CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

MedViewGen: Uma Ferramenta Baseada em Ontologias para Geração Automática de Interface do Usuário para Documentos em Registros Eletrônicos em Saúde

RODRIGO CARTAXO MARQUES DUARTE

João Pessoa

2011

RODRIGO CARTAXO MARQUES DUARTE

MEDVIEWGEN: UMA FERRAMENTA BASEADA EM ONTOLOGIAS PARA GERAÇÃO AUTOMÁTICA DE INTERFACE DO USUÁRIO PARA DOCUMENTOS EM REGISTROS ELETRÔNICOS EM SAÚDE

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba como requisito parcial para obtenção do título de Mestre em Informática.

Área de concentração: Sistemas de computação

Orientador:
Prof. Dr. Gustavo Henrique Matos Bezerra Motta

João Pessoa

2011

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de RODRIGO CARTAXO MARQUES DUARTE, candidato ao Título de Mestre em Informática na Área de Sistemas de Computação, realizada em 07 de fevereiro de 2011.

Aos sete dias do mês de fevereiro do ano dois mil e onze, às nove horas, no Auditório do Centro de Ciências Exatas e da Natureza da Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para examinar o candidato ao grau de Mestre em Informática, na área de "*Sistemas de Computação*", na linha de pesquisa "*Computação Distribuída*", o Sr. RODRIGO CARTAXO MARQUES DUARTE. A comissão examinadora foi composta pelos professores doutores: Gustavo Henrique Matos Bezerra Motta (DI-UFPB), Orientador e Presidente da Banca Examinadora, Clauirton de Albuquerque Siebra e Tatiana Aires Tavares (DI-UFPB), como examinadores internos e Magdala de Araújo Novaes (UFPE), como examinador externo. Dando início aos trabalhos, o Prof. Gustavo Henrique M. Bezerra Motta, cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação intitulado "*MEDVIEWGEN: Uma Ferramenta baseada em Ontologias para Geração Automática de Interface com o Usuário para Documentos em Registros Eletrônicos em Saúde*". Concluída a exposição, o candidato foi argüido pela Banca Examinadora que emitiu o seguinte parecer: "*aprovado*". Assim sendo, deve a Universidade Federal da Paraíba expedir o respectivo diploma de Mestre em Informática na forma da lei e, para constar, eu, professora Tatiana Aires Tavares, Coordenadora deste Programa, servindo de secretária lavrei a presente ata que vai assinada por mim mesmo e pelos membros da Banca Examinadora. João Pessoa, 07 de fevereiro de 2011.

Tatiana Aires Tavares

Tatiana Aires Tavares

Prof. Dr. Gustavo Henrique M. Bezerra Motta
Orientador (DI-UFPB)

Profa. Dra. Tatiana Aires Tavares
Examinador Interno (DI-UFPB)

Prof. Dr. Clauirton de Albuquerque Siebra
Examinador Interno (DI-UFPB)

Profa. Dra. Magdala de Araújo Novaes
Examinador xterno (UFPE)

Agradecimentos

Primeiramente a Deus, por ter me dado forças para persistir e por ter me guiado da melhor forma possível na busca pelos meus objetivos.

Aos meus pais, João e Suzana, que sempre me deram apoio e se sacrificaram de todas as formas para que eu pudesse chegar aonde cheguei; ao meu irmão, Renan, pelo seu imenso apoio; e à minha namorada, Lidianne, pelo seu carinho e palavras de motivação que ajudaram bastante na realização deste árduo trabalho.

Ao professor Gustavo Motta, pela sua orientação presente, confiança no meu trabalho e extrema compreensão em todos os momentos desta caminhada.

Aos meus grandes amigos Gago, Careca, Galego, Boy, Komodo, Cabeção e Targino; pelos momentos de descontração que contribuíram enormemente nos momentos de cansaço e stress.

Aos meus companheiros do LArqSS: Hugo, Duilio, Luciano, Andreia, Walber e João; sem os quais eu não conseguiria alcançar meus objetivos pretendidos no trabalho.

À FINEP, pelo fomento ao projeto OpenCTI.

Sumário

Lista de Figuras e Gráficos

Lista de Tabelas

Lista de Acrônimos

ADL	*Archetype Description Language*
ANS	Agência Nacional de Saúde Suplementar
API	*Application Programming Interface*
CDS	*Clinical Decision Support*
EAV	*Entity-Attribute-Value*
EHR	*Electronic Health Record*
EJB	*Enterprise Java Beans*
EPR	*Electronic Patient Record*
FINEP	Financiadora de Estudos e Projetos
GUI	*Graphical User Interface*
HCI	*Human Computer Interaction*
HL7	*Health Level Seven*
IDE	*Integrated Development Environment*
IRI	*Internationalized Resource Identifier*
HULW	Hospital Universitário Lauro Wanderlei
JSF	*Java Server Faces*
LArqSS	Laboratório de Arquitetura e Sistemas de Software
NOB	Norma Operacional Básica
OMG	*Object Management Group*
OWL	*Web Ontology Language*
PRC	Padronização de Registros Clínicos
UFPB	Universidade Federal da Paraíba
UML	*Unified Modeling Language*
URI	*Universal Resource Identifier*
UTI	Unidade de Terapia Intensiva
W3C	*World Wide Web Consortium*
WG I	*WorkGroup I*
WSDL	*Web Service Description Language*
XML	*eXtensible Markup Language*

Resumo

DUARTE, R. C. M.. **MedViewGen: Uma Ferramenta Baseada em Ontologias para Geração Automática de Interface do Usuário para Documentos em Registros Eletrônicos de Saúde**. 2011. 172 p. Dissertação (Mestrado) – Departamento de Informática, Universidade Federal da Paraíba, João Pessoa, 2011.

A informática em saúde é uma área inovadora e que apresenta grandes desafios a serem vencidos. Como um de seus principais ramos de pesquisa encontram-se os Registros Eletrônicos em Saúde (*Electronic Health Records*, EHR), que são responsáveis, dentre outros, pelo armazenamento, exibição e manipulação de registros clínicos do paciente. Este trabalho é motivado por dois aspectos principais que vêm ganhando relevância no contexto de sistemas para EHR: flexibilidade do domínio da aplicação e apresentação multi-dispositivo. A flexibilidade do domínio inerente aos registros eletrônicos em saúde é um dos principais desafios enfrentados pelos projetistas de interface com o usuário atualmente. A utilização desses registros em larga escala em diversas plataformas de visualização e a crescente necessidade por utilização de ferramentas de suporte a decisão clínica são fatores relevantes. Abordagens presentes na literatura propõem modelos genéricos de representação de domínio e apresentação, com mapeamentos configuráveis entre as duas camadas, porém essa configuração é um processo oneroso e susceptível a erros. Este trabalho tem como objetivo o desenvolvimento de uma ferramenta de geração automática da GUI para documentos em saúde, entitulada MedViewGen, proporcionando independência do domínio de aplicação e de tecnologias de interface. Através da combinação dos metadados extraídos do domínio com metadados intrínsecos aos componentes de interface, ambos representados em ontologias OWL, criamos a possibilidade de mapear, em tempo de execução, quais componentes estão aptos para representar cada conceito biomédico na GUI. Como resultado tivemos uma instanciação da ferramenta para suportar a tecnologia JSF, integrando com OpenCTI e apresentando resultados satisfatórios na geração dos formulários, utilizando um conjunto de componentes de interface adequado para a representação dos documentos e alguns agentes de suporte a decisão clínica.

Palavras-chave: registros eletrônicos em saúde, *interface de usuário*, *ontologias*.

Abstract

DUARTE, R. C. M.. **MedViewGen: An Ontology-Based Tool to GUI Automatic Generation for Documents in Electronic Health Records**. 2011. 172 p. Dissertation (Masters) – Departamento de Informática, Universidade Federal da Paraíba, João Pessoa, 2011.

Health Informatics is an innovative area which presents major challenges to be overcome. As one of its main branches of research are Electronic Health Records (EHR), which are responsible, among other things, the storage, display and management of the patient's clinical records. This work is motivated primarily by two main aspects that have been gaining relevance in EHR context: Application domain flexibility and multi-device presentation. Domain flexibility inherent in the field of electronic health records is a key challenge faced by user interface designers today. The use of such records on a large scale in different enviroments and the increasing need for use of tools to clinical decision support are relevant factors too. Approaches in the literature propose generic models of domain and presentation development, with manual configurable mappings between both layers, but this configuration is very costly and error-prone. This work aims at developing a tool for GUI automatic generation for health documents, titled MedViewGen, providing independence from the scope and interface technologies. By combining metadata extracted from application domain with metadata attached to user interface components, both represented in OWL, we create the possibility to decide at run time, which components are most suitable to represent each biomedical concept. As a result we have an instantiation of the tool to support the JSF technology, and integrated to OpenCTI it presented satisfactory results in forms generation, using a set of interface components suitable for documents representation and some clinical decision support agents.

Keywords: electronic health record, graphical user interface, ontology.

Capítulo

1

Introdução

"A mente que se abre a uma nova idéia jamais voltará ao seu estado original"

Albert Einstein

O prontuário do paciente é o registro das informações clínicas, representadas em documentos, que são colhidas ao longo da vida do paciente (CFM, 2002). Mantém informações como a história do paciente, fatores de risco, anotações sobre anamnese, sinais vitais, exames clínicos, laudos, alergias conhecidas, imunizações, problemas de saúde, procedimentos terapêuticos e medicamentos, resposta a terapias, etc. (Marin *et al.*, 2003).

O armazenamento das informações do prontuário é normalmente realizado no formato tradicional, com a utilização de documentos em papel. As informações contidas nos documentos são, em geral, descritas em forma de texto livre e manuscrito. Entretanto, vários pontos fracos no formato de armazenamento em papel dos prontuários podem ser identificados, tais como caligrafia ilegível, dados ambíguos e incompletos, falta de terminologia única, fragmentação dos dados e falta de disponibilidade (Roukema *et al.*, 2006). Além disso, o crescimento do volume de documentos no prontuário ao longo do tempo dificulta a visibilidade das informações clínicas, tornando difícil para o profissional de saúde recuperar informações relevantes para o cuidado ao paciente.

Os registros eletrônicos em saúde (*Electronic Health Record*, EHR) são uma alternativa de armazenamento para os registros clínicos do paciente e estão associados a diversos benefícios potenciais, como compartilhamento de dados, suporte a decisão, avaliação de qualidade, pesquisa, colaboração e administração do cuidado ao paciente. A sua utilização resolve grande parte dos problemas de armazenamento

e disponibilidade que o prontuário em papel acarreta, com o emprego de bancos de dados de alta capacidadeque podem ser acessados em vários terminais simultaneamente. Muitos problemas relacionados ao prontuário do paciente já tem soluções plausíveis, enquanto outros ainda persistem como problemas em aberto que motivam muitas pesquisas na área.

1.1 Motivação

A informática em saúde é uma área inovadora e que apresenta grandes desafios a serem vencidos. Este trabalho é motivado por dois aspectos que vêm ganhando relevância no contexto de sistemas para EHR: flexibilidade do domínio da aplicação e utilização de ferramentas de apoio a decisão clínica.

1.1.1 Flexibilidade do Domínio da Aplicação

Os registros eletrônicos em saúde, diferentemente da maioria dos sistemas informatizados, contêm características que tornam difícil sua manutenibilidade e colaboração (Greenes, 2006; Miller e Sim, 2004). Uma delas é a flexibilidade requerida pelo domínio da aplicação em saúde, que pode ser visto como um agrupamento de vários sub-domínios específicos, sendo responsável por uma das maiores terminologias do conhecimento existentes (Rosenbloom *et al.*, 2006). Avanços científicos e descobertas na área de saúde são freqüentes, com o constante surgimento de medicamentos, procedimentos e tratamentos, assim como novas patologias e outras inovações. As estruturas de dados necessárias para a representação desses conceitos sofrem alterações à medida que os avanços ocorrem, modificando as estruturas dos documentos clínicos utilizados e inviabilizando a utilização na prática de modelos de dados convencionais (Dinu e Nadkarni, 2007).

Um sistema para EHR adaptável às mudanças deve ser genérico o suficiente para se adequar a diferentes domínios clínicos e contextos organizacionais sem a necessidade de reimplementação. O desenvolvimento da interface com o usuário (*Graphical User Interface*, GUI) desses sistemas traz desafios para os projetistas de software. Com a finalidade de interpretar e apresentar estruturas de documentos e conceitos biomédicos que não são conhecidos a priori, as arquiteturas de software concebidas atualmente propõem uma separação entre a definição das estruturas de dados do domínio e a definição dos componentes de apresentação (Van der Linden *et al.*, 2004 e 2009). Ambas as partes são definidas posteriormente ao período de implementação do sistema, e um mapeamento entre as elas deve ser criado, possibili-

tando a apresentação dos dados para o usuário. Porém, a criação desse mapeamento é um processo multidisciplinar, requerendo para sua realização profissionais com conhecimento tanto na área clínica quanto na área de GUI, o que reduz a quantidade de profissionais habilitados.

1.1.2 Apoio a Decisão Clínica

A flexibilidade é uma característica requerida também por uma área que vem ganhando cada vez mais relevância em sistemas para EHR devido a seus benefícios: o apoio a decisão clínica (*Clinical Decision Support*, CDS). O CDS tem como objetivo principal fornecer informações relevantes para melhorar a qualidade do atendimento ao paciente e da assistência à saúde em geral, com subsídios para formulação de políticas públicas. No EHR, em particular, o CDS pode, por exemplo, agregar valor aos documentos, utilizando-se desde a validação de campos simples, o cálculo automático de fórmulas ou a ativação de alertas e lembretes, até o auxílio à decisão do diagnóstico (Pizzol *et al.*, 2010). A lógica de negócio do CDS está intimamente relacionada com os conceitos biomédicos abrangidos e com o contexto de cada organização, o que a torna, assim como as próprias estruturas de conceitos, mutável ao longo do tempo e do espaço.

Para que o apoio a decisão clínica atinja seus objetivos, é necessária alguma interação entre os agentes relacionados ao CDS e o usuário através da apresentação e captura de informações da GUI. O preenchimento de um documento médico pode requerer a validação de alguns campos, por exemplo, e uma mensagem de erro ou um alerta deve ser exibido para o usuário caso o dado inserido não esteja de acordo com os parâmetros do validador.

Em um sistema para EHR genérico, a lógica de de negócio associada ao CDS não pode ser definida de forma *hard-coded*. Uma vez que as ontologias de conceitos biomédicos são mutáveis ao longo do tempo, e os agentes de CDS são estritamente relacionados a esses conceitos, é preciso uma forma de definição desses agentes de forma flexível. Como conseqüência disso, temos a necessidade de definir a forma de apresentação (componentes de interface, eventos, etc) das informações relacionadas a esses agentes também de forma flexível.

1.2 Objetivo

O objetivo geral deste trabalho é desenvolver uma ferramenta, denominada MedViewGen (*Med*ical *View Gen*erator), que permita a construção automática de interfa-

ces do usuário para registros eletrônicos em saúde em diferentes domínios de aplicação e capaz de se adaptar a contextos e necessidades específicas das organizações de saúde. Além disso, o MedViewGen deve possibilitar a integração entre a interface gráfica e ferramentas de apoio a decisão clínica. Como objetivos específicos, temos:

Objetivo 1. Desenvolver uma API que permita a identificação automática, em tempo de execução, de componentes de apresentação genéricos para elementos de domínio em documentos de saúde;

Objetivo 2. Estabelecer uma arquitetura de software flexível, que permita o acoplamento de novos componentes de software, possibilitando a visualização das informações do EHR em diversas tecnologias de interface e de forma integrada a ferramentas de CDS. O MedViewGen deve permitir que módulos para várias tecnologias distintas sejam integrados;

Objetivo 3. Estabelecer estruturas de definição dos metadados relativos aos componentes de interface e aos mapeamentos para tecnologias de interface, com base na linguagem de ontologias da web OWL (Dean e Schreiber, 2004);

Objetivo 4. Integrar a solução desenvolvida a um EHR, de modo a demonstrar a aplicabilidade dos métodos e da arquitetura utilizados na ferramenta. Utilizaremos para isso o sistema OpenCTI[1].

1.3 Justificativa

Como vimos na seção 1.1, o processo de construção dos mapeamentos entre elementos de domínio e componentes de interface é um processo manual multidisciplinar e susceptível a erros, requerendo para sua realização conhecimento conjunto da área médica e de interfaces gráficas.

O desenvolvimento de uma ferramenta que não apenas acelere o processo de mapeamento entre domínio e apresentação, mas que o faça de forma automática, em tempo de execução, com base nas características intrínsecas aos conceitos biomédicos; acarretaria em enorme economia de esforços durante o período de definição do domínio. Com a realização do mapeamento automaticamente, fica a cargo do modelador apenas a manipulação dos conceitos da ontologia biomédica, sem a preocupa-

[1] OpenCTI: Software de uma Central de Telemedicina para Apoio à Decisão Médica em Medicina Intensiva. Projeto financiado pela FINEP nº 01.08.0533.00.

ção de como eles serão representados na camada de apresentação. Ao alcançarmos os objetivos da seção 1.2, estaremos construindo uma ferramenta de geração automática de interfaces gráficas flexíveis quanto ao domínio da aplicação, uma vez que, para cada um dos objetivos temos:

Justificativa 1. A construção do módulo de mapeamento automático fará com que, em tempo de execução, a forma de apresentação para cada elemento de domínio em um documento seja atribuída. Isso proporcionará economia de esforços durante a modelagem, uma vez que não é necessário designar a forma de apresentação de cada elemento, além do que os modeladores do domínio não precisam ter conhecimentos sobre as tecnologias utilizadas para apresentação dos dados. Assim, erros humanos ocasionados durante a construção manual dos mapeamentos são reduzidos;

Justificativa 2. O estabelecimento de uma arquitetura de software bem definida torna o projeto re-usável em diversos contextos e aumenta a manutenibilidade do MedViewGen, assim como a facilidade de serem acopladas extensões posteriores. Uma arquitetura flexível permite a visualização de uma mesma base de dados para interfaces desenvolvidas em tecnologias distintas, não deixando o sistema específico a uma única forma de interação e permitindo que o sistema se adéqüe ao avanço da tecnologia na área. Além disso, permite uma integração com acoplamento fraco entre a GUI e os agentes de CDS;

Justificativa 3. O estabelecimento de modelos para construção de ontologias de componentes de interface e dos mapeamentos para tecnologia específica é crucial para que a ferramenta atribua componentes de interface automaticamente. Esses modelos ontológicos permitem a representação de informações dos componentes de GUI necessárias para a identificação de qual a melhor forma de apresentação para cada tipo de elemento de domínio em um documento. Além disso, o uso de ontologias permite que sejam realizadas alterações, inserções e remoções na definição dos componentes;

Justificativa 4. A integração do MedViewGen a um sistema EHR nos ajudará a testar a eficácia da ferramenta, tanto nas questões arquiteturais quanto na funcionalidade em si, e extrair informações que possam ser utilizadas em correções de projeto e em possíveis melhorias futuras.

1.4 Metodologia

A metodologia adotada aborda a definição das atividades principais do projeto, identificação das tecnologias que foram utilizadas e a definição de um processo de desenvolvimento que serviu de guia para a realização das atividades com qualidade e dentro dos prazos previstos no cronograma.

1.4.1 Atividades Realizadas

Para que os objetivos estabelecidos fossem alcançados, algumas atividades primordiais foram realizadas. A seguir apresentamos uma lista com essas atividades:

- Revisão bibliográfica: busca de artigos científicos relacionados à interface com o usuário em registros eletrônicos em saúde, identificando lacunas que possam ser exploradas pelo MedViewGen;
- Estudo das tecnologias de geração de interface web, necessárias para a construção de uma implementação de validação da ferramenta;
- Estudo aprofundado sobre ontologias OWL e as ferramentas necessárias para a construção aprimorada de ontologias. Esse estudo auxiliou a identificar as melhores práticas de modelagem a serem utilizadas na definição do modelo ontológico de componentes de interface e dos mapeamentos para tecnologias de GUI;
- Projeto da arquitetura do MedViewGen: definir cada camada, seus componentes e suas respectivas responsabilidades, com base em padrões de projeto consagrados e abertos;
- Implementação do MedViewGen: codificar e testar a arquitetura projetada, seguindo um processo de desenvolvimento evolucionário. A implementação deve conter, além dos componentes de geração automática de GUI, um protótipo de visualizador em tecnologia web escolhida para fins de testes e validação;

1.4.2 Tecnologias

Para alcançarmos o objetivo do trabalho, adotamos um conjunto de tecnologias e ferramentas auxiliares que apresentaremos a seguir. Porém, algumas das tecnologias julgamos serem fundamentais para a solução do problema. São elas: Java, UML e OWL.

Como base para a implementação do MedViewGen, escolhemos a linguagem de programação Java (Gosling *et al.*, 2003), por diversos motivos. Primeiramente, as tecnologias baseadas em Java estão atualmente entre as mais utilizadas para desenvolvimento de aplicações web. Dispõe de um conjunto de frameworks desenvolvidos por diversas empresas como *Sun Microsystems* e *Apache*. Outra característica importante é a portabilidade dos programas desenvolvidos nesta linguagem de programação. Com a utilização de uma máquina virtual, uma mesma aplicação Java executa em diversos ambientes. Além disso, várias ferramentas existentes para a linguagem oferecem um ambiente adequado para o desenvolvimento de aplicações de qualidade, desde IDE's até API's de manipulação de ontologias.

Alguns de seus frameworks merecem destaque. O EJB (*Enterprise Java Beans*) é um framework para aplicações distribuídas do lado do servidor (Panda *et al.*, 2009) . Ele provê a infra-estrutura necessária para a construção de aplicativos web oferecendo alguns serviços como gerenciamento de sessões, controle de transações e segurança. Esta será a principal tecnologia utilizada no desenvolvimento do MedViewGen. Outro framework que merece destaque é o JSF (*Java Server Faces*), que é utilizado na camada web de aplicações distribuídas. Esta será a principal tecnologia utilizada no desenvolvimento do visualizador de validação do MedViewGen.

A modelagem do software foi especificada na linguagem UML (Larman, 2007). O motivo dessa escolha é o fato de UML ser um padrão aberto, largamente adotado e especificado pela OMG (*Object Management Group*). O fato de ter sido criado na OMG, um consórcio de empresas de desenvolvimento de software de bastante sucesso, dá o respaldo necessário para que o padrão tenha continuidade.

Por ser uma linguagem amplamente usada para representação de conteúdo semântico na web, a OWL (Dean e Schreiber, 2004) foi selecionada para a definição dos metadados dos componentes de interface que serão utilizados no processo de geração automática da GUI e na definição dos mapeamentos em instâncias de tecnologia específica, permitindo interoperabilidade com outras ontologias e maior disseminação da informação representada. Utilizaremos para a modelagem das ontolo-

gias o Protegé (Protegé, 2010), ferramenta desenvolvida em Java que contém uma API integrada de acesso a ontologias.

1.4.3 Processo de Desenvolvimento

Para o desenvolvimento deste trabalho, escolhemos o processo de desenvolvimento evolucionário. Segundo Sommerville (2006), o desenvolvimento evolucionário é o mais adequado para sistemas de pequeno e médio porte. A idéia central por traz do modelo evolucionário está no desenvolvimento de uma implementação inicial que vai sendo refinada em cada iteração, de acordo com o *feedback* dos usuários, até que se alcance uma versão satisfatória do sistema. Neste modelo de processo, em vez de serem seqüenciais, as atividades de especificação, desenvolvimento e validação são intercaladas.

Uma das principais vantagens do modelo evolucionário que levou à sua escolha é que a especificação pode ser realizada de forma incremental. Uma vez que o trabalho desenvolvido é inovador, o grau de incerteza com relação aos requisitos é grande, dificultando a realização da especificação por completo antes do início do desenvolvimento. No desenvolvimento do MedViewGen, um protótipo inicial foi desenvolvido com um número reduzido de requisitos e posto para avaliação. À medida que o projeto avançava, novas versões vão foram sendo desenvolvidas com requisitos cada vez mais refinados, até que os objetivos fossem alcançados.

1.5 Estrutura do Trabalho

Esta dissertação de mestrado está estruturada de acordo com a seguinte lista de capítulos:

- No segundo capítulo, “Fundamentação Teórica”, apresentaremos os conceitos fundamentais para o desenvolvimento do trabalho. Começaremos com um esclarecimento sobre o contexto atual dos registros eletrônicos em saúde, discorreremos sobre as características relevantes no tocante a interface com o usuário e ontologias OWL e, por fim, apresentaremos os trabalhos relacionados presentes atualmente na literatura;
- No terceiro capítulo, “Arquitetura do MedViewGen”, levantaremos o conjunto de requisitos que desejamos que nossa ferramenta satisfaça, através de uma arquitetura de software que gere interface dinamicamente para a visualização de documentos. Neste mesmo capítulo apre-

sentaremos soluções para os requisitos explicitados de forma conceitual, sem entrar nos detalhes de implementação;

- No quarto capítulo, “Modelagem Semântica e Algoritmos de Tradução”, apresentaremos os aspectos relativos aos modelos semânticos estabelecidos e os algoritmos a eles associados;
- O quinto e penúltimo capítulo, “Resultados”, apresenta os resultados obtidos com a implementação do MedViewGen, demonstrando exemplos representativos das características fundamentais do MedViewGen;
- No sexto e último capítulo, “Considerações Finais”, discutiremos sobre os resultados obtidos, comparando-os com os trabalhos relacionados, ressaltando pontos onde melhorias podem ser realizadas, motivando trabalhos futuros na área.

Capítulo

2

Fundamentação Teórica

> "A ciência não é uma ilusão, mas seria uma ilusão acreditar que poderemos encontrar noutro lugar o que ela não nos pode dar"
>
> *Sigmund Freud*

Este capítulo tem como objetivo descrever conceitos fundamentais para o desenvolvimento da arquitetura da ferramenta MedViewGen, incluindo aqueles presentes na literatura relacionada. Começamos na seção 2.1 discorrendo sobre os registros eletrônicos em saúde e seu panorama atual. Na seção 2.2 apresentamos conceitos importantes relacionados à GUI que são utilizados no trabalho. A seção 2.3 traz uma introdução sobre ontologias OWL e seu uso na área de saúde. Por fim, apresentamos na seção 2.4 os trabalhos relacionados a interfaces com o usuário em EHR e abordagens de geração automática de GUI de forma geral, explicando e identificando lacunas a serem exploradas.

2.1 Registros Eletrônicos em Saúde

No capítulo 1 introduzimos o conceito de registro eletrônico em saúde como uma alternativa de informatização dos prontuários em papel. Na verdade, ele é bem mais que isso. Nesta seção, iremos discorrer um pouco mais a fundo sobre o EHR, apresentando as características que o torna diferente da maioria dos sistemas de informação convencionais, e apresentando o panorama atual.

2.1.1 Introdução

A introdução da informática no ambiente de saúde teve início na década de sessenta, com o desenvolvimento de sistemas que compreendiam análises estatísticas e epi-

demiológicas. Com o aprimoramento da tecnologia, permitiu-se o desenvolvimento de sistemas administrativos hospitalares, sistemas de controle de farmácias e laboratórios (Petry e Lopes, 2005). Entretanto, a magnitude deste potencial não tem sido eficazmente desfrutada pelas organizações de saúde. A automatização do processo de atendimento ao paciente revelou inegáveis benefícios, porém, este progresso continua aquém do avanço alcançado em outras áreas do conhecimento, como engenharia, física e finanças. Isto se dá, principalmente, pelo fato das soluções tecnológicas em saúde terem sido concebidas por organizações isoladas, sofrendo adaptações apenas para necessidades específicas de cada contexto em particular (Greenes, 2006).

A essência dos sistemas de informação na área de saúde está vinculada ao Registro Eletrônico em Saúde (EHR). O EHR tem a finalidade de reunir informações referentes ao processo de atendimento clínico de um paciente, incluindo dados demográficos, registros de alergias, diagnóstico de doenças, prescrição de medicamentos, resultados de exames, entre outros. Dessa forma, o EHR pode proporcionar melhoria na qualidade de atendimento, se garantida a integridade de informações, possibilitando uma assistência médica mais eficaz.

A possibilidade de manipular as informações de saúde eletronicamente trouxe como conseqüência a necessidade de estabelecer um padrão de saúde para permitir a troca de informações. Assim, diversas organizações e grupos de pesquisa têm-se reunido nos últimos anos para propor regras que viabilizem a interoperabilidade de sistemas de saúde. Veremos a seguir alguns dos principais estudos na área.

2.1.2 Panorama Atual

Ainda nos dias atuais, diversos sistemas que manipulam informações de saúde não se comunicam entre si. É possível encontrar sistemas pertencentes à mesma organização de saúde, mas que são incapazes de trocar informações, acarretando problemas desnecessários como, por exemplo, a presença de vários cadastros para um mesmo paciente e a dificuldade de recuperação das informações a ele relacionadas. Assim, padrões que permitam o compartilhamento de informações são essenciais (Dogac *et al.*, 2007).

A padronização na informática em saúde é necessária devido a diversos fatores, dentre os quais podemos destacar: diversidade de conceitos e termos, plataformas de hardware e software distintas, dificuldade de busca e comunicação de informações e o uso de sistemas de apoio à decisão. A característica mais importante de um EHR quanto à padronização é promover o compartilhamento de informações entre diferentes usuários autorizados. Para isto, é necessária a interoperabilidade de

informação em um EHR e a interoperabilidade entre sistemas que trocam e compartilham informações de um EHR. São verificados dois níveis principais de interoperabilidade de informações (Dogac *et al.*, 2007):

- Interoperabilidade funcional, que diz respeito à interação de dois ou mais sistemas (equipamentos, sistemas de informação, bases de dados) para trocar informações de acordo com um conjunto de regras definidas;

- Interoperabilidade semântica, que corresponde à capacidade de sistemas compartilharem informações compreendidas através da definição de conceitos de domínio.

Grupos de pesquisa nacionais e internacionais investem em estudos relativos à padronização de sistemas para EHR. Veremos a seguir algumas dessas iniciativas.

2.1.2.1 Padrões Internacionais

Alguns dos principais padrões internacionais desenvolvidos para proporcionar a troca de informações na saúde são: HL7 (HL7, 2010), CEN/TC251 (CEN/TC251, 2010) e openEHR (Garde *et al.*, 2007).

O HL7 é um padrão proprietário, desenvolvido através de uma organização sem fins lucrativos denominada *Health Level Seven*. A sua terceira versão, que propõe modelos de informação para representar o ambiente de saúde, foi parcialmente aprovada pela ANSI- SDO (*American National Standards Institute- Standards Development Organization*). Nela são definidas as estruturas das mensagens que representam informações clínicas, administrativas e financeiras consideradas fundamentais em um ambiente hospitalar.

Na Europa, temos uma composição de quatro grupos de trabalho responsáveis pela normatização da área de saúde no continente, denominado CEN/TC251 (Comitê Técnico Euripeu). O WorkGroup I (WG I) é responsável pela elaboração de padrões para o registro eletrônico de saúde, estabelecendo as normas para a representação do conteúdo e a estrutura dos registros de saúde, definindo formatos para representação de conceitos, termos, regras e mecanismos para o compartilhamento e a troca de informações. O pré-padrão prEN 13606 (*Health Informatics - Electronic Health Record Communication - Part 1 Reference Model*) define um modelo conceitual para estruturar dados médicos de maneira uniforme, conhecido como CEN 13606/archetypes, preservando o significado e contexto dos dados.

O OpenEHR é uma fundação sem fins lucrativos com a finalidade de desenvolver especificações de forma aberta para a representação e comunicação em EHR,

baseada em pesquisas e na experiência de implementação. OpenEHR fornece modelos de informação e de serviços para EHR, *workflow* de informações clínicas, demográficas e arquétipos que são utilizados para modelar conceitos clínicos. Além disso, o openEHR fornece exemplos de implementação com código aberto para facilitar o entendimento e uso do padrão proposto (Petry, 2006).

2.1.2.2 Padrões Nacionais

No Brasil, são encontradas três iniciativas de padronização: elaboração da Padronização de Registros Clínicos (PRC, 2010); desenvolvimento do Sistema Cartão Nacional de Saúde (Barros *et al.*, 2004); e um padrão para a Troca de Informação em Saúde Suplementar (TISS, 2007) .

A Padronização de Registros Clínicos (PRC) foi estabelecido em março de 1998, através da criação do Comitê Temático Interdisciplinar. Ela tem como principal objetivo a criação de padrões para a construção de prontuários informatizados. O PRC promove a padronização de dados como a identificação do paciente, dados clínicos relevantes, diagnósticos, instituições, fonte pagadora e procedimentos realizados. A definição de alguns destes dados, após padronizados, serviu como base para as definições do registro de atendimento do Cartão Nacional de Saúde.

Tendo como objetivo a identificação individualizada dos pacientes, acompanhamento do registro do atendimento em saúde em todo o território nacional e a possibilidade de desenvolvimento do repositório nacional de atendimentos, o Cartão Nacional de Saúde foi enunciado pela Norma Operacional Básica (NOB) de 1996. Com isso, o padrão é capaz de definir a estrutura e o conteúdo da informação, permitindo a integração dos diversos sistemas de informações existentes.

O TISS, estabelecido pela ANS (Agência Nacional de Saúde Suplementar), tem como principal objetivo a padronização da troca eletrônica de informações administrativas e financeiras entre operadoras e prestadores de saúde. Com isto será possível uma simplificação dos processos envolvidos na saúde suplementar e conseqüentemente, uma redução de custos administrativos entre os participantes (operadoras e prestadoras de serviços).

2.2 Interface Gráfica do Usuário

Para que os sistemas de software interajam com usuários humanos, deve existir uma forma de comunicação entre eles. Essa comunicação é realizada através da interface do usuário. Existem vários tipos de interface do usuário, dentre os quais se destacam

as interfaces gráficas, que, por meio de uma mescla de gráficos e informações textuais, possibilitam aos usuários a realização de suas tarefas dentro do sistema.

A interface gráfica do pode motivar o usuário na utilização do sistema e, dependendo de suas características, tornar-se uma grande ferramenta para o usuário (Hoekman, 2007). Por outro lado, se mal projetada, pode se transformar em um ponto decisivo na rejeição de um sistema.

O objetivo das interfaces atuais é fornecer uma maior qualidade nas interações humano-computador. Uma interface gráfica bem desenvolvida deve ser de fácil utilização pelo usuário, mostrando claramente as alternativas disponíveis a cada passo da interação sem confundir nem deixar o usuário inseguro. O foco do usuário deve ser o problema para o qual o sistema se propõe a resolver, não tendo que desperdiçar parte de sua atenção com dificuldades geradas por uma interface mal projetada.

No âmbito das interfaces gráficas, elementos visuais denominados **componentes de interface**, contendo imagens representando dados e tarefas disponíveis que são manipuladas diretamente pelo usuário, foram concebidos com o objetivo de melhor estruturar as interfaces gráficas. A utilização desses componentes tem como objetivo tornar a interação com o usuário mais natural e de fácil utilização. É comum confundir os conceitos, porém os componentes de interface não constituem os dados nem as tarefas, sendo apenas seus signos capazes de representá-los de maneira visual para o usuário (Oliveira Netto, 2004).

Os sistemas de software atuais não são vistos apenas como processamento de dados. São encarados como mensagens complexas enviadas dos projetistas para os usuários. Pesquisas na área Interação Humano-Computador (HCI - *Human Computer Interaction*) conscientizaram os programadores da necessidade de uma comunicação em interfaces nos dois sentidos, isto é, os usuários não só enviam mensagens para os programas de aplicação, mas também recebem e interpretam mensagens provenientes da ou através da aplicação. A HCI, quando examinada sob o ponto de vista da comunicação, mostra que os sistemas possuem dois papéis de comunicação inerentes: eles são geradores e receptores de mensagens e eles próprios constituem mensagens enviadas dos projetistas para os usuários através do meio.

2.2.1 Componentes de Interface

De acordo com a definição utilizada neste trabalho, um componente de interface é um elemento visual que permite a entrada e/ou exibição de dados entre o usuário e o software. Utilizamos também para este trabalho uma classificação de componentes

de interface baseada no seu tipo de interação com o usuário, separando-os em dois grupos: componentes de entrada e componentes de saída.

Componentes de entrada são componentes de interface que permitem ao usuário expressar alguma informação, de forma textual ou de outra natureza distinta, durante sua interação com o sistema. Um exemplo desse tipo de componente é uma simples caixa de texto, onde o usuário, através do teclado, pode inserir dados que serão processados pelo sistema. **Componentes de saída** são componentes que permitem a exibição para o usuário de informações advindas do próprio sistema, seja através de mensagens textuais ou outras formas de representação gráfica de informação.

2.2.2 Layouts

Para que um determinado conjunto de componentes de interface faça sentido para o usuário e lhe permita interagir com o sistema, é necessário que haja uma organização ergonômica na tela. Os *layouts* são formatos padrão de organização de componentes, utilizados para facilitar a navegabilidade do usuário através do sistema, agrupando os componentes de acordo com suas funções e características (Avellar e Duarte, 2010).

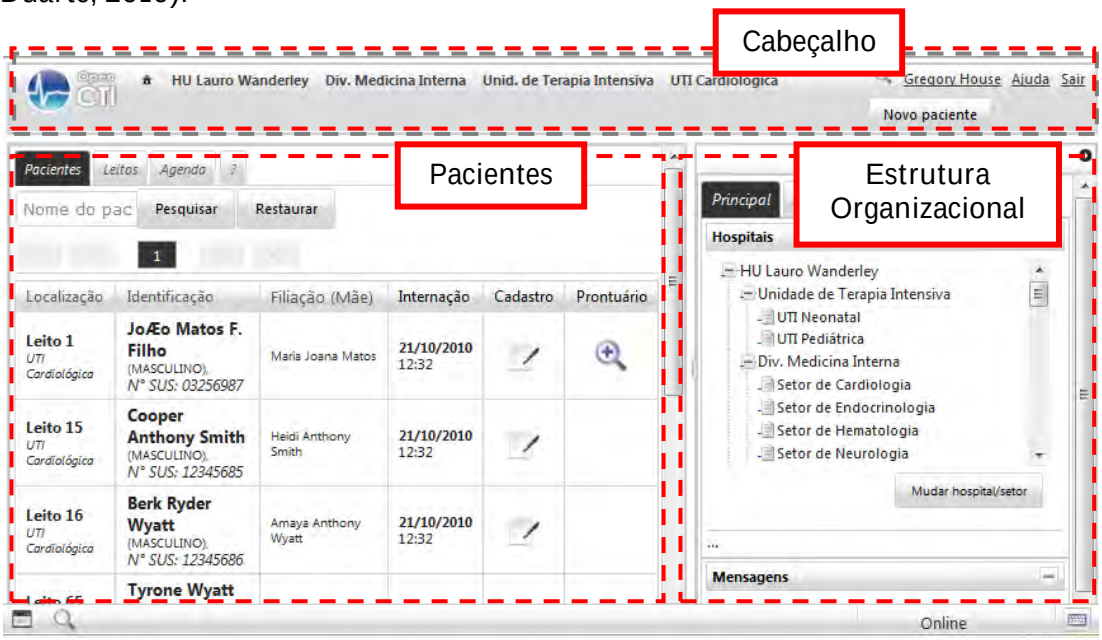

Figura 1: Página inicial do OpenCTI

Como podemos ver no exemplo da Figura 1, os componentes na tela estão agrupados de acordo com seu propósito. Na parte superior temos o cabeçalho, contendo informações gerais do sistema; na parte inferior esquerda, temos uma listagem dos pacientes internados; e na parte inferior direita, temos a estrutura organi-

zacional do hospital. O layout da interface não se relaciona apenas à sua aparência, que contribui para a experiência subjetiva e emocional de recepção. Na verdade o layout, com o estilo visual, fontes, cores, imagens, é parte integrante do design da informação e da funcionalidade dos aplicativos.

2.2.3 Formulários

Um formulário é um conjunto de componentes de interface, além de um estilo de interação, que permite ao usuário entrar dados em um sistema. Os formulários representam uma forma estruturada de entrada de dados, sendo muito utilizados em sistemas de informação dos mais diversos ramos.

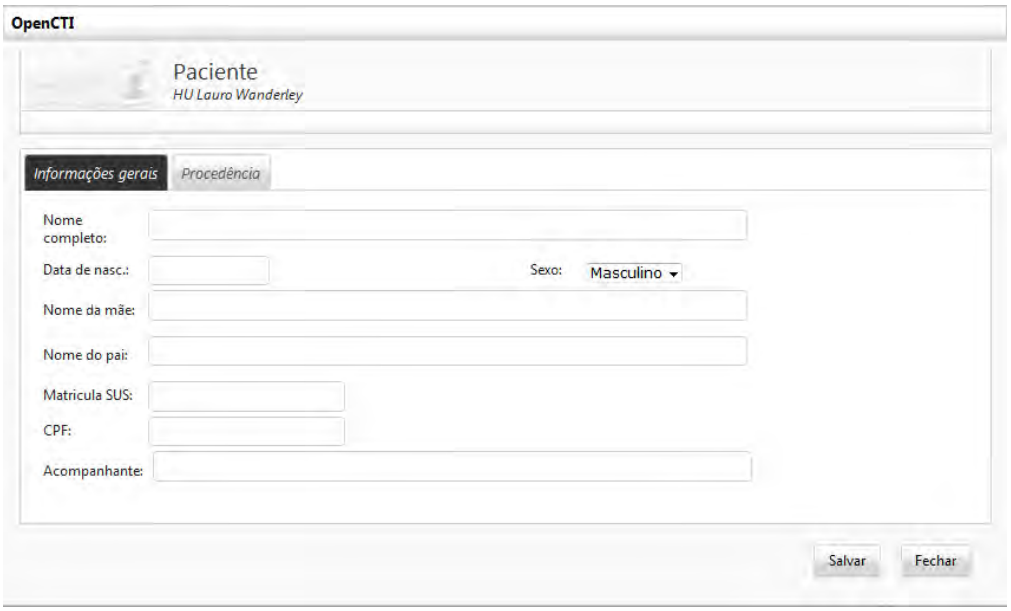

Figura 2: Formulário de cadastro de paciente do OpenCTI

A **Figura 2** apresenta um exemplo de formulário extraído do OpenCTI. Neste formulário temos um conjunto de componentes de interface organizados de forma que o usuário possa inserir dados relativos a um novo paciente. Podemos notar que o usuário interage com o sistema inserindo apenas os dados que o sistema requer, de forma estruturada, e não de forma livre, em texto corrido. Esta forma de representação visual da informação facilita o processamento por parte da máquina, porém diminui a liberdade do usuário.

2.2.4 Eventos

Eventos representam ações do usuário na sua interação com o sistema. Determinadas ações do usuário ativam eventos do sistema que, por sua vez, podem iniciar a execução automática de determinadas ações. Aproveitando o exemplo da **Figura 2**,

durante o cadastro de dados de um novo paciente, ao se pressionar no botão “salvar”, um evento é lançado no sistema. Esse evento é então capturado e, por sua vez, ativa a execução do processo de gravação dos dados.

2.3 Ontologias OWL

Ontologias são usadas para capturar o conhecimento sobre algum domínio de interesse. Uma ontologia descreve os conceitos do domínio e também as relações que mantêm entre esses conceitos. Diferentes linguagens de ontologia fornecem diferentes funcionalidades, sendo cada uma delas melhor ou pior de acordo com a necessidade. O mais recente desenvolvimento em linguagens de ontologia, a OWL é padrão da *World Wide Web Consortium* (W3C, 2011). Essa linguagem permite descrever não apenas conceitos, mas também fornece um rico conjunto de operadores (por exemplo, interseção, união e negação). É baseado em um modelo lógico diferente que torna possível para os conceitos serem definidos semanticamente, assim como descritos. Conceitos complexos podem ser construídos a partir de definições de conceitos simples e dos relacionamentos que eles possuem entre si. Uma ontologia OWL é constituída basicamente por três tipos de conceitos: Indivíduos, propriedades e classes.

2.3.1 Classes

Uma das principais ferramentas de descrição em OWL são as classes. As classes são espécies de conjuntos com elementos que contêm determinadas características. Na definição de uma classe, devem ser consideradas as características de interesse que os elementos a serem agrupados devem *ter*. Além disso, as classes em sua forma mais simples se relacionam em forma de uma hierarquia taxonômica, ou seja, apenas de forma classificatória. Um fato sobre classes OWL que deve ser destacado é que todas as classes descendem direta ou indiretamente da classe owl:*Thing*. Essa é uma classe nativa da OWL, que representa qualquer coisa. Fazendo-se um paralelo com Java, a classe owl:*Thing* é similar à classe *Object*.

2.3.2 Indivíduos

Os indivíduos podem ser definidos como os elementos das classes, ou seja, os indivíduos são os membros das classes. Utilizando-se a nomenclatura orientada a objetos, pode-se dizer que eles são as instâncias das classes.

2.3.3 Propriedades

Utilizando apenas classes e indivíduos, obtém-se uma semântica exclusivamente taxonômica. Desta forma, para agregar valor às descrições taxonômicas, OWL provê a possibilidade de descrição de propriedades.

As propriedades especificam fatos sobre indivíduos por meio de afirmações sobre a classe ou diretamente sobre os indivíduos. Além disto, existem dois tipos de propriedades em OWL, as propriedades de dado (*data type properties* ou simplesmente *data properties*) e as propriedades de objeto (*object properties*).

Os *object properties* são propriedades que relacionam dois indivíduos quaisquer descritos na ontologia. Os *data properties*, por sua vez, são propriedades que ligam dados em forma de valores de algum tipo a um indivíduo.

2.4 Trabalhos Relacionados

Nesta seção serão apresentados os principais trabalhos relacionados à problemática exposta na seção 1.1. Começaremos, na seção 2.4.1, discorrendo sobre as abordagens atuais para interfaces em EHR, identificando os principais problemas ainda não solucionados. Seguiremos na seção 2.4.2 com uma breve discussão a respeito dos trabalhos relacionados a possíveis abordagens de solução relacionadas à concepção do MedViewGen.

2.4.1 Interface com o Usuário em Sistemas EHR

Como vimos brevemente na seção 1.1, são várias as complicações existentes no desenvolvimento de interfaces com o usuário para registros eletrônicos em saúde. O fato de o domínio da aplicação em sistemas para EHR ser mutável ao longo do tempo traz consigo a necessidade de criar formas de apresentação e interação adaptativas.

Um dos principais trabalhos na área, realizado por (Schuler *et al.*, 2006), é proveniente do projeto OpenEHR, citado na seção 2.1.2.1 como uma das principais iniciativas internacionais de padronização em EHR. Em seu trabalho, Schuler ressalta a interoperabilidade semântica como um dos principais focos de estudo na área. Para ele, a utilização de todo o potencial oferecido através de EHRs interoperáveis só é alcançada através da utilização de interfaces ricas em usabilidade e customização. A motivação principal do trabalho é encontrar uma maneira de construir GUIs a partir de conceitos modelados na forma de arquétipos no modelo do OpenEHR. Em outras palavras, o objetivo principal do trabalho é encontrar uma forma de represen-

tar, para cada conceito biomédico, as informações relativas à apresentação desses conceitos para o usuário.

Para que os objetivos fossem alcançados, foi proposta uma abordagem baseada na definição das características de apresentação de cada conceito, assim como a definição do próprio conceito, na forma de arquétipos, proporcionando a geração de interfaces coerentes, customizáveis e com a presença de validação de dados. Atreladas a cada conceito biomédico, são descritas suas informações de apresentação, como o tipo de componente de interface que irá exibir os dados, o posicionamento, tipo de fonte, etc. Todas essas as informações genéricas de apresentação relacionadas a cada conceito são representadas utilizando-se a sintaxe de uma linguagem ADL (*Archetype Description Language*).

Outro trabalho com características bastante similares na área é proveniente de outro projeto de EHR, conhecido como PropeR (Van der Linden *et al.*, 2003, 2005). O trabalho de (Van der Linden *et al.*, 2009) é um dos mais recentes trabalhos relacionados a geração de GUI para EHR. No âmbito do projeto PropeR, um EHR baseado na web foi criado usando uma versão simplificada da abordagem CEN 13606 archetypes. O foco principal é na implementação de um sistema com domínio flexível baseado nesses arquétipos, com representações genéricas das telas. Com base no trabalho de Schuler, é comprovada a possibilidade de se gerar representações genéricas de GUI utilizando arquétipos, porém os resultados obtidos revelam que a informação é utilizada como base apenas para obter a apresentação, em vez de exibir as informações de uma forma familiar para o usuário. Van der Linden propõe então apoiar a definição de um formato de exibição ótima, mantendo a GUI genérica.

A abordagem de dois modelos (*Two-Model Approach*) utilizada por Van der Linden, que é a base para os arquétipos, tem provado ser útil na separação entre o desenvolvimento de software e a definição do conhecimento. Isso inspirou os autores a aplicarem a mesma abordagem para o domínio GUI, em uma tentativa de permitir a geração de boa qualidade e apresentações usáveis sem a necessidade de que cada estrutura de dados seja conhecida antecipadamente durante o projeto do sistema. A separação entre as definições dos conceitos biomédicos e sua forma de apresentação permite o reuso das informações de apresentação em diversos conceitos biomédicos com características similares, diminuindo a redundância existente na

abordagem de Schuler.

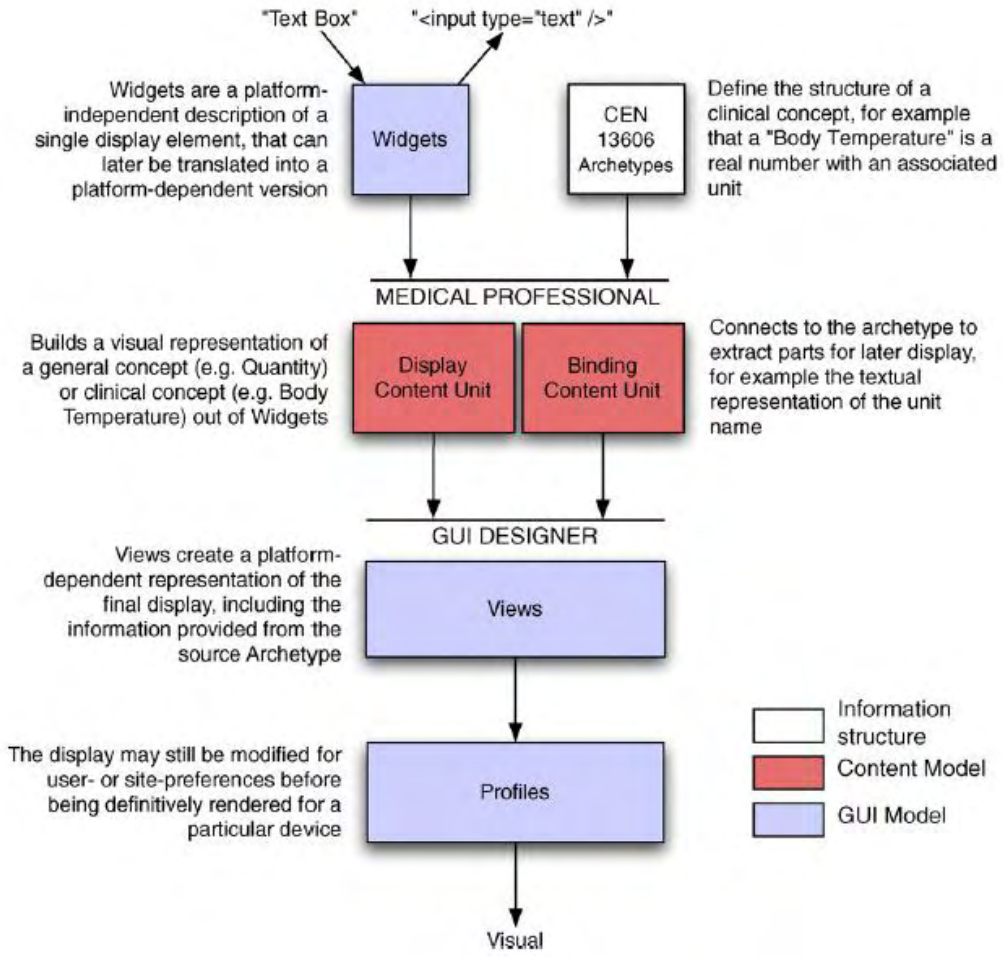

Figura 3: Diagrama demonstrativo do modelo adotado por Van der Linden *et al*. (2009)

Como podemos ver na **Figura 3**, as definições dos conceitos são descritas no componente "**CEN 13606 Archetypes**", enquanto as definições dos elementos de apresentação são descritas no componente "**Widgets**". A ligação entre as duas partes é feita através dos componentes "**Display Content Unit**" e "**Binding Content Unit**", que contêm o mapeamento de quais componentes de interface são utilizados por quais conceitos biomédicos. As informações até esse ponto são descritas de modo genérico, ou seja, de forma independente de plataforma. Os componentes "**Views**" e "**Profiles**" fazem a conversão para um formato dependente de plataforma, de acordo com as características do visualizador do usuário, como tecnologia de interface, dispositivo de visualização, etc.

Um ponto importante a ser ressaltado com relação aos dois trabalhos apresentados é que ambos têm como foco apenas a exibição (saída) dos dados. A parte que diz respeito à entrada de dados é citada como trabalho futuro, deixando uma lacuna e uma área a ser explorada. Em termos de comparação entre os dois modelos,

o modelo apresentado por Van der Linden mostra-se mais robusto que o de Schuler por que sua arquitetura possibilita maior independência e fraco acoplamento entre o domínio, composto pelos conceitos biomédicos, e a interface gráfica, composta pelos elementos de apresentação. Apesar disto, em ambas as propostas, é necessário que para cada conceito biomédico seja definida sua forma de apresentação, e qualquer alteração no domínio requer que sejam feitas também alterações na apresentação, ainda que esta alteração seja apenas uma configuração.

2.4.2 Geração de GUI

Problemas relacionados à criação de interfaces dinâmicas estão presentes em sistemas de software de diversas áreas. A literatura abrange algumas abordagens de geração de interface baseadas em modelos configuráveis.

Um dos principais trabalhos nessa linha foi realizado por Jiang *et al.* (2007). Seu trabalho tem como objetivo a geração automática de GUI para *web-services*. Como é característica dos serviços da web, a interoperabilidade entre softwares desenvolvidos em tecnologias distintas, a necessidade de independência dos serviços em relação à tecnologia das interfaces com o usuário também se insere nesse contexto.

A interação entre *web-services* acontece através de interfaces (de software) especificadas em uma linguagem padrão denominada WSDL (*Web Service Description Language*). A idéia do trabalho de Jiang e colaboradores é inserir nessas interfaces de serviço informações sobre a estruturação genérica da GUI e as relações entre seus componentes de interface e as estruturas de dados do serviço. Através dessa definição genérica, vários serviços clientes podem oferecer interfaces com o usuário em tecnologias distintas e atendendo automaticamente às alterações realizadas no serviço fornecedor.

Uma abordagem diferente para representação genérica de interface, baseada em ontologias, é proposta por Luna *et al.* (2009). Nesse trabalho, é descrito um meta-modelo baseado em ontologias para a representação de componentes de interface e suas relações para a geração de aplicações web. Além da própria estrutura visual, ações relacionadas à GUI são descritas no meta-modelo. A partir dessa descrição genérica, torna-se viável a construção de aplicações web adaptáveis a mudanças.

2.5 Considerações finais

Neste capítulo foram apresentados os conceitos e tecnologias principais, como a conceituação relacionada a interfaces gráficas e a descrição dos principais elementos constituintes da linguagem OWL, necessários ao embasamento do trabalho proposto. A partir da análise de trabalhos relacionados à interface com o usuário e aos registros eletrônicos em saúde, pudemos enxergar melhor pontos fracos em cada uma das abordagens e questões que não foram ainda exploradas. Ficou clara, desta forma, a carência por um método mais eficaz de geração da interface, que seja mais facilmente configurável e que tenha como foco a entrada de dados.

Capítulo

3

Arquitetura do MedViewGen

> "Eu acredito demais na sorte. E tenho constatado que, quanto mais duro eu trabalho, mais sorte eu tenho"
>
> *Thomas Jefferson*

Este capítulo trata das questões de projeto e da definição de uma arquitetura para o MedViewGen. Para compreender com clareza o contexto e as características da ferramenta descritas no capítulo, é necessário elencar alguns pontos externos ao escopo deste trabalho. A seção 3.1 trata justamente desses pontos, descrevendo em alto nível a arquitetura do OpenCTI e aprofundando um pouco nas características do EHR que são essenciais ao funcionamento do MedViewGen. Na seção 3.2 apresentaremos a contribuição do presente trabalho, especificando o modelo arquitetural do MedViewGen em termos dos requisitos e dos componentes de software estabelecidos e, por fim, na seção 3.3 apresentaremos as considerações finais do capítulo.

3.1 OpenCTI

O OpenCTI é uma iniciativa da UFPB (Universidade Federal da Paraíba), por meio do LArqSS (Laboratório de Arquitetura e Sistemas de Software), laboratório vinculado ao Departamento de Informática e da UTI (Unidade de Terapia Intensiva) do HULW (Hospital Universitário Lauro Wanderley), financiado pela FINEP (Financiadora de Estudos e Projetos), que prevê, nos seus objetivos, a construção de um EHR para apoio à decisão em medicina intensiva. A arquitetura do EHR apresentada nesta seção não pertence ao escopo deste trabalho, porém é essencial para a compreensão das seções subsequentes que tratam da arquitetura do MedViewGen.

Buscando propor uma solução para os problemas inerentes aos registros eletrônicos em saúde, uma arquitetura flexível com a utilização de ontologias em diversos contextos foi proposta para o OpenCTI. O projeto está em desenvolvimento e espera-se, ao final, obter um EHR com flexibilidade do domínio em diversas camadas, desde a definição dos dados até a apresentação dos mesmos para o usuário, e oferecendo infra-estrutura para implementação de técnicas de suporte à decisão clínica e de interoperar com outros EHRs, servindo como base para a telemedicina.

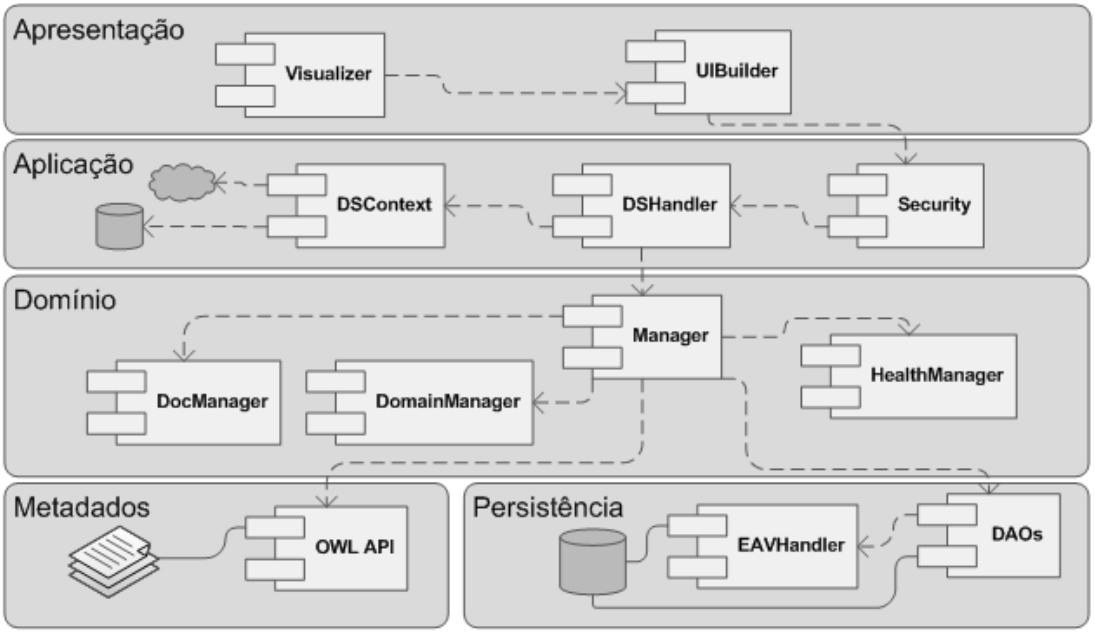

Figura 4: Arquitetura do OpenCTI (Pizzol *et al*., 2010)

A arquitetura projetada para o OpenCTI (**Figura 4**) foi dividida em quatro camadas, cada uma delas com responsabilidades bem definidas. Seguindo um fluxo *botton-up*, começamos com a camada de persistência/metadados, que é responsável por fazer o acesso direto à base de dados, juntamente com os metadados extraídos das especificações OWL. A camada imediatamente acima é a camada de domínio, responsável pela gerência do ciclo de vida dos documentos e das operações referentes ao cuidado ao paciente, servindo como uma fachada para as operações do sistema. A camada de aplicação é responsável por prover a segurança das informações através dos mecanismos de autenticação e controle de acesso, além de oferecer uma arquitetura que possibilite a integração de agentes de CDS ao EHR. Por fim, a camada de apresentação é responsável por gerar a interface para o usuário de forma estática ou dinâmica (dependendo das características dos dados) e fazer a ligação das entidades do domínio com a GUI.

3.1.1 Modelo de Dados

As terminologias biomédicas apresentam grande número de conceitos que são organizados de forma hierárquica. Para que fosse possível usufruir da semântica dos dados armazenados pelo sistema, e assim perceber as relações de um conceito com os demais, a linguagem OWL (Allemang, 2008) foi empregada na modelagem dos metadados do EHR. A utilização de ontologias para a modelagem oferece a flexibilidade necessária para a adaptação às mudanças na definição dos conceitos biomédicos requerida pelo EHR, além de favorecer a correlação dos diversos conceitos modelados em terminologias/ontologias já existentes, visto que cada uma delas possui um propósito específico.

A natureza das informações clínicas não favorece o uso de uma modelagem convencional das entidades e relacionamentos em banco de dados relacionais porque essa requer o conhecimento antecipado, durante o desenvolvimento, dos dados que serão persistidos, seus tipos e seus relacionamentos com outras entidades do sistema. O problema é que as informações clínicas são complexas, com entidades, atributos e relacionamentos variando conforme o local e com o passar do tempo. Por exemplo, informações clínicas de uma UTI pediátrica geralmente não são mesmas que as de uma UTI adulto, numa mesma instituição ou entre instituições. Isto porque cada instituição tem a suas próprias práticas e cultura, pacientes de diferentes categorias (adultos ou crianças) requerem informações específicas. Na modelagem convencional, cada entidade (no caso do EHR, cada conceito biomédico) seria modelada como uma tabela no banco de dados, e cada um de seus atributos como colunas dessa tabela, tornando tal modelagem bastante difícil de ser realizada na prática.

O esquema de modelagem de dados da persistência utilizado no projeto (componente *EAVHandler* da **Figura 4***)* foi o *Entity-Attribute-Value*, ou EAV (Dinu e Nadkarni, 2007). O EAV propõe uma divisão entre dados e metadados, representando cada um destes em uma tabela distinta no banco. No OpenCTI, as entidades e os atributos são representados na ontologia OWL e os dados referentes a estas entidades são armazenados em uma tabela de dados principal, que se utiliza de outras tabelas auxiliares para o armazenamento de tipos de dados diferentes.

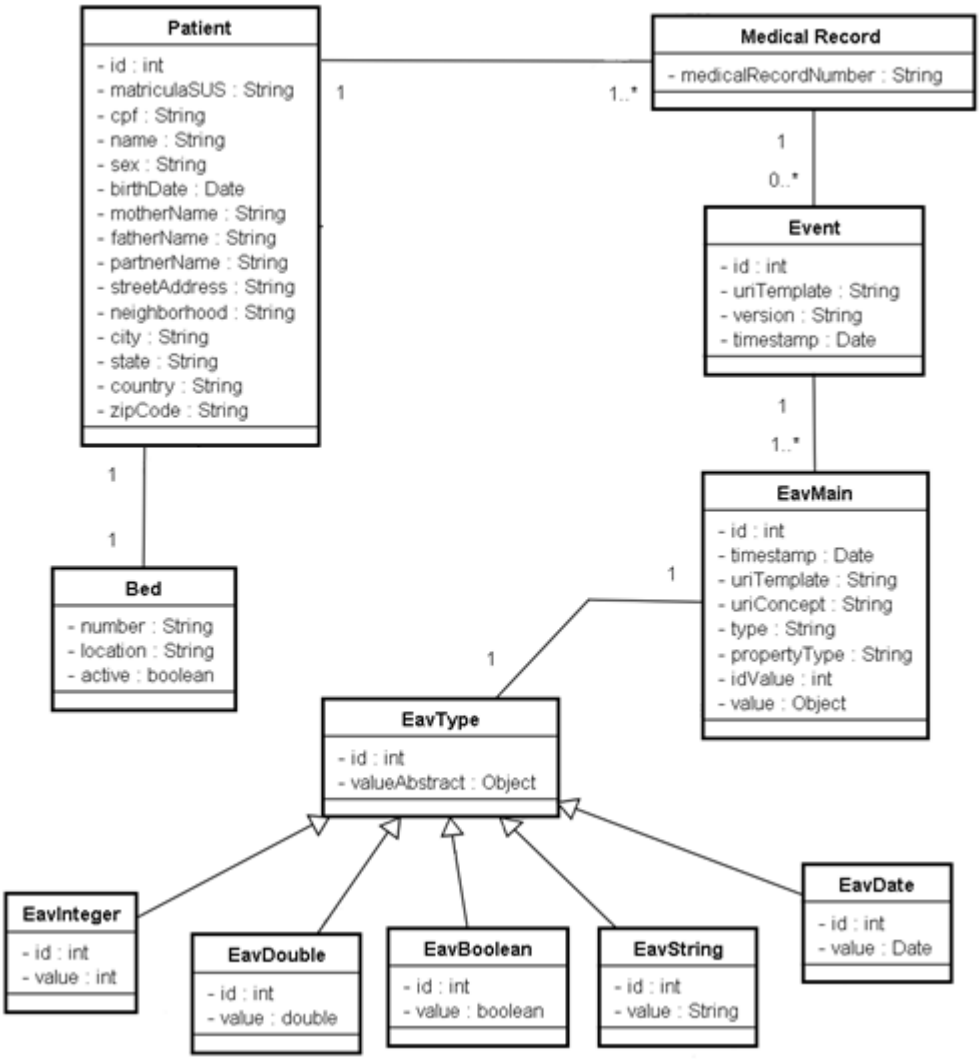

Figura 5: Diagrama de classes do banco de dados EAV do OpenCTI

A estrutura do banco de dados do OpenCTI (Duarte, 2010) está representada na **Figura 5**. Nela podemos destacar a tabela *MedicalRecord*, que armazena os dados do prontuário do paciente; a classe *Patient*, que armazena os dados demográficos do paciente; a classe *Event*, que armazena os dados referentes às fichas (documentos) dos pacientes; e a classe *EAVMain*, que armazena dados dos conceitos biomédicos presentes nas fichas. Temos ainda 5 tabelas auxiliares para armazenar os valores atômicos em cada tipo de dado suportado.

Este esquema de representação dos dados diminui quantitativamente o número de tabelas no banco de dados e possibilita maior flexibilidade a mudanças, uma vez que qualquer inclusão que se deseje fazer na estrutura dos dados pode ser resolvida modificando-se a ontologia biomédica. Os valores para cada atributo, independente do conceito a que se refere, são representados por linhas na tabela de

dados principal (classe *EAVMain*), evitando desta forma que os dados já cadastrados anteriormente contenham valores nulos para os novos atributos.

3.1.2 Documentos

Em registros de saúde armazenados em papel, os conceitos biomédicos costumam ser organizados em documentos direcionados a propósitos específicos. Com o intuito de manter o baixo hiato entre os elementos do domínio real e os objetos da aplicação, a camada de domínio utiliza conceito de documento como um elemento que estrutura conceitos biomédicos de acordo com as necessidades de cada setor e área numa organização de saúde, representando abstratamente um documento utilizado na rotina dos *workflows* organizacionais. As informações neles registradas são organizadas de acordo com os tipos de atividades realizadas por equipes multidisciplinares e multiprofissionais.

Os documentos de saúde, embora possam ser vistos como um agrupamento de conceitos biomédicos, são ortogonais a eles. Um mesmo conceito pode ser utilizado em vários documentos, bem como alterações realizadas sobre a estrutura dos documentos não devem influenciar na base de conceitos biomédicos. Os conceitos biomédicos e os documentos utilizados no sistema são especificados em ontologias. Ambas as ontologias são modeladas na linguagem OWL. Em sua representação, a estrutura dos diferentes documentos de saúde apenas referencia conceitos biomédicos, proporcionando o reuso de conceitos em documentos distintos.

3.1.2.1 Estrutura dos Documentos

As instâncias dos documentos no sistema fazem a junção dos dados com os metadados, desde o momento em que são criados (componente *DocManager* da **Figura 4**) até o ponto em que são assinados digitalmente pelo médico responsável (componente *HealthManager*). A estrutura dos documentos do OpenCTI está representada, de forma ilustrativa, na **Figura 6**.

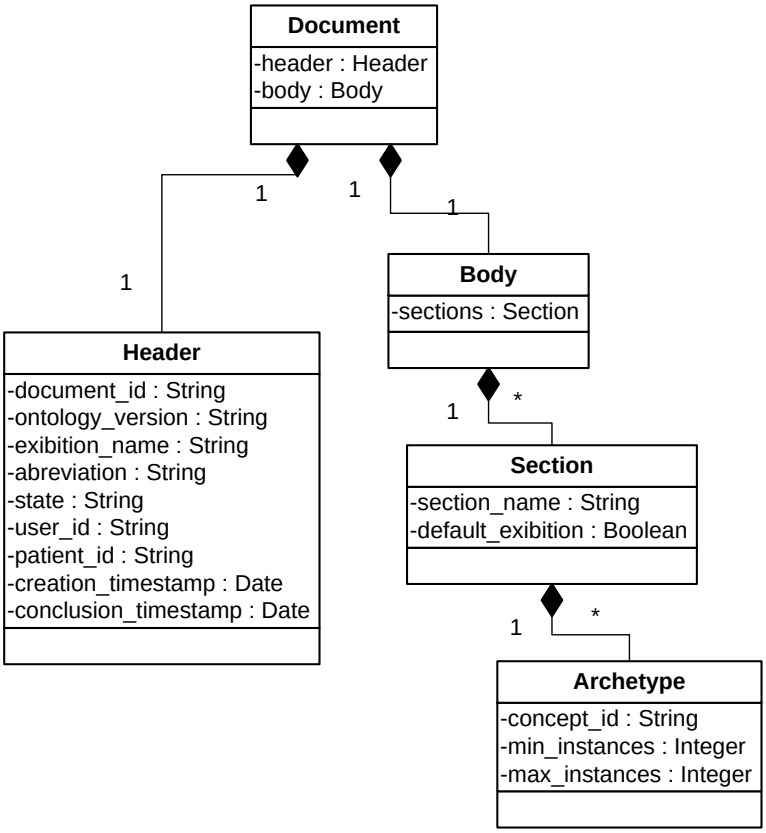

Figura 6: Diagrama de classes da estrutura dos documentos

Apesar de utilizarmos ontologias para a modelagem dos documentos, usamos a notação UML para a representação ilustrativa, com o intuito de facilitar a compreensão por parte do leitor. Essa notação é utilizada na representação de outras ontologias do MedViewGen em seções posteriores. Como mostrado na **Figura 6**, um documento é dividido em duas partes principais. A primeira delas contém informações de cabeçalho (*header*) do documento. O cabeçalho contém informações importantes para o controle dos documentos, como identificador do documento no sistema e a versão da ontologia utilizada, dentre outras. Seus campos são descritos a seguir.

- **Document_id:** Identifica o tipo do documento. Esse identificador é recuperado a partir da URI (*Universal Resource Identifier*) do indivíduo que representa o documento na ontologia de documentos;
- **Ontology_version:** Indica a versão da ontologia utilizada na criação do documento. Este campo é necessário para permitir o controle de versões das ontologias, de modo que um documento criado em outras versões possam ter a estrutura de seu conteúdo recuperada;
- **Exibition_name:** Contém a *string* que representa o nome do tipo do documento que deve ser exibido para o usuário;

- **Abreviation:** Contém uma abreviação do nome de exibição do documento para ser utilizado pela GUI em diversos contextos;
- **State:** Indica o estado atual do documento em seu ciclo de vida. O valor *new* é atribuído no momento em que o documento é criado. No momento em que ele está pronto para a manipulação pelo usuário, o valor *in_use* é atribuído. O estado *saved* indica que o documento está salvo temporariamente, porém não persistido. No momento em que o usuário indica a finalização do documento, seu estado passa a ser *signed* e ele é persistido no banco de dados.
- **User_id:** Indica o identificador do usuário que criou o documento;
- **Patient_id:** Indica o identificador do paciente para o qual o documento foi criado. O documento será, quando finalizado, associado ao prontuário deste paciente;
- **Creation_timestamp:** Indica a data e hora de criação do documento;
- **Conclusion_timestamp:** Indica a data e hora de finalização da edição do documento.

A segunda parte do documento contém o corpo (*body*) do documento. Toda a divisão do documento em seções e a organização dos conceitos no documento estão representadas neste campo. A seguir temos a descrição dos campos que nele estão contidos:

- **Section:** Representa uma seção do documento. Uma seção pode conter outras seções como sub-seções da seção corrente, como forma de melhor organizar o documento. Uma seção contém zero ou mais arquétipos (*archetype*);
- **Section_name:** Contém o nome de exibição da seção na GUI.
- **Default_exibition:** Algumas seções do documento permanecem invisíveis para o usuário, a menos que o mesmo indique que deseja exibi-la. Este campo indica se a seção deve ser exibida por padrão.
- **Archetype:** Elemento que faz a ligação entre o documento e um conceito biomédico. Cada conceito em um documento é representado por um arquétipo. É este elemento que possibilita a utilização de um mesmo conceito em diversos documentos.
- **Concept_id:** Identificador do conceito que o arquétipo representa no documento.
- **Min_instances:** Número mínimo de instâncias do conceito que devem estar presentes no documento.

- **Max_instances:** Número máximo de instâncias do conceito que devem estar presentes no documento.

Essa estrutura de documentos é representada na forma de ontologia. Para que fosse possível encapsular esses dados de forma que pudessem ser manipulados pelo sistema em tempo de execução, foi definida a classe (em Java) *HealthDocument*, e sua estrutura é descrita na próxima seção.

3.1.2.2 HealthDocument

A classe *HealthDocument* (**Figura 7**) é uma classe genérica capaz de representar qualquer documento do OpenCTI. Como apresentado na seção anterior, a estrutura de documentos na ontologia possui um conjunto de elementos de forma a poderem representar um documento em saúde. Porém, à medida que os usuários interagem com o sistema, novos documentos podem surgir e essa estrutura pode sofrer algumas alterações. Por exemplo, além dos campos *Header* e *Body*, um campo *Footer* (rodapé) pode ser adicionado à estrutura para conter alguns dados não comportados pela atual estrutura.

A classe *HealthDocument* deve ser flexível o suficiente para comportar essas alterações na estrutura dos documentos sem a necessidade de re-implementação e re-compilação da classe. Para concretizar isto, foi utilizada como base a própria estrutura ontológica da OWL para a representação de seus conceitos, fazendo uma analogia dos elementos orientados a objetos da classe com os elementos orientados a ontologias da linguagem OWL.

De acordo com o apresentado na seção 2.3, toda classe da OWL herda da classe genérica *Thing*. Cada instância de uma determinada classe é denominada *indivíduo*, onde cada um desses indivíduos está relacionado a outros indivíduos por meio de *ObjectProperties*. Há ainda os *DataProperties*, que representam os atributos dos indivíduos de uma determinada classe.

De forma análoga às ontologias, foram definidas três classes orientadas a objeto para a representação desses conceitos. São elas: *Thing*, *Relationship* e *DataProperty*.

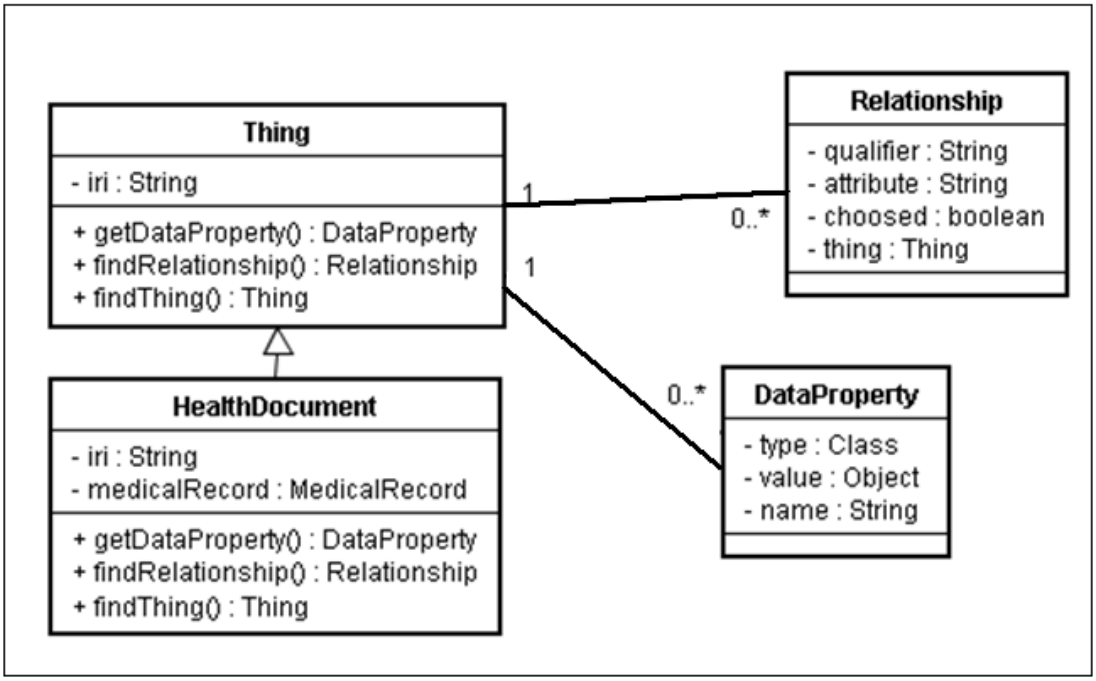

Figura 7: Diagrama de classes da representação genérica da ontologia no OpenCTI

A **Figura 7** apresenta o diagrama de classes com a definição de cada uma delas. A classe *Thing* representa analogamente a classe *Thing* das ontologias OWL. Cada objeto Java dessa classe representa um indivíduo da ontologia. Em *Thing* tem-se um campo chamado *iri* do tipo String, que guarda a IRI (*Internationalized Resource Identifier*) do indivíduo. Essa classe contém ainda dois conjuntos, um de *Relationship* e outro de *DataProperty*, representando os relacionamentos e propriedades de dados do indivíduo.

Na classe *Relationship*, temos os campos *qualifier*, *attribute*, *choosed* e *thing*. Os campos *attribute* e *qualifier* representam o nome e o tipo, respectivamente, do relacionamento. O campo *choosed* é um valor booleano que indica se o *Relationship* está selecionado ou não. O campo *thing* é o conceito com o qual o outro conceito que contém *Relationship* se relaciona. Por exemplo, um objeto da classe *Thing* que representa o conceito *Ficha_de_evolucao_medica* tem como campo um objeto da classe *Relationship* que contém como campo um outro objeto da classe *Thing* representando o conceito *Corpo_da_ficha_de_evolucao_medica*, indicando que o invidíduo *Ficha_de_evolucao_medica* da classe *Document* na ontologia possui relacionamento *hasPart* com o indivíduo *Corpo_da_ficha_de_evolucao_medica* da classe *Body*.

A classe *Thing* pode representar qualquer conceito de qualquer ontologia. A classe *HealthDocument* é uma especialização da classe *Thing* para a representação

exclusiva de documentos. Além dos campos e métodos da super-classe, a classe *HealthDocument* contém um campo que identifica a qual prontuário o documento pertence (*medicalRecord*). Possui também métodos extras especializados na busca de conceitos em um documento (*findThing()*).

3.1.3 Suporte a Decisão Clínica

Além de lidar com os aspectos de autorização e autenticação (componente *Security*), a camada de aplicação também trata dos aspectos relativos ao suporte a decisão clínica ou CDS. Esse suporte é realizado por meio de um uma ontologia de CDS que permite definir e integrar diferentes componentes, como alertas, validadores e lembretes, oferecendo um módulo de CDS para ser utilizado por diferentes especialidades, áreas profissionais, setores e/ou organizações (Pizzol *et al.*, 2010).

Para tal, é oferecido um meio de entrada dos dados que serão analisados (informação contextual e o histórico do paciente) (Wright *e Sittig*, 2008), um meio para persistir os dados específicos do CDS e um meio de comunicação para a interoperabilidade com outros sistemas (componente *DSContext*). Dessa maneira, elabora-se um ambiente de CDS ortogonal aos conceitos biomédicos e que se adapta às mais diferentes necessidades.

3.1.4 Interface do Usuário

A camada de apresentação é dividida em dois módulos principais. O módulo estático, responsável pela apresentação dos elementos estáticos do domínio e pelo fluxo de navegação, e o módulo dinâmico, encarregado de gerar e gerenciar a GUI dos elementos dinâmicos do sistema, como os documentos.

Para o módulo estático, foi realizado um projeto de interface no esquema convencional. Para cada funcionalidade, é sabida de antemão a estruturação dos dados que devem ser entrados pelo usuário bem como dos que devem ser exibidos. O cadastro de pacientes, por exemplo, é um exemplo dessas funcionalidades. A estrutura das entidades envolvidas (neste caso, o paciente) é sabida durante o desenvolvimento do sistema, sofrendo poucas alterações durante o ciclo de vida do software. Desta forma, a criação de uma interface gráfica para a o cadastro de paciente pode ser realizada sem maiores problemas, uma vez que esta tem um caráter mais permanente.

Para apresentação dos documentos do OpenCTI, é necessária a existência de uma modelagem genérica de interfaces que sejam geradas de modo automático de acordo com o domínio da aplicação. Essa geração automática de interface incorpora

informações específicas da situação corrente do usuário (contexto do usuário) no design da interface e separa a interface do usuário da lógica de aplicação.

Para tornar isso possível, uma API da camada de apresentação fica responsável por gerar a interface em tempo de execução, com base em características extraídas da ontologia de conceitos biomédicos e de documentos de saúde. Essa API faz parte da ferramenta MedViewGen, proposta nesta dissertação. O MedViewGen elabora inicialmente um formato de interface genérica de entrada e/ou saída que será traduzida em componentes utilizados na construção da apresentação final para o usuário. Para cada tipo de visualizador uma interface diferente é gerada de acordo com as tecnologias específicas utilizadas.

Com a utilização de uma ontologia de componentes de interface, constrói-se um método de design que é independente dos ambientes de aplicação específicos, propiciando facilidade de manutenção e reuso dos componentes de aplicação e apresentação. Como resultado, espera-se poder utilizar visualizadores baseados em tecnologias diferentes para representar o mesmo conjunto de conceitos biomédicos. A arquitetura da ferramenta é descrita em detalhes na seção 3.2 a seguir.

3.2 Modelo Arquitetural do MedViewGen

Esta seção apresenta a arquitetura do MedViewGen em termos da definição de seus requisitos e de uma solução conceitual para o problema. O conteúdo desta seção é contribuição original deste trabalho, e terá os detalhes relativos aos modelos semânticos e algoritmos utilizados apresentados nos próximos capítulos. Na seção 3.2.1, faremos uma decomposição do problema gerando requisitos a serem satisfeitos, detalhando os objetivos da ferramenta. A seção 3.2.2 descreve de modo geral o processo de geração automática da interface. Por fim, nas seções 3.2.3 e 3.2.4 é descrito um modelo da arquitetura do MedViewGen através da especificação dos componentes de software utilizados na solução, de suas interações e do modelo semântico utilizado.

3.2.1 Características

É importante lembrar que o MedViewGen é uma ferramenta com foco no desenvolvedor. Seu principal objetivo é facilitar o trabalho dos desenvolvedores projetistas de interfaces para EHR, aliviando-os da tarefa de modelar cada formulário para cada tipo de documento que o sistema suporta, e poupá-los das constantes atualizações que as mudanças no domínio requerem.

A arquitetura do MedViewGen deve ser projetada de forma que seja reusável em diversos contextos. Além de atribuir componentes de interface para elementos de domínio de forma automática, deve ser flexível o suficiente para se adaptar a diversos domínios de aplicação em EHR e independente de tecnologia de interface utilizada.

3.2.1.1 Atribuição Automática de Componentes

Como vimos na seção 1.1.1, o domínio da aplicação varia de acordo com o ambiente onde o EHR é estabelecido. Conceitos biomédicos, tipos de documentos e agentes de CDS são exemplos de elementos do domínio que variam com o tempo e de acordo com o contexto organizacional, inviabilizando a utilização de interfaces convencionais (estáticas). Algumas abordagens na literatura, descritas na seção 2.4.1, promovem interfaces flexíveis através da configuração dos mapeamentos entre os elementos de domínio e os componentes de interface de forma manual.

O presente trabalho automatiza esse processo de mapeamento. Com a utilização de um modelo de informação baseado em metadados referentes a cada componente de interface, o MedViewGen deve construir, em tempo de execução, um formulário para a representação de um documento clínico requisitado pelo usuário. Para cada elemento de domínio pertencente ao documento, um ou mais componentes de interface devem ser atribuídos, proporcionando a ligação entre as estruturas de dados e a GUI, de modo que o usuário possa realizar entrada de dados clínicos no prontuário do paciente e visualizar dados clínicos já persistidos no EHR de forma organizada e coerente.

3.2.1.2 Independência de Tecnologia de Interface

Na seção 1.1.1, vimos a crescente demanda por novas formas de visualização e interação com sistemas eletrônicos e o conseqüente desenvolvimento de novas tecnologias, tanto na parte de software quanto de hardware. O trabalho proposto tem como um de seus objetivos possibilitar a interação do usuário independentemente de qual tecnologia tenha sido utilizada no desenvolvimento da interface gráfica. Deve ser estabelecida uma arquitetura fracamente acoplada, na qual módulos de apresentação em tecnologias distintas possam ser integrados. Independentemente da linguagem de programação ou dispositivo utilizados, o MedViewGen deve ser capaz de construir formulários para a visualização de documentos e possibilitar interação com ferramentas de apoio à decisão clínica (CDS).

3.2.1.3 Flexibilidade dos mapeamentos

Elementos de domínio são elementos flexíveis que mudam ao longo do tempo. Para a exibição correta desses elementos variáveis, algumas alterações na forma como o mapeamento é realizado pode ser requerida. Uma combinação de componentes de interface existentes ou a utilização de um novo componente de interface desenhado exclusivamente para determinado elemento podem ser requeridas, por exemplo. A utilização de diversas tecnologias de interface também é um fator variável. Alguns ambientes utilizam monitores sensíveis ao toque e interfaces mais reativas, por exemplo, para a interação com o usuário, enquanto outras dispõem apenas de interfaces web padrão, acessíveis a partir de terminais convencionais.

Com a utilização de mapeamentos estáticos definidos em código fonte, alterações na forma como os mapeamentos ocorrem acarretaria em reprogramação da GUI a cada novo elemento inserido no domínio. O MedViewGen deve proporcionar um esquema flexível de mapeamentos, onde os modeladores de domínio possam personalizar o EHR de acordo com o ambiente onde ele está inserido, sem a necessidade de programação, compilação e outras atividades requeridas por um mapeamento *hard-coded*.

3.2.1.4 Integração com o CDS

Assim como os conceitos biomédicos pertencentes aos documentos, os elementos de CDS são caracterizados como elementos de domínio, e podem requerer exibição de informações para o usuário e execução de eventos relacionados ao propósito do agente de CDS referente. Durante o processo de geração de interface, componentes de interface devem ser atribuídos com o intuito de exibir essas informações.

Porém, apenas a atribuição de componentes não é suficiente para este tipo de elemento. Uma ligação entre a GUI gerada e o módulo de CDS deve existir para tornar possível a interatividade que os agentes de CDS proporcionam. Os agentes de CDS são projetados para fornecer informações importantes ao usuário na medida em que ele interage com o sistema. Enquanto o usuário interage com a interface gráfica, eventos como entrada de valores, eventos de teclado e eventos de mouse, por exemplo; devem ser capturados pela camada de apresentação e repassados ao módulo de CDS. Após processar os dados recebidos, o módulo de CDS deve enviar a resposta de volta à camada de apresentação e então exibida para o usuário. Para isso, deve haver uma integração entre a GUI e o módulo de CDS, proporcionando a captura e tratamento de eventos originados pelo usuário.

3.2.2 Geração Automática da GUI

A geração automática da GUI é o processo de atribuir os componentes adequados para a apresentação de um conjunto de elementos de domínio na construção do esquema final de interação do usuário com os documentos. No MedViewGen, esse processo é dividido em três etapas, representadas na **Figura 8**.

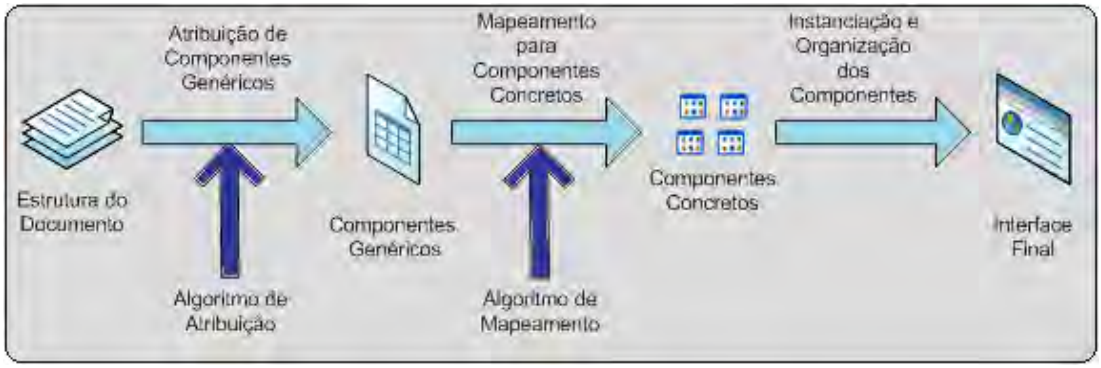

Figura 8: Processo de Atribuição de Componentes

A primeira etapa refere-se ao processo de geração dos mapeamentos entre elementos de domínio e componentes genéricos de interface. Um componente genérico descreve o comportamento de determinado tipo de componente, especificando características tais como os tipos de dados suportados, eventos a que atende, tipo de interação, se o componente é de entrada e/ou saída, se o componente é um *container*, dentre outras. Eles contêm informações discriminantes relativas à apresentação do componente, porém sem especificar em qual tecnologia de interface o componente deve ser instanciado. Por exemplo, um componente genérico *inputText* contém informações relativas a um componente de entrada de texto, porém não especifica que este deve ser um campo de texto em JSF (Java *Server Faces*), Java *Swing* ou outra tecnologia. Qualquer componente que satisfaça às características de um componente genérico poderá ser uma instância concreta desse componente genérico na interface do usuário.

Para que o processo referente à primeira etapa ocorra, deve haver um algoritmo de atribuição bem definido que irá receber como entrada um documento, processar as informações relevantes e gerar como saída uma árvore de componentes genéricos. Com base em metadados relativos aos componentes de interface, o algoritmo percorre cada elemento do documento e lhe atribui o melhor componente para a representação do elemento na GUI. Esse algoritmo é descrito em detalhes na seção4.1.2.

A segunda etapa é responsável pela tradução do componente genérico (abstrato) em um componente concreto de tecnologia de interface específica. Um componente concreto é uma especialização de um componente genérico para determinada tecnologia. Um componente *JTextField*, por exemplo, pode ser mapeado como um componente concreto correspondente ao componente genérico *inputText* para a tecnologia Java Swing. Para cada componente abstrato deve existir pelo menos um componente concreto correspondente de uma tecnologia de interface, incluindo os seus respectivos atributos e eventos. Supondo que existam n implementações de visualizadores em tecnologias de interface distintas, para cada componente genérico devem existir n componentes concretos, um para cada tecnologia, proporcionando assim que a aplicação seja acessível a partir de diversos tipos de visualizadores, possivelmente em dispositivos distintos.

Assim como a atribuição de componentes genéricos, o processo de mapeamento também requer a definição de um algoritmo. O algoritmo de mapeamento recebe como entrada uma árvore de componentes genéricos, processa os mapeamentos de componentes, atributos e eventos; e gera como saída uma árvore de componentes concretos. A descrição detalhada do algoritmo encontra-se na seção 4.2.2.

Na terceira e última etapa, o processo de geração de interface é finalizado. Uma vez atribuída a árvore de componentes de interface necessários para a representação dos elementos do documento, cada um desses componentes deve ser instanciado. Seus atributos, especificados nas etapas anteriores, trarão informações necessárias para o estabelecimento da ligação entre o elemento de domínio e o componente, para o posicionamento do componente no formulário e para o ajuste de algumas características intrínsecas de cada componente especificamente. Dessa forma, obtemos, ao final do processo, o formulário que possibilitará a interação do usuário com um documento do sistema.

3.2.3 Modelo de Componentes do MedViewGen

Na seção 3.2.2, descrevemos o processo de geração automática de GUI em alto nível. Em termos de organização do processo, podemos dividi-lo em algumas etapas principais, facilitando o entendimento por meio da quebra de um problema de maior complexidade em problemas menores e de melhor compreensão. Em termos de software, cada etapa da geração deve ser atribuída como responsabilidade de um componente, para que em conjunto, consigam realizar o processo de forma eficaz. Definimos nesta seção a divisão da ferramenta em componentes de software, definindo suas respectivas atribuições e as interações que devem ocorrer entre eles du-

rante o processo. A **Figura 9** ilustra a arquitetura do MedViewGen e as interações entre os componentes em um cenário típico de geração automática para visualizar um documento.

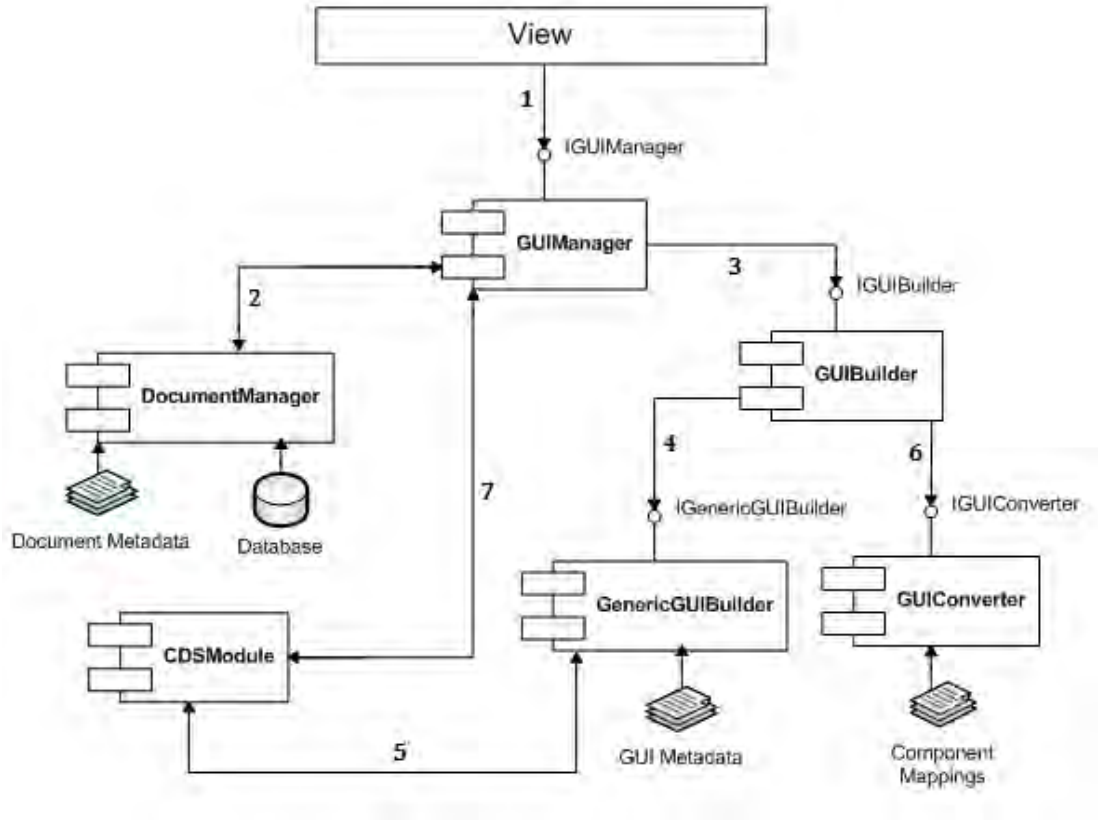

Figura 9: Modelo de componentes do MedViewGen

Na **Figura 9** estão representad0s os componentes necessários para que a ferramenta realize suas tarefas. Os componentes *DocumentManager* e *CDSModule* são componentes externos à ferramenta, porém essenciais ao processo. Os componentes *GUIManager*, *GUIBuilder*, *GenericGUIBuilder* e *GUIConverter* são componentes internos da ferramenta. Como estamos falando em termos de componentes de software, é necessário que existam interfaces bem definidas para cada um dos componentes internos. Essas interfaces estão descritas no Apêndice B.

O processo de geração começa a partir de uma requisição de visualização e/ou edição de um documento pelo usuário **(1)**. A requisição originada na GUI (componente *View*) é encaminhada ao componente gerenciador da GUI (*GUIManager*), responsável por gerenciar o processo de geração automática do formulário **(3)** e por manter o estado atual dos documentos e informações do contexto do usuário, recebendo eventos da GUI e repassando-os aos componentes responsáveis. Com base nos metadados do documento requisitado **(2)**, recuperados pelo gerenciador de do-

cumentos (*DocumentManager*), nas informações dos componentes de interface expressas na ontologia de GUI e nas informações relativas aos elementos de CDS implementados, o componente de geração automática (*GenericGUIBuilder*) elabora um formato de interface genérico para a apresentação do documento **(4)**. A estrutura da interface genérica completa é repassada para o *GUIConverter* **(6)** que traduz os componentes genéricos para componentes da tecnologia específica do visualizador do usuário**,** que são posteriormente utilizados na construção do formulário final (*GUIBuilder*). Após gerado, o formulário é retornado para ser integrado à GUI. À medida que o usuário interage com o sistema, alguns eventos podem ser lançados devido à presença de agentes de apoio a decisão clínica. Esses eventos são capturados pelo *GUIManager* e repassados para o módulo de CDS **(7)**, representado pelo componente *CDSModule*, que processará os dados e enviará uma resposta ao usuário.

Como podemos ver, alguns componentes da arquitetura do MedViewGen interagem com componentes externos para a realização de seu trabalho. Mostraremos a seguir uma descrição mais detalhada de cada componente e suas responsabilidades:

3.2.3.1 View

Este é um componente externo ao MedViewGen implementado em tecnologia de interface específica, podendo existir diversas implementações deste componente integrados à ferramenta, como descrito em um dos requisitos do MedViewGen na seção 3.2.1.2. A interação com o usuário é feita através deste componente, que por sua vez repassa parte dos eventos gerados para o componente *GUIManager*, que se encarregará do processamento desses eventos.

3.2.3.2 GUIManager

O componente *GUIManager* é responsável por gerenciar o processo de geração automática e por capturar eventos originados no formulário gerado. Durante a geração automática, ele recebe as solicitações originadas no componente *View*, recupera o documento solicitado (dados e metadatos) e envia para o componente *GUIBuilder*, que retornará o formulário construído. Esse formulário é então retornado pra o componente *View*. Durante a interação do usuário com o formulário, o *GUIManager* fica responsável por capturar os eventos gerados e repassar para os agentes de CDS responsáveis, que por sua vez processarão os dados do evento e, dependendo do resultado, invocarão alguma ação da GUI via *GUIManager*. Deve existir apenas uma implementação da interface *IGUIManager*.

3.2.3.3 DocumentManager

O *DocumentManager* é um componente externo ao MedViewGen e utilizado pelo *GUIManager* para a recuperação de documentos do sistema, tanto para documentos novos quanto para documentos persistidos na base de dados. Este é o único ponto de acesso aos dados do sistema. A especificação de sua interface não faz parte do escopo deste trabalho.

3.2.3.4 GUIBuilder

Este componente é responsável por construir um formulário de entrada e/ou saída para um documento. Ele recebe a solicitação de geração do componente *GUIManager*, juntamente com os metadados dos documentos e as informações relativas ao visualizador do usuário. Com base nos metadados extraídos do documento solicitado, o *GUIBuilder* solicita de outros componentes (*GenericGUIBuilder* e *GUIConverter*) que executem o algoritmo de atribuição de componentes de interface e o mapeamento para tecnologia do usuário corrente. Uma árvore de componentes de interface é gerada. O *GUIBuilder* fica responsável então por instanciar os componentes da árvore e inicializar os seus atributos. Em termos de implementação do componente, para cada tecnologia de interface suportada pelo EHR deve haver uma implementação da interface *IGUIBuilder*.

3.2.3.5 GenericGUIBuilder

Este componente é responsável por gerar, a partir dos metadados extraídos de um documento, uma árvore de componentes de interface genéricos. Este tipo de componente é descrito na seção 3.2.2. Deve existir apenas uma implementação da interface *IGenericGUIBuilder*. O algoritmo de atribuição automática de componentes descrito na seção 4.1.2 é executado com base nos metadados dos documentos e nos metadados relativos aos componentes de interface. Este segundo grupo de metadados é recuperado de uma ontologia de GUI que contém as informações necessárias para a atribuição dos componentes para cada elemento de domínio pertencente ao documento. Ao final da execução do algoritmo, a árvore de componentes é retornada para o componente *GUIBuilder*.

3.2.3.6 GUIConverter

Este componente é responsável por traduzir a árvore de componentes genéricos em componentes concretos. Deve haver apenas uma implementação da interface *IGUIConverter*. Componente concreto é o tipo de componente que é dependente de pla-

taforma, ou seja, é específico para determinada tecnologia. Cada componente genérico, para que seja instanciado e exibido para o usuário, deve ser traduzido em um componente concreto.

No momento da tradução, os dados relativos à tecnologia do visualizador do usuário são enviados juntamente com a árvore de componentes genéricos na solicitação de mapeamento. Cada componente genérico deve ser mapeado para um componente da tecnologia especificada nesses dados. Ao final do mapeamento, uma árvore de componentes concretos é retornada ao componente *GUIBuilder*.

3.2.3.7 CDSModule

O *CDSModule* é um componente externo ao MedViewGen utilizado pelo *GenericGUIBuilder* para fornecer informações a respeito dos agentes de CDS implementados. Suas interfaces não fazem parte do escopo desse trabalho. Enquanto os conceitos biomédicos do documento são percorridos para a atribuição de componentes, o *GenericGUIBuilder* verifica se existem agentes de CDS ativos para o cada conceito, e em caso positivo atribui componentes adicionais e/ou eventos para tornar possível a ativação do CDS para o usuário

Depois que o processo de geração automática da GUI é finalizado, o usuário, por meio de sua interação com o sistema, pode ativar alguns eventos associados ao suporte à decisão, como a validação online de campos do formulário, por exemplo. Esses eventos são capturados pelo componente *GUIManager* e repassados ao módulo de CDS. Cabe ao *CDSModule* processar os dados do evento e devolver uma resposta ao usuário, por meio da realização de alguma ação na interface do usuário.

3.2.4 Modelagem Semântica

Para que a geração de formulários de forma automática seja possível, é necessário estabelecer um esquema de organização de metadados de forma que seja possível extrair semântica a partir desse esquema. Como podemos observar na **Figura 9**, os componentes *GenericGUIBuilder* e *GUIConverter* acessam metadados para que suas respectivas tarefas sejam realizadas com sucesso. Esses metadados contêm informações organizadas de forma a dar um sentido a elas, tornando possível o processo de atribuição e mapeamento de componentes.

No caso do *GenericGUIBuilder*, para a criação automática dos mapeamentos entre os elementos presentes em um documento e seus respectivos componentes de apresentação, é preciso ter conhecimento sobre quais componentes são mais adequados para cada tipo de conceito. Essas informações devem estar descritas em al-

gum lugar, de forma que possam ser alteradas e que sua semântica possa ser recuperada computacionalmente. A utilização de um modelo ontológico de componentes de interface, descrito na linguagem OWL, possibilita a automatização desse processo. Apresentaremos em capitulos subseqüentes os detalhes do modelo ontológico estabelecido.

É importante ressaltar a diferença entre a definição do modelo ontológico e a criação de uma ontologia. O modelo ontológico estabelecido indica a estruturação dos dados que uma ontologia possui. É o modelo ontológico quem define a semantica da iinformação contida em uma ontologia. Uma ontologia pode ser vista como uma instanciação de um modelo ontológico, ou seja, a estrutura juntamente com os dados nela descritos. Para um determinado modelo podemos ter diversas ontologias, cada uma com informações distintas, porém descritas em termos de uma mesma estrutura.

O modelo ontológico estabelecido para componentes de interface deve possibilitar uma descrição formal dos componentes da GUI, incluindo informações (metadados) relativas a quais tipos de conceitos biomédicos cada componente está habilitado a representar. Metadados relativos a tipos de dados suportados, características de entrada e/ou saída de dados, possibilidade de inclusão de componentes filhos (container), dentre outros, devem ser especificados.

Para o componente *GUIConverter*, temos a utilização dos mapeamentos entre componentes genéricos e componentes concretos. A construção do mapeamento entre componentes genéricos e concretos não é tão simples quanto a definição de pares entre os dois tipos de componentes. A estrutura de cada componente, incluindo seus atributos e eventos, deve também ser representada e mapeada. Por ser uma abordagem eficaz na representação de conceitos hierárquicos, resolvemos descrever um modelo ontológico também para a representação desse mapeamento.

3.3 Considerações Finais

Apresentamos nesse capítulo a arquitetura do MedViewGen. A descrição de alto nível do processo de geração serve como base para o entendimento do processo como um todo e para o esclarecimento de alguns conceitos utilizados nos próximos capítulos. Com a visão geral da arquitetura demonstrada na seção 3.2.3, pudemos enxergar melhor os relacionamentos entre os componentes da ferramenta e ter uma base de conhecimento para uma melhor compreensão dos detalhes apresentados a seguir, no capítulo 4.

Capítulo

4

Modelagem Semântica e Algoritmos de Tradução

"Eu acredito demais na sorte. E tenho constatado que, quanto mais duro eu trabalho, mais sorte eu tenho"

Thomas Jefferson

No capítulo 3 apresentamos uma visão geral da arquitetura do MedViewGen e de como seus componentes interagem uns com os outros. Contudo, não especificamos as ontologias e algoritmos utilizados no processo. A modelagem semântica é uma das principais contribuições desse trabalho, pois estabelece a infra-estrutura necessária para que tanto a geração de formulários dinâmicos quanto a integração com ferramentas de CDS.

Este capítulo traz aspectos relativos aos modelos ontológicos utilizados na atribuição de componentes genéricos e no mapeamento para componentes concretos, bem como os algoritmos utilizados no processamento desses modelos. Na seção 4.1 temos a descrição do algoritmo de atribuição de componentes genéricos e da estruturação da ontologia de interface, que estão relacionados ao processo de atribuição de componentes genéricos de *GenericGUIBuilder* (ver **Figura 9**). Na seção 4.2 descrevemos o modelo ontológico e o algoritmo utilizados em *GUIConverter* no processo de tradução dos componentes genéricos em componentes concretos. Por fim, na seção 4.3, descrevemos em detalhes como é realizada a integração da GUI com o módulo de CDS do OpenCTI.

4.1 Atribuição de Componentes Genéricos

De acordo com a arquitetura apresentada no capítulo 3, quando o usuário gera uma requisição para a geração de um formulário para um documento, a primeira e uma das principais etapas do processo de geração automática é o processo de atribuição de componentes de interface genéricos para cada elemento de domínio, realizada pelo componente *GenericGUIBuilder* (ver **Figura 9**). Como vimos na seção 3.2.2, um componente genérico representa de forma abstrata um componente de interface, através da descrição de suas características como formato de entrada de dados, possíveis formas de interação do usuário, eventos relacionados, etc. Porém, essa representação abstrata não entra em detalhes de implementação específicos de cada tecnologia. Um campo de entrada de texto, por exemplo, é um campo de entrada de texto independente de qual tecnologia esteja implementado. Ele deve permitir que o usuário digite uma seqüência de caracteres que sirvam como entrada para processamento em software. Um componente de interface implementado que satisfaça essa característica é um campo de entrada de texto.

A seguir, veremos como estão estruturados dos metadados em um modelo ontológico de componentes de interface. Na seção 4.1.2 temos a definição do algoritmo de atribuição de componentes utilizado em *GenericGUIBuilder* e dos os metadados necessários para sua realização.

4.1.1 Modelo Ontológico para Atribuição de Componentes Genéricos

Como vimos na seção 3.2.4, o MedViewGen emprega um processo automático de atribuição de componentes para os elementos presentes em um documento, tendo por base uma ontologia de interface contendo os metadados necessários ao processo. A **Figura 10** contém uma representação em UML da estrutura da ontologia. Vale ressaltar que utilizamos a notação UML para representar o modelo ontológico com o intuito de facilitar a compreensão, porém a modelagem da ontologia é feita com base nos conceitos e características da linguagem OWL.

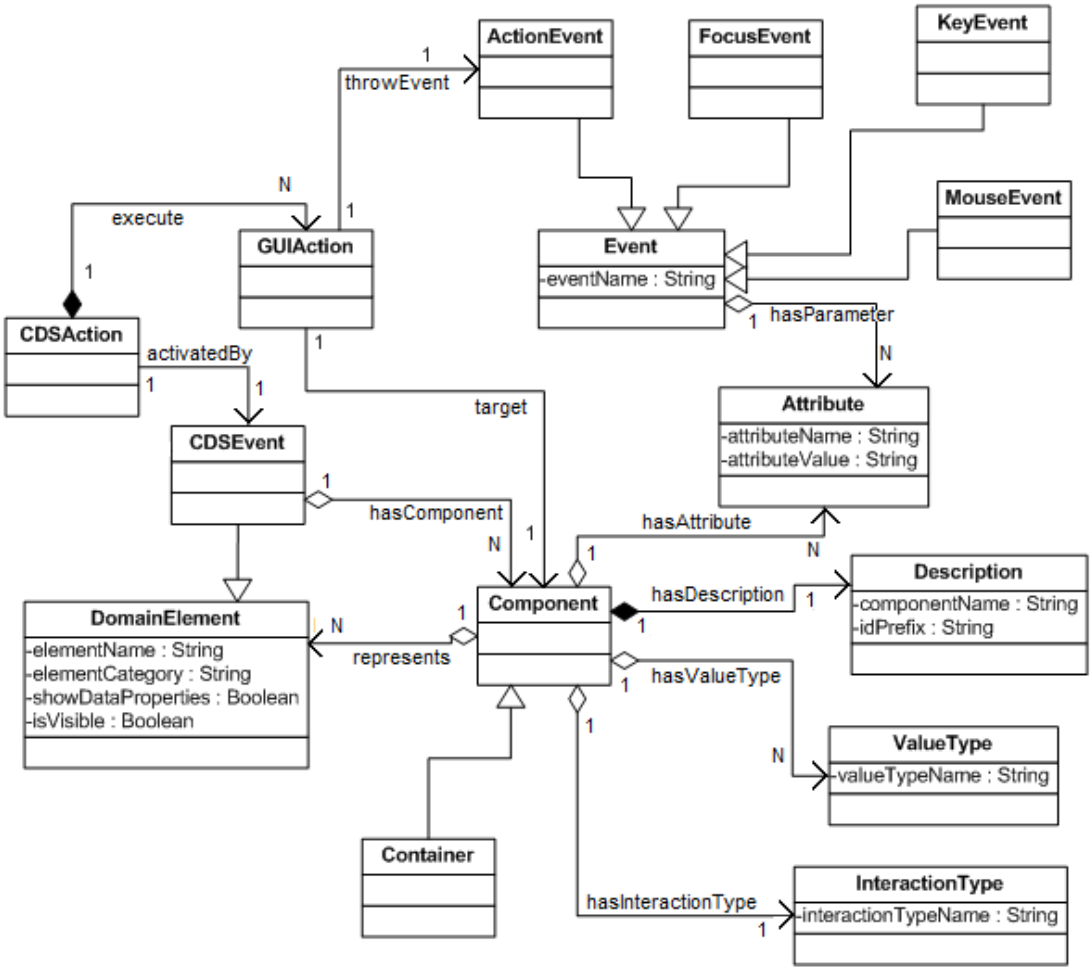

Figura 10: Estrutura da Ontologia de Interface Genérica

Como vimos na seção 2.3, ontologias OWL são formadas por indivíduos relacionados entre si pelos *objectProperties*. Cada indivíduo é uma instância de pelo menos uma classe da ontologia. A estrutura da **Figura 10** representa as classes da ontologia e os relacionamentos que elas têm umas com as outras. A classe principal da ontologia, como era de se esperar, é a classe *Component*, que contém a representação de um componente de interface. A classe *Container* é uma especialização da classe *Component*, para os componentes de interface que são contêineres. As demais classes da ontologia são classes auxiliares, que contêm, cada uma delas, metadados que expressam características distintas relativas aos componentes. Algumas dessas classes estão diretamente relacionadas ao processo de atribuição automática de componentes (*Component, Container, Description, ValueType, InteractionType, DomainElement* e *Attribute*), enquanto outras estão relacionadas à integração da interface gráfica com o módulo de CDS (*CDSEvent, CDSAction, GUIAction, Event,*

ActionEvent, FocusEvent, KeyEvent e *MouseEvent*). A seguir descreveremos cada uma das classes da ontologia em detalhes:

4.1.1.1 DomainElement

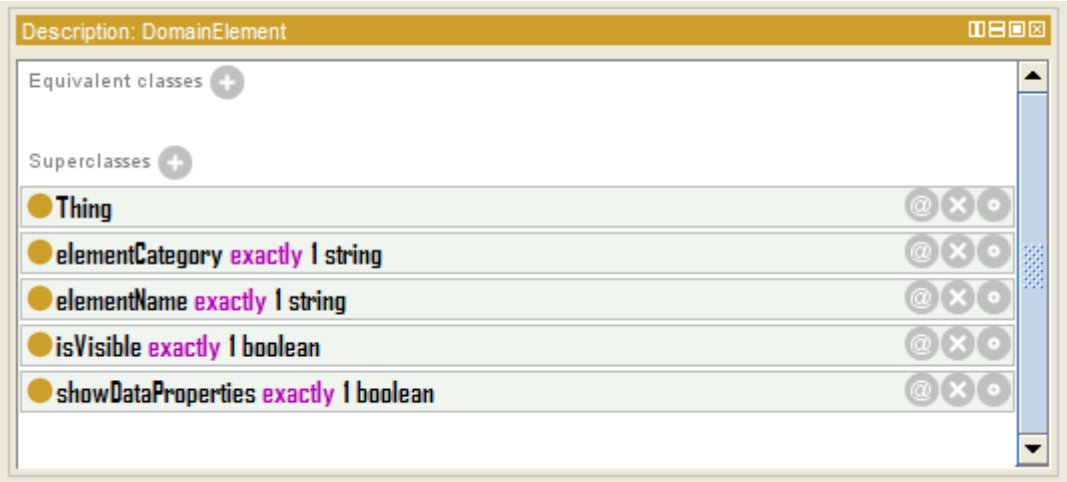

Figura 11: Estrutura da classe *DomainElement*

A classe *DomainElement* representa os tipos de elementos de domínio que podem estar contidos em um documento. Cada indivíduo desta classe na ontologia representa um tipo de elemento. Podem ser, por exemplo, uma seção de um documento (*Section*) ou um arquétipo que especifica um conceito biomédico (*Archetype*), e para cada tipo de elemento existe um ou mais componentes de interface habilitados. Podem ser ainda algum elemento de CDS integrado ao documento, como um *cds_error*, por exemplo.

Os indivíduos da classe *Component* se relacionam aos indivíduos dessa classe para que seja possível filtrar os componentes para determinado tipo de elemento, c0nforme será descrito no tópico **Filtragem pela classe**, na seção 4.1.2. Nessa etapa da atribuição de componentes genéricos, apenas os componentes de interface que estão associados ao elemento de domínio passado como parâmetro são selecionados. Na ontologia, essa relação entre componente de interface e elemento de domínio é dado pela propriedade *represents*. Cada indivíduo da classe *DomainElement* pode estar associado a mais de um componente. A classe *DomainElement* possui três campos:

- **elementName:** Indica o nome do elemento. Este nome deve ser único, pois ele é usado como identificador do elemento pelo algoritmo de atribuição de componentes.

- **elementCategory:** Indica a categoria à qual o elemento pertence. Elementos pertencentes aos conceitos biomédicos, por exemplo, são de natureza distinta dos elementos de CDS. Portanto são de categorias distintas.
- **isVisible:** Indica se o elemento de domínio deve ou não aparecer para o usuário. Esse campo é utilizado geralmente por alguns elementos de domínio relacionados ao CDS, como uma mensagem de erro por exemplo, que não deve ser exibida por default, somente após a captura de determinado evento.
- **showDataProperties:** Os elementos de domínio, assim com os demais indivíduos de qualquer modelo ontológico OWL, possuem campos de dados chamados *data properties*, como descrito na seção 2.3.2. Durante a geração da GUI, alguns desses *data properties* devem ser exibidos para o usuário, enquanto alguns são utilizados apenas pelo sistema para controle. Este campo indica quais elementos devem ter seus *data properties* exibidos para o usuário e quais não devem.

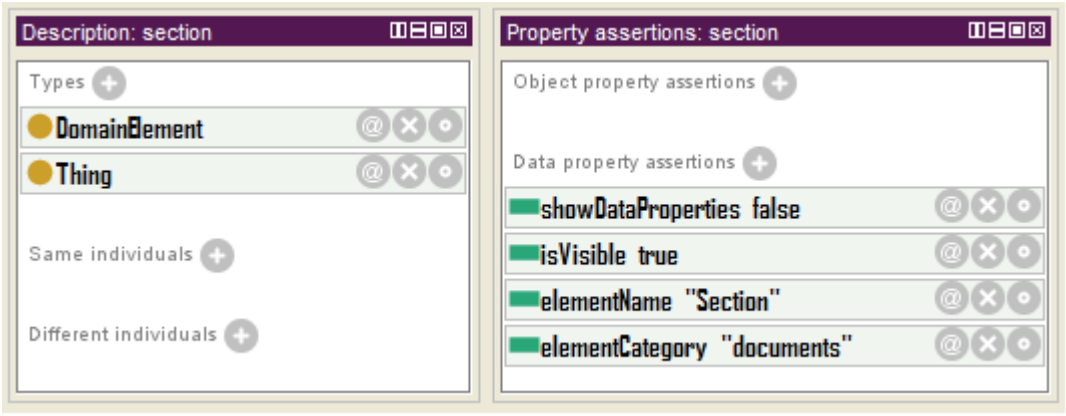

Figura 12: Elemento *section* representado na ontologia de GUI

A **Figura 12** contém um exemplo com a representação do elemento de domínio *section* na ontologia de GUI. A string "*Section*" indica o nome do elemento de domínio que a classe referencia. Durante a execução do algoritmo de atribuição de componentes para um determinado documento, para cada seção do documento encontrada, uma busca na base de componentes é realizada. Os componentes da ontologia serão filtrados de acordo com o tipo de elemento de domínio, e apenas os componentes que tiverem associação com o tipo de elemento especificado são retornados. Neste caso, ao encontrar uma seção em um documento, apenas os componentes relacionados com o *DomainElement section* são retornados como resultado da filtragem.

O valor booleano *false* no campo *showDataProperties* indica que os campos da seção encontrada no documento não devem ser exibidos para o usuário, pois es-

ses campos contém apenas informações de controle interno, como a ordem em que deve ser exibida no documento. A string *"documents"* indica que o elemento *section* é pertencente à categoria de documentos.

4.1.1.2 Description

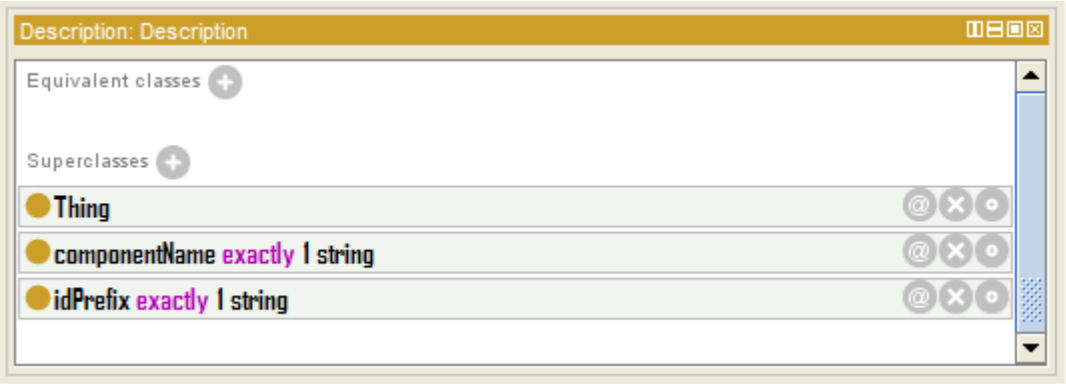

Figura 13: Estrutura da classe *Description*

O objetivo da classe *Description* é encapsular os campos que descrevem um componente genérico univocamente. Os indivíduos dessa classe na ontologia estão associados aos indivíduos da classe *Component*. Cada individuo da classe *Component* está relacionado a apenas um indivíduo da classe *Description*, e vice-versa. O campo *componentName* foi especificado na descrição do componente, representando o identificador do componente na ontologia, porém não identifica univocamente um determinado componente durante a geração do formulário. O campo *idPrefix* é utilizado para este fim. Associando-se esse campo a um número seqüencial atribuído durante a etapa de atribuição de componentes genéricos, podemos identificar univocamente qualquer dos componentes gerados e, a partir do prefixo, podemos descobrir qual o tipo do componente sem ter que realizar uma pesquisa mais aprofundada.

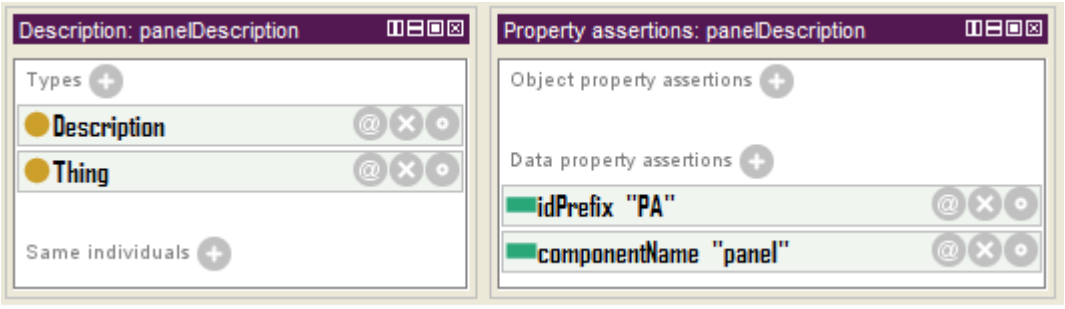

Figura 14: Descrição do componente *panel* na ontologia de GUI

Na **Figura 14**, temos um exemplo com a representação da descrição do componente *panel* na ontologia. A string "*panel*" especificada no campo *componentNa-*

me identifica esse componente na ontologia. Este campo é usado para representar o tipo de componente durante a geração da interface genérica e será usado posteriormente para o mapeamento do componente genérico para um componente específico de alguma tecnologia de interface.

4.1.1.3 Attribute

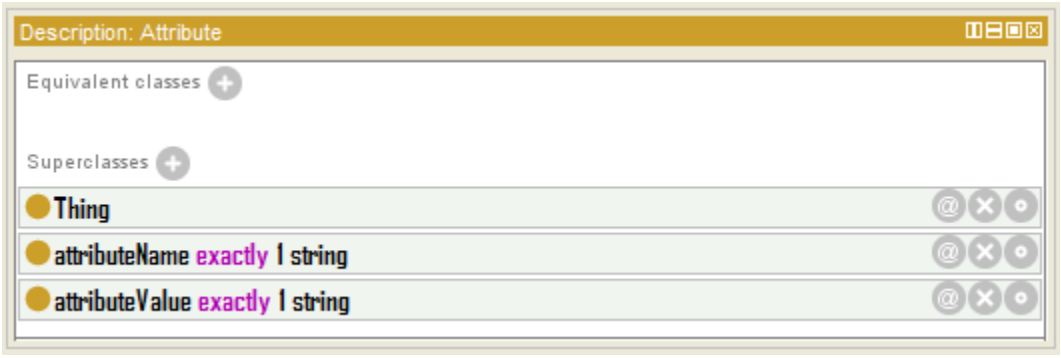

Figura 15: Estrutura da classe Attribute na ontologia de GUI

A classe *Attribute* representa um atributo de um componente de interface. Um atributo é qualquer característica do componente, desde características estéticas como largura e altura; características de posicionamento, como localização do componente na tela; ou até características de comportamento, como a definição de uma máscara para entrada de dados.

Todo atributo é caracterizado por um nome, que o identifica perante os demais atributos, e um valor de determinado tipo de dado. Para maior facilidade de representação na ontologia, os valores de atributos são representados como texto (string), ficando a cargo da aplicação fazer as devidas conversões quando necessárias.

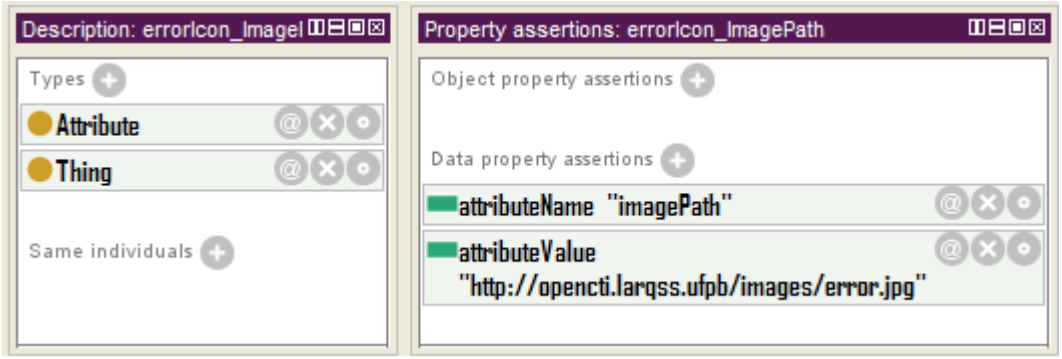

Figura 16: Representação do atributo imagePath na ontologia de GUI

Na **Figura 16**, temos um exemplo com a representação de um atributo do componente *errorIcon* (íconde de erro). O componente *errorIcon* é um indivíduo da

classe *Component*. Como podemos observar, esse componente possui um atributo chamado *imagePath* que indica o caminho da imagem que deve ser exibida no ícone.

4.1.1.4 ValueType

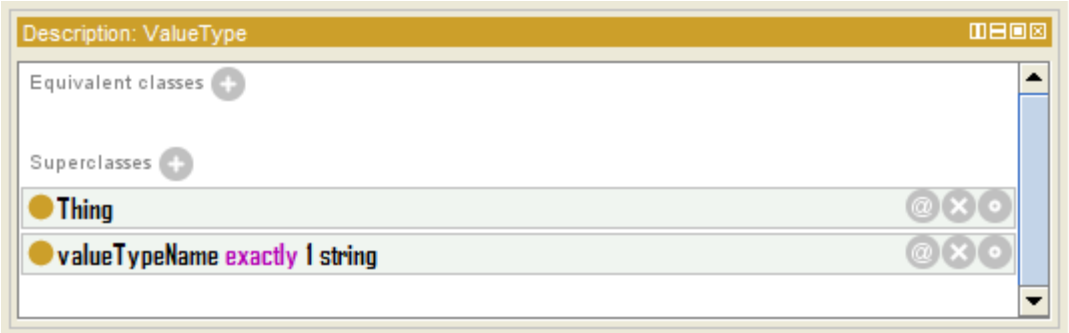

Figura 17: Estrutura da classe *ValueType* na ontologia de GUI

Indivíduos da classe *ValueType* contêm informações sobre quais tipos de valores podem ser representados pelo componente. Representam os tipos de dados, dentre os quais estão inteiro, real, texto, data, etc.

A presença do *ValueType* na definição dos metadados dos componentes é importante para que seja possível realizar uma das etapas da atribuição de componentes genéricos, caracterizada **Filtragem pelo tipo de dado**, descrita na seção 4.1.2. Neste filtro, apenas os componentes que podem representar o tipo de dado passado por parâmetro são selecionados.

Cada indivíduo da classe *ValueType* está relacionado a vários componentes de interface. De forma recíproca, para cada componente podem existir diversos indivíduos dessa classe, indicando que o componente de interface está apto a representar um ou mais tipo de dados.

4.1.1.5 *InteractionType*

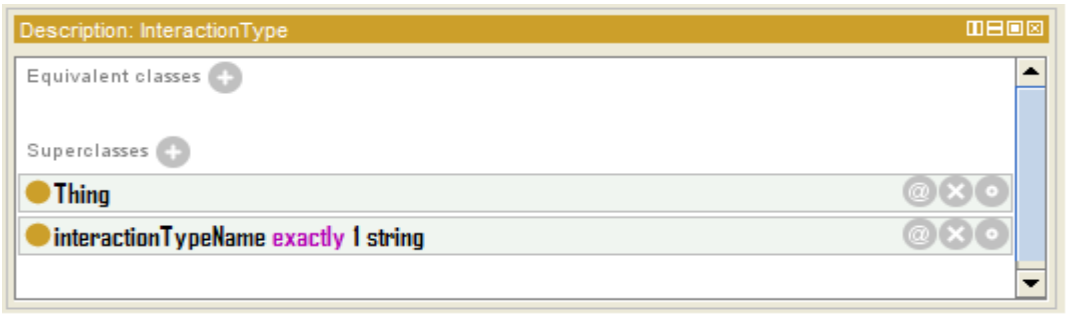

Figura 18: Representação da classe InteractionType na ontologia de GUI

Indivíduos da classe *InteractionType* representam os tipos de interação possíveis, indicando, por exemplo, se o usuário, ao interagir com um componente de interface, deve selecionar uma dentre várias opções fornecidas (tipo *Selection*) ou se deve entrar com valores em texto livre (tipo *Free*).

A presença desse campo na definição dos metadados dos componentes é importante para que seja possível realizar uma das etapas da atribuição de componentes genéricos, caracterizada **Filtragem pelo tipo de interação**, descrita na seção 4.1.2. Neste filtro, apenas os componentes que tiverem o tipo de interação especificado são selecionados.

4.1.1.6 *Component*

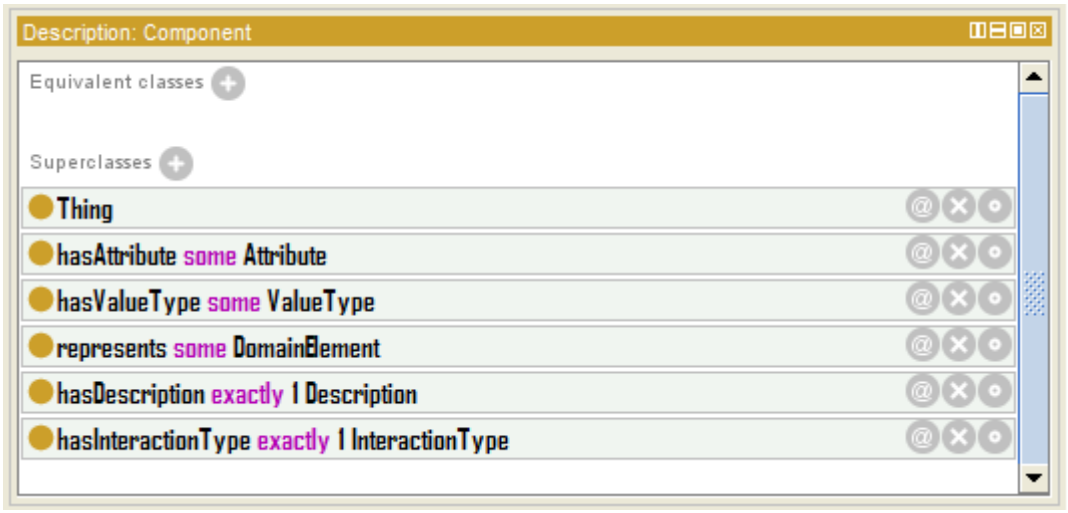

Figura 19: Estrutura da classe *Component*

A classe *Component* é a classe principal da ontologia. Ela representa um componente de interface genérico. Esta classe, assim como as demais da ontologia, herdam da classe *Thing*, que é a superclasse de qualquer ontologia OWL. Indivíduos da classe *Component* representam de forma abstrata um componente de interface genérico,

contendo, através do relacionamento com indivíduos de outras classes da ontologia, informações (metadados) relevantes para a identificação de quais elementos o componente referente pode representar na GUI. Esses metadados são utilizados pelo algoritmo de atribuição de componentes de interface para os elementos de domínio pertencentes a um documento, servindo como parâmetro para que seja realizada uma filtragem dos componentes, de acordo com o propósito de cada elemento.

Cada componente é composto por uma descrição (*Description*) que fornece dados que identificam univocamente cada componente. Os indivíduos da classe *Component* são relacionados a indivíduos da classe *Description* pelo relacionamento *hasDescription*. O relacionamento *represents* indica quais **tipos** de elementos do domínio (*DomainElement*) o componente está habilitado a representar. Um elemento do domínio representa um elemento pertencente à estrutura de um documento, apresentada na seção 3.1.2.1. Dois componentes distintos podem estar relacionados a um mesmo elemento de domínio, e um componente pode estar relacionado com um ou mais elementos de domínio.

A classe *Component* tem ainda um relacionamento com uma classe que faz a representação dos atributos de valor (*ValueType*). Indivíduos dessa classe contêm informações sobre quais tipos de valores podem ser representados pelo componente. Cada indivíduo da classe *Component* pode ter vários relacionamentos com indivíduos da classe *ValueType*. O tipo de interação do componente é indicado pelo relacionamento com a classe *InteractionType*. Nesse caso, apenas um relacionamento pode existir entre as duas classes, uma vez que cada componente possui apenas um tipo de interação.

4.1.1.7 Container

A classe *Container* é uma especialização da classe *Component* para representar componentes que agrupam outros componentes. Até o atual estágio do estudo, não há nenhuma diferença em termos dos campos entre uma classe e outra, porém é necessário que haja essa especificação para que seja possível identificar quais componentes podem conter componentes filhos e quais não podem.

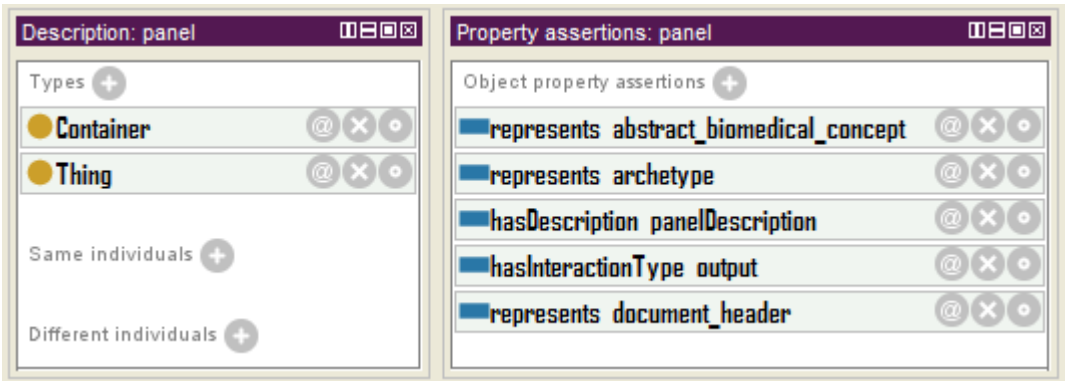

Figura 20: Componente *panel* representado na ontologia de GUI

A **Figura 20** apresenta um exemplo da instanciação de um indivíduo da classe *Container* na ontologia. O indivíduo representado corresponde ao componente *panel*. Esse indivíduo está relacionado a um indivíduo da classe *Description* (indivíduo *panelDescription*). A presença do relacionamento *represents* indica quais tipos de elementos de domínio podem ser representados pelo componente *panel*. Durante a execução do algoritmo de atribuição de componentes, na etapa de **Filtragem pela classe** (seção 4.1.2)**,** sempre que aparecer algum elemento dentre os presentes na estrutura (*abstract_biomedical_concept, archetype ou document_header*), o componente *panel* será atribuído como componente candidato.

4.1.1.8 CDSEvent

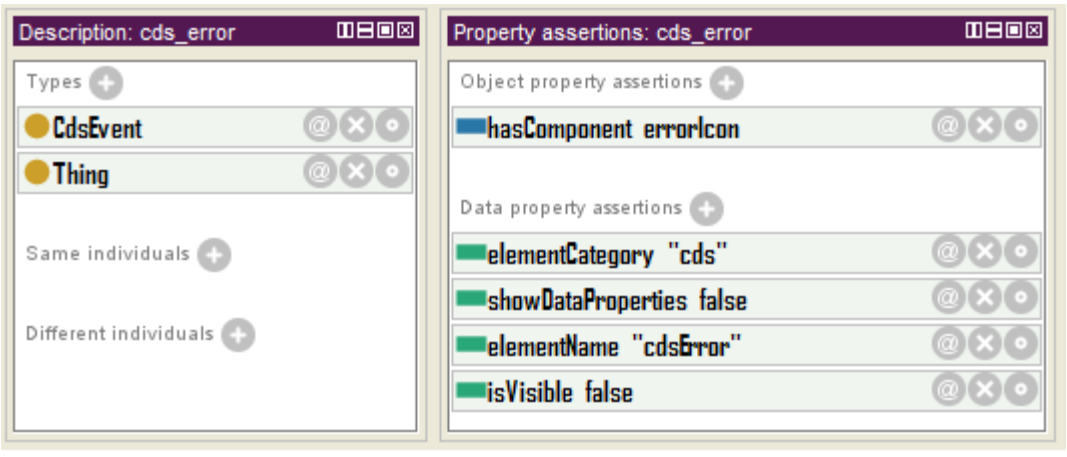

Figura 21: Representação do elemento *cds_error* na ontologia de GUI

A classe *CDSEvent* é uma especialização da classe *DomainElement* para a representação de elementos de domínio relacionados ao CDS. A **Figura 21** contém um exemplo com a modelagem do elemento *cds_error*, que representa um agente de CDS que valida valores de entrada de determinados conceitos biomédicos e exibe um ícone de erro caso esses valores estejam em desconformidade com determinados parâmetros.

Como podemos observar, a única diferença estrutural entre as classes *DomainElement* e *CDSEvent* é que os indivíduos desta podem ter associados componentes de interface, através do relacionamento *hasComponent*. Como veremos na seção 4.3, que descreve a integração da GUI com o módulo de CDS, alguns elementos de domínio associados ao CDS requerem a atribuição de um ou mais componentes de interface para a realização de sua tarefa. Este é o caso do *cds_error*.

Os campos *elementName* e *elementCategoty* indicam o nome e a categoria do elemento. O campo *showDataProperties* igual a *false* indica que nenhum dos *dataProperties* do elemento devem ser exibidos para o usuário. O campo *isVisilbe* igual a false indica que, apesar do *cds_error* requerer a inclusão do componente *errorIcon* no formulário, este componente não deve ser exibido por default. Apenas por meio de um ou mais eventos esse componente deve mudar seu status de visibilidade.

4.1.1.9 CDSAction

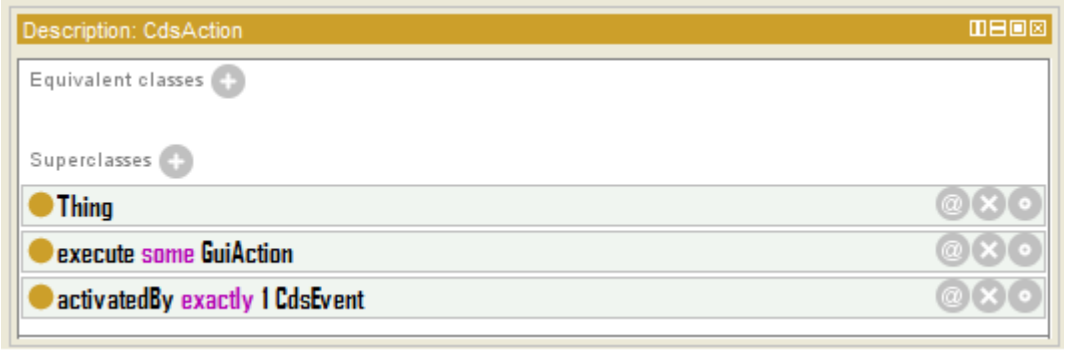

Figura 22: Representação da classe *CDSAction* na ontologia de GUI

A classe *CDSAction* encapsula várias ações que devem ser executadas na GUI de uma só vez, após a ocorrência de um evento de CDS. Como podemos ver na **Figura 22**, os indivíduos desta classe se relacionam com os indivíduos da classe *CDSEvent* por meio do relacionamento *activatedBy*. O relacionamento *execute* associado à classe *GuiAction* representa as diversas ações que devem ser executadas na GUI caso um determinado evento de CDS ocorra. Utilizando o exemplo do *cds_error*, quando um evento desse tipo ocorre, uma instância da classe *CDSAction,* chamada *cds_error_action* é ativada, iniciando a execução de uma instância da classe *GuiAction*, que fará com que o componente *error_icon* (ícone de erro) fique visível para o usuário.

4.1.1.10 GuiAction

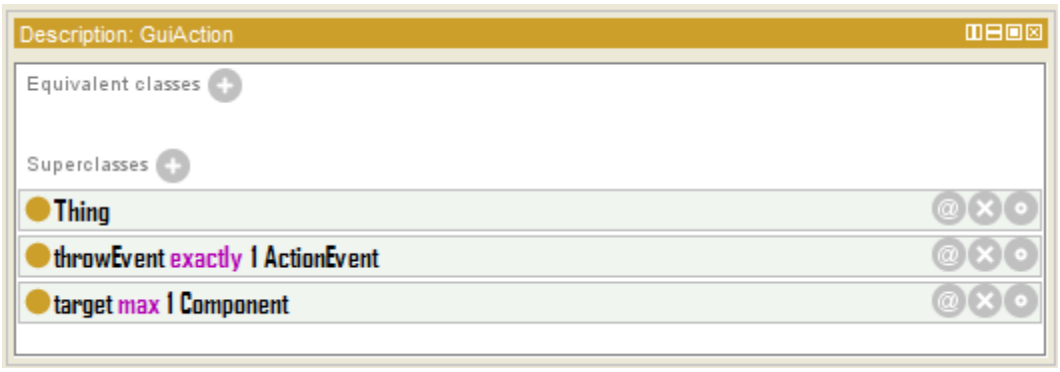

Figura 23: Representação da classe GuiAction na ontologia de GUI

A classe *GuiAction* representa uma ação que deve ser executada na GUI após a ocorrência de um evento de CDS. As ações na GUI são executadas por meio de eventos, no caso, *ActionEvents*. Um evento pode ser a alteração do estado de um componente, a mudança do valor de determinado campo, etc. Geralmente essas ações estão associadas a um determinado componente, porém isso não é obrigatório, como veremos na seção 4.3. Essa associação, quando existir, deve ser feita através do relacionamento *target*, que indica o componente alvo da ação.

4.1.1.11 Event

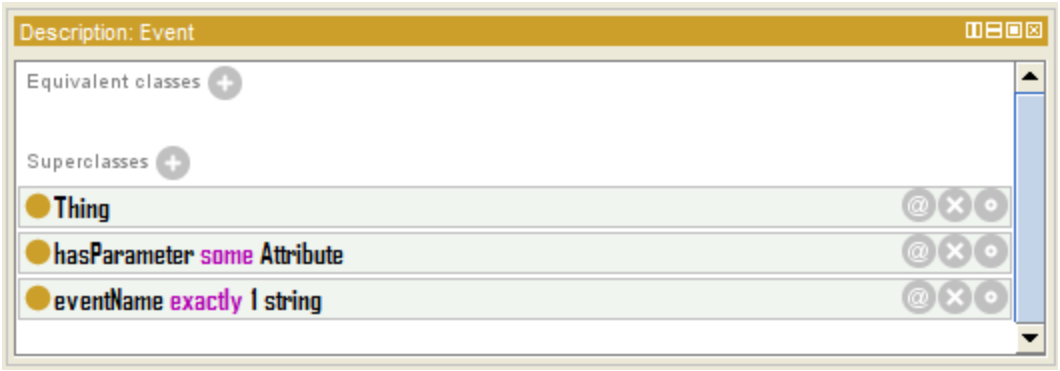

Figura 24: Representação da classe *Event* na ontologia de GUI

A classe *Event* representa eventos da GUI. Alguns desses eventos são lançados pelo usuário durante sua interação com o sistema, outros são executados automaticamente após algum processamento.

Na **Figura 10** podemos observar que a classe *Event* possui várias especializações. São elas: *ActionEvent*, *FocusEvent*, *KeyEvent* e *MouseEvent*. Já vimos na seção 4.1.1.10 um exemplo de uso da classe *ActionEvent*, onde esse tipo de evento é lançado automaticamente pela ferramenta após a captura de determinado evento de CDS. Os demais tipos de eventos são todos lançados pelo usuário.

O campo *eventName* indica o nome do evento. O relacionamento *hasParameter* com a classe *Attribute* indica quais são os parâmetros que devem ser passados ao lançar o evento. Por exemplo, um evento que altera a visibilidade de determinado componente de interface deve receber um parâmetro booleano que indica se o componente deve ser exibido (*true*) ou não (*false*).

4.1.2 Algoritmo de Atribuição de Componentes

Na definição dos conceitos biomédicos contidos em um documento, não estão presentes informações sobre como estes conceitos devem ser exibidos para o usuário. Além disto, elementos de CDS também contidos num documento requerem exibição de informações para o usuário. Deve-se, portanto, definir um algoritmo que, quando executado, irá atribuir, com base nos metadados recuperados da ontologia, qual componente de interface está habilitado para a representação de cada conceito biomédico.

Ao receber uma solicitação de geração de um formulário para um documento, o elemento *GenericGUIBuilder* recebe como parâmetro de entrada um objeto da classe *HealthDocument*, descrita na subseção 3.1.2.2, que representa o documento. Os elementos do documento serão percorridos um a um, e para cada um deles será atribuído um componente de interface. Uma vez atribuído o componente para o elemento, serão percorridos todas as propriedades de dados do elemento. Caso esses dados devam ser exibidos para o usuário, componentes de interface são atribuídos para eles. Cada um dos elementos filhos na hierarquia também será percorrido, e o algoritmo deve ser aplicado recursivamente para estes elementos, atribuindo componentes de interface para eles, obtendo-se ao final uma hierarquia de componentes de interface apta a representar o documento na GUI.

Vamos agora descrever o processo de atribuição para um elemento. Esse elemento terá seus metadados recuperados e, com base nesses metadados, será executada uma seqüência de filtros sobre o conjunto de componentes candidatos, até que se obtenha um ou mais componentes habilitados. Esses filtros irão retornar apenas os componentes que satisfizerem a um conjunto de características específicas, e para determinado elemento de domínio, apenas o componente que satisfizer todos os requisitos será retornado, sendo então atribuído para o elemento de domínio em questão. Apresentaremos a seguir cada um dos filtros:

- **Filtragem pela Classe:** Todo elemento em um documento possui uma classe. Se o elemento for um conceito biomédico, sua classe será *BiomedicalConcept*; se for uma seção de um documento, sua classe será *Section*; e as-

sim por diante. Por sua vez, cada componente de interface contém metadados relativos a quais classes de elementos ele está habilitado a representar (subseção 4.1.1.1). O papel do filtro é verificar quais componentes estão habilitados a representar elementos (*DomainElement*) da classe especificada;

- **Filtragem pelo Número de Filhos:** Existem componentes de interface que servem como "agrupadores" ou recipientes de outros componentes, mantendo uma organização interna para eles. Esses componentes são chamados *Containers*. Quando um elemento de domínio possui outros sub-elementos na hierarquia, é necessário que esses sub-elementos sejam representados na GUI de forma relacionada ao elemento pai, como elementos internos. Uma forma de fazer isso é utilizando *containers* para a representação dos conceitos pai. O papel desse filtro é verificar o número de elementos filhos de um elemento e, caso esse valor seja maior que zero, apenas componentes ditos *Containers* (ver subseção 4.1.1.7) devem ser mantidos entre os componentes candidatos;

- **Filtragem pelo Tipo de Dado:** Cada componente de interface é capaz de representar alguns tipos de dados e outros não. Por exemplo, o campo de entrada de texto pode representar, além de texto em si, valores numéricos reais e de ponto flutuante. Um tipo de dado mais estruturado como o tipo *Date*, que é dividido em 3 campos de dados, não é suportado por esse componente. Dessa forma, o papel desse filtro é, para cada propriedade de dados de um elemento que requer representação na GUI, habilitar apenas os componentes capazes de representar o tipo de dado da propriedade. Essa filtragem é feita de acordo com os elementos da classe *ValueFeatures* (ver subseção 4.1.1.4) associados ao componente de interface;

- **Filtragem pelo Tipo de Interação:** Alguns componentes são usados como forma de entrada de texto livre, outros proporcionam a seleção de um objeto em um conjunto pré-estabelecido de opções, outros servem apenas como saída de dados, dentre outras formas de interação com o usuário. Esse filtro deve, de acordo com características do elemento, habilitar apenas componentes que apresentam o tipo de interação solicitado. Essa filtragem é feita de acordo com os elementos da classe *InteractionType* (ver subseção 4.1.1.5) associados ao componente de interface.

Conforme apresentado, cada um dos filtros recebe como entrada uma árvore de componentes genéricos e retorna essa árvore modificada de acordo com o propósito do filtro, possivelmente com menos componentes que a árvore de entrada. A definição dos filtros, porém, não é suficiente para o entendimento do processo de atribuição. A ordem com que eles são aplicados e a identificação das entradas e saídas de cada filtro são essenciais para o entendimento do processo. A seguir, apresentamos as etapas do algoritmo utilizado na atribuição de componentes genéricos. O algoritmo é relativamente curto, devido a sua natureza recursiva. Para auxiliar na explicação do fluxo do algoritmo, definimos algumas variáveis, como segue:

- **elementosDeDomínio**: Representa uma lista de elementos de domínio. Pode representar os elementos diretamente associados ao documento ou elementos filhos destes na hierarquia;
- **elementoCorrente**: Representa o elemento de domínio selecionado no momento;
- **componenteCorrente**: Representa o componente de interface genérico selecionado no momento;
- **componentesCandidatos**: Representa uma lista de componentes candidatos a se tornarem componentes atribuídos;
- **componentesAtribuidos**: Representa uma lista de componentes já atribuídos a determinados elementos de domínio;
- **dataPropertySelecionado:** Representa uma propriedade de dados selecionada no momento.

Para que o algoritmo seja executado, é necessário definir o estado inicial de tais variáveis. As variáveis *elementoCorrente* e *componenteSelecionado* são inicialmente nulas. As listas *componentesCandidatos* e *componentesAtribuidos* são inicialmente vazias. A lista *elementosDeDomínio* inicialmente contém os elementos de domínio diretamente associados ao documento, ou seja, os elementos de nível mais alto na hierarquia, pertencentes à raiz do documento. O pseudo-código do algoritmo é apresentado na **Figura 25**.

```
1  method atribuirComponentes (elementosDeDomínio) {
2      componenteCorrente = null;
3      componentesAtribuidos = new List();
4      contador = 0;
5      while (contador < elementosDeDomínio.size()) {
6         elementoCorrente = elementosDeDomínio.nextElement();
7         componentesCandidatos = getAllGenericComponents();
```

```
componentesCandidados = filtragemPelaClasse (componentesCandidatos, elementoCorrente.getClasse());
componentesCandidados = filtragemPeloNumeroDeFilhos (componentesCandidatos, elementoCorrente.getNumFilhos());
componentesCandidados = filtragemPeloTipoDeInteração (componentesCandidatos, elementoCorrente.getInteração());
if (componentesCandidatos.size() == 1) {
    componenteCorrente = componentesCandidados[0];
    componentesAtribuidos.add(componenteCorrente);
} else {
    return ERRO;
}
if (elementoCorrente.showDataProperties() == true) {
    contador_2 = 0;
    dataProperties = elementoCorrente.getDataProperties();
    while (contador_2 < dataProperties.size()) {
        dataPropertySelecionado = dataProperties.next();
        componentesCandidatos = getAllGenericComponents();
        componentesCandidados = filtragemPelaClasse (componentesCandidatos, "dataProperty");
        componentesCandidados = filtragemPeloTIpoDeValor (componentesCandidatos, dataPropertySelecionado.getValueType());
        if (componentesCandidatos.size() == 1) {
            componenteCorrente.addFilho( componentesCandidados[0]);
        } else {
            return ERRO;
        }
        contador_2 = contador_2 + 1;
    }
}
if (elementoCorrente.getNumFilhos() > 0) {
    componenteCorrente.addFilhos(atribuirComponentes (elementoCorrente.getFilhos())
}
contador = contador + 1;
}
return componentesAtribuidos;
```

Figura 25: Pseudo-código do algoritmo de atribuição de componentes genéricos

O algoritmo contém uma iteração que percorre os elementos diretamente relacionados ao documento. De acordo com a atual definição de documento do O-

penCTI, apresentada na seção 3.1.2, esses elementos são o cabeçalho (*Header*) e o corpo (*Body*) do documento. Para cada um desses elementos é executada uma **Filtragem pela classe** (linha 8). Apenas componentes habilitados a representar elementos da classe do elemento corrente são mantidos como *componentes candidatos*. Na linha 10, caso o elemento corrente possua elementos filhos na hierarquia, o componente atribuído a este elemento necessariamente deve ser um container, e por este motivo é aplicada uma **Filtragem pelo número de filhos** sobre os componentes candidatos resultantes da etapa anterior. Na linha 12 é executada uma **Filtragem pelo tipo de interação**.

A linha 14 verifica se apenas um componente restou como resultado dos filtros e, em caso negativo, o algoritmo encerra e um erro é lançado. Caso nenhum componente seja atribuído, existe de fato uma inconsistência na definição dos componentes, uma vez que não comporta determinados tipos de elementos de domínio. Para o caso da existência de mais de um componente candidato, o erro é lançado devido à não existência de um tratamento dessa divergência na escolha de componentes no estágio atual do trabalho.

As linhas 23 a 38 representam um laço iterativo que percorre as propriedades de dados (*dataProperty*) do elemento corrente. A execução desse laço só acontecerá caso as propriedades de dados do elemento corrente devam ser representadas na GUI (linha 20). Essa condição é necessária porque alguns elementos de domínio, a exemplo do cabeçalho do documento, contêm propriedades de dados que têm a função apenas de controle interno da aplicação, não correspondendo a dados que devem ser preenchidos pelo usuário. Caso o valor da *flag* seja positivo, para cada uma das propriedades de dados deve ser atribuído um componente de interface. Os componentes então sofrem os processos de **Filtragem pela classe** (linha 26) e **Filtragem pelo tipo de dado** (linha 28), onde como resultado deve haver apenas 1 componente candidato a ser construído. Durante a construção, é criada uma instância do componente genérico onde seus atributos e eventos (descritos na seção 4.1.1) são definidos. Em seguida, o componente é adicionado à lista de componentes filhos do componente corrente. Caso a lista de componentes candidatos não possua nenhum ou possua mais de um componente, o algoritmo encerra a execução e uma mensagem de erro é lançada, assim como ocorre na linha 18.

Na linha 41, os elementos filhos do elemento corrente são passados como parâmetro para a execução recursiva do algoritmo, que deve atribuir componentes de interface para cada um deles antes de passar para o próximo elemento. O algoritmo encerra quando não houver mais nenhum elemento de domínio a ser percorrido no

documento. Como resultado, é obtida uma estrutura genérica da GUI, formada por componentes de interface em diversos níveis de hierarquia, representando elementos e propriedades de dados. Essa estrutura genérica ainda não pode ser instanciada para a criação da GUI final, uma vez que é composta de componentes genéricos que não possuem implementação direta correspondente.

Como pudemos observar, o processo de atribuição automática de componentes é realizado com base em metadados extraídos dos elementos de domínio e em metadados inerentes aos componentes de interface, representados na ontologia de componentes genéricos de interface. O cruzamento dessas informações nos dá a possibilidade de selecionar os componentes mais apropriados para cada tipo de elemento.

4.2 Mapeamento para tecnologia de GUI

Para cada tecnologia de interface suportada pelo EHR, deve existir um meio no qual estarão definidos os mapeamentos de cada conceito abstrato para o componente correspondente na tecnologia, incluindo também os mapeamentos dos atributos e eventos dos componentes. A árvore de componentes genéricos passa por um *parsing*, que determinará quais componentes serão utilizados para a construção da GUI, com base nas informações recuperadas da ontologia de mapeamento. Esse processo de mapeamento é realizado pelo componente *GUIConverter* (ver **Figura 9**) da arquitetura do MedViewGen, logo após a atribuição de componentes genéricos. É importante frisar que este mapeamento é distinto do mapeamento entre conceitos biomédicos (e documentos onde são referenciados) e componentes de interface, de natureza mais complexa, mencionando na seção 4.1. O mapeamento tecnológico realiza apenas a tradução dos componentes abstratos em componentes concretos que serão utilizados, posteriormente, na construção da interface final para o usuário.

Para a realização do mapeamento, existe a necessidade de saber, a partir do tipo de visualizador e/ou plataforma que o usuário está acessando, qual tecnologia de componentes de interface será utilizada. Para isso, logo após a requisição de acesso do usuário ao sistema, é realizada uma análise do seu contexto e o armazenamento dessas informações. Em futuras interações do usuário com o sistema, recuperam-se as informações do contexto do usuário e, a partir desses dados, determina-se a tecnologia de GUI utilizada.

Como dito na seção 3.2.4, assim como a descrição dos componentes genéricos, os mapeamentos para componentes concretos também são descritos por meio

de ontologia. Nas próximas seções descreveremos a estrutura da ontologia de mapeamento e o algoritmo utilizado no processo de tradução dos componentes.

4.2.1 Modelo Ontológico de Mapeamento

Apesar do processo de tradução ser mais simples que o processo de atribuição de componentes genéricos, cada um dos atributos e eventos associados aos componentes genéricos devem ser traduzidos também, criando um modelo de informação um pouco mais complexo que um simples mapeamento um para um. Assim como nas demais ontologias apresentadas, utilizamos a notação UML para a representação da ontologia de mapeamento (**Figura 26**).

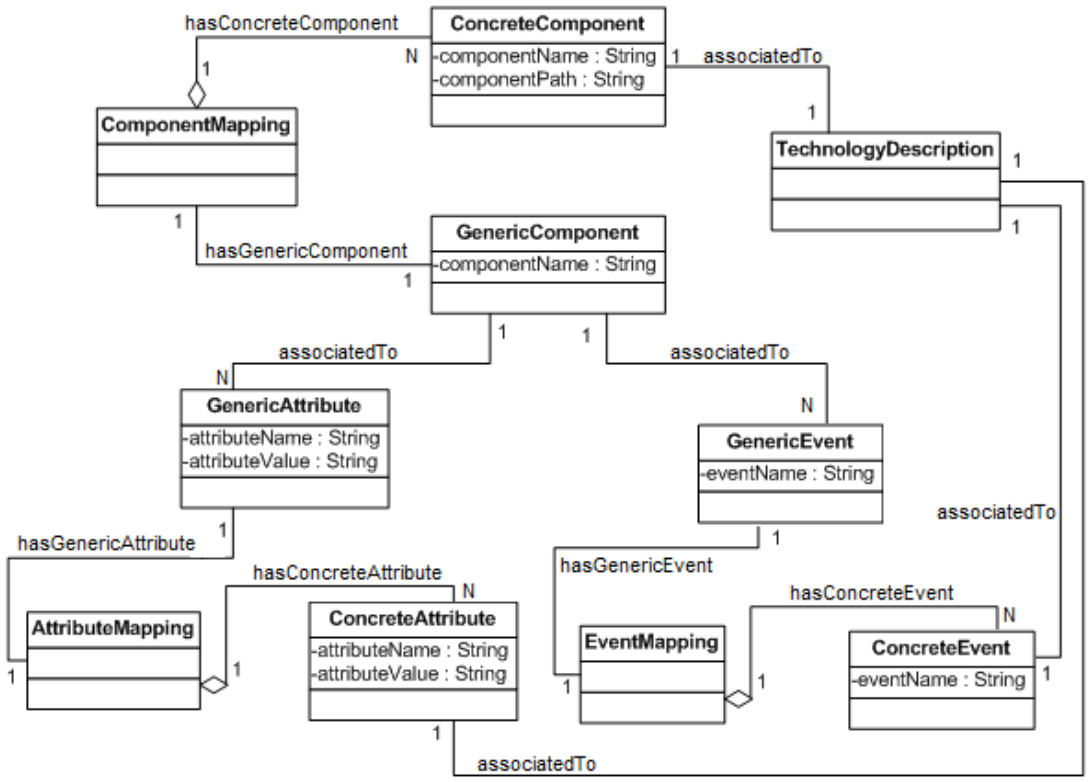

Figura 26: Estrutura da ontologia de mapeamento para componentes concretos

A estrutura da **Figura 26** representa as classes da ontologia e os relacionamentos que elas têm umas com as outras. As classes *ComponentMapping*, *AttributeMapping* e *EventMapping* representam os mapeamentos de componentes, atributos e eventos, respectivamente. Os indivíduos dessas classes contêm relacionamentos com classes que representam itens genéricos e concretos, representados pelas classes *GenericComponent, ConcreteComponent, GenericAttribute, ConcreteAttribute, GenericEvent* e *ConcreteEvent*. A classe *TechnologyDescription* encapsula os

campos que identificam as possíveis tecnologias de interface suportadas pelo sistema. A seguir descreveremos cada uma das classes da ontologia em detalhes.

4.2.1.1 TechnologyDescription

A **Figura 27** representa a estrutura da classe *TechnologyDescription*. Essa classe contém informações sobre as tecnologias de interface suportadas pelo sistema. É importante frisar que tecnologia de interface engloba dispositivo de visualização (PC, palmtop, i-phone, etc) e plataforma de software (jsf, Java-swing, etc). Para cada combinação suportada deve haver uma instância dessa classe na ontologia de mapeamento.

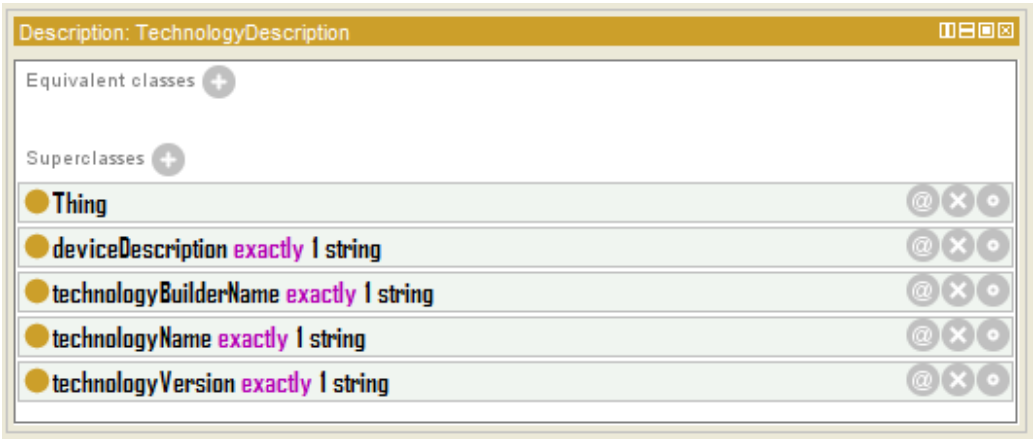

Figura 27: Estrutura da classe *TechnologyDescription* na ontologia de mapeamento

O campo *deviceDescription* identifica qual o tipo de dispositivo utilizado. O campo *technologyName* identifica determinada tecnologia, sem levar em consideração a versão, que é representada no campo *technologyVersion*.

De acordo com a arquitetura do MedViewGen, podem existir simultaneamente várias implementações da interface *UIBuilder*, sendo uma para cada tecnologia. No momento da geração automática da interface, é necessário saber qual a instância correta a ser utilizada. Para esta finalidade, foi inserido o campo *technologyBuilderName*, que permite identificar qual o componente de software correto para determinada tecnologia.

4.2.1.2 *ComponentMapping*

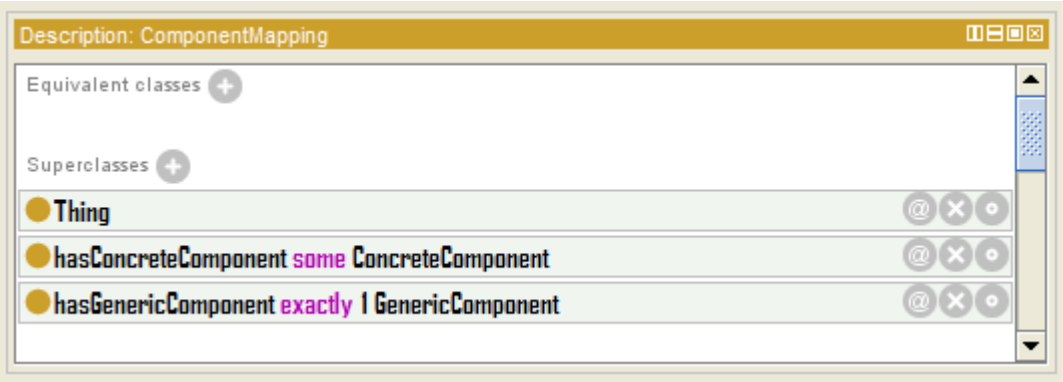

Figura 28: Estrutura da classe *ComponentMapping* na ontologia de mapeamento

Cada instância da classe *ComponentMapping* representa um mapeamento entre determinado componente genérico e uma série de componentes concretos. O relacionamento *hasGenericComponent* liga a instância dessa classe a uma instância da classe *GenericComponent*. O relacionamento *hasConcreteComponent* liga a instância dessa classe a uma ou mais instâncias da classe *ConcreteComponent*. É importante observar que a classe não mapeia um componente genérico para um único componente concreto. Para cada uma das tecnologias de interface, existe um componente concreto correspondente ao componente genérico, e por isso o relacionamento é de um para muitos.

4.2.1.3 *GenericComponent*

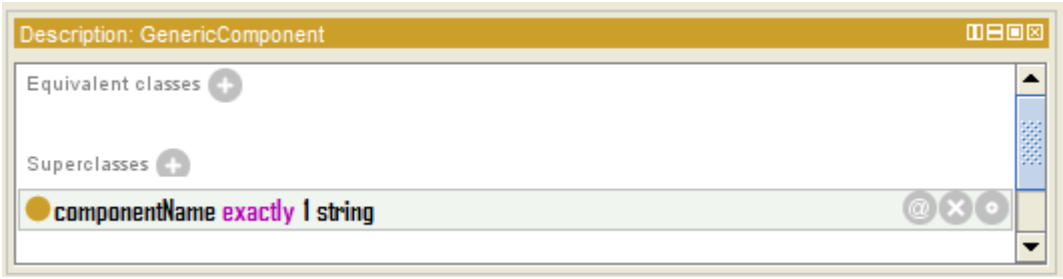

Figura 29: Estrutura da classe *GenericComponent* na ontologia de mapeamento

A classe *GenericComponent* representa um componente genérico. Seu único campo é o identificados *componentName*. O valor desse campo deve casar com o valor proveniente da classe *Description* da ontologia de interface descrita na seção 4.1.1.2. É através desse cruzamento de dados que possibilitamos a tradução do componente genérico em um componente concreto.

4.2.1.4 *ConcreteComponent*

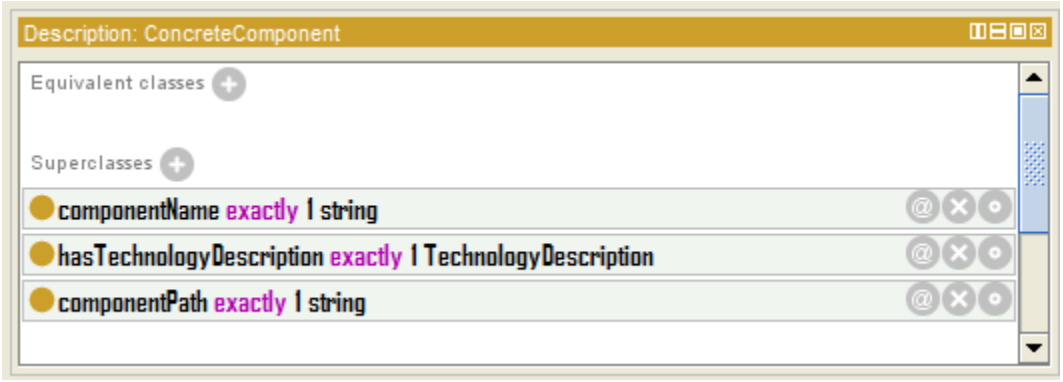

Figura 30: Estrutura da classe *ConcreteComponent* na ontologia de mapeamento

A classe *ConcreteComponent* representa um componente concreto. Os indivíduos dessa classe possuem um identificador representado pelo campo *componentName*.

Como trabalhamos com o conceito de componentes de interface e utilizamos linguagens orientadas a objetos na construção do sistema, cada componente de interface é, na verdade, um objeto instanciado em determinada linguagem. Para que seja possível instanciar o componente, é necessário especificar onde a classe do objeto está localizada. O campo *componentPath* indica o caminho dessa classe. É importante frisar que o termo classe nesse contexto não é o mesmo que vínhamos utilizando anteriormente quando descrevíamos a ontologia, e sim no contexto de linguagem de programação orientada a objetos.

Por fim, cada componente concreto deve estar associado a uma tecnologia de interface. Essa associação é feita pelo relacionamento *hasTechnologyDescription*.

4.2.1.5 *AttributeMapping*

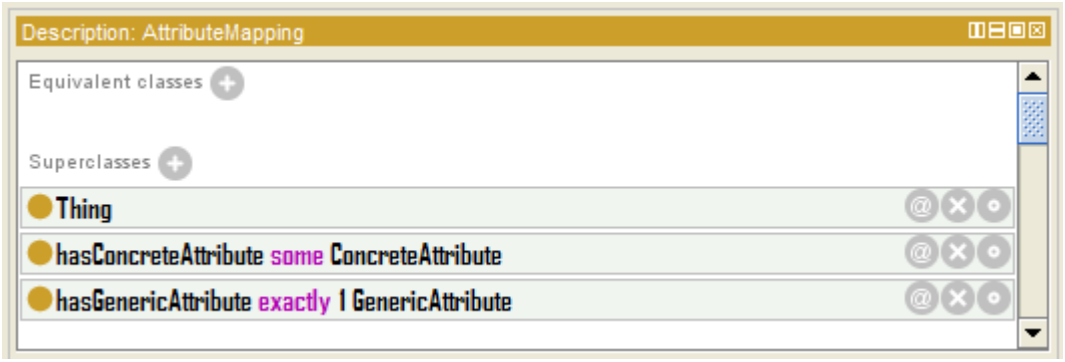

Figura 31: Estrutura da classe *AttributeMapping* na ontologia de mapeamento

Cada instância da classe *AttributeMapping* representa um mapeamento entre determinado atributo genérico e uma série de atributos concretos. O relacionamento *hasGenericAttribute* liga a instância dessa classe a uma instância da classe *Generi-*

cAttribute. O relacionamento *hasConcreteAttribute* liga a instância dessa classe a uma ou mais instâncias da classe *ConcreteAttribute*. É importante observar que a classe não mapeia um atributo genérico para um único atributo concreto. Para cada uma das tecnologias de interface, existe um atributo concreto correspondente ao atributo genérico, e por isso o relacionamento é de um para muitos.

4.2.1.6 GenericAttribute

A classe *GenericAttribute* representa um atributo genérico. Os indivíduos dessa classe são identificados pelo campo *attibuteName*. O valor desse campo deve casar com o valor proveniente da classe *Attribute* da ontologia de interface descrita na seção 4.1.1.3. É através desse cruzamento de dados que possibilitamos a tradução do atributo genérico em um atributo concreto.

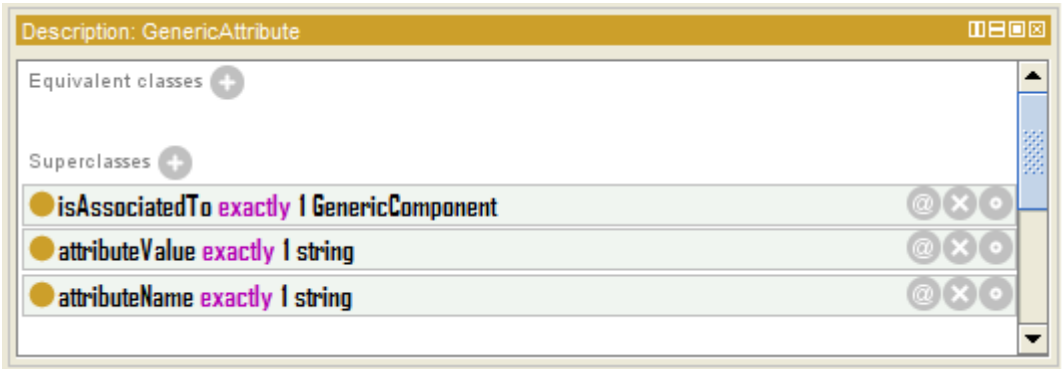

Figura 32: Estrutura da classe *GenericAttribute* na ontologia de mapeamento

A classe ainda possui o campo *attributeValue*, que representa o valor do atributo.

4.2.1.7 *ConcreteAttribute*

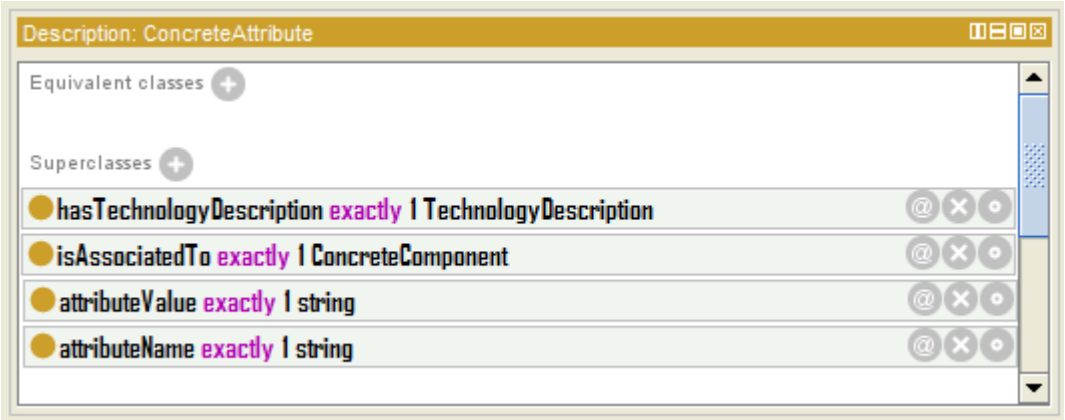

Figura 33: Estrutura da classe *ConcreteAttribute* na ontologia de mapeamento

A classe *ConcreteAttribute* representa um atributo de um componente concreto. Os indivíduos da classe são identificados pelo campo *attributeName*. O campo *attributeValue* representa o valor do atributo.

Cada um dos indivíduos dessa classe deve estar associado a um indivíduo da classe *TechnologyDescription*. Essa associação é feita através do relacionamento *hasTechonologyDescription*.

4.2.1.8 *EventMapping*

Cada instância da classe *EventMapping* representa um mapeamento entre determinado evento genérico e uma série de eventos concretos. O relacionamento *hasGenericEvent* liga a instância dessa classe a uma instância da classe *GenericEvent*. O relacionamento *hasConcreteEvent* liga a instância dessa classe a uma ou mais instâncias da classe *ConcreteEvent*.

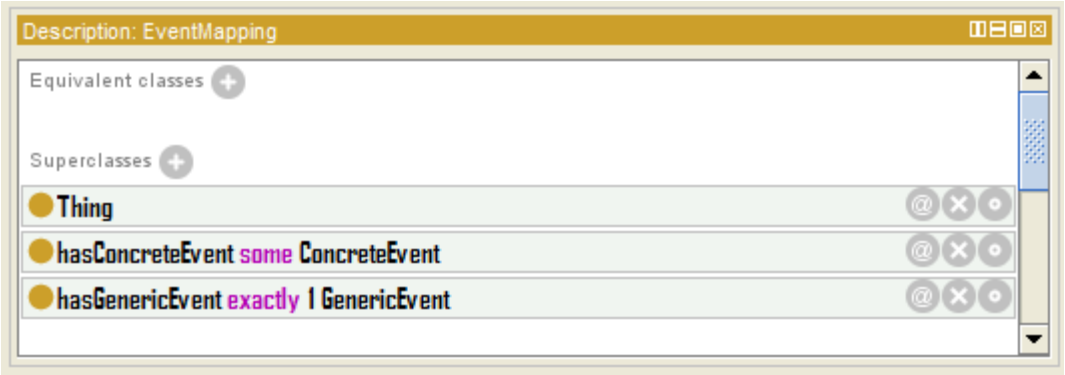

Figura 34: Estrutura da classe *EventMapping* na ontologia de mapeamento

É importante observar que a classe não mapeia um evento genérico para um único evento concreto. Para cada uma das tecnologias de interface, existe um evento

concreto correspondente ao evento genérico, e por isso o relacionamento é de um para muitos.

4.2.1.9 GenericEvent

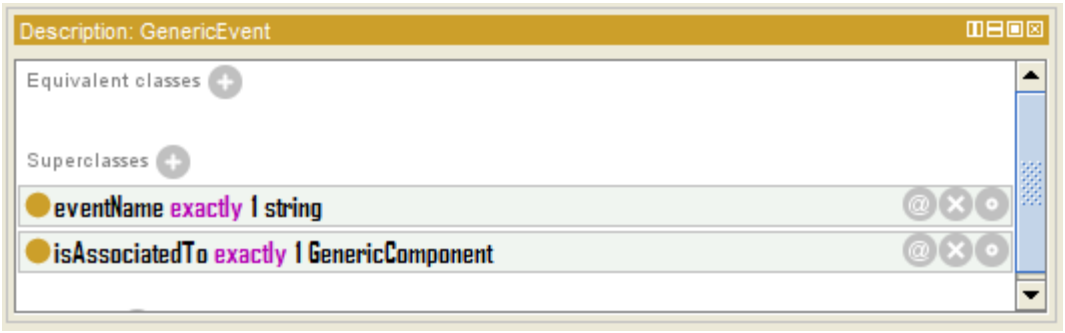

Figura 35: Estrutura da classe *GenericEvent* na ontologia de mapeamento

A classe *GenericEvent* representa um evento genérico. Os indivíduos dessa classe são identificados pelo campo *eventName*. O valor desse campo deve casar com o valor proveniente da classe *Event* da ontologia de interface descrita na seção 4.1.1.3. É através desse cruzamento de dados que possibilitamos a tradução do evento genérico em um evento concreto.

4.2.1.10 ConcreteEvent

A classe *ConcreteEvent* representa um evento de um componente concreto. Os indivíduos da classe são identificados pelo campo *eventName*. Cada um dos indivíduos dessa classe deve estar associado a um indivíduo da classe *TechnologyDescription*. Essa associação é feita através do relacionamento *hasTechonologyDescription*.

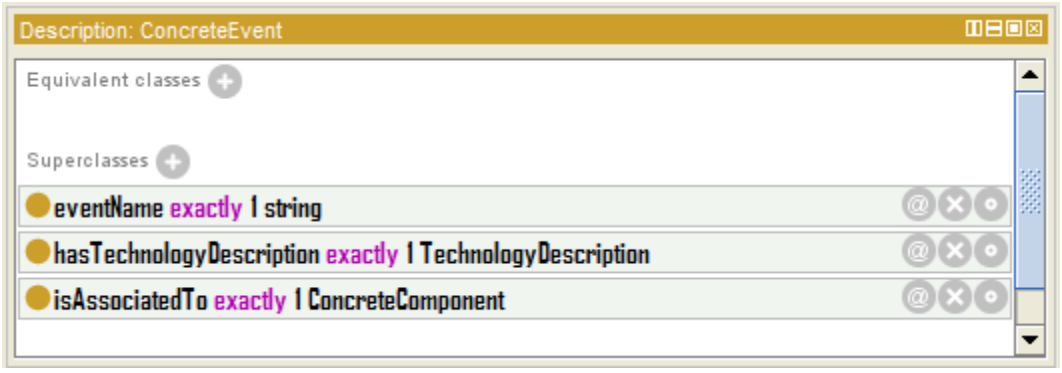

Figura 36: Estrutura da classe *ConcreteEvent* na ontologia de mapeamento

4.2.2 Algoritmo de Mapeamento

Cada componente de interface possui uma lista de atributos possíveis, relativos às mais diversas características do componente como posicionamento na tela, por exemplo; e uma lista de eventos que podem ser lançados e/ou capturados por ele. Es-

sas informações estão representadas de forma genérica na ontologia de interface descrita na subseção 4.1.1 e de forma específica na ontologia de mapeamento descrita na subseção 4.2.1. Assim como os componentes, os seus atributos e eventos devem ser mapeados, criando um processo um pouco mais complexo que necessita a definição de um algoritmo.

Ao receber uma solicitação para a tradução dos componentes genéricos, o elemento *GUIConverter* recebe como parâmetros de entrada uma árvore de componentes genéricos e os dados da tecnologia de interface. Os componentes serão percorridos um a um, e para cada um deles será atribuído um componente concreto, na tecnologia especificada. Uma vez atribuído o componente concreto, serão percorridos os atributos e eventos. Para cada atributo/evento genérico deve ser atribuído um atributo/evento concreto, na tecnologia especificada. Cada um dos componentes filhos na hierarquia também será percorrido, e o algoritmo deve ser aplicado recursivamente para estes componentes, atribuindo componentes concretos para eles, obtendo-se ao final uma árvore de componentes de interface apta a representar o documento na GUI. A seguir temos a descrição do algoritmo de tradução passo-a-passo. Para auxiliar na explicação do fluxo do algoritmo, definimos algumas variáveis, como segue:

- **componentesGenericos**: Representa uma lista de componentes genéricos;
- **componenteCorrente**: Representa o componente genérico selecionado no momento;
- **componenteConcreto**: Representa um componente traduzido pra a uma tecnologia específica;
- **atributosGenericos**: Representa uma lista com os atributos de um componente genérico;
- **atributoCorrente:** Representa o atributo genérico selecionado no momento;
- **eventosGenericos:** Representa uma lista com os eventos de um componente genérico;
- **eventoCorrente:** Representa um evento geérico selecionado no momento;
- **componentesTraduzidos**: Representa uma lista de componentes já traduzidos;
- **tecnologiaInterface:** Representa os dados da tecnologia de interface do usuário corrente.

Para que o algoritmo seja executado, é necessário definir o estado inicial das variáveis. As variáveis *componenteCorrente, atributoCorrente* e *eventoCorrente* são inicialmente nulas. As lista *componentesTraduzidos* é inicialmente vazia. A lista *componentesGenericos* inicialmente contém os componentes genéricos de nível mais alto na hierarquia, pertencentes à raiz do documento. O pseudo-código do algoritmo é apresentado na **Figura 37**.

```
1  method converteComponentes (componentesGenericos) {
2      componenteCorrente = null;
3      componentesTraduzidos = new List();
4      contador = 0;
5      while (contador < componentesGenericos.size()) {
6        componenteCorrente = componentesGenericos.nextElement();
7        componentMapping = Ontology.getMapping(componenteCorrente);
8        componenteConcreto = componentMapping.getComponent (tecnologiaIn-
9            terface);
10       componentesTraduzidos.add(componenteConcreto);
11       atributosGenericos = componenteCorrente.getAtributosGenericos;
12       for (atributosGenericos) {
13           attributeMapping = Ontology.getAttributeMapping (atributoGe-
14               nerico);
15           atributoConcreto = attributeMapping.getAtributoConcreto (tec-
16               nologiaInterface);
17           componenteConcreto.add(atributoConcreto);
18       }
19       eventosGenericos = componenteCorrente.getEventosGenericos;
20       for (eventosGenericos) {
21           eventMapping = Ontology.getEventMapping (eventoGenerico);
22           eventoConcreto = eventMapping.getEventoConcreto (tecnologia-
23               Interface);
24           componenteConcreto.add(eventoConcreto);
25       }
26       if (componenteCorrente.getNumFilhos() > 0) {
27           componenteCorrente.addFilhos(atribuirComponentes (elemento-
28               Corrente.getFilhos())
29       }
30       contador = contador + 1;
31     }
32     return componentesTraduzidos;
33   }
```

Figura 37: Pseudo-código do algoritmo de mapeamento

O algoritmo representa um laço iterativo que percorre os componentes genéricos presentes na hierarquia. Para cada componente genérico existe na ontologia de

mapeamento um elemento da classe *ComponentMapping*, que é recuperado na linha 7. A partir desse elemento, o componente concreto pode ser mapeado utilizando as informações da tecnologia da interface so usuário armazenadas na variável *tecnologiaInterface* (linha 8). Nas linhas 13 a 25 temos dois laços que irão fazer um processo de tradução similar ao feito para os componentes, mas dessa vez para traduzir os atributos e os eventos desses componentes. Como resultado final da execução do algoritmo, temos uma estrutura de árvore de componentes concretos, pronta para ser instanciada em determinada tecnologia de interface.

4.3 Integração com o CDS

Como vimos na seção 3.2.1.4, os elementos de CDS inseridos nos documentos requerem exibição de informações na GUI, além da captura e processamento de eventos da GUI para fins diversos. Para o provimento de tais funcionalidades, deve haver um meio de integração entre as o módulo de CDS e a GUI. Os eventos da GUI devem ser interceptados e direcionados aos agentes de CDS adequados e, após o processamento das informações, a resposta deve ser devolvida à GUI.

A interação entre a camada de apresentação e o módulo de CDS acontece em dois momentos. O primeiro é durante a geração de um formulário para um documento (etapa **5** da **Figura 9**), quando acontece a atribuição de componentes de interface para determinados elementos de CDS (subseção 4.3.1) e a injeção de eventos de gatilho nos componentes com a finalidade de acionar os agentes de CDS (subseção 4.3.2). O segundo momento (etapa **7** da **Figura 9**) acontece após a etapa de geração da interface, quando o usuário interage com o sistema ao preencher um formulário e o auxílio à decisão clínica funciona de fato (seção 4.3.3).

Para facilitar o entendimento, utilizaremos para as próximas subseções um cenário simples com um validador de campo para ilustrar o processo. O cenário é o seguinte: O usuário entra com determinado valor em uma caixa de entrada de texto (*inputText*), correspondente ao conceito biomédico “freqüência cardíaca”, de determinado documento, e passa para o próximo campo. Nesse momento, o sistema processa os dados de entrada e, caso o valor esteja em desconformidade com os parâmetros, um ícone de erro deve ser exibido. Caso o valor esteja correto, nada acontece e o usuário prossegue com sua atividade.

Nas seções subseqüentes descrevemos em detalhe cada uma das etapas necessárias para a integração da GUI com o módulo de CDS.

4.3.1 Atribuição de Componentes

Alguns agentes de auxílio à decisão clínica (não todos) necessitam exibir informações para o usuário, logo necessitam de componentes de interface para isso. Observando o cenário descrito na seção anterior, podemos notar que, ao final do cenário, um ícone de erro deve ser exibido. Esse ícone corresponde a um componente de interface atribuído a determinado elemento de CDS no momento da geração do formulário, mais especificamente na etapa de atribuição de componentes genéricos. O pseudo-código abaixo apresenta o processo de atribuição de componentes para os elementos de CDS.

```
1  method atribuiComponentesDeCDS (componentesGenericos) {
2      for (componentesGenericos) {
3          elementoDeDominio = componenteGenerico
4              .getElementoAssociado();
5          if (CDSModule.existeAgente(elementoDeDominio)) {
6              agente = CDSModule.getAgente(elementoDeDominio);
7              cdsComponent = Ontology.getCDSComponent(agente);
8              if (cdsComponent != null) {
9                  componenteGenerico.addFilho(cdsComponent);
10             }
11         }
12     }
13 }
```

Figura 38: Pseudo-código da atribuição de componentes para elementos de CDS

Para cada elemento de domínio do documento é verificado se existem agentes de CDS associados ao elemento, mediante informações resgatadas do módulo de CDS. Caso exista algum agente, a ferramenta verifica na ontologia de interface se existem componentes de interface para determinado agente. Caso exista, esse componente deve ser atribuído e ter seus atributos e eventos setados, de acordo com o descrito na ontologia.

No caso do cenário descrito na seção anterior, quando o elemento "freqüencia cardíaca" fosse selecionado, o MedViewGen verificaria a existência de algum agente de CDS para ele. O elemento de nome *cds_error* é então retornado. Ao buscar componentes de interface para esse agente, um componente denominado *error_icon* (ícone de erro) é atribuído. Esse componente tem seus atributos e eventos setados e é então adicionado na árvore de componentes genéricos.

É importante observar que o componente é instanciado logo na construção do formulário, e não no momento em que o usuário está preenchendo o formulário.

O componente fica invisível até que algum evento altere um de seus atributos e o torne visível para o usuário.

4.3.2 Inserção de Eventos

Como vimos anteriormente, para que o CDS entre em ação, é necessário que algum evento do usuário seja lançado, capturado e processado. Para que isso ocorra, é necessário que, no momento da geração do formulário, sejam configurados eventos gatilho associados aos componentes de interface.

A configuração de um evento abrange a habilitação do componente para lançar o evento e a criação de um *listener* (escutador) para esse evento. Voltando ao exemplo do cenário, para a caixa de texto referente ao conceito "freqüência cardíaca", deve haver um evento gatilho configurado. Esse evento ocorrerá sempre que o componente perder o foco (evento *lostFocus*) e será capturado por um *listener* específico para receber o evento. A **Figura 39** apresenta um pseudo-código com o método que faz a configuração dos eventos gatilho.

```
1  method configuraEventos (componentesGenericos) {
2      for (componentesGenericos) {
3          elementoDeDominio = componenteGenerico
4              .getElementoAssociado();
5          cdsEvent = cdsManager.getEvents(elementoDeDominio);
6          guiEvents = cdsEvent.getGuiEvents();
7          for (CDSModule.existeAgente(elementoDeDominio)) {
8              genericEvent = new GenericUIEvent();
9              geneticEvent.setNome(guiEvent.getName());
10             genericEvent.setAção("uiManager.throwCDSEvent");
11             genericEvent.addArgumento(elementoDeDominio);
12
13             componenteGenerico.addEvento(genericEvent);
14         }
15     }
16 }
```

Figura 39: Pseudo-código da configuração dos eventos

Para configurar os eventos de gatilho é necessário saber quais eventos estão habilitados para cada conceito biomédico. Novamente, quem fornece essas informações é o módulo de CDS, que as contém modeladas em uma de suas ontologias. Como podemos ver no pseudo-código da **Figura 39**, para cada componente de interface pertencente à estrutura gerada para o documento, o método extrai a IRI do conceito

associado (variável *elementoDeDominio*) e recupera os eventos associados ao conceito através da interface *cdsManager* (componente *CDSModule* da arquitetura apresentada no capítulo 3). O objeto *cdsEvent* recuperado pode conter um ou mais eventos de interface. Por exemplo, associado ao conceito "freqüênciacardíaca", além do evento *lostFocus* poderíamos ter outro evento qualquer associado. Para cada um dos eventos de interface da lista, o método associa um objeto da classe *GenericUIEvent*, que encapsula as informações necessárias para a configuração do evento, que ocorrerá quando os componentes forem instanciados em *UIBuilder* (ver **Figura 9**).

Associada ao evento, está a chamada do método *throwCDSEvent*. Esse método, pertencente ao componente *UIManager* da arquitetura do MedViewGen, é o responsável por repassar os eventos para o módulo de CDS e é chamado sempre que qualquer evento do formulário é lançado. Na próxima seção veremos o que esse método faz e como funciona o CDS durante a interação do usuário.

4.3.3 CDS em execução

Depois que o formulário é gerado, o usuário passa então a entrar dados através dele. Alguns dos campos do formulário podem conter validadores ou outro tipo de suporte à decisão, que irão fornecer informações ao usuário de acordo com suas ações e com os dados de entrada fornecidos. Essas informações são fornecidas por meio de alterações na configuração anterior do formulário, como a mudança de um determinado dado exibido ou a renderização de um novo componente, por exemplo.

Vamos utilizar o cenário descrito no início da seção 4.3 para ilustrar o processo. O usuário, ao preencher o campo freqüência cardíaca e passar ao próximo, faz com que um evento seja lançado, conforme configuração exemplificada na seção 4.3.2. O *listener* configurado para tal conceito irá capturar tal evento e executar o método *throwCDSEvent*, passando como parâmetro a IRI do conceito "freqüência cardíaca" e o valor de entrada. O controle é então repassado ao módulo de CDS que será responsável pelo processamento. Enquanto isso, o usuário continua suas atividades normalmente. Caso o valor entrado pelo usuário seja considerado válido, nada acontece. Caso contrário, o módulo de CDS envia uma mensagem de volta à GUI, informando a ocorrência de um *cdsError*.

Nesse momento, o controle é repassado ao *UIManager*, e o método *executeGUIEvent* é executado. Na **Figura 40** temos uma adaptação do código do método para ilustrar o processo.

```
public void executeGUIEvent(String cdsEventName, String iri,
        List<Object> message, Map<String, String> properties,
        IAppContext appContext) {

    CDSAction cdsAction = eventManager.getCDSAction(cdsEventName);
    List<GUIAction> guiActions = cdsAction.getGuiActions();

    for (GUIAction guiAction : guiActions) {

        ActionEvent actionEvent = guiAction.getActionEvent();

        OntologyElement ontologyElement = (OntologyElement) document.findThing(iri).get(0);
        String fieldPath = documentFieldPathManager.getFieldPath(ontologyElement);

        String componentIdPrefix = guiAction.getTargetComponentId();
        String genericComponentName = targetComponent.getComponentDescription().getComponentName();
        List<GenericUIAttribute> genericParameters = actionEvent.getParameters();

        List<ConcreteUIAttribute> concreteParameters = uiConverter.convertAttributes(
                    guiAction.getTargetComponent().getComponentDescription(),
                    genericParameters, uiBuilder.getTechnologyDescription());

        GenericUIComponent genericFatherComponent =
                genericUIStructure.findGenericComponent(fieldPath, GenericUIStructure.FIELD_PATH);

        String id = genericFatherComponent.findChildComponent(componentIdPrefix).getId();
        UIComponent uiComponent = uiBuilder.getComponent(id);

        uiBuilder.renderiza(uiComponent, concreteParameters);
}
```

Figura 40: Trecho de código adaptado do método executeGUIEvent

Como mostrado na figura, a primeira etapa do processo é recuperar da ontologia de interface quais são os *GUIActions* associados ao evento de CDS. Para cada *GUIAction* recuperado da ontologia, é extraído o *ActionEvent* associado e o prefixo identificador do componente que sofrerá a ação.

Com o *ActionEvent* em mãos, podemos saber quais parâmetros do componente alvo serão alterados. Como os parâmetros são recuperados da ontologia de componentes genéricos, a lista possui os nomes genéricos dos atributos, que devem ser convertidos em nomes concretos da tecnologia utilizada pelo usuário.

Para que as alterações sejam concluídas, é necessário renderizar as alterações na interface do usuário. A partir do prefixo recuperado da ontologia, o componente alvo é encontrado, como mostrado na penúltima linha. A partir daí, as alterações em seus parâmetros são executadas e exibidas no ambiente do usuário.

4.4 Considerações Finais

Neste capítulo foram apresentados aspectos de implementação da ferramenta, como a estrutura das ontologias utilizadas, os algoritmos de atribuição de componentes e de mapeamento e os processos associados ao CDS. Cada classe pertencente a cada

ontologia foi descrita com relação à sua estrutura e propósito. Alguns trechos de código foram adaptados para facilitar a compreensão, porém outros correspondem aos originais. Nas próximas seções apresentaremos os resultados obtidos após a conclusão da fase de implementação.

Capítulo

5

Resultados

"O sucesso é uma conseqüência, não um objetivo"

Gustave Flaubert

Como vimos no capítulo 3, o MedViewGen foi desenvolvido no contexto do projeto OpenCTI. De acordo com a arquitetura do OpenCTI, a camada de apresentação é dividida em uma parte estática, responsável pelas tarefas básicas do sistema, e uma parte dinâmica, responsável pelos formulários dos documentos. Tanto para a parte estática quanto para a parte dinâmica está sendo usada a tecnologia *Java Server Faces* (JSF) como tecnologia base da camada de apresentação.

Foram implementados todos os componentes da arquitetura proposta, utilizando a linguagem Java e os Frameworks *EJB* e *Seam*. Para integrar o MedViewGen ao OpenCTI, e assim validar a funcionalidade da ferramenta, criamos uma instância da interface G*UIBuilder* (ver seção 3.2.3) para JSF, tornando possível assim a geração de formulários para documentos nesta tecnologia específica.

No Capítulo 4 descrevemos os modelos ontológicos para as ontologias de componentes genéricos e de mapeamento para componentes concretos. Esses modelos representam a estrutura da semântica da informação. Porém, esses modelos são esquemas que devem ser preenchidos com informações para poderem ter utilidade real. Criamos então uma instância do modelo ontológico de componentes genéricos (seção 4.1.1) com um conjunto pré-estabelecido de componentes e suas características instrinsecas. Instanciamos também a ontologia de mapeamentos para dar suporte aos componentes JSF, especificando a correspondência de cada componente genérico em um componente específico da tecnologia.

Diferentemente do componente *GUIBuilder*, que precisa de uma instância

específica para cada tecnologia de interface, as implementações de *GUIManager*, *GenericGUIBuilder* e *GUIConverter* são únicas, independente de qual seja a tecnologia de interface em questão.

Ao todo, foram modelados 5 tipos de documentos: Ficha de admissão, ficha de evolução médica, ficha de evolução fisioterapêutica, certidão de óbito e histórico da enfermagem. Dentre essas fichas, escolhemos a certidão de óbito para especificar de forma mais detalhada e exercitar a ferramenta. Devido aos prazos para a conclusão do presente trabalho, não foi possível realizar validações quanto à performance. Nas subseções seguintes apresentaremos em detalhes os resultados obtidos com a implementação do MedViewGen.

5.1 Geração de Formulários

Durante a etapa de análise de modelos de fichas médicas da UTI do HULW, foi observada a necessidade de alguns componentes essenciais para a geração dos formulários, de forma que quaisquer formulários de documentos com características similares de apresentação possam ser gerados a partir de um conjunto pré-estabelecido de componentes. A **Tabela 1** apresenta uma lista com os componentes utilizados e sua descrição.

Componente Genérico	Descrição do Componente
CheckBox	Caixa de seleção de valor booleano
ComboBox	Caixa de seleção de um valor em uma lista
Icon	Ícone de exibição de uma imagem
InputText	Caixa de entrada de texto
OutputText	Campo de saída de texto
Panel	Painel agrupador de componentes
Panelgrid	Painel agrupador de componentes com *layout* de grade
TabContainer	Painel que contém uma ou mais abas
TabPanel	Aba

Tabela 1: Lista de componentes genéricos utilizados

No momento da atribuição de um componente genérico para determinado elemento de domínio, o algoritmo de atribuição utiliza informações descritas na ontologia para determinar qual o componente correto. A **Tabela 2** apresenta a lista dos componentes genéricos habilitados para cada tipo de elemento de domínio (*Domai-*

nElement).

Elemento de Domínio	Componentes Genéricos Habilitados
AbstractBiomedicalConcept	Panel
Archetype	Panel
ConcreteBiomedicalConcept	InputText
DataProperty	OutputText
DocumentBody	TabContainer
DocumentHeader	Panel
QualitativeBiomedicalConcept	ComboBox, CheckBox, InputText
QuantitativeBiomedicalConcept	InputText
Section	TabPanel
Unit	OutputText

Tabela 2: Lista de componentes habilitados para cada tipo de elemento de domínio

Para cada tipo de elemento de domínio existem um ou mais componentes habilitados a representá-los na GUI. No entanto, apenas um componente pode ser atribuído a cada elemento de domínio. Essa escolha é feita com base no tipo de valor (*valueType*) e no tipo de interação (*interactionType*) que cada componente pode representar. A **Tabela 3** apresenta os tipos de valor e de interação de cada componente.

Componente Genérico	Tipo de Valor	Tipo de Interação
CheckBox	boolean	many_of_a_list
ComboBox	string	one_of_a_list
Icon	image	output
InputText	float, integer, string	free
OutputText	float, integer, string	output
Panel	-	output
Panelgrid	-	output
TabContainer	-	output
TabPanel	-	output

Tabela 3: Tipos de valor e de interação para cada componente genérico

Existem mais informações na ontologia de interface além das apresentadas neste capítulo. A descrição OWL da ontologia completa está no Apêndice A.

Alguns pontos relevantes merecem destaque com relação aos dados da **Tabela 3**. Como podemos observar, os elementos do tipo *Container* (isto é, componentes que agrupam outros componentes) não possuem tipo de valor associado. Isso acontece porque eles não estão habilitados a representar dados atômicos, como valores numéricos ou textuais. O tipo de interação *output* indica que o componente funciona apenas como saída de dados, ou seja, nenhum dado pode ser entrado diretamente através dele.

Outro ponto importante é a diferenciação entre os componentes *comboBox* e *checkBox*, com relação ao tipo de interação. O *comboBox* é um componente utilizado para a seleção de conceitos mutuamente exclusivos. Apenas uma opção dentre as presentes na lista pode ser selecionada (*one_of_a_list*). O *checkBox* por sua vez permite que várias das opções presentes na lista sejam selecionadas ao mesmo tempo (*many_of_a_list*). Ou seja, utiliza-se *checkBox* quando as opções não são mutuamente exclusivas. Para identificar qual desses dois componentes deve ser utilizado, o algoritmo de atribuição checa se o elemento de domínio possui uma lista de exclusão mútua (*MutualExclusionList*). Caso essa lista exista, os elementos que estejam dentro dessa lista são atribuídos como opções dentro de um *comboBox*. Caso contrário, para cada uma das opções é atribuído um *checkBox* afim de possibilitar a seleção mútua entre eles.

Os dois tipos de entrada de dados apresentados (*many_of_a_llist* e *one_of_a_list*) possuem restrição quanto aos dados que podem ser entrados, pois dispõem de uma lista pré-definida de valores que o usuário pode selecionar. O tipo de interação *free* indica que o usuário pode escolher de forma livre os dados que serão entrados, ou seja, em texto livre.

Com a árvore de componentes genéricos montada, resta instanciar os componentes. Como a tecnologia JSF foi escolhida para a validação da ferramenta, a etapa de conversão dos componentes (*GUIConverter*) deve traduzir cada componente Genérico em um componente JSF, para posterior instanciação. A **Tabela 4** apresenta a lista de componentes utilizados, indicando o nome do componente genérico e o caminho utilizado para a instanciação do componente concreto.

Componente Genérico	Caminho do Componente Concreto
CheckBox	javax.faces.component.html.HtmlSelectBooleanCheckbox
ComboBox	javax.faces.component.html.HtmlSelectOneMenu
Icon	javax.faces.component.html.HtmlPanelGroup
InputText	javax.faces.component.html.HtmlInputText
OutputText	javax.faces.component.html.HtmlOutputText
Panel	org.primefaces.component.panel.Panel
Panelgrid	javax.faces.component.html.HtmlPanelGrid
TabContainer	org.primefaces.component.tabview.TabContainer
TabPanel	org.primefaces.component.tabview.Tab

Tabela 4: Lista de componentes utilizados

Além dos componentes em si, seus atributos e eventos devem ser mapeados de nomes genéricos para nomes correspondentes na tecnologia JSF. Como cada componente possui vários atributos e eventos, não colocaremos aqui a listagem dos atributos e eventos utilizados. Os mapeamentos de componentes, atributos e eventos utilizados podem ser encontrados no Apêndice A.

A partir da lista de componentes concretos, a instância *GUIBuilder* para JSF gera o formulário para exibição no ambiente do usuário.Uma cópia da estrutura e mantida no servidor para que alterações possam ser realizadas enquanto o usuário interage com o sistema.

Algumas dificuldades de implementação relativas à ligação do JavaBean com o formulário JSF foram encontradas, devido ao dinamismo dos campos e conseqüente impossibilidade da programação (a priori) das classes Java do *bean* de forma estática e manual. Soluções envolvendo criação de classes por reflexão surgiram como alternativa para contornar esse problema, porém a solução encontrada mais adequada foi programação, compilação e carga automática em tempo de execução das classes do *bean*, utilizando a API Javassist (Javassist, 2010). No momento da geração, o bean é programado dinamicamente, de acordo com o formulário gerado, compilado pela API e posto em atividade. A partir daí, as alterações nos dados do documento feitas pelo usuário são armazenadas no bean, ficando aptas para serem persistidas no banco de dados e/ou utilizadas para outros fins.

Outra característica relevante implementada na ferramenta é o armazenamento do formulário gerado para uso futuro. O processo de geração demonstrou ser

um pouco demorado para uso em aplicações online que requerem velocidade de acesso. Se a cada requisição de visualização de um formulário, todo o processo de geração tiver de se repetir, pode surgir a partir daí um gargalo para o sistema, acarretando em demora para o usuário. Para resolver tal problema, o formulário, quando gerado pela primeira vez, é serializado e armazenado em cache. Nas próximas requisições, o formulário é apenas enviado para o usuário, desde que nenhuma das ontologias tenha sofrido alterações. Nesse caso, a ferramenta deverá limpar o cache e regerar todos os formulários novamente, à medida que forem requisitados.

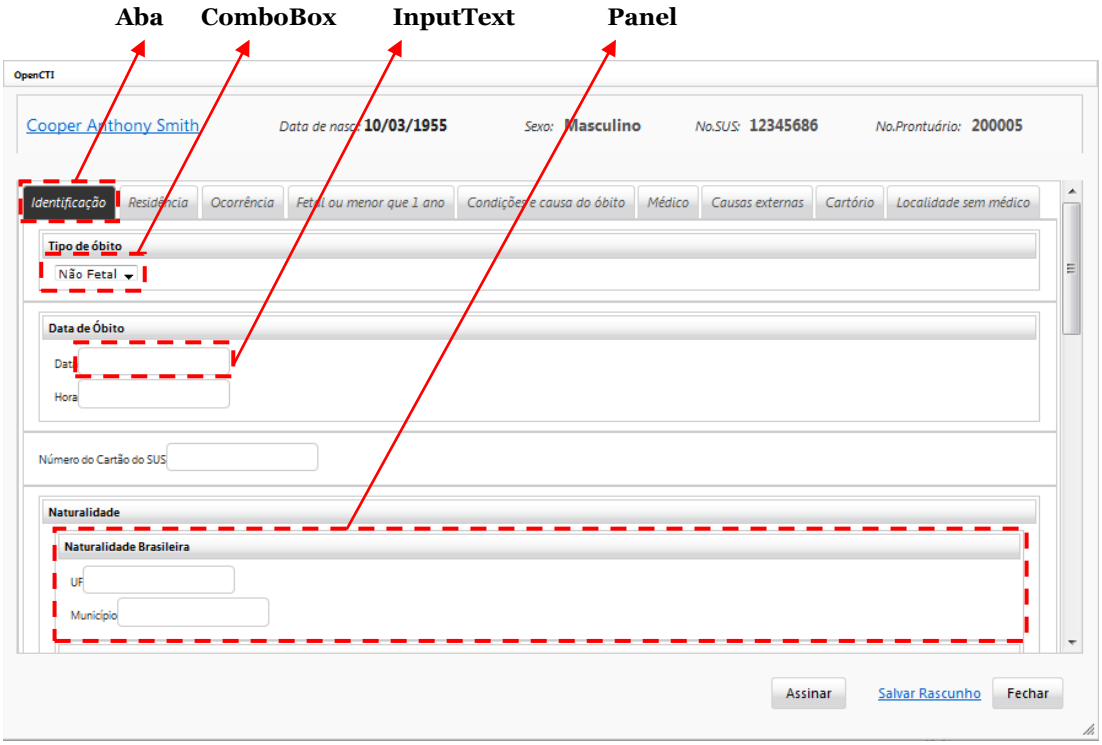

Figura 41: Formulário gerado para o documento de certidão de óbito

A **Figura 41** apresenta o formulário gerado para o documento de certidão de óbito. Como podemos ver, o formulário contém várias abas (componente genérico *TabPanel*), uma para cada seção do documento, como descrito na ontologia de interface. Dentro de cada aba, temos um conjunto de painéis (componente *Panel*) que podem representar uma série de tipos de conceitos. Quanto mais níveis existirem na hierarquia dos conceitos biomédicos, mais painéis aninhados são atribuídos. No caso do conceito "tipo de óbito" existe apenas um nível hierárquico, porém, para o conceito "naturalidade" podemos ver que existem pelo menos dois níveis hierárquicos.

Para os conceitos dos tipos *QualitativeBiomedicalConcept* e *QuantitativeBiomedicalConcept* temos dois casos neste formulário. No caso do conceito "Tipo de

óbito" temos uma lista de possíveis valores, e esses valores são mutuamente exclusivos entre si, fato que fez com que o componente *comboBox* fosse atribuído. Já para o campo "Data do óbito" tempos dois campos de entrada de dados sem lista de valores possíveis especificada. Nesse caso, foi atribuído para cada campo (*dataProperty*) um componente *inputText*, possibilitando a entrada de dados em texto livre.

Há ainda alguns componentes do tipo *checkBox* em uma parte do formulário não exibida na figura. Todas as abas desse formulário estão disponibilizadas no Apêndice C.

5.2 Integração com o módulo de CDS

Como forma de validação da ferramenta de CDS associada à interface gráfica foram implementados dois agentes. O primeiro deles é um validador associado ao conceito "freqüência cardíaca", presente na ficha de admissão. Esse validador vai indicar se o valor de entrada utilizado está ou não dentro de uma faixa de valores permitidos. Caso não esteja, um ícone de erro deve ser exibido, conforme mostra a **Figura 42**.

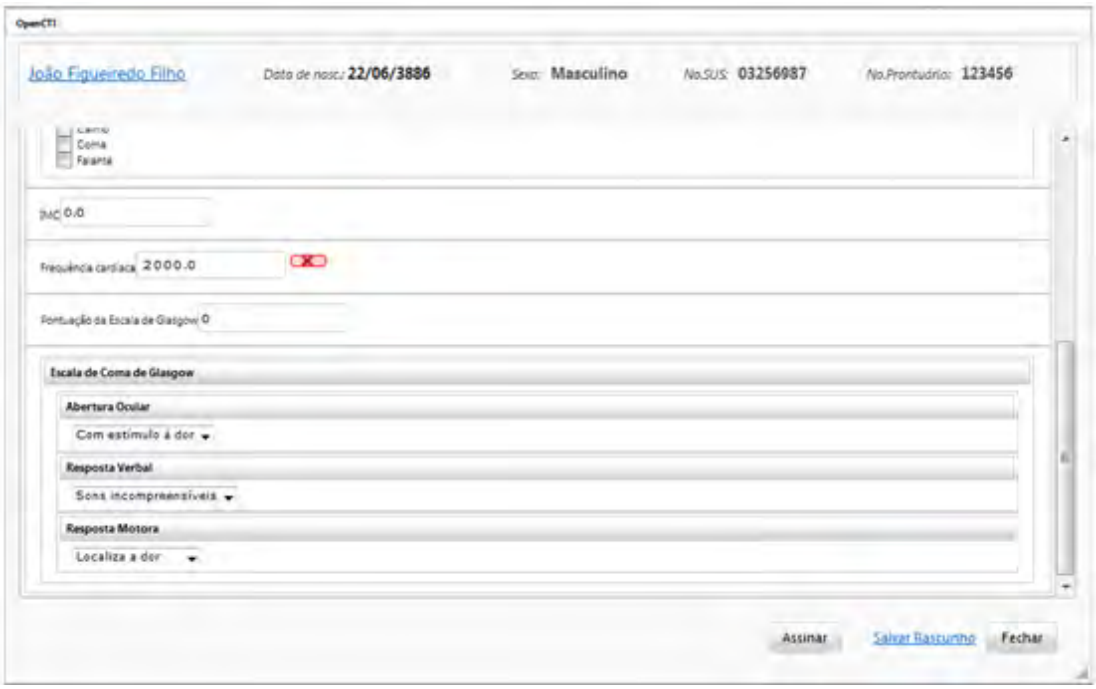

Figura 42: CDSError para o conceito freqüência cardíaca

Como podemos ver, o valor **2000.0** está fora da faixa de valores permitidos, que no caso desse agente específico começa a partir do valor **0.0** e vai até o valor **300.0**. Ao entrar com um valor dentro da faixa, o ícone de erro some.

O segundo agente calcula o valor do campo "IMC" (índice de massa corpórea), no mesmo documento, com base nos valores dos campos "peso" e "altura". O

calculo é feito dividindo-se o peso pelo quadrado da altura. A **Figura 43** mostra o valor do IMC calculado automaticamente a partir dos valores de peso (80kg) e altura (1.70m). Como resultado, temos o valor do campo IMC (27.68166).

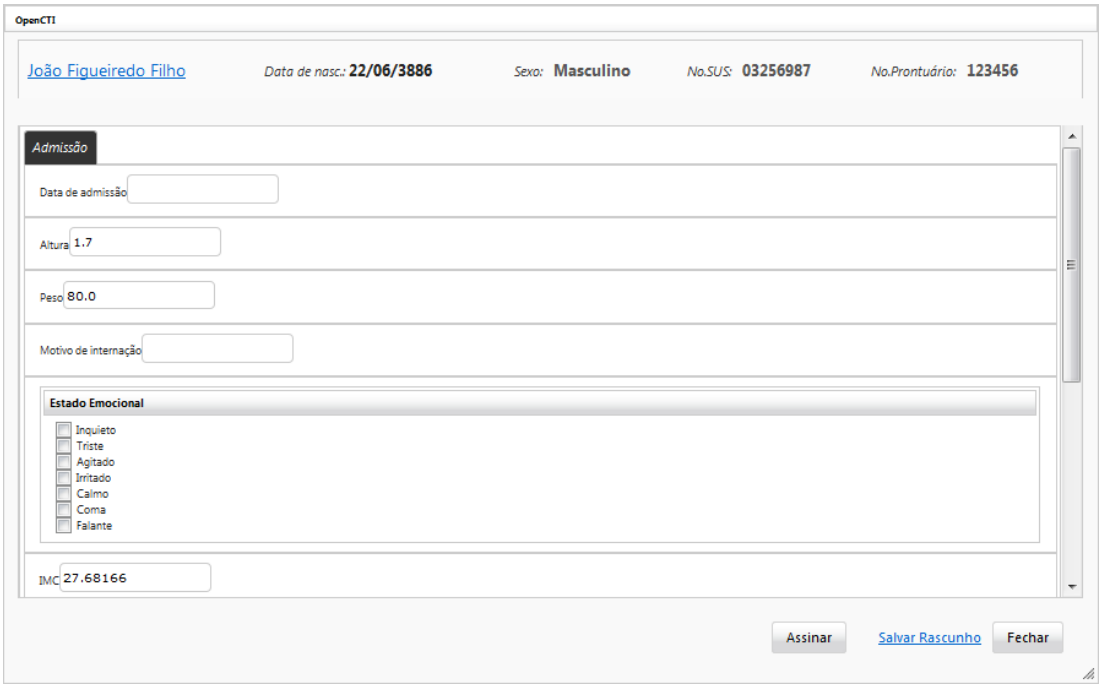

Figura 43: CDSFulfill para o conceito IMC

5.3 Considerações Finais

Neste capítulo apresentamos os resultados obtidos a partir da implementação da ferramenta e de sua integração ao OpenCTI, utilizando a tecnologia JSF para validar as funcionalidades desenvolvidas. Devido à grande quantidade de documentos que o OpenCTI possui, apenas alguns com características mais relevantes foram utilizados, com a finalidade de demonstrar a aplicabilidade da ferramenta. Nos próximos capítulos teremos a discussão e as conclusões do trabalho.

Capítulo

6

Considerações Finais

"A persistência é o caminho do êxito"

Charles Chaplin

Até aqui apresentamos cada etapa do processo de concepção da ferramenta MedViewGen, e os resultados obtidos a partir de sua utilização no OpenCTI. A seção 6.1 apresenta uma breve discussão dos principais pontos observados, no que diz respeito principalmente à geração dos formulários e à integração com o modulo de CDS, comparando os resultados obtidos com os trabalhos presentes na literatura. A seção 6.2 apresenta as conclusões do trabalho e na seção 6.3 temos os trabalhos futuros que se seguirão tendo como base o presente trabalho.

6.1 Discussão

Para melhor organizar as idéias, dividimos a discussão por assunto. Nas seções 6.1.1 e 6.1.2 apresentamos as considerações relevantes sobre a geração de formulários e a integração da interface gráfica com o módulo de CDS, respectivamente. Na seção 6.1.3 apresentamos uma comparação entre o MedViewGen e os trabalhos relacionados.

6.1.1 Geração dos Formulários

A meta da geração automática da GUI é reduzir a preocupação dos modeladores em definir a forma como os formulários serão montados e como os conceitos serão exibidos. Com a utilização de uma ontologia de interface para componentes genéricos, demonstramos que pouca ou nenhuma alteração é preciso ser feita para que um novo tipo de documento possa ser habilitado pelo sistema. A ontologia descreve as ca-

racterísticas dos componentes de interface e indica quais tipos de elementos de domínio que eles estão habilitados a representar. Um novo tipo de documento irá possuir, em geral, esses mesmos tipos de elementos (seção, subseção, arquétipo, etc.) e, em geral, irá necessitar do mesmo conjunto de componentes para construir seu formulário. Logo, podemos observar que temos um ganho em escala com a utilização da ferramenta, pois independentemente do número de tipos de documentos, a mesma ontologia poderá gerar formulários para todos eles.

Com relação à utilização do MedViewGen para a geração em diversas tecnologias, devido a algumas limitações de tempo do projeto apenas o módulo para JSF foi desenvolvido, impossibilitando a comparação em mais de uma tecnologia de interface. Contudo, a implementação em JSF apresentou resultados satisfatórios, conseguindo criar formulários para o conjunto inicial de documentos do OpenCTI, embora ainda sejam necessários refinamentos da forma de apresentação dos documentos, organização dos componentes na tela e utilização de layouts, de forma a aumentar a usabilidade e a aceitação do usuário.

É importante ressaltar que este trabalho teve como foco desenvolver uma ferramenta que se limita a fornecer a infra-estrutura necessária para a geração de interfaces de forma automática e provar a eficácia dos métodos utilizados. Refinamentos conseqüentes de trabalhos futuros ainda devem ser realizadas com a finalidade de melhorar a qualidade dos formulários gerados.

Outra limitação da ferramenta que merece destaque é que, dependendo da forma como a ontologia for configurada, mais de um componente pode ser atribuído a um mesmo conceito durante o processo de geração da GUI, porém a ferramenta não dá suporte à escolha de componente neste caso. É necessário um acréscimo ao algoritmo de atribuição que torne possível fazer essa escolha de forma consistente.

6.1.2 Integração com o CDS

A abordagem para CDS utilizada no OpenCTI tem como principal padrão o fraco acoplamento entre a camada de apresentação e a camada de aplicação. Com a integração da GUI com o módulo de CDS proporcionado pelo MedViewGen, é possível ganhar em escalabilidade quando muitos campos possuem agentes de CDS habilitados, pois uma mesma configuração de componentes de interface para determinado agente de CDS pode atender a diversos conceitos biomédicos e vários documentos. Isso acontece porque na ontologia temos apenas uma configuração para cada tipo de agente, que pode ser reutilizado em vários conceitos biomédicos, e não uma confi-

guração para par conceito-agente. As informações referentes a quais conceitos estão associados a quais agentes é responsabilidade do módulo de CDS.

Um ponto importante com relação à atribuição de componentes para os elementos de CDS a ser observado é que esse processo é realizado antes que o CDS enter em ação de fato, ou seja, os componentes já estão presentes no formulário (de forma invisível) antes mesmo que o usuário integaraja com o sistema e gere eventos que ativem o suporte a decisão. Essa escolha foi feita com o intuito de tornar a interação do usuário com o sistema mais rápida e agradável, deixando o gargalo da atribução para o momento da abertura do formulário, quando já é esperado que demore mais um pouco. Caso os componentes fossem atribuídos durante a interação do usuário, os acessos à ontologia e a instanciação dos componentes poderiam em alguns momentos de sobrecarga do servidor acarretar em lentidão da ferramenta de suporte a decisão.

Com relação à interação do usuário com o formulário, quando um evento é gerado, o servidor o captura, processa e devolve uma resposta ao cliente. Em alguns casos essa troca de mensagens entre cliente e servidor poderia ser evitada, inserindo-se no próprio formulário gerado os códigos que o servidor executa para processar os eventos e lançar a resposta. Nos casos apresentados no capítulo de resultados, tanto o *cds_error* quanto o *cds_fulfill* poderiam ser executados no browser, uma vez que são apenas comparações de valores e cálculos simples. Porém, alguns agentes de CDS mais complexos (ainda não finalizados, mas com implementação em andamento) requerem buscas de informações em bancos de dados remotos com base nos dados entrados pelo usuário. Com isso, resolvemos manter o processamento do CDS no servidor, embora futuramente alterações devam ser realizadas buscando otimizar caso a caso.

6.1.3 Comparação com trabalhos relacionados

Como mencionado em capítulos anteriores, o MedViewGen faz parte de uma arquitetura mais abrangente relacionada ao OpenCTI. Essa arquitetura tem como característica a ampla utilização de ontologias em diversas camadas do EHR, inclusive na GUI. Essa escolha ocorreu graças ao grande poder de representação que as ontologias OWL oferecem quando se deseja armazenar e recuperar semântica intrínseca dos conceitos biomédicos.

De acordo com os trabalhos relacionados apresentado na seção 2.4.1, ambas as abordagens neles utilizadas são baseadas em arquétipos, diferenciando apenas no modelo de definição de arquétipos utilizada. Para cada conceito presente no domínio

deve haver uma forma de representação, modelada também como arquétipo, que apresenta a formatação genérica da GUI para a representação do conceito. Este tipo de abordagem não é desejável quando o profissional que realiza a definição do domínio é especialista apenas na área de saúde ou apenas na área tecnológica. Isso se justifica pelo fato de que quem melhor sabe como representar a terminologia biomédica utilizada deve ser especialista no subdomínio clínico correlato, portanto, um profissional da área de saúde (possivelmente um médico especialista na área). Por outro lado, um profissional da área de saúde provavelmente não possui a mesma destreza para definir a melhor forma de representação daqueles conceitos para os usuários do sistema, uma vez que não possui (em geral) conhecimentos sobre interfaces gráficas e usabilidade, dentre outros necessários à construção de uma boa interface com o usuário. De forma recíproca, um profissional da área de tecnologia (um *web designer*, por exemplo) tem formação e conhecimentos suficientes sobre usabilidade e como as informações podem ser melhor apresentadas para o usuário, porém não tem noção de como os conceitos devem ser estruturados.

A utilização do MedViewGen como forma de descartar a necessidade de um profissional da área de interface na modelagem do domínio tem se mostrado uma abordagem promissora. Pequenas alterações realizadas nas ontologias de domínio não requeriram necessidade de alteração na ontologia de interface, o que demonstra que para o conjunto de conceitos utilizados houve redução do trabalho do modelador.

O trabalho de Van der Linden (2009) apresenta uma melhor modularização em relação ao trabalho de Schuler (2006), separando as estruturas de definição dos conceitos da sua forma de apresentação, fazendo a ligação entre as duas através de um mapeamento definido separadamente. Essa abordagem adotada facilita o reuso de formas de exibição por vários conceitos e possibilita a criação de várias combinações de componentes, mas ainda assim é necessário definir manualmente para cada conceito qual a sua forma de apresentação na GUI.

Uma característica importante no trabalho de Van der Linden é o foco na saída de dados (exibição). O trabalho de Schuler é mais genérico neste sentido. O trabalho aqui proposto foca nas características de entrada de dados não abordadas em profundidade por Van der Linden, buscando possibilitar interação cada vez mais reativa do usuário com o EHR.

Com relação à integração da interface com ferramentas de suporte a decisão clínica (CDS), o trabalho de Schuler apresenta uma abordagem onde validadores simples podem ser descritos juntamente com a descrição dos elementos de gráficos,

que por sua vez estão atrelados à descrição dos conceitos biomédicos. Essa abordagem peca em dois aspectos: Simplicidade e acoplamento forte entre aplicação e apresentação. Primeiramente, como apenas validadores simples podem ser implementados, a limitação quanto ao suporte à decisão é muito grande. A utilização do MedViewGen possibilita que agentes mais complexos como cálculos, algoritmos mais gerais, baseados em técnicas de inteligência artificial, e consultas a bancos de dados remotos sejam suportadas pelo EHR. Quanto ao forte acoplamento, como a descrição dos validadores está atrelada à própria definição do conceito biomédico, torna-se difícil separar uma e outra, podendo acarretar em problemas caso uma das duas camadas sofra grandes alterações. A **Tabela 5** apresenta um quadro comparativo entre os trabalhos relacionados.

	Schuler	Van der Linden	MedViewGen
Interface configurável	✔	✔	✔
Reuso dos elementos de apresentação	X	✔	✔
Saída de dados	✔	✔	✔
Entrada de dados	X	X	✔
Suporte para CDS	✔	X	✔
Modelo semântico de representação de componentes de interface	X	X	✔
Necessidade de configuração manual a cada alteração	✔	✔	X

Tabela 5: Comparativo dos trabalhos relacionados

6.2 Conclusões

Com o desenvolvimento da API de geração automática e a implementação do módulo para JSF, pudemos gerar, com um conjunto pré-estabelecido de componentes, formulários para os documentos até então suportados pelo OpenCTI, atingindo o objetivo 1 deste trabalho (seção 1.2), embora melhorias quanto à usabilidade e refinamentos nas interações do usuário sejam necessárias para tornar o MedViewGen uma ferramenta completa. De modo geral, a infra-estrutura necessária para a execução do CDS também demonstrou resultados satisfatórios. Um conjunto pequeno de agentes utilizados deu-se devido ao tempo limitado, porém este é um trabalho que deverá ser refinado posteriormente.

A arquitetura projetada demonstrou ser flexível o suficiente para comportar a tecnologia JSF e possibilitar a integração com o OpenCTI, atingindo os objetivo 2 e 4. Devido às limitações de tempo não foi possível desenvolver implementações em outras tecnologias, porém buscamos projetar da interface *IGUIBuilder* a mais genérica possível com a finalidade de poder comportar novas tecnologias de interface em trabalhos futuros.

Uma das principais contribuições deste trabalho foram os modelos semânticos estabelecidos, atingindo o objetivo 3. A partir destes modelos uma série de componentes de interfaces podem ser descritos. Instâncias do modelo ontológico podem ser criadas para domínios clínicos distintos do utilizado pelo OpenCTI, onde os documentos não obrigatoriamente devem estar organizados da mesma maneira e em termos dos mesmos elementos, tornando o MedViewGen uma ferramenta flexível nesse aspecto.

Com a realização deste trabalho ficou comprovada a aplicabilidade dos métodos utilizados na geração dos formulários dinamicamente, desde os algoritmos de atribuição e mapeamento até as ontologias modeladas. Acreditamos proporcionar uma aceleração no processo de criação de interfaces para documentos clínicos, tendo como principais contribuições o aumento da escalabilidade, a diminuição do acoplamento entre domínio e aplicação e a integração de ferramentas complexas de CDS.

6.3 Trabalhos Futuros

Alguns aspectos da ferramenta ainda podem ser melhorados utilizando-se a infraestrutura implementada e consolidada. Alguns trabalhos já estão sendo realizados por alunos de iniciação científica do LArqSS, no sentido de trabalhar sobre os layouts e da melhoria da interface como um todo, buscando aumentar a qualidade dos formulários gerados.

Um trabalho de refinamento do MedViewGen está sendo iniciado. Esse trabalho terá como objetivo quebrar algumas limitações e expandir os requisitos da ferramenta, tornando as interfaces geradas personalizáveis de acordo com os perfis dos usuários. Espera-se também poder proporcionar ao usuário a escolha do melhor componente em determinados casos onde mais de um componente é atribuído, caso este não suportado atualmente pela ferramenta.

Apêndice A

Ontologias OWL do MedViewGen

Ontologia de interface

A seguir apresentaremos o código OWL resultante da implementação da ontologia utilizada pela ferramenta para a atribuição de componentes genéricos para os formulários do OpenCTI.

```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY UIComponent "http://opencti.larqss.ufpb/UIComponent.owl#" >
]>


<rdf:RDF xmlns="http://opencti.larqss.ufpb/UIComponent.owl#"
     xml:base="http://opencti.larqss.ufpb/UIComponent.owl"
     xmlns:UIComponent="http://opencti.larqss.ufpb/UIComponent.owl#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology rdf:about=""/>



    <!--

///////////////////////////////////////////////////////////////////////////
//////////
    //
    // Object Properties
    //

///////////////////////////////////////////////////////////////////////////
//////////
     -->
```

```
<!-- http://opencti.larqss.ufpb/UIComponent.owl#activatedBy -->

<owl:ObjectProperty rdf:about="#activatedBy">
    <rdfs:label>activatedBy</rdfs:label>
</owl:ObjectProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#execute -->

<owl:ObjectProperty rdf:about="#execute">
    <rdfs:label>execute</rdfs:label>
</owl:ObjectProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#hasAttribute -->

<owl:ObjectProperty rdf:about="#hasAttribute">
    <rdfs:label>hasAttribute</rdfs:label>
</owl:ObjectProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#hasComponent -->

<owl:ObjectProperty rdf:about="#hasComponent">
    <rdfs:label>hasComponent</rdfs:label>
</owl:ObjectProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#hasDescription -->

<owl:ObjectProperty rdf:about="#hasDescription">
    <rdfs:label>hasDescription</rdfs:label>
</owl:ObjectProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#hasEvent -->

<owl:ObjectProperty rdf:about="#hasEvent">
    <rdfs:label>hasEvent</rdfs:label>
</owl:ObjectProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#hasInputComponent -->

<owl:ObjectProperty rdf:about="#hasInputComponent">
    <rdfs:label
        >hasInputComponent</rdfs:label>
</owl:ObjectProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#hasInteractionType -->

<owl:ObjectProperty rdf:about="#hasInteractionType">
    <rdfs:label
```

```
            >hasInteractionType</rdfs:label>
    </owl:ObjectProperty>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#hasParameter -->

    <owl:ObjectProperty rdf:about="#hasParameter">
        <rdfs:label>hasParameter</rdfs:label>
    </owl:ObjectProperty>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#hasValueFeatures -->

    <owl:ObjectProperty rdf:about="#hasValueFeatures">
        <rdfs:label
            >hasValueFeatures</rdfs:label>
    </owl:ObjectProperty>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#hasValueType -->

    <owl:ObjectProperty rdf:about="#hasValueType">
        <rdfs:label>hasValueType</rdfs:label>
    </owl:ObjectProperty>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#represents -->

    <owl:ObjectProperty rdf:about="#represents">
        <rdfs:label>represents</rdfs:label>
    </owl:ObjectProperty>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#target -->

    <owl:ObjectProperty rdf:about="#target">
        <rdfs:label>target</rdfs:label>
    </owl:ObjectProperty>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#throwEvent -->

    <owl:ObjectProperty rdf:about="#throwEvent">
        <rdfs:label>throwEvent</rdfs:label>
    </owl:ObjectProperty>



    <!--

///////////////////////////////////////////////////////////////////////////
///////////
    //
    // Data properties
    //

///////////////////////////////////////////////////////////////////////////
///////////
     -->
```

```
<!-- http://opencti.larqss.ufpb/UIComponent.owl#attributeName -->

<owl:DatatypeProperty rdf:about="#attributeName">
    <rdfs:label>attributeName</rdfs:label>
</owl:DatatypeProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#attributeValue -->

<owl:DatatypeProperty rdf:about="#attributeValue">
    <rdfs:label>attributeValue</rdfs:label>
</owl:DatatypeProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#componentName -->

<owl:DatatypeProperty rdf:about="#componentName">
    <rdfs:label>componentName</rdfs:label>
</owl:DatatypeProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#elementCategory -->

<owl:DatatypeProperty rdf:about="#elementCategory">
    <rdfs:label>elementCategory</rdfs:label>
</owl:DatatypeProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#elementName -->

<owl:DatatypeProperty rdf:about="#elementName">
    <rdfs:label>elementName</rdfs:label>
</owl:DatatypeProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#eventName -->

<owl:DatatypeProperty rdf:about="#eventName">
    <rdfs:label>eventName</rdfs:label>
</owl:DatatypeProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#idPrefix -->

<owl:DatatypeProperty rdf:about="#idPrefix">
    <rdfs:label>idPrefix</rdfs:label>
</owl:DatatypeProperty>



<!-- http://opencti.larqss.ufpb/UIComponent.owl#interactionTypeName -->

<owl:DatatypeProperty rdf:about="#interactionTypeName">
    <rdfs:label
        >interactionTypeName</rdfs:label>
```

```
    </owl:DatatypeProperty>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#isVisible -->

    <owl:DatatypeProperty rdf:about="#isVisible">
        <rdfs:label rdf:datatype="&xsd;boolean">isVisible</rdfs:label>
    </owl:DatatypeProperty>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#layoutDescription -->

    <owl:DatatypeProperty rdf:about="#layoutDescription">
        <rdfs:label
            >layoutDescription</rdfs:label>
    </owl:DatatypeProperty>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#paramName -->

    <owl:DatatypeProperty rdf:about="#paramName">
        <rdfs:label>paramName</rdfs:label>
    </owl:DatatypeProperty>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#paramValue -->

    <owl:DatatypeProperty rdf:about="#paramValue">
        <rdfs:label>paramValue</rdfs:label>
    </owl:DatatypeProperty>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#showDataProperties -->

    <owl:DatatypeProperty rdf:about="#showDataProperties">
        <rdfs:label
            >showDataProperties</rdfs:label>
    </owl:DatatypeProperty>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#valueTypeName -->

    <owl:DatatypeProperty rdf:about="#valueTypeName">
        <rdfs:label>valueTypeName</rdfs:label>
    </owl:DatatypeProperty>



    <!--

///////////////////////////////////////////////////////////////////////////
///////////
    //
    // Classes
    //

///////////////////////////////////////////////////////////////////////////
///////////
     -->
```

```
    <!-- http://opencti.larqss.ufpb/UIComponent.owl#ActionEvent -->

    <owl:Class rdf:about="#ActionEvent">
        <rdfs:label>ActionEvent</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Event"/>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#Attribute -->

    <owl:Class rdf:about="#Attribute">
        <rdfs:label>Attribute</rdfs:label>
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#attributeName"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#attributeValue"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#CdsAction -->

    <owl:Class rdf:about="#CdsAction">
        <rdfs:label>CdsAction</rdfs:label>
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#execute"/>
                <owl:someValuesFrom rdf:resource="#GuiAction"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#activatedBy"/>
                <owl:onClass rdf:resource="#CdsEvent"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#CdsEvent -->

    <owl:Class rdf:about="#CdsEvent">
        <rdfs:label>CdsEvent</rdfs:label>
```

```
        <rdfs:subClassOf rdf:resource="#DomainElement"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasComponent"/>
                <owl:someValuesFrom rdf:resource="#Component"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#Component -->

    <owl:Class rdf:about="#Component">
        <rdfs:label>Component</rdfs:label>
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasEvent"/>
                <owl:someValuesFrom rdf:resource="#Event"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasAttribute"/>
                <owl:someValuesFrom rdf:resource="#Attribute"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasDescription"/>
                <owl:onClass rdf:resource="#Description"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasValueType"/>
                <owl:someValuesFrom rdf:resource="#Component"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasInteractionType"/>
                <owl:onClass rdf:resource="#InteractionType"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#represents"/>
                <owl:someValuesFrom rdf:resource="#DomainElement"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <owl:disjointWith rdf:resource="#Description"/>
        <owl:disjointWith rdf:resource="#DomainElement"/>
        <owl:disjointWith rdf:resource="#InteractionType"/>
        <owl:disjointWith rdf:resource="#ValueFeatures"/>
        <owl:disjointWith rdf:resource="#ValueType"/>
    </owl:Class>
```

```
    <!-- http://opencti.larqss.ufpb/UIComponent.owl#Container -->

    <owl:Class rdf:about="#Container">
        <rdfs:label>Container</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Component"/>
        <owl:disjointWith rdf:resource="#Description"/>
        <owl:disjointWith rdf:resource="#DomainElement"/>
        <owl:disjointWith rdf:resource="#InteractionType"/>
        <owl:disjointWith rdf:resource="#ValueFeatures"/>
        <owl:disjointWith rdf:resource="#ValueType"/>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#Description -->

    <owl:Class rdf:about="#Description">
        <rdfs:label>Description</rdfs:label>
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#componentName"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#idPrefix"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#DomainElement -->

    <owl:Class rdf:about="#DomainElement">
        <rdfs:label>DomainElement</rdfs:label>
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#elementCategory"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isVisible"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;boolean"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#showDataProperties"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
```

```
                <owl:onDataRange rdf:resource="&xsd;boolean"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#elementName"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#Event -->

    <owl:Class rdf:about="#Event">
        <rdfs:label>Event</rdfs:label>
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#eventName"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasParameter"/>
                <owl:someValuesFrom rdf:resource="#Attribute"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#FocusEvent -->

    <owl:Class rdf:about="#FocusEvent">
        <rdfs:label>FocusEvent</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Event"/>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#GuiAction -->

    <owl:Class rdf:about="#GuiAction">
        <rdfs:label>GuiAction</rdfs:label>
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#target"/>
                <owl:onClass rdf:resource="#Component"/>
                <owl:maxQualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxQualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#throwEvent"/>
                <owl:onClass rdf:resource="#ActionEvent"/>
                <owl:qualifiedCardinality
```

```
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#InteractionType -->

    <owl:Class rdf:about="#InteractionType">
        <rdfs:label>InteractionType</rdfs:label>
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#interactionTypeName"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#KeyEvent -->

    <owl:Class rdf:about="#KeyEvent">
        <rdfs:label>KeyEvent</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Event"/>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#MouseEvent -->

    <owl:Class rdf:about="#MouseEvent">
        <rdfs:label>MouseEvent</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Event"/>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#ValueFeatures -->

    <owl:Class rdf:about="#ValueFeatures">
        <rdfs:label>ValueFeatures</rdfs:label>
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasValueType"/>
                <owl:someValuesFrom rdf:resource="#ValueType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#ValueType -->

    <owl:Class rdf:about="#ValueType">
        <rdfs:label>ValueType</rdfs:label>
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#valueTypeName"/>
```

```
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://www.w3.org/2002/07/owl#Thing -->

    <owl:Class rdf:about="&owl;Thing"/>



    <!--

////////////////////////////////////////////////////////////////////////////////
///////////
    //
    // Individuals
    //

////////////////////////////////////////////////////////////////////////////////
///////////
     -->




    <!--
http://opencti.larqss.ufpb/UIComponent.owl#abstract_biomedical_concept -->

    <DomainElement rdf:about="#abstract_biomedical_concept">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >abstract_biomedical_concept</rdfs:label>
        <elementName rdf:datatype="&xsd;string"
            >AbstractBiomedicalConcept</elementName>
        <elementCategory rdf:datatype="&xsd;string"
            >biomedicalConcepts</elementCategory>
        <showDataProperties
rdf:datatype="&xsd;boolean">false</showDataProperties>
        <isVisible rdf:datatype="&xsd;boolean">true</isVisible>
    </DomainElement>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#archetype -->

    <DomainElement rdf:about="#archetype">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>archetype</rdfs:label>
        <elementName rdf:datatype="&xsd;string">Archetype</elementName>
        <elementCategory
rdf:datatype="&xsd;string">documents</elementCategory>
        <showDataProperties
rdf:datatype="&xsd;boolean">false</showDataProperties>
        <isVisible rdf:datatype="&xsd;boolean">true</isVisible>
    </DomainElement>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#biomedical_concept -->
```

```
    <DomainElement rdf:about="#biomedical_concept">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >biomedical_concept</rdfs:label>
        <elementName rdf:datatype="&xsd;string"
            >BiomedicalConcept</elementName>
        <elementCategory rdf:datatype="&xsd;string"
            >biomedicalConcepts</elementCategory>
        <showDataProperties
rdf:datatype="&xsd;boolean">false</showDataProperties>
        <isVisible rdf:datatype="&xsd;boolean">true</isVisible>
    </DomainElement>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#boolean -->

    <owl:Thing rdf:about="#boolean">
        <rdf:type rdf:resource="#ValueType"/>
        <rdfs:label>boolean</rdfs:label>
        <valueTypeName
rdf:datatype="&xsd;string">xsd:boolean</valueTypeName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#cdsElement -->

    <DomainElement rdf:about="#cdsElement">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>cdsElement</rdfs:label>
        <elementCategory rdf:datatype="&xsd;string">CDS</elementCategory>
        <elementName rdf:datatype="&xsd;string">cdsElement</elementName>
        <showDataProperties
rdf:datatype="&xsd;boolean">false</showDataProperties>
        <isVisible rdf:datatype="&xsd;boolean">false</isVisible>
    </DomainElement>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#cdsError_Action -->

    <owl:Thing rdf:about="#cdsError_Action">
        <rdf:type rdf:resource="#CdsAction"/>
        <rdfs:label>cdsError_Action</rdfs:label>
        <execute rdf:resource="#cdsError_GuiAction_1"/>
        <activatedBy rdf:resource="#cds_error"/>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#cdsError_GuiAction_1 -->

    <owl:Thing rdf:about="#cdsError_GuiAction_1">
        <rdf:type rdf:resource="#GuiAction"/>
        <rdfs:label
            >cdsError_GuiAction_1</rdfs:label>
        <target rdf:resource="#errorIcon"/>
        <throwEvent rdf:resource="#setVisible_True"/>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#cdsFulfill_Action -->
```

```
    <CdsAction rdf:about="#cdsFulfill_Action">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >cdsFulfill_Action</rdfs:label>
        <execute rdf:resource="#cdsFulfill_GuiAction_1"/>
        <activatedBy rdf:resource="#cds_fulfill"/>
    </CdsAction>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#cdsFulfill_GuiAction_1 -
->

    <owl:Thing rdf:about="#cdsFulfill_GuiAction_1">
        <rdf:type rdf:resource="#GuiAction"/>
        <rdfs:label
            >cdsFulfill_GuiAction_1</rdfs:label>
        <throwEvent rdf:resource="#changeValue"/>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#cds_error -->

    <CdsEvent rdf:about="#cds_error">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>cds_error</rdfs:label>
        <elementCategory rdf:datatype="&xsd;string">cds</elementCategory>
        <elementName rdf:datatype="&xsd;string">cdsError</elementName>
        <showDataProperties
rdf:datatype="&xsd;boolean">false</showDataProperties>
        <isVisible rdf:datatype="&xsd;boolean">false</isVisible>
        <hasComponent rdf:resource="#errorIcon"/>
    </CdsEvent>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#cds_fulfill -->

    <CdsEvent rdf:about="#cds_fulfill">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>cds_fulfill</rdfs:label>
        <elementCategory rdf:datatype="&xsd;string">cds</elementCategory>
        <elementName rdf:datatype="&xsd;string">cdsFulfill</elementName>
        <isVisible rdf:datatype="&xsd;boolean">false</isVisible>
        <showDataProperties
rdf:datatype="&xsd;boolean">false</showDataProperties>
    </CdsEvent>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#changeValue -->

    <ActionEvent rdf:about="#changeValue">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>changeValue</rdfs:label>
        <eventName rdf:datatype="&xsd;string">changeValue</eventName>
    </ActionEvent>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#checkbox -->

    <owl:Thing rdf:about="#checkbox">
        <rdf:type rdf:resource="#Component"/>
```

```
        <rdfs:label>checkbox</rdfs:label>
        <hasValueType rdf:resource="#boolean"/>
        <hasDescription rdf:resource="#checkboxDescription"/>
        <hasInteractionType rdf:resource="#many_of_a_list"/>
        <represents rdf:resource="#qualitative_biomedical_concept"/>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#checkboxDescription -->

    <Description rdf:about="#checkboxDescription">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >checkboxDescription</rdfs:label>
        <idPrefix rdf:datatype="&xsd;string">CX</idPrefix>
        <componentName rdf:datatype="&xsd;string">checkBox</componentName>
    </Description>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#comboBox -->

    <Component rdf:about="#comboBox">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>comboBox</rdfs:label>
        <hasDescription rdf:resource="#comboBoxDescription"/>
        <hasInteractionType rdf:resource="#one_of_a_list"/>
        <represents rdf:resource="#qualitative_biomedical_concept"/>
        <hasValueType rdf:resource="#string"/>
    </Component>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#comboBoxDescription -->

    <owl:Thing rdf:about="#comboBoxDescription">
        <rdf:type rdf:resource="#Description"/>
        <rdfs:label
            >comboBoxDescription</rdfs:label>
        <idPrefix rdf:datatype="&xsd;string">CB</idPrefix>
        <componentName rdf:datatype="&xsd;string">comboBox</componentName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#concept_group -->

    <owl:Thing rdf:about="#concept_group">
        <rdf:type rdf:resource="#DomainElement"/>
        <rdfs:label>concept_group</rdfs:label>
        <elementName rdf:datatype="&xsd;string">conceptGroup</elementName>
        <showDataProperties
rdf:datatype="&xsd;boolean">false</showDataProperties>
        <elementCategory rdf:datatype="&xsd;string">gui</elementCategory>
        <isVisible rdf:datatype="&xsd;boolean">true</isVisible>
    </owl:Thing>



    <!--
http://opencti.larqss.ufpb/UIComponent.owl#concrete_biomedical_concept -->

    <owl:Thing rdf:about="#concrete_biomedical_concept">
        <rdf:type rdf:resource="#DomainElement"/>
```

```
        <rdfs:label
            >concrete_biomedical_concept</rdfs:label>
        <elementName rdf:datatype="&xsd;string"
            >ConcreteBiomedicalConcept</elementName>
        <elementCategory rdf:datatype="&xsd;string"
            >biomedicalConcepts</elementCategory>
        <isVisible rdf:datatype="&xsd;boolean">true</isVisible>
        <showDataProperties
rdf:datatype="&xsd;boolean">true</showDataProperties>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#data_property -->

    <DomainElement rdf:about="#data_property">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>data_property</rdfs:label>
        <elementName rdf:datatype="&xsd;string">DataProperty</elementName>
        <elementCategory
rdf:datatype="&xsd;string">documents</elementCategory>
        <showDataProperties
rdf:datatype="&xsd;boolean">true</showDataProperties>
        <isVisible rdf:datatype="&xsd;boolean">true</isVisible>
    </DomainElement>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#date -->

    <owl:Thing rdf:about="#date">
        <rdf:type rdf:resource="#ValueType"/>
        <valueTypeName rdf:datatype="&xsd;string">xsd:date</valueTypeName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#dateInputText -->

    <owl:Thing rdf:about="#dateInputText">
        <rdf:type rdf:resource="#Component"/>
        <rdfs:label>dateInputText</rdfs:label>
        <hasValueType rdf:resource="#date"/>
        <hasDescription rdf:resource="#dateInputTextDescription"/>
        <hasInteractionType rdf:resource="#free"/>
        <hasAttribute rdf:resource="#input_dateMask"/>
        <represents rdf:resource="#quantitative_biomedical_concept"/>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#dateInputTextDescription
-->

    <Description rdf:about="#dateInputTextDescription">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >dateInputTextDescription</rdfs:label>
        <idPrefix rdf:datatype="&xsd;string">DI</idPrefix>
        <componentName rdf:datatype="&xsd;string">inputText</componentName>
    </Description>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#document -->
```

```
    <owl:Thing rdf:about="#document">
        <rdf:type rdf:resource="#DomainElement"/>
        <rdfs:label>document</rdfs:label>
        <elementName rdf:datatype="&xsd;string">Document</elementName>
        <elementCategory
rdf:datatype="&xsd;string">documents</elementCategory>
        <showDataProperties
rdf:datatype="&xsd;boolean">false</showDataProperties>
        <isVisible rdf:datatype="&xsd;boolean">true</isVisible>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#document_body -->

    <DomainElement rdf:about="#document_body">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>document_body</rdfs:label>
        <elementName rdf:datatype="&xsd;string">Body</elementName>
        <elementCategory
rdf:datatype="&xsd;string">documents</elementCategory>
        <showDataProperties
rdf:datatype="&xsd;boolean">false</showDataProperties>
        <isVisible rdf:datatype="&xsd;boolean">true</isVisible>
    </DomainElement>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#document_header -->

    <owl:Thing rdf:about="#document_header">
        <rdf:type rdf:resource="#DomainElement"/>
        <rdfs:label>document_header</rdfs:label>
        <elementName rdf:datatype="&xsd;string">Header</elementName>
        <elementCategory
rdf:datatype="&xsd;string">documents</elementCategory>
        <showDataProperties
rdf:datatype="&xsd;boolean">false</showDataProperties>
        <isVisible rdf:datatype="&xsd;boolean">true</isVisible>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#errorIcon -->

    <Component rdf:about="#errorIcon">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>errorIcon</rdfs:label>
        <represents rdf:resource="#cdsElement"/>
        <hasDescription rdf:resource="#errorIconDescription"/>
        <hasAttribute rdf:resource="#errorIcon_ImagePath"/>
        <hasValueType rdf:resource="#image"/>
        <hasInteractionType rdf:resource="#output"/>
        <hasAttribute rdf:resource="#visibility_false"/>
    </Component>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#errorIconDescription -->

    <Description rdf:about="#errorIconDescription">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >errorIconDescription</rdfs:label>
```

```
        <idPrefix rdf:datatype="&xsd;string">EI</idPrefix>
        <componentName rdf:datatype="&xsd;string">icon</componentName>
    </Description>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#errorIcon_ImagePath -->

    <owl:Thing rdf:about="#errorIcon_ImagePath">
        <rdf:type rdf:resource="#Attribute"/>
        <rdfs:label
            >errorIcon_ImagePath</rdfs:label>
        <attributeValue rdf:datatype="&xsd;string"
            >http://opencti.larqss.ufpb/images/error.jpg</attributeValue>
        <attributeName rdf:datatype="&xsd;string">imagePath</attributeName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#float -->

    <owl:Thing rdf:about="#float">
        <rdf:type rdf:resource="#ValueType"/>
        <rdfs:label>float</rdfs:label>
        <valueTypeName rdf:datatype="&xsd;string">xsd:float</valueTypeName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#free -->

    <owl:Thing rdf:about="#free">
        <rdf:type rdf:resource="#InteractionType"/>
        <rdfs:label>free</rdfs:label>
        <interactionTypeName
rdf:datatype="&xsd;string">free</interactionTypeName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#getFocus -->

    <FocusEvent rdf:about="#getFocus">
        <rdf:type rdf:resource="#Event"/>
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>getFocus</rdfs:label>
        <eventName rdf:datatype="&xsd;string">getFocus</eventName>
    </FocusEvent>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#image -->

    <owl:Thing rdf:about="#image">
        <rdf:type rdf:resource="#ValueType"/>
        <rdfs:label>image</rdfs:label>
        <valueTypeName rdf:datatype="&xsd;string">image</valueTypeName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#inputListDescription -->

    <owl:Thing rdf:about="#inputListDescription">
        <rdf:type rdf:resource="#Description"/>
```

```
        <rdfs:label
            >inputListDescription</rdfs:label>
        <idPrefix rdf:datatype="&xsd;string">IL</idPrefix>
        <componentName rdf:datatype="&xsd;string">inputList</componentName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#inputText -->

    <owl:Thing rdf:about="#inputText">
        <rdf:type rdf:resource="#Component"/>
        <rdfs:label>inputText</rdfs:label>
        <hasValueType rdf:resource="#float"/>
        <hasInteractionType rdf:resource="#free"/>
        <hasDescription rdf:resource="#inputTextDescription"/>
        <hasValueType rdf:resource="#integer"/>
        <represents rdf:resource="#qualitative_biomedical_concept"/>
        <represents rdf:resource="#quantitative_biomedical_concept"/>
        <hasValueType rdf:resource="#string"/>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#inputTextDescription -->

    <owl:Thing rdf:about="#inputTextDescription">
        <rdf:type rdf:resource="#Description"/>
        <rdfs:label
            >inputTextDescription</rdfs:label>
        <idPrefix rdf:datatype="&xsd;string">IT</idPrefix>
        <componentName rdf:datatype="&xsd;string">inputText</componentName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#input_dateMask -->

    <Attribute rdf:about="#input_dateMask">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>input_dateMask</rdfs:label>
        <attributeValue
rdf:datatype="&xsd;string">99/99/9999</attributeValue>
        <attributeName rdf:datatype="&xsd;string">dateMask</attributeName>
    </Attribute>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#integer -->

    <owl:Thing rdf:about="#integer">
        <rdf:type rdf:resource="#ValueType"/>
        <rdfs:label>integer</rdfs:label>
        <valueTypeName
rdf:datatype="&xsd;string">xsd:integer</valueTypeName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#keyPressed -->

    <KeyEvent rdf:about="#keyPressed">
        <rdf:type rdf:resource="#Event"/>
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>keyPressed</rdfs:label>
```

```
        <eventName rdf:datatype="&xsd;string">keyPressed</eventName>
    </KeyEvent>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#keyReleased -->

    <Event rdf:about="#keyReleased">
        <rdf:type rdf:resource="#KeyEvent"/>
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>keyReleased</rdfs:label>
        <eventName rdf:datatype="&xsd;string">keyReleased</eventName>
    </Event>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#keyTyped -->

    <Event rdf:about="#keyTyped">
        <rdf:type rdf:resource="#KeyEvent"/>
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>keyTyped</rdfs:label>
        <eventName rdf:datatype="&xsd;string">keyTyped</eventName>
    </Event>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#list -->

    <owl:Thing rdf:about="#list">
        <rdf:type rdf:resource="#ValueType"/>
        <rdfs:label>list</rdfs:label>
        <valueTypeName rdf:datatype="&xsd;string">xsd:list</valueTypeName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#lostFocus -->

    <FocusEvent rdf:about="#lostFocus">
        <rdf:type rdf:resource="#Event"/>
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>lostFocus</rdfs:label>
        <eventName rdf:datatype="&xsd;string">lostFocus</eventName>
    </FocusEvent>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#many_of_a_list -->

    <InteractionType rdf:about="#many_of_a_list">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>many_of_a_list</rdfs:label>
        <interactionTypeName
rdf:datatype="&xsd;string">many_of_a_list</interactionTypeName>
    </InteractionType>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#mouseEntered -->

    <MouseEvent rdf:about="#mouseEntered">
        <rdf:type rdf:resource="#Event"/>
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>mouseEntered</rdfs:label>
```

```
        <eventName rdf:datatype="&xsd;string">mouseEntered</eventName>
    </MouseEvent>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#mouseExited -->

    <MouseEvent rdf:about="#mouseExited">
        <rdf:type rdf:resource="#Event"/>
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>mouseExited</rdfs:label>
        <eventName rdf:datatype="&xsd;string">mouseExited</eventName>
    </MouseEvent>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#one_of_a_list -->

    <InteractionType rdf:about="#one_of_a_list">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>one_of_a_list</rdfs:label>
        <interactionTypeName
rdf:datatype="&xsd;string">one_of_a_list</interactionTypeName>
    </InteractionType>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#output -->

    <InteractionType rdf:about="#output">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>output</rdfs:label>
        <interactionTypeName
rdf:datatype="&xsd;string">output</interactionTypeName>
    </InteractionType>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#outputText -->

    <Component rdf:about="#outputText">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>outputText</rdfs:label>
        <represents rdf:resource="#data_property"/>
        <hasValueType rdf:resource="#float"/>
        <hasValueType rdf:resource="#integer"/>
        <hasInteractionType rdf:resource="#output"/>
        <hasDescription rdf:resource="#outputTextDescription"/>
        <hasValueType rdf:resource="#string"/>
        <represents rdf:resource="#unit"/>
    </Component>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#outputTextDescription --
>

    <owl:Thing rdf:about="#outputTextDescription">
        <rdf:type rdf:resource="#Description"/>
        <rdfs:label
            >outputTextDescription</rdfs:label>
        <idPrefix rdf:datatype="&xsd;string">OT</idPrefix>
        <componentName rdf:datatype="&xsd;string">outputText</componentName>
    </owl:Thing>
```

```
    <!-- http://opencti.larqss.ufpb/UIComponent.owl#output_list -->

    <owl:Thing rdf:about="#output_list">
        <rdf:type rdf:resource="#InteractionType"/>
        <rdfs:label>output_list</rdfs:label>
        <interactionTypeName
rdf:datatype="&xsd;string">output_list</interactionTypeName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#panel -->

    <Container rdf:about="#panel">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>panel</rdfs:label>
        <represents rdf:resource="#abstract_biomedical_concept"/>
        <represents rdf:resource="#archetype"/>
        <represents rdf:resource="#document_header"/>
        <hasInteractionType rdf:resource="#output"/>
        <hasDescription rdf:resource="#panelDescription"/>
    </Container>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#panelDescription -->

    <owl:Thing rdf:about="#panelDescription">
        <rdf:type rdf:resource="#Description"/>
        <rdfs:label
            >panelDescription</rdfs:label>
        <idPrefix rdf:datatype="&xsd;string">PA</idPrefix>
        <componentName rdf:datatype="&xsd;string">panel</componentName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#panelGrid -->

    <owl:Thing rdf:about="#panelGrid">
        <rdf:type rdf:resource="#Container"/>
        <rdfs:label>panelGrid</rdfs:label>
        <represents rdf:resource="#concept_group"/>
        <hasInteractionType rdf:resource="#output"/>
        <hasDescription rdf:resource="#panelGridDescription"/>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#panelGridDescription -->

    <owl:Thing rdf:about="#panelGridDescription">
        <rdf:type rdf:resource="#Description"/>
        <rdfs:label
            >panelGridDescription</rdfs:label>
        <idPrefix rdf:datatype="&xsd;string">PG</idPrefix>
        <componentName rdf:datatype="&xsd;string">panelGrid</componentName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#panel_value_features -->
```

```
    <owl:Thing rdf:about="#panel_value_features">
        <rdfs:label
            >panel_value_features</rdfs:label>
    </owl:Thing>



    <!--
http://opencti.larqss.ufpb/UIComponent.owl#qualitative_biomedical_concept --
>

    <DomainElement rdf:about="#qualitative_biomedical_concept">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >qualitative_biomedical_concept</rdfs:label>
        <elementName rdf:datatype="&xsd;string"
            >QualitativeBiomedicalConcept</elementName>
        <elementCategory rdf:datatype="&xsd;string"
            >biomedicalConcepts</elementCategory>
        <isVisible rdf:datatype="&xsd;boolean">true</isVisible>
        <showDataProperties
rdf:datatype="&xsd;boolean">true</showDataProperties>
    </DomainElement>



    <!--
http://opencti.larqss.ufpb/UIComponent.owl#quantitative_biomedical_concept -
->

    <DomainElement rdf:about="#quantitative_biomedical_concept">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >quantitative_biomedical_concept</rdfs:label>
        <elementName rdf:datatype="&xsd;string"
            >QuantitativeBiomedicalConcept</elementName>
        <elementCategory rdf:datatype="&xsd;string"
            >biomedicalConcepts</elementCategory>
        <isVisible rdf:datatype="&xsd;boolean">true</isVisible>
        <showDataProperties
rdf:datatype="&xsd;boolean">true</showDataProperties>
    </DomainElement>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#section -->

    <owl:Thing rdf:about="#section">
        <rdf:type rdf:resource="#DomainElement"/>
        <rdfs:label>section</rdfs:label>
        <elementName rdf:datatype="&xsd;string">Section</elementName>
        <elementCategory
rdf:datatype="&xsd;string">documents</elementCategory>
        <showDataProperties
rdf:datatype="&xsd;boolean">false</showDataProperties>
        <isVisible rdf:datatype="&xsd;boolean">true</isVisible>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#setVisible_True -->

    <ActionEvent rdf:about="#setVisible_True">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>setVisible_True</rdfs:label>
```

```
        <eventName rdf:datatype="&xsd;string">setRendered</eventName>
        <hasParameter rdf:resource="#visibility_true"/>
    </ActionEvent>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#setVisible_false -->

    <owl:Thing rdf:about="#setVisible_false">
        <rdf:type rdf:resource="#ActionEvent"/>
        <rdfs:label
            >setVisible_false</rdfs:label>
        <eventName rdf:datatype="&xsd;string">setRendered</eventName>
        <hasParameter rdf:resource="#visibility_false"/>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#string -->

    <owl:Thing rdf:about="#string">
        <rdf:type rdf:resource="#ValueType"/>
        <rdfs:label>string</rdfs:label>
        <valueTypeName rdf:datatype="&xsd;string">xsd:string</valueTypeName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#suggestion -->

    <CdsEvent rdf:about="#suggestion">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>suggestion</rdfs:label>
        <elementName rdf:datatype="&xsd;string">Suggestion</elementName>
        <elementCategory rdf:datatype="&xsd;string">cds</elementCategory>
        <isVisible rdf:datatype="&xsd;boolean">false</isVisible>
        <showDataProperties
rdf:datatype="&xsd;boolean">true</showDataProperties>
    </CdsEvent>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#tabContainer -->

    <Container rdf:about="#tabContainer">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>tabContainer</rdfs:label>
        <represents rdf:resource="#document_body"/>
        <hasInteractionType rdf:resource="#output"/>
        <hasDescription rdf:resource="#tabContainerDescription"/>
    </Container>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#tabContainerDescription
-->

    <owl:Thing rdf:about="#tabContainerDescription">
        <rdf:type rdf:resource="#Description"/>
        <rdfs:label
            >tabContainerDescription</rdfs:label>
        <idPrefix rdf:datatype="&xsd;string">TC</idPrefix>
        <componentName
rdf:datatype="&xsd;string">tabContainer</componentName>
    </owl:Thing>
```

```
    <!-- http://opencti.larqss.ufpb/UIComponent.owl#tabPanel -->

    <owl:Thing rdf:about="#tabPanel">
        <rdf:type rdf:resource="#Container"/>
        <rdfs:label>tabPanel</rdfs:label>
        <hasInteractionType rdf:resource="#output"/>
        <represents rdf:resource="#section"/>
        <hasDescription rdf:resource="#tabPanelDescription"/>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#tabPanelDescription -->

    <owl:Thing rdf:about="#tabPanelDescription">
        <rdf:type rdf:resource="#Description"/>
        <rdfs:label
            >tabPanelDescription</rdfs:label>
        <idPrefix rdf:datatype="&xsd;string">TP</idPrefix>
        <componentName rdf:datatype="&xsd;string">tabPanel</componentName>
    </owl:Thing>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#unit -->

    <DomainElement rdf:about="#unit">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>unit</rdfs:label>
        <elementName rdf:datatype="&xsd;string">Unit</elementName>
        <elementCategory
rdf:datatype="&xsd;string">documents</elementCategory>
        <isVisible rdf:datatype="&xsd;boolean">true</isVisible>
        <showDataProperties
rdf:datatype="&xsd;boolean">true</showDataProperties>
    </DomainElement>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#visibility_false -->

    <Attribute rdf:about="#visibility_false">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >visibility_false</rdfs:label>
        <attributeValue rdf:datatype="&xsd;boolean">false</attributeValue>
        <attributeName rdf:datatype="&xsd;string">rendered</attributeName>
    </Attribute>



    <!-- http://opencti.larqss.ufpb/UIComponent.owl#visibility_true -->

    <owl:Thing rdf:about="#visibility_true">
        <rdf:type rdf:resource="#Attribute"/>
        <rdfs:label>visibility_true</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">rendered</attributeName>
        <attributeValue rdf:datatype="&xsd;boolean">true</attributeValue>
    </owl:Thing>
```

```
    <!--

///////////////////////////////////////////////////////////////////////////
///////////
    //
    // General axioms
    //

///////////////////////////////////////////////////////////////////////////
///////////
     -->

    <rdf:Description>
        <rdf:type rdf:resource="&owl;AllDisjointClasses"/>
        <owl:members rdf:parseType="Collection">
            <rdf:Description rdf:about="#Description"/>
            <rdf:Description rdf:about="#DomainElement"/>
            <rdf:Description rdf:about="#InteractionType"/>
            <rdf:Description rdf:about="#ValueFeatures"/>
            <rdf:Description rdf:about="#ValueType"/>
        </owl:members>
    </rdf:Description>
</rdf:RDF>



<!--    Generated    by    the    OWL    API    (version    2.2.1.1138)
http://owlapi.sourceforge.net -->
```

Ontologia de mapeamento

A seguir apresentaremos o código OWL resultante da implementação da ontologia utilizada pela ferramenta para o mapeamento entre componentes genéricos e componentes concretos.

```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY GUIMappings
"http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#" >
]>


<rdf:RDF xmlns="http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#"
     xml:base="http://larqss.di.ufpb.br/ontologias/GUIMappings.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

xmlns:GUIMappings="http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#">
```

```
    <owl:Ontology rdf:about=""/>



    <!--

///////////////////////////////////////////////////////////////////////////
///////////
    //
    // Object Properties
    //

///////////////////////////////////////////////////////////////////////////
///////////
     -->




    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#hasConcreteAttribute -->

    <owl:ObjectProperty rdf:about="#hasConcreteAttribute">
        <rdfs:label
            >hasConcreteAttribute</rdfs:label>
    </owl:ObjectProperty>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#hasConcreteComponent -->

    <owl:ObjectProperty rdf:about="#hasConcreteComponent">
        <rdfs:label
            >hasConcreteComponent</rdfs:label>
    </owl:ObjectProperty>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#hasConcreteEvent -->

    <owl:ObjectProperty rdf:about="#hasConcreteEvent">
        <rdfs:label
            >hasConcreteEvent</rdfs:label>
    </owl:ObjectProperty>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#hasGenericAttribute -->

    <owl:ObjectProperty rdf:about="#hasGenericAttribute">
        <rdfs:label
            >hasGenericAttribute</rdfs:label>
    </owl:ObjectProperty>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#hasGenericComponent -->

    <owl:ObjectProperty rdf:about="#hasGenericComponent">
        <rdfs:label
            >hasGenericComponent</rdfs:label>
```

```
    </owl:ObjectProperty>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#hasGenericEvent
-->

    <owl:ObjectProperty rdf:about="#hasGenericEvent">
        <rdfs:label>hasGenericEvent</rdfs:label>
    </owl:ObjectProperty>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#hasParameter --
>

    <owl:ObjectProperty rdf:about="#hasParameter">
        <rdfs:label>hasParameter</rdfs:label>
    </owl:ObjectProperty>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#hasTechnologyDescription
-->

    <owl:ObjectProperty rdf:about="#hasTechnologyDescription">
        <rdfs:label
            >hasTechnologyDescription</rdfs:label>
    </owl:ObjectProperty>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#isAssociatedTo
-->

    <owl:ObjectProperty rdf:about="#isAssociatedTo">
        <rdfs:label>isAssociatedTo</rdfs:label>
    </owl:ObjectProperty>



    <!--

///////////////////////////////////////////////////////////////////////////
///////////
    //
    // Data properties
    //

///////////////////////////////////////////////////////////////////////////
///////////
     -->




    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#attributeName -
->

    <owl:DatatypeProperty rdf:about="#attributeName">
        <rdfs:label>attributeName</rdfs:label>
    </owl:DatatypeProperty>
```

```
    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#attributeValue
-->

    <owl:DatatypeProperty rdf:about="#attributeValue">
        <rdfs:label>attributeValue</rdfs:label>
    </owl:DatatypeProperty>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#componentName -
->

    <owl:DatatypeProperty rdf:about="#componentName">
        <rdfs:label>componentName</rdfs:label>
    </owl:DatatypeProperty>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#componentPath -
->

    <owl:DatatypeProperty rdf:about="#componentPath">
        <rdfs:label>componentPath</rdfs:label>
    </owl:DatatypeProperty>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#deviceDescription -->

    <owl:DatatypeProperty rdf:about="#deviceDescription">
        <rdfs:label
            >deviceDescription</rdfs:label>
    </owl:DatatypeProperty>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#eventName -->

    <owl:DatatypeProperty rdf:about="#eventName">
        <rdfs:label>eventName</rdfs:label>
    </owl:DatatypeProperty>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#parameterType -
->

    <owl:DatatypeProperty rdf:about="#parameterType">
        <rdfs:label>parameterType</rdfs:label>
    </owl:DatatypeProperty>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#technologyBuilderName --
>

    <owl:DatatypeProperty rdf:about="#technologyBuilderName">
        <rdfs:label
            >technologyBuilderName</rdfs:label>
    </owl:DatatypeProperty>
```

```
    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#technologyName
-->

    <owl:DatatypeProperty rdf:about="#technologyName">
        <rdfs:label>technologyName</rdfs:label>
    </owl:DatatypeProperty>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#technologyVersion -->

    <owl:DatatypeProperty rdf:about="#technologyVersion">
        <rdfs:label
            >technologyVersion</rdfs:label>
    </owl:DatatypeProperty>



    <!--

///////////////////////////////////////////////////////////////////////////
///////////
    //
    // Classes
    //

///////////////////////////////////////////////////////////////////////////
///////////
     -->




    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#Attribute -->

    <owl:Class rdf:about="#Attribute">
        <rdfs:label>Attribute</rdfs:label>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#attributeValue"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#attributeName"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#AttributeMapping -->

    <owl:Class rdf:about="#AttributeMapping">
        <rdfs:label
            >AttributeMapping</rdfs:label>
```

```
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasGenericAttribute"/>
                <owl:onClass rdf:resource="#GenericAttribute"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasConcreteAttribute"/>
                <owl:someValuesFrom rdf:resource="#ConcreteAttribute"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isAssociatedTo"/>
                <owl:onClass rdf:resource="#GenericComponent"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#Component -->

    <owl:Class rdf:about="#Component">
        <rdfs:label>Component</rdfs:label>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#componentName"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#ComponentMapping -->

    <owl:Class rdf:about="#ComponentMapping">
        <rdfs:label
            >ComponentMapping</rdfs:label>
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasConcreteComponent"/>
                <owl:someValuesFrom rdf:resource="#ConcreteComponent"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasGenericComponent"/>
                <owl:onClass rdf:resource="#GenericComponent"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>
```

```
    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#ConcreteAttribute -->

    <owl:Class rdf:about="#ConcreteAttribute">
        <rdfs:label
            >ConcreteAttribute</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Attribute"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isAssociatedTo"/>
                <owl:onClass rdf:resource="#ConcreteComponent"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasTechnologyDescription"/>
                <owl:onClass rdf:resource="#TechnologyDescription"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#ConcreteComponent -->

    <owl:Class rdf:about="#ConcreteComponent">
        <rdfs:label
            >ConcreteComponent</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Component"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#componentPath"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasTechnologyDescription"/>
                <owl:onClass rdf:resource="#TechnologyDescription"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#ConcreteEvent -
->

    <owl:Class rdf:about="#ConcreteEvent">
        <rdfs:label>ConcreteEvent</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Event"/>
        <rdfs:subClassOf>
            <owl:Restriction>
```

```
                <owl:onProperty rdf:resource="#hasTechnologyDescription"/>
                <owl:onClass rdf:resource="#TechnologyDescription"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isAssociatedTo"/>
                <owl:onClass rdf:resource="#ConcreteComponent"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#Event -->

    <owl:Class rdf:about="#Event">
        <rdfs:label>Event</rdfs:label>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#eventName"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#EventMapping --
>

    <owl:Class rdf:about="#EventMapping">
        <rdfs:label>EventMapping</rdfs:label>
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isAssociatedTo"/>
                <owl:onClass rdf:resource="#GenericComponent"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasConcreteEvent"/>
                <owl:someValuesFrom rdf:resource="#ConcreteEvent"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasGenericEvent"/>
                <owl:onClass rdf:resource="#GenericEvent"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>
```

```
    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#GenericAttribute -->

    <owl:Class rdf:about="#GenericAttribute">
        <rdfs:label
            >GenericAttribute</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Attribute"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isAssociatedTo"/>
                <owl:onClass rdf:resource="#GenericComponent"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#GenericComponent -->

    <owl:Class rdf:about="#GenericComponent">
        <rdfs:label
            >GenericComponent</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Component"/>
    </owl:Class>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#GenericEvent --
>

    <owl:Class rdf:about="#GenericEvent">
        <rdfs:label>GenericEvent</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Event"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isAssociatedTo"/>
                <owl:onClass rdf:resource="#GenericComponent"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#TechnologyDescription --
>

    <owl:Class rdf:about="#TechnologyDescription">
        <rdfs:label
            >TechnologyDescription</rdfs:label>
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#technologyBuilderName"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
```

```
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#technologyName"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#deviceDescription"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#technologyVersion"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>



    <!-- http://www.w3.org/2002/07/owl#Thing -->

    <owl:Class rdf:about="&owl;Thing"/>



    <!--

////////////////////////////////////////////////////////////////////////
///////////
    //
    // Individuals
    //

////////////////////////////////////////////////////////////////////////
///////////
     -->




    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#checkBox -->

    <owl:Thing rdf:about="#checkBox">
        <rdf:type rdf:resource="#GenericComponent"/>
        <rdfs:label>checkBox</rdfs:label>
        <componentName rdf:datatype="&xsd;string">checkBox</componentName>
        <componentPath rdf:datatype="&xsd;string"
            >ja-
vax.faces.component.html.HtmlSelectBooleanCheckbox</componentPath>
    </owl:Thing>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#checkBoxMapping
-->
```

```
    <owl:Thing rdf:about="#checkBoxMapping">
        <rdf:type rdf:resource="#ComponentMapping"/>
        <rdfs:label>checkBoxMapping</rdfs:label>
        <hasGenericComponent rdf:resource="#checkBox"/>
        <hasConcreteComponent rdf:resource="#jsf_checkBox"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#checkBox_Value_Attribute
Mapping -->

    <AttributeMapping rdf:about="#checkBox_Value_AttributeMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >checkBox_Value_AttributeMapping</rdfs:label>
        <isAssociatedTo rdf:resource="#checkBox"/>
        <hasGenericAttribute
rdf:resource="#checkBox_Value_GenericAttribute"/>
        <hasConcreteAttribute
rdf:resource="#jsf_checkBox_Value_ConcreteAttribute"/>
    </AttributeMapping>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#checkBox_Value_GenericAt
tribute -->

    <owl:Thing rdf:about="#checkBox_Value_GenericAttribute">
        <rdf:type rdf:resource="#GenericAttribute"/>
        <rdfs:label
            >checkBox_Value_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">value</attributeName>
        <isAssociatedTo rdf:resource="#checkBox"/>
    </owl:Thing>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#comboBox -->

    <GenericComponent rdf:about="#comboBox">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>comboBox</rdfs:label>
        <componentName rdf:datatype="&xsd;string">comboBox</componentName>
    </GenericComponent>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#comboBoxMapping
-->

    <ComponentMapping rdf:about="#comboBoxMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>comboBoxMapping</rdfs:label>
        <hasGenericComponent rdf:resource="#comboBox"/>
        <hasConcreteComponent rdf:resource="#jsf_comboBox"/>
    </ComponentMapping>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#comboBox_Elements_Attrib
uteMapping -->
```

```
    <AttributeMapping rdf:about="#comboBox_Elements_AttributeMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >comboBox_Elements_AttributeMapping</rdfs:label>
        <isAssociatedTo rdf:resource="#comboBox"/>
        <hasGenericAttribute
rdf:resource="#comboBox_Elements_GenericAttribute"/>
        <hasConcreteAttribute
rdf:resource="#jsf_comboBox_SuggestionValues_ConcreteAttribute"/>
    </AttributeMapping>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#comboBox_Elements_Generi
cAttribute -->

    <GenericAttribute rdf:about="#comboBox_Elements_GenericAttribute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >comboBox_Elements_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">elements</attributeName>
        <isAssociatedTo rdf:resource="#comboBox"/>
    </GenericAttribute>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#icon -->

    <GenericComponent rdf:about="#icon">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>icon</rdfs:label>
        <componentName rdf:datatype="&xsd;string">icon</componentName>
    </GenericComponent>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#iconMapping -->

    <ComponentMapping rdf:about="#iconMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>iconMapping</rdfs:label>
        <hasGenericComponent rdf:resource="#icon"/>
        <hasConcreteComponent rdf:resource="#jsf_icon"/>
    </ComponentMapping>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText -->

    <GenericComponent rdf:about="#inputText">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>inputText</rdfs:label>
        <componentName rdf:datatype="&xsd;string">inputText</componentName>
    </GenericComponent>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputTextMapping -->

    <ComponentMapping rdf:about="#inputTextMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
```

```
            >inputTextMapping</rdfs:label>
        <hasGenericComponent rdf:resource="#inputText"/>
        <hasConcreteComponent rdf:resource="#jsf_inputText"/>
    </ComponentMapping>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_DateMask_Attri
buteMapping -->

    <owl:Thing rdf:about="#inputText_DateMask_AttributeMapping">
        <rdf:type rdf:resource="#AttributeMapping"/>
        <rdfs:label
            >inputText_DateMask_AttributeMapping</rdfs:label>
        <isAssociatedTo rdf:resource="#inputText"/>
        <hasGenericAttribute
rdf:resource="#inputText_dateMask_GenericAttribute"/>
        <hasConcreteAttribute
rdf:resource="#jsf_inputText_DateMask_ConcreteAttribute"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_Disabled_Attri
buteMapping -->

    <AttributeMapping rdf:about="#inputText_Disabled_AttributeMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >inputText_Disabled_AttributeMapping</rdfs:label>
        <isAssociatedTo rdf:resource="#inputText"/>
        <hasGenericAttribute
rdf:resource="#inputText_Disabled_GenericAttribute"/>
        <hasConcreteAttribute
rdf:resource="#jsf_inputText_Disabled_ConcreteAttribute"/>
    </AttributeMapping>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_Disabled_Gener
icAttribute -->

    <owl:Thing rdf:about="#inputText_Disabled_GenericAttribute">
        <rdf:type rdf:resource="#GenericAttribute"/>
        <rdfs:label
            >inputText_Disabled_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">disabled</attributeName>
        <isAssociatedTo rdf:resource="#inputText"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_Label_Attribut
eMapping -->

    <owl:Thing rdf:about="#inputText_Label_AttributeMapping">
        <rdf:type rdf:resource="#AttributeMapping"/>
        <rdfs:label
            >inputText_Label_AttributeMapping</rdfs:label>
        <isAssociatedTo rdf:resource="#inputText"/>
        <hasGenericAttribute
```

```
rdf:resource="#inputText_Label_GenericAttribute"/>
        <hasConcreteAttribute
rdf:resource="#jsf_inputText_Label_ConcreteAttribute"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_Label_GenericA
ttribute -->

    <GenericAttribute rdf:about="#inputText_Label_GenericAttribute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >inputText_Label_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">label</attributeName>
        <isAssociatedTo rdf:resource="#inputText"/>
    </GenericAttribute>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_Value_Attribut
eMapping -->

    <owl:Thing rdf:about="#inputText_Value_AttributeMapping">
        <rdf:type rdf:resource="#AttributeMapping"/>
        <rdfs:label
            >inputText_Value_AttributeMapping</rdfs:label>
        <isAssociatedTo rdf:resource="#inputText"/>
        <hasGenericAttribute
rdf:resource="#inputText_Value_GenericAttribute"/>
        <hasConcreteAttribute
rdf:resource="#jsf_inputText_Value_ConcreteAttribute"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_Value_GenericA
ttribute -->

    <owl:Thing rdf:about="#inputText_Value_GenericAttribute">
        <rdf:type rdf:resource="#GenericAttribute"/>
        <rdfs:label
            >inputText_Value_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">value</attributeName>
        <isAssociatedTo rdf:resource="#inputText"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_Visible_Attrib
uteMapping -->

    <AttributeMapping rdf:about="#inputText_Visible_AttributeMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >inputText_Visible_AttributeMapping</rdfs:label>
        <isAssociatedTo rdf:resource="#inputText"/>
        <hasGenericAttribute
rdf:resource="#inputText_Visible_GenericAttribute"/>
        <hasConcreteAttribute
rdf:resource="#jsf_inputText_Visible_ConcreteAttribute"/>
```

```
    </AttributeMapping>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_Visible_Generi
cAttribute -->

    <owl:Thing rdf:about="#inputText_Visible_GenericAttribute">
        <rdf:type rdf:resource="#GenericAttribute"/>
        <rdfs:label
            >inputText_Visible_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">visible</attributeName>
        <isAssociatedTo rdf:resource="#inputText"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_dateMask_Gener
icAttribute -->

    <owl:Thing rdf:about="#inputText_dateMask_GenericAttribute">
        <rdf:type rdf:resource="#GenericAttribute"/>
        <rdfs:label
            >inputText_dateMask_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">dateMask</attributeName>
        <isAssociatedTo rdf:resource="#inputText"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_getFocus -->

    <owl:Thing rdf:about="#inputText_getFocus">
        <rdf:type rdf:resource="#GenericEvent"/>
        <rdfs:label
            >inputText_getFocus</rdfs:label>
        <eventName rdf:datatype="&xsd;string">getFocus</eventName>
        <isAssociatedTo rdf:resource="#inputText"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_getFocus_Event
Mapping -->

    <EventMapping rdf:about="#inputText_getFocus_EventMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >inputText_getFocus_EventMapping</rdfs:label>
        <isAssociatedTo rdf:resource="#inputText"/>
        <hasGenericEvent rdf:resource="#inputText_getFocus"/>
        <hasConcreteEvent rdf:resource="#jsf_InputText_onFocus"/>
    </EventMapping>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_keyPressed -->

    <owl:Thing rdf:about="#inputText_keyPressed">
        <rdf:type rdf:resource="#GenericEvent"/>
```

```
        <rdfs:label
            >inputText_keyPressed</rdfs:label>
        <eventName rdf:datatype="&xsd;string">keyPressed</eventName>
        <isAssociatedTo rdf:resource="#inputText"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_keyPressed_Eve
ntMapping -->

    <owl:Thing rdf:about="#inputText_keyPressed_EventMapping">
        <rdf:type rdf:resource="#EventMapping"/>
        <rdfs:label
            >inputText_keyPressed_EventMapping</rdfs:label>
        <isAssociatedTo rdf:resource="#inputText"/>
        <hasConcreteEvent rdf:resource="#jsf_InputText_keyPressed"/>
        <hasGenericEvent rdf:resource="#jsf_InputText_keyPressed"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_keyReleased --
>

    <GenericEvent rdf:about="#inputText_keyReleased">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >inputText_keyReleased</rdfs:label>
        <eventName rdf:datatype="&xsd;string">keyReleased</eventName>
        <isAssociatedTo rdf:resource="#inputText"/>
    </GenericEvent>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_keyReleased_Ev
entMapping -->

    <EventMapping rdf:about="#inputText_keyReleased_EventMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >inputText_keyReleased_EventMapping</rdfs:label>
        <isAssociatedTo rdf:resource="#inputText"/>
        <hasGenericEvent rdf:resource="#inputText_keyReleased"/>
        <hasConcreteEvent rdf:resource="#jsf_InputText_keyReleased"/>
    </EventMapping>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_lostFocus -->

    <GenericEvent rdf:about="#inputText_lostFocus">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >inputText_lostFocus</rdfs:label>
        <eventName rdf:datatype="&xsd;string">lostFocus</eventName>
        <isAssociatedTo rdf:resource="#inputText"/>
    </GenericEvent>
```

```
    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#inputText_lostFocus_Even
tMapping -->

    <owl:Thing rdf:about="#inputText_lostFocus_EventMapping">
        <rdf:type rdf:resource="#EventMapping"/>
        <rdfs:label
            >inputText_lostFocus_EventMapping</rdfs:label>
        <isAssociatedTo rdf:resource="#inputText"/>
        <hasGenericEvent rdf:resource="#inputText_lostFocus"/>
        <hasConcreteEvent rdf:resource="#jsf_InputText_onBlur"/>
    </owl:Thing>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf -->

    <owl:Thing rdf:about="#jsf">
        <rdf:type rdf:resource="#TechnologyDescription"/>
        <rdfs:label>jsf</rdfs:label>
        <technologyVersion
rdf:datatype="&xsd;string">1.0</technologyVersion>
        <technologyName rdf:datatype="&xsd;string"
            >JAVA SERVER FACES</technologyName>
        <deviceDescription rdf:datatype="&xsd;string">PC</deviceDescription>
        <technologyBuilderName
rdf:datatype="&xsd;string">jsfBuilder</technologyBuilderName>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_InputText_keyPressed
-->

    <owl:Thing rdf:about="#jsf_InputText_keyPressed">
        <rdf:type rdf:resource="#ConcreteEvent"/>
        <rdfs:label
            >jsf_InputText_keyPressed</rdfs:label>
        <eventName rdf:datatype="&xsd;string">keyPressed</eventName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_inputText"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_InputText_keyRelease
d -->

    <owl:Thing rdf:about="#jsf_InputText_keyReleased">
        <rdf:type rdf:resource="#ConcreteEvent"/>
        <rdfs:label
            >jsf_InputText_keyReleased</rdfs:label>
        <eventName rdf:datatype="&xsd;string">keyReleased</eventName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_inputText"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_InputText_onBlur -->

    <owl:Thing rdf:about="#jsf_InputText_onBlur">
```

```
        <rdf:type rdf:resource="#ConcreteEvent"/>
        <rdfs:label
            >jsf_InputText_lostFocus</rdfs:label>
        <eventName rdf:datatype="&xsd;string">onBlur</eventName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_inputText"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_InputText_onFocus --
>

    <ConcreteEvent rdf:about="#jsf_InputText_onFocus">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >jsf_InputText_onFocus</rdfs:label>
        <eventName rdf:datatype="&xsd;string">onFocus</eventName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_inputText"/>
    </ConcreteEvent>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_checkBox --
>

    <owl:Thing rdf:about="#jsf_checkBox">
        <rdf:type rdf:resource="#ConcreteComponent"/>
        <rdfs:label>jsf_checkBox</rdfs:label>
        <componentName rdf:datatype="&xsd;string">h:checkBox</componentName>
        <componentPath rdf:datatype="&xsd;string"
            >ja-
vax.faces.component.html.HtmlSelectBooleanCheckbox</componentPath>
        <hasTechnologyDescription rdf:resource="#jsf"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_checkBox_Value_Concr
eteAttribute -->

    <owl:Thing rdf:about="#jsf_checkBox_Value_ConcreteAttribute">
        <rdf:type rdf:resource="#ConcreteAttribute"/>
        <rdfs:label
            >jsf_checkBox_Value_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">value</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_checkBox"/>
    </owl:Thing>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_comboBox --
>

    <ConcreteComponent rdf:about="#jsf_comboBox">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>jsf_comboBox</rdfs:label>
        <componentName rdf:datatype="&xsd;string">h:comboBox</componentName>
        <componentPath rdf:datatype="&xsd;string"
            >javax.faces.component.html.HtmlSelectOneMenu</componentPath>
        <hasTechnologyDescription rdf:resource="#jsf"/>
```

```
    </ConcreteComponent>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_comboBox_SuggestionV
alues_ConcreteAttribute -->

    <owl:Thing rdf:about="#jsf_comboBox_SuggestionValues_ConcreteAttribute">
        <rdf:type rdf:resource="#ConcreteAttribute"/>
        <rdfs:label
            >jsf_comboBox_SuggestionValues_ConcreteAttribute</rdfs:label>
        <attributeName
rdf:datatype="&xsd;string">possibleValues</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_comboBox"/>
    </owl:Thing>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_icon -->

    <ConcreteComponent rdf:about="#jsf_icon">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>jsf_icon</rdfs:label>
        <componentName
rdf:datatype="&xsd;string">HtmlPanelGroup</componentName>
        <componentPath rdf:datatype="&xsd;string"
            >javax.faces.component.html.HtmlPanelGroup</componentPath>
        <hasTechnologyDescription rdf:resource="#jsf"/>
    </ConcreteComponent>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_inputText -
->

    <owl:Thing rdf:about="#jsf_inputText">
        <rdf:type rdf:resource="#ConcreteComponent"/>
        <rdfs:label>jsf_inputText</rdfs:label>
        <componentName
rdf:datatype="&xsd;string">h:inputText</componentName>
        <componentPath rdf:datatype="&xsd;string"
            >javax.faces.component.html.HtmlInputText</componentPath>
        <hasTechnologyDescription rdf:resource="#jsf"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_inputText_DateMask_C
oncreteAttribute -->

    <owl:Thing rdf:about="#jsf_inputText_DateMask_ConcreteAttribute">
        <rdf:type rdf:resource="#ConcreteAttribute"/>
        <rdfs:label
            >jsf_inputText_DateMask_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">dateMask</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_inputText"/>
    </owl:Thing>



    <!--
```

```
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_inputText_Disabled_C
oncreteAttribute -->

    <owl:Thing rdf:about="#jsf_inputText_Disabled_ConcreteAttribute">
        <rdf:type rdf:resource="#ConcreteAttribute"/>
        <rdfs:label
            >jsf_inputText_Disabled_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">disabled</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_inputText"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_inputText_Label_Conc
reteAttribute -->

    <owl:Thing rdf:about="#jsf_inputText_Label_ConcreteAttribute">
        <rdf:type rdf:resource="#ConcreteAttribute"/>
        <rdfs:label
            >jsf_inputText_Label_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">label</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_inputText"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_inputText_Value_Conc
reteAttribute -->

    <owl:Thing rdf:about="#jsf_inputText_Value_ConcreteAttribute">
        <rdf:type rdf:resource="#ConcreteAttribute"/>
        <rdfs:label
            >jsf_inputText_Value_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">value</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_inputText"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_inputText_Visible_Co
ncreteAttribute -->

    <owl:Thing rdf:about="#jsf_inputText_Visible_ConcreteAttribute">
        <rdf:type rdf:resource="#ConcreteAttribute"/>
        <rdfs:label
            >jsf_inputText_Visible_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">visible</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_inputText"/>
    </owl:Thing>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_outputLabel
-->

    <owl:Thing rdf:about="#jsf_outputLabel">
        <rdf:type rdf:resource="#ConcreteComponent"/>
        <rdfs:label>jsf_outputLabel</rdfs:label>
```

```
        <componentName
rdf:datatype="&xsd;string">h:outputLabel</componentName>
        <componentPath rdf:datatype="&xsd;string"
            >javax.faces.component.html.HtmlOutputText</componentPath>
        <hasTechnologyDescription rdf:resource="#jsf"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_outputText_Label_Con
creteAttribute -->

    <ConcreteAttribute rdf:about="#jsf_outputText_Label_ConcreteAttribute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >jsf_outputText_Label_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">label</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_outputLabel"/>
    </ConcreteAttribute>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_panel -->

    <ConcreteComponent rdf:about="#jsf_panel">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>jsf_panel</rdfs:label>
        <componentName rdf:datatype="&xsd;string">HtmlPanel</componentName>
        <componentPath rdf:datatype="&xsd;string"
            >org.primefaces.component.panel.Panel</componentPath>
        <hasTechnologyDescription rdf:resource="#jsf"/>
    </ConcreteComponent>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_panelGrid -
->

    <owl:Thing rdf:about="#jsf_panelGrid">
        <rdf:type rdf:resource="#ConcreteComponent"/>
        <rdfs:label>jsf_panelGrid</rdfs:label>
        <componentPath rdf:datatype="&xsd;string"
            >javax.faces.component.html.HtmlPanelGrid</componentPath>
        <componentName rdf:datatype="&xsd;string">panelGrid</componentName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_panelGrid_Width_Conc
reteAttribute -->

    <owl:Thing rdf:about="#jsf_panelGrid_Width_ConcreteAttribute">
        <rdf:type rdf:resource="#ConcreteAttribute"/>
        <rdfs:label
            >jsf_panelGrid_Width_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">width</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_panelGrid"/>
    </owl:Thing>
```

```
    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_panelGrid_columns_Co
ncreteAttribute -->

    <ConcreteAttribute rdf:about="#jsf_panelGrid_columns_ConcreteAttribute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >jsf_panelGrid_columns_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">columns</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_panelGrid"/>
    </ConcreteAttribute>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_panel_Label_Concrete
Attribute -->

    <ConcreteAttribute rdf:about="#jsf_panel_Label_ConcreteAttribute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >jsf_panel_Label_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">label</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_panel"/>
    </ConcreteAttribute>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_panel_Visible_Concre
teAttribute -->

    <ConcreteAttribute rdf:about="#jsf_panel_Visible_ConcreteAttribute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >jsf_panel_Visible_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">visible</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_panel"/>
    </ConcreteAttribute>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_tab -->

    <owl:Thing rdf:about="#jsf_tab">
        <rdf:type rdf:resource="#ConcreteComponent"/>
        <rdfs:label>jsf_tab</rdfs:label>
        <componentPath rdf:datatype="&xsd;string"
            >org.primefaces.component.tabview.Tab</componentPath>
        <componentName rdf:datatype="&xsd;string">tab</componentName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
    </owl:Thing>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_tabView -->

    <ConcreteComponent rdf:about="#jsf_tabView">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>jsf_tabView</rdfs:label>
        <componentPath rdf:datatype="&xsd;string"
```

```
            >org.primefaces.component.tabview.TabView</componentPath>
        <componentName rdf:datatype="&xsd;string">tabView</componentName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
    </ConcreteComponent>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_tabView_Label_Concre
teAttribute -->

    <ConcreteAttribute rdf:about="#jsf_tabView_Label_ConcreteAttribute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >jsf_tabView_Label_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">label</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_tabView"/>
    </ConcreteAttribute>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_tabView_Visible_Conc
reteAttribute -->

    <ConcreteAttribute rdf:about="#jsf_tabView_Visible_ConcreteAttribute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >jsf_tabView_Visible_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">visible</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_tabView"/>
    </ConcreteAttribute>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_tab_Label_ConcreteAt
tribute -->

    <ConcreteAttribute rdf:about="#jsf_tab_Label_ConcreteAttribute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >jsf_tab_Label_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">label</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_tab"/>
    </ConcreteAttribute>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#jsf_tab_Visible_Concrete
Attribute -->

    <owl:Thing rdf:about="#jsf_tab_Visible_ConcreteAttribute">
        <rdf:type rdf:resource="#ConcreteAttribute"/>
        <rdfs:label
            >jsf_tab_Visible_ConcreteAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">visible</attributeName>
        <hasTechnologyDescription rdf:resource="#jsf"/>
        <isAssociatedTo rdf:resource="#jsf_tab"/>
    </owl:Thing>
```

```
    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#outputText -->

    <owl:Thing rdf:about="#outputText">
        <rdf:type rdf:resource="#GenericComponent"/>
        <rdfs:label>outputText</rdfs:label>
        <componentName rdf:datatype="&xsd;string">outputText</componentName>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#outputTextLabel_GenericA
ttribute -->

    <GenericAttribute rdf:about="#outputTextLabel_GenericAttribute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >outputTextLabel_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">label</attributeName>
        <isAssociatedTo rdf:resource="#outputText"/>
    </GenericAttribute>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#outputTextMapping -->

    <owl:Thing rdf:about="#outputTextMapping">
        <rdf:type rdf:resource="#ComponentMapping"/>
        <rdfs:label
            >outputTextMapping</rdfs:label>
        <hasConcreteComponent rdf:resource="#jsf_outputLabel"/>
        <hasGenericComponent rdf:resource="#outputText"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#outputText_Label_Attribu
teMapping -->

    <AttributeMapping rdf:about="#outputText_Label_AttributeMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >outputText_Label_AttributeMapping</rdfs:label>
        <hasConcreteAttribute
rdf:resource="#jsf_outputText_Label_ConcreteAttribute"/>
        <isAssociatedTo rdf:resource="#outputText"/>
        <hasGenericAttribute
rdf:resource="#outputTextLabel_GenericAttribute"/>
    </AttributeMapping>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#panel -->

    <owl:Thing rdf:about="#panel">
        <rdf:type rdf:resource="#GenericComponent"/>
        <rdfs:label>panel</rdfs:label>
        <componentName rdf:datatype="&xsd;string">panel</componentName>
    </owl:Thing>
```

```
    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#panelGrid -->

    <GenericComponent rdf:about="#panelGrid">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>panelGrid</rdfs:label>
        <componentName rdf:datatype="&xsd;string">panelGrid</componentName>
    </GenericComponent>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#panelGridMapping -->

    <ComponentMapping rdf:about="#panelGridMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >panelGridMapping</rdfs:label>
        <hasConcreteComponent rdf:resource="#jsf_panelGrid"/>
        <hasGenericComponent rdf:resource="#panelGrid"/>
    </ComponentMapping>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#panelGrid_Width_Attribut
eMapping -->

    <AttributeMapping rdf:about="#panelGrid_Width_AttributeMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >panelGrid_Width_AttributeMapping</rdfs:label>
        <hasConcreteAttribute
rdf:resource="#jsf_panelGrid_Width_ConcreteAttribute"/>
        <isAssociatedTo rdf:resource="#panelGrid"/>
        <hasGenericAttribute
rdf:resource="#panelGrid_Width_GenericAttribute"/>
    </AttributeMapping>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#panelGrid_Width_GenericA
ttribute -->

    <GenericAttribute rdf:about="#panelGrid_Width_GenericAttribute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >panelGrid_Width_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">width</attributeName>
        <isAssociatedTo rdf:resource="#panelGrid"/>
    </GenericAttribute>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#panelGrid_columns_Attrib
uteMapping -->

    <AttributeMapping rdf:about="#panelGrid_columns_AttributeMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >panelGrid_columns_AttributeMapping</rdfs:label>
        <hasConcreteAttribute
rdf:resource="#jsf_panelGrid_columns_ConcreteAttribute"/>
```

```
        <isAssociatedTo rdf:resource="#panelGrid"/>
        <hasGenericAttribute
rdf:resource="#panelGrid_columns_GenericAttribute"/>
    </AttributeMapping>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#panelGrid_columns_Generi
cAttribute -->

    <owl:Thing rdf:about="#panelGrid_columns_GenericAttribute">
        <rdf:type rdf:resource="#GenericAttribute"/>
        <rdfs:label
            >panelGrid_columns_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">columns</attributeName>
        <isAssociatedTo rdf:resource="#panelGrid"/>
    </owl:Thing>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#panelMapping --
>

    <ComponentMapping rdf:about="#panelMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>panelMapping</rdfs:label>
        <hasConcreteComponent rdf:resource="#jsf_panel"/>
        <hasGenericComponent rdf:resource="#panel"/>
    </ComponentMapping>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#panel_Label_AttributeMap
ping -->

    <AttributeMapping rdf:about="#panel_Label_AttributeMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >panel_Label_AttributeMapping</rdfs:label>
        <hasConcreteAttribute
rdf:resource="#jsf_panel_Label_ConcreteAttribute"/>
        <isAssociatedTo rdf:resource="#panel"/>
        <hasGenericAttribute rdf:resource="#panel_Label_GenericAttribute"/>
    </AttributeMapping>


    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#panel_Label_GenericAttri
bute -->

    <owl:Thing rdf:about="#panel_Label_GenericAttribute">
        <rdf:type rdf:resource="#GenericAttribute"/>
        <rdfs:label
            >panel_Label_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">label</attributeName>
        <isAssociatedTo rdf:resource="#panel"/>
    </owl:Thing>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#panel_Visible_AttributeM
apping -->
```

```
    <owl:Thing rdf:about="#panel_Visible_AttributeMapping">
        <rdf:type rdf:resource="#AttributeMapping"/>
        <rdfs:label
            >panel_Visible_AttributeMapping</rdfs:label>
        <hasConcreteAttribute
rdf:resource="#jsf_panel_Visible_ConcreteAttribute"/>
        <isAssociatedTo rdf:resource="#panel"/>
        <hasGenericAttribute rdf:resource="#panel_Visible_GenericAttibute"/>
    </owl:Thing>


    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#panel_Visible_GenericAtt
ibute -->

    <GenericAttribute rdf:about="#panel_Visible_GenericAttibute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >panel_Visible_GenericAttibute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">visible</attributeName>
        <isAssociatedTo rdf:resource="#panel"/>
    </GenericAttribute>


    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#tabContainer --
>

    <GenericComponent rdf:about="#tabContainer">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label>tabContainer</rdfs:label>
        <componentName
rdf:datatype="&xsd;string">tabContainer</componentName>
    </GenericComponent>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#tabContainerMapping -->

    <ComponentMapping rdf:about="#tabContainerMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >tabContainerMapping</rdfs:label>
        <hasConcreteComponent rdf:resource="#jsf_tabView"/>
        <hasGenericComponent rdf:resource="#tabContainer"/>
    </ComponentMapping>



    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#tabContainer_Label_Attri
buteMapping -->

    <AttributeMapping rdf:about="#tabContainer_Label_AttributeMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >tabContainer_Label_AttributeMapping</rdfs:label>
        <hasConcreteAttribute
rdf:resource="#jsf_tabView_Label_ConcreteAttribute"/>
        <isAssociatedTo rdf:resource="#tabContainer"/>
        <hasGenericAttribute
rdf:resource="#tabContainer_Label_GenericAttribute"/>
    </AttributeMapping>
```

```
    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#tabContainer_Label_Gener
icAttribute -->

    <owl:Thing rdf:about="#tabContainer_Label_GenericAttribute">
        <rdf:type rdf:resource="#GenericAttribute"/>
        <rdfs:label
            >tabContainer_Label_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">label</attributeName>
        <isAssociatedTo rdf:resource="#tabContainer"/>
    </owl:Thing>


    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#tabContainer_Visible_Att
ributeMapping -->

    <AttributeMapping rdf:about="#tabContainer_Visible_AttributeMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >tabContainer_Visible_AttributeMapping</rdfs:label>
        <hasConcreteAttribute
rdf:resource="#jsf_tabView_Visible_ConcreteAttribute"/>
        <isAssociatedTo rdf:resource="#tabContainer"/>
        <hasGenericAttribute
rdf:resource="#tabContainer_Visible_GenericAttribute"/>
    </AttributeMapping>
    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#tabContainer_Visible_Gen
ericAttribute -->

    <owl:Thing rdf:about="#tabContainer_Visible_GenericAttribute">
        <rdf:type rdf:resource="#GenericAttribute"/>
        <rdfs:label
            >tabContainer_Visible_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">visible</attributeName>
        <isAssociatedTo rdf:resource="#tabContainer"/>
    </owl:Thing>


    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#tabPanel -->

    <owl:Thing rdf:about="#tabPanel">
        <rdf:type rdf:resource="#GenericComponent"/>
        <rdfs:label>tabPanel</rdfs:label>
        <componentName rdf:datatype="&xsd;string">tabPanel</componentName>
    </owl:Thing>



    <!-- http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#tabPanelMapping
-->

    <owl:Thing rdf:about="#tabPanelMapping">
        <rdf:type rdf:resource="#ComponentMapping"/>
        <rdfs:label>tabPanelMapping</rdfs:label>
        <hasConcreteComponent rdf:resource="#jsf_tab"/>
        <hasGenericComponent rdf:resource="#tabPanel"/>
    </owl:Thing>



    <!--
```

```
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#tabPanel_Label_Attribute
Mapping -->

    <owl:Thing rdf:about="#tabPanel_Label_AttributeMapping">
        <rdf:type rdf:resource="#AttributeMapping"/>
        <rdfs:label
            >tabPanel_Label_AttributeMapping</rdfs:label>
        <hasConcreteAttribute
rdf:resource="#jsf_tab_Label_ConcreteAttribute"/>
        <isAssociatedTo rdf:resource="#tabPanel"/>
        <hasGenericAttribute
rdf:resource="#tabPanel_Label_GenericAttribute"/>
    </owl:Thing>


    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#tabPanel_Label_GenericAt
tribute -->

    <GenericAttribute rdf:about="#tabPanel_Label_GenericAttribute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >tabPanel_Label_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">label</attributeName>
        <isAssociatedTo rdf:resource="#tabPanel"/>
    </GenericAttribute>

    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#tabPanel_Visible_Attribu
teMapping -->

    <AttributeMapping rdf:about="#tabPanel_Visible_AttributeMapping">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >tabPanel_Visible_AttributeMapping</rdfs:label>
        <hasConcreteAttribute
rdf:resource="#jsf_tab_Visible_ConcreteAttribute"/>

6.4          <isAssociatedTo rdf:resource="#tabPanel"/>

        <hasGenericAttribute
rdf:resource="#tabPanel_Visible_GenericAttribute"/>
    </AttributeMapping>

    <!--
http://larqss.di.ufpb.br/ontologias/GUIMappings.owl#tabPanel_Visible_Generic
Attribute -->

    <GenericAttribute rdf:about="#tabPanel_Visible_GenericAttribute">
        <rdf:type rdf:resource="&owl;Thing"/>
        <rdfs:label
            >tabPanel_Visible_GenericAttribute</rdfs:label>
        <attributeName rdf:datatype="&xsd;string">visible</attributeName>
        <isAssociatedTo rdf:resource="#tabPanel"/>
    </GenericAttribute>
</rdf:RDF>


<!-- Generated by the OWL API (version 2.2.1.1138)
http://owlapi.sourceforge.net -->
```

Apêndice B

Interfaces do MedViewGen

IGUIManager

A seguir apresentaremos o código-fonte da interface *IGUIManager*.

```
1  /**
2  * Retorna o formulario do documento passado como parametro.
3  */
4  public synchronized Object getDocumentForm(String documentType,
5                HealthDocument healthDocument, String operationType)
6        throws DocumentCreateException, DocumentNotFoundException,
7                UIGenerationException, OWLOntologyCreationException,
8                OntologyConsistencyException, NoClassIndividualException,
9                InstantiationException, IllegalAccessException;
10
11 /**
12 * Metodo que executa a gravacao do documento atual no banco de dados.
13 */
14 public synchronized String saveDoc(javax.faces.event.ActionEvent evt)
15       throws IllegalArgumentException, IllegalAccessException,
16                InvocationTargetException, InvalidClassException,
17                DataPropertyDefaultDataException, BackEndComponentNot-
18 Found, EventCreateException, DynamicGUIException;
19
20 /**
21 * Limpa a árvore de filhos de componentes jsf do formulário do
22 * documento e os campos do Back-End JavaBean de um healthdocument
23 */
24 public String closeForm(javax.faces.event.ActionEvent evt);
```

```
25
26  /**
27  * Metodo que executa a gravacao do documento atual como um rascunho no
28  * diretorio do usuário.
29  */
30  public synchronized String saveSketch(javax.faces.event.ActionEvent
31  evt) throws InvalidClassException, IllegalArgumentException,
32         BackEndComponentNotFound, DynamicGUIException,
33         DataPropertyDefaultDataException, IllegalAccessException,
34         nvocationTargetException;
35
36  /**
37  * Limpa os valores dos campos do Back-end JavaBean de um determinado
38  * tipo de healthdocument
39  */
40  public void clearDocumentBean() throws IllegalArgumentException,
41         IllegalAccessException;
42
43  /**
44  * Lanca um evento capturavel pelo modulo de CDS.
45  */
46  public void throwCDSEvent(String eventName, String fieldPath)
47         throws InvalidClassException, IllegalArgumentException,
48         BackEndComponentNotFound, DataPropertyDefaultDataException,
49         IllegalAccessException, InvocationTargetException,
50         UICDSException, ThingNotFoundException, UIGenerationException,
51         DynamicGUIException;
52
53  /**
54  * Metodo que executa um evento de GUI.
55  */
56  public void executeGUIEvent(String cdsEventName, String iri,
57         List<Object> message, Map<String, String> properties,
58         IAppContext appContext) throws UICDSException,
59         ThingNotFoundException, UIMappingsException,
60         UIConvertionException;
```

IGUIBuilder

A seguir apresentaremos o código-fonte da interface *IGUIBuilder*.

```
1  /**
2  * Método que gera um formulário para determinado documento.
3  Cada
```

```
4  * implementacao dessa interface irá gerar formulários para uma
5  * tecnologia diferente.
6  * @param document - Documento cuja interface sera gerada.
7  * @param documentType - Tipo do documento.
8  * @param operationType - Indica se deve ser gerada uma GUI
9  para entrada
10 * ou saída de dados.
11 * @param technologyDescription - Descritor da tecnologia de
12 GUI para
13 * a qual o formulário deve ser gerado.
14 */
15 public Object getDocumentForm(HealthDocument document, String
16 documentType, String operationType) throws UIGenerationExcep-
17 tion,
18       OWLOntologyCreationException, OntologyConsistencyExcep-
19 tion,
20       NoClassIndividualException, InstantiationException,
21       IllegalAccessException;
22
23 /**
24 * Método que retorna o gerenciados dos caminhos dos campos do
25 * documento associados aos componentes de interface.
26 */
27 public DocumentFieldPathManager getDocumentFieldPathManager();
28
29 /**
30 * Retorna a estrutura de componentes genericos.
31 */
32 public GenericUIStructure getGenericUIStructure ();
33
34 /**
35 * Retorna o HealthDocument que está sendo utilizado no momento.
36 */
37 public HealthDocument getHealthDocument();
38
39 /**
40 * Retorna o nome do bean gerado para a interação do formulario
41 * de documento com a aplicacao.
42 */
43 public String getDocumentBeanName();
44
45 /**
46 * Retorna a descricao tecnologica do Builder.
```

```
47  */
48  public TechnologyDescription getTechnologyDescription();
49
50  /**
51  * Retorna um determinado componente instanciado no form de documento.
52
53  */
54  public Object getComponent (String id, Object actualDocumentForm, UIVi-
55  ewRoot viewRoot);
56
57  /**
58  * Reseta o estado do componente ao seu estado inicial.
59  */
60  public void resetComponentState(Object uiComponent)
61        throws UIGenerationException, IllegalArgumentException,
62        IllegalAccessException, InvocationTargetException;
63
64  /**
    * Altera os valores dos atributos passados como parametro.
    */
    public void parseAttributes (Object fatherComponent, Object childCompo-
    nent, ArrayList<ConcreteUIAttribute> concreteUIAttributes)
          throws UICDSException;
```

IGenericGUIBuilder

A seguir apresentaremos o código-fonte da interface *IGenericGUIBuilder*.

```
1  /**
2  * Retorna a estrutura generica de componentes de interface para um
3  * determinado documento de entrada.
4  * @param document - Documento para o qual a interface sera gerada.
5  */
6  public synchronized GenericUIStructure generateUIStructure (HealthDocu-
7  ment document)
8         throws UIOntologyException, UIComponentNotFoundException,
9         UIComponentConsistencyException, UIGenericGenerationException,
10        InstantiationException, IllegalAccessException;
```

IGUIConverter

A seguir apresentaremos o código-fonte da interface *IGUIConverter*.

```
1  /**
2  * Metodo que executa o mapeamento dos componentes genericos para
```

```
3  * componentes especificos da tecnologia do visualizador do usuario.
4  */
5  public ArrayList<ConcreteUIComponent> executeMapping (ArrayL-
6  ist<GenericUIComponent> genericList, TechnologyDescription technology-
7  Description) throws UIMappingsException, UIConvertionException;
8
9  /**
10 * Metodo que converte uma lista de atributos genéricos em atributos
11 * concretos.
12 */
13 public ArrayList<ConcreteUIAttribute> convertAttributes (ComponentDe-
14 scription componentDescription, ArrayList<GenericUIAttribute> generi-
15 cAttributes, TechnologyDescription technologyDescription)
16       throws UIMappingsException, UIConvertionException;
```

Apêndice C

Frames do Certificado de Óbito

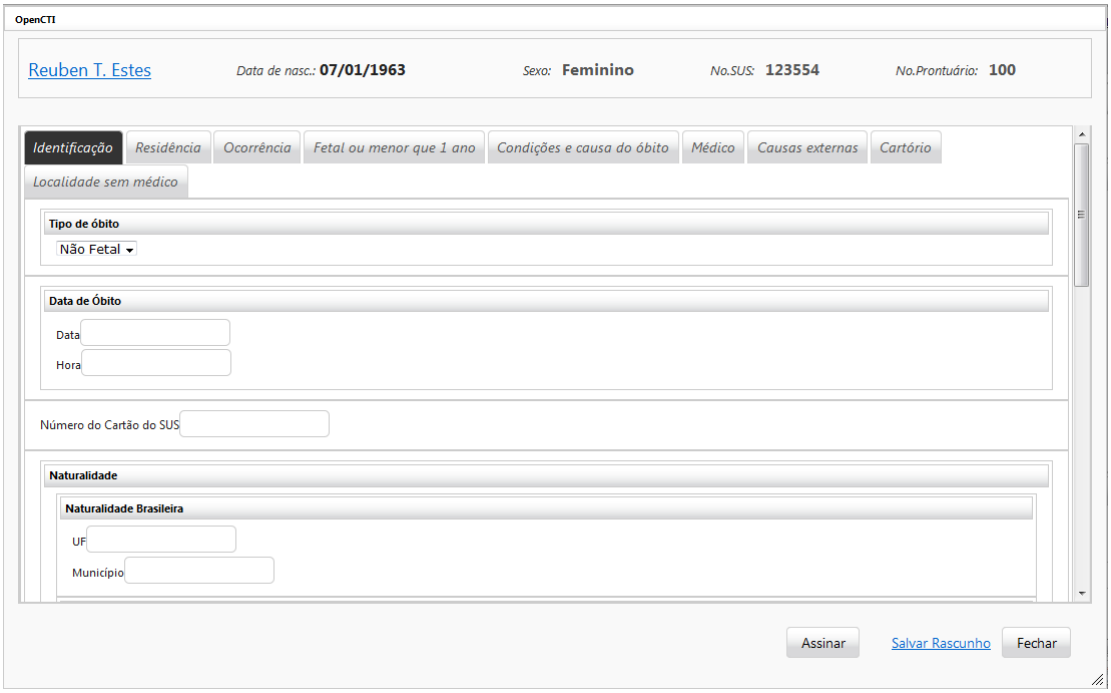

Figura 44: Certificado de óbito: Aba Identificação

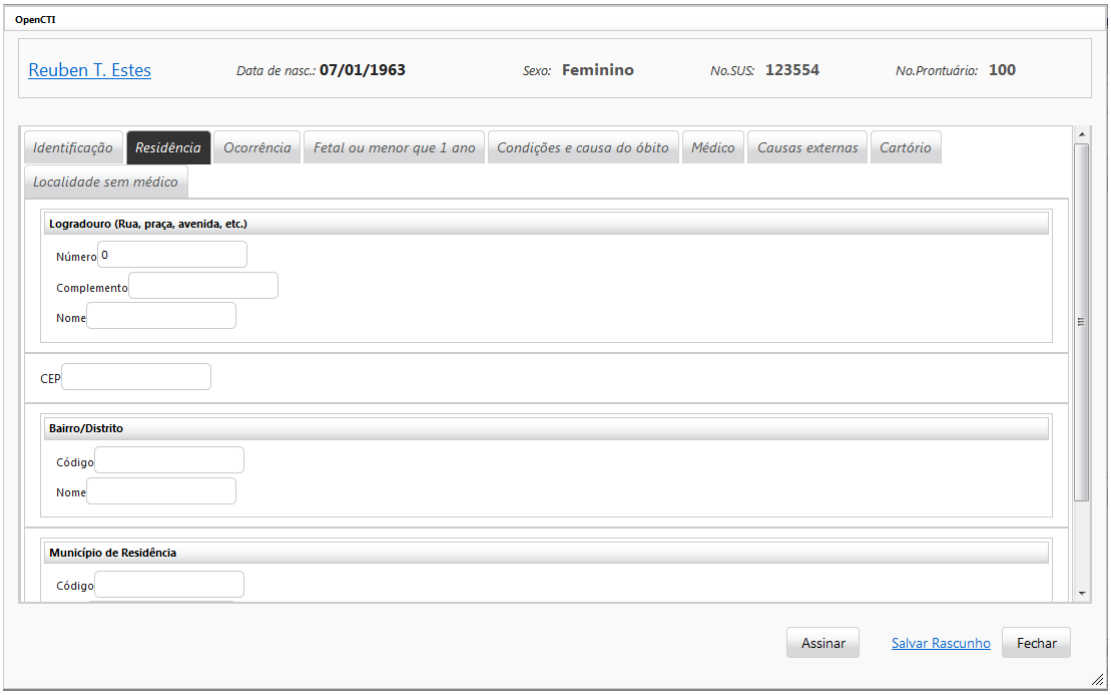

Figura 45: Certificado de óbito: Aba Residência

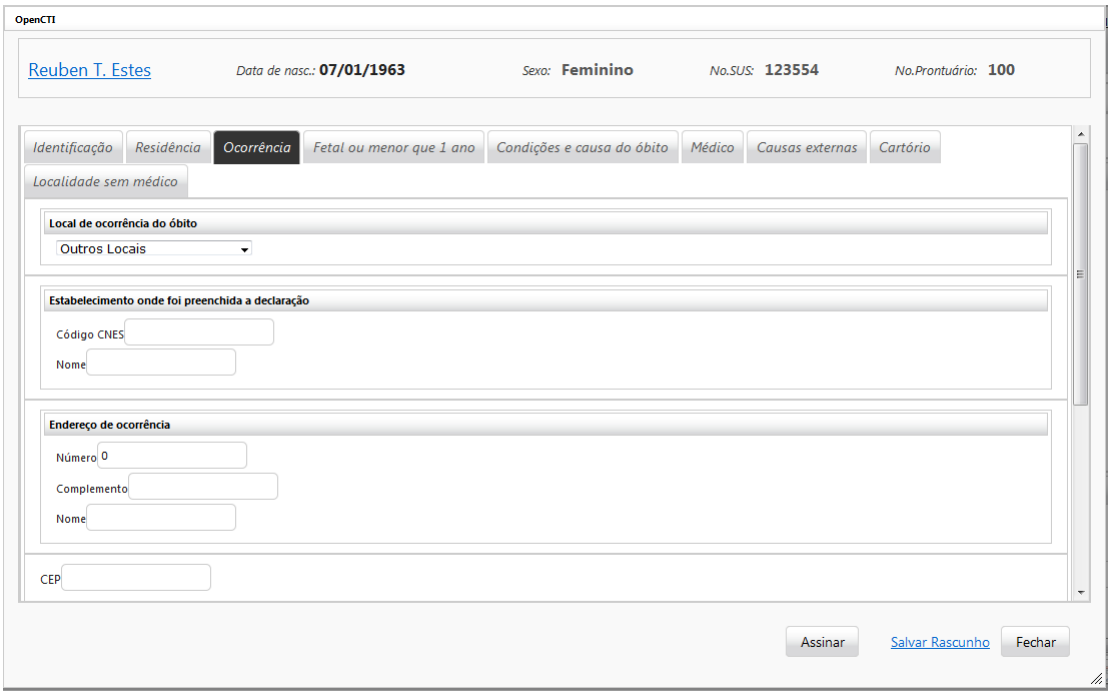

Figura 46: Certificado de óbito: Aba Ocorrência

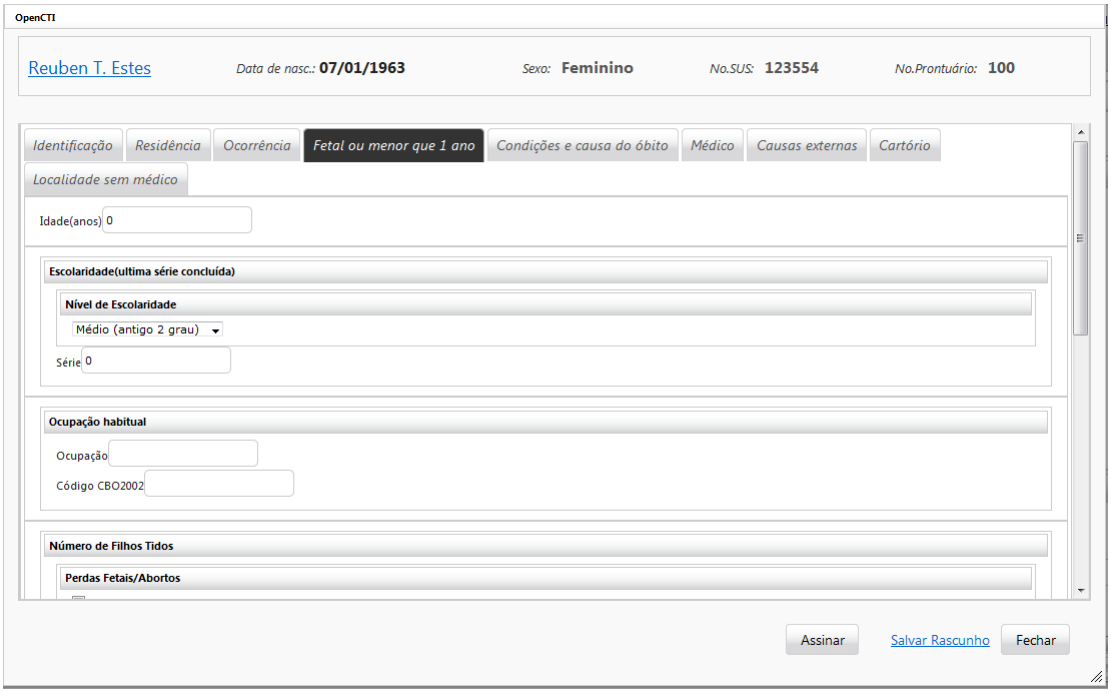

Figura 47: Certificado de óbito: Aba Fetal ou menos que 1 ano

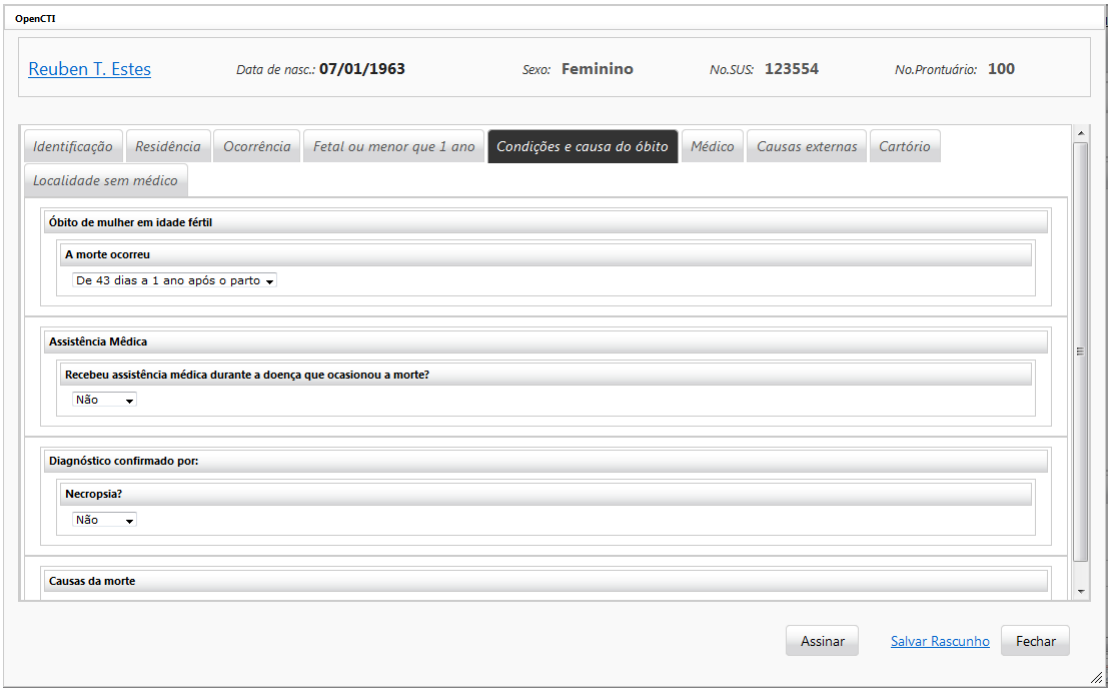

Figura 48: Certificado de óbito: Aba Condições e causa de óbito

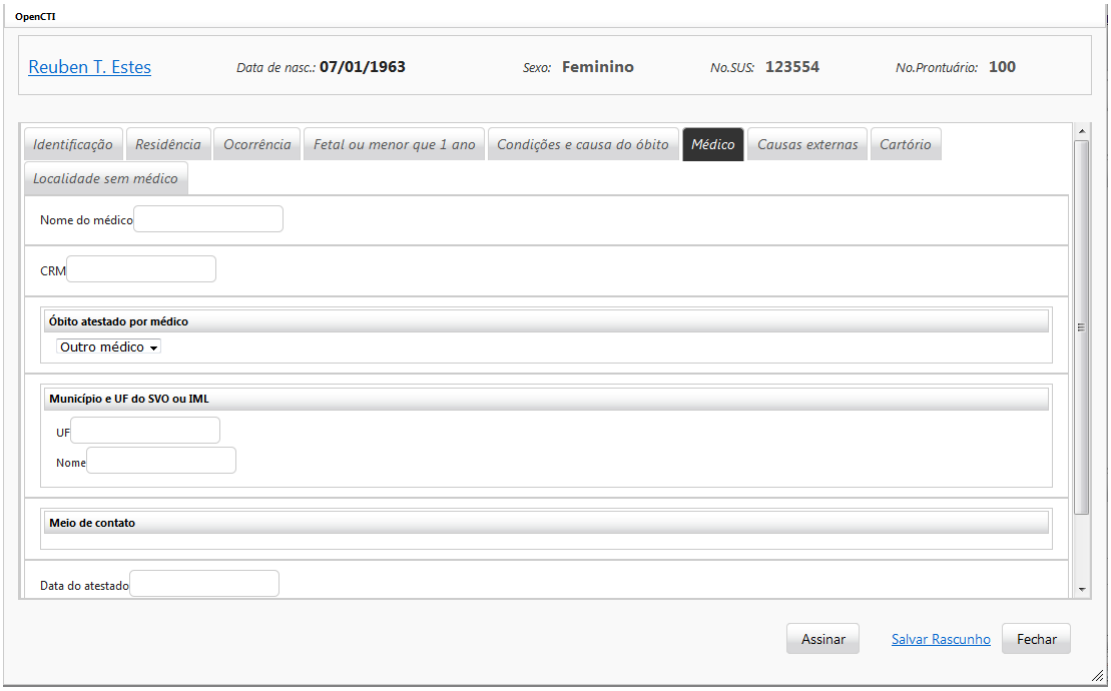

Figura 49: Certificado de óbito: Aba Médico

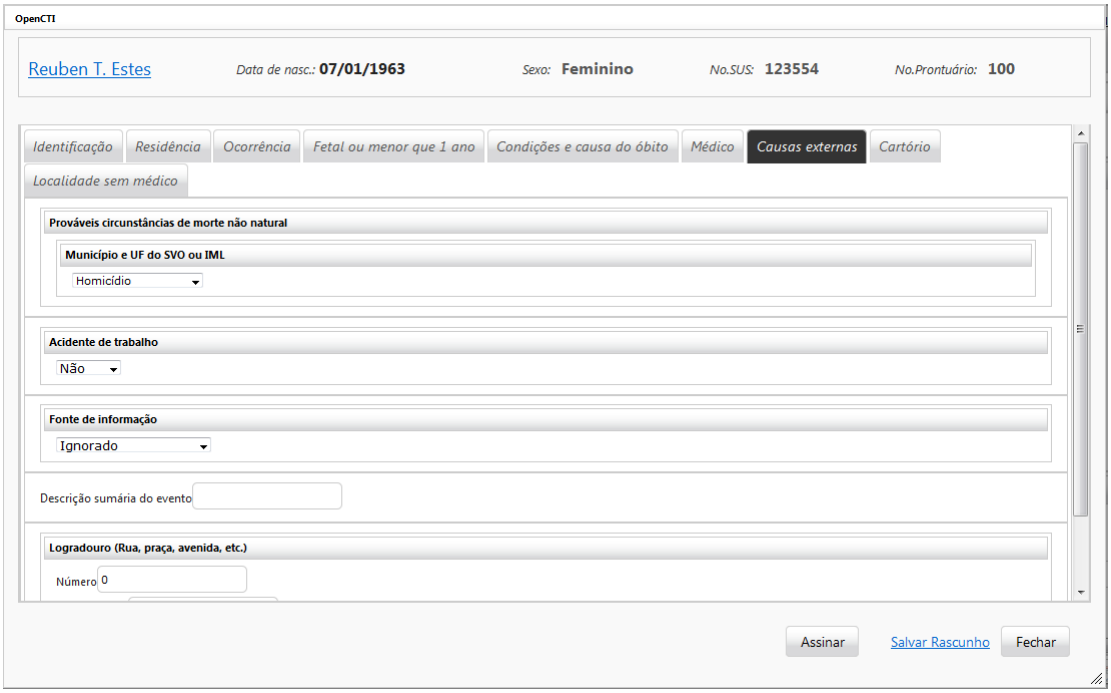

Figura 50: Certificado de óbito: Aba Causas externas

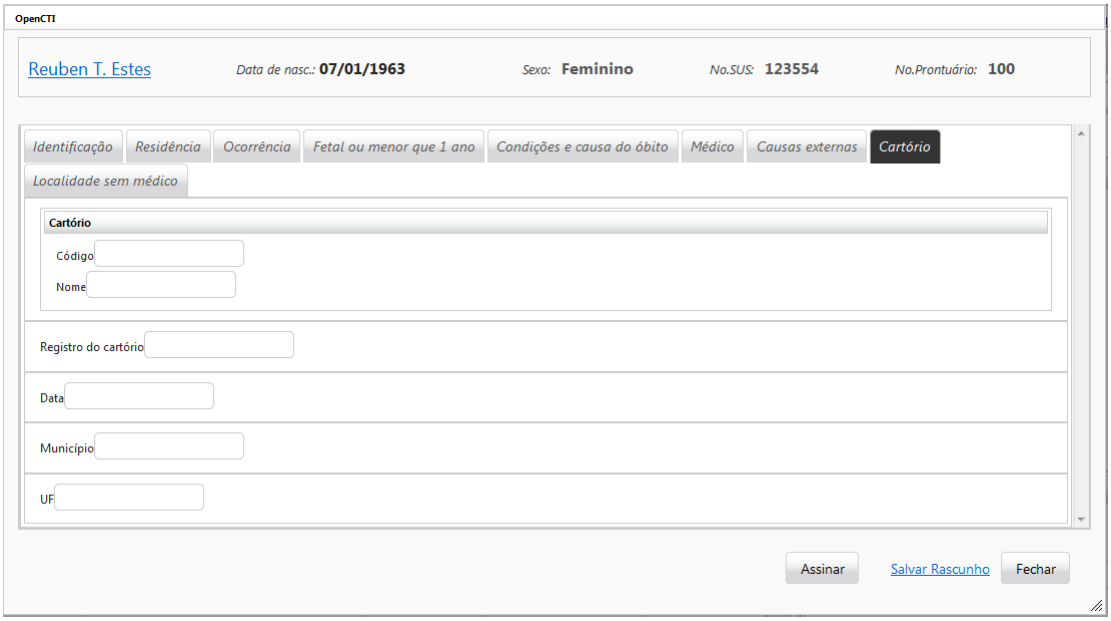

Figura 51: Certificado de óbito: Aba Cartório

Referências

Allemang D., Hendler J. (2008). “Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL”. Morgan Kaufmann.

ANS. Available from: http://www.ans.gov.br

Atalag K. Archetype Based Domain Modeling For Health Information Systems. A Thesis Submitted to the Graduate School of Informatics of the Middle East Technical University. July 2007.

Avellar e Duarte. Layout da interface. Available from: http://www.avellareduarte.com.br/projeto/desenvolvimento/desenvolvimento2/desenvolvimento2.htm

Barros, R.S.M., Ferreira, S.M.G., HEXSEL, R.A. (2004), “Desenvolvimento de Solução Única de Software para o Sistema Cartão Nacional de Saúde”, Anais do VIII Congresso Brasileiro de Informática em Saúde, Natal, 29/09 - 02/10.

Beale T. Archetypes Constraint-based Domain Models for Future-proof Information Systems. Available from: www.deepthought.com.au.

CEN/TC251 Health Informatics. Available from: http://www.centc251.org/WGI/WGI-scope.htm.

Conselho Federal de Medicina (Brasil). Resolução CFM no 1.638/2002. Brasília, DF: Conselho Federal de Medicina, 10 jul. 2002. Disponível em: <http://www.portalmedico.org.br/resolucoes/cfm/2002/1638_2002.htm>. Acessado em: 26 jul. 2010.

Dean M., Schreiber G. 2004. OWL Web Ontology Language Reference. [Online] Available at: http://www.w3.org/TR/owl-ref/ [Accessed 20 March 2010].

Dinu V, Nadkarni P. Guidelines for the effective use of entity–attribute–value modeling for biomedical databases. International Journal of Medical Informatics. 2007; (76): 769–779.

Dogac A, Namli T, Okcan A, et al. Key Issues of Technical Interoperability Solutions in eHealth. eHealth. 2007: 1-10.

Duarte, R.C.M. Framework Baseado na Modelagem EAV para a persistência de dados clínicos em registros eletrônicos em saúde. Monografia de conclusão de curso. UFPB, 2010.

Garde S, Knaup P, Hovenga EJS, Heard S. (2007). Towards Semantic Interoperability for Electronic Health Records: Domain Knowledge Governance for openEHR Archetypes. Methods of Information in Medicine 46(3): 332–343.

Gosling J., Joy B., Steele G. and Bracha G. The Java™ Language Specification Third Edition ISBN 0-321-24678-0. Disponível em: http://java.sun.com/docs/books/jls/download/langspec-3.0.pdf. Acessado em: 27 Jul. 2010.

Greenes R. Definition, Scope and Challenges. In:_____(Ed.). Clinical Decision Support: The Road Ahead. Academic Press, 2006. cap. 1. p. 3-29.

Health Level 7 Inc. (last accessed 18 January 2010). Available from: http://www.hl7.org.

Health Informatics—Electronic health Record communication. Part 1. Reference Model, CEN/TC251 prEN13606-1:2005:E, March 2005.

Health Informatics—Electronic health Record communication. Part 2. Archetypes, CEN/TC251 prEN13606-2:2005:E, December 2005.

Hoekman R. Designing the Obvious: A Common Sense Approach to Web Application Design. New Riders, 2007.

iText PDF [cited 2010 Mar 19]. Available from http://itextpdf.com/

Javassist. (2010) Java Programming Assistant API [cited 2010 Mar 20]. Available from http://www.csg.is.titech.ac.jp/~chiba/javassist/html/

Jiang He, I-Ling Yen. (2007). Adaptive User Interface Generation for Web Services. e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference : 536 – 539.

Karine Petry e Paula Marien Albecht Lopes. (2005). Modelos para interoperabilidade de sistemas hospitalares utilizando o padrão HL7. Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação, UFSC, 2005. Disponível em: http://projetos.inf.ufsc.br/arquivos_projetos/projeto_377/Interoperabilidade%20de%20Sistemas%20Hospitalares%20Utilizando%20o%20Padr%E3o%20HL7.pdf

Kavaldjian S, Raneburger D, Falb J, Kaindl H, Ertl D. Semi-automatic user interface generation considering pointing granularity. Proceedings of the 2009 IEEE international conference on Systems, Man and Cybernetics. 2009. P. 2052-2058.

Larman C. Utilizando UML e Padrões. Bookman, 3ª Edição. 2007.

Luna A, Schwabe D. Ontology Driven Dynamic Web Interface Generation. 8th International Workshop on Web Oriented Software Technology (2009).

Marin, H. F.; Massad, E.; Azevedo-Neto, R. S. Prontuário eletrônico do paciente: definições e conceitos. In: _____ (Ed.). O prontuário eletrônico do paciente na assistência, informação e conhecimento médico. Washington, DC: Organização Pan-Americana de Saúde, 2003. 213 p.

Miller R, Sim I. Physician's use of electronic medical records: barriers and solutions. Health Affairs. 2004; 23(2): 116-126.

OLIVEIRA NETTO, Alvim Antônio de. IHC – Interação Humano Computador – Modelagem e Gerência de Interfaces com o Usuário. Florianópolis: VisualBooks, 2004.

Petry, M; Marien, P; Wangenheim, A. Padrões para Interoperabilidade na Saúde. Universidade Federal de Santa Catarina, Brasil. 2006.

Pizzol DAS, Serafim EP, Duarte RCM, Figueiredo JFM; Motta, GHMB. An ontology-based EHR: representing data, documents, user interface and decision support in healthcare. Proceedings of IADIS International Conference e-Health 2010: 1-4. (aceito para publicação)

Pizzol DAS, Serafim EP, Duarte RCM, Figueiredo JFM; Motta, GHMB. Arquitetura Flexível Baseada em Ontologias para o Registro Eletrônico de Saúde. Relatório técnico do projeto OpenCTI.

Pizzol DAS, Serafim EP, Luciano, Clauirton, Figueiredo JFM, Silva Sérgio LD, Motta GHMB. Um Modelo Baseado em Ontologia para Agentes Multipropósitos de Apoio à Decisão Clínica integrado ao Registro Eletrônico em Saúde. Artigo aceito para publicação no 12º Congresso Brasileiro de Informática em Saúde – CBIS'2010.

Panda D, Rahman R, Lane D. EJB3 em ação. Alta Books, 2ª Edição. Rio de Janeiro, 2009.

PRC. Disponível em: http://www.datasus.gov.br/prc. Acessado em: 27 Jul. 2010.

Protegé web site. Disponível em: http://protege.stanford.edu/. Acessado em: 27 Jul. 2010.

Putman, J. Architecting with RM-ODP. Prentice Hall, 2001. 834 p.

Rosenbloom ST, Miller RA, Johnson KB, Elkin PL, Brown SH. Interface terminologies: facilitating direct entry of clinical data into electronic health record systems. J Am Med Inform Assoc. 2006; (13): 277-288.

Roukema J, Renske K, Bleeker E, van Ginneken A, Van der Lei J, Moll H. Paper Versus Computer: Feasibility of an Electronic Medical Record in General Pediatrics. Pediatrics 2006; (117):15-21.

Schuler T., Garde S., Heard S., Beale T. 2006. Towards automatically generating graphical user interfaces from openEHR archetypes. Stud Health Technol Inform. 124 (2006) 221-6.

Scott Bill. Designing Web Interfaces: Principles and Patterns for Rich Interactions. 1st Edition. O'Reilly, 2009.

TISS. Disponível em: www.ans.gov.br/portal/site/_hotsite_tiss/pdf/. Acessado em: 10 Out 2010.

Van der Linden H, Boers G, Tange H, Talmon J, Hasman A. (2003). PropeR: a multi disciplinary EPR system. Int. J. Med. Inform. 70 (2003) 149–160.

Van der Linden H, Grimson J, Tange H, Talmon J, Hasman A. Archetypes: the PropeR way. Medinfo 11. 2004: 1110–1114.

Van der Linden H, Talmon J, Tange H, Grimson J, Hasman A. 2005. PropeR revisited. Int. J. Med. Inform. 74 (2005) 235–244.

Van der Linden H, Austin, T, Talmon J. Generic screen representations for future-proof systems, is it possible? There is more to a GUI than meets the eye. Comput Methods Programs Biomed. 2009; (95): 213-226.

Sittig DF. (2008). SANDS: a service-oriented architecture for clinical decision support in a National Health Information Network. In Journal Biomedical Informatics. pp 962-981.

W3c. Disponível em: http://www.w3.org/. Acessado em: 31/01/2011.