busca por uma imagem similar será feita no servidor BIRAM. O primeiro passo é obter os papéis do usuário. Segundo o modelo (Figura 37), um profissional de saúde tem um ou mais papéis. É possível recuperar a informação do papel com o comando da Listagem 4, o qual separa o cadastro dos profissionais com login igual a ‘ramon.moreno’ e armazena os papéis na variável *pps*.

```
( (:instance ProfissionalSaude ?prof)
  (cadastro ?prof ?cads)
  (login ?cads "ramon.moreno")
  (papeis ?prof ?pps) )
```

Listagem 4 – Análise da base para descobrir o papel do usuário.

Para limitar a resposta ao papel de residente, é feita uma alteração no comando, incluindo esta restrição do tipo de papel.

```
( (:instance ProfissionalSaude ?prof)
  (cadastro ?prof ?cads)
  (login ?cads "ramon.moreno")
  (papeis ?prof ?pps)
  (nome ?pps "Residente") )
```

Listagem 5 – Análise da base para o usuário com papel Residente.

Se o resultado for vazio, quer dizer que a pessoa não tem um papel de residente. Se o resultado tiver algum valor, o usuário é um residente.

Os papéis do médico podem ser vistos na Figura 42.

Figura 42 – Papeis do médico.

Uma vez de posse da informação contextual, é necessário criar a regra que realiza a pesquisa no BIRAM. A regra elaborada é mostrada na Figura 43, a seguir.

```
<rule name="Busca BIRAM">
  <parameter identifier="memoria">
    <class>java.util.Hashtable</class>
  </parameter>

  <java:condition>memoria.get("user.residente")!=null</java:condition>
  <java:condition>memoria.get("algoritmo")!=null</java:condition>

  <java:consequence>
    odin.source.BiramDataSource src = new odin.source.BiramDataSource();
    // cria o holder dos parametros de pesquisa
    ParameterBlock pb = new ParameterBlock();
    // acrescenta o algoritmo selecionado como parametro
    pb.add(memoria.get("algoritmo"));
    // recupera o documento corrente
    Document d =GerenciadorInterface.getInstance().getCurrentDocument();
    // acrescenta o documento como parametro
    pb.add(d);
    // realiza a pesquisa
    src.search(pb);
    // obtem o resultado da pesquisa
    Document resp = src.getResult();
    // exibe o resultado
    GerenciadorInterface.getInstance().showDoc(resp);
  </java:consequence>
</rule>
```

Figura 43 – Regra para busca automática por imagens.

Segundo esta regra, o BIRAM é pesquisado se o usuário for residente e se tiver sido definido um algoritmo de pesquisa da imagem. Porém, como não foi definido nenhum algoritmo, esta regra nunca é disparada.

Necessitamos então acrescentar outra regra ao sistema, que defina qual é o algoritmo a ser utilizado. A regra escolhida é mostrada na Figura 44.

```
01 <rule name="Configura algoritmo">
02   <parameter identifier="memoria">
03     <class>java.util.Hashtable</class>
04   </parameter>
05
06   <java:condition>memoria.get("lista.algoritmos")!=null</java:condition>
07
08   <java:consequence>
09     //codigo simplificado por didatismo
10     DocImagemDicom d = (DocImagemDicom)GerenciadorInterface.
11       getInstance().getCurrentDocument();
12     DicomHeader dh = d.getHeader();
13     String mod = dh.getString(Tags.Modality);
14     Algoritmo[] algs = (Algoritmo[])memoria.get("lista.algoritmos");
15     for(int i=0;i<algs.length;i++){
16       String modAlg = algs.getModalidade();
17       // Testa se a modalidade suportada pelo algoritmo
18       // é da mesma modalidade da imagem.
19       if(modAlg.equals(mod)){
20         memory.put("algoritmo",algs[i]);
21         drools.modifyObject(memory);//notifica que mudou estado
22         break;
23       }
24     }
25   </java:consequence>
26 </rule>
```

Figura 44 – Regra que determina o algoritmo mais apropriado para pesquisa no BIRAM.

Esta regra é ativada se existir uma lista dos algoritmos possíveis de serem utilizados. Se existirem, os algoritmos são percorridos, buscando aquele que for mais apropriado para a imagem corrente, usando como critério de escolha a modalidade da imagem. Assim, a regra seleciona o primeiro algoritmo que é capaz de ser aplicado àquela modalidade e coloca essa informação na memória. Em seguida, notifica que a

memória foi alterada (*modifyObject*, linha 21), o que ocasiona o disparo da primeira regra, uma vez que agora todas as condições são satisfeitas. Apesar da seleção do algoritmo ter sido feita da forma mais simples, isso não impede que seja elaborada uma forma mais complexa, inclusive com mais regras, que incrementem a “inteligência” do visualizador contextual.

3.3.2 CASO 2: Notificação de exames do HIS

Uma outra aplicação possível para o visualizador contextual é notificar a existência de exames no Sistema de Informação Hospitalar (HIS), referentes ao paciente cujas imagens estão sendo observadas.

Uma maneira simples de fazer isso é através das regras mostradas na Figura 45.

```
01 <rule name="Busca HIS">
02   <parameter identifier="memoria">
03     <class>java.util.Hashtable</class>
04   </parameter>
05   <parameter identifier="header">
06     <class>odin.document.HeaderDicom</class>
07   </parameter>
08   <java:condition>header.getString(Tags.InstitutionName)
09     .equals(" InCor ")</java:condition>
10   <java:consequence>
11     HISInCorDataSource src = new HISInCorDataSource();
12     ParameterBlock pb = new Parameterblock();
13     String id = header.getString(Tags.PatientID);
14     pb.addInt(HISInCorDataSource.PESQUISA_IDPAT);
15     pb.add(id);
16     src.search(pb);
17     DocSearchResult resp = src.getResult();
18     if(resp.size()>0){
19       memoria.put("exames.his", resp);
20       drools.modifyObject(memoria);
21     }
22   </java:consequence>
23 </rule>
24 <rule name="Confirma HIS">
25   <parameter identifier="memoria">
26     <class>java.util.Hashtable</class>
27   </parameter>
```

```
28 <java:condition>memoria.get("exames.his")!=null</java:condition>
29 <java:consequence>
30     int retVal = JOptionPane.showConfirmDialog(null,"Existem exames"
31         " no HIS do mesmo paciente. Deseja vê-los?", YES_NO_OPTION);
32     if(retVal==JOptionPane.YES_OPTION){
33         Document doc = (Document)memoria.get("exames.his");
34         GerenciadorInterface.getInstance().showDoc(doc);
35     }
36 </java:consequence>
37 </rule>
```

Figura 45 – Regras para busca de exames no HIS do InCor.

Neste caso, a primeira regra efetua uma busca no HIS do InCor, verificando se a imagem foi gerada no hospital (através do parâmetro *InstitutionName*, linha 8). Se algum resultado é encontrado, a memória recebe o resultado da pesquisa (*put(...)* linha 19) e é feita a notificação de que a memória foi alterada (linha 20). Isto causa o disparo da regra 2, a qual pergunta ao usuário se ele deseja ver o resultado da pesquisa (linha 30). Se o usuário concordar, o resultado é mostrado (linha 34).

O resultado de uma pesquisa pode ser visto na Figura 46.

Figura 46 – Resultado de pesquisa no HIS do InCor.

Uma maneira mais intuitiva de apresentar os dados dos exames é em forma gráfica. Por exemplo, para determinar o infarto de miocárdio, ou re-infarto, é comum a utilização de resultados de laboratório, mais especificamente, a medida de mioglobina, troponinas T e I e a fração MB da creatinofosfoquinase (CK-MB) (Joint European Society of Cardiology/American College of Cardiology Committee, 2000). Nesses casos, um paciente tem vários exames do mesmo tipo, adquiridos em diferentes datas e horários, sendo melhor observados em forma de gráfico, por indicar tendências. Além disso, acrescentamos a condição hipotética de que somente em equipamentos da Unidade Clínica de Emergência será feita essa visualização (Figura 47. Essa unidade é uma subclasse de Local, descrita no modelo pela Figura 30).

Figura 47 – Unidade Clínica de Emergência.

Neste caso, necessitamos obter a localização do computador, o que pode ser feito pela pesquisa mostrada na Listagem 6. Para isso, utiliza-se o modelo da Figura 29 e considera-se que a configuração de rede é um item de software do computador.

```
( (:instance Computador ?cmp)
  (software ?cmp ?soft)
  (:instance TCP_IP ?soft)
  (ip ?soft "200.122.123.111")
  (local ?cmp ?loc)
  (nome ?loc ?nm) )
```

Listagem 6 – Pesquisa para obter o local físico de acesso a partir do IP.

O comando pode ser traduzido como: selecione todas as instâncias de Computador, cujo item de software seja do tipo TCP_IP e tenha endereço IP '200.122.123.111'. Em seguida, coloque o nome do local na variável *nm*.

Isto permite criar a regra da Figura 48, a qual determina que se a unidade a partir da qual o usuário acessa o visualizador for a Unidade Clínica de Emergência, a forma de *display* preferido para dados é na forma de gráficos.

```
01 <rule name="Display para a UCEME">
02   <parameter identifier="memoria">
03     <class>java.util.Hashtable</class>
04   </parameter>
05
06   <java:condition>memoria.get("user.local")
07     .equals("UCEME")</java:condition>
08
09   <java:consequence>
10     memoria.put("preferred.display", "Grafico");
11     drools.modifyObject(memoria);
12   </java:consequence>
```

Figura 48 – Regra que permite a busca por localização.

Modificando a regra da Figura 45, podemos obter o comportamento desejado, como mostrado na Figura 49, sendo o resultado da ação apresentado no gráfico de CK-MB da Figura 50.

```
27 <rule name="Confirma HIS">
28   <parameter identifier="memoria">
29     <class>java.util.Hashtable</class>
30   </parameter>
31
32   <java:condition>memoria.get("exames.his")!=null</java:condition>
33
34   <java:consequence>
35     int retVal = JOptionPane.showConfirmDialog(null,"Existem exames"
36       " no HIS do mesmo paciente. Deseja vê-los?", YES_NO_OPTION);
37     if(retVal==JOptionPane.YES_OPTION){
38       Document doc = (Document)memoria.get("exames.his");
39       if(memoria.get("preferred.display").equals("Graficos")){
40         GeraGraficoCKMB gg = new GeraGraficoCKMB();
41         gg.setDocument(doc);
42         Document doc1 = gg.process();
43         GerenciadorInterface.getInstance().showDoc(doc1);
```

```
44     }
45     else{
46         GerenciadorInterface.getInstance().showDoc(doc);
47     }
48 }
49 </java:consequence>
50 </rule>
```

Figura 49 – Regra modificada. Mudanças em tom mais escuro.

Figura 50 – Gráfico de quantidade de creatinofosfoquinase (CK-MB) de um paciente ao longo do tempo.

Visualizador Contextual: Arquitetura e Implementação

*“The way to get started is to quit
talking and begin doing.”*

Walt Disney

Neste Capítulo, apresenta-se a implementação do visualizador contextual, mostrando sua arquitetura e resultados obtidos. Na seção 4.1, é mostrada a arquitetura geral do visualizador. Em 4.2, são mostradas, em detalhes, as fontes de dados contextuais desenvolvidas para serem acessadas pelo visualizador: servidor DICOM, CIAS e BIRAM. Em 4.3, descreve-se a implementação do visualizador contextual e mostram-se os resultados obtidos.

4.1 Arquitetura de software

O Visualizador Contextual é composto de três elementos principais: os *plugins*, o gerenciador de interface e o contexto (Figura 51). Existem três tipos de *plugins*: visualizadores, ferramentas e fontes de dados. O contexto é dividido em modelo e sistema de inferência. A partir das fontes de dados é possível acessar os servidores de dados remotos. Para este projeto, foram desenvolvidos três servidores: um servidor DICOM, o qual é presente em qualquer sistema PACS e corresponde ao padrão para acesso a imagens médicas; um servidor CIAS, o qual é um padrão relativamente recente do CORBAmed, que permite interoperabilidade entre instituições para troca de imagens; e o servidor BIRAM, uma base de casos de imagens, que fornece busca de

similaridade de imagens por conteúdo. O diagrama da inter-relação desses itens é mostrado na Figura 51.

Figura 51 – Esquema geral do Visualizador Contextual.

O desenvolvimento do Visualizador Contextual utilizou padrões de projeto e metodologias orientadas a objeto, além do desenvolvimento evolucionário. Também se preocupou em permitir o uso de bibliotecas já existentes, como o JAI (*Java Advanced Image*, descrita no Anexo B), Visad (2004), além de bibliotecas para visualização dos dados, como Java 3D (ver Anexo B).

4.2 Fontes de dados desenvolvidas para o projeto

Nos itens a seguir são mostrados os programas desenvolvidos como parte do projeto PACS-HC, respectivamente, o servidor DICOM, CIAS e BIRAM.

O servidor DICOM desenvolvido foi utilizado como parte de um projeto *open-source*, chamado MiniWEBPACS, que contou com o apoio do CNPq¹⁵.

¹⁵ Disponível em <http://miniwebpacs.sourceforge.net/>

O servidor CIAS é a única implementação, de nosso conhecimento, da especificação correspondente da OMG.

Apesar de terem sido utilizadas as fontes desenvolvidas, a arquitetura proposta é geral, suportando outros serviços.

4.2.1 Servidor DICOM

O servidor DICOM foi desenvolvido utilizando-se a biblioteca *dcm4che* (2004) e os bancos de dados MS Access, no desenvolvimento inicial, e PostgreSQL (2004) na implementação final.

O padrão DICOM é definido pela ACR-NEMA (2000), sendo composto de 16 partes e mais de 50 suplementos. Além de tratar da transmissão de imagens médicas, o padrão atualmente também trata da visualização das mesmas (por exemplo, *Grayscale Display*) e da codificação de laudos (através do DICOM *Structured Reporting*, também conhecido como DICOM SR).

Um servidor DICOM deve ser capaz de atender pedidos de transferência de imagens, tanto para armazenamento quanto para recuperação. Também deve ser capaz de responder às pesquisas por parâmetros e verificação de disponibilidade.

Os clientes de um servidor DICOM são equipamentos de aquisição de imagens, como tomógrafos por raios X e equipamentos de ressonância magnética, ou então são estações de trabalho. Estas últimas permitem a visualização das imagens com fins de revisão ou de processamento, por exemplo, para segmentar uma imagem de tomografia e visualizar veias e artérias tridimensionalmente.

Os serviços oferecidos pelo servidor DICOM são: C-STORE, para armazenamento dos dados, C-FIND para pesquisa, C-MOVE e C-GET para recuperação de imagens e, finalmente, ECHO para verificação. A letra C no nome dos comandos se refere às imagens, que são, dentro da representação do DICOM, elementos Compostos.

A biblioteca *dcm4che* implementa o protocolo para transmissão e armazenamento de imagens e é um código fonte aberto escrito em Java. A partir da biblioteca, foram implementados os comandos C-STORE, C-FIND e C-MOVE. O comando ECHO já é parte integrante da biblioteca e o C-GET é o menos utilizado, sendo dada preferência ao uso de C-MOVE, razão pela qual o mesmo não foi implementado.

Como é necessário que haja persistência dos dados e seja possível fazer pesquisas para recuperá-los, utilizou-se um banco de dados MS Access inicialmente e um banco PostgreSQL, posteriormente. As tabelas utilizadas nos bancos de dados são mostradas na Figura 52. O modelo de tabelas segue a representação de imagens do padrão DICOM, a qual é baseada no conceito que um Paciente possui um ou mais Estudos, que por sua vez possui uma ou mais Séries. As Séries contêm uma ou mais Imagens. Isso se reflete nas tabelas e em seus relacionamentos (Img_PatientLevel, Img_StudyLevel, Img_SeriesLevel e Img_ImageLevel com relacionamentos 1 para N entre as tabelas).

O modelo original das tabelas foi inspirado no CTN (*Central Test Node*, uma implementação de referência de um servidor DICOM provida pela RSNA – *Radiological Society of North América* ¹⁶) e nas tabelas utilizadas no Instituto do Coração.

Figura 52 – Diagrama de tabelas utilizadas na implementação do servidor DICOM.

¹⁶ <http://www.rsna.org/>

4.2.2.1 C-STORE

O armazenamento físico das imagens DICOM é realizado externamente ao banco de dados. Isto é, as imagens são salvas em uma pasta do sistema de arquivos e mantém-se no banco uma referência ao arquivo salvo. Isso porque as imagens DICOM podem atingir tamanhos muito grandes (700MB), não sendo conveniente armazenar esse dado em tabelas, especialmente por questões de *backup* e restauração.

As imagens geradas no padrão DICOM contêm, no cabeçalho, informações sobre os diferentes níveis de informação (estudo, séries, etc), as quais são utilizadas para preencher as tabelas do banco. Através da biblioteca pode-se recuperar um objeto do tipo *Dataset* (ver Listagem 7), o qual é utilizado para salvar a imagem em disco e para inserção no banco de dados.

```
protected void doCStore(ActiveAssociation assoc, Dimse rq, Command rspCmd)
    throws IOException
{
    InputStream in = rq.getDataAsStream();
    ...

    protected synchronized void update(File file, Dataset ds,
        String callingAE, String calledAE) throws SQLException {
    ...
        sql = "INSERT INTO IMG_PATIENTLEVEL(PACI_ID, "+
            "                                PATNAM, "+
            "                                PATBIRTIM, "+
            "                                PATBIRDAT, "+
            "                                PATSEX, "+
            "                                NUMPATRELSTU, "+
            "                                NUMPATRELSER, "+
            "                                NUMPATRELIMA, "+
            "                                INSERTDATETIME, "+
            "                                OWNER) "+
            "VALUES (?,?,?,?,?,?,?,?,?) ";
        statement = con.prepareStatement(sql);
        statement.setString(1, ds.getString(Tags.PatientID));
        statement.setString(2, ds.getString(Tags.PatientName, "UNKNOWN"));
        statement.setString(3, ds.getString(Tags.PatientBirthTime, ""));
        java.util.Date date = ds.getDate(Tags.PatientBirthDate);
        if(date!=null && date.getTime()>0){
            System.out.println("date.time="+date.getTime()+" "+date);
            statement.setDate(4, new java.sql.Date(date.getTime()));
        }
        else
            statement.setNull(4, Types.DATE);
    }
```



```
statement.setString(5,ds.getString(Tags.PatientSex));
statement.setInt(6,0);
statement.setInt(7,0);
statement.setInt(8,0);
statement.setTimestamp(9,new
java.sql.Timestamp(System.currentTimeMillis()));
statement.setString(10,callingAE);
statement.executeUpdate();
...
```

Listagem 7 – Parte da implementação do C-Store. Mostra como a informação da imagem (representada por *Dataset*) é armazenada em banco, para pesquisa posterior.

4.3.2.2 C-MOVE

O comando C-MOVE é uma requisição para a transferência entre duas entidades DICOM. Cada entidade tem um nome e o servidor DICOM, obrigatoriamente, deve poder relacionar esse nome a um IP e porta conhecidos. No caso, esse relacionamento é feito na tabela *Img_AECliente* (não mostrada).

Quando uma requisição chega, ela contém parâmetros que permitem discernir uma imagem (ou conjunto de imagens) dentre aquelas conhecidas pelo servidor. Outro parâmetro é o nome da entidade DICOM para a qual a imagem (ou imagens) deve(m) ser enviada(s). Na implementação, utiliza-se um SQL diferente dependendo do parâmetro da requisição, de forma a obter o caminho para as imagens solicitadas. Uma vez obtidas as imagens, elas são transmitidas para a entidade DICOM apropriada (Listagem 8).

```
protected MultiDimseRsp doCMove(ActiveAssociation assoc, Dimse rq,
    Command rspCmd)
    throws IOException, DcmServiceException {
    try {
        Command c = rq.getCommand();
        Dataset ds = rq.getDataset();// read out dataset
        InetAddress ia = assoc.getAssociation().getSocket().getInetAddress();
        String ip = ia.getHostAddress();
        return new MoveMultiDimseRsp(c, ds, serverAETitle, ip);
    } catch (Exception e) {
        throw new DcmServiceException(Status.UnableToProcess, e);
    }
}
```

...

```
// obten o caminho aonde as imagens estao
String serLevel = "SELECT IMG_PATH "+
    "FROM IMG_IMAGELEVEL "+
    ",      IMG_SERIESLEVEL " +
    "WHERE IMG_IMAGELEVEL.SERINSUID = IMG_SERIESLEVEL.SERINSUID " +
    "AND   IMG_IMAGELEVEL.SERINSUID = ? ";
```

Listagem 8 – Trechos da implementação do C-Move. Dependendo da requisição, utiliza-se um comando SQL diferente para obter o caminho para as imagens em disco. Após isso, as imagens são transmitidas para a entidade DICOM apropriada.

4.3.2.3 C-FIND

Em uma requisição do tipo C-FIND, são passados diversos parâmetros de pesquisa, na forma de pares nome-valor, os quais são utilizados para descobrir quais são as instâncias de imagens conhecidas pelo servidor que tem parâmetros similares (por exemplo, encontrar todas as imagens conhecidas pelo servidor cuja modalidade é ‘XA’ ou seja, todas as imagens de hemodinâmica que estão armazenadas no banco de dados).

Para implementar esse serviço, criou-se uma tabela de mapeamento dos dados, que converte um parâmetro DICOM em um nome de coluna no banco de dados (Figura 53). Assim é possível gerar dinamicamente uma resposta para a requisição de pesquisa, criando um comando SQL sob demanda (Listagem 9).

Figura 53 – Tabela para mapeamento de parâmetros DICOM para colunas do banco de dados.

```
...
String sql = "SELECT MAP_TABLE "+
    ",      MAP_COLUMN "+
    ",      MAP_TYPE "+
    "FROM IMG_MAP_TABLES "+
    ",      IMG_DICOM_DICT "+
    "WHERE DIC_ID = MAP_DIC_ID "+
    "AND DIC_GRUPO = ? "+
```

```

        "AND DIC_ELEMENTO = ? ";

PreparedStatement statement = con.prepareStatement(sql);

Vector foundTags = new Vector();
Vector foundTypes = new Vector();
Vector missingTags = new Vector();
Vector tables = new Vector();

String select = "SELECT ";
String from = "FROM ";
String where = "WHERE ";

for (Iterator it = command.iterator(); it.hasNext();) {
    DcmElement el = (DcmElement)it.next();
    int tag = el.tag();
    String group = Tags.toHexString(tag >>> 16, 4);
    String element = Tags.toHexString(tag & 0xffff, 4);
    statement.setString(1,group);
    statement.setString(2,element);
    ResultSet rs = statement.executeQuery();
    if(rs.next()){
        foundTags.add(new Integer(tag));
        String tabela = rs.getString(1);
        String coluna = rs.getString(2);
        select += tabela+"."+coluna+", ";
        if(!tables.contains(tabela)){
            tables.add(tabela);
            from += tabela + ", ";
        }
        int tipo = rs.getInt(3);
        foundTypes.add(new Integer(tipo));
        String value = el.getString(null);
        if(value!=null && !value.equals("")){
            //tratar wild cards (?,*) e range de datas
            if(tipo!=1){
                where += tabela + "." + coluna + "=" + value + " AND ";
            }
            else{
                where += tabela + "." + coluna + "<=" + value + " AND ";
            }
        }
        else{
            missingTags.add(new Integer(tag));
        }
    }
}

statement.close();
if(foundTags.size()==0) return;
...

```

Listagem 9 – Trecho de código ilustrando a criação dinâmica de consulta SQL a partir da requisição de um C-Find.

4.2.2 CIAS

O CIAS (*Clinical Image Access Service*) é parte da especificação do CORBAMED (*CORBA in Medicine*) e foi descrito no Capítulo 2 (seção 2.3).

A proposta do CORBAMED (OMG, 1998) é extremamente interessante pois ela, em teoria, permite que diferentes instituições, com sistemas legados ou não, possam trocar informações entre si. Isto é possível porque o CORBA (*Common Object Request Broker Architecture*) permite que se utilizem linguagens diferentes (por exemplo, Java e C++) para troca de informações de forma transparente. Na prática, isto quer dizer que é possível definir um método, por exemplo, *recuperarImagem(string id)* em IDL (uma linguagem parecida com C++ e Java) e fazer um mapeamento desse método (ou serviço) para a linguagem desejada, gerando os chamados *stubs* e *skeletons*, que nada mais são que conversores de tipos (ver Figura 54). O CORBA converte os parâmetros de chamada em um protocolo intermediário (IIOP – *Internet Inter-ORB Protocol*) utilizando o *stub* e novamente converte do IIOP para a linguagem do servidor utilizando o *skeleton* e faz o inverso durante a resposta do servidor.

Figura 54 – Esquema geral de funcionamento do CORBA.

O CORBAMED foi padronizado pelo OMG (*Object Management Group*)¹⁷, mesma criadora do CORBA, e é conhecido atualmente como *Healthcare Taskforce*. Entre outros serviços, ele padroniza interfaces para acesso a informações de pacientes (chamado PIDS – *Person Identification Service*), informações clínicas (COAS, *Clinical Observation Access System*) e imagens médicas (CIAS, *Clinical Image Access Service*).

¹⁷ <http://www.omg.org/>

Segundo a especificação do CIAS, ele “não tem a intenção de supplantar [o padrão] DICOM. Em lugar disso, ele tem a intenção de ser um ‘wrapper’ em torno de porções do padrão DICOM, de maneira a permitir o acesso a imagens clínicas e informação relacionada aonde a riqueza completa do padrão DICOM não é necessária (como no caso do Prontuário Eletrônico do Paciente). Os requisitos do CIAS somente tratam do acesso a imagens clínicas que não suporte diagnóstico a partir das imagens”. Apesar da última ressalva, a interface permite a obtenção de imagens não processadas (*raw*), o que mantém a qualidade das mesmas.

O CIAS possui vários níveis de *conformance*, e o implementado foi o mais simples, definido pela interface IDL *ImageAccess*. Esta interface é relacionada com o COAS como mostrado na Figura 55. Observando-se a relação de herança entre a interface *ImageAccess* do CIAS e *QueryAccess* do COAS pode-se perceber que o CIAS é uma especialização do COAS, herdando os métodos do mesmo. Dessa maneira, a implementação do CIAS implica também na implementação do COAS. As interfaces do CIAS e COAS são mostradas na Figura 56.

Figura 55 – Relacionamento entre a interface do CIAS (*ImageAccess*) e a interface do COAS (*QueryAccess*).

Além da definição dos métodos, a especificação da OMG para imagens determina estruturas de dados (iguais a *structs* em C) que representam uma imagem, e que são passados como resposta a uma pesquisa. As estruturas seguem o modelo mostrado na Figura 57.

Figura 56 – Interface de pesquisa de imagens (esquerda) e interface de busca de observações clínicas (direita).

Figura 57 – Representação de uma imagem segundo o modelo do CIAS.

A implementação do CIAS foi feita a partir das tabelas mostradas na Figura 58, as quais são baseadas no padrão DICOM e já existentes no InCor, não havendo por isso uma correspondência exata entre as estruturas de imagem e os campos das tabelas.

A implementação do CIAS revelou algumas deficiências do padrão do CORBAMED, conforme descrito no Capítulo 2, pois apesar de ser possível transmitir dados de maneira transparente entre programas de instituições diferentes e com linguagens de programação diferentes, ainda assim a interoperabilidade entre os sistemas não está garantida. Isto porque as interfaces do CIAS e do COAS foram definidas de forma muito abstrata, não permitindo que seja explicitado claramente o tipo de dado que é passado entre as instituições (Moreno, 2002a), especialmente para observações clínicas. Também constatou-se que a implementação, tanto do CIAS como do COAS, é muito custosa, em termos de tempo de desenvolvimento e especialização dos programadores, sugerindo-se um *framework* para desenvolvimento mais rápido de implementações do sistema (Moreno, 2003).

Figura 58- Tabelas do CIAS.

4.2.3 BIRAM

O BIRAM (Base de Imagens Relacional para Algoritmos e Métricas) é uma base de imagens médicas que aceita pesquisas por características intrínsecas da imagem, como, por exemplo, o histograma (isto é, busca por similaridade). O BIRAM armazena imagens, laudos associados e algoritmos de processamento. O laudo tem suas

conclusões codificadas segundo um dicionário médico (atualmente, utiliza-se o CID-10, mas suporta outros dicionários) e as imagens devem estar no formato DICOM. Os algoritmos de processamento devem ser feitos em Java, pois é utilizada a serialização do vetor de características, como explicado mais adiante. O acesso ao servidor é realizado através de RMI (*Remote Method Invocation*).

O funcionamento do BIRAM pode ser resumido da seguinte maneira: uma imagem DICOM é enviada para o servidor, a qual é indexada no banco de dados, preenchendo as tabelas de Paciente, Estudo, Series e Imagem (mostradas na Figura 59). De acordo com as características da imagem (por exemplo, modalidade), ela é processada pelos algoritmos armazenados, gerando vetores de características que são armazenados no banco de dados (tabela Alg_x_Img, Figura 59). Uma vez armazenada no banco, a imagem pode ser pesquisada pelos clientes, os quais enviam seus vetores de características para *matching*. Essa sequência de eventos é mostrada graficamente na Figura 60.

Figura 59 – Diagrama das tabelas do BIRAM.

Figura 60 – Diagrama de sequência exemplificando o funcionamento do BIRAM.

Os vetores de características armazenados no banco são, na realidade, classes Java serializadas (*Resultado* da Figura 61)¹⁸. Um resultado encapsula o conhecimento de como medir a distância para outro vetor de características (*calcDist()*). Já a classe *Algoritmo* (também na Figura 61) descreve informações sobre o tipo de imagens a que o algoritmo se aplica (modalidade, número de bits e parte do corpo). O método *process(imagem)* da classe *Algoritmo* gera um objeto do tipo *Resultado*.

¹⁸ A serialização é uma técnica para tornar uma instância persistente, isto é, guardar as informações do estado do objeto de maneira que seja possível de restaurá-lo posteriormente.

Figura 61 – Modelo de um algoritmo no BIRAM. A classe *Algoritmo* descreve informações sobre o tipo de imagens a que ele se aplica (modalidade, número de bits e parte do corpo). O método *process(imagem)* da classe *Algoritmo* gera um objeto do tipo *Resultado*, o qual é armazenado no banco de forma serializada, isto é, convertido em um *array* de bytes.

Na Listagem 10, é mostrado um trecho de código que demonstra como é possível realizar o *matching* no banco de dados¹⁹. Através de um *SELECT*, obtêm-se os vetores de características do banco, na forma de BLOBS (*Binary Large Objects*). Os BLOBS são convertidos de um array de bytes para um objeto Java do tipo *Resultado*, o qual representa o vetor de características. De posse do objeto *Resultado*, é possível realizar a comparação com outro objeto de mesmo tipo, através do método *calcDist(Resultado)*.

```

public ImagemInfo[] pesquisarImagens(Result recebido, int alg_id, int maxSize)
throws java.rmi.RemoteException {
    ...
    String SQL = "SELECT IMG_SOPINSUID, AXI_VETOR_CARAC FROM IMAGEM, ALG_X_IMG
    WHERE IMAGEM.IMG_SOPINSUID = AXI_IMG_ID AND AXI_ALGORITMO_ID = ?";
    PreparedStatement statement = connection.prepareStatement(sql);
    statement.setInt(1, alg_id);
    ResultSet rs = statement.executeQuery();
    while(rs.next()){
        String imgID = rs.getString(1);
        //de-serializa do banco e reconstrói o objeto Java
        java.io.ObjectInputStream in = new
    
```

¹⁹ Evidentemente o método mostrado é ineficiente, mas esse método pode ser melhorado, por exemplo, com o uso de *slim trees*, como em Traina (2002).

```
        java.io.ObjectInputStream(rs.getBinaryStream(2));
        Resultado r = (Resultado)in.readObject();
        in.close();
        double dist = r.calcDist(resRecebido);
        rank(imgID, dist);
    }
    ...
```

Listagem 10 – Exemplo de código realizando o *matching* de um vetor de características.

Na Listagem 11 pode-se ver a interface RMI completa para acesso ao BIRAM, enquanto que na Figura 62 são mostradas duas telas de um cliente que permite salvar laudos na base de imagens.

```
package baseimagens;
import java.rmi.*;

public interface Servidor extends Remote{
    // Recupera a imagem original DICOM do banco
    public byte[] recuperarImagem(String id) throws java.rmi.RemoteException;
    // Armazena uma imagem DICOM no banco
    public String armazenarImagemDicom(String id, byte[] array, String desc, boolean
    pending) throws RemoteException, WrongFormatException;
    // Realiza a pesquisa a partir de um vetor de características (Result)
    public ImagemInfo[] pesquisarImagens(Result rs, int alg_id, int maxSize) throws
    RemoteException;
    // Informa quais os algoritmos disponiveis
    public AlgoritmoInfo[] listarAlgs() throws RemoteException;
    // Lista os arquivos associados a um algoritmo
    public String[] listAlgorithmFiles(int algID) throws RemoteException;
    // Recupera um arquivo de um algoritmo
    public byte[] getAlgorithmFile(int algID, String fileName) throws RemoteException;
    // Submete um algoritmo na forma de jar
    public void sendAlgorithm(String className, byte[] classOrJar, boolean append) throws
    RemoteException;
    // Aplica algoritmo ao conjunto de imagens no banco
    public ApplyResponse[] applyAlg(int algID) throws RemoteException;
    // Associa um laudo a uma imagem do banco.
    // Após finalizar o laudo não poder ser alterado.
    public void salvarLaudo(Laudo laudo, boolean finalizar) throws RemoteException,
    WrongFormatException, ReportAlreadyFinalizedException;
    // Pesquisa a partir do laudo. Utilizando os codigos
    public Laudo[] pesquisarLaudo(String[] codes) throws RemoteException;
    // Pesquisa o laudo para um estudo especifico
    public Laudo pesquisarLaudo(String stuinsuid) throws RemoteException;
    public Paciente pesquisarPaciente(String id) throws RemoteException;
```

```
public Paciente[] pesquisarPaciente(Paciente params) throws RemoteException;
public Estudo[] pesquisarEstudo(Estudo params) throws RemoteException;

// lista os dicionarios
public DicInfo[] listarDicionarios() throws RemoteException;

//obtem todos os codigos possiveis de um dicionario.
public ItemCodificado[] listarCodigos(int dicId) throws RemoteException;
}
```

Listagem 11 – Interface Java para acesso ao BIRAM.

Figura 62 – Cliente para digitação de laudos. À esquerda, texto livre e, à direita, laudo codificado com CID-10.

O BIRAM possui as seguintes vantagens: (i) Pode receber diversos tipos de algoritmos de processamento de imagens, desde que sejam feitos em Java; (ii) Permite que o laudo seja codificado segundo um dicionário médico, então é possível buscar imagens por laudo e vice-versa; (iii) Permite que os algoritmos sejam aplicados somente em imagens com características adequadas, por exemplo, somente em imagens de ressonância magnética; (iv) Prevê o uso de regiões de interesse, apesar de ainda não implementado, para aplicação do algoritmo de processamento (como uma descrição geométrica, ou como máscara).

4.3 Implementação do visualizador contextual

Primeiramente será mostrada a arquitetura de classes do visualizador e, em seguida, os resultados da implementação.

O visualizador é baseado na noção de um documento eletrônico, que representa a abstração de uma informação que pode ser exibida na tela (Figura 63). Um documento (classe *Document*), é algo que pode ser aberto (*open()*), fechado (*close()*) e ser associado a um ou mais modos de apresentação (*addDisplay()*).

Figura 63 – Abstração de um documento.

Um documento é consistente com o padrão de projeto chamado *Observer* (veja padrões de projeto no Apêndice A), no qual as modificações no *Observable* (ou seja, o documento) refletem-se nos *Observers* (isto é, nos visualizadores).

A escolha por essa representação foi feita por permitir associar o mesmo documento com diferentes tipos de apresentação, atendendo a um dos requisitos do visualizador, que é a capacidade de exibir a informação de uma maneira diferente de acordo com o usuário que acessa o sistema.

O Documento pode ser especializado para representar informações mais interessantes para o visualizador contextual. Assim o Documento tem duas subclasses principais: *DocImagem*, o qual representa de forma genérica uma imagem qualquer e *DocLaudo*, o qual representa genericamente, um laudo (Figura 63). Estes dois tipos de documentos são considerados as fontes de informação contextual mais relevantes para o sistema. A classe *DocImagem* define, simplesmente, dois métodos: *getHeader()* e *setHeader()*, representando a divisão da imagem como sendo composta de um cabeçalho (*header*) e de um corpo. Esta classe possui quatro subclasses: *DocImagem1D*, *DocImagem2D*, *DocImagem2DDynamic* e *DocImagem3D*.

A imagem 1D é representada implicitamente como sendo um *array* de dados do tipo inteiro ou *double*. Essa escolha foi tomada pois os dados de imagens unidimensionais são em geral de menor dimensão e podem ser mantidos em memória, permitindo então um acesso direto à informação, o que é desejável quando se realiza um processamento intensivo dos dados. O modelo é incompleto, pois faltam métodos acessórios para dados de calibração da imagem, mas que não são imediatamente relevantes para o desenvolvimento do visualizador contextual e podem ser resolvidos ou com a adição desses métodos na interface ou através da especialização do *header*, para representar imagens unidimensionais.

A imagem 2D é altamente baseada na própria representação do Java, utilizando a classe *BufferedImage*. Essa representação é desejável, pois permite que sejam utilizadas todas as ferramentas de processamento desenvolvidas para Java que utilizam essa interface, além de facilitar o trabalho de outros desenvolvedores, que não terão que fazer conversões entre formatos para exibir e processar a imagem.

A classe *DocImagem2DDinamico* representa imagens dinâmicas bidimensionais. Essa representação oculta a forma pela qual as imagens são armazenadas internamente

(não adota, por exemplo, um *array* para representar a imagem), pois começam a ser relevantes considerações sobre a quantidade de memória utilizada e eficiência de processamento (Moreira et al, 2002). Por fim é definido o *DocImagem3D*, que representa uma imagem tri-dimensional. Classes de novas dimensões podem ser adicionada, bastando para isso estender a classe *DocImagem*.

Uma vez definida a representação de um documento, é possível derivar as representações dos visualizadores e das ferramentas de processamento. Basicamente, as ferramentas de processamento têm dois métodos relevantes: *canProcess(Document)* e *process()* (Figura 64).

Figura 64 – Esquema de classes dos *plugins*.

O primeiro método é utilizado para habilitar e desabilitar as ferramentas de processamento, assim que uma nova imagem é aberta, e o segundo é utilizado quando o usuário pressiona o menu referente a uma determinada ferramenta. Neste modelo não se pressupõe nenhuma informação sobre o tipo de processamento que está sendo realizado.

A única ação tomada automaticamente é mostrar um novo documento se algo for retornado do método *process*.

Os visualizadores (classe *Viewer*), por sua vez, tem como métodos principais: *display()*, *reset()* e *update()* sendo cada um responsável por, respectivamente, exibir a imagem corrente, limpar a exibição de qualquer imagem que esteja sendo mostrada e exibir novamente a imagem corrente. O método *update()* é a outra parte do padrão *Observer* utilizada para o documento mencionado anteriormente. Foram desenvolvidos visualizadores básicos para cada uma das dimensões de imagens.

Além da representação do documento, é necessário observar a representação das fontes de dados (*DataSource*, Figura 64), o uso de *widgets* e os *layouts*.

As fontes de dados são responsáveis por recuperar, a partir de um dispositivo qualquer, um documento para exibição. As fontes de dados estão divididas em dois tipos: dados locais e dados remotos. Essa separação é feita, pois o acesso a dados locais é diferenciado daquele feito a dados remotos. No primeiro, temos a representação do sistema de arquivos, e, no segundo, temos um acesso remoto, normalmente realizado por TCP/IP e *sockets*. Uma fonte de dados genérica (*DataSource* - Figura 64) tem como métodos principais *open()*, *close()* e *getDocuments()*. A fonte de dados local (*LocalDataSource*) tem uma referência específica ao sistema de arquivos com a passagem de um parâmetro *java.io.File*, enquanto que uma fonte de dados remota (*RemoteDataSource*) está associada a uma interface de pesquisa (método *getSearchGUI()*), uma vez que os parâmetros para acesso da mesma são conhecidos somente no momento da execução. O algoritmo de utilização de uma fonte de dados é:

- a) Para uma fonte local: deixar o usuário selecionar um arquivo. Uma vez selecionado, percorrer a lista de fontes conhecidas e avaliar se existe alguma capaz de abrir o arquivo (consultando o método *canOpen()*). Encontrada uma fonte capaz, abrir o arquivo (*open()*) e obter o documento correspondente. Mostrar o documento aberto;
- b) Para uma fonte remota: a partir do menu, o usuário seleciona qual fonte de dados deseja acessar. Obtém-se, a partir da fonte, a interface de pesquisa (*getSearchGUI()*) que é exibida para o usuário. Internamente, a fonte remota localiza e consulta seu servidor (por exemplo, o servidor DICOM) e retorna

uma lista de documentos que melhor se ajustam com a pesquisa feita. A partir da lista, o usuário seleciona a imagem que deseja ver.

Os *widgets* são componentes de interface que permitem a criação de novas opções de visualização a partir dos componentes existentes. Por exemplo, um componente *ImageDisplayWidget* permite exibir uma imagem 2D e realizar a medida de intensidade em uma imagem (Figura 65). Se for necessário desenvolver um novo visualizador (por exemplo, para imagens de raios X), pode-se utilizar este componente e reduzir o tempo de desenvolvimento.

Figura 65 – Componente reutilizável, “*widget*”.

Os *layouts* são os componentes básicos nos quais todos os outros componentes se apóiam. Os *layouts* determinam o local no qual os visualizadores e painéis de pesquisa serão dispostos, permitindo a modificação da disposição dos elementos na tela. É possível fazer, por exemplo, um *layout* no qual as imagens são exibidas em janelas individuais, ao invés de associadas a um único quadro.

Outros componentes relevantes a serem mencionados na arquitetura do visualizador contextual são o Gerenciador de Interface e o Gerenciador de *Plugins*.

O Gerenciador de Interface (Figura 66) é responsável pelo controle e integração de todos os elementos exibidos na tela. Ele é responsável pela construção dos menus, controle de exibição dos documentos e ferramentas. É o elemento central da arquitetura e permite o uso de informações contextuais no visualizador. É modelado como um *Singleton* (ver anexo A, em padrões de projeto) e é também um *Abstract Factory*, delegando parte do processamento a uma implementação subjacente. Essa solução de

projeto permite que o gerenciador de interface seja trocado, de forma a mudar o comportamento do visualizador. Assim, é possível ter um visualizador simples, sem nenhum comportamento contextual, ou um visualizador inteligente, simplesmente mudando-se o Gerenciador de Interface.

O Gerenciador de *Plugins* (Figura 66) segue a mesma arquitetura do Gerenciador de Interface (*Singleton* com *Abstract Factory*), de modo a desvincular a forma como os *plugins* são gerenciados daquela como eles são acessados. Assim é possível ter um gerenciador de *plugins* que lê as configurações dos mesmos a partir de um XML ou, alternativamente, outro que importa dinamicamente *plugins* através da rede e os instala na máquina do usuário. Pode-se notar que o desenvolvimento privilegiou a flexibilidade do visualizador para poder utilizar e recombina suas partes componentes, tornando-o o mais reconfigurável e extensível possível.

Figura 66 – Gerenciador de Interface e Gerenciador de Plugins.

A tela inicial do visualizador contextual é exibida na Figura 67. É pedido um *login* ao usuário, o qual, uma vez autorizado, permite recuperar seu perfil a partir do modelo de contexto.

Figura 67 – Tela inicial do visualizador contextual, requisitando *login*.

O funcionamento da exibição inicial do visualizador é mostrado no diagrama de sequência da Figura 68. O visualizador é iniciado pelo Gerenciador de Interface, o qual inicializa o Gerenciador de Plugins, a interface gráfica com o usuário (*OdinFrame*) e o sistema de regras. Em seguida, exibe a tela de *login* e, de posse do *login* do usuário, pode inicializar o perfil do mesmo, consultando o modelo de contexto (não mostrado na figura). Pode também avaliar ações disparando as regras (*fireAllRules()*).

Figura 68 – Diagrama de sequência, mostrando a inicialização do visualizador contextual.

Na Figura 69 é mostrada a visualização de uma imagem 1D. No caso, a imagem é resultante de um processamento realizado sobre outra imagem, criando um histograma.

Figura 69 - Visualização de imagem 1D.

Além das imagens 1D, também é possível visualizar imagens 2D estáticas e dinâmicas, mostradas na Figura 70 (a imagem dinâmica mostra somente um quadro da sequência).

No caso de imagens de *Gated Blood Pool*²⁰, o visualizador exibido é aquele mostrado na Figura 71, que permite o processamento específico, como o cálculo de fração de ejeção, mostrado na Figura 72. A fração de ejeção é uma medida da capacidade de bombeamento de sangue do coração, calculada como a diferença de volume entre a diástole e sístole, dividido pelo volume na diástole e medido em porcentagem ($FE = (V_{DIA} - V_{SIS}) / V_{DIA} * 100\%$). A diástole é caracterizada por relaxamento muscular e enchimento dos ventrículos com sangue e a sístole é caracterizada por contração muscular e esvaziamento dos ventrículos. No caso da imagem de *Gated Blood Pool* o volume é proporcional à soma das intensidades dos *pixels* na área do ventrículo.

Figura 70 – À esquerda, visualização de imagem bidimensional DICOM, com medida de intensidade. À direita, visualização de imagem 2D dinâmica (somente um frame).

²⁰ Gated Blood Pool é um estudo da função cardíaca realizado com o uso de radiofármaco e sincronização da aquisição dinâmica com o eletrocardiograma.

Figura 71 – Visualização de imagem de *Gated Blood Pool* em modo cine.

Figura 72 – Cálculo de fração de ejeção sobre imagens de *Gated Blood Pool*, através da delimitação de regiões de interesse.

É possível também, utilizando-se a biblioteca Visad (Visad, 2004) e a API Java 3D, visualizar imagens tridimensionais através de iso-superfície, isto é, conectando-se pontos com mesma intensidade, criando-se uma superfície para exibição. Como somente são exibidos os voxels que formam a iso-superfície, é possível manipular rapidamente a imagem mudando-se o ângulo de exibição (Figura 73). Essa forma de exibição é bastante apropriada para apresentar e manipular imagens reorganizadas em volumes a partir de cortes tomográficos (como nas imagens de Tomografia Computadorizada, por ressonância magnética, Tomografia Computadorizada por Emissão de Fóton Único e Tomografia por Emissão de Pósitrons).

Figura 73 – Visualização de imagem 3D utilizando a biblioteca Visad. À esquerda, cabeça humana e à direita, *phantom* MCAT (<http://www.bme.unc.edu/mirg/mcat/>).

Para as ferramentas de processamento, utilizou-se o JAI (*Java Advanced Imaging*) e implementaram-se as funcionalidades básicas de processamento: controle de brilho e contraste, *zoom*, rotação e aplicação de filtros digitais. Um exemplo do uso do JAI é mostrado na Figura 74.

Figura 74 – Imagem de radiográfica de tórax processada com brilho e contraste modificados, rotacionada e com dimensão reduzida.

Cada *plugin* tem associada uma configuração (Figura 64) chamada *PreferenceEditor*. As configurações de todos os *plugins* podem ser acessadas através de uma interface única, como ilustrado na Figura 75.

Figura 75 – Configurações dos *plugins*. À esquerda, tela inicial de configuração. À direita, configuração do servidor DICOM.

O visualizador também suporta a pesquisa nas fontes de dados remotos, como mostrada nas Figura 76 a Figura 78. Na pesquisa do BIRAM, a imagem utilizada como

referência de busca por similaridade é aquela que está sendo visualizada pelo usuário no momento da pesquisa. O algoritmo desejado para pesquisa é selecionado em uma *combo Box* (Figura 76). Os algoritmos implementados atualmente são histograma simples, histograma métrico (Bueno, 2002; Rosa²¹, 2002) e centro de massa.

Figura 76 – Interface de pesquisa do BIRAM com lista de resultados.

No CIAS (Figura 77), a pesquisa é feita pelo código qualificado (*qualified code*), sendo os níveis de pesquisa: Paciente, Estudo, Series e Imagem. A pesquisa pode ser restringida por um intervalo de tempo e/ou através de pares nome-valor com parâmetros equivalentes aos do DICOM (como modalidade e número do estudo).

Na pesquisa DICOM (Figura 78), selecionam-se o servidor DICOM (item *peer* na Figura 78) e o nível de pesquisa (*patient* ou *study*), e se preenchem as informações de pesquisa, sendo os campos possíveis nome de paciente, data de nascimento, identificador do paciente, sexo, data, descrição do estudo e número do estudo. É possível também fazer verificação do servidor (*echo*).

²¹ Algoritmo gentilmente cedido por Natália Rosa e Agma Traina.

Figura 77 – Interface de pesquisa do CIAS para chamada pelo método *get_observation_by_qualified_code()*.

Figura 78 – Interface de pesquisa do Servidor DICOM com lista de resultados.

Além de dados de imagens médicas também é possível acessar fontes de dados que recuperem dados textuais, como páginas WEB, conforme mostram a Figura 79 e

Figura 80, nas quais se recuperam dados do PubMed²². O PubMed é um serviço da NLM (*National Library of Medicine*)²³, e inclui mais de 15 milhões de citações de artigos biomédicos desde a década de 50. Estas citações são do MEDLINE e jornais científicos adicionais.

Figura 79 – Busca no site PubMed

Figura 80 – Visualização de texto recuperado do site PubMed.

²² <http://www.ncbi.nlm.nih.gov/entrez/query.fcgi>

²³ <http://www.nlm.nih.gov/>

A implementação do modelo do contexto foi feita utilizando-se o Protégé (Figura 81), descrito no Capítulo 2. Sua principal característica é tornar fácil a entrada de dados, pois é gerada dinamicamente uma interface para preenchimento de instâncias das entidades definidas (Figura 82). Também é possível customizar a interface de entrada de dados, por exemplo, definindo a interface para um valor numérico de faixa limitada como sendo uma barra de rolagem.

Figura 81 – Representação do modelo de contexto no programa Protégé (2004).

Figura 82 – Tela de entrada de dados do Protégé.

O uso do Protégé também é interessante porque ele pode atuar como um servidor, permitindo que vários clientes acessem o mesmo modelo, e se o mesmo for alterado, o efeito é refletido em todos os clientes.

Discussão

*“The aim of argument, or of discussion,
should not be victory, but progress”*

Joseph Joubert

Neste Capítulo é feita uma discussão sobre os diferentes aspectos do desenvolvimento do Visualizador Contextual. Na seção 5.1 é discutida a validade do Visualizador Contextual para atender às necessidades do profissional de Saúde na visualização de imagens médicas. Em 5.2, compara-se o modelo adotado com as representações definidas na revisão do Capítulo 2. E, em 5.3, discutem-se as limitações do modelo e possíveis extensões para seu aperfeiçoamento.

5.1 Contribuição do contexto para visualização

O modelo de contexto e o sistema de regras são uma contribuição importante para o desenvolvimento de sistemas interativos na visualização de imagens médicas e também para utilização dentro de instituições de Saúde.

A utilização do sistema de inferência combinado com o modelo de contexto proposto permite flexibilidade e poder expressivo para a definição de ações elaboradas na interação do Visualizador Contextual com o usuário. A flexibilidade decorre da utilização das regras, pois as lógicas de interação mais complexas, tradicionalmente embutidas nas aplicações, podem agora ficar isoladas do programa principal. Assim, mudanças na lógica das regras não implicam em mudanças no executável. Por outro lado, a capacidade de se criar contextos com alto nível de abstração concede ao visualizador o poder de expressar relacionamentos e necessidades no nível

organizacional dos hospitais, normalmente ignoradas na elaboração de programas de visualização e processamento de imagens médicas.

Ademais, o modelo proposto permite a customização do contexto para representar um ambiente específico em uma instituição de saúde, uma necessidade para organizações reais.

A visualização contextual também permite lidar com o problema da sobrecarga de informação (Brender et al 2000; Chittaro, 2001), indicado pelo relatório da *Society for Computer Applications in Radiology* (Andrioli, 2003) como um dos problemas futuros a serem enfrentados pelo radiologista, através da adaptação da forma de visualização e uso do contexto para realce das informações mais relevantes.

5.1.1 Visualização de Imagens

O objetivo principal de um visualizador de imagens médicas é permitir a visualização adequada das informações do paciente. O modo como esse objetivo é alcançado depende das características organizacionais, do tipo de imagem e do usuário, implicando, portanto, em políticas de exibição diferenciadas em cada organização. Os casos mostrados no item 3.3 ilustram possíveis implementações para visualização de imagens médicas. Entretanto, a explicitação de regras permite a definição de outras políticas que se ajustem a diferentes ambientes e culturas encontradas em instituições de saúde.

Um visualizador de imagens médicas pode ser genérico, para atender a diferentes tipos de usuários e realizar múltiplas tarefas, ou então, ser específico, para realizar um pequeno conjunto de tarefas. Exemplos dessas abordagens são, respectivamente, o ImageJ (Figura 83)²⁴, e o MVE (Figura 84)²⁵.

A vantagem do visualizador genérico é permitir o processamento e visualização de diferentes tipos de imagem, sem necessidade de troca de programa. Por outro lado, perde-se o foco da atividade principal que está sendo realizada, pois é preciso navegar entre as diferentes ferramentas disponíveis e existe um compromisso entre a generalidade e o desempenho. O visualizador específico tem melhor desempenho, mas tem um número de tarefas limitado e não é apropriado a qualquer tipo de imagem.

²⁴ <http://rsb.info.nih.gov/ij/>

²⁵ <http://www.medical-image-processing.info/MVE/>

Figura 83 – Visualizador genérico de imagens ImageJ.

Figura 84 – Visualizador específico MVE.

Com o visualizador contextual, é possível atingir um compromisso entre as duas abordagens, com a manutenção do foco na atividade e possibilidade de diferentes formas de visualização e processamento.

5.1.2 Adaptação de acordo com o contexto

No Capítulo 2, discutiu-se o significado do contexto e os requisitos necessários para implementá-lo. A seguir, os requisitos são re-elencados e comentados sob a luz da solução adotada.

1. *Disponibilização de novos dados contextuais sem impactar no modelo utilizado.* Através do Protégé é fácil modificar o modelo e inserir novos dados, ao contrário, por exemplo, da utilização direta de arquivos XML ou do uso de bancos de dados relacionais (em uso tradicional, uma vez que o Protégé pode ser acoplado a um banco de dados). O modelo desenvolvido nesta tese é genérico e deve ser especializado para cada ambiente de uso, uma vez que os contextos disponíveis serão diferentes (por exemplo, diferentes fontes de dados). Porém, a qualidade do modelo oferecido somente pode ser determinada com o desenvolvimento de várias extensões para o visualizador, que abordem todos os pontos do modelo e estressem sua maleabilidade para suportar necessidades em ambientes e situações diferenciadas. Mesmo assim, permanece o fato de a arquitetura proposta ser válida para o desenvolvimento de aplicações contextuais para imagens médicas e dados relacionados.

2. *Descobrir dinamicamente uma nova fonte de contexto e utilizá-la.* A necessidade de descobrir dinamicamente novas fontes de contexto pode ser questionada. Na abordagem de uso de ontologia, supõe-se que todos os visualizadores contextuais e regras possuem um conhecimento comum sobre o domínio que está sendo tratado. Isto é, os elementos presentes no sistema (imagens, laudos, infra-estrutura) têm um significado claro para todos os que o utilizam e é exatamente assim que se obtém um contexto comum e pode-se realizar interações complexas. Porém, a descoberta de novas fontes de contexto pode ser implícita quando for feita a especialização de uma classe. Por exemplo, uma regra que utiliza a informação sobre imagens DICOM continua válida se for aplicada sobre uma subclasse da mesma, uma vez que todas as propriedades são mantidas.

3. *Agregar dados contextuais.* O sistema de inferência por regras com *forward-chaining* naturalmente agrega informações. Por exemplo, se é necessária a informação sobre a preferência do usuário em relação a duas formas de *display*, é possível criar uma regra que une a informação de tipo de gráfico preferido com nível de expertise do usuário e inferir a preferência de *display*.

4. *Permitir trabalhar no nível de abstração desejado.* Como o modelo deve ser conhecido por quem o utiliza, não existe um componente específico que faça explicitamente a tradução de uma informação de um nível para outro (por exemplo, converter uma coordenada cartesiana em uma coordenada polar). O que existe é a representação em um documento eletrônico, com subtipos definidos (por exemplo, *DocImagem* e *DocLaudo*). Assim, o tipo de informação é conhecido e é responsabilidade de quem o utiliza converter a informação para o formato desejado.

5. *Lidar com mudanças lentas e rápidas do contexto.* A mudança no contexto no Visualizador Contextual é tratada internamente ou externamente. Por exemplo, a mudança na quantidade de memória livre do computador pode ser feita externamente utilizando uma *thread* que continuamente averigua a memória disponível e modifica a informação na memória do contexto (*memoria.put("memória.acabando", "true")*) e notifica a mudança para as regras interessadas (*drools.modifyObject(memória)*). Internamente, uma regra pode modificar o estado do sistema (por exemplo, a regra SE processamento= 'Reconstrução3D' ENTAO Verifique memória livre).

6. *Garantir segurança no acesso ao contexto.* No caso do visualizador contextual, algumas informações do usuário são locais, não sendo sujeitas a acesso por terceiros, e outras estão contidas no modelo. Não é objetivo deste trabalho levar em consideração a segurança ao acesso dos dados, apesar de ser um fator importante para uso em produção. O modelo foi originalmente desenhado para ser utilizado por um único visualizador. Porém, como o Protégé pode ser usado como um servidor, existe a possibilidade que múltiplos visualizadores utilizem um mesmo modelo centralizando o controle e a ontologia. Neste caso, uma possibilidade é fazer um *Proxy* que intermedeie a conversação com o servidor, garantindo a segurança dos dados. Outra opção é aguardar versões futuras do Protégé que implementem o acesso seguro ao servidor.

7. *Poder definir contextos complexos.* O uso de uma dada ontologia permite descrever informações muito complexas, mas é incapaz de descrever *todas* as

informações. Tudo depende do modelo adotado, daquilo que se deseja fazer com a ontologia e da capacidade do especialista em modelar a informação. Uma possibilidade é a incorporação de diferentes ontologias. Assim, é possível unir ontologias de especialistas de diferentes áreas para obter um modelo mais robusto e confiável. A complexidade de contexto passível de ser atingida também depende do sistema de inferência utilizado. No presente caso, utilizam-se regras. Mas uma outra possibilidade é o uso de agentes inteligentes colaborativos. Neste caso, é possível atingir-se um grau mais alto de interação e troca de informações do que simplesmente através de regras e será discutido mais adiante.

8. *Fazer pesquisas complexas sobre o contexto.* Com o uso de uma linguagem similar a Lisp, como mostrado, é possível inquirir o modelo de forma sintética e eficiente. Já a manipulação direta da informação, por exemplo, determinar, a partir de um ECG, se existe uma elevação no segmento ST, é feita, no Visualizador, diretamente com código Java. Idealmente, pode-se representar o ECG de forma complexa e extrair a informação desejada somente a partir de sua representação (por exemplo, no modelo da Figura 85, pode-se obter facilmente a informação de elevação no segmento ST do ECG). O problema é que é necessário partir da representação física (por exemplo, um arquivo) para o modelo, e seria preciso um programa para reconhecer os diferentes segmentos, suportar diversas técnicas de extração desta informação, complicando a implementação do sistema. Uma maneira mais apropriada seria a definição de um conjunto de requisitos mínimos definidos na interface do ECG e relegar os detalhes específicos para subclasses dessa interface (por exemplo, uma classe ECG_ST, que somente avalia esse item em particular) ou então a um processamento externo (por exemplo, uma ferramenta *ExtraiSegmentoST*) ficando a cargo de quem utiliza essa informação saber a classe ou subclasse correta a ser utilizada.

Figura 85 – Modelo de um ECG.

9. *Definir a confiabilidade de uma informação.* É relacionada com o item anterior. Seguindo com o exemplo do ECG, como existem diferentes técnicas para segmentar o sinal, existe também um erro associado a essa segmentação, que varia com o algoritmo. Deve ser possível então a associação dessa informação com a sua representação. No modelo aqui proposto, o erro está implícito na informação. Por exemplo, se for definida uma classe *DocECG* e uma ferramenta *ExtraiSegmentoST* esta última deve ser conhecida pelo utilizador da ferramenta (quem escreve a regra), o qual sabe o erro associado a ela.

Outros temas relevantes para discussão são:

- Os médicos realizam um grande número de tarefas colaborativas (Sung et al, 2000). Como lidar com isso?
- A utilização de regras tem duas desvantagens: deve ser feita por especialistas e pode ocorrer o conflito de regras (Rosenwald e Liu, 1997);
- Como ir além das imagens médicas e criar o visualizador contextual de informações clínicas?
- Como evitar o efeito de perda de controle sobre a aplicação, ou seja, o usuário ser surpreendido por ações indesejadas (Barkhuus e Dey, 2003; Paymans et al, 2004).

A primeira questão pode ser respondida de duas formas: a primeira é que esse não é um objetivo desta tese, e a segunda é que é possível fazê-lo utilizando técnicas de

trabalhos colaborativos. Uma das abordagens possíveis é a utilização de agentes inteligentes que trabalham em benefício do usuário, procurando reconhecer uma tarefa e atuar em seu favor (Yan e Selker, 2000; Nguyen e Wobcke, 2005). Essa arquitetura é suportada pelo visualizador pois o Gerenciador de Interface é uma classe abstrata, permitindo que a mesma seja substituída por um gerenciador que, por exemplo, utilize agentes inteligentes. Nesse sentido, existe um *framework* chamado JADE (*Java Agent DEvelopment Framework*)²⁶ que abstrai a tarefa de comunicação entre agentes e oferece serviços de resolução de nomes e páginas amarelas e suporta o uso de ontologias.

O problema indicado na segunda questão pode ser minimizado com o uso de editores de regras que avaliem as interações, avisando de possíveis inconsistências e forneçam interfaces mais amigáveis. Vale notar que o DROOLS permite o uso de scripts também em Python e Groovy²⁷ não estando limitado a Java.

Quanto ao terceiro item, o visualizador contextual de informações médicas está parcialmente desenvolvido nesta tese, pois as imagens médicas são dificilmente desvinculadas das informações médicas correspondentes. As informações médicas foram modeladas genericamente, através do conceito de *RaciocínioMédico* e Documentos Eletrônicos. Com o *RaciocínioMédico*, é possível representar diferentes formas de conhecimento, envolvendo, por exemplo, os procedimentos realizados pelo paciente. Por exemplo, se o exame de CK-MB está alto e a fração de ejeção calculada no exame de cintilografia é pequena, então é um indicativo de isquemia do músculo cardíaco. Neste caso, poderia ser útil fornecer uma informação ao usuário. Porém, esta representação é demasiadamente genérica para atender às necessidades dos médicos e ser uma solução definitiva para o visualizador contextual de informações médicas. Uma alternativa é a utilização de um modelo mais específico, que mantenha o foco em uma especialidade, por exemplo, em cardiologia.

Um problema com o uso de regras é que a interação final obtida pode não ser necessariamente aquela esperada pelo usuário. Neste caso, a solução é sempre oferecer (Bardram, 2004) o auxílio de forma discreta, permitindo que o usuário rejeite a alternativa. Também deve estar claro para o usuário qual ação será feita, de modo que ele não seja surpreendido, por uma ação diferente da esperada.

²⁶ <http://jade.tilab.com/>

²⁷ <http://drools.org/>

5.2 Vantagens e desvantagens

No caso do visualizador contextual proposto, as vantagens são claras:

- Uso de informação contextual para auxiliar o usuário em sua interação com o Visualizador;
- Acesso a sistemas distribuídos e heterogêneos e integração com diversos sistemas de informação existentes;
- Extensibilidade de ferramentas e forma de visualização de informação;
- Possibilidade de trabalhos futuros na área com relevância de pesquisa;
- Contribuição para o desenvolvimento do Prontuário Eletrônico do Paciente.

As principais limitações do Visualizador Contextual implementado são a necessidade de um especialista para elaborar as regras, uma vez que estão envolvidos o conhecimento do modelo e programação Java, e a necessidade de manutenção das regras a medida que seu número aumenta.

5.3 Trabalhos futuros

O Visualizador Contextual implementado resolve algumas questões complexas de visualização de informações médicas e imagens, e abre perspectivas para outros trabalhos e para resolver questões já levantadas.

Uma possibilidade para melhorar o visualizador contextual é o uso de agentes inteligentes como parte do sistema de inferência. Os agentes inteligentes permitem a realização de trabalhos colaborativos, além de serem específicos para uma atividade, por exemplo, preencher formulários de dados. Também existe uma infra-estrutura de software pronta em Java (o *framework* JADE) que permite o rápido desenvolvimento de novos agentes.

Outra melhoria é o refinamento do modelo do contexto. O modelo depende da complexidade dos casos de uso utilizados, e quanto maior o número de casos e experiências práticas para validar o modelo, melhor o seu refinamento. Uma vez que arquitetura básica já está bem desenvolvida, é necessário criar casos, de preferência reais, que tentem atender às necessidades dos médicos em diferentes situações e ambientes.

É necessário desenvolver novas ferramentas de processamento e visualização dos dados. Com uma quantidade maior de ferramentas, pode-se testar interações mais complexas, melhorando a capacidade de resposta do sistema e enriquecendo a interação com o usuário.

Um subproduto da tese foi a base de imagens BIRAM, que demonstra ser uma forma interessante de armazenar casos, permitindo o uso de algoritmos que levam em conta informações da modalidade da imagem e parte do corpo, além de suportar laudos. Um trabalho futuro poderá ser o aprimoramento do BIRAM para melhorar o seu tempo de resposta e incorporar o uso de regiões de interesse e, posteriormente, o uso do DICOM *Structured Reporting* para os laudos.

Um objetivo em médio prazo é a transformação do Visualizador Contextual em um projeto de código fonte aberto, integrável com o MiniWEBPACS (Figueiredo et al, 1999; Urtiga et al, 2004) e disponível para uso livre, ampliando a solução de PACS e permitindo a melhoria do Visualizador e teste do sistema em ambientes diferenciados.

Conclusões

“What you get by achieving your goals is not as important as what you become by achieving your goals.”

Zig Ziglar

Este trabalho apresentou um modelo e um sistema para visualização contextual de imagens médicas como parte do sistema PACS-HC, mantendo, porém, independência deste sistema, através da abstração na representação do contexto e do acesso a fontes de dados.

O visualizador proposto tem a vantagem de separar o conhecimento (modelo do contexto) e as ações (regras) da implementação, permitindo uma grande flexibilidade na combinação dos dois para criar novas formas de interação com o usuário e auxílio ao profissional de saúde.

A arquitetura proposta, comprovada pela implementação, permitiu a rápida inclusão de diversas fontes heterogêneas de informação (DICOM, CIAS, HIS, bases de imagens e referências da literatura) e sua utilização contextual em benefício do usuário. Além disso, demonstrou que existe um potencial para assistência ao médico através do uso de informações clínicas como contexto, acenando com a possibilidade de desenvolvimento de um visualizador contextual de informações médicas.

Como contribuições adicionais podemos citar:

- Desenvolvimento e implementação de uma arquitetura que suporta a utilização de fontes de informação heterogêneas e distribuídas, inclusive a busca de imagens similares em bancos de casos;

- Possibilidade de extensão de ferramentas, visualizadores e fontes de dados. A extensibilidade permite a evolução ao longo do tempo, estendendo a vida útil do programa e permitindo grande flexibilidade, especialmente para uso dentro de instituições de pesquisa que necessitam testar e incorporar novas técnicas e ferramentas;
- Implementação das interfaces do CIAS e percepção das limitações de interoperabilidade permitidas pela interface e proposta de correções através da união com DICOM SR;
- Desenvolvimento de uma base de imagens e algoritmos (BIRAM), a qual apresenta características a serem exploradas futuramente;
- O desenvolvimento de um servidor DICOM com código fonte aberto e parte de um projeto de PACS para clínicas de pequeno porte (MiniWEBPACS);

Em conclusão, demonstrou-se, com o sistema proposto, que é viável a integração de sistemas heterogêneos e distribuídos, com uso inteligente da informação disponível, para o auxílio aos profissionais de Saúde, sendo este um importante passo em direção ao desenvolvimento de um eficiente Prontuário Eletrônico do Paciente, objeto de estudo de vários grupos de pesquisa.

Apêndice A – Conceitos Básicos de OO

“Feliz aquele que transfere o que sabe e aprende o que ensina”

Cora Coralina

A.1 Orientação a Objeto

O enfoque tradicional de modelagem para a construção de sistemas de informação baseia-se na compreensão desse sistema como um conjunto de programas que, por sua vez, executam processos sobre dados. O enfoque de modelagem por objetos vê o mundo como uma coletânea de objetos que interagem entre si, apresentam características próprias as quais são representadas pelos seus atributos (dados) e operações (processos). A relação entre as duas formas de modelagem é mostrada na Figura 86 (Furlan, 1998).

Figura 86 – Enfoque baseado em sistema versus enfoque baseado em objeto (Furlan, 1998).

O ponto de partida para compreensão da orientação a objeto é a definição de Objetos, Classes, Instâncias e Mensagens.

Os **Objetos** representam entidades que possuem diferentes estados e comportamentos. Eles são, nesse ponto, análogos a objetos da vida real. Podemos citar como exemplos de objetos reais: um cachorro, uma mesa, um televisor, uma bicicleta. Todos esses objetos possuem um comportamento e um estado. Assim, um cachorro possui um estado: nome, cor, raça, se tem fome ou não; e também possui um comportamento: latir, balançar o rabo, comer. A bicicleta tem um estado (marcha atual, cadência de pedaladas, duas rodas, número de marchas) e um comportamento (freiar, acelerar, mudança de marcha). No caso da Orientação a Objetos (OO) o estado é representado pelas *variáveis* de um objeto e seu comportamento pelos seus *métodos*. Porém, a representação da OO não é limitada ao modelo de objetos reais. Também é possível modelar elementos abstratos, como um evento da interface gráfica, por exemplo, representando a ação de um usuário pressionando um botão do mouse ou uma tecla do teclado²⁸. Uma forma comum de representação de um objeto é mostrada na Figura 87.

Figura 87 – Representação de um objeto.

As variáveis de um objeto são chamadas *variáveis de uma instância*, pois contém o estado de um objeto em particular (por exemplo, a bicicleta da minha irmã e não qualquer bicicleta) e pelo fato de um objeto em particular ser chamado na terminologia de OO de **Instância**.

A representação mostrada na Figura 87 ainda tem a função de mostrar o conceito de **encapsulamento**, no qual os elementos do núcleo do objeto são ocultos dos outros objetos no programa. O ocultamento das variáveis internas tem duas justificativas: a possibilidade de alteração da representação interna do objeto sem que haja alteração dos métodos, e, portanto, sem necessidade de alterar a forma de interação com outros

²⁸ adaptado de <http://java.sun.com/docs/books/tutorial/java/concepts/object.html>

objetos; a segunda razão é evitar que outros objetos alterem inadvertidamente o estado interno do objeto, causando comportamentos imprevistos. Porém a representação com encapsulamento é uma visão ideal de um objeto e, por razões práticas, é possível expor determinadas variáveis e ocultar alguns de seus métodos.

Normalmente, um objeto é um componente em um programa maior e que contém vários objetos. Através da interação entre os objetos é possível obter funcionalidades de mais alto nível e comportamento mais complexo. A comunicação entre objetos é realizada através de **Mensagens**, como representado na Figura 88.

Figura 88 – Troca de mensagem entre objetos.

A troca de mensagem equivale à chamada de um método de um objeto a partir de outro objeto passando, se necessário, parâmetros. Seguindo com o exemplo da bicicleta poderíamos ter uma chamada de um objeto Maria para o objeto Minha_Bicicleta passando a mensagem 'mudarMarcha' com o parâmetro do número da marcha para o qual se deseja mudar, representado por:

Minha_Bicicleta.mudarMarcha(numero)

A utilização de mensagens fornece dois benefícios principais:

- O comportamento dos objetos é expresso através de seus métodos, de forma que a passagem de mensagens suporta toda interação possível entre dois objetos;
- Objetos não necessitam estar no mesmo processo ou mesmo na mesma máquina para enviar e receber mensagens entre si.

Finalmente, temos o conceito de **Classe**. No exemplo das bicicletas, vimos que estávamos tratando de uma bicicleta em específico, porém, existem milhares de bicicletas no mundo, caracterizando uma classe de objetos do tipo Bicicleta, que compartilham características em comum. Assim, uma Classe é um projeto, ou um protótipo, que define as variáveis e métodos comuns a todos os objetos de um determinado tipo (Figura 89).

Figura 89 – Classes e instâncias.

Além das variáveis de instância é possível haverem variáveis de classe, as quais possuem uma informação que é compartilhada entre todas as instâncias da classe. Por exemplo, se todas as bicicletas possuísem o mesmo numero de marchas e utilizássemos uma variável de instância, teríamos uma cópia da variável para cada instância. Se definirmos uma variável de classe teremos apenas uma cópia compartilhada entre todas as instâncias.

Dessas definições derivamos outros conceitos importantes em Orientação a Objetos, as quais são: Herança. Polimorfismo e Interface.

Sistemas orientados a objeto permitem que classes sejam definidas a partir de outras classes. Por exemplo, *mountain-bikes* e bicicletas de dois assentos são diferentes tipos de bicicletas, sendo chamados de **subclasses** de Bicicleta. De forma similar, a Bicicleta é uma **superclasse** de *mountain-bike*. Cada subclasse **herda** as variáveis e métodos de sua superclasse, podendo adicionar novos métodos e variáveis próprios, especializando a superclasse. Subclasses também podem superpor métodos da superclasse e prover implementações especializadas para esses métodos, sendo essa característica chamada de **polimorfismo** (do grego ‘muitas formas’, pois o mesmo método tem diferentes comportamentos). O conceito de herança é representado na Figura 90.

Figura 90 – Herança.

Em algumas linguagens de programação pode existir a herança múltipla, na qual a classe filho herda os atributos e métodos de duas classes. A herança múltipla pode ocasionar problemas se houverem métodos com mesma assinatura entre as classes sendo necessário determinar qual das implementações será utilizada.

A herança pode ir além de um nível de especialização, permitindo a elaboração de uma hierarquia de classes. As vantagens da herança são:

- Reutilização de código, uma vez que os métodos são acessíveis na subclasse;
- Podem ser utilizadas classes abstratas que definem comportamentos genéricos.

Uma classe abstrata define e pode implementar parcialmente um comportamento, deixando partes indefinidas e sem implementação, que serão completadas por outros programadores de acordo com a necessidade deles.

Uma **Interface** é um contrato na forma de um conjunto de métodos e declarações de constantes. Quando uma classe implementa uma interface, ela se compromete a implementar todos os métodos declarados naquela interface. A interface é uma forma de simular a herança múltipla, pois não existe conflito no uso de diferentes interfaces.

A.2 Diagramas UML

A orientação a objeto tornou necessário o desenvolvimento de metodologias para análise e desenvolvimento de software. As metodologias mais famosas desenvolvidas foram OMT (*Object Modelling Technique*) de Rumbaugh, OOD (*Object*

Oriented Design) de Booch e OOSE (*Object-Oriented Software Engineering*) de Jacobson. Uma vez que os métodos de Booch e OMT estavam crescendo independentemente e sendo reconhecidos pela comunidade usuária como métodos de classe mundial, seus autores juntaram forças através da Rational Corporation para forjar uma unificação completa de seus trabalhos. Em outubro de 1995 lançaram um rascunho do Método Unificado na versão 0.8, sendo este o primeiro resultado concreto de seus esforços que mais tarde, com a colaboração de Jacobson, se tornaria a UML (*Unified Modeling Language*).

A UML pode ser utilizada para:

- Mostrar as fronteiras de um sistema e suas funções principais utilizando atores e casos de uso;
- Ilustrar a realização de casos de uso com diagramas de interação;
- Representar uma estrutura estática de um sistema utilizando diagramas de classe;
- Modelar o comportamento de objetos com diagramas de transição de estado;
- Revelar a arquitetura de implementação física com diagramas de componente e implantação;
- Estender a sua funcionalidade através de estereótipos.

Aqui serão mostrados somente os casos de uso, diagramas de interação, diagramas de classe e de transição de estado.

A.2.1 Diagrama de classe

Uma classe é representada como um retângulo sólido como mostrado na Figura 91. O primeiro compartimento é para o nome da classe e é obrigatório. O segundo e o terceiro são opcionais e contém respectivamente, os atributos e as operações definidas para a classe. Os símbolos ao lado dos atributos e das operações indicam sua visibilidade (visibilidade pública (vazio), protegida (chave) ou privada (cadeado)).

A sintaxe completa para descrição de um atributo é:

nome_atributo:tipo_atributo=valor_inicial

Por exemplo, threshold:int=50.

Enquanto que a representação de um método é dada por:

nome_metodo (argumentos) : tipo_retorno

Por exemplo, processarImagem (imagem, threshold) : Imagem.

Figura 91 - Representação de uma Classe segundo a UML (Furlan, 1998).

As classes são relacionadas entre si através de associações. As associações representam as informações de:

- Multiplicidade;
- Ordenação;
- Qualificador;
- Agregação / composição;
- Navegabilidade;
- Especificador de interface;
- Generalização.

A forma como essas informações são representadas em UML são mostradas na Figura 92.

Figura 92 – Papéis em associação (Furlan, 1998).

A seguir são apresentadas em mais detalhes cada uma das representações e seus significados.

Multiplicidade: Mostra a relação de cardinalidade entre duas classes correspondendo às noções de obrigatório, opcional, um para muitos, muitos para

muitos, de maneira similar ao enfoque de entidade/relacionamento do modelo relacional (para bancos de dados). Na Figura 93 vemos um exemplo de multiplicidade, que pode ser interpretado como ‘um cliente faz 0 ou N pedidos’ se lido a partir da classe Cliente ou ‘um pedido tem necessariamente um cliente’ se lido a partir da classe Pedido.

Figura 93 - Exemplo de multiplicidade um para muitos.

São exemplos de multiplicidade:

0..1 → Zero ou um

* → Maior ou igual a zero

1..15 → de 1 a 15, inclusive

Ordenação: Especifica se os elementos relacionados de multiplicidade maior que 1 devem ou não ser ordenados.

Qualificador: Representa uma associação indireta entre duas classes, normalmente através de um elemento do tipo matriz, mapa ou dicionário associativo (em Java poderia ser um Hashtable ou um Map). No interior da caixa que representa o qualificador são colocadas as chaves necessárias para acessar o elemento. No exemplo da Figura 94, para acessar um item de pedido é necessário identificar o produto como argumento.

Figura 94 – Exemplo de associação qualificada (Furlan, 1998).

Agregação: Uma agregação é uma forma especial de associação utilizada para mostrar que um tipo de objeto é composto, pelo menos em parte, de outro em uma relação de todo/parte. Indica semanticamente que o objeto parte é um atributo do objeto todo, e que a vida do primeiro é dependente do segundo, ou seja, a parte só é criada quando o todo for criado.

Figura 95 – Relação de agregação.

Composição: A composição é similar a agregação, porém, a relação entre o todo e a parte é feita **por valor**, em oposição à relação **por referência** da agregação. A relação por valor quer dizer que o objeto Todo declara uma instância real do objeto Parte dentro do seu próprio corpo, fazendo com que o objeto esteja contido no Todo. Uma relação de composição é mostrada na Figura 96.

Figura 96 - Relação de composição.

Em Java os tipos primitivos são passados por valor, enquanto que as matrizes e objetos são passados por referência.

Navegabilidade: Indica se uma instância de uma classe possui referência para instâncias de outra classe, ou seja, se uma pode acessar a outra. Se nada for indicado, as associações têm navegação bidirecional. No caso demonstrado na Figura 97, o objeto Alvo é referenciado pelo objeto Fonte, ou seja, o objeto Fonte tem acesso ao objeto Alvo. Porém, o inverso não é verdade. O objeto Alvo não pode navegar para o objeto Fonte, pois não possui referência.

Figura 97 – Navegabilidade.

Generalização: Indica a relação de herança. No exemplo da Figura 98 a superclasse é mais genérica do que as subclasses. É uma representação do conceito mostrado no item 2.1 (Figura 90).

Figura 98 – Especialização / generalização.

A.2.2 Diagrama de caso de uso

São diagramas que auxiliam a compreender melhor as exigências para o desenvolvimento de um projeto. Os casos de uso não detalham como o caso será implementado nem qual é o seu funcionamento interno. Os principais propósitos dos casos de uso são:

- Descrever os requisitos funcionais do sistema de forma consensual entre os usuários e desenvolvedores de sistemas;
- Fornecer uma descrição consistente e clara sobre as responsabilidades que devem ser cumpridas pelo sistema, além de formar a base para o diagrama de classes;
- Oferecer as possíveis situações do mundo real para o teste do sistema.

Na Figura 99 são mostrados os principais componentes da representação de um caso de uso: o Ator, que inicia a interação, uma seta indicando com que caso de uso a interação é realizada e os diferentes casos de uso (elipses), os quais possuem cada um sua denominação.

Figura 99 – Diagrama de casos de uso.

Um Ator é um agente que interage com o sistema, um tipo de usuário ou categoria com papel definido, podendo representar seres humanos, máquinas,

dispositivos ou outros sistemas. Exemplos de atores são: cliente usuário, gerente, computador e impressora.

Os casos de uso representam as formas como os atores interagem com o sistema. Na Figura 100 é mostrado um exemplo de caso de uso para um médico imagenologista acessando um sistema de visualização e processamento.

Figura 100 – Exemplo de Caso de Uso.

A.2.3 Diagrama de seqüência

Tem como função mostrar as interações entre instâncias (ou objetos) com seqüência temporal claramente definida. Cada objeto é desenhado como um retângulo no topo da página com uma linha vertical tracejada projetada para baixo, a qual é chamada linha de vida do objeto (Figura 101).

Figura 101 – Diagrama de seqüência.

As mensagens trocadas entre os objetos representam chamadas a métodos. O diagrama de sequência expõe o aspecto do modelo que enfatiza o comportamento dos objetos em um sistema, incluindo as suas operações, interações, colaborações e histórias de estado em forma sequencial de troca de mensagens e representação explícita de ativação de operações.

A.3 Design Patterns

Os *Design Patterns*, ou Padrões de Projeto, são padrões para desenvolvimento de programas orientados a objeto que foram desenvolvidos e testados por projetistas experientes. Os padrões de projeto, se bem utilizados, garantem a reutilização de código e flexibilidade, diminuindo a necessidade de reprojeto à medida que surgem novos requisitos para o programa.

Gamma et al (2000) introduziram o conceito de *Design Patterns* e forneceram um catálogo dos mesmos, os quais são apresentados de forma resumida na Tabela 3.

Tabela 3 – Resumo dos padrões de projeto e sua classificação (Extraído de Gamma, 2000).

Escopo	Classe	Propósito		
		De Criação	Estrutural	Comportamental
		Factory Method	Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Aqui serão apresentados somente os seguintes *Design Patterns*: Singleton, Abstract Factory, Factory Method, Mediator, Observer, Strategy e Visitor.

Singleton: tem como objetivo garantir que uma classe tenha somente uma instância e fornecer um ponto global de acesso para a mesma.

Figura 102 – Estrutura de um *Singleton*.

As vantagens do padrão *Singleton* são:

- Acesso controlado a instância única: só se obtêm acesso ao *Singleton* através do método `getInstance()`, fornecendo controle total sobre o acesso;
- Espaço de nomes reduzido: evita a poluição do espaço de nomes com variáveis globais;
- Permite um refinamento de operações e da representação: A classe *Singleton* pode ter subclasses e é fácil configurar um programa com uma instância desta classe estendida;
- Permite um número variável de instâncias: o padrão torna fácil mudar de idéia, permitindo mais de uma instância da classe *Singleton*. Somente a operação que permite acesso à instância de *Singleton* precisa ser mudada.

Abstract Factory: tem como objetivo fornecer uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas. No exemplo da Figura 103 isso é realizado especificando-se as relações de alto nível (*WidgetFactory* cria janelas (*Window*) e barras de rolagem (*Scrollbar*)) sem especificar qual a instância específica a ser utilizada. Isso é feito pelas subclasses de *WidgetFactory*, *Window* e *Scrollbar*.

Figura 103 – Exemplo de uma *Abstract Factory*.

O *AbstractFactory* é utilizado quando um sistema deve ser independente de como seus produtos são criados, compostos ou representados.

Factory Method: tem como objetivo definir uma interface para criar um objeto, mas deixar para as subclasses decidirem que classe instanciar. No exemplo da Figura 104 pode-se ver o funcionamento deste padrão: uma classe abstrata é definida com métodos genéricos (*Application*) e possui referência a elementos a serem criados (*Document*). Porém, somente uma subclasse específica do elemento deve ser criada (*MyDocument*) e quem possui o conhecimento para isso é um subclasse de *Application* (*MyApplication*).

Figura 104 – Exemplo do padrão *Factory Method*.

O padrão *Factory Method* deve ser utilizado quando:

- Uma classe não pode antecipar a classe de objetos que deve criar;
- Uma classe quer que suas subclasses especifiquem os objetos que criam;
- Classes delegam responsabilidade para uma dentre várias subclasses auxiliares,

e se deseja localizar o conhecimento de qual subclasse auxiliar deve receber a delegação.

Mediator: tem como objetivo definir um objeto que encapsula a forma como um conjunto de objetos interage. O *Mediator* promove o acoplamento fraco ao evitar que os objetos se refiram uns aos outros explicitamente e permite variar as suas interações independentemente. Na Figura 105 é mostrado um exemplo do padrão *Mediator*. O exemplo consiste em uma caixa de diálogo com diversos componentes interativos como botões e listas. O *Director* (*DialogDirector*) possui conhecimento dos diversos componentes (subclasses de *Widget*) e controla a forma como eles interagem (sendo notificado pelo método *widgetChanged* da classe *Widget*, o qual deve ser invocado por cada subclasse de *Widget*). Com o uso do padrão *Mediator* todo o conhecimento das interações fica confinado a classe *Director* a qual pode ser substituída para modificar o comportamento do conjunto.

Figura 105 – Exemplo do padrão *Mediator*.

Observer: tem como objetivo definir uma dependência um-para-muitos entre objetos, de maneira que, quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente. O assunto (*Subject*) recebe vários observadores (*Observer*) e os notifica quando ocorre alguma mudança no seu estado (através de *notify()*). Uma vez notificados, os observadores obtêm o novo estado (*getState()*) do *Observer* e sincronizam-se com ele (Figura 106).

Figura 106 – Estrutura do padrão *Observer*.

Strategy: tem como objetivo definir uma família de algoritmos, encapsular cada uma delas e torná-las intercambiáveis. *Strategy* permite que o algoritmo varie independentemente dos clientes que o utilizam. Utiliza a herança e o polimorfismo para obter essa característica como mostrado na Figura 107.

Figura 107 – Exemplo do padrão *Strategy*.

Visitor: tem como objetivo representar uma operação a ser executada nos elementos de uma estrutura de objetos. O padrão *Visitor* permite definir uma nova operação sem mudar as classes dos elementos sobre os quais opera. No exemplo da Figura 108 pode-se ver o funcionamento deste padrão: um programa (*Program*) possui uma árvore de nós de diversos tipos (subclasses de *Node*) e deve realizar diferentes processamentos em cada um desses nós. Para isso, ele utiliza subclasses de *NodeVisitor* que realizam diferentes operações sobre os nós. Os nós são percorridos passando-se o visitante (método *accept()*) por todos os nós. Cada nó chama o método correspondente ao seu tipo no objeto visitante, que, por sua vez, realiza o processamento.

Figura 108 – Exemplo do padrão *Visitor*.

Apêndice B – Java

*“Remember, even if you don't love
coffee, coffee loves you”*

Alex Gottschalk

B.1 Introdução

O Java é uma linguagem de programação que evoluiu da linguagem chamada Oak. A linguagem Oak foi desenvolvida no início dos anos 90 pela Sun Microsystems como uma linguagem independente de plataforma com o objetivo de permitir a intercomunicação de ferramentas de entretenimento, como consoles de vídeo game e videocassetes²⁹. Oak foi primeiramente indicada para ser utilizada em aparelhos eletrônicos para televisão feitos para prover serviços de vídeo sob demanda. Mas nessa época o negócio de dispositivos eletrônicos diminuiu de força, ao mesmo tempo em que a Internet prosperava. Quando os desenvolvedores do Oak perceberam essa tendência, eles mudaram seu foco para a Internet e para o desenvolvimento de um *browser* habilitado para utilizar a linguagem Oak, chamado de *WebRunner*. O nome da linguagem foi modificado para Java e o *WebRunner* tornou-se o *browser* HotJava. O desenvolvimento da Internet estimulou a evolução do Java e sua adoção por muitos programadores e vendedores de software.

As seguintes características diferenciam o Java das demais linguagens de programação:

- **Independência de plataforma:** Compiladores Java não produzem código nativo para uma plataforma particular. Ao invés disso, eles produzem instruções ‘byte code’, para serem interpretadas por uma Java Virtual Machine (JVM). Para fazer o código Java funcionar em uma determinada plataforma, basta escrever um interpretador de bytecodes que simule uma JVM. E por isso, o mesmo bytecode compilado pode ser executado sem modificações em qualquer plataforma que suporte Java. Essa característica é diagramada na Figura 109.

²⁹ Adaptado de <http://www.engin.umd.umich.edu/CIS/course.des/cis400/Java/java.html>

Figura 109 – Funcionamento da linguagem Java (Extraído de <http://java.sun.com/docs/books/tutorial/getStarted/cupojava/win32.html>).

- **Orientação a objeto:** o Java é puramente orientado a objeto, pois todas as classes são descendentes de uma única classe raiz (a classe Object. A única exceção são os tipos primitivos int, double, float, long e short que não são objetos).

- **Uma biblioteca padrão rica:** uma das características mais atraentes do Java é sua biblioteca padrão. O ambiente do Java tem centenas de classes que dão suporte a manipulação de strings, uso de *threads*, gerenciamento de exceções, geração de números aleatórios, leitura e escrita de dados, comunicação de rede, desenvolvimento de interfaces gráficas, e execução remota.

- **Uma sintaxe bem elaborada:** o uso de *stack traces*, herança única, gerenciamento automático de memória (através do *Garbage Collector*) e a ausência de ponteiros diminuem a chance de um programador cometer erros durante a elaboração de um programa, tornando-o mais confiável e estável.

Um dos problemas da linguagem Java é derivado de uma das suas qualidades. O gerenciamento de memória não permite que sejam feitos programas otimizados para obter o melhor desempenho do equipamento. Isso é particularmente verdadeiro quando é necessário realizar processamentos intensivos e que utilizam grandes quantidades de memória, como, por exemplo, o processamento de imagens médicas tridimensionais ou quadrimensionais. A solução mais adequada para esta limitação é a utilização de grandes quantidades de memória RAM (512MBytes ou maior).

Outra deficiência, também em termos de performance, é que, pelo fato do Java ser independente de plataforma, é difícil acessar o hardware específico da máquina (por exemplo, placas gráficas aceleradoras de desempenho).

B.2 Java Advanced Imaging (JAI)

É uma API (*Application Programming Interface*) para processamento de imagens. Inclui algumas características como avaliação tardia, imagens de multi-resolução, gerenciamento de meta-informação e imageamento através da rede. As principais características são³⁰:

- *Tiling* – permite que os usuários processem somente uma seção de uma imagem por vez, diminuindo os requisitos de largura de banda e processamento ao mesmo tempo mantendo a possibilidade de mudar rapidamente de imagem para imagem.
- Processamento de imagem independente de resolução – oferece a habilidade de descrever uma imagem da maneira que ela deveria aparecer, independentemente de sua resolução. Isto também torna possível definir um *pipeline* de operações, o qual pode ter suas operações alteradas individualmente e re-executadas.
- Modo de execução adiado – melhora a performance geral suspendendo o trabalho desnecessário sobre uma imagem, exceto nos dados da região requisitada.
- Imageamento via rede – simplifica o processamento de imagens distribuídas para programas que tem capacidade de se adaptar ao potencial do cliente. Por exemplo, um cliente com baixa capacidade de processamento (*thin client*) pode passar a cadeia de processamento para o servidor e retornar somente o resultado para que seja exibido pelo cliente.

Além dessas características o JAI pode ser integrado com o Java2D da edição padrão do Java.

B.3 Java3D

O Java 3D API é um conjunto de classes Java que servem como interface para um sistema sofisticado de renderização tridimensional. O programador trabalha em alto nível sobre métodos para criar e manipular objetos 3D.

O Java 3D é baseado em grafos como mostrado na Figura 110.

³⁰Referência: <http://java.sun.com/products/java-media/jai/collateral/datasheet.html>

Figura 110 – Organização dos grafos de cenas do Java 3D.

De maneira sucinta, um grafo de cena é constituído de:

- *VirtualUniverse* – Contém todos os elementos de cena;
- *Locale* – Determina a posição a partir de qual os outros objetos se referenciam;
- *BranchGroup* – Nó que contém grupos de objetos;
- *TransformGroup* – Contem a posição do objeto relativa ao Locale, e os objetos propriamente ditos;
- *BehaviorNode* – Contém o comportamento dos objetos ligados ao TransformGroup. Pode ser um comportamento como rotação, translação, mudança de forma (*morphing*), etc;
- *Shape3D* – É o objeto em si. Contém Appearance e Geometry que determinam as características do objeto;
- *Appearance* – Determina se o objeto é colorido, transparente, com textura ou sem, etc;
- *Geometry* – Determina o formato do objeto. Contém seqüências de vértices 3D;
- *ViewPlatform* – Converte os objetos em uma imagem 2D que possa ser mostrada no dispositivo de saída (por exemplo, a tela do computador).

Referências Bibliográficas

1. ACR-NEMA, *Digital Imaging and Communications in Medicine. Part 3: Information Object Definitions*, 2000.
2. ADELHARD, K.; HOLZEL, D., UBERLA, K. *Design elements for a computerized patient record*, Methods. Inf. Med. 38 187–193, 1999.
3. ADOMAVICIUS, G.; SANKARANARAYANAN, R.; SEN, S.; TUZHILIN A. *Incorporating Contextual Information in Recommender Systems Using a Multidimensional Approach*. ACM Transactions on Information Systems, a ser publicado, 2004.
4. ANDRIOLI, K.P., *Addressing the Coming Radiology Crisis: The Society for Computer Applications in Radiology Transforming the Radiological Interpretation Process (TRIP™) Initiative*, Novembro 2003. Acessível em <<http://www.scarnet.org/pdf/TRIPwhitepaper1103.pdf>>.
5. BARKHUUS, L., DEY, A.K. *Is context-aware computing taking control away from the user? Three levels of interactivity examined*. UBICOMP 2003, 5th International Symposium on Ubiquitous Computing, pp. 149-156, 2003
6. BARDRAM J.E., *Applications of Context-Aware Computing in Hospital Work – Examples and Design Principles*, SAC 2004 – Symposium on Applied Computing, Nicosia, 2004.
7. BARDRAM, J.E.; CHRISTENSEN, H.B., *Middleware for Pervasive Healthcare – A White Paper*, Advanced Topic Workshop Middleware for Mobile Computing, Heidelberg, 2001.
8. BARNDEN, J.A.; LEE, M.G. *An implemented Context System that Combines Belief Reasoning, Metaphor-Based Reasoning and Uncertainty Handling*. Proceedings of the Second International and Interdisciplinary Conference on Modelling and Using Context (Context'99), 1999.

9. BERG M. *Implementing information systems in health care organizations: myths and challenges*. International Journal of Medical Informatics, 64: 143-156, 2001.
10. BICK, U.; LENZEN, H., *PACS: the silent revolution*, European Radiology, Vol. 9, pp. 1152-1160, 1999.
11. BIDGOOD W.D.; BRAY B.; BROWN N.; MORI, A. R.; SPACKMAN K.A.; GOLICHOWSKI A.; Jones, R.H.; Korman L; Dove, B.; Hildebrand L.; Berg, M., *Image Acquisition Context: Procedure Description Attributes for Clinically Relevant Indexing and Selective Retrieval of Biomedical Images*, Journal of the American Medical Informatics Association, V. 6, N. 1, 1999.
12. BINKHUYSEN, F. H. B., *Required Functionality of PACS from clinical point of view*, International Journal of Biomedical Computing, 30, pp.187-191, 1992.
13. BISDIKIAN C.; CHRISTENSEN J.; DAVIS II J.; EBLING M.R.; HUNT G.; JEROME W.; LEI H.; MAES S.; SOW D., *Enabling location-based applications* Proceedings of the 1st international workshop on Mobile commerce, pp 38-42, 2001.
14. BRENDER J.; NOHR C.; McNAIR, P. *Researchs and priorities in health informatics*, International Journal of Medical Informatics, vol.58-59, pp. 257-289, 2000.
15. BROWN, P.J., *The Stick-e Document: a Framework for Creating Context-aware Applications*, Proceedings of EP'96, Palo Alto, pp 259-272, 1996.
16. BUENO, J. M., *Suporte à recuperação de imagens médicas baseadas em conteúdo através de Histogramas Métricos*, Tese Doutorado, São Carlos, 2002.
17. CAPRA, L.; EMMERICH W.; MASCOLO, C, *Reflective Middleware Solutions for context-Aware Application*, in Proceedings of REFLECTION 2001. The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns, Kyoto, 2001.

18. CCOW - *Health Level Seven Context Management Standard*. Documento eletrônico. Disponível em: <<http://www.hl7.org/Special/committees/visual/visual.cfm>>. Acesso em: Janeiro 2004.
19. CCOW - *Health Level Seven Context Management Standard: Component Technology Mapping: Web/HTTP, Draft Version 1.5*, January 19, 2003. Disponível em <<http://www.hl7.org/Special/committees/visual/drafts/drafts.htm>>.
20. CHEN G.; KOTZ D. *A Survey of Context-Aware Mobile Computing Research*, Dartmouth Computer Science Technical Report TR2000-381, 2000.
21. CHEN H.; FININ T.; JOSHI A. *An Ontology for Context-Aware Pervasive Computing Environments*, Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, 3, Vol. 18, pp. 197-207, 2004.
22. CHITTARO L. *Information visualization and its application to medicine*. Artificial Intelligence in Medicine, 22, pp 81-88, 2001.
23. CIMINO J.J., *Distributed cognition and knowledge-based controlled medical terminologies*, Artificial Intelligence in Medicine, 12, pp 153-168, 1998.
24. CLUNIE, D.A., *DICOM Structured Reporting*, PixelMed Publishing, Bangor, Pennsylvania, 2000.
25. CRANFIELD S.; PURVIS M. *Uml as an ontology modelling language*. In Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99), 1999.
26. CRAPO A.W.; ARAGONES A.V.; PRICE, J.E.; VARMA A. *Using Ontology to Drive an Adaptive Learning Interface*, Technical Report, General Electric Company, 2003. Disponível em < <http://www.crd.ge.com/cooltechnologies/pdf/2003grc247.pdf> >.
27. DCM4CHE. *Sítio que disponibiliza a documentação, códigos-fonte e binários do produto dcm4che*. 2004. Disponível em <<http://sourceforge.net/projects/dcm4che/>>. Acesso em: 12 nov 2004.

28. DeLOACK S. A., *Multiagent Systems Engineering: A methodology And Language for Designing Agent Systems*, Proceedings of Agent Oriented Information Systems '99, pp. 45-57, 1999.
29. DENT L; BOTICARIO J; McDERMOTT J; MITCHELL T; ZABOWSKI D. A *Personal Learning Apprentice*, Ed. William Swartout, Proceedings of the 10th National Conference on Artificial Intelligence - AAAI-92, pp. 96-103, 1992.
30. DEVEDZIC, V. *A survey of modern knowledge modeling techniques*. Expert Systems with Applications 17, pp 275-294, 1999.
31. DEY, A.K., *Providing Architectural Support for Building Context-Aware Applications*. PhD Thesis, Georgia Institute of Technology, November, 2000.
32. DEY A.K.; MANKOFF J.; ABOWD G.; Carter S. *Distribution of ambiguous context in aware environments*, UIST 2000 – User Interface Software & Technology Symposium, 2002.
33. DEY, A.K.; SALBER D.; ABOWD G. D. *A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications*. Human-Computer Interaction, 16(2-4), pp. 97–166, 2001.
34. ENCARNACÃO, LM; STOEV, SL. *An Application-Independent User Support System Exploiting Action-Sequence Based User Modelling*. Proceedings of the seventh international conference on User modeling, pp. 245 – 254, 1999.
35. FLEMING, M; COHEN, R., *User Modeling in the Design of Interactive Interface Agents*, 7-th International Conference on User Modeling, Banff, Canada, 1999.
36. FURLAN, J.D. *Modelagem de Objetos através da UML: análise e desenho orientados a objeto*. 1ª ed., Makron Books, pp 329, 1998.
37. FURUIE, SS; REBELO, MS; GUTIERREZ, MA; MORENO RA; MOTTA, G.H.M.B.; BERTOZZO, N.; NARDON, F.B.; FIGUEIREDO, J.C.B.; OLIVEIRA, P.P.M.; TACHINARDI, U. *Prontuário Eletrônico de Pacientes: integrando*

- informações clínicas e imagens médicas*. Revista Brasileira de Engenharia Biomédica, v. 19, n. 3, p. 103-115, 2003.
38. FURUIE, SS; TACHINARDI U ,BERTOZZO N, GAZETTA M ,RAMOS J A, TEIXEIRA JC. *Digitized Paper-Based Medical Records as Part of a HIS/PACS*. Radiology, 221 (P) Fascículo: suppl. RSNA 2001:761, 2001.
39. GAMMA, E., HELM R., JOHNSON R., VLISSIDES J., *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos*, Ed. Bookman, 2000.
40. GONZALEZ, R.C.; WOODS R.E., *Processamento de Imagens Digitais*, Ed. Edgard Blücher Ltda, São Paulo, 2000.
41. GOULARTE R. *Personalização e adaptação de conteúdo baseadas em contexto para TV Interativa*, Tese de Doutorado, 2003
42. GRIMSON J. *Delivering the electronic healthcare record for the 21st century*. International Journal of Medical Informatics, 64: 111-127, 2001.
43. GRUBER, T. R. *A translation approach to portable ontologies*. Knowledge Acquisition, 5(2):199-220, 1993.
44. HENRICKSEN, K; LIVINGSTONE S; INDULSKA J. *Towards a hybrid approach to context modeling reasoning and interoperation*. Proc. Of the Ubicomp'2004 First International Workshop on Advanced Context Modelling, Reasoning and Management, pp. 54-61, 2004.
45. HONG, JI. *The Context Fabric: An Infrastructure for Context-Aware Computing*, In Proceedings of Doctoral Consortium, Human Factors in Computing Systems: CHI 2002, 2002.
46. HONG, J.I.; LANDAY J.A., *An Infrastructure Approach to Context-Aware Computing*, Human Computer Interaction, Vol. 16, 2001.

47. HONG, S.J.; *Use of Contextual Information for feature Ranking and Discretization*. IEEE Transactions on Knowledge and Data Engineering, Volume 9, Issue 5, pp. 718 – 730, 1997.
48. HORVITZ E. *Principles of Mixed-Initiative User Interfaces*, Proc. of CHI'99, pp. 159-166, 1999.
49. HUANG, H.K. (Ed.) *PACS basic principles and applications*, Ed. Wiley-Liss, 2000.
50. IGDA. AI Interface Standards Committee – The 2004 AIISC Report, 2004. Acessível em <<http://www.igda.org/ai/report-2004/rbs.html>>.
51. JOINT EUROPEAN SOCIETY OF CARDIOLOGY / AMERICAN COLLEGE OF CARDIOLOGY. *Miocardic Infarction Redefined – A Consensus Document of the Joint European society of Cardiology / American College of Cardiology Comitee for the Redefinition of Myocardial Infarction*. Journal of the American College of Cardiology, Vol. 36, No 3, 2000.
52. KASIF, S.; SALZBERG, S.; WALTZ, D.; RACHLIN, J.; AHA, D. *a Probabilistic Framework for Memory-Based Reasoning*, 1998.
53. KATEHAKIS D, LELIS P, KARABELA E, TSIKNAKIS M, ORPHANOUDAKIS S: *An Environment for the Creation of an Integrated Electronic Health Record in HYGEIAnet, the Regional Health Telematics Network of Crete*. TEPR 2000, Your Connection to Electronic Healthcare, May 9-11, 2000, Vol. 1, pp. 89-98, 2000.
54. KLEIN G.O. *Standardization of health informatics – results and challenges*, Methods of Information in Medicine, Vol 41(4), pp. 261-70, 2002.
55. KOZIEROK, R.; MAES, P., *A Learning Interface Agent for Scheduling Meetings*, Intelligent User Interfaces, 1993.
56. LASHKARI Y.; METRAL M.; MAES P., *Collaborative Interface Agents*, Proceedings of the Twelfth National Conference on Artificial Intelligence, 1994.

57. LIEBERMAN H.; SELKER T. *Out of Context: Computer systems that adapt to, and learn from context*, IBM Systems Journal, Vol. 39, No. 3&4, 2000.
58. LIU G.C., COOPER JG., SCHOEFLERB K.M.; HAMMOND E. *Standards for the Electronic Health Record Emerging from Health Care's Tower of Babel* . Proceedings of AMIA 2001, 2001.
59. LOPES, A. C.; FILHO, M. C. C. *Clínica Médica Contemporânea II*. São Paulo: Sanrer, 1995.
60. MAES, P. *Agents that Reduce Work and Information Overload*, Communications of ACM, July, Vol. 37, No.7, 1994.
61. MASSAD E.; MARIN, H.F.; AZEVEDO NETO, R.S. (editores); *O Prontuário Eletrônico do Paciente na Assistência, Informação e Conhecimento Médico*, São Paulo, 2003.
62. MOREIRA, J.; MIDKIFF S.; GUPTA M.; WU, P. ARTIGAS e ALMASI G.. *NINJA: Java for High Performance Numerical Computing*, Scientific Programming, pages 19--33, 10(1), 2002.
63. MORENO, R. A., FIALES, V.R., FURUIE, S. S. *Clinical Image Access Service: Achieving Interoperability Using CORBA* In: Proceedings of SPIE, Medical Imaging 2002, San Diego, Califórnia, Vol. 4685, pp. 217 – 227, 2002a.
64. MORENO, R. A., FURUIE, S. S. *Integration Of Heterogeneous Systems Via DICOM And The Clinical Image Access Service* In: Computers in Cardiology 2002, 2002, Memphis, Tennessee. Proceedings of Computers in Cardiology, 2002b.
65. MORENO, R. A., FURUIE, S. S. *Framework for the Development of the Clinical Image Access Service using Java*, In: Proceedings of SPIE, Medical Imaging 2003, San Diego, Califórnia, Vol. 5033 pp.432 – 439, 2003.
66. MOSTÉFAOUI S. K., *Towards a Context-Oriented Services Discovery and Composition Framework*, AI2IA - AI Moves to IA: Workshop on Artificial Intelligence, Information Access, and Mobile Computing, Acapulco, 2003.

67. MOTTA, G. H. M. B. *Um modelo de autorização contextual para o controle de acesso ao prontuário eletrônico do paciente em ambientes abertos e distribuídos*. Tese de doutorado, São Paulo, 2003.
68. NARDON, F.B.; REBELO, M.F.S.; FURUIE, S.S.; MOURA JR., L.A., *Modeling the Electronic Patient Record – The Heart Institute of São Paulo Case*, Anais do TEPR2000, San Francisco, EUA, 2000.
69. NGUYEN, A.; WOBCKE, W. *An agent-based approach to dialogue management in personal assistants*. Proceedings of the 10th international conference on Intelligent user interfaces, pp.137-144, 2005.
70. OLIVEIRA JR P.P.M; REBELO M.S.; PILON P.E.; GUTIERREZ M.A.; TACHINARDI U. *vMon-mobile provides wireless connection to the electronic patient record*. Proceedings of SPIE (4685): 438-445, 2002.
71. OMG 2001a, *Clinical Image Access Service Specification*, June, 2001. Acessível a partir de <<http://cgi.omg.org/docs/dtc/01-08-01.pdf>>.
72. OMG 2001b, *Clinical Observation Access Service Specification*, 2001. Acessível em <<http://www.omg.org/cgi-bin/doc?formal/2001-04-06>>.
73. OMG 2000, *Resource Access Decision Facility Specification*, 2000. Acessível em <<http://www.omg.org/cgi-bin/doc?dtc/00-08-06>>.
74. OMG 1999. *CORBAmed: Healthcare Domain Specifications*. Acessível em <<http://www.omg.org/cgi-bin/doc?formal/99-03-01.pdf>>.
75. OMG 1998, *Patient Identification Service Specification*, 1998. Acessível em <<http://cgi.omg.org/docs/corbamed/98-02-29.pdf>>.
76. OMG. Object Management Group, *Manual* 1996. Disponível em <<http://www.omg.org>>, 1996.

77. PAYMANS, T.F.; LINDENBERG, J.; NEERINCX, M. *Usability trade-offs for adaptive user interfaces: ease of use and learnability*. Proceedings of the 9th international conference on Intelligent user interface, pp. 301-303, 2004.
78. POSTGRESQL, Site do banco de dados *open-source* PostgreSQL. 2004. Disponível em <<http://www.postgresql.org/>>. Acesso em: 12 nov. 2004.
79. PROTÉGÉ. Sítio que disponibiliza a documentação, códigos fontes e binários do programa Protégé. 2004. Disponível em <<http://protege.stanford.edu/>>. Acesso em: 12 nov. 2004.
80. ROSA, N. A., *Uma Abordagem Prática e Eficiente de Consultas por Similaridade para Suporte a Diagnóstico por Imagens*, Tese Mestrado, São Carlos, 2002.
81. ROSENWALD, G.W.; LIU C.C. *Rule-Based System Validation through Automatic Identification of Equivalence Classes*, IEEE Transactions on Knowledge and Data Engineering, Vol 9 (1), pp. 24 – 31, 1997.
82. RUSSELL S.; NORVIG P. *Artificial Intelligence: a modern approach*. Prentice Hall, 1995. ISBN 0-13-103805-2.
83. SCHILIT, B.N.; ADAMS, N.; WANT, R., *Context-Aware Computing Applications*, In Proceedings of the Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, December, pp. 85-90. IEEE Computer Society, 1994.
84. SHEPPARD, J. W.; SALZBERG S. L., *Combining Genetic Algorithms with Memory Based Reasoning*, In Forrest, S., Ed., Proceedings of the Sixth International Conference on Genetic Algorithms, San Mateo, California, pp. 452-459, 1995.
85. SOMMERVILLE, p. 11 – Sommerville, I. Software Engineering. 5. ed. Addison-Wesley, 742p, 1996.
86. SONKA M.; FITZPATRICK J.M. (Eds.) *Handbook of Medical Imaging – Volume 2. Medical Image Processing and Analysis*, SPIE Press, Washington, 2000.

87. STANFILL, C.; WALTZ, D. *Toward Memory-Based Reasoning*, Communications of ACM, Vol 29, No. 12, 1986.
88. STASIU, R.K., *Proposta de um modelo de objetos para informações armazenadas em arquivos DICOM*, Tese de mestrado, Pontifícia Universidade Católica do Paraná, 1999.
89. SUNG, M.Y.; KIM, M. S.; KIM, E.J.; YOO, J.H.; SUNG M.W., *CoMed: a real-time collaborative medicine system*, International Journal of Medical Informatics, 57, pp. 117-126, 2000.
90. TACHINARDI, U.; OLIVEIRA, P. P. M.; REBELO, M. S.; PILON, P. E.; GUTIERREZ, M. A., *Presenting Real-Time Monitoring Data on Handheld Computers Using Wireless Networks*, proceedings TEPR2002 , Seattle, 2002.
91. TACHINARDI U, FURUIE SS, BERTOZZO N, MOURA L, GUTIERREZ MA, MELO CP. *Hypermedia patient data retrieval and presentation through www*. Proc. of 19th Symposium on Computer Applications in Medical Care: 551-555, 1995.
92. VISAD. *Sítio que disponibiliza a documentação e códigos fontes do software Visad (Visualization for Algorithm Development)*. 2004. Disponível em <<http://www.ssec.wisc.edu/~billh/visad.html>>. Acesso em: 16 nov. 2004.
93. WINOGRAD T., *Architectures for Context*, Human-Computer Interaction, Vol. 16: pp 401-419, December 2001.
94. WOOLDRIGE, M.; JENNINGS N. R., *Intelligent Agents: Theory and Practice*, Knowledge Engineering Review, Vol. 10(2), June 1995, Cambridge University Press, 1995.
95. YAN, H.; SELKER T., "Context-aware office assistant", in Proceedings of the 2000 International Conference on Intelligent User Interfaces, pp 276-279, New Orleans, 2000.

Produção bibliográfica e técnica

Os seguintes trabalhos foram publicados durante o transcorrer da tese e relacionados à mesma.

Trabalhos em revista

1. FURUIE, SS; REBELO, MS; GUTIERREZ, MA; MORENO RA; MOTTA, G.H.M.B.; BERTOZZO, N.; NARDON, F.B.; FIGUEIREDO, J.C.B.; OLIVEIRA, P.P.M.; TACHINARDI, U. *Prontuário Eletrônico de Pacientes: integrando informações clínicas e imagens médicas*. Revista Brasileira de Engenharia Biomédica, v. 19, n. 3, p. 103-115, dezembro 2003, © SBEB - Sociedade Brasileira de Engenharia Biomédica. ISSN 1517-3151.

Trabalhos em congressos internacionais

1. MORENO, R A; FURUIE, S. S.. *Contextual medical image viewer*. In: Radiological Society of North America scientific assembly and annual meeting, 2004, Chicago. : Radiological Society of North America scientific assembly and annual meeting program. p. 171-171, 2004.

2. MORENO, R. A., FURUIE, S. S. *Contextual Medical Image Viewer*, In Proceedings SPIE, Medical Imaging 2004: PACS and Imaging Informatics; Osman M. Ratib, H. K. Huang; Eds, Vol. 5371, pp. 382 – 390, 2004.

3. MORENO, R. A., FURUIE, S. S. *Framework for the Development of the Clinical Image Access Service using Java*, In: Proceedings of SPIE, Medical Imaging 2003, San Diego, Califórnia, Vol. 5033 pp.432 – 439, 2003.

4. FURUIE, S. S.; REBELO, M. F. S.; MORENO, R A et al. *Integrating Medical Images and Clinical Information: InCor's Experience*. In: XII International Conference

on Computing, 2003, México. Avances en: Ciencias de la Computación. México: Instituto Politecnico Nacional. v. 3. p. 67-72, 2003.

5. MORENO, R. A., FIALES, V.R., FURUIE, S. S. *Clinical Image Access Service: Achieving Interoperability Using CORBA* In: Proceedings of SPIE, Medical Imaging 2002, San Diego, Califórnia, Vol. 4685, pp. 217 – 227, 2002.

6. MORENO, R. A., FURUIE, S. S. *Integration Of Heterogeneous Systems Via DICOM And The Clinical Image Access Service* In: Computers in Cardiology 2002, 2002, Memphis, Tennessee. Proceedings of Computers in Cardiology . , 2002.

7. MORENO, R. A., FURUIE, S. S. *A Context Model for Medical Image Applications*, In: World Multiconference on Systemics, Cybernetics and Informatics, 2001, Orlando. SCI 2001 - Proceedings. Orlando: International Institute of Informatics and Systemics, 2001. v.X. p.477 – 478.

8. FURUIE, S. S.; MORENO, R A; REBELO, M. F. S. et al. *Visualizador Multimídia Contextual para a Telemedicina*. In: 1º Congresso Internacional de Telemedicina, Educação e Treinamento à Distância, 1999. 1º Congresso Internacional de Telemedicina, Educação e Treinamento à Distância. p. 1-4.

Trabalhos em congressos nacionais

1. MORENO, R A; FURUIE, S. S.. *Utilização de contexto para visualização de Imagens Médicas*. In: IX Congresso Brasileiro de Informática em Saúde, 2004, Ribeirão Preto. Anais do IX Congresso Brasileiro de Informática em Saúde, 2004.

2. URTIGA, K.; TAKIZAWA, K.; MORENO, RA GUTIERREZ, MA. *Mini-WEBPACS: Sistema Compacto para Armazenamento e Distribuição de Imagens Médicas em Ambientes Clínico-Hospitalares*. In: IX Congresso Brasileiro de Informática em Saúde, 2004, Ribeirão Preto, São Paulo. IX Congresso Brasileiro de Informática em Saúde. p. 186-191, 2004.

3. MORENO, R. A., FURUIE, S. S., FIALES, V. *Interoperabilidade de Imagens Médicas Baseado no Serviço de Acesso a Imagens Clínicas (CIAS)* In: Congresso Brasileiro de Informática em Saúde, 2002, Natal. Anais do CBIS2002. , 2002. v. CDROM.
4. FURUIE, S. S., REBELO, M. F. S., GUTIERREZ, M. A., MORENO, R. A., NARDON, F. B., MOTTA, G. H. M. B., FIGUEIREDO, J. C. B., BERTOZZO JUNIOR, N., FIALES, V. *Prontuário Eletrônico em Ambiente Distribuído e Heterogêneo: a Experiência do InCor* In: Congresso Brasileiro de Informática em Saúde, 2002, Natal. Anais do CBIS2002. , 2002. v.CDROM.
5. FIGUEIREDO, J. C. B., FURUIE, S. S., GUTIERREZ, M. A., MOTTA, G. H. M. B., MORENO, R. A. *Sistema Compacto para Armazenamento e Distribuição de Imagens Médicas em Ambientes Clínico-Hospitalares* In: Congresso Brasileiro de Informática em Saúde, 2002, Natal. Anais do CBIS2002. , 2002.
6. MORENO, R. A., FURUIE, S. S. *Visualizador Contextual de Imagens Médicas Orientado a Objeto* In: XVII Congresso Brasileiro de Engenharia Biomédica, 2000, Florianópolis. , 2000.
7. MORENO, R. A., FURUIE, S. S. *Visualização de dados distribuídos utilizando CORBA, XML e Java* In: Congresso Brasileiro de Informática em Saúde, 2000, São Paulo. , 2000.
8. MORENO, R. A., FURUIE, S. S., REBELO, M. F. S., NARDON, F. B. *Visualizador Multimídia Contextual para a Telemedicina* In: TELMED'99, 1999, São Paulo. , 1999.

Apresentações e palestras

1. MORENO, R. A. Seminário 'Web Applications', Instituto do Coração (InCor), Ciclo de Seminários da Divisão de Informática, 2003.
2. MORENO, R. A. Apresentação da palestra 'Distribuição de Imagens Médicas em ambiente heterogêneo', Objetos 6006 - A Vanguarda dos Sistemas Empresariais, Grupo de Usuários da SUCEsu – SP, 2002.
3. MORENO, R. A. Seminário 'DICOM Structured Reporting', Instituto do Coração (InCor), Ciclo de Seminários da Divisão de Informática, 2002.
4. MORENO, R. A. Seminário 'Nova Versão do Prontuário InCor', Instituto do Coração (InCor), Ciclo de Seminários da Divisão de Informática, 2002.
5. MORENO, R. A. Seminário 'Visualizador contextual de imagens médicas', Instituto do Coração (InCor), Ciclo de Seminários da Divisão de Informática, 2002.
6. MORENO, R. A. Tutorial 'Padrões para imagens em Prontuário Eletrônico', Congresso Brasileiro de Informática em Saúde - CBIS 2002, 2002.
7. MORENO, R. A. Comunicação DICOM, Instituto do Coração (InCor), Ciclo de Seminários da Divisão de Informática, 2001.
8. MORENO, R. A. Visualizador Contextual de Imagens Médicas, UNIFESP - Reunião Científica do Departamento de Informática em Saúde, 2001.
9. MORENO, R. A. Visualização Contextual de Imagens Médicas, Instituto do Coração – InCor, Ciclo de Seminários da Divisão de Informática, 2001.
10. MORENO, R. A. Introdução ao Java 3D, Instituto do Coração (InCor), Ciclo de Seminários da Divisão de Informática, 2000.
11. MORENO, R. A. Java e Orientação a Objeto, Instituto do Coração (InCor), Ciclo de Seminários da Divisão de Informática, 1999.
12. MORENO, R. A., FURUIE, S. S. Visualização Contextual de Imagens Médicas, Instituto do Coração (InCor), Ciclo de Seminários da Divisão de Informática, 1999.