Diego Santos de Andrade Pizzol

MULTIPERSON-CDS: FRAMEWORK MULTIPROPÓSITO E PERSONALIZÁVEL BASEADO EM ONTOLOGIA PARA O APOIO À DECISÃO CLÍNICA

João Pessoa

2011

Diego Santos de Andrade Pizzol

MULTIPERSON-CDS: FRAMEWORK MULTIPROPÓSITO E PERSONALIZÁVEL BASEADO EM ONTOLOGIA PARA O APOIO À DECISÃO CLÍNICA

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

Diego Santos de Andrade Pizzol

MULTIPERSON-CDS: FRAMEWORK MULTIPROPÓSITO E PERSONALIZÁVEL BASEADO EM ONTOLOGIA PARA O APOIO À DECISÃO CLÍNICA

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

COMISSÃO EXAMINADORA

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

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

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

_____
Prof. Dr. Marco Antônio Gutierrez
Examinador Externo (Incor HC.FM-USP)

João Pessoa, ___ de _____ de 2011.

Agradecimentos

Agradeço a Deus por me dar saúde e condições de viver uma vida plena, mesmo que seja menos do que eu desejasse e mais do que eu merecesse.

Depois dele, mas não menos grato, aos meus pais, Kátia e Domingos, que sempre se empenharam em me dar tudo do bom e do melhor. Graças à bondade, ao zelo e à atenção de minha mãe e ao bom-humor e à cautela de meu pai que consegui ter serenidade, força e perspicácia para trilhar os caminhos da vida. Todas as minhas vitórias foram, por conseqüência, vitórias deles.

Aos diversos amigos que fiz durante o curso de graduação e que as circunstâncias da vida fizeram apenas nos unir cada vez mais. Deles, a minha gratidão é maior para com aquele que já não está entre nós em carne, mas sempre é lembrado com muita alegria: Gustavo Cavalcanti.

Ao meu orientador, Gustavo Motta, pelo exemplo de caráter, pela oportunidade oferecida e pela confiança depositada em mim.

Aos companheiros de laboratório do LArqSS que souberam transformar um ambiente de estudo e trabalho num ambiente agradável, coeso e animado.

À FINEP, pelo fomento ao projeto OpenCTI.

"You, me or nobody, is gonna hit as hard as life. But ain't about how hard you hit. It's about how hard you can get hit... and keep moving forward... how much you can take, and keep moving forward. THAT'S HOW WINNING IS DONE!..."

Rocky Balboa (Rocky VI)

Resumo

PIZZOL, D. S. A. MultiPersOn-CDS: Framework Multipropósito e Personalizável baseado em Ontologia para o Apoio à Decisão Clínica. 149 p. Dissertação (Mestrado) – Departamento de Informática, Universidade Federal da Paraíba, João Pessoa, 2011.

Cada vez mais organizações de saúde são atraídas pelas vantagens oriundas do processo de automatização do atendimento em saúde, por meio da tecnologia da informação, como agilidade, precisão, segurança, controle e economia. Fato evidenciado pelo aumento nos últimos anos dos esforços de diversos países para a elaboração de projetos que visem alavancar o uso da tecnologia da informação em saúde. Apesar disso, não existe um padrão, nem tampouco um conjunto de padrões, que resolvam garantidamente todos os problemas das organizações de saúde, devido à complexidade do domínio de saúde e as discrepâncias inerentes a cada uma delas. Além disso, o uso do registro eletrônico em saúde isoladamente garante melhoria apenas na acessibilidade e legibilidade das informações. Para alcançar níveis satisfatórios de segurança ao paciente, de qualidade no atendimento em saúde, aumentando a eficiência e diminuindo o custo, faz-se necessário a utilização de um sistema de apoio à decisão clínica. Embora seja notória a sua relevância e haja várias soluções, utilizando o apoio à decisão clínica, atualmente, há poucas implementações de apoio à decisão para a rotina clínica que tenham alcançado resultados significativos em relação à melhoria dos resultados e dos processos em saúde. O MultiPersOn-CDS objetiva solucionar esse problema, através de um framework estendível e personalizável que possibilite, através do uso de ontologias, a elaboração de agentes contextuais especializados para o apoio a decisão clínica com foco tanto na assistência clínica quanto no fluxo de trabalho dos profissionais de saúde, visando sobretudo a melhoria da prestação de saúde. Com a construção e validação do MultiPersOn, permite-se a criação de agentes multipropósitos de apoio à decisão clínica, capazes de se adaptar às mudanças inerentes à área de saúde, tornando possível a disseminação desses agentes pelas organizações de saúde, devido a maior simplicidade de realizar alterações e customizações no *framework*. Como resultado, os esforços podem então ser concentrados na construção de agentes de apoio à decisão clínica, sem se preocupar na adequação deles com as aplicações, tentando-se assim diminuir as barreiras tecnológicas para a utilização e disseminação das ferramentas de apoio à decisão clínica nos registros eletrônicos em saúde.

Palavras-chave: apoio à decisão clínica, ontologia, agentes multipropósitos.

Abstract

PIZZOL, D. S. A. MultiPersOn-CDS: Framework Multipropósito e Personalizável baseado em Ontologia para o Apoio à Decisão Clínica. 149 p. Dissertation (Masters) – Departamento de Informática, Universidade Federal da Paraíba, João Pessoa, 2011.

Over and over health organizations are attracted by the benefits from the process of automation in health care through information technology, such as speed, accuracy, security, control and economy. Evidenced in recent years by the effort increasing of several countries for the development of projects that aim to leverage the use of information technology in health. Nevertheless, there is no standard, nor a set of standards, to solve guaranteed all the problems of health organizations, due to the complexity of the health field and the inherent discrepancies in each one of them. In addition, the use of electronic health record alone ensures only improvement on accessibility and readability of information. To achieve satisfactory levels of patient safety, quality in health care, increasing efficiency and reducing the cost, it is necessary to use a clinical decision support system. Although it is notorious its relevance and there are several solutions using clinical decision support, only few implementations for the clinic routine have achieved significant results in relation to improving results and procedures in health. The MultiPersOn-CDS aims to solve this problem through an extendible and customizable framework that enables, through the use of ontologies, the development of specialized contextual agents to support clinical decision making with a focus on clinical care and in the workflow of professional health, adressed mainly at improving the health care. For the construction and validation of MultiPersOn allows the creation of multipurpose agents of clinical decision support, able to adapt to the inherent changes in the health field, making possible the dissemination of these agents by health organizations due to greater simplicity of make changes and customizations in the framework. As a result, efforts can then be concentrated on building clinical decision support agents, without worrying about the adequacy of them with applications, thus trying to reduce the technological barriers to the use and dissemination of clinical decision support tools in electronic health record.

Keywords: clinical decision support, ontology, multipurpose agents.

Lista de Figuras e Gráficos

Lista de Tabelas

Lista de Acrônimos

ANS	Agência Nacional de Saúde Suplementar
API	Application Programming Interface
CDS	Clinical Decision Support
EAV	Entity-Attribute-Value
EDA	Event-driven Architecture
EHR	Electronic Health Record
EJB	Enterprise Java Beans
FINEP	Financiadora de Estudos e Projetos
GLIF	Guideline Interchange Format
GUI	Graphical User Interface
HL7	Health Level Seven
HULW	Hospital Universitário Lauro Wanderlei
KDOM	Knowledge-Data Ontological Mapper
LArqSS	Laboratório de Arquitetura e Sistemas de Software
MACA	Middleware de Autorização e Controle de Autenticação
NOB	Norma Operacional Básica
OMG	Object Management Group
OWL	Web Ontology Language
PRC	Padronização de Registros Clínicos
RCT	Randomized Controlled Trial
RDF	Resource Description Framework
RES	Registro Eletrônico em Saúde
RHIO	Regional Health Information Organizations
SNOMED	Systematized Nomenclature of Medicine
TISS	Troca de Informação em Saúde Suplementar
UFPB	Universidade Federal da Paraíba
UML	Unified Modeling Language
URI	Universal Resource Identifier
UTI	Unidade de Terapia Intensiva
W3C	World Wide Web Consortium

Sumário

Capítulo

1

Introdução

"O descontentamento é o primeiro passo na evolução de um homem ou de uma nação."

Oscar Wilde

Cada vez mais organizações de saúde são atraídas pelas vantagens oriundas do processo de automatização do atendimento em saúde, por meio da tecnologia da informação, como agilidade, precisão, segurança, controle e economia. Desde meados dos anos 60, esse processo revelou inegáveis benefícios, porém, até hoje não tem sido eficazmente desfrutado, estando aquém do avanço alcançado em outras áreas do conhecimento, como engenharia, física e finanças (GREENES, 2006). Isto se dá, principalmente, pelo fato de grande parte das soluções tecnológicas em saúde terem sido concebidas, provavelmente devido à sua complexidade e custo, para as necessidades específicas de cada organização de saúde isoladamente.

Por esse motivo, nos últimos anos aumentaram os esforços por diversos países para a elaboração de projetos que visem alavancar o uso da tecnologia da informação em saúde, tais como, RHIO (*Regional Health Information Organizations*) nos Estados Unidos, *Connecting for Health Initiative* na Inglaterra, *Medcom* e *Sundhed.dk* na Dinamarca e *Health@net* na Áustria. Além disso, tem-se empenhado, também, em aprimorar padrões já estabelecidos nessa área, como DICOM (DICOM, 2003) e HL7 Version 3 (HEALTH LEVEL SEVEN), fomentando assim a disseminação do prontuário eletrônico do paciente (RES – Registro Eletrônico em Saúde) (HAFNER; BREU, 2008).

1.1 Motivação

Apesar de todo esse esforço global, não existe um padrão, nem tampouco um conjunto de padrões, que resolvam garantidamente todos os problemas das organizações de saú-

de, devido à complexidade do domínio de saúde e as discrepâncias inerentes a cada uma delas. Dessa maneira, esforços são despendidos na elaboração de vários padrões com propósitos similares para contextos diferentes, oriundos de várias iniciativas, que ao invés de facilitar o desenvolvimento desses sistemas, acabam tornando-os cada vez mais heterogêneos. Assim, embora tenha havido um avanço ao nível nacional das iniciativas, ao se engajarem em resolver o problema, somente com uma maior coordenação global é que isto poderá ser solucionado.

E mais, sistemas de classificação e codificação do conhecimento são fundamentais para o pleno funcionamento do RES, uma vez que, a falta deles aumenta a chance de ocorrerem erros. Isto ocorre, pois eles interferem diretamente no significado e na qualidade das informações que, por ventura, podem ser trocadas entre diferentes organizações de saúde. E mesmo assim, isso não garante a elaboração de uma estrutura universal que seja capaz de permitir que a equipe multidisciplinar de saúde usufrua de todas as informações requeridas durante o atendimento diário, sendo então necessária a junção das vantagens das informações estruturadas com a riqueza do texto livre (LEWIS, 2002).

Outro aspecto desses sistemas, é que eles servem apenas para um domínio específico de acordo com a linguagem ou o propósito de cada um deles. Por exemplo, alguns sistemas de classificação e codificação amplamente utilizados, como o CID-10 que classifica as doenças, lesões e causas de morte pela etimologia e localização anatômica no corpo (COTE *et al.*, 1993) tem um propósito bem delineado, enquanto que o SNOMED-CT que é uma terminologia médica geral para indexar eventos nos registros dos pacientes (COTE *et al.*, 1993), não possuem tradução oficial para o português ainda.

E ainda, o uso do RES isoladamente, garante apenas melhoria na acessibilidade e legibilidade das informações, sem se mostrar relacionado com avanços na qualidade do atendimento em saúde (LINDER *et al*, 2007). Para alcançar níveis satisfatórios de segurança ao paciente, de qualidade no atendimento em saúde, aumentando a eficiência e diminuindo o custo, faz-se necessário a utilização de um sistema de apoio à decisão clínica (*Clinical Decision Support*, CDS) (OSHEROFF *et. al*, 2006). O CDS tem como objetivo mostrar informações relevantes para melhorar a qualidade do atendimento ao paciente e da assistência à saúde em geral, com subsídios para formulação de políticas públicas.

Devido a sua importância e a sua atuação abrangente, uma gama de soluções divergentes tem sido criada nos últimos anos. Como, soluções que geram lembretes em papel para pacientes em atendimento (DINI *et al.*, 2000), que são direcionadas aos pro-

fissionais de saúde e completamente integradas ao RES (BATES *et al.*, 1999) ou que notificam por *pagers* os responsáveis por pacientes internados sobre alertas e resultados de exames (Kuperman *et al.*, 1999). Essas soluções variam nas funcionalidades: servindo para alertas de interação de medicamentos, para sugestão de diagnósticos, para a geração de lembretes, entre outros; variam nos meios utilizados: usando telefones, emails, ou o próprio RES; e nos seus usuários e alvos: podendo ser direcionadas para os profissionais de saúde atuarem sobre os pacientes internados ou pacientes em atendimento (que realizam consultas em ambulatórios) ou direcionadas para o próprio paciente.

Embora seja notória a sua relevância e haja várias soluções, utilizando o CDS, atualmente, há poucas implementações do CDS para a rotina clínica que tenham alcançado resultados significativos em relação a melhoria dos resultados e dos processos em saúde (CHAUDHRY *et al.*, 2006). Numa tentativa de identificar e descrever os desafios-chave que devem ser superados para a implantação de um CDS de alta qualidade, Sittig *et al.* (2008) elaborou uma lista com os dez conceitos mais desafiadores na decisão clínica. Sendo eles: melhorar a interface homem-computador; resumir informações ao nível do paciente; priorizar e filtrar recomendações para o usuário; combinar recomendações para pacientes com comorbidades; utilizar informações em texto livre para direcionar o CDS; priorizar o desenvolvimento e implementação do conteúdo do CDS; minerar grandes bases de dados clínicos para criar novos CDS; disseminar boas práticas no projeto, desenvolvimento e implementação do CDS; criar uma arquitetura para compartilhar módulos executáveis e serviços do CDS e criar repositórios de CDS acessíveis pela internet.

Isto corrobora com Greenes (2006), que afirma que as razões para a falta da ampla disseminação e adoção do CDS são ambas tecnológicas e não-tecnológicas, aparentando estarem relacionadas com a complexidade de elaborar o CDS. Sendo, isto verdade não apenas para tipos complexos de CDS, como seleção de tratamentos e diagnóstico diferencial, mas até mesmo para formas bem mais simples, como alertas e lembretes. Na verdade, conclui Greenes (2006), a dificuldade em implantar e disseminar o CDS é em grande parte, devido à falta de reconhecimento do quão difícil a tarefa é e a falta de ferramentas e recursos para tornar a tarefa mais fácil.

De fato, obter informações clínicas e inteligentemente filtrar os dados dos pacientes para a equipe de saúde ou para o próprio paciente, envolve um grande desafio. Para ajudar a garantir que o CDS fosse implantado com sucesso, Sirajuddin *et al.* (2009) criou um conjunto de regras para serem aplicadas ao CDS: *five rights*. Ou seja, o CDS deve interferir com a informação certa (*right information*), para a pessoa certa (*right per-*

son), no formato certo (*right format*), através do canal certo (*right channel*), no ponto certo do fluxo de trabalho (*right time in workflow*).

Além disso, vários estudos indicam a possibilidade do surgimento de problemas imprevisíveis ou conseqüências adversas não desejáveis com a utilização do computador no auxílio à saúde (ASH *et al.*, 2004). Isto também pode ocorrer com o CDS, segundo Ash et al. (2007), que constatou a existência de algumas conseqüências indesejáveis, tais como: substituição ou mudança de papéis antes realizados por pessoas; problemas com a atualização do conteúdo do CDS; erros ou enganos no conteúdo do CDS; rigidez do sistema que por vezes atrapalha no fluxo de trabalho; sobrecarga de alertas (*alert fatigue*) que podem desviar ou cansar o usuário; potencias erros oriundo de problemas diversos, como a falta de resposta de uma ação imediata ou problemas com o auto-completar.

1.2 Objetivo

Este trabalho tem como objetivo geral elaborar um *framework* estendível e personalizável que facilite a criação e disseminação de mecanismos de suporte à decisão clínica, possibilitando a elaboração, através de uso de ontologias, de agentes contextuais especializados para o apoio a decisão clínica. Visando, sobretudo, a melhoria da prestação de saúde de uma forma geral, o *framework* permite o enfoque tanto na assistência clínica quanto no fluxo de trabalho dos profissionais de saúde. E, além disso, deve prever suporte para as constantes variações e especificidades do domínio da saúde, facilitar a manutenabilidade e a interoperabilidade entre o CDS e as partes envolvidas, bem como, proporcionar uma maior separação entre o domínio da aplicação e a lógica interna dos agentes de CDS que irão atuar sobre esse domínio.

Para alcançar este objetivo geral, faz-se necessária a realização de submetas, descritas através dos seguintes objetivos específicos:

Objetivo 1. Definir uma estrutura na ontologia que serva de modelo para a criação dos agentes de CDS;

Objetivo 2. Implementar um componente responsável por construir os agentes de CDS, utilizando as informações contidas na ontologia e na lógica interna de cada agente;

Objetivo 3. Implementar um componente que gerencia as notificações da interface com o usuário, permitindo assim encaminhar a lógica dos agentes de CDS para a execução ao passo que estes forem ativados pelas ações do usuário ou pelo sistema;

Objetivo 4. Implementar um componente para executar a lógica dos agentes de CDS, de acordo com a descrição contida na ontologia e no arquivo que contêm sua lógica interna;

Objetivo 5. Implementar um componente que forneça de maneira transparente aos agente de CDS a possibilidade de adicionar recursos de infra-estrutura, como persistência de dados e consulta à bases de dados remotas;

Objetivo 6. Implementar um componente para interagir com a interface com o usuário de acordo com a descrição na ontologia e o resultado da lógica de cada um dos agente do CDS;

Objetivo 7. Integrar o MultiPersOn com o OpenCTI[1], possibilitando que o *framework* capture as informações da interface com o usuário oriundas do OpenCTI e envie de volta para ele o resultado da lógica de cada um dos agentes de CDS;

Objetivo 8. Criar alguns tipos experimentais de agentes de CDS com a finalidade de validar e comprovar a aplicabilidade do *framework*:

a. Elaborar um agente para a validação de campos, tanto com base em valores limítrofes fisiológicos, como também utilizando outros campos;
b. Elaborar um agente para o cálculo de fórmulas automáticas que utilizem diversos campos;
c. Elaborar um agente para sugerir um conjunto de opções ao usuário, podendo este obedecer a vários critérios, como uma ordenação conforme seu histórico de entradas ou conforme o histórico de todos os pacientes;
d. Elaborar um agente para o aviso de tarefas a serem realizadas que alerta o usuário sobre um evento que deve ser realizado num dado instante de tempo;
e. Elaborar um agente para a leitura de dispositivos externos, a fim de simular a leitura das máquinas à beira do leito dos pacientes;

[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.

f. Elaborar um agente para informar dados adicionais, previamente configurados ou dinamicamente gerados através de informações contidas em base remotas de dados.

1.3 Justificativa

Enquanto que uma solução ideal para os diversos problemas na área de saúde não é encontrada, muitos padrões e soluções específicas de diferentes complexidades são criadas. Tentando-se fugir disto, objetivou-se com esse trabalho a elaboração de uma solução, que ao contrário das demais, servisse para vários fins, com foco nos problemas de baixa complexidade, como validadores, sugestões, cálculo de fórmulas, entre outras, que tivessem uma utilidade prática bastante elevada, durante o preenchimento do RES.

Ao alcançarmos os objetivos específicos enumerados na seção 1.2, estaremos beneficiando os usuários RES que utilizam o *framework*, uma vez que, para cada um deles temos:

Justificativa 1. A utilização de ontologias para a representação do conhecimento possibilita a interpretação do conhecimento tanto por pessoas quanto pelas máquinas, já que, ela facilita a organização e clarifica a semântica do conhecimento armazenado;

Justificativa 2. Ao separar as informações sobre os agentes de CDS da sua lógica interna, permite-se especializar sua criação e diminuir a dependência dos seus aspectos tecnológicos, aumentando assim a eficácia e a aplicabilidade do *framework*;

Justificativa 3. Utilizando a ontologia e integrando-se com um dispositivo de interação com o usuário, permite-se interagir diretamente com o usuário, da maneira mais indicada, no momento em que a intervenção deve ocorrer;

Justificativa 4. A adição de recursos de infra-estrutura, como persistência de dados e o acesso à bases de dados remotas, permite inserir maior robustez ao *framework*, oferecendo um meio para agentes com estados guardarem suas informações e inserir nos agentes informações provenientes de fontes externas;

Justificativa 5. A interação com o usuário, por meio dos dispositivos de interfaceamento com o usuário, agrega valor, de fato, ao RES, pois possibilita a ação efetiva dos agentes de CDS;

Justificativa 6. A integração do *framework* com o OpenCTI, evidencia a importância, robustez e a aplicabilidade do *framework* num contexto institucional real;

Justificativa 7. A elaboração de diversos tipos experimentais de agentes de CDS serve como validação da sua multi-aplicabilidade, bem como, corrobora com os seguintes benefícios:

a. A validação de campos é uma tarefa de complexidade simples, entretanto, pode evitar erros e enganos no dia-a-dia do preenchimento do RES;
b. O cálculo de fórmulas automático, apesar de também ser uma tarefa bastante simples, agiliza e otimiza o atendimento em saúde, garantindo, ainda, sua corretude;
c. A sugestão de preenchimento, possibilita a otimização do atendimento em saúde, através da adaptação do *framework* às diferentes necessidades dos profissionais de saúde;
d. O aviso de tarefas permite lembrar a ocorrência de atividades rotineiras que, por ventura, foram esquecidas pelos profissionais de saúde, aumentando assim a qualidade do atendimento de saúde.
e. A leitura automática de dispositivos externos, agiliza o atendimento em saúde e garante a corretude das informações, evitando erros de digitação;
f. O fornecimento de informações adicionais, melhora a qualidade da prestação de saúde, em situações duvidosas ou não usuais para os profissionais de saúde.

1.4 Metodologia

A elaboração deste trabalho seguiu as seguintes atividades:

- Levantamento Bibliográfico: Coleta de artigos científicos e livros das mais diversas áreas que abrangessem de alguma forma o CDS. A partir daí, filtragem das publicações mais recentes que fossem relevantes.
- Elaboração de um modelo para o CDS: Projeto de um modelo para o CDS que levasse em consideração os aspectos abordados na literatura e ao mesmo tempo acrescente algo de inovador.

- Comparação do modelo criado com a literatura: Confronto entre a solução elaborada com os aspectos positivos e negativos da literatura para ponderar as características de fato inovadoras.
- Implementação do *framework*: Uma implementação de uma versão beta, preliminar, da solução com todas as possíveis funcionalidades.
- Modelagem de agentes de CDS: Criação de alguns agentes de CDS, através da modelagem deles na ontologia para serem usados durante a implementação do *framework*.
- Validação do *framework*: Execução, teste e validação de todos os tipos de agentes supracitados para garantir a segurança, robustez e diversidade do *framework*.

1.4.1 Desenvolvimento

As etapas de implementação do *framework,* modelagem de agentes de CDS e validação do *framework* foram realizadas através do uso de algumas tecnologias aplicadas sob um processo de desenvolvimento, conforme segue descrito.

1.4.1.1 Tecnologias

Como tecnologia de representação de conhecimento, utilizou-se a linguagem OWL (*Ontology Web Language*), junto do editor, Protégé-OWL, para a manipulação dos conceitos na ontologia. Na implementação do *framework*, utilizou-se a linguagem Java, usufruindo-se dos componentes do JBoss Seam que foram criados para abstrair a complexidade de vários níveis arquiteturais, facilitando seu uso e servindo como base para a construção de mecanismo robustos. Para por em funcionamento essas tecnologias, utilizou-se o servidor de aplicação, JBoss AS. Além disso, a linguagem Groovy foi escolhida, funcionando como linguagem de script, para codificar a lógica dos agentes de CDS. E para dar suporte à linguagem Groovy, bem como, a tantas outras, usou-se uma API de execução de scripts para Java, o JSR-223.

1.4.1.2 Processo de Desenvolvimento

O processo de desenvolvimento adotado foi o evolucionário, onde as etapas de especificação, projeto, implementação e validação ocorrem simultaneamente. Uma especificação preliminar foi realizada e uma versão inicial da implementação foi construída para atender os requisitos da especificação preliminar, necessitando de sucessivas iteração até por fim obter um produto que atendesse a todas as especificações.

Durante esse processo de desenvolvimento, foi utilizada a linguagem de modelagem UML (OMG, 2007) para representar graficamente os elementos do *framework*, a fim de

facilitar sua compreensão e legibilidade. Alguns desses diagramas estão presentes durante o decorrer do texto.

1.5 Estrutura do Trabalho

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

- No segundo capítulo, "Aspectos relevantes para o MultiPersOn", discutiremos os conceitos envolvidos no entendimento do problema que o MultiPersOn se propõe a resolver, bem como, a classe de problemas relacionados à construção do MultiPersOn-CDS, seguida de algumas soluções relacionadas com o tema deste trabalho;
- No capítulo seguinte, "Design do MultiPersOn-CDS", explicitaremos de maneira detalhada as características e os componentes da solução proposta;
- No quarto capítulo, "O Modelo Semântico do MultiPersOn", analisaremos os pormenores da estrutura do modelo semântico do *framework* e suas implicações;
- No capítulo subseqüente, "Resultados", examinaremos os resultados obtido com uma a partir da integração do MultiPersOn com uma aplicação que será utilizada num cenário real;
- No sexto e último capítulo, "Considerações Finais", discutiremos sobre as vantagens da solução proposta, comparando com as soluções existentes, a fim de evidenciar as inovações trazidas pelo *framework*;

Capítulo

2

Aspectos relevantes para o MultiPersOn

"Se queres prever o futuro, estuda o passado."

Confúcio

A origem do CDS é antiga, através da incorporação dos sistemas de apoio a decisão aos sistemas de informação em saúde, embora esses tivessem suporte apenas a uma análise retrospectiva dos dados financeiros e administrativos (OLIVEIRA, 2002). Desde essa época, o CDS evoluiu bastante, passando a incorporar técnicas sofisticadas, como a inteligência artificial, para agregar valor de várias formas ao registro eletrônico em saúde e objetivar tanto a melhoria da qualidade do atendimento em saúde, quanto a eficiência dos recursos, que por sua vez, acabam reduzindo os custos com materiais, procedimentos e exames.

2.1 O registro eletrônico em saúde

Todavia, muito antes da origem do CDS, isto é, no século V a.C., Hipocrates já incentivava os médicos a realizarem registros escritos dos achados médicos (BEMMEL; VAN BEMMEL, 1997). Estes registros foram os precursores do atual prontuário do paciente, que visa garantir que os fatos e eventos clínicos de cada paciente fiquem registrados de maneira sistemática, permitindo o acesso a essas informações pelos diferentes profissionais de saúde (SLEE V.; SLEE D.; SCHMIDT, 2000).

Assim, além de possibilitar a comunicação entre as equipes de saúde responsáveis pelo atendimento e criar uma memória sobre a saúde do paciente, ele, também, serve como registro legal das ações médicas, fomentando a pesquisa e a gerência administrativa de cobranças e autorizações.

Apesar de ser um modelo utilizado há vários anos, o prontuário em papel possui algumas limitações, a exemplo da falta da disponibilidade, pois pode ser perdido ou estar sendo usado e da falta de legibilidade, pois seu conteúdo é livre e manuscrito, podendo estar incompleto ou ambíguo.

Com o intuito de melhorar a acessibilidade, segurança e agilidade da prestação de saúde com o prontuário, idealizou-se uma versão eletrônica do prontuário, denominada registro eletrônico em saúde (RES), que objetiva ser um repositório das informações de saúde, clínicas e administrativas, ao longo da vida de cada paciente (MASSAD; MARIN; AZEVEDO NETO, 2003).

Apesar da sua utilização comprovadamente melhorar a qualidade do atendimento em saúde, há diversos riscos e obstáculos que dificultam o seu desenvolvimento e implantação (MURPHY; HANKEN; WATERS, 1999). Por isso, atualmente, nos Estados Unidos, apenas entre 1,5% e 5,9% dos hospitais possuem soluções abrangentes de RES, em contrapartida, 7,6% a 43% dos hospitais têm implementações básicas de sistemas de RES (JHA; *et al.*, 2009; PEDERSEN; GUMPPER, 2008). Essa variação de valores decorre da falta de padronização da definição das funcionalidades básicas necessárias para os sistemas de RES. Por isso, definições mais restritivas que levam em consideração mais aspectos, como integração múltipla com sistemas clínicos, acesso a testes de laboratórios e sistemas de prescrição médica, possuem valores mais baixos.

2.2 Padrões de informática em saúde

Para permitir a ampla cobertura da prestação de saúde, colaborando com a melhoria da qualidade e da eficiência do seu atendimento, necessita-se de mecanismos que dêem suporte ao compartilhamento de informações clínicas entre as diferentes organizações de saúde. Por isso, com o intuito de permitir a comunicação e coordenação entre os RES dessas organizações, uma série de padrões de arquitetura para o RES, que é o elemento central das aplicações em saúde, foi desenvolvida internacionalmente.

2.2.1 Padrões internacionais

Os padrões internacionais mais recentes criados por algumas organizações e comitês, a exemplo do HL7, CEN/TC251 e OpenEHR, utilizam uma abordagem com dois modelos na sua arquitetura, sendo um genérico e outro específico. O modelo genérico lida com os conceitos tecnológicos, referentes a aspectos funcionais, enquanto que o outro modelo representa os conceitos específicos do domínio, necessários para os aspectos semânticos (BLOBEL, 2006).

O HL7 é uma organização sem fins lucrativos, denominada *Health Level Seven*, que definiu um conjunto de padrões proprietários para a informática em saúde (HEALTH LEVEL SEVEN). Dentre os vários padrões estabelecidos pelo HL7, o padrão de comunicação, HL7 Version 2.x, foi o que obteve maior aceitação internacional. Esse padrão tem a finalidade de permitir a comunicação de dados clínicos, administrativos e financeiros entre diferentes organizações de saúde. Recentemente, foi lançado o HL7 Version 3.x que utiliza um novo paradigma para a troca de mensagens (MDF – *Message Description Framework*).

O CEN/TC251 é um comitê técnico Europeu responsável pela normatização da área de saúde na Europa e composto por quatro grupos de trabalho, sendo o WorkGroup I (WG I) responsável pela elaboração de padrões para RES, através do estabelecimento de princípios para a representação do conteúdo e da estrutura do RES; da forma de representação de conceitos e terminologias; e dos mecanismos para a troca de informações (COMITÉ EUROPÉENDE NORMALISATION).

O openEHR é uma fundação sem fins lucrativos com o objetivo de desenvolver especificações abertas, detalhadas e testadas para normatizar a representação e comunicação do RES, baseado em mais de quinze anos de pesquisas e experiências reais de implementação. O openEHR estipula um série de recursos para o RES, como por exemplo, modelos de informação e de serviços; e workflow de informações clinicas e demográficas. Além disso, ele disponibiliza uma implementação de referência com código aberto que facilita seu entendimento e uso (OPENEHR FOUNDATION).

2.2.2 Padrões Nacionais

No Brasil, três iniciativas de padronização se destacam: a padronização de registros clínicos (PRC), o cartão nacional de saúde e o padrão para troca de informação em saúde suplementar (TISS).

A PRC surgiu em 1998 com a criação de um Comitê Temático Interdisciplinar (CTI) para estabelecer, através de padrões abertos, padrões para a construção de prontuários. O PRC estabeleceu a padronização de dados relativos à identificação do paciente, à instituição de saúde, à fonte pagadora, ao estado clínico e aos procedimentos realizados.

O cartão nacional de saúde foi concebido em 1996 pela Norma Operacional Básica (NOB) com o intuito de identificar individualmente os pacientes e possibilitar o acompanhamento do RES independente da localização das organizações de saúde, fomentando assim a construção do repositório nacional de atendimentos em saúde. Para

tal, o padrão define a estrutura e o conteúdo das informações do RES, permitindo assim sua interoperabilidade (MASSAD;MARIN;AZEVEDO NETO, 2003).

O TISS foi estabelecido pela ANS (Agência Nacional de Saúde Suplementar) com o objetivo de padronizar a troca eletrônica de informações administrativas e financeiras entre operadoras e prestadores de saúde, simplificando os processos envolvidos para reduzir custos entre os participantes (operadoras e prestadoras de serviços) e a ANS.

2.3 O CDS

O esforço desprendido por diversas organizações internacionais em desenvolver soluções que façam uso do RES, evidencia as vantagens que podem ser alcançadas na melhoria da qualidade da prestação de saúde, através de sua utilização. Apesar disso, enquanto que as terapias médicas evoluem cada vez mais, a prática médica fica aquém, possibilitando a existência de erros que poderiam ser evitados. Neste aspecto, os sistemas de informação, apesar de serem ainda subutilizados, têm um papel crítico (INSTITUTE OF MEDICINE, 2001). Em especial, a utilização do CDS não constitui apenas uma boa idéia, mas sim, uma parte essencial e central das funcionalidades do RES (CHO et al., 2010). Sua utilização pode reduzir erros no tratamento, melhorando assim a qualidade da prestação de saúde.

2.3.1 Definição do CDS

Talvez, por sua grande abrangência e utilidade, a tarefa de definir o CDS de maneira sucinta acabe sendo um trabalho laborioso e delicado. Greenes *et al.* (2006) tenta delimitar o CDS através das suas características e vantagens, afirmando que:

1) O CDS é o uso do computador para levar o conhecimento relevante para dar suporte ao atendimento em saúde e o bem estar do paciente.

Osheroff et al. (2009) tenta ir além e define o CDS a partir dos seus beneficiários e do instante de sua atuação. Ele afirma que:

2) O CDS fornece aos profissionais e à equipe de saúde, aos pacientes e a outros indivíduos, conhecimento e informações personalizadas específicas, inteligentemente filtradas e apresentadas no instante adequado para melhorar a saúde e o atendimento em saúde.

Arriscando-se mais, Sim et al. (2001) tenta ser mais pragmático e direto, ao afirmar que:

3) Os sistemas de CDS são *softwares* projetados para fornecer uma ajuda direta na tomada de decisão dos profissionais de saúde, através da comparação das

características individuais do pacientes com uma base informatizada de conhecimento em saúde. Gerando, assim, afirmações e recomendações específicas para cada um deles a fim de serem apresentadas aos profissionais ou ao paciente durante a tomada de decisão.

Com o auxílio dessas três definições, podemos entender melhor as características do CDS e ao mesmo tempo formular uma definição própria que tente englobar os aspectos mais importantes citados por cada uma delas. Assim, o CDS tem como foco tanto os profissionais de saúde, quanto o próprio paciente ou até mesmo outros indivíduos, que, através da filtragem de informações, utilizando uma base de conhecimento, fornece, no momento apropriado da prestação de saúde, afirmações ou recomendações que podem ajudar no atendimento do paciente ou, até mesmo, subsidiar políticas públicas em saúde.

2.3.2 Classificações do CDS

Devido à grande variabilidade das soluções existentes de CDS e a dificuldade de adequá-las à definição de CDS, vários autores tentaram utilizar diferentes abordagens para classificar essas soluções de acordo com diferentes características encontradas.

A seguir, analisaremos três dessas abordagens, onde diferentes perspectivas foram levadas em consideração na sua elaboração. A primeira abordagem classifica os CDS de acordo com o a base de conhecimento utilizada, seja através de uma base fixa que é consultada para realizar o apoio à decisão ou através de nenhuma base de conhecimento previamente existente, necessitando então que o conhecimento seja adquirido através de técnicas de aprendizagem. A segunda abordagem tenta analisar quatro aspectos da aplicabilidade funcional das soluções de CDS, isto é, o instante de interação do CDS no processo de tomada de decisão, a iniciativa da interação do CDS com o usuário, o nível de customização alcançado de acordo com as informações clínicas específicas e, por fim, a facilidade de acesso relacionado com o esforço e tempo demandado para obter uma informação. Por último, utilizaremos a taxonomia de Berlin *et al.* (2006) para analisar a abordagem que faz uso de um esquema taxonômico, através de um divisão sistemática das características específicas das soluções de CDS.

2.3.2.1 CDS com e sem nenhuma base de conhecimento

Uma dessas categorizações envolve a distinção entre a base de conhecimento usada pelo CDS, isto é, o CDS baseado em conhecimento e CDS baseado em nenhum conhecimento (BERNER, 1999). O CDS baseado em conhecimento surgiu a partir dos sistemas especialistas, que objetivavam criar programas que simulassem o raciocínio humano

(SHORTLIFFE, 1973). Entretanto, esse tipo de CDS não visa mais simular o raciocínio humano, mas, sim, auxiliar os profissionais de saúde na tomada de decisão, através do fornecimento de informações específicas do paciente para ajudar o usuário, ao invés de fornecer a solução para os problemas (MILLER, 1990).

Atualmente o CDS baseado em conhecimento vem sendo usado para diversos propósitos, como o auxílio ao diagnóstico, auxílio na prescrição de medicamentos, auxílio aos pedidos de exames, entre outros. Para inferir as informações sobre um paciente, ele utiliza regras de produção do tipo IF-THEN ou associações probabilísticas de valores. As regras de produção permitem sugerir ações quando certas condições são satisfeitas e as associações probabilísticas possibilitam relacionar valores, como os sinais e sintomas de doenças para a sugestão de um diagnóstico (BERNER, 1999).

Por outro lado, o CDS baseado em nenhum conhecimento, usa uma técnica de inteligência artificial, chamada de máquina de aprendizagem, que permite ao computador aprender a partir de experiências passadas ou reconhecer padrões nos dados clínicos (MARAKAS, 1999), sendo as redes neurais artificiais e os algoritmos genéticos utilizados nesse tipo de CDS.

2.3.2.2 As quatro dimensões do CDS

Uma abordagem diferente foi utilizada por Perreault e Metzger (1999) para avaliar as soluções de CDS. Os autores criaram uma classificação baseada em quatro aspectos distintos, denominado por eles de dimensões, para analisar os sistemas de CDS.

A primeira dimensão leva em consideração o instante da atuação do CDS na tomada de decisão e no processo de prestação de saúde, podendo ocorrer em três instantes: durante o processo de coleta de informações, através do fornecimento de informações relevantes, durante o processo de tomada de decisão, através da listagem das intervenções apropriadas ou depois da tomada de decisão, através da revisão das ações e de possíveis problemas.

A dimensão seguinte avalia a interação do CDS com o usuário sobre a perspectiva da parte que toma a iniciativa na interação. Assim, quando o CDS age automaticamente sem o pedido do usuário, diz-se que a interação é ativa e, analogamente, quando o CDS reage a uma requisição do usuário, denomina-se passiva. A interação ativa, geralmente, ocorre com o uso de alertas, *pop-up,* ou lembretes fixos no canto da tela, enquanto que a passiva ocorre, através de conjuntos de ordens (*order set*), contendo recomendações para os casos particulares ou através da consulta pelos profissionais de saúde

de informações clínicas específicas. Os autores ressaltam, ainda, que é importante encontrar o equilíbrio entre ambos os tipos de interação para permitir o auxílio aos profissionais de saúde sem atrapalhar no seu fluxo de trabalho.

Outra dimensão, diz respeito ao nível de customização alcançado de acordo com informações clínicas específicas. Para se adequar, da melhor maneira, às necessidades de cada situação clínica, é necessário algum tipo de integração com os dados clínicos dos pacientes, tornando o CDS, dessa forma, mais útil e ágil.

A última dimensão se relaciona com a facilidade de acesso, ligado ao esforço e ao tempo demandado para obter uma informação, como por exemplo, a quantidade de passos ou a necessidade de re-informar um dado para saber o estado clínico de um paciente. A portabilidade também deve ser considerada como um aspecto relevante na obtenção das informações, pois a facilidade do acesso às informações independentemente da localização favorece a efetivação do CDS.

2.3.2.3 Taxonomia do CDS

Outra tentativa de categorizar as discrepâncias entre os diferentes sistemas de CDS surgiu através do uso de taxonomias para descrever os mais variados aspectos encontrados. Existe uma série de taxonomias criadas para esse propósito, entretanto vamos nos concentrar na taxonomia elaborada por Berlin et al. (2006), pois, segundo ele, foi a primeira especialmente construída para facilitar a avaliação dos sistemas de CDS.

Berlin e seus colegas criaram sua taxonomia, dividindo os sistemas de CDS em cinco amplas categorias: contexto, fonte de conhecimento e dos dados, apoio à decisão, entrega da informação e fluxo de trabalho. Para melhor entendermos cada categoria, analisaremos a seguir cada categoria.

A categoria do contexto leva em consideração aspectos do ambiente clínico de operação do CDS (hospitais com pacientes internados ou ambulatórios com pacientes em atendimento); as tarefas clínicas suportadas pelo CDS (prevenção, diagnóstico, acompanhamento, entre outras); tipos de resultados otimizados pelo CDS (foco no auxílio ao paciente, ou há, também, foco na melhoria do sistema); a relação temporal entre a prestação de saúde à beira de leito com as recomendações do CDS (independentemente ou concomitantemente com o aparecimento dos problemas); a influência da organização no comportamento do CDS (interferências externas de setores administrativos ou financeiros); e as barreiras potenciais existentes para a realização das ações recomendadas (fatores socioeconômicos ou objetivos conflitantes).

Enquanto que a categoria da fonte de conhecimento e dos dados aborda temas, como a origem do conhecimento clínico usado na geração das recomendações (utilização de *guidelines* ou envolvimento de pessoas na construção do conhecimento); a origem dos dados dos pacientes utilizados para gerar as recomendações (dados capturados de fichas em papel ou de algum meio eletrônico, a exemplo do RES); o formato dos dados de entrada no CDS (texto livre ou um padrão estruturado); o nível de customização das recomendações em relação aos dados e ao histórico dos pacientes (recomendações genéricas ou específicas para cada caso); e os mecanismos de atualização do CDS para se adequar às mudanças na base do conhecimento (alterações automáticas ou manuais).

Por outro lado, a categoria de apoio à decisão se preocupa com os métodos utilizados pelo mecanismo de raciocínio do CDS na geração das recomendações (baseado em regras de produção ou em outros métodos, como redes neurais artificiais); a urgência com que as ações recomendadas pelo CDS necessitam ser realizadas; a clareza das recomendações (perceptíveis de forma explícita ou implícita pelo usuário); a complexidade logística das ações recomendadas pelo CDS (são ações simples ou de complexidade elevada); e os tipos de respostas requeridas para os usuários pelas recomendações do CDS (respostas não obrigatórias ou respostas com objetivo de seguir alguma regulamentação, com espaço para justificar o não fornecimento).

A categoria de entrega da informação, por sua vez, lida com questões referentes ao formato das recomendações do CDS (eletrônicas e integradas ao RES ou impressas em papel); ao modo de fornecimento das recomendações do CDS (solicitadas ou não pelos usuários do CDS); à integração com ferramentas que objetivem realizar as ações recomendadas pelo CDS; ao fornecimento de explicação sobre as recomendações oferecidas pelo CDS; e à interatividade existente na interação dos usuários com o CDS.

Por fim, a categoria de fluxo de trabalho aborda quais são os usuários que interagem com o CDS (pacientes, profissionais de saúde ou outros indivíduos); quem são as pessoas influenciadas diretamente pelas recomendações do CDS; quem são, se houver, os intermediários responsáveis por inserir os dados da fonte de conhecimento no CDS; quem são, se houver, os intermediários responsáveis por calibrar as recomendações geradas pelo CDS; e o nível de integração com o fluxo de trabalho da organização (se houve acréscimo de novos procedimentos ou mudança no fluxo de trabalho para atender as necessidades do CDS).

2.3.2.4 Correlação entre as classificações

Cada uma das classificações acima utiliza uma perspectiva diferente para realçar sobre um determinado ponto de vista os aspectos relevantes do CDS. Podemos, então utilizá-las para tentar classificar o MultiPersOn.

O MultiPersOn é uma solução que visa dar suporte ao desenvolvimento de mecanismos de CDS, não se preocupando como estes serão desenvolvidos. Desta forma, o *framework* suportaria soluções com ou sem base de conhecimento, segundo a classificação de Berner (1999).

Com relação à classificação de Perreault e Metzger (1999), o MultiPersOn foi projetado para ter flexibilidade de interação com sistemas de informação em saúde, proporcionando alcançar todas as dimensões sugeridas pelos autores. Isso significa que o *framework* suporta interações em todas as etapas possíveis do processo de prestação de saúde, seja de forma passiva ou ativa, sendo facilmente customizável aos contextos das organizações de saúde e objetivando o menor esforço para sua utilização.

Por sua vez, correlacionando com a taxonomia de Berlin *et al.* (2006), o *framework* pode ser aplicado amplamente à todas as categorias, dependendo do forma de aplicação do MultiPersOn e dos mecanismos de CDS utilizados, embora que sozinho não alcance nenhum das categorias, pois sua função é ser o alicerce para o desenvolvimento dos mecanismos de CDS.

2.3.3 Utilização do CDS

Conforme já foi dito, o CDS pode ser utilizado para proporcionar a melhoria do atendimento em saúde de diversas formas. Para compreender de fato, sua aplicabilidade, faz-se necessário analisarmos os diferentes tipos de ferramentas de CDS concebidas. Para tal, usaremos as diferentes soluções elencadas por Metzger & MacDonald (2002).

Os autores dividem as ferramentas de CDS em três categorias com base na tarefa que o profissional de saúde está tentando realizar. Sendo elas: o fornecimento à beira de leito de informações clínicas relevantes, a assistência no gerenciamento individual de pacientes ou a aplicação de recomendações de saúde sobre uma amostra de pacientes.

Acerca do fornecimento à beira de leito de informações clínicas relevantes, os autores destacam, conforme a Tabela 1, a utilização dessas ferramentas de CDS para responder as perguntas mais freqüentes e para a junção das informações na realização de tarefas específicas.

Tabela 1 – Listagem com as diferentes utilizações do CDS para o fornecimento de informações relevantes à beira do leito.

	Tipo	Descrição	Aplicabilidade
1.	Base de conhecimento sintetizada para responder as perguntas mais freqüentes	Utiliza referências eletrônicas indexadas por sintomas e condições	Fornece respostas rápidas aos problemas que freqüentemente aparecem na prática
2.	Base de conhecimento ligada às tarefas específicas	Une informações de referências às tarefas específicas, como a prescrição médica	Fornece acesso imediato às informações relevantes demandadas para realizar uma tarefa

Fonte: Metzger J, Macdonald K. Clinical Decision Support for the Independent Physician Practice. California HealthCare Foundation. 2002.

Quanto à assistência no gerenciamento individual de pacientes, encontra-se uma grande variedade de ferramentas com diversos propósitos, vide Tabela 2. Como por exemplo, a auto-avaliação de pacientes, o cálculo automático de fórmulas, os gráficos com indicadores temporais relevantes para a condição do paciente, o resumo das informações do paciente com seus problemas assinalados, a checagem de medicamentos, os conjuntos de ordens médicas ou os modelos de documentos gerados automaticamente.

Tabela 2 – Listagem com as diferentes utilizações do CDS para o gerenciamento individual de pacientes.

	Tipo	Descrição	Aplicabilidade
1.	Auto-avaliação do paciente	Resume e sintetiza as informações sobre o estado e o histórico do paciente	Fornece um resumo de informações do paciente que poderiam ser perdidas durante a coleta de informações
2.	Cálculo clínico	Cálculo de fórmulas e algoritmos presentes na prática clínica	Auxilia em recordar fórmulas e algoritmos e realizar cálculos complexos
3.	Folhas eletrônicas do fluxo dos indicadores temporais de uma doença	Gráficos ou tabelas representando o gerenciamento dos indicadores do estado de uma doença em relação ao tempo	Fornece fácil atualização do histórico e do estado atual do paciente
4.	Resumo das informações do paciente com os problemas assinalados	Inclusão de informações sobre doenças crônicas no cabeçalho do documento, no resumo do estado do paciente ou em ambos	Relembra os profissionais de saúde que o paciente está sob um estado crônico.
5.	Checagem de medicamentos	Checagem de medicamentos com exibição de mensagem	Adverte de possíveis contra-indicações ou problemas de dosagens com os medicamentos
6.	Conjuntos de ordens	Conjunto organizado de diagnóstico ou tratamentos para uma doença específica	Fornece um meio rápido de prescrever conjuntos recomendados de testes, medicações, etc.
7.	Modelo de documentação	Modelos estruturados para capturar problemas, identificando tópicos relevantes e entrada de texto livre apropriada	Guia a entrada de dados, através de tópicos e observações relacionadas e permite completude dos dados via texto livre

Fonte: Metzger J, Macdonald K. Clinical Decision Support for the Independent Physician Practice. California HealthCare Foundation. 2002.

Por fim, em relação à aplicação de recomendações de saúde sobre uma amostra de pacientes, Tabela 3, evidencia-se o uso dessas ferramentas de CDS para o registro de doenças ou o rastreamento de pacientes sob certos cuidados, o gerenciamento de lembretes para o bem-estar do paciente ou a listagem de paciente com intervenções a serem feitas.

Tabela 3-Listagem com as diferentes utilizações do CDS para a aplicação de recomendações de saúde sobre uma amostra de pacientes.

	Tipo	Descrição	Aplicabilidade
1.	Registro de doenças ou rastreamento de pacientes	Disponibiliza informações sobre as orientações de intervenções (realizadas, a realizar, resultados)	Fornece informações atualizadas sobre o cumprimento das recomendações dos pacientes sob certo estado
2.	Gerenciamento de lembretes de bem-estar ou doença	Mostra as orientações de intervenções atrasadas, através de mensagem ou no resumo do estado do paciente	Avisa sobre possíveis falhas na prestação de saúde com base nas orientações de idade ou doença
3.	Listagem dos pacientes com intervenções para serem feitas	Relata quais pacientes estão fora das orientações devido a intervenções atrasadas ou ao estado clínico	Identifica possível pacientes com falhas na prestação de saúde que não estão de acordo com as orientações

Fonte: Metzger J, Macdonald K. Clinical Decision Support for the Independent Physician Practice. California HealthCare Foundation. 2002.

2.3.4 Estrutura do CDS

O objetivo das aplicações de CDS é interagir com o usuário, auxiliando no processo de tomada de decisão para agregar valor no atendimento em saúde, deixando, entretanto, a tomada de decisão em si, sob responsabilidade do profissional de saúde. Mas para que isto ocorra, é necessário concentrar esforços nos aspectos de projeto, facilitando sua construção e viabilizando sua disseminação.

Diante disso, Greenes (2006) vislumbrou que um modelo de componentes conceituais e de suas interações poderia tornar mais clara as características e os relacionamentos entre cada um dos componentes, visto que geralmente eles se permeiam, sendo confundidos uns com os outros durante a implementação. Então, para entender melhor a natureza do CDS, ele teve como ponto de partida as tarefas realizadas durante um apoio à decisão clínica, isto é:

1. Inicialização ou chamada a partir de algum processo do ambiente do sistema de CDS;
2. Obtenção dos dados, fornecidos pelo usuário, recuperados do RES ou proveniente da entidade invocadora do sistema de CDS;
3. Utilização do conhecimento, através de regras, algoritmos ou relações semânticas, sejam locais ou recuperadas de uma base de conhecimento;

Fonte: Greenes RA. Features of Computer-based Clinical Decision Support. In: Greenes RA. Clinical Decision Support: The Road Ahead. Boston: Academic Press; 2006.

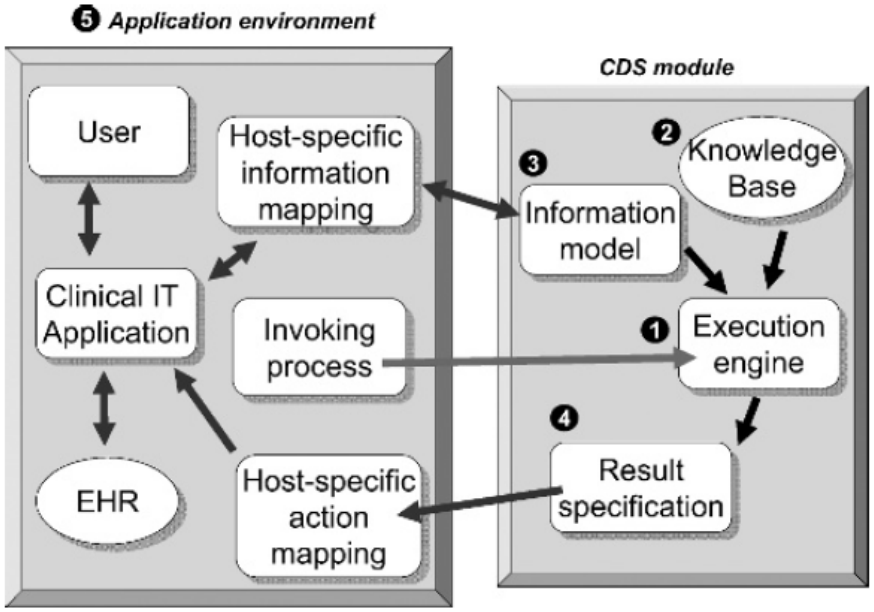

Figura 2.1 - Modelo de Greenes dos componentes conceituais e suas interações.

4. Transformação dos parâmetros de entrada e do conhecimento, utilizando algum tipo de modelo de decisão para gerar um resultado específico para o paciente;
5. Apresentação do resultado, que geralmente é uma recomendação de ação, por meio do ambiente da aplicação.

Utilizando essa seqüência de tarefas, ele estabeleceu cinco componentes conceituais, junto de suas interações, que podem ser vistos mais detalhadamente na Figura 2.1. Estes componentes estão sempre presentes no CDS, seja de forma implícita ou explícita, independente do propósito e da complexidade das suas funcionalidades.

O primeiro deles é o mecanismo de execução (*execution engine*) que realiza um processo computacional com o objetivo de avaliar uma entrada e produzir uma saída. Esse mecanismo, geralmente, está associado a um modelo de decisão (*decision model*), a exemplo de expressões lógicas, algoritmos ou grupos de associações de elementos, que indicam como os dados de entradas devem ser avaliados. Apesar de estar associados, sua separação traz como benefícios a flexibilidade e a portabilidade, uma vez que, permitiria que o mecanismo de execução fosse reimplementado em diferentes plataformas independente das demais partes do CDS.

O componente de base de conhecimento (*knowledge base*) representa o conjunto de informação que alimenta o modelo de decisão. Podendo essa base ser estruturada ou não, variando de acordo com o propósito desejado. A existência da base de conheci-

mento desassociada do mecanismo de execução garante uma série de vantagens, como a utilização do mecanismo de execução em diversas bases similares, gerência independente, maior facilidade e transparência na sua gestão e possibilidade de compartilhamento e disseminação, através de um formato padrão.

O modelo de informação (*information model*) refere-se à especificação dos diferentes tipos de informações que o modelo computacional de CDS irá precisar. Sua especificação formal fornece portabilidade ao CDS, já que, um mesmo recurso pode ser utilizado em diversas configurações, como por exemplo, de forma interativa com o usuário, recuperando dados do RES ou em várias plataformas diferentes. É importante que na sua especificação não contenha apenas o seu formato, mas também, a taxonomia ou esquema de codificação para sua representação e englobe informações básicas dos elementos de dados, como unidades, métodos de obtenção, janela de tempo, etc. Por questões de portabilidade, deve-se utilizar um modelo de informação padronizado, a exemplo do HL7 v.3 RIM, e para interoperabilidade entre diferentes sistemas, faz-se necessário um mapeamento entre as diferentes representações de informações.

Para lidar com os diversos tipos de resultados gerados dinamicamente pelo mecanismo de execução, foi previsto o componente de especificação de resultado (*result especification*). Isto se dá, através de um mapeamento dos diferentes tipos de resultados que se deseja executar com as ações ou resultados suportados pelo ambiente da aplicação, como por exemplo, um resultado calculado, uma ação recomendada ou um relatório formatado. A separação desse componente dos demais favorece a portabilidade, permitindo que o sistema de CDS possa ser utilizado por diferentes dispositivos em diferentes plataformas.

Por fim, o ambiente da aplicação (*application environment*) é responsável pela comunicação do CDS com o usuário, seja por diálogos interativos, recuperando dados do RES ou apresentando um resultado do CDS. Além de trocar informações clínicas, dados contextuais descrevendo a configuração da aplicação, o usuário, ou o propósito também podem ser trocados. O sucesso do CDS depende do nível de integração com as demais aplicações, pois integrações muito acopladas limitam a portabilidade e o reuso dos sistemas de CDS.

2.3.5 Impacto do CDS

Para mensurar, de fato, o quão importante o CDS pode ser para a qualidade do atendimento em saúde, deve-se analisar quais aspectos que compõem sua qualidade e como estes são influenciados pelo CDS. Segundo Donabedian (2005), a qualidade do atendi-

mento em saúde compreende a estrutura, o processo e os resultados da prestação de saúde, podendo a estrutura ser decomposta, segundo Carayon et al. (2006), em pessoas, organização, tecnologias, tarefas e ambiente.

Apesar de todos esses elementos serem relevantes na utilização do CDS, apenas alguns estudos usam métodos sistemáticos, como o *randomized controlled trial* (RCT), para avaliar a eficácia do CDS. E ainda, a maioria das pesquisas examina somente os efeitos do CDS no processo de prestação de saúde, com foco no auxílio à tomada de decisão, sem ponderar sobre os resultados ou a estrutura (BERNER, 2009).

Apesar de limitados, esses estudos revelam aspectos relevantes. Estudos realizados com alertas e lembretes mostraram que eles podem alterar as ações e decisões dos profissionais de saúde, reduzindo os erros médicos, além de promover a triagem preventiva e uso de recomendações baseadas em evidencias nas prescrições médicas (TROWBRIDGE; WEINGARTEN, 2001). Entretanto, resultados de estudos sobre a eficácia do uso do CDS para evitar eventos adversos de medicamentos (AMMENWERTH *et al.*, 2008) , em casos de trombose venosa profunda (KUCHER *et al.*, 2005) e diabetes (MEIGS, 2003), revelaram variância entre nenhum efeito positivo e efeitos negativos.

Alguns desses resultados podem ser atribuídos a problemas metodológicos na realização dos estudos, ao não cumprimento por parte dos pacientes das ordens médicas recomendações pelo CDS ou, até mesmo, a rejeição pelos profissionais de saúde de recomendações corretas do CDS. Além disso, a maioria dos estudos recai sobre sistemas de CDS que geram recomendações, sugerindo aos profissionais de saúde mudanças em suas ações, tal como ocorre com o uso de alertas. Ao invés de mostrar alternativas para as suas ações, como por exemplo, através de conjuntos de ordens médicas (*order sets*) (BERNER, 2009). Isto corrobora com Tierney (2003), que percebeu a existência de uma maior dificuldade dos usuários em mudarem seus planos de ações do que terem seus planos relembrados durante a realização de uma determinada tarefa.

Outro aspecto relevante é a eficiência das recomendações feitas automaticamente para os profissionais de saúde. Pois, quando o usuário do CDS requisita as informações, a probabilidade de utilizá-las é maior do quando elas são apresentadas automaticamente, entretanto essa busca por informações é muito rara (TAMBLYN, 2008). Tentativas de tornar os alertas menos intrusivos e mais discretos foram feitas, mas eles acabaram não atraindo a atenção dos profissionais de saúde e não contribuindo, assim, com a mudança no comportamento deles (ROSENBLOOM, 2005). Além disso, a integração do CDS com o fluxo de trabalho também pode influenciar na sua usabilidade,

uma vez que quanto maior for sua adequação ao fluxo de trabalho, mais intuitivo e prático será para os profissionais de saúde.

Uma abordagem que revelou melhoria na utilização dos alertas foi através do uso de alertas categorizados (*tiered alerts*) (PATERNO, 2009). Este se caracteriza por segregar os alertas em diferentes níveis de acordo com uma prioridade específica, permitindo assim ignorar os alertas com baixa prioridade, enquanto que os de alta prioridade devem ser tratados obrigatoriamente. Esse tipo de alerta é interessante, pois se adapta a realidade dos registros eletrônicos, onde a maioria dos alertas gerados são genéricos e sem muita importância, impactando muito pouco na qualidade do atendimento em saúde, ao passo que, a minoria dos alertas gerados são extremamente relevantes e podem fazer a diferença durante a prestação de saúde, devendo, portanto, serem obrigatoriamente observados (BERNER, 2009).

Embora haja poucos estudos sobre as implicações do CDS na estrutura das organizações de saúde, Berlin, Sorani e Sim (2006) concluíram, em seu estudo, que é necessário um aumento no número de profissionais para implementar o CDS, além de pessoas extras para manter sua base de conhecimento. Outros estudos analisaram aspectos como o tempo e custo do CDS. Apkon (2005) notou que dependendo de como o CDS foi projetado e integrado ao RES, ele pode demandar mais tempo dos profissionais de saúde durante a prestação de saúde, podendo desta forma, afetar no custo e na produtividade. Enquanto isso, Field et al. (2008) concluiu que metade do custo durante o desenvolvimento do CDS estava relacionado com a revisão do conteúdo pelos profissionais de saúde. Embora, este valor pudesse ser diminuído, através do uso de base de dados já existentes, mas mesmo assim, algum esforço seria necessário para adequar essas bases de dados às características locais. Todos esses aspectos devem ser levados em consideração durante o projeto, a implementação, a manutenção e a avaliação para, de fato, medir o tempo e custo demandados para a obtenção de melhorias na qualidade do atendimento em saúde.

2.4 Agentes

Dentro do contexto de saúde, a utilização de sistemas de apoio à decisão baseados em agentes para auxiliar no processo de tomada de decisão tem se mostrado bastante relevante. Pois permite que os profissionais de saúde rapidamente tenham acesso às informações, da maneira mais adequada, para facilitar o diagnóstico e tratamento de enfermidades (FOSTER *et al.*, 2005).

Entretanto, devido ao seu uso diverso, o termo agente não possui uma definição clara e precisa, variando em complexidade e escopo, dependendo da sua utilização em cada uma das soluções. Como exemplo, Russell e Norvig (1995) definem um agente como algo que percebe seu ambiente através de sensores e atua sobre o ambiente através de atuadores. A IBM afirma que agentes inteligentes são entidades de software que realizam um conjunto de operações em nome de um usuário ou de outro programa, com certo nível de independência ou autonomia e utilizam algum conhecimento ou representação dos objetivos ou vontades do usuário (GILBERT *et al.*,1995). Já, Coen (1994) diz que agentes de software são programas com objetivo de dialogar, negociar e coordenar a transferência de informação.

Diante dessas variadas definições, abordando diferentes pontos de vista sobre os agentes, Franklin e Graesser (1996) criaram uma definição que tenta capturar a essência dos agentes e determinar uma classificação para eles. Após analisar as demais definições, os autores concluíram que um agente autônomo é um sistema localizado dentro de um ambiente ou faz parte dele e que sente e atua sobre o ambiente, ao longo do tempo, em busca de seus objetivos para afetar o que sente no futuro. Além disso, eles elaboraram uma lista de propriedades que descrevem os diferentes tipos de agentes, no qual qualquer agente, segundo a definição deles, deve conter no mínimo as seguintes propriedades: reativo (reagir às mudanças no ambiente), autônomo (controla suas ações), temporalmente contínuo (existência prolongada), orientado a metas (não age somente em resposta ao ambiente).

Diferentemente de Franklin e Graesser (1996), Russell e Norvig (1995) categorizaram os agentes inteligentes em cinco classes de acordo com a capacidade e o nível de inteligência de cada um deles. Sendo o mais simples de todos, o agente reativo, que apenas age de acordo com sua percepção do ambiente, utilizando para isso, regras do tipo "condição-ação". Ele possui a vantagem de ser bastante simples, modular e eficiente, mas sua inteligência é limitada, tem pouca autonomia e não pode armazenar uma seqüência perceptiva. O agente baseado em modelos com estado interno guarda seu estado interno para escolher suas ações e utiliza num modelo do ambiente. Como desvantagem, ele possui pouca autonomia e necessita saber como o ambiente funciona. Já o agente cognitivo baseado em objetivo utiliza as informações do modelo do ambiente junto dos seus objetivos para decidir qual ação realizar. Este agente apesar de ser mais complexo e menos eficiente é mais flexível e autônomo que os anteriores. Por sua vez, o agente otimizador utiliza, além do modelo e dos objetivos, uma função de utilidade para selecionar a ação mais útil para ao agente. Com isso, ele possui a vantagem de lidar com

objetivos conflitantes, mas ainda não possui adaptabilidade. E por fim, o agente com aprendizagem consegue avaliar suas ações para ponderar sua reação nas próximas vezes, permitindo assim iniciar em um ambiente desconhecido e se tornar mais competente do que inicialmente era.

Assim, dependendo do tipo específico de cada agente, ele pode ser usado para um determinado propósito, a exemplo de agentes que buscam informações sobre o paciente para tentar descobrir uma relação de parentesco (SWAIN e KIM, 2004); que cooperam para escolher o tratamento de um paciente (LANZOLA *et al.*, 1999); que monitoram e obtêm um diagnóstico do quadro clínico do paciente (LARSSAN E HAYES-ROTH, 1998); que colhem informações do paciente em diferentes hospitais (MEUNIER, 1999), entre outros tantos.

2.5 Ontologia

Os sistemas de CDS são bastante dependentes das informações clínicas, pois necessitam entendê-las para poder realizar suas funções. Por esse motivo, as informações precisam ser estruturadas e codificadas, possibilitando assim serem executadas de forma precisa por a uma lógica de decisão. O texto livre, além de impreciso, é ambíguo, tornando inviável sua utilização pelos mecanismos de raciocínio computacional.

Uma forma computacionalmente mais vantajosa de codificar as informações é através da representação hierárquica e sistemática dos conceitos e dos seus relacionamentos. A essa representação dos conceitos associados, através de um modelo conceitual, dá-se o nome de ontologia (HUFF, 2006).

A ontologia é uma especificação de uma conceituação, uma descrição dos conceitos e relacionamentos que podem existir para um agente ou uma comunidade de agentes (GRUBER, 2010). Podendo ser usada por pessoas ou computadores para três propósitos básicos: comunicação; inferência e reuso computacional; e gerenciamento de conhecimento. A comunicação é alcançada através da definição padronizada e clara dos conceitos disponíveis para todos os usuários da ontologia. Como o conhecimento contido na ontologia é codificado, ele se torna fácil de ser entendido e usado. E, por fim, os mecanismos de criação das ontologias podem ser facilmente estendidos para permitirem a manutenção simplificada do conhecimento (GRUNINGER; LEE, 2002).

Por esses motivos, a ontologia vem sendo bastante utilizada para a representação não ambígua dos complexos conceitos do domínio de saúde, favorecendo assim o reuso e o compartilhamento das informações clínicas dos pacientes (PISANELLY, 2004). Alguns exemplos de ontologias criadas para a representação de terminologias em

saúde são: GALEN, MeSH (*Medical Subject Headings*), SNOMED-CT (*Systematized Nomenclature of Medicine, ClinicalTerms*),UMLS (*Unified Medical Language System*) and ON9.

Para tornar possível a codificação desses conceitos, diversas linguagens foram criadas, linguagens estas que na sua grande maioria são declarativas baseadas em lógica de primeira ordem ou em lógica descritivas, como por exemplo: KIF, RDF (*Resource Description Framework*), KL-ONE, DAML+OIL e, a mais usada, OWL (*Web Ontology Language*).

2.6 O projeto OpenCTI

Tentando melhorar a qualidade da prestação de saúde na UTI do Hospital Universitário Lauro Wanderley, através dos benefícios com o uso do RES do paciente e de ontologias para a representação do conhecimento, surgiu o OpenCTI. O OpenCTI é um projeto de um software livre, financiado pela FINEP e em estado de desenvolvimento que utiliza da telemedicina para auxiliar na decisão médica de pacientes internados em UTIs de hospitais distantes de centros de referências que, por ventura, não possuam médicos especialistas em medicina intensiva. Assim, através da gestão das informações clínicas e da colaboração entre equipes de saúde geograficamente separadas, será possível fornecer, pelos especialistas localizados nos grandes centros, uma segunda opinião médica, melhorando dessa maneira a qualidade da prestação de saúde.

Para que essa segunda opinião médica possa ser a mais rápida, segura e plena possível, faz-se necessária a utilização do RES, facilitando a acessibilidade e segurança das informações clínicas e fomentando a agilidade na prestação de saúde. Além disso, deve-se usufruir de uma representação estruturada dos dados, como por exemplo, através do uso de ontologias, garantindo, desta maneira, uma conformidade das informações clínicas trocadas entre os profissionais de saúde.

Com base nessa premissa, o openCTI faz uso da ontologia não apenas para a representação dos conceitos biomédicos, mas sim, de todo o domínio da aplicação, isto é, dos documentos que contém as informações clínicas dos pacientes. Estes documentos podem ser representados independentemente dos conceitos biomédicos, pois são ortogonais a eles. Assim, pode-se representar, semanticamente, os documentos do prontuário do paciente, facilitando sua expressividade e compreensão, e ainda, possibilitar uma maior mutabilidade do sistema, uma vez que, o domínio de saúde é bastante dinâmico, estando sempre evoluindo, através do surgimento de novos tratamentos e medicamentos ou da obrigatoriedade de preenchimento de novas informações.

Entretanto, com essa abordagem orientada à ontologia, a aplicação passa a ser criada dinamicamente, pois, agora, depende dos aspectos semânticos definidos na ontologia. Assim, busca-se uma solução flexível que seja viavelmente adaptada para a cultura organizacional das instituições de saúde e que ao mesmo tempo de suporte a integração destas com demais instituições.

Levando-se isso em consideração, podemos ter uma idéia simplificada da arquitetura do OpenCTI, conforme mostra a Figura 2.2. Nela, podemos perceber a existência de uma camada de metadados, responsável por armazenar e tratar os dados das ontologias, uma camada de persistência, que armazena os dados tradicionais e dados esparsos, oriundos do domínio biomédico, uma camada de domínio que representa a lógica de negócio da aplicação, uma camada de aplicação, que lida com aspectos de segurança e, por fim, uma camada de apresentação, responsável por gerar a interface dinamicamente com base na ontologia dos documentos de saúde.

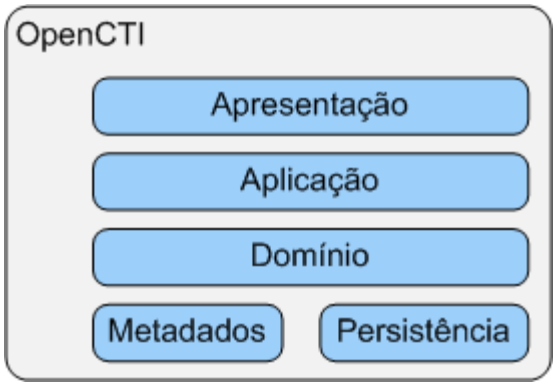

Figura 2.2 - Arquitetura simplificada do OpenCTI.

Para entender cada uma das camadas supracitadas, facilitando assim a compreensão do objetivo da solução, bem como, do seu potencial de utilização, segue abaixo uma descrição mais detalhada das camadas do OpenCTI. Vale ressaltar que para aumentar a produtividade, acelerando e facilitando o desenvolvimento da aplicação, foram utilizados na implementação, componentes EJB associados ao servidor de aplicação web JBoss Seam.

2.6.1 Camada de Metadados

A camada de metadados é o pilar central da aplicação, sobre a qual todo o sistema foi construído. Ela é responsável por fornecer um meio de manipulação das informações contidas nas ontologias, além de salvaguardá-las. Essas informações ontológicas têm duas naturezas distintas: uma representa os conceitos biomédicos, que caracterizam

enfermidades, partes do corpo, intensidade, entre outras características de saúde, e a outra representa os documentos de saúde que compõem o RES do paciente, isto é, os diversos campos, subcampos, estruturas e componentes utilizados para a registro das informações de saúde. Desta forma, ambas as informações são ortogonais e distintas, podendo ser modeladas separadamente.

Para a codificação dessas diferentes ontologias, o OpenCTI utilizou a linguagem OWL na modelagem de uma ontologia de documentos de saúde, *LightDocument.owl*, e uma ontologia de conceitos de biomédicos, *LightBiomedicalConcept.owl*.

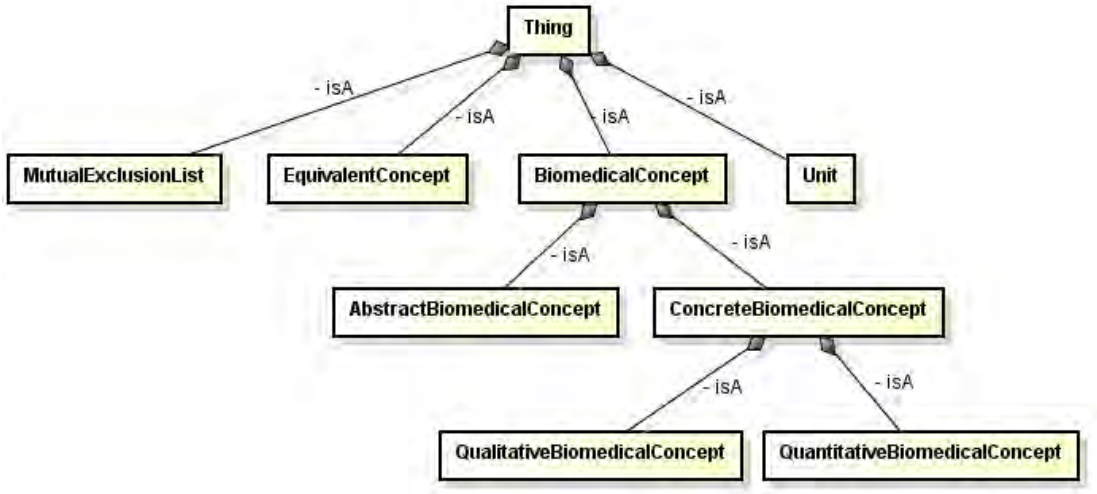

Figura 2.3 - Estrutura da ontologia de conceitos biomédicos do OpenCTI

A estrutura da ontologia de conceitos biomédicos pode ser vista, através da Figura 2.3, que de acordo com os princípios da linguagem OWL, obriga a todos os elementos a serem subtipos de *Thing*. A partir daí, temos os conceitos biomédicos (*BiomedicalConcept*) que podem ser de duas formas distintas, conceitos biomédicos abstratos (*AbstractBiomedicalConcept*) ou conceitos biomédicos concretos (*ConcreteBiomedicalConcept*). Os conceitos abstratos não possuem valores associados, eles são agrupamentos semânticos de conceitos biomédicos concretos. Em contrapartida, os conceitos concretos representam realmente algo substancial e podem ser subdivididos em duas categorias, qualitativos (*QualitativeConcreteBiomedicalConcept*) ou quantitativos (*QuantitativeConcreteBiomedicalConcept*). Os conceitos concretos qualitativos representam características subjetivas e discretas, geralmente associadas às observações clínicas, por meio de conceitos que indicam intensidade ou qualidade, como "forte", "ausente", "bom", entre outros. Por outro lado, os conceitos quantitativos representam conceitos numéricos e em geral estão associados a uma unidade de medida (*Unit*). Apesar dos conceitos quantitativos estarem associados às unidades (*Unit*), estas não dependem daqueles para existirem, podendo ser

representada isoladamente como um subtipo de *Thing*. Além deles, para representar certos conceitos que são equivalentes a outros ou fazem parte de uma lista de exclusão mútua, foram criados respectivamente as estruturas, *EquivalentConcept* e *MutualExclusionList*, enriquecendo assim ainda mais o modelo.

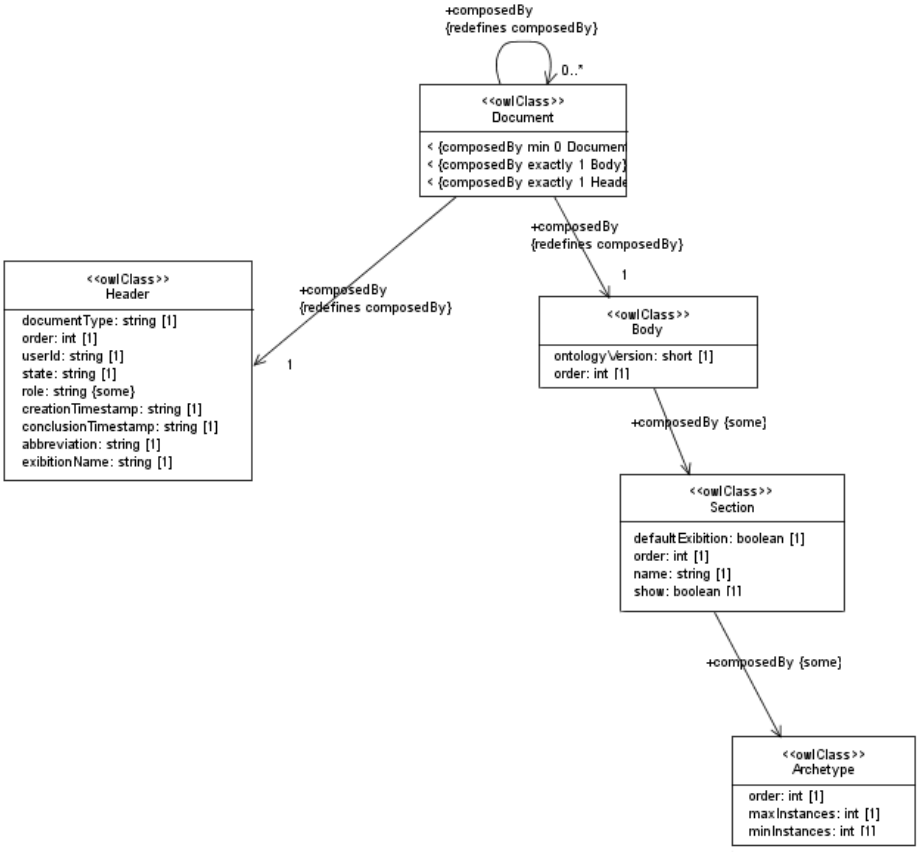

Figura 2.4 - Diagrama de classes da estrutura dos documentos de saúde.

Já a ontologia de documentos de saúde segue o diagrama de classes da Figura 2.4, no qual é possível perceber os elementos estruturais de um documento genérico. Desta forma, um documento de saúde pode ser simples, isto é, possuir apenas um cabeçalho que o descreve e um corpo com seu conteúdo, ou ser composto, conter diversos outros documentos associados a este documento. Cada documento deve possuir no cabeçalho uma série de informações sobre o próprio documento, como o seu tipo, ordem, estado, número identificador e o papel do usuário que o criou, instante de tempo da sua criação e conclusão, seu nome de exibição e nome abreviado. Cada documento deve também conter um corpo, que vai possuir uma ordem, uma versão da ontologia e pode conter várias seções. Cada seção, por sua vez, vai ter uma ordem, um valor de exibição padrão, um nome, um valor de exibição e pode possuir vários arquétipos. Cada arquétipo representa um conjunto de conceitos biomédicos agrupados e deve conter uma or-

dem e um valor máximo e mínimo de instancias. A presença dos arquétipos tem papel vital no OpenCTI, uma vez que, ele é o responsável por associar conceitos ortogonais, isto é, os conceitos biomédicos e dos documentos de saúde, permitindo assim uma maior facilidade na gerência e na modelagem desses conceitos sem prejuízo para o domínio da aplicação.

Além das informações codificadas nas ontologias, a camada de metadados usa uma API, OWL-API, para poder manipular essas informações e convertê-las para a orientação a objetos, tornando-as compreensíveis para o resto da aplicação.

2.6.2 Camada de Persistência

Como a aplicação é orientada à ontologia, a camada de persistência teve que ser projetada para dar suporte a toda essa dinâmica de informações na qual os conceitos podem ser facilmente criados ou removidos. Além disso, os constantes avanços da área de saúde fazem com que o conjunto de dados clínicos, por si só, seja altamente dinâmico, volátil e esparso (JOHNSON, 1996). E em contrapartida, também há no RES informações tradicionais, isto é, dados demográficos do paciente, como nome, sobrenome, logradouro, que necessitam obrigatoriamente estarem presentes. Para lidar com essas diferentes facetas, foi escolhida uma abordagem mista, com a utilização da modelagem relacional tradicional para representar as informações triviais e da modelagem EAV (*Entity Atributte Value*), desenvolvido por DINU e colaboradores (2007), para a representação dos dados clínicos.

O EAV também utiliza o modelo relacional, entretanto se destaca por separar os dados dos seus metadados, de maneira que, os dados fiquem em uma única tabela, enquanto que seus metadados em outras, associados através de referências. Com isso, as informações clínicas dos pacientes são armazenadas de forma mais eficiente, visto que, apenas as informações preenchidas são mantidas, evitando que o banco fique esparso.

Com base nisso, podemos ver na Figura 2.5, o esquema do bando de dados do OpenCTI. Nela, as informações demográficas dos pacientes estão contidas na entidade *Patient*, as informações sobre o leito, o prontuário e a ficha de saúde, estão representados nas entidades *Bed*, *MedicalRecord* e *Event*, respectivamente, e os dados clínicos estão indicados pelas entidades *EavMain*, *EavType*, *EavInteger*, *EavDouble*, *EavBoolean*, *EavString*, *EavDate*, sendo, o primeiro para conter os valores dos dados, o segundo para representar os diversos tipos possíveis dos dados, e os demais para representar cada um dos tipos dos dados.

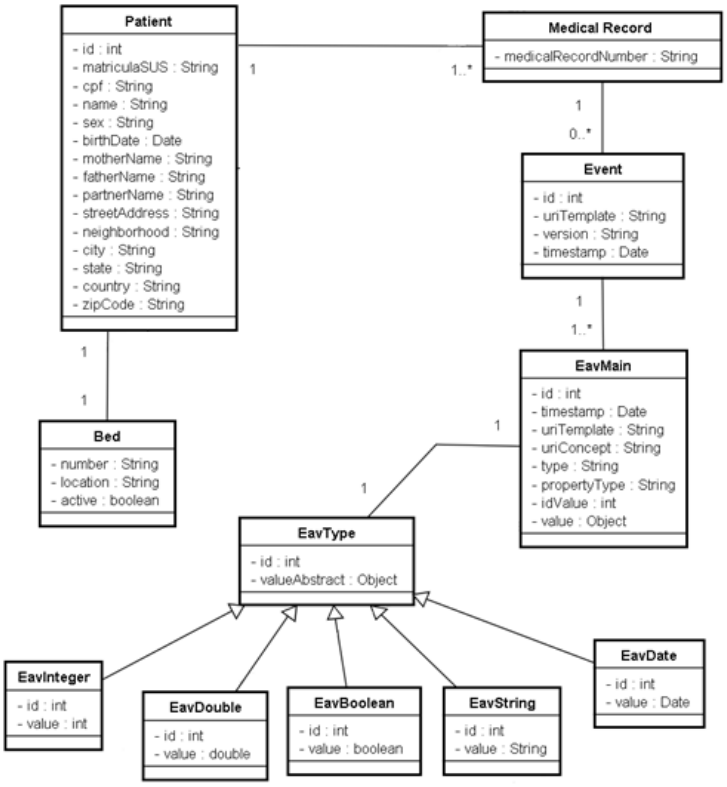

Figura 2.5 - Diagrama de classes do banco de dados EAV do OpenCTI

Através desse esquema pode-se, então, tirar proveito de maneira eficiente da abordagem relacional para lidar com dados tradicionais (que contém as informações sobre o paciente), através do modelo relacional tradicional, e com os dados clínicos, através da abordagem EAV para segregar os dados dos seus metadados.

2.6.3 Camada de Domínio

Conforme dito anteriormente, os dados clínicos são ortogonais aos documentos de saúde, isto é, seus conceitos são independentes. Entretanto, os documentos utilizam os conceitos de saúde para armazenar informações clínicas dos pacientes.

Para que a aplicação possa representar os mais diversos documentos de saúde, a partir das informações contidas na ontologia, foi criada uma estrutura que abstrai para o resto da aplicação a utilização da ontologia, fornecendo uma maneira simples, padronizada e intuitiva de manipulação desses objetos. Assim, uma estrutura genérica que representa a ontologia foi empregada, como pode ser visto na Figura 2.6, através de uma classe que representa qualquer conceito na ontologia (*Thing*) e que tem associada a ela relacionamentos (*Relationship*) e propriedades (*DataProperty*) que a descrevem. E utilizando esse esquema foi possível elaborar uma estrutura genérica para a representação

dos documentos de saúde (*HealthDocument*), capaz de se adaptar a documentos de saúde distintos, bem como, dos seus conceitos de saúde associados.

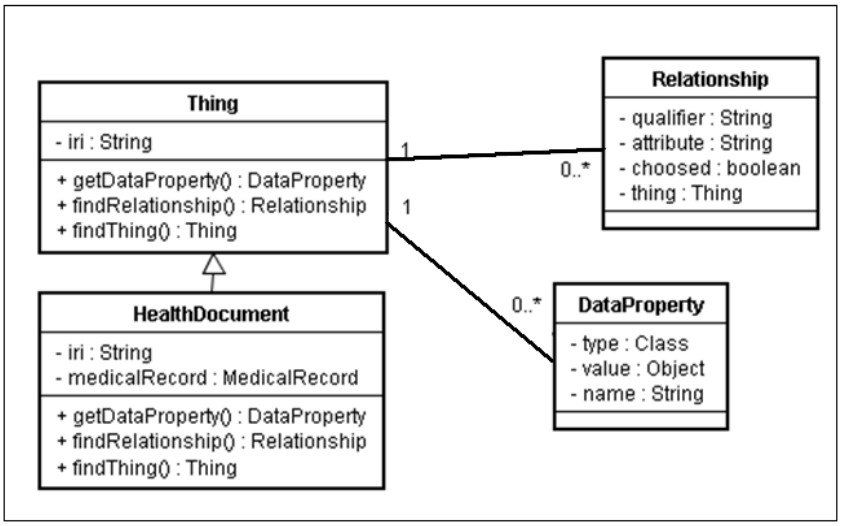

Figura 2.6 - Diagrama de classes da representação genérica da ontologia no OpenCTI

Essa solução possibilita a interligação na camada de domínio das informações de saúde do paciente, isto é, os dados clínicos que podem ser facilmente modificados, com as informações demográficas do paciente, utilizando para isso a classe *MedicalRecord*, que representa o RES de cada paciente.

2.6.4 Camada de Aplicação

No contexto de um hospital de saúde, o controle e a gerência do acesso às informações de saúde dos pacientes, assim como, seu armazenamento, não é apenas importante e, sim, indispensável, conforme consta na resolução nº 1821/2007 do Conselho Federal de Medicina (CFM), mais especificamente no art. 3º:

> "Autorizar o uso de sistemas informatizados para a guarda e manuseio de prontuários de pacientes e para a troca de informação identificada em saúde, eliminando a obrigatoriedade do registro em papel, desde que esses sistemas atendam integralmente aos requisitos do "Nível de garantia de segurança 2 (NGS2)", estabelecidos no Manual de Certificação para Sistemas de Registro Eletrônico em Saúde;"

O Manual de Certificação para Sistemas de Registro Eletrônico em Saúde, supracitado, é um documento elaborado pelo CFM em conjunto com a Sociedade Brasileira de Informática em Saúde (SBIS) que visa padronizar diferentes patamares de segurança, dando condições para o processo de certificação de sistemas informatizados em saúde.

Desta forma, a camada de aplicação lida, entre outras coisas, com os aspectos referentes à autenticação e autorização do acesso aos recursos do sistema. Para tal, faz uso do MACA, um *Middleware* de Autorização e Controle de Autenticação, desenvolvido por MOTTA (2004).

2.6.5 Camada de Apresentação

Para lidar com a fluidez dos dados de domínio, se faz necessária uma camada de apresentação que seja dinâmica. Mais ainda, para garantir o acesso a esses dados, independente da localização e tecnologia, esta camada deve ser flexível em relação as tecnologias e aos dispositivos que serão utilizados para acessar esses dados.

Conforme a solução elaborada por Duarte (2011), foi usado uma abordagem mista, onde um módulo dinâmico é o responsável por tratar os dados clínicos dos pacientes e um módulo estático é o responsável por lidar com os dados demográficos do paciente, bem como, os fluxos de navegação do usuário.

O modulo dinâmico utiliza uma ontologia de componentes genéricos que incorpora informações específicas do contexto do usuário para gerar a interface com o usuário dinamicamente em tempo de execução. Possibilita assim que os documentos de saúde sejam visíveis, independentemente da sua estrutura, o que facilita a customização desses documentos. E ainda, permite que esses documentos fiquem acessíveis em diversos dispositivos móveis, como *tablets* e *handhelds*. Em contrapartida, para o módulo estático, uma abordagem tradicional foi seguida com a criação das páginas manualmente, pois é sabida a estrutura dos dados de antemão.

2.7 Panorama Atual das Soluções de CDS

Desde o surgimento dos primeiros sistemas de informação em saúde, um vasto esforço tem sido empregado em criar soluções para melhorar a qualidade da prestação de saúde, por meio da geração automatizada de orientações, recomendações e/ou alertas. Com esse intuito, diversas soluções como Arden Syntax, EON, PRODIGY-3, PROforma, Asbru, GUIDE, Prestige, GLIF3 e Gello.

Dentre essas soluções, analisaremos mais detalhadamente duas delas: Arden Syntax e Gello. A primeira por ser bastante conhecida e consolidada, enquanto a segunda, por ser mais recente e se mostrar bastante promissora.

2.7.1 Arden Syntax

A Arden Syntax é uma gramática para representar e processar informações e recomendações clínicas de forma padronizada. Atualmente, é um padrão ANSI (*American National Standards Institute*), sendo mantido pela HL7.

Fonte: Jenders RA. Decision Rules and Expressions. In: Greenes RA. Clinical Decision Support: The Road Ahead. Boston: Academic Press; 2006

```
maintenance:
        title:          Screen for positive troponin I;;
        filename:       troponin;;
        version:        1.40;;
        institution:    World-Famous Medical Center;;
        author:         Robert A. Jenders, MD, MS (jenders@uda.edu);;
        specialist:     ;;
        date:           2005-08-15;;
        validation:     research;;

library:
        purpose:        Screen for evidence of recent myocardial infarction;;
        explanation:    Triggered by storage of troponin result.  Sends message
                            if result exceeds threshold;;
        keywords:       troponin; myocardial infarction;;
        citations:      ;;

knowledge:
        type:           data-driven;;
        data:
            troponin_storage := event {storage of troponin};

            /* get test result */
            tp := read last {select result from test_table where
                  test_code = 'TROPONIN-I'};

            threshold := 1.5;

            /* email for research log */
            email_dest := destination {'email', 'name'= "jenders@uda.edu"};
        ;;
        evoke:  troponin_storage;;

        logic:

          if (tp is not number) then conclude false;
          endif;

          if tp > threshold then conclude true;
          else conclude false;
          endif;

        ;;
        action:
          write "Patient may have suffered a myocardial infarction."   ||
            "Troponin I = " || tp || " at " time of troponin
          at email_dest;

          ;;
       urgency:      50;;
end:
```

Figura 2.7 – Exemplo de um MLM de triagem de paciente que recentemente tiveram infarto do miocárdio, através da análise da quantidade de tropanina.

Sua estrutura é formada por diversos arquivos, denominados MLM (*Medical Logic Modules*), contendo, dentro deles, a lógica e referências aos dados clínicos para a realização de tomadas de decisões de baixa complexidade. Cada MLM é organizado em três categorias: manutenção (*maintenance*), biblioteca (*library*) e conhecimento (*knowledge*). Cada uma deles possui um ou mais pares de atributo-valor que exprimem uma assertiva sobre o domínio do módulo. Dessa forma, conforme pode ser visto na Figura 2.7, que representa um módulo de triagem de paciente com infarto do miocárdio recente, as categorias de manutenção e biblioteca definem as meta-informações sobre o módulo, enquanto que a categoria de conhecimento descreve sua lógica e referências aos

dados clínicos. Dentro da categoria de conhecimento há: o atributo *data,* que é usado para representar os dados necessários dentro da lógica do módulo; o atributo *evoke*, que identifica as condições sobre as quais o módulo deve ser executado; o atributo *logic*, que contém cálculos e regras IF-THEN utilizadas para realizar alguma ação sobre os dados; o atributo *action*, que determina qual ação deve acontecer quando o resultado da lógica retorna o valor *true*; além de vários outros atributos.

Dois aspectos tornam o Arden Syntax muito importante. Primeiro, é considerado um padrão de fato, sendo suportado por diversos sistemas de informação clínicas. Segundo, é o seu modelo simplificado de representação do conhecimento que facilita seu entendimento e uso. Em contrapartida, algumas características limitam sua flexibilidade. Todas as referências aos dados locais do ambiente de utilização dependem da plataforma em uso, devendo ser descritos entre chaves na definição de cada valor do atributo *data*. Como seu modelo é simplificado, o Arden Syntax requer uma infra-estrutura substancial para poder ser posto em prática. E por fim, cada módulo seu define de forma explícita quando e como as mensagens de CDS devem ser construídas e enviadas ao usuário (JENDERS, 2006).

2.7.2 Gello

Gello é uma linguagem de expressão de *guidelines*[1] orientada a objetos para o auxílio à decisão clínica. Ela é o resultado do esforço do DSG (*Decision Systems Group*) em colaboração com o CDSTC (*HL7 Clinical Decision Support Technical Committee*) e visa fornecer um formato para codificação e manipulação de dados para a comunidade HL7 (Sordo *et al.*, 2003).

Ela pode ser usada na construção de consultas para extrair e manipular dados; na elaboração de expressões para inferir sobre os dados que podem ser usados na criação de alertas, lembretes ou *guidelines*; e na elaboração de expressões, fórmulas e consultas para outras aplicações. Para tanto, Gello utiliza um modelo de dados, chamado vMR (*virtual medical record*), que é um framework genérico orientado a objeto derivado do HL7 RIM no qual não há dependência de classes ou tabelas específicas. Possibilitando assim, a representação abstrata das entradas e saídas das informações clínicas que podem ser trocadas entre o mecanismo de CDS e os sistemas de informação em saúde. Isto faz com que ela funcione como uma interface padronizada para sistemas heterogê-

[1] Protocolos de conduta clínica com o intuito de guiar e orientar um procedimento médico.

neos de RES, permitindo o acesso a estruturas de dados em diferentes formatos com o mesmo código em Gello.

Atualmente, foi implementada a primeira versão do compilador para GELLO, utilizando GLIF (*Guideline Interchange Format*) na construção dos guidelines e das restrições (MEDICAL OBJETCS). Um exemplo possível de utilização dessa implementação para o mesmo caso da Figura 2.7 pode ser visto na Figura 2.8 abaixo.

Fonte: Jenders RA. Decision Rules and Expressions. In: Greenes RA. Clinical Decision Support: The Road Ahead. Boston: Academic Press; 2006

```
let lastTroponin: Observation = Observation→select(code=
    ("SNOMED-CT", "102683006")).sortedBy(effectiveTime.high).last()

let threshold : PhysicalQuantity =
    Factory.PhysicalQuantity( "1.5, ng/dl")

let threshold_for_osteodystrophy : int = 70

let myocardial_infarction :Boolean =    if lastCreatinine <> null and
    lastCreatine.value.greaterThan(threshold)
 then
    true
 else
    false
 Endif

if myocardial_infarction then
     whatever action or message
else
     whatever action or message
endif
```

Figura 2.8 - Exemplo de código em GELLO para a triagem de paciente que recentemente tiveram infarto do miocárdio, através da análise da quantidade de tropanina.

A sua grande diferença é a orientação a objetos e a abstração dos dados que permite sua utilização independentemente do modelo de dados escolhido. Entretanto, além dele ser mais complexo que o Arden Syntax e não possibilitar a representação de todos os *guidelines*, seu uso é recomendado apenas para estruturas de dados orientadas a objetos, não sendo indicado para outras estruturas de dados, como matrizes, por exemplo.

2.7.3 Soluções utilizando ontologia

Tentando usufruir das vantagens oriundas das ontologias, tais como, reuso, compartilhamento e manutenção, muitas soluções também usaram ontologias para criar mecanismos de CDS.

Além das diversas soluções, sua utilização também tem sido bastante variada. Bouamrane e colaboradores (2009) tiraram proveito da ontologia para a representação de conceitos do domínio, bem como definir as regras para a geração de protocolos de exames e recomendações em pacientes com risco pré-operatório. A Figura 2.9 mostra

um exemplo desse uso (utilizando a linguagem OWL e a ferramenta Protegè para sua manipulação) na recomendação de um exame de ECG (eletrocardiograma) em pacientes que possuam doença cardiovascular, tenham mais de 16 anos de idade e possuam grau de risco, de acordo com a ASA (*American Society of Anaesthesiologists*), igual a 2 e grau do procedimento cirúrgico igual a 1.

Fonte: Bouamrane M, Rector A, Hurrell M. Development of an ontology for a preoperative risk assessment clinical decision support system. 2009.

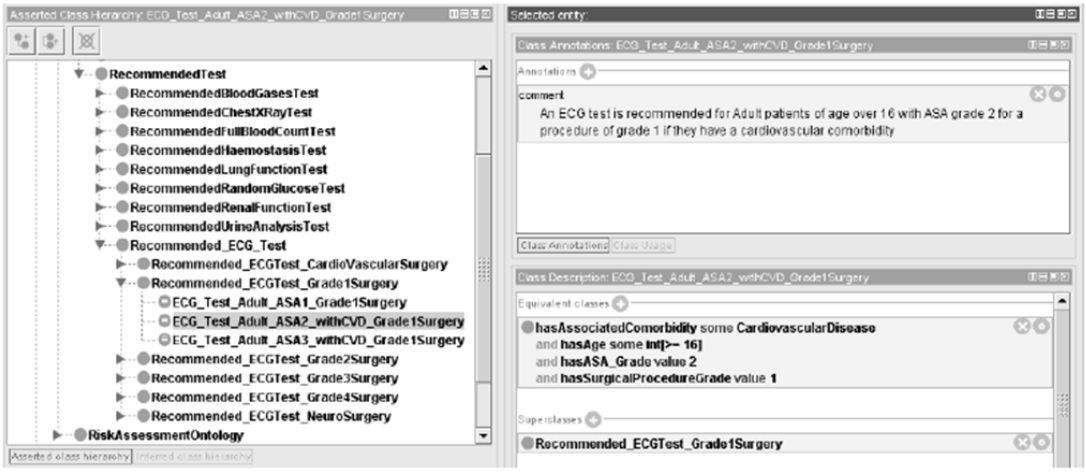

Figura 2.9 - Exemplo da utilização da ontologia na definição de uma regra para a recomendação de exame de ECG (Eletrocardiograma).

Por outro lado, Peleg e seus colegas (2008) utilizaram a ontologia para mapear *guidelines* de acordo com os RES de cada instituição de saúde. Dessa forma, desenvolveram um framework, *Knowledge-Data Ontological Mapper* (KDOM), que visa abstrair as discrepâncias entre diversas instituições de saúde, através da representação dos conceitos específicos de cada instituição por meio do KDOM, fomentando o compartilhamento dos *guidelines* entre as instituições.

2.7.4 Soluções comerciais

O CDS vem sendo usado para diversos fins, utilizando conceitos de domínio complexos que, por conseqüência, o torna bastante oneroso, visto que, em geral, processos de saúde demandam equipamentos, técnicas e profissionais de ponta. Assim, várias empresas criaram suas soluções comerciais para tentar tirar proveito desse nicho de mercado, tentando então oferecer subsídios para a resolução dos mais variados problemas.

Uma dessas soluções comerciais usadas na gestão de drogas é a Micromedex® Healthcare Series, que lida com aspectos de dosagem e identificação de drogas, intera-

ções entre drogas, toxicologia, entre outras funcionalidades. Uma visão geral desse sistema pode ser vista na Figura 2.10, onde um grupo de funcionalidades específicas pode ser visualizado em cada uma das abas do sistema.

Fonte: MICROMEDEX Healthcare Series. Mai. 2005. Disponível em: <http://naples.tju.edu/Ask/Help/handouts/micromedex.pdf>.

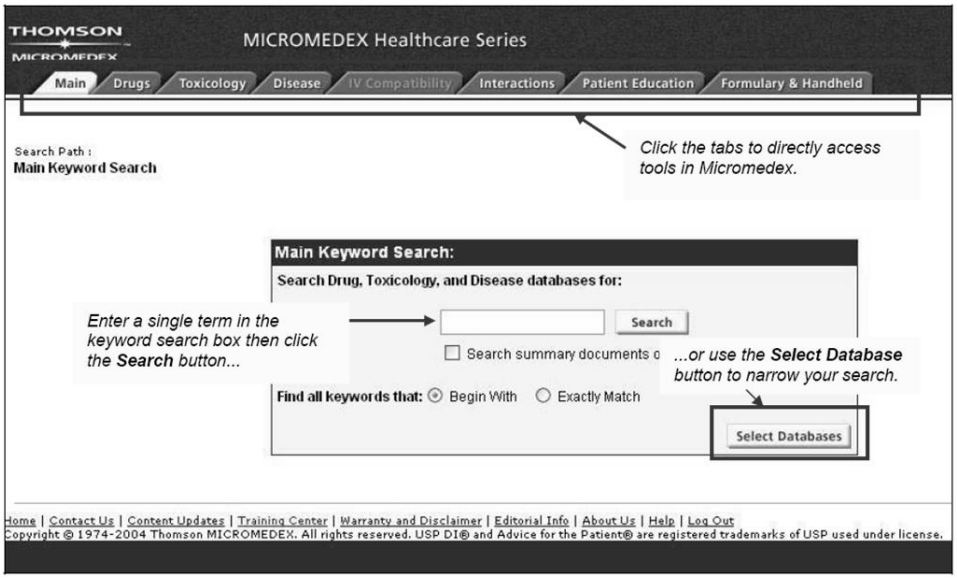

Figura 2.10 - Tela inicial do sistema da Micromedex®.

Uma outra solução comercial é o UpToDate Inc. que serve como um repositório eletrônico de informações clínicas para profissionais de saúde, pacientes e educadores, através da resposta a perguntas clínicas em diversas especialidades, tais como cuidado primário em adultos, endocrinologia, obstetria, pediatria, infectologia, entre outras. Fundada em 1989 a UpToDate atualmente fornece suas soluções em três formatos: Web, desktop e em PDAs.

A Zynx Health é mais uma solução comercial, que oferece soluções de CDS baseadas em evidência médica e em experiência clínica, bem como, plano de conduta médica, regras de decisão clínica, sugestões de valores, prognósticos e *guidelines* que visam melhorar a qualidade da prestação de saúde durante o uso do RES. Esta solução foi personalizada para diversas instituições e atualmente vem sendo utilizada por centenas de diferentes instituições de saúde.

2.8 Considerações finais

O CDS evoluiu bastante, principalmente, nesses últimos anos, passando a incorporar recomendações não só no âmbito da saúde como também no processo de negócio das organizações de saúde, o que permitiu sua aplicabilidade das mais variadas formas. A-pesar desse avanço, o panorama atual está muito aquém do desejável, vários aspectos, como a complexidade do domínio de saúde tem dificultado a implantação dos sistemas de informação em saúde, bem como dos sistemas de CDS. Necessita-se vencer essas barreiras para tornar factível a utilização o quanto antes dos sistemas de CDS, pois estes além de diminuir os custos das organizações de saúde, interferem diretamente na qualidade da prestação de saúde, evitando erros e, conseqüência, salvando vidas.

Capítulo

3

Design do MultiPersOn-CDS

> "O gênio é um por cento de inspiração e noventa e nove por cento de transpiração."
>
> *Thomas Edison*

Para entender como o MultiPersOn-CDS funciona, necessitamos analisar os aspectos que levaram sua criação, o que ele se propõe a fazer, a estrutura dos seus elementos, como a ontologia o auxilia, quais as características dos seus agentes e, por fim, como ele pode interagir com as demais aplicações.

3.1 Princípios

A construção de um sistema de CDS está fadada ao fracasso se vários aspectos não forem ponderados, antes mesmo da etapa de projeto, tornando factível sua implantação e utilização pelos usuários finais. Por essa razão, o MultiPersOn se preocupou com os seguintes pontos: flexibilidade, portabilidade, interoperabilidade, manutenibilidade, separação de propósitos, reatividade e relevância.

3.1.1 Flexibilidade

Os sistemas de CDS devem se adequar as diferentes necessidades, oriundas das variações do domínio ou dos métodos de interação. As variações do domínio são decorrentes de atualizações de conceitos clínicos ou surgimento de novos procedimentos, medicamentos, entre outros. Os métodos de interação com os sistemas de CDS, por sua vez, podem divergir amplamente dependendo da cultura e dos recursos utilizados pelas organizações de saúde. Assim, para realizar o CDS, esses sistemas devem poder capturar os dados sendo preenchidos no RES, bem como, os dados de sensores corporais situados nos pacientes. E, através da interface gráfica com o usuário ou por meio de qualquer

outro dispositivo de interação com o usuário, como SMS ou email, gerar um auxílio à decisão clínica.

3.1.2 Portabilidade

Para serem amplamente difundidos, os sistemas de CDS devem ser capazes de funcionar em diferentes plataformas, atraindo assim cada vez mais organizações de saúde. Além disso, os algoritmos de CDS utilizados devem poder ser executados na linguagem em que foram codificados, permitindo assim a utilização de algoritmos de CDS já consagrados com implementações garantidamente corretas.

3.1.3 Interoperabilidade

Os sistemas de CDS devem oferecer meios para trocar informações com diferentes sistemas em diferentes plataformas, proporcionando assim a transmissão de informações de CDS para os mais variados dispositivos. Para que isto seja alcançado, faz-se necessário que esses sistemas tenham um baixo acoplamento com o resto das aplicações, tornando-se independente da implementação do RES ou do meio de entrada e saída dos dados.

3.1.4 Manutenibilidade

A utilização de um sistema de CDS não acarreta mudanças na organização apenas durante o seu desenvolvimento, uma grande mudança deve acontecer após sua implantação para garantir, de agora em diante, a manutenção da infra-estrutura necessária para o seu funcionamento. Além disso, os mecanismos de CDS desenvolvidos para o domínio da saúde, que é bastante complexo, podem se tornar de difícil gerência com o passar do tempo, dependendo da escalabilidade e complexidade deles. Por esse motivo, devem-se buscar soluções que valorizem a facilidade de manutenção, favorecendo assim sua utilização de forma duradoura.

3.1.5 Separação de propósitos

Devem-se elaborar soluções que tentem isolar a lógica dos métodos de CDS da lógica do sistema de CDS propriamente dito. A lógica por detrás de um método de CDS é uma só, que em tese independe do sistema de CDS utilizado. Entretanto, muitas vezes ela sofre alterações, senão reestruturação, para se adaptar as necessidades dos sistemas de CDS. Desta forma, ao buscar esta distinção, pode-se utilizar profissionais especializados para realizar tarefas independentes, podendo ter um especialista no sistema de CDS, mais familiarizado com a área de sistemas de informação, responsável pela gerência do

sistema, bem como, sua adequação aos diferentes métodos de CDS. E em paralelo, pode haver um especialista na criação de métodos de CDS, mais familiarizado com o domínio de saúde, sendo responsável por criar, testar e validar esses métodos.

3.1.6 Reatividade e relevância

Os sistemas de CDS devem sempre auxiliar os profissionais de saúde, de forma rápida e com qualidade, para proporcionar um benefício garantido e imediato que possa ser percebido na qualidade da prestação de saúde, esperando-se assim, agilizar e evitar erros durante o atendimento. Contudo, é preciso ter cautela para não permear rapidez com intromissão. A busca frenética para auxiliar de forma imediata os profissionais de saúde, pode trazer como revés, o desgaste por parte dos mesmos ao lidar com informações incessantes e desnecessárias gerada pelo o sistema de CDS, resultando na ineficiência do atendimento em saúde. Assim, os sistemas de CDS devem buscar balancear a reatividade com a relevância das suas ações perante os profissionais de saúde.

3.2 Definindo o MultiPersOn

Há um grande número de soluções de CDS, muitas isoladas sem serem aplicadas diretamente a um sistema de CDS, outras concebidas como serviços por soluções comerciais, ou integradas a sistemas de CDS. Diante desse universo confuso e nebuloso de soluções existentes, o design do MultiPersOn busca deixar mais claro seu propósito, suas características e suas limitações.

O MultiPersOn é um *framework* de suporte para o apoio a decisão clínica que visa auxiliar os profissionais de saúde, tanto na parte clínica, através da recomendação de ações no domínio da saúde, bem como, no fluxo de trabalho, através, por exemplo, do complemento automático de palavras ou de avisos das atividades a serem realizadas. Além disso, o MultiPersOn não depende de nenhum dispositivo de interação específico, podendo interagir com qualquer outro dispositivo ou sistema que satisfaça seu modelo de interoperabilidade. Para obter essa independência e garantir a conformidade das suas ações, é necessária a implementação dos eventos gerados pelo *framework,* conforme a descrição conceitual de cada um deles. Esses eventos representam os diversos métodos de CDS, podendo eles ser estendidos e personalizados livremente.

E ainda, para facilitar a implementação e a manutenção, o *framework* utiliza como unidade básica de CDS, agentes, pois oferecem claramente a separação da lógica com a estrutura. Assim, a estrutura do agente é descrita por meio de uma ontologia que contém as informações necessárias para a execução da sua lógica, além de outras infor-

mações contextuais que, por ventura, possam ser úteis em sua lógica, como informações sobre a aplicação, o usuário ou paciente. Vale ressaltar que os agentes foram concebidos para não ter inteligência ou para ter uma inteligência de forma limitada, realizando ações simples ou de baixa complexidade, mas que lidam com informações muito valiosas e poderosas para a melhoria da qualidade da prestação de saúde.

3.3 Funcionamento do MultiPersOn

Para conseguir auxiliar os profissionais de saúde, os agentes do MultiPersOn precisam tomar ciência das características do ambiente no qual o MultiPersOn está inserido. Além disso, a comunicação entre o *framework* e ambiente externo deve ser a mais desacoplada possível, a fim de favorecer sua interoperabilidade. Deste modo, foi adotado um modelo de comunicação baseado em eventos que devem ser lançados quando ocorrerem alterações nos sistemas que interagem com o MultiPersOn. É importante notar que esta comunicação, por ser "frouxa", deve ser feita de maneira sincronizada e consentida entre o MultiPersOn e os sistemas que interagem com ele, evitando que algum evento seja ignorado ou usado erroneamente, corroborando, assim, para que o funcionamento do sistema aconteça conforme o esperado.

Um aspecto relevante diz respeito aos agentes do MultiPersOn, que tiverem seus ciclos de vida limitados, devido a complexidade do domínio de saúde, pois poderia acarretar em uma degradação da performance do sistema como um todo. Desta forma, os agentes foram classificados em agentes passivos que são ativados por algum evento externo, realizam uma lógica e adormecem, e os agentes ativos que são ativos ciclicamente, de acordo com um período tempo, para realizar uma lógica específica.

Além disso, duas etapas bastante distintas definem o ciclo de vida desses agentes. A primeira etapa é durante a construção dos agentes que ocorre quando o *framework* é iniciado. A segunda etapa, a de execução dos agentes, acontece quando os agentes são ativados, sendo que esta depende das características específicas de cada um dos agentes, enquanto que a primeira é mandatória e ocorre ao mesmo tempo para todos eles.

Para a descrição das informações específicas de cada agente, optou-se por utilizar ontologias, pois são capazes de representar as informações necessárias para a construção dos agentes de CDS de maneira robusta e padronizada, facilitando assim a interoperabilidade entre sistemas. Como essas informações estão desassociadas da lógica de cada um dos agentes, o MultiPersOn buscou, também, utilizar uma solução robusta para a execução dessa lógica, através da não obrigatoriedade de uma determinada linguagem para a representação da lógica do agente. Isto foi alcançado por meio do JSR-223

Scripting for the Java Platform que visa definir uma maneira padronizada e portável para permitir que programas escritos em linguagens de *scripts* sejam executados em Java. Esta solução apresenta compatibilidade com diversas linguagens de *scripts*, tais como, javascript, ruby, python, groovy, etc.

Dito isto, podemos entender melhor o funcionamento geral do MultiPersOn, conforme pode também ser visto na Figura 3.1. Assim, quando alguma alteração acontecer no RES ou em outros dispositivos e esta for repassada para o MultiPersOn, este checará se esse evento ativa alguma agente (Checar Evento) previamente construído (Construir Agentes), em caso afirmativo, esse agente será ativado (Ativar Agentes) e posto em execução (Executar Agentes). Antes, porém, obterá as informações contextuais necessárias para sua execução (Obter Contexto), decorrendo num resultado que será então enviado (Enviar Resultado) para um dos sistemas que interagem com o *framework*.

Ademais, os agentes ativos são construídos através de um escalonador de tarefas, chamado Quartz, que ativa os agentes num determinado instante de tempo para serem executados (Escalonar Agentes) e por fim envia o resultado para os sistemas externos que se comunicam com MultiPersOn.

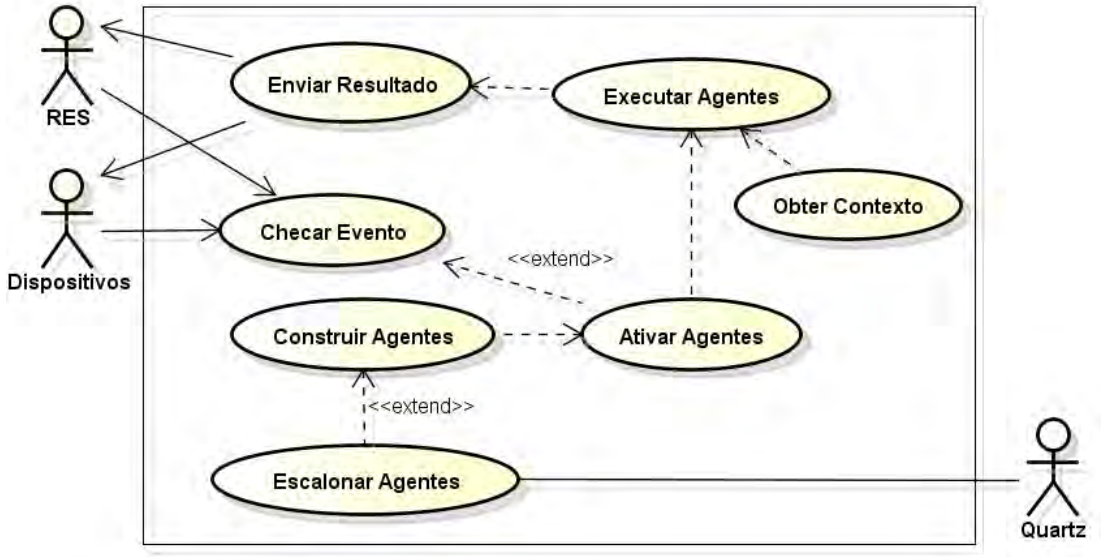

Figura 3.1 - Diagrama de caso de uso do fluxo de funcionamento geral do MultiPersOn.

Através desse funcionamento, o *framework* visa auxiliar o processo de tomada de decisão, por meios dos agentes de CDS, objetivando alcançar todos os princípios supracitados no item 3.1.

3.4 Arquitetura do MultiPersOn

Sabendo sobre o funcionamento geral do MultiPersOn, um melhor entendimento sobre cada um de seus elementos se faz necessário para possibilitar a utilização e disseminação do framework, através da sua aplicação num cenário real, fomentando assim a melhoria na prestação do atendimento em saúde.

Conforme foi dito anteriormente, o MultiPersOn utiliza agentes para a realização da lógica dos métodos de CDS. Esses agentes são inicialmente construídos quando o *framework* é iniciado, ficando adormecidos para serem executados de acordo com as características de cada um. Assim, conforme pode ser visto na Figura 3.2, o *framework* é iniciado pela instância da classe *CDSManager* (elemento central da aplicação) ao invocar o método *init*() que carrega as dependências e constrói todos os agentes de CDS. Para tanto, chama o método *buildAgentsModel*() da classe *CDSBuilder* que tem como objetivo ler a ontologia que descreve os agentes e extrair as informações necessárias para a construção de uma instância da classe *AgentModel* para cada um dos agentes. Caso haja algum agente ativo, ou seja, que não depende de estímulos do ambiente externo para ser ativado, este será encaminhado para a instância da classe *CDSQuartz* que utiliza o *framework* de escalonamento de tarefas, Quartz, através do método *scheduler*(…) para agendar a ativação desses agentes. Desta forma, todos os agentes, sejam ativos ou passivos, são construídos, ficando adormecidos (*AgentModel*) para serem executados quando forem ativados.

Os agentes que forem ativados mediante alguma alteração no ambiente externo, isto é, por meio dos eventos notificados ao MultiPersOn, através da chamada do método *eventRaised*(…), são clonados pelo método *cloneObject*(*agentModel*) da classe *CDSUtil* e utilizam as informações oriundas do evento lançado para carregar o agente com suas informações contextuais, por meio de uma instância da classe *AgentWrapper*. A partir disso, ele é enviado para a ativação através do método *activateAgent (agent)* da instância da classe *CDSEngine*, tem seu contexto carregado e segue para ser executado. Para realizar sua execução, uma *thread* é criada para cada agente, isto é, uma instância de *AgentThread*, sendo encaminhado para a interface de escalonamento de processos do Java, *ExecutorService*, que no instante apropriado executa a instância de *AgentThread.* Durante sua execução, caso este deseje enviar alguma informação ao ambiente externo, o método *convey*(…) da classe *CDSResult* é chamado para que este possa repassar a informação para o canal correto, de acordo com o contexto do apropriado do agente, através do lançamento do evento de retorno, *raiseEvent*(…).

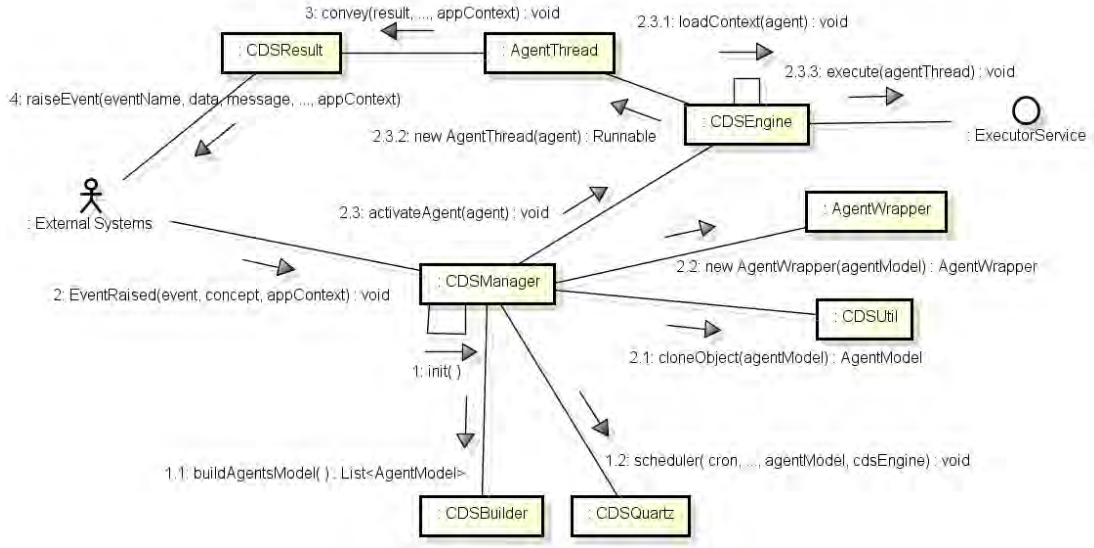

Figura 3.2 - Diagrama de Comunicação UML do MultiPersOn.

Os agentes ativos procedem de maneira análoga, sendo que ao invés deles serem ativados através do passo 2, *eventRaised*(…), são ativados pelo Quartz, tendo então todos os mesmos passos subseqüentes dos agentes passivos.

Para que fosse possível essa troca de informações entre o MultiPersOn e o ambiente externo, isto é, as demais aplicações que interagem com o *framework*, foi utilizado uma arquitetura dirigida à eventos (*event-driven architecture*, EDA). Esse padrão arquitetural pode ser aplicado na troca de mensagens por meio de componentes de software fracamente acoplados ou serviços, onde existe um emissor de eventos, responsável por lançá-los assim que eles acontecerem, e um consumidor de eventos, responsável por reagir a eles o quanto antes.

Essa interação das aplicações com o MultiPersOn se dá por meio da interface, *ICDSManager*, fornecida pelo *framework* para a obtenção de informações referentes ao seu funcionamento ou para a notificação de eventos externos. Essa interface, mostrada na Figura 3.3, possui cinco métodos, sendo um deles responsável por recarregar os agentes, *updateAgents( )*, útil quando ocorrer alguma alteração na lógica dos agentes, outro, *getLoadedAgents(*), objetiva listar os agentes que foram construídos e estão adormecidos, já o outro, *getRunnningAgents*( ), visa listar os agentes em execução num dado instante de tempo, e , por fim, restam respectivamente o método de notificação de eventos externos ao MultiPersOn, *eventRaised*(…), e o método que informa os eventos de CDS associados a um conceito, *getEventsBind*(…).

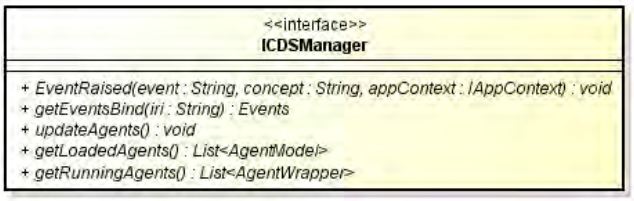

Figura 3.3 – Diagrama de classe UML da Interface *ICDSManager*, responsável pela obtenção de informações referentes ao *framework* e pela notificação dos eventos externos.

Com relação à notificação de um evento externo, esta deve ser acompanhada da *String* contendo o nome desse evento, outra *String* com um nome unívoco do conceito relacionado com esse evento e o contexto da aplicação em que ele esta inserido. Essas informações são importantes, pois utilizando o nome do evento junto com o nome unívoco do conceito é possível saber quais agentes de CDS irão ser ativados. Por outro lado, o contexto da aplicação é indispensável para obter as informações contextuais e possibilitar que os sistemas externos tratem adequadamente os eventos lançados pelo MultiPersOn.

O método *getEventBind*(…) que indica os eventos de CDS associados a um conceito é utilizado para que a geração da interface gráfica possa se adaptar as reações dos mecanismos de CDS, permitindo que o sistema funcione harmonicamente, sem que nenhum evento oriundo do MultiPersOn seja ignorado e, para tal, necessita como parâmetro do nome unívoco do recurso ou IRI (*Internationalized Resource Identifier*) em questão.

3.4.1 Classes funcionais do MultiPersOn

A estrutura das classes funcionais do MultiPersOn pode ser resumida, conforme mostrado na Figura 3.4, nos seguintes elementos: *CDSManager*, *CDSBuilder*, *CDSProperties*, *CDSEngine*, CDSQuartz e *CDSResult*.

Inicialmente os agentes são construído a partir da descrição na ontologia (*CDSBuilder*), sendo então armazenados no *CDSManager*, através de um mapeamento dos agentes com seus ativadores (*Trigger*). Quando o *CDSManager* recebe uma notificação de um evento externo, ele checa se algum agente é ativado por esse evento. Caso seja ativado, o agente será enviado para o *CDSEngine* e sua lógica será executada concomitante com a de outros agentes já em execução. Durante a execução os agentes podem usar o *CDSResult* para notificar o acontecimento de certos eventos de CDS, permitindo assim a interação com os dispositivos externos ou outras aplicações.

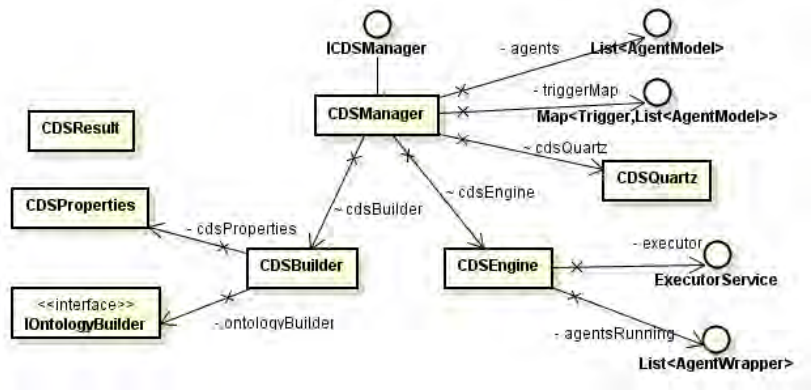

Figura 3.4 - Diagrama de classe UML simplificado, contendo as principais classes funcionais do MultiPersOn.

Agora que já temos algum conhecimento sobre o funcionamento do MultiPersOn, estamos aptos a entender detalhadamente cada uma das classes encontradas na Figura 3.4, a fim de compreendermos melhor o papel específico de cada uma delas.

Primeiramente, vamos nos concentrar na classe central do *framework,* que está em destaque na Figura 3.5, a classe *CDSManager*. Ela é uma implementação da interface *ICDSManager*, possuindo instâncias de outras três classes: *CDSBuilder*, *CDSQuartz* e *CDSEngine*. Ela é a responsável por armazenar a lista de agentes passivos construídos a partir da ontologia, através da instância de *CDSBuilder,* para serem executados pela instância de *CDSEngine* quando estes forem ativados. Para armazenar, de maneira eficiente, qual evento externo ativa quais agentes, foi criado um *hash*, chamado *triggerMap*, que tem como chave as informações que descrevem o evento de ativação e como valor uma lista de agentes a serem ativados pelo evento. No caso dos agentes ativos, eles são encaminhados para a instância de *CDSQuartz*, onde são agendados para serem executados pela instancia do *CDSEngine* num dado instante de tempo.

Além da implementação dos métodos da interface *ICDSManager*, a classe *CDSManager* ainda possui os métodos *init*( ) e *remove*( ) que devem ser chamados respectivamente para carregar e para encerrar corretamente o *framework.* E ainda, os métodos *mapInfraStrutucture*(…) e *mapResult*(…) que servem, respectivamente, para adicionar recursos de infra-estrutura aos agentes e preparar os agentes para encaminharem seus eventos para o ambiente externo, de acordo com o que foi descrito por cada um deles na ontologia.

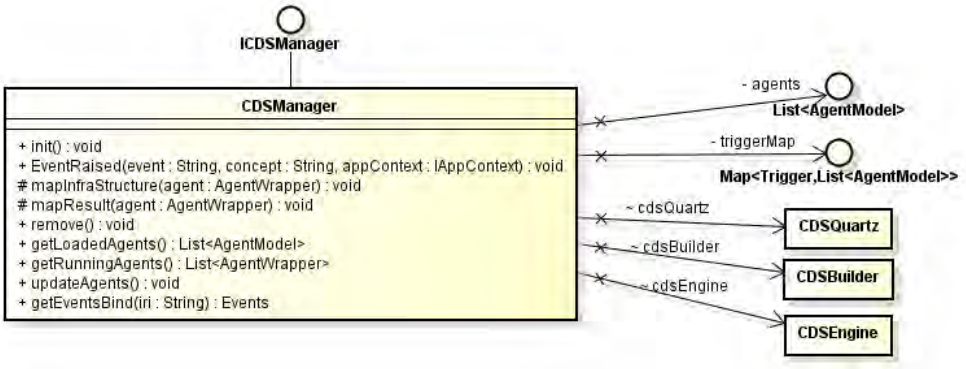

Figura 3.5 – Diagrama de classe UML da classe principal classe do *framework*.

Para iniciar a construção dos agentes, deve-se invocar o método *buildAgentsModel*() da classe *CDSBuilder,* ilustrado na Figura 3.6. Durante esse processo, necessita-se saber o endereço das ontologias utilizadas para descrever os agentes. Conforme será explicitado no capítulo seguinte, existem duas ontologias, uma que descreve os agentes e outra de configuração que descreve os recursos utilizados pelos agentes. Esses endereços são processados através da classe *CDSProperties* que procura o valor do atributo `multiperson.cds.ontology.config` e `multiperson.cds.ontology.agents` num arquivo específico, utilizando valores pré-definidos caso não encontre este arquivo. Sabendo onde se encontram as ontologias, elas são então carregadas através da interface *IOntologyBuilder*, que visa carregar e converter as ontologias de OWL para OO, de acordo com a estrutura da Figura 2.6, permitindo assim a construção dos agentes.

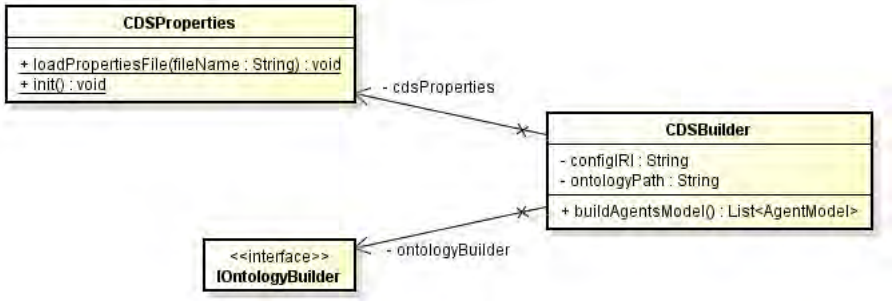

Figura 3.6 - Diagrama de classe UML mostrando as estruturas responsáveis por informar o endereço das ontologias e carregá-las para construir os agentes.

É importante ressaltar que nesta etapa todos os agentes independentemente do tipo, seja passivo ou ativo, são carregados a partir das informações contidas nas ontologias. Entretanto, o *CDSManager* envia os agentes ativos para serem agendados, através do método *scheduler*(…) da instância da classe *CDSQuartz,* mostrada na Figura 3.7, que,

por sua vez, utiliza o *framework* do Quartz para realizar esse agendamento. Para que esse agendamento seja possível, é preciso passar como parâmetro uma instância de *Date*, que representa a data limite de repetição da condição de iteração, o intervalo de iteração no formato dos escalonadores de tarefas dos sistemas baseados em Unix (cron), o agente que deve ser agendado para a execução, as informações referentes à ativação do agente e a instância do *CDSEngine* responsável por sua execução.

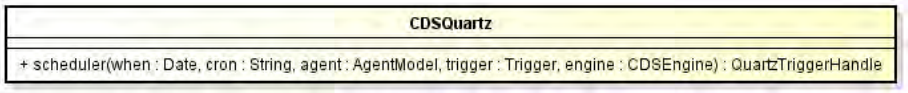

Figura 3.7 – Diagrama de classe UML contendo a classe responsável pelo agendamento da execução dos agentes passivos.

Independentemente da forma com que os agentes cheguem para ser executados, seja pela ativação de um evento externo ou pelo agendamento do Quartz, eles irão de uma maneira ou de outra ser enviados para a execução pela instância de *CDSEngine*. Essa execução é realizada através de um recurso gerenciado pelo próprio Java que cria um *pool* de *threads* sob demanda e utiliza o algoritmo de escalonamento *round-robin*, sendo inicializado junto da instância do *CDSEngine* por meio do método *init*( ), conforme explicitado na Figura 3.8. Para manter um maior controle e informar ao ambiente externo quais agentes estão sendo executados, o *CDSEngine* guarda uma lista dos agentes em execução, isto é, *List<AgentWrapper,>* e possibilita o acesso a essa informação através do método *getAgentsRunning( )*.

Mas, antes de serem executados os agentes precisam ser encaminhados para o *CDSEngine*. Isto é feito por meio do método *activateAgent*(…), que se responsabiliza em carregar o agente, utilizando as suas informações de ativação. Estas informações são imprescindíveis para conseguir carregar corretamente seu contexto, *loadContext*(…), permitindo então prosseguir para sua execução.

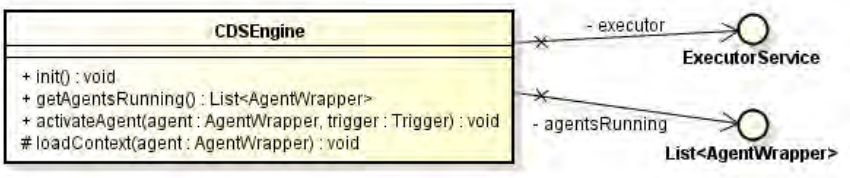

Figura 3.8 – Diagrama de classe UML da classe responsável pela execução dos agentes.

Durante sua execução, os agentes podem enviar informações ao ambiente externo, através de eventos gerados pelo *framework* de CDS. Para lidar com isto, a instân-

cia da classe *CDSResult*, ao receber cada informação enviada pelo agente, tem o papel de gerar e encaminhar os eventos de CDS para o ambiente externo. Como é mostrado na Figura 3.9, ela necessita para isso das informações sobre a forma de interação do resultado que deseja ser enviado, *Result*, a mensagem que deseja ser enviada, *List<Object>,* possíveis propriedades da mensagem, *Map<String, String>,* e o contexto da aplicação no qual a mensagem é direcionada, *IAppContext*.

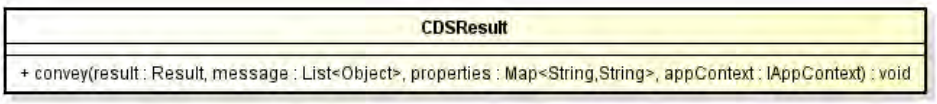

Figura 3.9 – Diagrama de classe UML da classe responsável por gerar e encaminhar os eventos ao ambiente externo.

O entendimento dessas classes funcionais do MultiPersOn é primordial para compreender o propósito do *framework*, pois além de caracterizar todo o seu funcionamento, elas operam sobre as classes estruturais do *framework* que serão vistas a seguir.

3.4.2 Classes Estruturais do MultiPersOn

Com o detalhamento das principais classes funcionais do MultiPersOn, podemos então analisar mais cuidadosamente cada uma das classes estruturais que são manipuladas por elas e que guardam todas as informações necessárias para o funcionamento do *framework*.

Para representar os agentes que foram construídos, guardando as informações necessárias para suas execução, utiliza-se a classe *AgentModel,* mostrada na Figura 3.10. Ela apresenta apenas uma série de campos contendo as informações que foram extraídas da ontologia, além dos métodos de acesso e modificação (*getters* e *setters*) de cada campo.

Como os agentes necessitam ser ativados para executar, foi também preciso definir o que ativaria (*Trigger*) cada um deles. Por isso, definiu-se que toda ativação poderia ativar um conjunto de procedimentos dentro da lógica do agente, dependendo da especificação do identificador único do evento externo, *eventID*, e do identificador único do alvo dessa ação, *targetID.* Isto permite que determinados trechos da lógica do agente sejam executados apenas se certas condições específicas forem alcançadas, aumentando o nível de controle sobre o agente e tornando a solução mais robusta.

Assim como os ativadores, foi criada uma classe para representar os recursos das aplicações que podem interagir com o MultiPersOn. Desta forma, todo recurso, ins-

tância da classe *Resource*, é composto por uma string que identifica esse recurso e uma lista de *hashs* que servem para representar as possíveis propriedades desse recurso.

Além disso, foi necessário definir alguns escopos, isto é, uma delimitação para a aplicação de um determinado contexto, para possibilitar a associação correta dos recursos utilizados na lógica dos agentes. Assim, conforme se pode visualizar na classe *CDSScope*, foram definidos quatro diferentes escopos: *document*, *patient*, *user*, *application*.

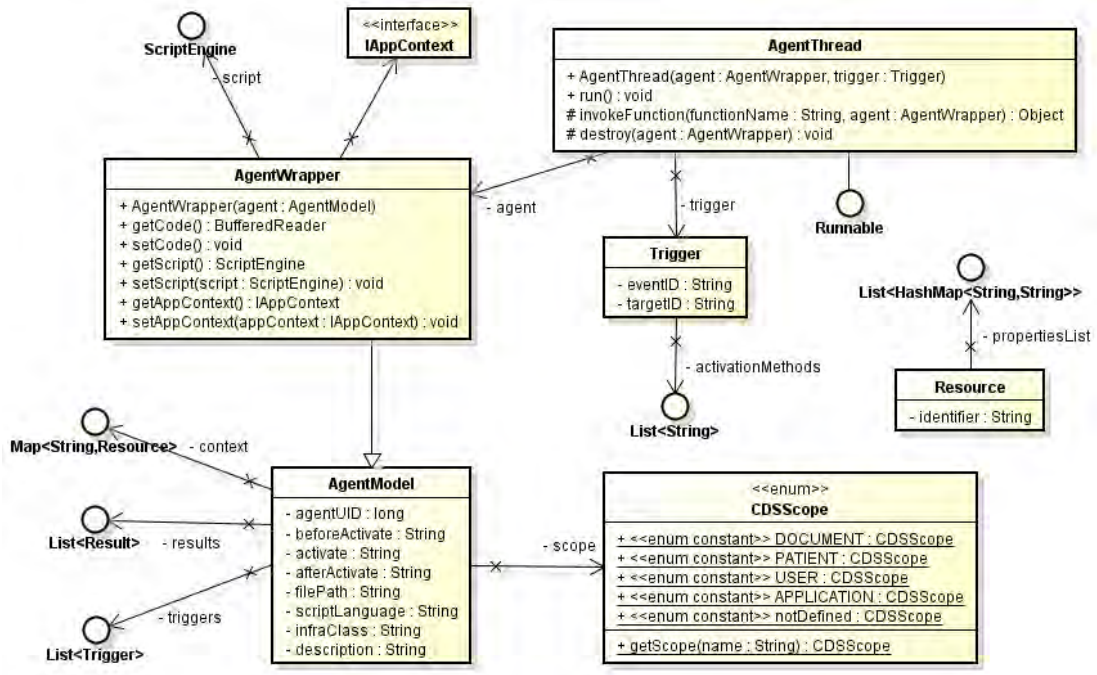

Figura 3.10 - Diagrama de classe UML das classes estruturais que definem as informações necessárias para a execução dos agentes.

Após o agente ser construído e ativado, ou seja, possuir uma instância de *AgentModel* e uma de *Trigger*, este deve ser então carregado, isto é, deve possuir um contexto da aplicação que está usando o *framework*, bem como, ter sua lógica carregada. O contexto da aplicação é representado pela instância da classe *IAppContext*, enquanto que a lógica do agente carregada pelo JSR-223 é representada interface do *ScriptEngine*. Nesse instante, a instância de *AgentModel* deve ser transformada em uma instância de *AgentWrapper*. Este funciona como um invólucro que engloba a instância de *AgentModel,* associando a ele o contexto da aplicação, a lógica do agente e, ainda, modificando e acrescentando alguns comportamentos.

Agora que o agente foi carregado, ou seja, encapsulado pelo *AgentWrapper*, este pode então ser executado. Para que todos os agentes executem sua lógica independente uns dos outros, um novo objeto deve ser criado para lidar com esse problema, o *AgentT-*

hread. Este objeto implementa a interface *Runnable*, permitindo assim que seu método *run*() seja executado por uma das *threads* do *ExecutorService*. Ao iniciar a execução, dependendo das configurações de ativação do agente, um ou mais procedimentos da sua lógica podem ser executados através do método *invokeFunction*(…). Durante cada ativação três procedimentos podem ser executados consecutivamente, sendo o segundo obrigatório e os demais opcionais. O primeiro deles, *beforeActivate*, visa carregar valores previamente armazenados, o segundo , *activate*, visa realizar o processamento da lógica e o último, *afterActivate*, visa armazenar os valores processados para serem utilizados posteriormente. Ao final, o método *destroy*(…) será chamado para que todos os valores alterados sejam resetados e o agente descartado. Vale a pena notar que os agentes são descartados com o propósito de otimizar a performace da aplicação, pois como cada agente corresponde a uma *thread* caso eles permanecessem em execução, haveria para cada usuário várias *threads,* acarretando assim na sobrecarga da aplicação.

Entretanto, antes da sua execução finalizar, o agente pode enviar alguma informação ao ambiente externo. Para permitir isto, foi criada a classe *Result* (Figura 3.11) que representa a forma de interação do resultado, isto é, sobre que dado o evento deve ser aplicado, *data*, qual evento do CDS deve ser lançado, *cdsEvent*, e qual o nome desse evento, *name*. Assim, de acordo com as características descritas por cada um dos agentes na ontologia, seus eventos de CDS serão injetados na sua lógica, podendo ser referenciados, através do atributo *name* dos seus eventos.

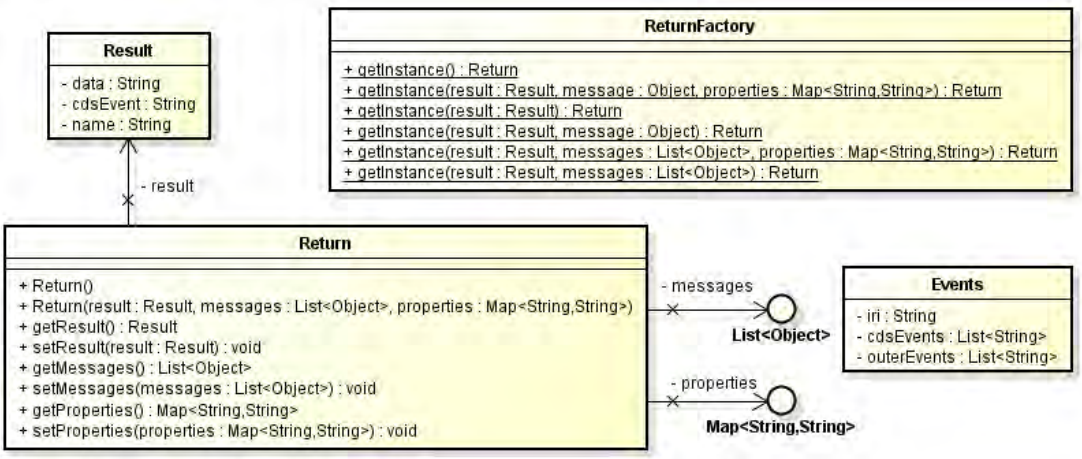

Figura 3.11 – Diagrama de classe UML das classes estruturais que definem as informações referentes aos eventos.

Desta maneira, é possível saber como e para quem a informação deve ser enviada, mas falta ainda saber qual informação. Isto foi resolvido com a classe *Return* que,

além de uma instância da classe *Return*, possui uma lista de objetos, *List<Object>*, para representar a mensagem que será enviada ao ambiente externo, bem como, um *hash, Map<String, String>,* para permitir a representação das propriedades dessa mensagem.

Mas, como o *framework* visa ser utilizado por diferentes linguagens de scripts, a permissão deliberada da criação da instância de *Return* dentro da lógica do agente seria confusa e despadronizada. Assim, a classe *ReturnFactory* foi desenvolvida e injetada dentro da lógica dos agentes, possibilitando que instancias de *Return* fossem criadas a partir de referencias para os *Result* previamente injetados. Isto, não apenas amarra as instâncias da classe *Return* a uma instância da classe *Result* válida, mas também facilita e padroniza a elaboração dos eventos de CDS para o ambiente externo. Com a referência do objeto *Return* em mãos, basta apenas retornar o objeto no procedimento em execução para que este seja encaminhado para o *CDSResult* tratar.

Por fim, não tendo relação direta com a geração dos eventos de CDS, mas correlacionado a ele, existe a classe *Events* (Figura 3.11) que serve para informar sobre um conceito, ou melhor, um *Resource*, quais são os eventos de CDS e do ambiente externo que podem ser lançados sobre ele. Isto é muito útil para possibilitar uma melhor reatividade da interface com o usuário, permitindo que ela seja construída de acordo com as características de cada conceito.

Em suma, a classe *AgentWrapper* contém uma série de atributos que visam representar de forma genérica um agente, fomentando assim a flexibilidade e portabilidade dos métodos de CDS. Ao ser englobada pela classe *AgentWrapper*, cada agente passa a ter um identificador unívoco, um escopo, uma lista de ativadores (*Trigger*), uma lista com as diferentes formas de interação dos resultados (*Result*), um contexto dentro da aplicação, uma lógica a ser executada e um mapeamento de atributos e valores que serão injetados dentro da sua lógica para representar suas informações contextuais. Além disso, contém uma descrição sobre o próprio agente, o nome da linguagem utilizada na sua lógica, o nome da classe de infra-estrutura utilizada, juntamente com os nomes dos métodos para serem executados quando o agente for ativado (*beforeActivate*, *activate* e *afterActivate*). Todas essas informações são obtidas segundo a descrição de cada um dos agentes na ontologia, com exceção do seu identificador que é gerado no instante da sua criação e da lógica dos agentes que é carregada a partir de um arquivo em separado.

3.4.3 Classes Estruturais para Aplicação do Contexto

Para que os agentes possam acessar as informações corretas durante a execução da sua lógica, é preciso saber qual seu contexto da aplicação. O contexto da aplica-

ção vai indicar quem é o usuário utilizando o sistema, sobre quais dados estão sendo gerados os eventos e qual a estrutura que manipula esses dados. A partir daí, pode-se utilizar o escopo para processar adequadamente estas informações para obter as informações requisitadas pelo agente.

Como cada aplicação que interage com o MultiPersOn tem suas próprias estrutura de dados e se comportam de maneira diferente, cada uma delas deve implementar um contexto de aplicação, possibilitando assim que o *framework* acesse os dados dessas aplicações. Na Figura 3.12 podemos perceber dois contextos de aplicação: *HealthApplicationContext* e o *TradicionalApplicationContext*. Os dois contextos foram criado para a mesma aplicação, o OpenCTI, pois ela utiliza estruturas bastante distintas para realizar suas funções. O primeiro foi criado para acessar os dados de saúde dos pacientes, enquanto que a segunda para acessar os dados demográficos e informações sobre a aplicação.

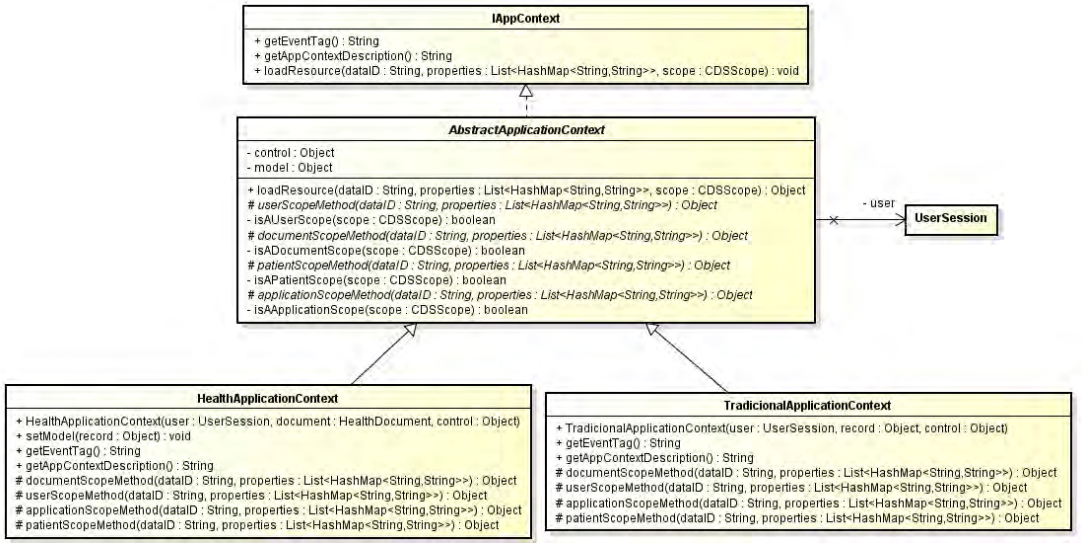

Figura 3.12 – Diagrama de classe UML das classes responsáveis pelos contextos das aplicações.

Assim, evidencia-se que todo contexto de aplicação deve ser uma subclasse de *AbstractApplicationContext* e que deve implementar seus métodos abstratos para que o contexto possa ser usado. Como esta classe implementa a interface *IAppContext*, todo contexto de aplicação pode ser tratado por essa interface.

Na classe *AbstractApplicationContext* podemos observar os três elementos do contexto da aplicação: as informações referentes ao usuário (*UserSession*), os dados que serão acessados (*model*) e a estrutura que manipula os dados (*control*). Além dos métodos abstratos herdados, essa classe também possui quatro métodos abstratos, sendo um para

cada escopo do MultiPersOn. Pois dependendo do escopo do agente, será preciso manipular os dados de maneira diferente para conseguir acessar os dados das aplicações.

Da mesma forma que o contexto e o escopo variam de aplicação para aplicação, o modo como os conceitos são identificados em cada uma delas também pode variar. Por esse motivo, para que cada conceito possa ser interpretado especificamente pela aplicação ao qual o recurso pertença, foi criada a interface *IConceptIdentifier* na qual cada uma das aplicações deve implementar para processar o conceito corretamente.

Na Figura 3.13, podemos ver algumas dessas implementações de acordo com um propósito específico. O próprio *framework* também implementa a interface *IConceptIdentifer* para conseguir interpretar alguns conceitos da sua maneira.

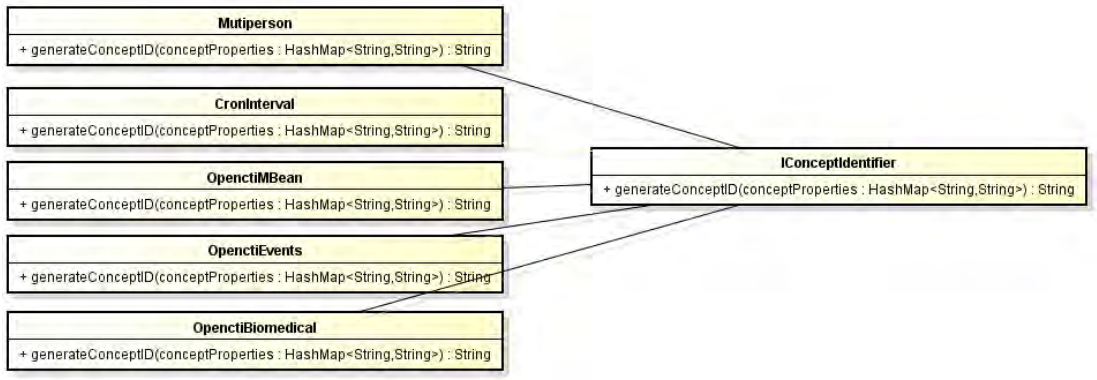

Figura 3.13 – Diagrama de classe UML com diversas classes que utilizam as informações retiradas da ontologia diferentemente de acordo com cada um desses conceitos.

Essas duas estruturas básicas: *AbstractApplicationContext* e o *IConceptIdentifier,* permitem que o framework seja aplicado nas mais diversas circunstancias, fomentando assim a disseminação da solução e colaborando para a sua robustez.

3.4.4 O Processo de Criação dos Agentes

Muitos antes dos agentes serem ativados para executar as lógicas de CDS, é necessário que o *framework* seja carregado para permitir a construção dos agentes, possibilitando assim que estes possam contribuir para a melhoria da qualidade na prestação de saúde.

Como este processo de criação dos agentes é uma etapa fundamental para o *framework*, vamos visualizar detalhadamente esse processo, através da Figura 3.14.

Este processo se inicia quando o MultiPersOn é inicializado pelo servidor de aplicação no qual ele está contido, por meio do método *init*(), que é invocado automaticamente junto com o *framework*. A partir daí, o método *buildAgentsModel*() é chamado para dar início ao processo de criação dos agentes.

A principal classe responsável por esse processo é a classe *CDSBuilder*. Ela cria uma instância de *CDSProperties* que ao ser criado, checa se há um arquivo de configuração com os valores das variáveis necessárias para o funcionamento do *framework*. Entre elas, o endereço da ontologia dos agentes, o endereço da ontologia de configuração dos recursos e o endereço da lógica dos agentes de CDS. Caso não encontre este arquivo, os valores default são carregados.

Independentemente da forma, seja pelo arquivo ou através dos valores default, a instância de *CDSBuilder* irá tentar carregar a ontologia dos agentes, invocando o método *loadOntology*(…).Após carregada a ontologia dos agentes, é a vez das lógicas dos agentes de CDS serem também carregadas, permitindo assim que, através do método *createAgents*(…), os agentes sejam construídos. Durante essa etapa, as informações contidas na ontologia dos agentes e sua lógica são processadas e associadas a cada um dos agentes, gerando com isso uma lista de *AgentModel*.

Depois de sua construção, os agentes são separados em ativos ou passivos, sendo então os ativos encaminhados para que o *CDSQuartz* possa agendar a ativação deles, por meio do método *scheduler*(…). Os agentes remanescentes, ou seja, os passivos são então mapeados num *hash*, utilizando seus ativadores como chave do *hash* para que ao receber um evento externo, dependendo do ativador deste evento, rapidamente sejam sabidos quais agentes devam ser ativados.

Desta forma, unindo as informações da ontologia de agentes com a lógica contida em arquivos a parte, pode-se construir os agentes de CDS, separando os conhecimentos necessários para elaborar sua lógica das informações necessárias para permitir que o agente seja executado pelo MultiPersOn.

3.4.5 O Processo de Execução dos Agentes

Com todos os agentes devidamente construídos, o *framework* está pronto para reagir ao ambiente externo, cumprindo assim com seu objetivo de auxiliar a prestação de saúde.

Conforme já foi dito anteriormente, os agentes passivos dependem de uma ativação externa para serem encaminhados para a execução, enquanto que os agentes ativos são ativados pelo gerenciador de tarefas do Quartz. Assim sendo, ambos diferem apenas no primeiro passo do processo de execução, tendo todo o resto exatamente igual.

Como os agentes ativos podem ser vistos como um caso peculiar de agentes passivos que são ativados periodicamente por uma entidade pseudo-externa, iremos apenas analisar mais detalhadamente os agentes passivos.

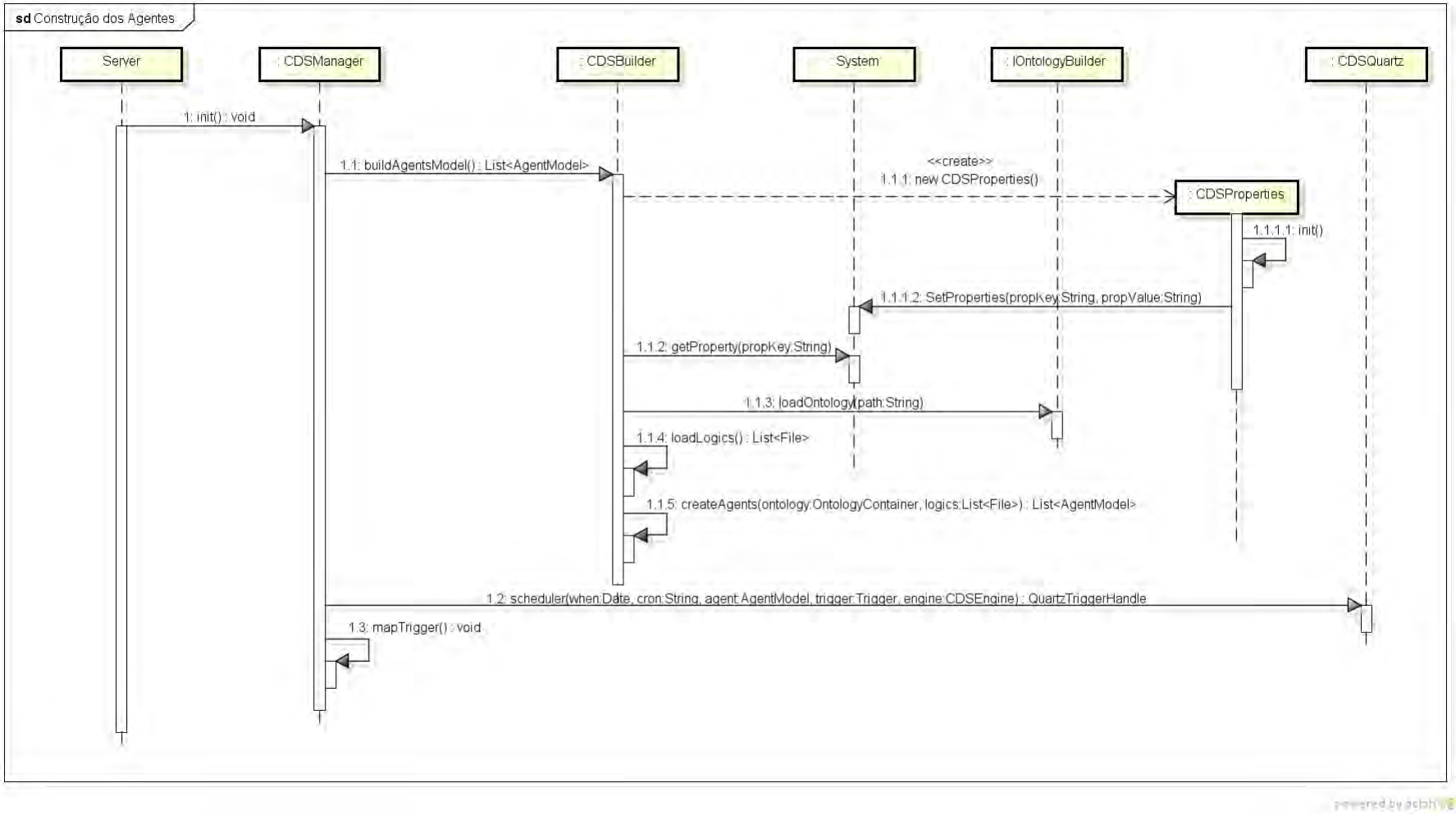

Figura 3.14 – Diagrama de classe UML representando todas as etapas do processo de criação dos agentes de CDS do MultiPersOn.

Assim, o processo de execução dos agentes se inicia através do lançamento de um evento externo, representado na Figura 3.15, pelo método *eventRaised*(…) e seus parâmetros, que contem informações referentes ao nome do evento, ao nome do conceito e ao contexto da aplicação que lançou o evento. Então, um objeto de ativação, *Trigger*, é gerado para encapsular as informações do nome do evento e do conceito, sendo utilizado para obter uma lista dos agentes que devem ser ativados a partir desse evento.

Desta forma, cada um dos agentes dessa lista será encaminhado para a execução seguindo a mesma descrição a seguir. Primeiramente, o agente construído é clonado e encapsulado pela instância de *AgentWrapper* que carrega sua lógica e adiciona o contexto da aplicação onde o agente deve atuar. Por fim, a ele é adicionado os recursos de infra-estrutura que os agentes podem usufruir (como conexões com banco de dados ou acesso à base de dados remotas) e, também, as informações sobre a forma de interação dos resultados dos agentes para possibilitar o envio destes resultados para o ambiente externo.

Feito isto, então, o agente é enviado para ser ativado na instância da classe *CDSEngine*, através da chamada do método *activateAgent*(…). Mas, antes de ser executado, o agente precisa ser carregado, necessitando, para isso, obter os dados contextuais que serão carregados, o contexto de aplicação do agente e seu escopo. Os dados contextuais são representados pelo *hash*, *Map<String, Resource>*, contendo como chave o nome do recurso utilizado dentro da lógica do agente e como valor o objeto que possui as informações necessárias para o carregamento desse recurso.

Com estas informações, é possível então invocar o método *loadResource*(…) do contexto de aplicação do agente para que cada um dos seus recursos seja carregado de acordo com escopo do agente. Isto fará com que a classe de *AbstractApplicationContext* redirecione a chamada para o método especifico que trata cada escopo na instância da subclasse que implementa o contexto do agente.

Depois que o agente foi carregado com seu contexto, uma instância de *AgentThread* deve ser criada para lidar com sua execução, sendo então enviado para a interface de execução, *ExecutorService.* A partir daí, em algum momento esta interface irá invocar o método *run( )* da instância de *AgentThread* e o agente começará a execução.

Durante a execução, os três procedimentos do agente *beforeActivate*, *activate* e *afterActivate* podem ser chamados, dependendo da configuração do agente. Caso o agente utilize os três procedimentos, o primeiro a ser invocado será o procedimento contido

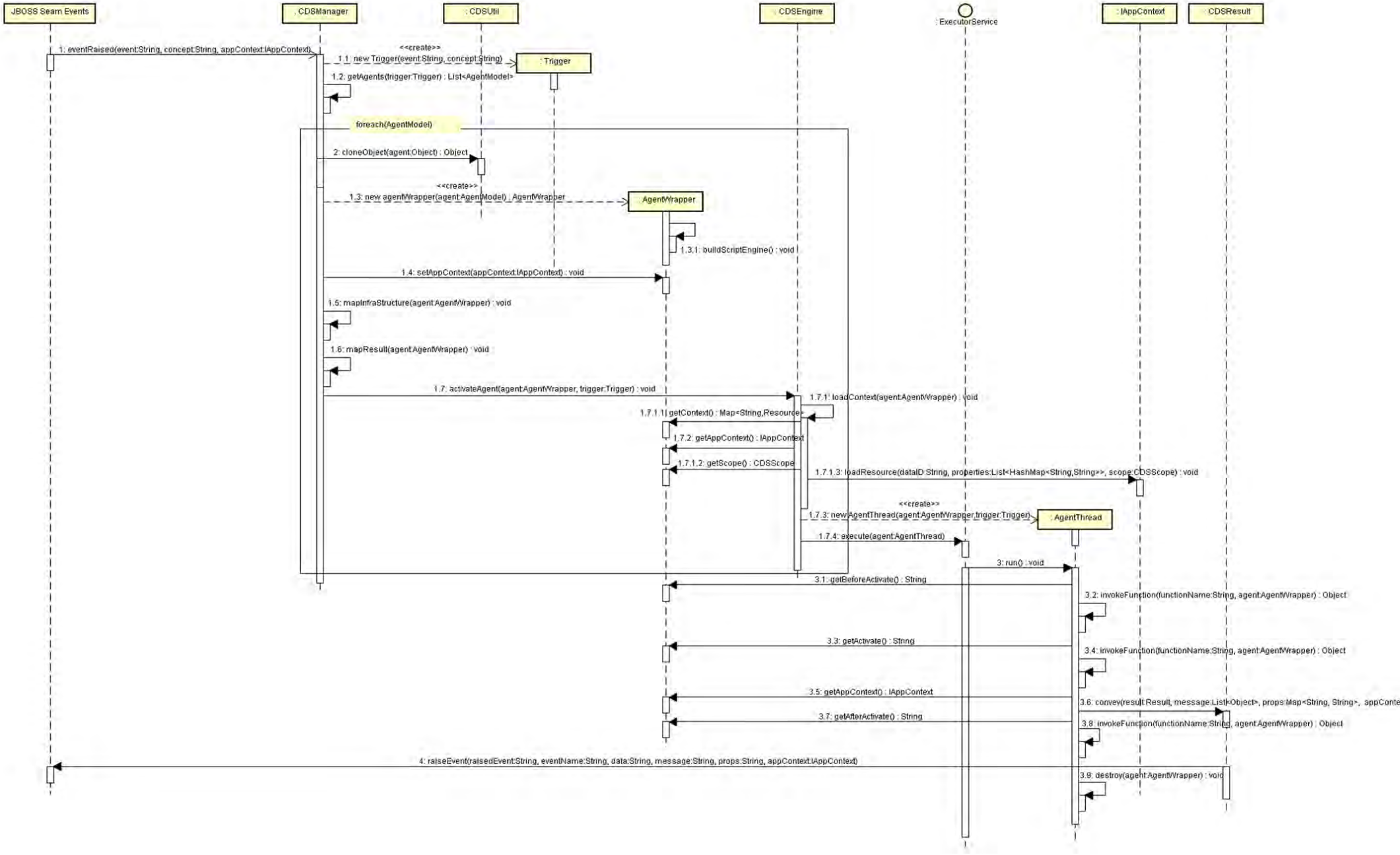

Figura 3.15 - Diagrama UML representando todas as etapas do processo de execução dos agentes de CDS do MultiPersOn.

no atributo *beforeActivate*, que tem como objetivo realizar pré-processamentos necessários para a lógica principal do agente, como o carregamento das informações de alguma base de dados. Depois, será chamado o procedimento que o atributo *activate* representa, que deve conter a lógica principal do agente de CDS, podendo neste procedimento retornar alguma informação, *Return*, para que o MultiPersOn encaminhe-a ao ambiente externo. O último procedimento a ser invocado será o contido no atributo *afterActivate*, que objetiva realizar pós-processamentos para auxiliar a lógica principal do agente, como persistir valores calculados. Por fim, o método *destroy*(…) é chamado e os valores alterados dos agentes são desfeitos e o agente descartado. Mais uma vez, é importante notar que os agentes são descartados para não degradar a performance da aplicação, otimizando assim seu desempenho.

Caso o agente deseje enviar alguma informação ao ambiente externo, ele deve retornar algum objeto *Return* no procedimento descrito pelo atributo *activate.* Estes objetos retornados serão processados e encaminhados para a instância de *CDSResult*, através do métodos *convey*(…). Chegando lá, um evento será lançado, *raiseEvent*(…), contendo o nome da aplicação para qual o evento se destina, o nome do evento em si, o conceito alvo do evento, a mensagem de resultado e suas propriedades, bem como, o contexto da aplicação que irá tratar o evento.

Assim, um sistema poderia se beneficiar com o MultiPersOn, recebendo, por meio de eventos, informações para agregar valor a sua aplicação, necessitando para isso apenas da criação dos agentes de CDS, da notificação das alterações ocorridas na aplicação e da captura dos eventos lançados pelo *framework*.

3.5 O Cenário de Implantação

Com o objetivo de permitir a interação entre as mais diversas aplicações e, também, o acesso direto da lógica dos agentes às aplicações, buscou-se um modelo de interoperabilidade simplificado, de fácil acoplamento entre as partes, que esteja em conformidade com os princípios elucidados no item 3.1. Por essa razão, foi escolhido o framework JBOSS Seam para lidar com a gerência do MultiPersOn, como também, para tratar dos aspectos referentes aos lançamentos dos eventos entre o MultiPersOn e as aplicações. O Seam é uma plataforma robusta de desenvolvimento para aplicações em Java que abstrai a implementação de muitos aspectos usuais no desenvolvimento de aplicações que não estão relacionados diretamente com o a lógica de negócio do problema.

3.5.1 Requisitos de tecnológicos

Para que os agentes de CDS do *framework* possam interagir com as aplicações, é necessário definir na ontologia de configuração de recursos do MultiPersOn, quais serão os eventos lançados por elas. Permitindo que, ao importar essas ontologias, os agentes possam utilizar esses eventos como seus ativadores. Em contrapartida, para que os agentes de CDS possam agir de maneira correta e eficaz, as aplicações devem dar suporte a todos os eventos definidos na ontologia de configuração de recursos do MultiPersOn. E mais, faz-se necessário uma interface com o usuário bastante reativa e dinâmica que se comprometa em implementar os eventos gerados pelo MultiPersOn fidedignamente, conforme a implementação de referência feito por Duarte (2011).

Além disso, o MultiPersOn deve estar no mesmo container das aplicações que desejam interagir com o *framework* para permitir que os seus agentes possam acessar as informações oriundas das aplicações. Possibilitando assim que o código da lógica dos agentes, independente da linguagem, possa acessar as informações contextuais que necessita para realizar sua lógica interna. Devendo apenas existir uma implementação do da *engine* do JSR 223 para as linguagens dos agentes.

3.5.2 Cenário de interação numa organização de saúde

O contexto das aplicações das organizações de saúde é heterogêneo, tendo um elevado número de diferentes soluções com variadas tecnologias para diversos dispositivos espalhadas por toda a organização. Tentando se moldar a esse contexto caótico, a Figura 3.16 revela como o MultiPersOn pode interagir com esses variados sistemas.

Como foi dito anteriormente, para que os agentes do MultiPersOn possam realizar suas lógicas é necessário descrever cada um desses agentes na ontologia de agentes, bem como os recursos da aplicação que são utilizados, na ontologia de configuração de recursos. Entretanto, para que essas ontologias possam ser acessadas independente do local onde o MultiPersOn está localizado, deve-se utilizar um *servidor web*, que disponibilize-as na rede. Além disso, o MultiPersOn deve estar dentro de uma aplicação que utilize o *JBOSS Seam Framework* para permitir que este gerencie o framework, dando suporte a arquitetura dirigida a eventos utilizada no MultiPersOn.

Satisfeitas estas condições, o MultiPersOn pode acessar o recursos da aplicação na qual ele está inserido ou acessar recursos remotos localizados em outras aplicações dentro do mesmo container ou em outro servidor. Para isso, necessita-se de um *middleware* que utiliza alguma tecnologia, como RMI (*Remote Method Invocation*), para emular localmente os objetos. A implementação deste *middleware* está além do escopo deste tra-

balho, entretanto seu desenvolvimento torna estendível o *framework* para o cenário explicito na Figura 3.16.

De acordo com esse cenário, o MultiPersOn acessa as informações da ontologia que são disponibilizadas por um *servidor web* e troca informações com a camada de apresentação da aplicação A, através de eventos. Para que os agentes do MultiPersOn realizem sua lógica, a camada de negócio também é acessada pelo *framework*, fornecendo assim as informações necessárias aos agentes. Além disso, o MultiPersOn também pode interagir com API's proprietárias de algum equipamento, como por exemplo, um equipamento de monitoramento das sinais vitais de um paciente, fornecendo assim diversas informações aos agentes de CDS.

Por fim, utilizando-se do *middleware* de comunicação heterogênea, pode-se possibilitar a troca de informações entre os agentes de CDS da aplicação A com outras aplicações no mesmo servidor, como é o caso da aplicação B ou de aplicações localizadas em outros servidores, como acontece com a aplicação C.

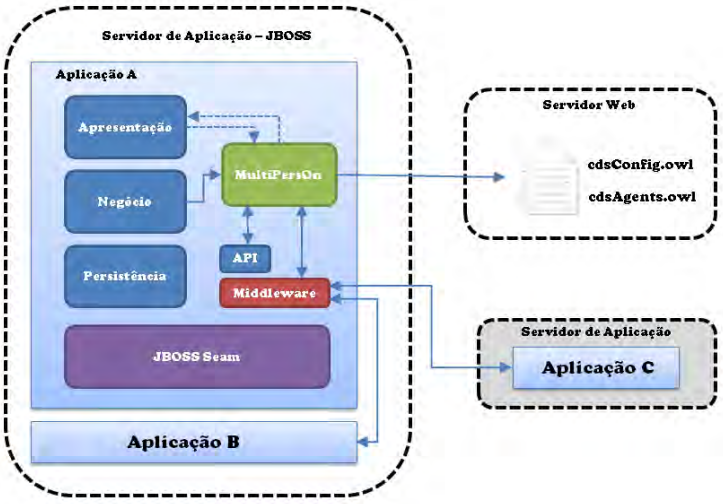

Figura 3.16 - Cenário de utilização do MultiPersOn numa organização de saúde.

Vale ressaltar ainda, que ortogonal a tudo isso, os próprios agentes podem fazer uso de soluções de comunicação, adicionando-as como recursos de infra-estrutura de cada um dos agentes. Isso possibilitará que um determinado agente possa, por exemplo, acessar bases de dados remotas, consultar *web services*, entre outros serviços.

Este cenário, exposto através da Figura 3.16, representa de maneira bastante real, um possível cenário de uma organização de saúde, onde varias soluções utilizando tecnologias diferentes e, por vezes, geograficamente distantes, precisam interoperar para trocar informações de saúde sobre os pacientes.

3.5.3 Considerações finais

Através dessa arquitetura, do cenário de utilização e da definição das ontologias, o *framework* propõe a construção de agentes que possam, de fato, suportar uma melhor prestação de saúde, independentemente da tecnologia e do modelo de RES usado, com foco na manutenabilidade, flexibilidade e reatividade do CDS.

A flexibilidade e portabilidade do MultiPersOn se dá pelo isolamento da lógica do agente da sua estrutura, que é descrita pela ontologia, bem como, a separação dos elementos tecnológicos responsáveis por seu mapeamento e execução. O *framework* em si, é desprovido de qualquer mecanismo de mapeamento e execução, entretanto fornece as condições necessárias para a elaboração destes, independentemente da tecnologia utilizada.

Vejamos a seguir a descrição detalhada da estrutura semântica utilizada para descrever os agentes, bem como, das suas características, a fim de possibilitar sua criação e tirar proveito de todos os recursos disponíveis.

Capítulo

4

O Modelo Semântico do MultiPersOn

"A pintura deve ser uma poesia muda e a poesia uma pintura que fale."

Plutarco

Antes que o agente possa realizar sua lógica para auxiliar no processo de atendimento em saúde, outras etapas devem ser alcançadas, ele deve definir suas caracteristicas e os recursos utilizados, através das ontologias do MultiPersOn, possibilitando assim ser construído que, por sua vez, permitirá, enfim, sua execução. Essa utilização de ontologias visa desacoplar os agentes dos aspectos tecnológicos, a fim de permitir uma maior liberdade na sua construção, bem como, aumentar o grau de especialização, através da separação da lógica dos agentes em relação às suas características e as características dos recursos utilizados por eles ou do próprio MultiPerson.

4.1 As ontologias do MultiPersOn

Como o funcionamento do MultiPersOn está condicionado ás informações referentes a cada um dos seus agentes e aos recursos utilizados por eles, criou-se duas ontologias, a ontologia de configuração de recursos e a ontologia dos agentes do MultiPersOn. Apesar de distintas e com propósitos específicos, elas se relacionam intimamente, pois a ontologia dos agentes usa a ontologia de configuração de recursos, sendo então separadas apenas para facilitar a gerência destas informações.

Elas foram definidas utilizando a linguagem de descrição de ontologias OWL, através da ferramenta Protegé. Para fins didáticos, elas aparecem expostas a seguir simplificadas e modeladas em UML. Entretanto, caso necessite-se de uma

pesquisa mais detalhada sobre elas, ambas podem ser encontradas na íntegra no Apêncide A.

A ontologia de recursos do MultiPersOn, além de ser útil na definição dos agentes de CDS, também promove um baixo acoplamento entre o MultiPersOn e as aplicações que interagem com ele, possibilitando-o acessar às informações contidas nessas aplicações de maneira transparente para os agentes do *framework*.

Para tanto, essa ontologia visa definir uma estrutura para as aplicações que interagem com o MultiPersOn, bem como, os recursos que necessitam ser acessados pelos agentes. Assim, conforme explicitado na Figura 4.1, pela estrutura da ontologia de configuração de recursos, os dados (*Data*) das aplicações serão representados por uma estrutura maior, denominada conceito (*Concept*), que, além do dado em si, possui meta-informações que o descrevem. A classe *Data* não possui nenhuma restrição na ontologia, permitindo assim que qualquer informação seja representada, funcionando apenas como uma classe sinalizadora para indicar ao *framework* que aquele indivíduo representa um dado.

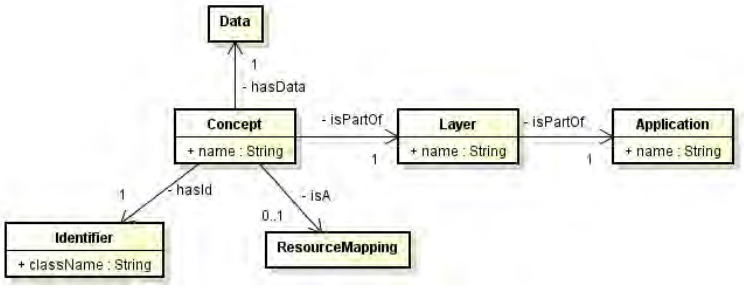

Figura 4.1 – Representação das classes da ontologia de configuração de recursos.

Além do dado, o conceito também contém: um atributo *name* para sua descrição textual, um relacionamento com a camada (*Layer*) da aplicação no qual ele pertence, um relacionamento opcional com a classe *ResourceMapping* para indicar se o conceito é um recurso que precisa ser mapeado ou se é apenas um conceito abstrato e um relacionamento com o identificador responsável por saber como o conceito deve ser identificado pelo *framework*. O relacionamento com a camada da aplicação serve, além de indicar a qual camada de determinada aplicação o recurso pertence, para organizar melhor os conceitos, agregando valor a eles e permitindo que sejam tratados pelo *framework* de maneira diferenciada. O relacionamento com a classe *ResourceMapping*, além de indicar se o conceito é um recurso que será mapeado para algum agente, facilita e garante que a definição dos agentes seja feita corretamente, evitando que os conceitos abstratos sejam utilizados erroneamente. E, por fim, o relacionamento com o identificador serve

para informar ao MultiPersOn qual subclasse da interface *IConceptIdentifier,* da Figura 3.13, será responsável por tratar corretamente o conceito de maneira a identificá-lo univocamente.

Assim, vamos considerar a hipótese da elaboração de um agente que envie uma mensagem de texto para um profissional de saúde quando a freqüência cardíaca de um paciente indicar que ele está com taquicardia, passando do limiar de 120 batimentos por minuto e alertando ao profissional para que este intervenha. Assumamos também que este profissional trabalha na unidade de terapia intensiva de um hospital qualquer e que a valor da freqüência cardíaca é capturado por um equipamento externo que encaminha de tempos em tempos esse valor para o MultiPersOn.

Antes mesmo de implementar sua lógica, devemos definir na ontologia de configuração de recursos os conceitos que vão ser usados pelo agente para que este possa acessá-los, além dos eventos que são lançados pelo equipamento para o MultiPersOn e vice-versa. Então, devemos primeiro definir a aplicação representada pelo equipamento externo, atribuindo um nome a ela e definindo suas camadas. Em seguida, conforme exemplificado na Figura 4.2, vamos definir o conceito freqüência cardíaca, através da definição do atributo *name*; do dado, isto é, de um indivíduo da classe *Data*; da classe que vai tratar esse indivíduo para identificar o conceito, criando um identificador único para ele; e da associação deste conceito a um indivíduo da classe *ResourceMapping* para indicar que ele é um recurso que vai ser mapeado.

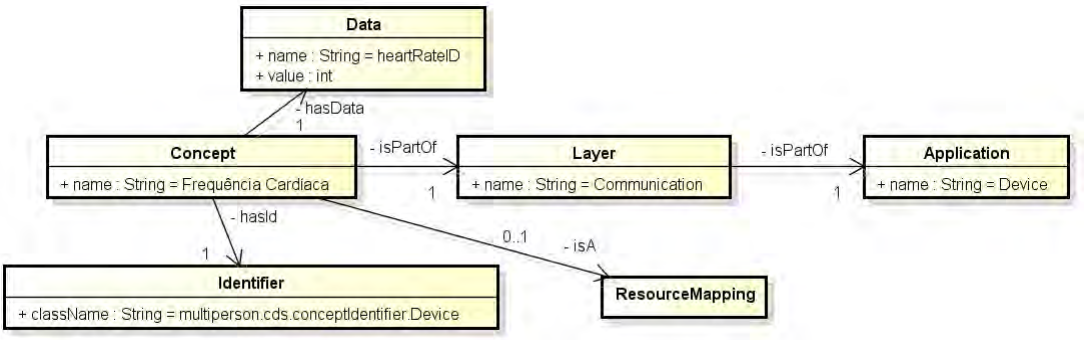

Figura 4.2 – Exemplo de representação um conceito na ontologia de configuração de recursos.

Os conceitos que representam os eventos de comunicação entre o MultiPersOn e o equipamento externo também precisam ser definidos de forma similar. Mas como estes não representam recursos que serão acessados pela lógica do agente, ou seja, são apenas conceitos abstratos, não devem ser associados ao indivíduo da classe *ResourceMapping*. Agora que todos os conceitos foram definidos, estes podem ser usados para a definição do agente na ontologia de agentes.

Para definir um agente, na ontologia de agentes do MultiPersOn, um indivíduo da classe *Agent* deve ser criado, Figura 4.3, juntamente de uma breve descrição do seu propósito, relacionando-se com os indivíduos dos seguintes tipos:

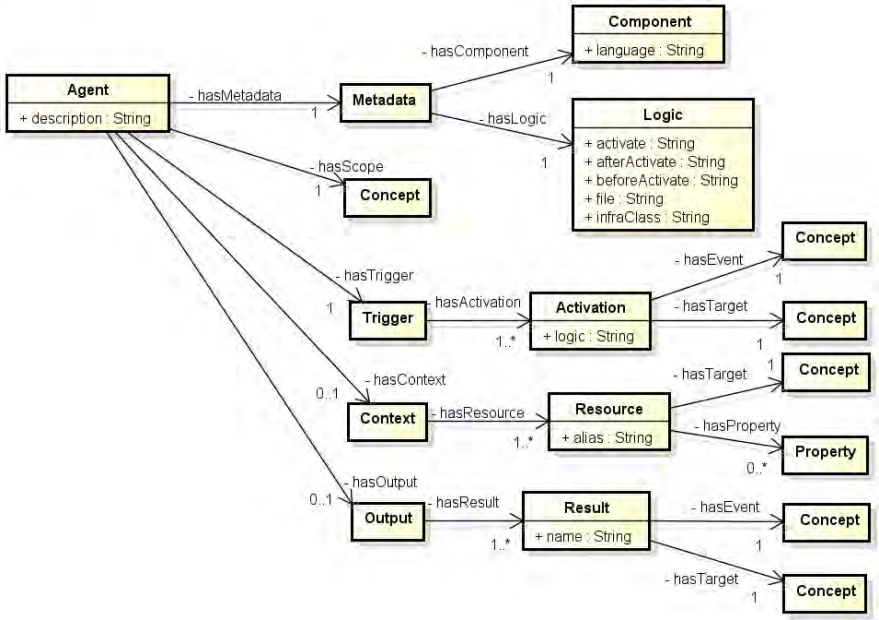

Figura 4.3 – Diagrama UML das classes da ontologia de agentes do MultiPersOn.

- hasMetadata **Metadata**: De caráter obrigatório e guarda as meta-informações, contidas nos indivíduos da classe *Logic* e *Component*, necessárias para o MultiPersOn conseguir construir o agente.
- hasLogic **Logic**: Responsável por indicar qual arquivo contém a lógica a ser executada pelo agente, bem como, quais métodos dentro dessa lógica devem ser executados e se o agente utiliza algum elemento de infraestrutura externo. Devem ser fornecidos obrigatoriamente.
- hasComponent **Component**: Também de caráter obrigatório e informa ao MultiPersOn qual a linguagem responsável pela execução do agente.
- hasScope **Concept**: Estabelece o tipo do escopo do agente, sendo, por isso, também obrigatório. O escopo é importante na delimitação da aplicação de um determinado contexto, possibilitando a associação correta dos recursos utilizados na lógica dos agentes
- hasTrigger **Trigger**: Define o conjunto de ativadores, responsáveis por ativar o agente para ser executado. Como não faz sentido criar um agente que nunca será ativado, sua definição é obrigatória, devendo conter pelo menos um ativador.

- hasActivation **Activation**: Fornece um ativador para executar os métodos do agente contidos nos atributos *activate*, *beforeActivate* e *afterActivate* da classe *Logic*. Além disso, é composto obrigatoriamente por um relacionamento com um evento e um dado, ambos da classe *Concept*.
- hasContext **Context**: Define o conjunto de informações contextuais utilizas na lógica interna do agente, sendo então opcional de acordo com suas necessidades.
- hasResource **Resource**: Descreve uma informação contextual útil para o agente, que será mapeada na aplicação por meio do conceito e das suas propriedades para ser injetada na sua lógica interna, através do seu *alias*.
- hasOutput **Output**: Define o conjunto de saídas do agente para interagir com a aplicação. Como podem existir agentes que fiquem apenas colhendo dados e não reajam, este relacionamento é opcional.
- hasResult **Result**: Fornece um resultado de interação do agente com aplicação que é definido através de um nome e composto obrigatoriamente por um relacionamento com um evento e um dado, ambos da classe *Concept*.

Após saber qual a função de cada parte da estrutura da ontologia de agentes, podemos retomar a definição do agente responsável pelo envio de mensagens de texto em caso de ocorrência de freqüência cardíaca maior do que 120 batimentos por minuto. A Figura 4.4 mostra como esse agente poderia ser definido, começando pela descrição do propósito do agente, depois a definição dos metadados, como, a linguagem que será utilizada pelo agente, bem como, o nome do arquivo e o procedimento que será invocado. É necessário também definir o escopo do agente, as informações relativas à ativação do agente para saber qual evento vai atuar sobre certo conceito, as informações contextuais que serão utilizadas dentro da lógica do agente e, por fim, as informações relativas a interação do agente com o ambiente externo, utilizando para isso certo evento sobre um determinado conceito.

Agora que já definimos os conceitos utilizados pelo agente e também sua própria estrutura, podemos nos concentrar na sua lógica interna. De acordo com as informações fornecidas na configuração do agente, sua lógica é composta por um código na linguagem Groovy contido no método *execut*() que fora salvo no arquivo AgentFC.groovy. Assim, a lógica desse agente poderia ser apresentada conforme a Figura 4.5.

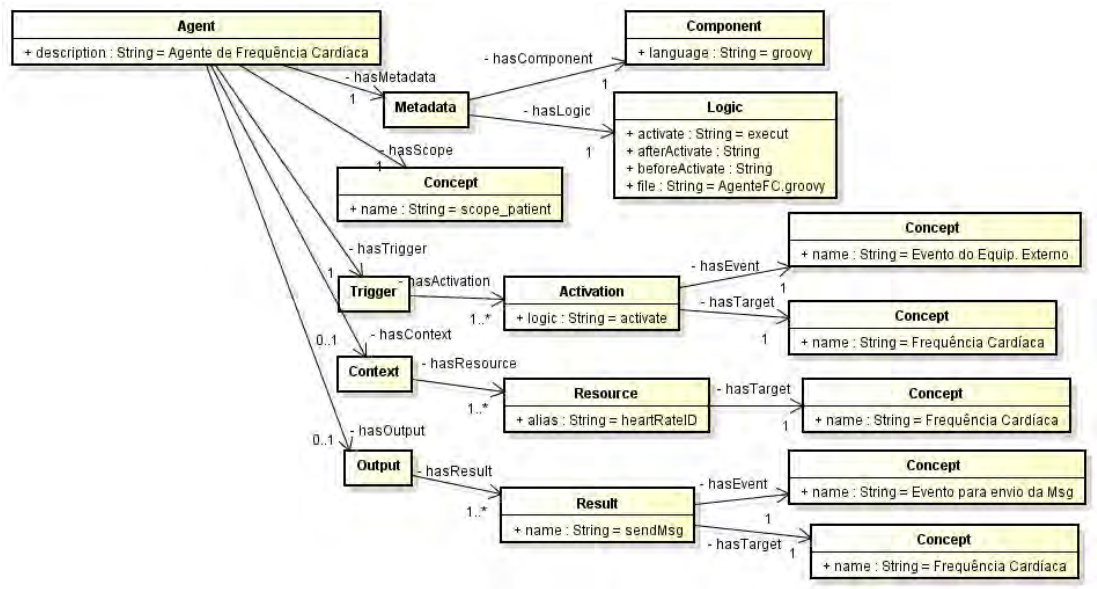

Figura 4.4 - Exemplo da representação de um agente na ontologia de agentes do MultiPersOn.

Neste código, vemos que o elemento *Data* foi injetado dentro do código, podendo ser acessado através do *alias* atribuído a ele na estrutura do contexto do agente. Assim, buscou-se o valor do atributo *value* da instância de *Data* e verificou-se que caso seu valor seja superior a 120 batimentos por minuto, uma mensagem acerca do conceito freqüência cardíaca, contendo a frase: "Valor anormal de freqüência cardíaca!" seria destinada a um equipamento externo. Esse equipamento externo, por sua vez, se encarregaria de enviar uma mensagem de texto do profissional de saúde que estivesse tratando deste paciente.

```
public execut()
{
    def heartRateValue = heartRateID.find { it.name=="value" }.value

    if (heartRateValue > 120 ) {
        msg = "Valor anormal de frequência cardíaca!"
        ret = returnFactory.getInstance(sendMsg, msg)
        cdsOutputList << ret
        return cdsOutputList
    }

}
```

Figura 4.5 - Trecho de código na linguagem Groovy para o tratamento da freqüência cardíaca de um paciente.

É importante que se destaque que alguns detalhes mais específicos foram omitidos, no exemplo acima, a fim de permitir uma melhor compreensão da utilização e da finalidade do agente. Por exemplo, supomos que o valor armazenado da freqüência cardíaca estava em batimentos por minutos, mas nada impede que fosse outra unidade. Necessitando, para tanto, que a classe *Data* do conceito 'Freqüência Cardíaca' contives-

se a informação da unidade de medida utilizada, possibilitando assim que a lógica do agente pudesse tratá-la de maneira adequada. Ou mesmo, com relação ao valor propriamente dito, nada impede que este valor limítrofe fosse obtido através das informações que definem o conceito ou até mesmo por meio de um elemento de infraestrutura mais robusto que utiliza técnicas de IA para retornar informações mais contextualizadas.

Além disso, alguns aspectos do contexto da aplicação não foram mencionados, como a associação do conceito 'Freqüência Cardíaca' com o paciente e do paciente com o profissional de saúde que está responsável por ele. Para lidar com isso, duas coisas devem ser feitas: primeiramente deve-se implementar um contexto para a aplicação, conforme explicitado na Figura 3.12, para que, depois, todos os escopos possam ser corretamente implementados, isto é, as informações relativas a aplicação possam ser encontradas apropriadamente, através do entendimento da estrutura dos objetos da aplicação com que se está interagindo.

4.2 Implicações das ontologias sobre os agentes

Os agentes do MultiPersOn, apesar de serem concebidos para realizar ações de baixa complexidade, eles foram projetados para ter liberdade e serem customizáveis, possibilitando assim se moldar aos mais variados propósitos das organizações de saúde. A forma com que os agentes irão atuar vai depender, então, dos seus ciclos de vida e dos seus escopos.

4.2.1 Ciclo de vida

A priori, todo agente tem seu ciclo de vida parecido, inicialmente eles são construídos, então, instanciados mediante a ocorrência de um evento que os ativem, são executados e por fim destruídos. Entretanto, durante a sua execução, cada um pode assumir uma grande variedade de configurações distintas, segundo suas definições na ontologia.

Essa variação decorre da divisão conceitual da execução da lógica dos agentes em três estados: *beforeActivate*, *activate* e *afterActivate*. Os estados *beforeActivate* e *afterActivater* são estados opcionais dos agentes, com a finalidade de realizarem ações internas, respectivamente, antes e depois da sua execução. Como exemplo, pode-se utilizá-los para carregar e salvar dados dos agentes num banco de dados do MultiPersOn, mas não é possível, através deles, interagir com as aplicações interligadas ao MultiPersOn. Já o *activate* representa o estado de execução e interação dos agentes do *framework* com as aplicações, sendo de caráter obrigatório para qualquer agente.

A configuração desses estados nos agentes se faz por meio dos elementos *Logic* e *Activation* da ontologia dos agentes. O primeiro define a existência e a assinatura do método para cada estado, enquanto que o segundo estipula as condições para sua ativação. Assim, no elemento *Logic*, os atributos *beforeActivate*, *activate* e *afterActivate* devem ser definidos para indicar qual é o procedimento dentro da lógica do agente responsável por cada um desses estados, ressaltando-se a obrigatoriedade do *activate*. Enquanto que o elemento *Activation* determina quais estados serão executados quando algum evento em certo conceito da aplicação ocorrer.

Como a existência desses estados e a forma de serem ativados podem variar bastante, uma grande quantidade de possibilidades pode ser definida para ser executada por cada agente. Assim, como exemplo, a Figura 4.6, explicita quatro possíveis percursos do ciclo de vida de execução de um mesmo agente.

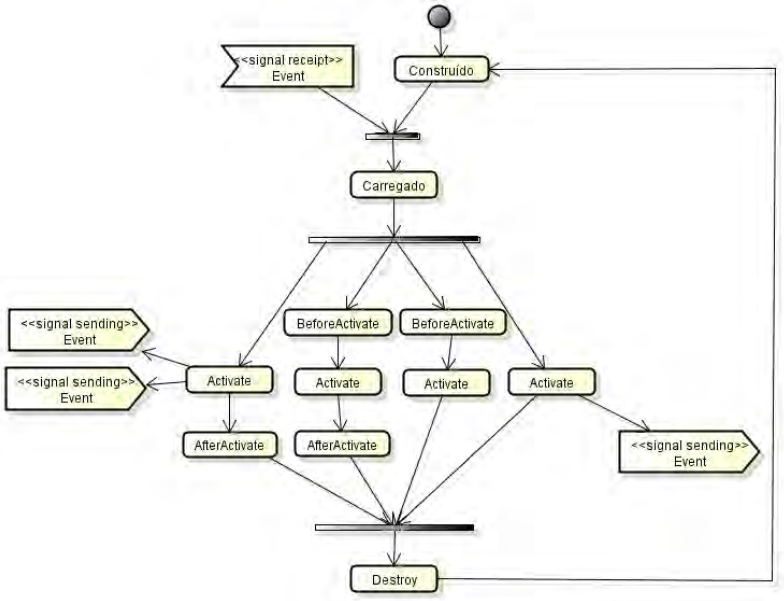

Figura 4.6 - Exemplo de quatro possíveis fluxos de ciclos de vida de um agente do MultiPersOn.

Desta maneira, um agente poderia ao ser ativado por um determinado evento seguir o percurso mais simples deles, mais a direita, no qual não possui o estado *beforeActivate* nem *afterActivate*, tendo, antes de ser destruído, seu estado *activate* executado, gerando um evento para a interação do agente com uma aplicação qualquer e retornando para o estado construído. Depois, ao ser ativado por outro evento poderia seguir o próximo percurso, da direita para esquerda, no qual não tem o estado *afterActivate*, sendo, antes de ser destruído, seus estados *beforeActivate* e *activate,* respectivamente, executados, antes de ir para o estado construído novamente. Mais tarde, ao ser ativado por outro evento, poderia utilizar o percurso seguinte onde todos os três estados são execu-

tados seguidamente, indo para o estado destruído e retornando para o construído. E por fim, ao ser ativado por mais outro evento, o agente é executado, gerando dois eventos distintos para a interação do agente com uma ou mais aplicações, indo em seguida para o estado *afterActivate*, antes de ser destruído para, enfim, retornar ao estado construído.

Essa gama de possibilidades é muito importante, pois alem de permitir a criação de agentes bastante robustos com comportamentos bem específicos, fomenta a criação de agentes bastante diversos que possam atuar das mais diversas formas.

4.2.2 Escopo

Além de seu ciclo de vida, a escolha do escopo dos agentes também pode ser determinante para o seu propósito de utilização. Por isso, quatro diferentes escopos foram elaborados, visando atender diferentes propósitos dentro do contexto das diferentes organizações de saúde. Cada um deles visa delimitar um determinado contexto, para possibilitar a associação correta dos recursos da aplicação utilizados na lógica dos agentes.

- *Document*: Escopo que objetiva ter acesso aos dados do documento (podendo este ser dados clínicos ou dados demográficos) que interagem com o *framework*, possibilitando que as informações referentes aos documentos possam ser utilizadas no contexto do agente.
- *Pacient*: Escopo que visa ter acesso aos dados do paciente já que estejam armazenadas no sistema, possibilitando que as informações previamente conhecidas referentes aos dados clínicos do paciente possam ser utilizadas no contexto do agente.
- *User*: Escopo que tem como finalidade acessar os dados do usuário, assim, informações relativas ao usuário, como profissão ou setor de atuação, podem ser usadas dentro do contexto do agente.
- *Application*: Escopo que visa ter acesso aos dados acerca da aplicação, possibilitando a aplicação realize certas ações com as aplicações que interagem com o *framework,* caso certas condições sejam satisfeitas.

4.3 Considerações finais

O entendimento das ontologias do MultiPersOn é muito importante para usufruir plenamente dos recursos do framework que apesar de seus agentes serem projetados para processamentos não tão complexos, ao atuarem no domínio de saúde, lidando com dados tão valiosos, tornam-se extremamente úteis e poderosos na melhoria da qualidade do atendimento em saúde.

A seguir encontraremos alguns exemplos de agentes criados através da definição dessas estruturas, possibilitando vislumbrar o processo de criação de agentes como um todo.

Capítulo

5

Resultados

"O verdadeiro homem mede a sua força, quando se defronta com o obstáculo."

Antoine de Saint-Exupéry

O MultiPersOn foi implementado e posto em prática junto com o OpenCTI para verificar sua aplicabilidade num cenário real de aplicação. Para que a multi-aplicabilidade do *framework* fosse evidenciada, alguns agentes foram elaborados de maneira que pudessem atuar das mais variadas formas. Entretanto, para agilizar esse processo de elaboração de agentes uma ferramenta de manipulação do modelo semântico na elaboração de agentes de CDS para o MultiPersOn foi criada.

5.1 Ferramenta para elaboração de agentes de CDS

Apesar de isolar os aspectos técnicos, referentes à estrutura dos agentes, dentro da lógica dos agentes, sua descrição através da ontologia é um processo laborioso e maçante. Para tentar reduzir isto, foi criada então uma ferramenta que visa agilizar e facilitar o processo de criação dos agentes de CDS para o MultiPersOn.

Essa ferramenta necessita das duas ontologias do modelo semântico para extrair e inserir as informações necessárias para a criação dos agentes. Conforme pode ser visto na Figura 5.1, inicialmente deve-se, então selecionar o endereço dessas ontologias para que então o resto da interface fique habilitada.

Caso deseje definir um agente a partir do começo, isto é, através da elaboração dos conceitos utilizados por ele, deve-se então primeiramente carregar a ontologia de configuração de recursos do MultiPersOn. Feito isto, a aba na interface gráfica ficará habilitada, possibilitando a definição de novos conceitos e evidenciando no canto direito a árvore gráfica da estrutura da organização.

Por exemplo, para elaborar o agente descrito no capítulo anterior, que envia uma mensagem de texto para um profissional de saúde quando a freqüência cardíaca de um paciente indicar que ele está com taquicardia, precisamos definir os conceitos utilizados por ele, representados na Figura 4.2. Para isto, podemos utilizar a ferramenta segundo a figura abaixo, criando assim um novo elemento *Data*, *Identifier*, *Layer*, *Application* e *ResourceMapping* de forma automatizada.

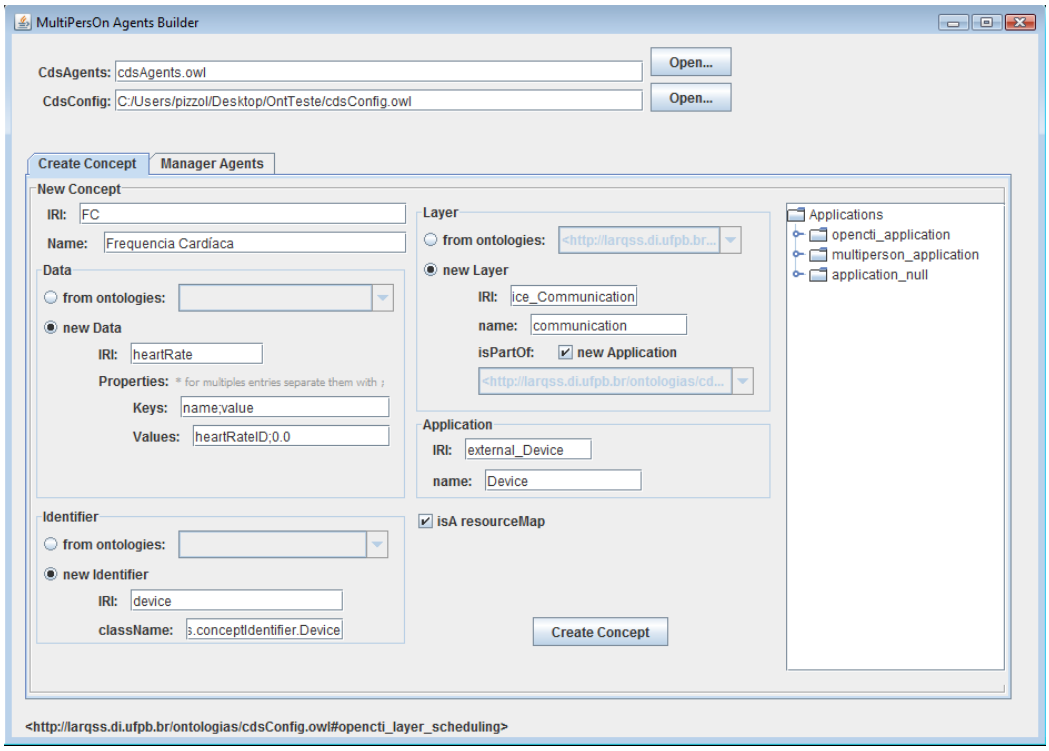

Figura 5.1 – Tela da elaboração do conceito da Figura 4.2.

Da mesma forma, devemos criar os demais conceitos utilizados pelo agente. Criados todos esses conceitos, podemos iniciar a criação do agente, através do carregamento da ontologia de agentes pela ferramenta, indo na aba *Manager Agents* e atribuindo os elementos básicos dos agentes. Isto é, o número de ativadores, de elementos contextuais e de resultados de interação, juntamente com o tipo do escopo e o nome do agente. Ao clicar no botão *Build Agent*, será possível visualizar o agente na árvore de componentes à esquerda, conforme a ilustração abaixo.

Todo agente recém-criado possui seus relacionamento com conceitos *null*, devendo então os seus conceitos ser atribuídos corretamente, conforme a Figura 4.4, para que eles possam ser utilizados.

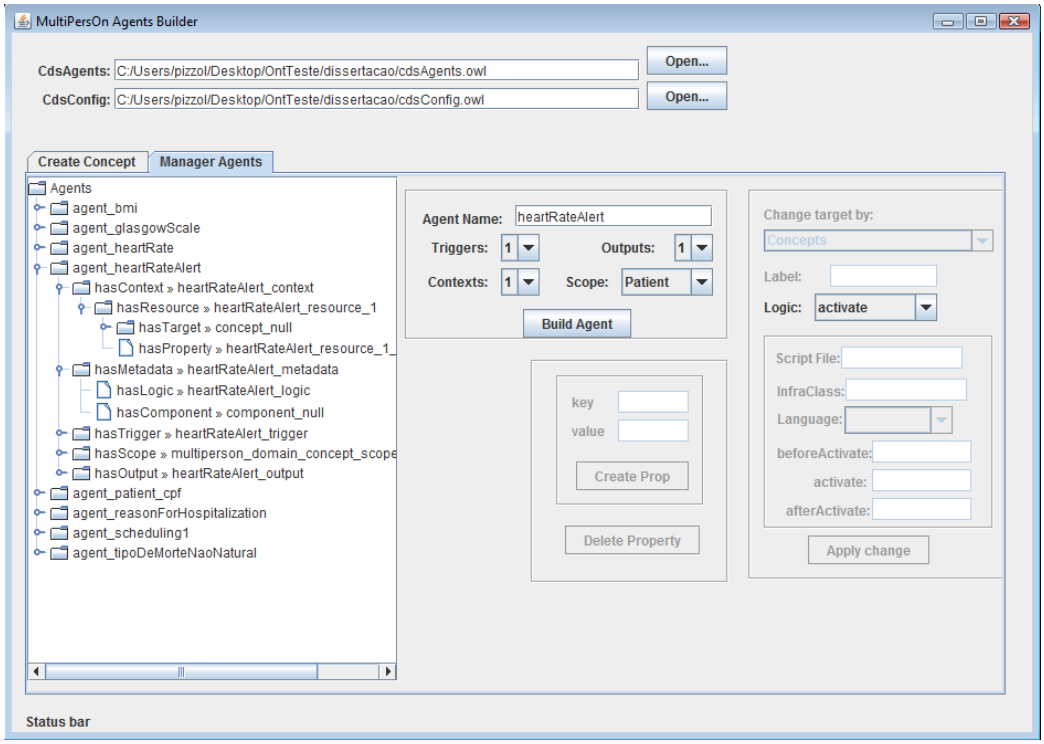

Figura 5.2 – Tela da criação do agente da Figura 4.4.

Através dessa ferramenta é possível, então, criar os agentes de maneira simplificada, sem que seja preciso entender detalhadamente sobre ontologia, facilitando e agilizando o processo de elaboração dos agentes de CDS do MultiPersOn.

5.2 A interação com o OpenCTI

Para ser utilizado num contexto real, o MutilPersOn foi aplicado ao OpenCTI, seguindo o seu cenário de utilização e resultando no esquema representado pela Figura 5.3.

Ambos funcionam dentro do servidor da aplicação JBOSS AS 5.1, juntamente com o *framework* JBOSS Seam, que visa facilitar a construção de sistemas, através da integração transparente das mais variadas tecnologias necessárias em cada uma das suas camadas, isto é, desde a persistência até a interface de interação com o usuário.

Na Figura 5.3, podemos perceber que o OpenCTI e o MultiPersOn compartilham o mesmo container dentro do servidor de aplicação, possibilitando a comunicação entre eles, por meio do acesso direto aos dados de domínio no OpenCTI (representado pela linha contínua) ou por meio da troca de eventos entre a interface gráfica do OpenCTI e o MultiPersOn (representados pelas linhas tracejadas).

Para permitir que o *framework* seja notificado dos eventos da interface gráfica e acesse os dados de domínio do OpenCTI, é necessário que a estrutura do OpenCTI seja

modelada, através da ontologia de configuração de recursos do MultiPersOn. Possibilitando que os agentes descritos na ontologia de agentes sejam ativados corretamente e podendo assim acessar as informações contextuais essenciais na suas lógicas internas.

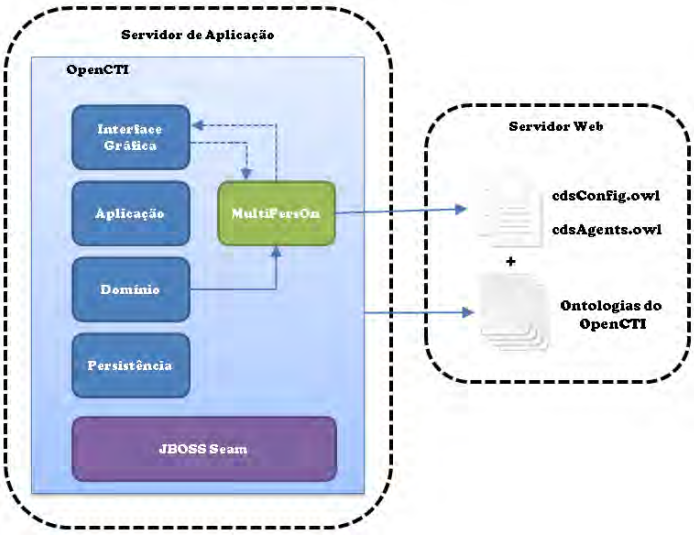

Figura 5.3 - Interação do MultiPersOn com o OpenCTI.

Da mesma forma, para que o OpenCTI seja notificado dos eventos de CDS e reaja de forma correta, é necessário que o OpenCTI implemente, de maneira fiel, os eventos de CDS descritos na ontologia de configuração de recursos do MultiPersOn. Fomentando, assim, a eficácia e corretude das mensagens de CDS geradas pelo *framework*.

5.3 Cenários de uso

Para validar este *framework,* foi elaborado um caso de uso contendo algumas funcionalidades, conforme mostrado na Figura 5.4, que poderiam ajudar na melhoria da qualidade da prestação de saúde no contexto do OpenCTI e que estão alinhadas com os objetivos descritos no Objetivo 8 do item 1.2. Cada uma dessas funcionalidades se baseou em um método de CDS diferente, tendo como objetivo evidenciar a ampla aplicabilidade do *framework*.

O método dos validadores de campos é responsável por checar se os valores entrados estão de acordo com certos parâmetros, como limites fisiológicos ou limites inferidos a partir do contexto do paciente (SERAFIM *et al.*, 2010). O método para o cálculo de fórmulas visa agilizar o preenchimento do RES, calculando automaticamente campos com fórmulas previamente conhecidas. O conjunto de sugestões gera uma lista de sugestões para os campos do RES com base no seu histórico de preenchimento. Já o

método de aviso de atividades alerta sobre atividades temporais que devem ser realizadas. A leitura de dispositivo captura dos dispositivos externos valores para serem utilizados no RES. E por fim, o método de informações adicionais mostra informações extras sobre determinados campos, a partir de bases de dados remotas.

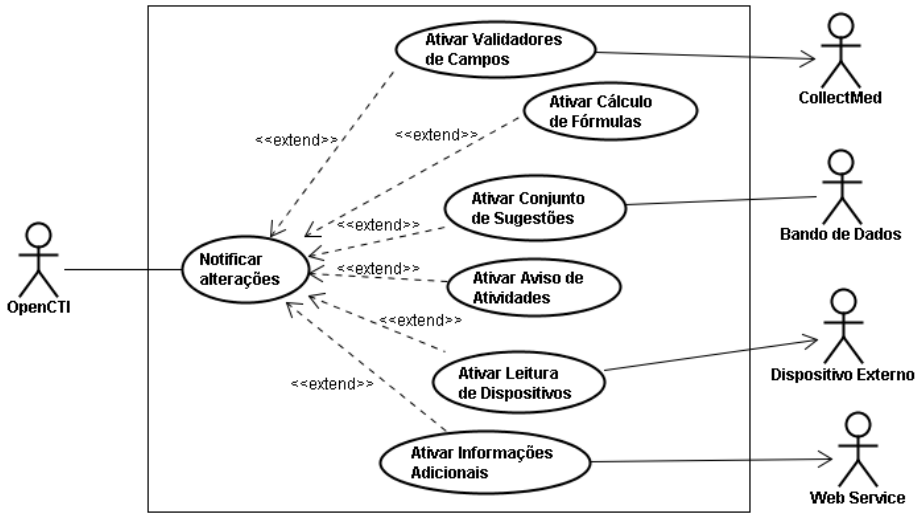

Figura 5.4 – Diagrama de caso de uso de funcionalidades dos agentes do MultiPersOn no contexto do OpenCTI.

Com base nesse diagrama, alguns agentes foram elaborados, a fim de validar a arquitetura e a aplicabilidade do MultiPersOn. Abaixo, cada um deles será descrito de forma sucinta, necessitando consultar os apêndices A e B para obter uma descrição mais detalhada sobre as descrições dos agentes nas ontologias e os códigos utilizados na sua lógica interna. Todos os códigos dos agentes aqui expostos foram elaborados, utilizando a linguagem Groovy, apesar de que, como o MultiPersOn faz uso do JSR-223 como mecanismo de execução da lógica dos agentes, qualquer outra linguagem suportada pelo JSR-223 poderia ter sido usada.

5.3.1 Validadores e Cálculo de Fórmulas

Dois agentes de validação foram criados com o intuito de comprovar a utilização do *framework* para esse propósito. Como boa parte da interface do OpenCTI é gerada automaticamente com base nas informações da sua ontologia de conceitos, conforme descrito na seção 2.6, mas há também alguns elementos tradicionais que são gerados com base em Java beans, então criou-se um agente para cada uma dessas realidades, ratificando sua aplicabilidade nos mais variados cenários. Além desses dois agentes de vali-

dação, foi criado também um agente para o calculo automático do IMC (índice de massa corpórea).

Apesar de diferir no propósito, ambos seguem o mesmo esquema de interação que está representado pela figura abaixo, onde um evento notifica o MultiPersOn que, por sua vez, ativa o agente responsável por tratar o evento e por fim o agente retorna um valor que é enviado para a aplicação que interagiu com o MultiPersOn por meio de um outro evento.

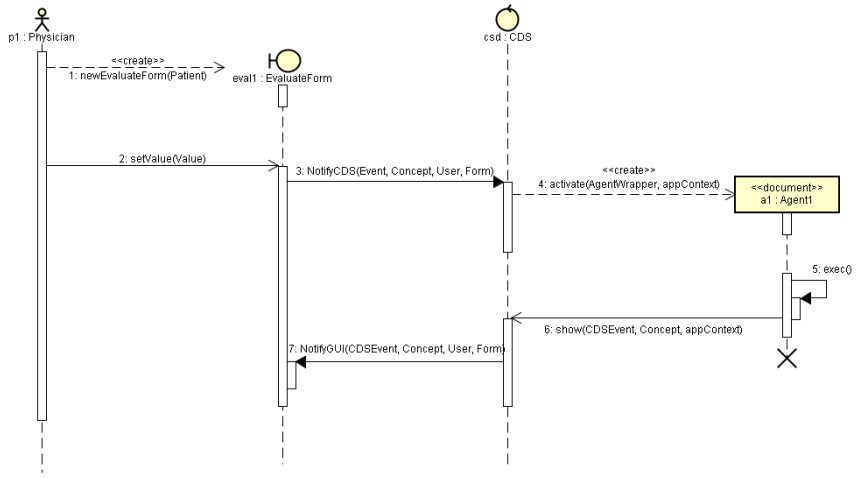

Figura 5.5 – Diagrama de seqüência das interações de um agente com escopo *document*.

Sabendo como os sistemas interagem um com o outro, vejamos agora cada um dos agentes mais detalhadamente, começando pelos validadores de campos.

O agente validador da interface tradicional verifica se o CPF (Cadastro de Pessoa Física) digitado para o cadastro de um novo paciente é um CPF válido, informando usuário caso haja ou não algum erro. Assim, o usuário (que no caso por se tratar de um cadastro de um novo paciente deve ser um profissional de saúde ou algum assistente deste) ao preencher as várias informações requeridas no processo de inserção de um paciente, como mostra a Figura 5.6, poderá contar com o auxílio do MultiPersOn para verificar a corretude do CPF do paciente.

Apesar do algoritmo de verificação de CPF ser simples e bastante conhecido, esta verificação é deveras importante para o OpenCTI, visto que este será aplicado a UTI adulta do HULW e juntamente com o número de matrícula do SUS poderá identificar univocamente o paciente. O armazenamento de forma equivocada desse dado pode impedir ou dificultar a localização de informações vitais para que os profissionais de saúde possam tratar o paciente da melhor forma possível.

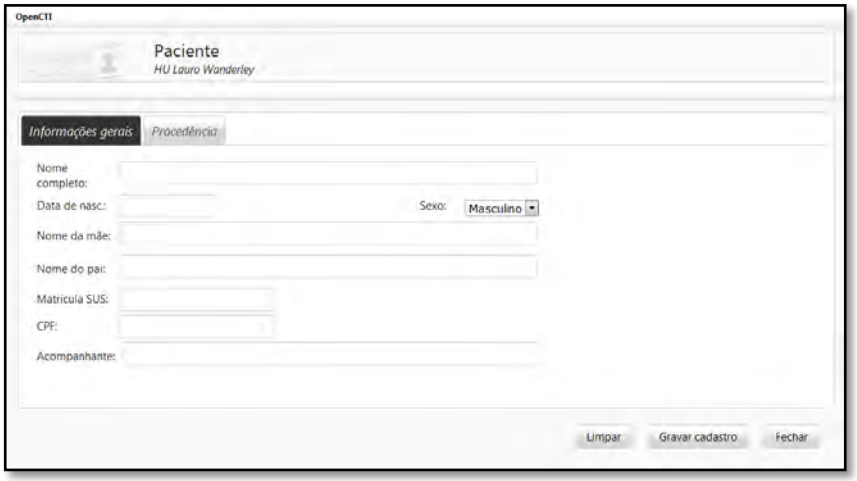

Figura 5.6 - Tela de inserção de um novo paciente no OpenCTI.

Desta forma, caso o usuário digite um valor de CPF inválido como ocorre com 0101010101, então uma mensagem de erro será mostrada de acordo com a figura abaixo.

Figura 5.7 – (a) Mensagem de erro gerada pelo agente validador de CPF do MultiPersOn, informando ao usuário que o CPF está incorreto. (b) Mensagem de aceitação gerada pelo agente validador de CPF do MultiPersOn, informando ao usuário que o CPF está correto

Da mesma maneira, caso o usuário digite um valor de CPF válido como ocorre com 11111111111, então uma mensagem de aceitação será mostrada de acordo com a figura abaixo.

Mas para que o validador possa atuar conforme mostrado, uma série de requisitos devem ser satisfeitos, começando pela definição do agente no modelo semântico do MultiPersOn. Como a única informação necessária para a validação do CPF é o próprio valor do CPF, sendo esta ação desencadeada quando o mesmo é preenchido, ele deve conter no seu modelo semântico apenas um ativador e um contexto para ser usado na sua lógica interna. Entretanto, como ele pode atuar validando ou invalidando o CPF, necessita agora ter dois resultados de interação, sendo um para cada caso. Assim, uma visão resumida das definições do agente de validação do CPF no modelo semântico do MultiPersOn seria a seguinte:

Scope: *Documento*

Trigger:

Activation_1:

Target: *CPF do paciente*

Event: *Perda de Foco*

Context:

Resource_1:

Target: *CPF do paciente*

Output:

Result_1:

Target: *CPF do paciente*

Event: *cdsError*

Result_2:

Target: *CPF do paciente*

Event: *cdsSuccess*

Além disso, para que o agente consiga realizar essa validação, é preciso que haja uma troca de eventos entre o MultiPersOn e OpenCTI, além da atuação correta da GUI para agir da forma especificada pelos eventos gerados pelo MultiPersOn.

```
if(cpf.length() == 11)
{
   ...
    if(ok) {
            println "Agent - SUCCES"
            def ret = returnFactory.getInstance(successMsg);
            cdsOutputList << ret;
        }else {
            println "Agent - ERROR"
            def msg = "CPF inválido!";
            def ret = returnFactory.getInstance(errorMsg, msg);
            cdsOutputList << ret;
        }

        return cdsOutputList

   }else {
        println "Agent - TAMANHO INVALIDO"
        def msg = "CPF inválido!"
        def ret = returnFactory.getInstance(errorMsg, msg)
        cdsOutputList << ret
        return cdsOutputList
   }
```

Figura 5.8 - Trecho do código da lógica interna do agente validador de CFP.

Dentro da lógica interna desse agente validador, fora o algoritmo de validação de CPF propriamente dito, há o trecho de código que deve indicar ao MultiPersOn como este deve agir, isto é, em que condições qual dos resultados de interação deve ser aplicados e quais as mensagens que estes devem conter.

Em contrapartida ao agente da interface tradicional, o agente validador da interface gerada automaticamente verifica se o valor da freqüência cardíaca do RES do paciente corrente está na faixa limítrofe aceitável de acordo com valores fisiológicos sabidos de antemão ou de acordo com a faixa de valores aceitáveis para o contexto do paciente. Ao contrário do exemplo anterior em que os valores de referência são fixos e retirados da literatura, o agente validor da interface dinâmica precisa obter informações específicas do contexto do paciente, a exemplo, do setor e da idade do paciente, visto que pacientes de UTI infantil tem valores de freqüência cardíaca diferentes na média dos valores de pacientes da UTI adulta. Este problema pode ser sanado, através do uso da solução proposta Serafim (2011) que utiliza técnicas de data minig para obter informações contextuais dos pacientes.

Independente da maneira que for escolhida para criar o agente, a mensagem de erro deve ser lançada quando o usuário estiver preenchendo o RES do paciente e este digitar um valor fora da faixa aceitável. Um trecho da tela da ficha de admissão do RES no OpenCTI pode ser visto na através da Figura 5.9.

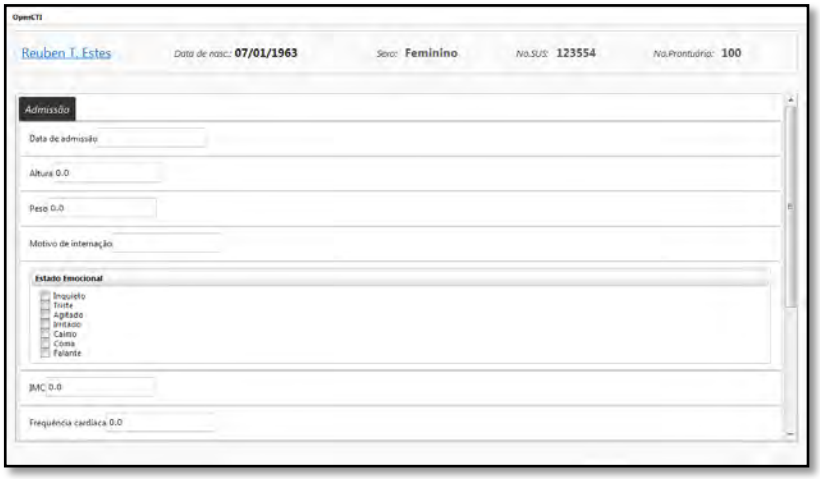

Figura 5.9 - Tela da ficha de admissão do RES do paciente no OpenCTI.

Desta forma, caso o usuário digite um valor de freqüência cardíaco fora da faixa aceitável, como ocorre com o valor 1100.00, uma mensagem de erro será mostrada de acordo com a Figura 5.10.

Assim, de forma análoga a descrição do modelo semântico do validador da interface tradicional, este validador apresenta basicamente a mesma estrutura de modelo semântico, variando apenas a quantidade de resultados de interação, isto é, há apenas um, que é a mensagem de erro quando o valor estiver fora da faixa aceitável. Isto ocorre, devido à natureza do dado, pois apesar da freqüência cardíaca ser sempre obrigatória, alguns valores mesmo estando dentro da faixa de valores aceitáveis podem estar e-

quivocados, impossibilitando assim afirmar que esta informação está correta, entretanto, quando este valor estiver fora da faixa fisiológica aceitável é garantido afirmar que o valor está equivocado.

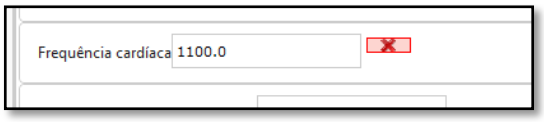

Figura 5.10 - Mensagem de erro gerada pelo agente validador de freqüência cardíaca do MultiPersOn, informando ao usuário que o valor inerido está fora da faixa aceitável.

Em relação ao código da lógica interna do agente de freqüência cardíaca, dependendo do recurso utilizado para realizar a tomada de decisão, pode-se utilizar um limite fixo para a validação, conforme mostrado na Figura 5.11. Este valor limítrofe pode ser definido no próprio código ou na própria definição do conceito por meio da ontologia de configuração de recursos do MultiPersOn.

```
if(valor < MIN || valor > MAX){
    println "Agent - ERROR"
    def msg = "Frequencia Cardiaca ${frequencia} fora dos valores limitrofes!!";
    def ret = returnFactory.getInstance(errorMsg, msg);
    cdsOutputList << ret;
    return cdsOutputList
}
```

Figura 5.11 - Trecho do código da lógica interna do agente validador de freqüência cardíaca, utilizando um faixa limítrofe fixa.

Ou, então, pode-se utilizar uma solução mais robusta, como a solução contextual desenvolvida por Serafim (2011), o CollectMed, que possibilita uma análise mais refinada com base em critérios específicos, como por exemplo, através da idade do paciente e do setor em que o paciente está internado. Perceba que para tirar proveito do CollectMed, deve-se indicar que este é elemento de infra-estrutura do agente e então referenciá-lo através da variável *cdsInfraStructure*, na sua definição na ontologia de agentes.

```
List<String> IRIs = {"age", "sector"};
List<Object> values = {valor};

if( cdsInfraStructure.hasValidator(IRIs, "UTI") ) {
    if ( !cdsInfraStructure.isValid(IRIs, values, "UTI") {
        println "Agent - ERROR"
        def msg = "Frequencia Cardiaca ${frequencia} fora dos valores aceitáveis!!";
        def ret = returnFactory.getInstance(errorMsg, msg);
        cdsOutputList << ret;
        return cdsOutputList
    }
}
```

Figura 5.12 - Trecho do código da lógica interna do agente validador de freqüência cardíaca, utilizando o CollectMed como solução para definição de um faixa limítrofe.

Por fim, vejamos agora o agente para o cálculo do IMC do paciente. Como se trata de informações sobre o paciente este atuará sobre a ficha de admissão do paciente mostrada na Figura 5.9. Entretanto, diferentemente do agente validador de freqüência cardíaca, este agente necessita de múltiplas informações do paciente e sua atuação em um conceito diferente dos quais serão utilizados para realizar o cálculo.

Para o cálculo do IMC deve-se ter o valor da altura e do peso do paciente, desta maneira, o modelo semântico do agente de cálculo do IMC pode ser visto resumidamente da seguinte forma:

Scope: *Documento*
Trigger:
    Activation_1:
        Target: *Peso do paciente*
        Event: *Perda de Foco*
    Activation_2:
        Target: *Altura do paciente*
        Event: *Perda de Foco*
Context:
    Resource_1:
        Target: *Peso do paciente*
    Resource_2:
        Target: *Altura do paciente*
Output:
    Result_1:
        Target: *IMC do paciente*
        Event: *cdsFulfill*

Assim, sempre que o valor do peso ou da altura do paciente for alterado, o agente será ativado para calcular automaticamente o valor do IMC e preencher o campo IMC do paciente com este valor, como pode ser visto na Figura 5.13.

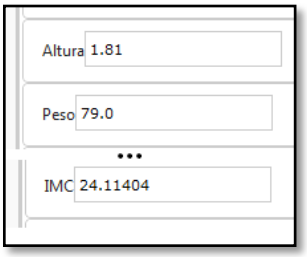

Figura 5.13 – Valor do IMC calculado e preenchido no campo específico, através do agente de cálculo de IMC.

Para realizar esse cálculo, o valor do peso do paciente é dividido pela altura ao quadrado do paciente, conforme pode ser visto no trecho de código abaixo.

```
if (pesoValor > 0 && alturaValor > 0) {
    msg = pesoValor / (alturaValor ** 2)
    println "Agent - FULFILL novoBMI: "+msg
    ret = returnFactory.getInstance(fulfillMsg, msg)
    cdsOutputList << ret
    return cdsOutputList
}
```

Figura 5.14 - Trecho do código da lógica interna do agente do cálculo do IMC.

Através da criação desses agentes de validação e cálculo automático de fórmulas pode-se verificar a aplicabilidade do framework para este propósito, bem como, testar e validar diversos aspectos arquiteturais e do modelo semântico do MultiPersOn, como: escopo de documentos, a ativação por mais de um critério, o contexto e o resultado com mais de um campo e o resultado sendo aplicado a um campo diferente do de ativação e de contexto do agente. Além disso, verifica-se a interoperabilidade por meio da inserção de recursos de infra-estrutura para a realização de processamentos mais elaborados dentro da lógica dos agentes.

5.3.2 Conjunto de Sugestões

Como solução para o componente que lista um conjunto de valores sugeridos de acordo com um método qualquer, foi criado um agente que sugere nomes de acordo com a freqüência de utilização, do mais usado para o menos usado. Esse agente, diferentemente do esquema anterior de interação dos agentes, tem a peculiaridade de utilizar mais de um procedimento para realizar seu objetivo, como está representado na Figura 5.15.

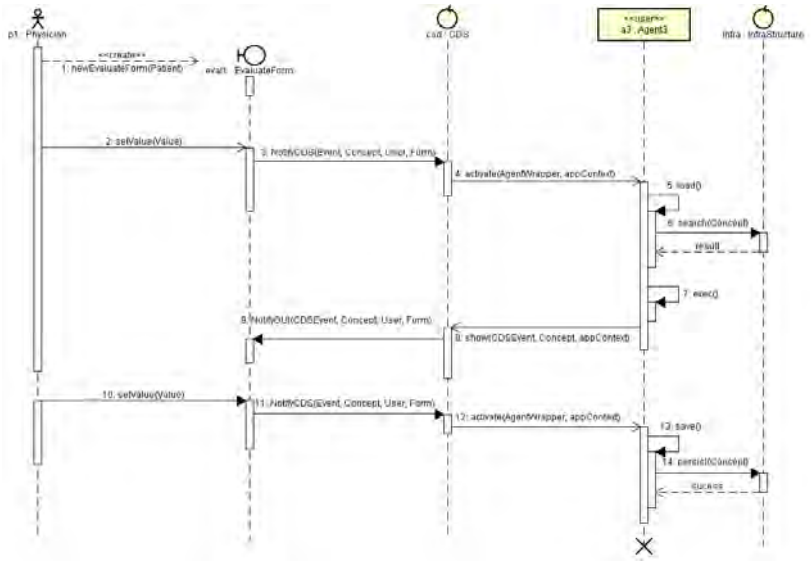

Figura 5.15 – Diagrama de seqüência das interações de um agente com escopo *user*.

Como o objetivo aqui não é validar um método específico de sugestão de valores, mas sim, evidenciar a aplicabilidade de qualquer método com esse propósito, quando um nome é digitado, este é salvo e um contador associado a ele é gerado. Permitindo assim, que quando um novo nome estiver sendo digitado, o agente será ativado para carregar os nomes anteriormente selecionados na ordem do mais freqüente para o menos freqüente.

Então como pode ser visto na figura abaixo, os nomes ao invés de serem sugeridos em ordem alfabética, foram sugeridos de acordo com a freqüência de seleção sendo o primeiro nome, 'Ivo Pitanguy', o mais selecionado anteriormente.

Figura 5.16 –Sugestões de preenchimento gerado pelo agente de conjunto de sugestões.

Para que a ordem dos nomes seja mostrada de acordo com a freqüência de seleção, uma estrutura simplificada de classificação foi elaborada e referenciada como um recurso de infra-estrutura no modelo semântico do agente. Além desta informação, os seguintes aspectos do modelo semântico são também importantes:

Scope: *User*
Trigger:
    Activation_1:
        Target: *Nome do residente*
        Event: *Tecla pressionada*
    Activation_2:
        Target: *Residente Selecionado*
        Event: *Clique do mouse*
Context:
    Resource_1:
        Target: *Nome do residente*
        Property: *source – document*
    Resource_2:
        Target: userName
Output:
    Result_1:
        Target: *Lista de nomes*
        Event: *cdsSuggestion*

Como o agente de sugestão visa manipular os dados oriundos do usuário, ou seja, os nomes mais recorrentes digitados por ele, então seu escopo deve ser o de usuário. Entretanto, na sua lógica interna ele necessita de uma informação referente ao documento que está sendo preenchido, por isso este possui uma propriedade com o valor 'document' no atributo 'source'. Além disso, o agente possui dois ativadores, um deles que ativa o método de carregar a lista de nomes quando o usuário digita um valor e outro que ativa o método de salvar o nome digitado pelo usuário quando este clica no botão 'selecionar'. Por fim, o resultado da interação será uma lista com os nomes mais recorrentes para serem sugeridos ao usuário.

Para que isto seja possível, segundo dito anteriormente, necessita-se de dois métodos, um para carregar a lista de nomes em ordem de freqüência e outro para salvar o novo nome selecionado, conforme pode ser visualizado na Figura 5.17.

```
public load() {

    def anterior = cdsInfraStructure.select(userName, "residentList");

    def residentList = [];
    anterior.each{ residentList.add(it.fieldAttribute)};

    def ret = returnFactory.getInstance(suggestionMsg, residentList);
    cdsOutputList << ret;
    return cdsOutputList;
}

public save() {

    cdsInfraStructure.update(userName,"residentList", residentSelected);
}
```

Figura 5.17 - Código da lógica interna do agente do conjunto de sugestões.

Desta maneira, o agente de sugestão tenta agilizar ao atendimento de saúde, aumentando a qualidade durante o preenchimento do RES do paciente, pois este evita equívocos de digitação que pode ser danosos, senão letais, ao paciente quando aplicados a campos, como seleção de pacote de drogas para tratamento, indicação de sintomas, entre tantos outros.

5.3.3 Aviso de Atividades

Ainda no intuito de mostrar que o *framework* serve a um propósito bastante genérico de funcionalidades, também foi criado um agente para gerar avisos de atividades para os profissionais de saúde. Desta forma, atividades recorrentes que acontecem em determinadas horas do dia ou em dias específicos podem ser agendadas para gerarem notificações para os respectivos usuários do sistema. Como estas atividades são agendadas, o

esquema de interação difere de todos os outros, iniciando a partir do MultiPerson e terminando numa notificação ao usuário, conforme expressado na Figura 5.18.

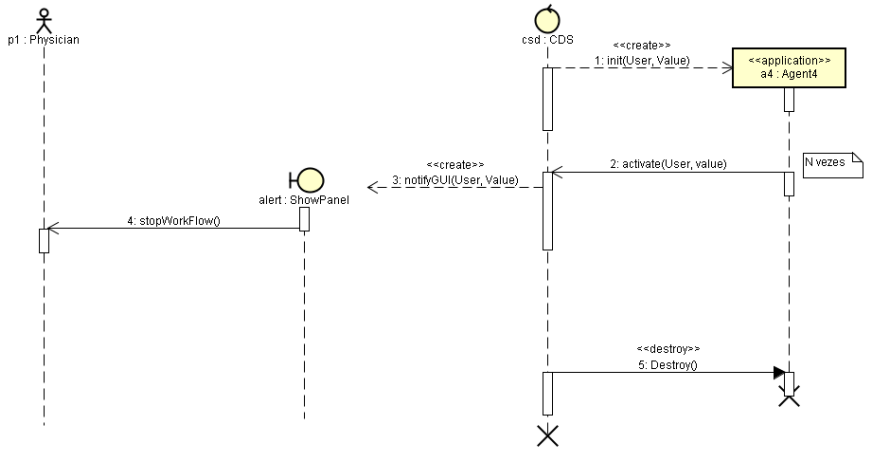

Figura 5.18 – Diagrama de seqüência das interações de um agente com escopo *application*.

Assim para que o usuário tomasse ciência desses eventos, um painel foi colocado no canto inferior direito do OpenCTI, mostrando todos as atividades para o usuário em questão, segundo indicado na figura abaixo.

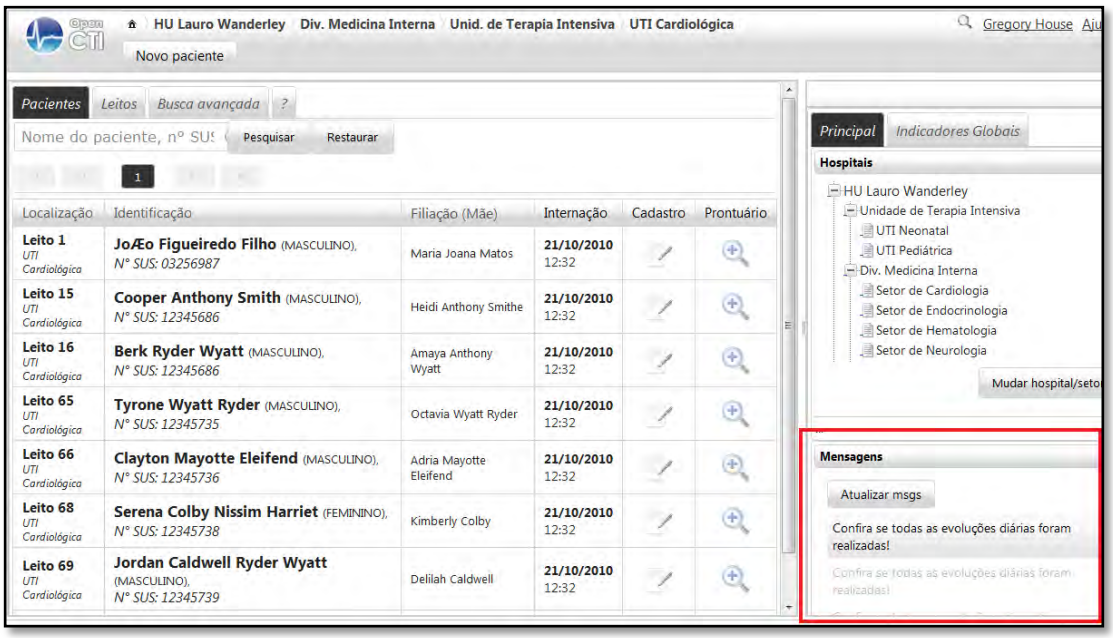

Figura 5.19 – Tela com a indicação do painel reservado para mostrar as atividades geradas para um usuário específico pelo agente de notificação de atividades.

A periodicidade dessas notificações é baseada na informação expressa no modelo semântico num formato de expressão periódica de tempo do UNIX e SOLARIS, denominada CRON, que é utilizado pelo *framework* de escalonamento e agendamento de tarefas, Quartz. Este formato permite de uma maneira simplificada e robusta, expres-

sar eventos temporais bastante complexos, usando para tanto cinco campos e alguns caracteres especiais que representam o evento de acordo com os segundos, minutos, horas, dias e meses do ano, respectivamente. Assim, para que as mensagens mostradas na figura acima sejam geradas de hora em horas, devemos atribuir ao agente o escopo de aplicação, pois a ação é iniciada diretamente por intermédio do MultiPersOn e usar o modelo semântico do agente da seguinte forma:

Scope: *Application*
Trigger:
Activation_1:
Target: *Escalonador*
Event: *"* * 1 * * ?"*
Context:
Resource_1:
Target: *Lista de usuários*
Output:
Result_1:
Target: *Painel de Mensagens*
Event: *cdsMessage*

Por fim, para que a mensagem de fato seja mostrada aos usuários do sistema, basta utiliza o código abaixo. Mensagens mais elaboradas poderiam ser facilmente criadas, checando o papel de cada usuário e atribuindo mensagens específicas para cada um deles de acordo com suas funções organizacionais.

```
public execut() {

    def msg = "Confira se todas as evoluções diárias foram realizadas!";
    def map = [:];
    userList.each{ map.put(it.username,msg); it.addMessage(msg); };

    def ret = returnFactory.getInstance(infoMsg, map);
    cdsOutputList << ret;
    return cdsOutputList

}
```

Figura 5.20 - Código da lógica interna do agente de notificação de atividades.

5.3.4 Leitura de Dispositivos

Como não foi possível utilizar nenhum dispositivo para coleta de sinais externos, permitindo assim integrá-lo com o RES do OpenCTI, nenhum agente para esse propósito foi criado. Entretanto, isto pode ser feito facilmente adicionando a API de um dispositivo qualquer ao PATH do projeto que executa o MultiPersOn, importando as dependências

e, por fim, usando normalmente os recursos da API do dispositivo. Isto é possível, graças ao JSR-223 que permite o uso pela lógica dos agentes de todas as bibliotecas que se encontram no seu PATH.

Mesmo que nenhum agente tenha sido criado, o exemplo abaixo mostra como esse recurso poderia ser utilizado dentro da lógica de um agente para facilitar o cálculo de um freqüência de distribuição. Para isso, podemos utilizar a API Commons Math, depois de adicioná-la no PATH do projeto, e importar a classe *Frequency* para usá-la normalmente dentro do código.

```
import org.apache.commons.math.stat.Frequency;

exec()
{
    Frequency f = new Frequency();
    f.addValue(1);
    f.addValue(1);
    f.addValue(1);
    f.addValue(2);
    f.addValue(-1);
    println f.getCount(1);   // displays 3
    println f.getPct(1);  // displays 0.6

}
```

Figura 5.21 - Código que poderia ser utilizado para calcular um curva de distribuição, através de um API específica para esse propósito.

Desta forma, independentemente da linguagem utilizada e da API necessária para a comunicação do MultiPersOn e o dispositivo externo, verifica-se que o *framework* suporta esse tipo de interação, validando mais um vez sua vasta aplicabilidade.

Uma outra solução também não testada por falta de dispositivos clínicos para a coleta de dados seria criar um agente que utilizasse os padrões de troca de informações que são amplamente difindudos da HL7 para, através de uma API específica, obter automaticamete dados desses dispositivos.

5.3.5 Informações adicionais

Por fim, um agente foi criado para adicionar informações de acordo com o CEP digitado pelo usuário. Este agente ao perceber que se trata de um código postal brasileiro procura numa base de dados as informações referentes ao CEP; como logradouro, bairro, cidade e estado, preenchendo os respectivos campos automaticamente caso estas informações sejam encontradas.

Além disso, o agente segue o cenário de utilização da Figura 5.5, sendo que este realiza uma consulta a uma base de dados remota, por meio do seu recurso de infraestrutura, para obter as informações adicionais através do numero do CEP. A priori,

esta consulta seria através de um Web Service, mas devido a dificuldades em encontrar um serviço deste tipo em Web Service, foi utilizado um site que possui essas informações, sendo então essas informações carregadas via método GET do protocolo HTTP.

Exposto isto, vejamos como foi elaborado o modelo semântico para que o agente de informações adicionais possa completar todos os campos com as informações oriundas da base de dados remota.

Scope: *Documento*
Trigger:
    Activation_1:
        Target: *CEP*
        Event: *Perda de Foco*
Context:
    Resource_1:
        Target: *País*
    Resource_2:
        Target: *CEP*
Output:
    Result_1:
        Target: *Rua*
        Event: *cdsFulfill*
    Result_2:
        Target: *Bairro*
        Event: *cdsFulfill*
    Result_3:
        Target: *Cidade*
        Event: *cdsFulfill*
    Result_4:
        Target: *Estado*
        Event: *cdsFulfill*

Como atua sobre as informações presente no documento corrente, o agente possui escopo de documento e é ativado, após o usuário digitar o valor do CEP. Para confirmar se o valor do código postal é do Brasil, o agente necessita também da informação referente ao campo país. Além disso, para preencher os campos com as informações retiradas a partir do CEP os campos necessitam ser inseridos no resultado de interação do agente.

Assim, o agente utiliza o código abaixo para, a partir do domínio http://viavirtual.com.br, obter as informações a partir do CEP. Isto é feito através do

processamento da separação da string de resposta do site, para separar as valores corretos dos campos.

```
public exec() {

    if(country == "brasil" || country == "BRASIL") {

        if(cep.length() == 8) {

            def url = "http://viavirtual.com.br/webservicecep.php?cep=" + cep;
            def stringInteira = cdsInfraStructure.getHttpRequest(url)
            ...
            if(listaDeStrings.size() == 5) {

            def street = returnFactory.getInstance(streetMsg, [listaDeStrings[0]])
            def neighborhood = returnFactory.getInstance(neighborhoodMsg, [listaDeStrings[1]])
            def city = returnFactory.getInstance(cityMsg, [listaDeStrings[2]])
            def state = returnFactory.getInstance(stateMsg, [listaDeStrings[4]])

            cdsOutputList << street << neighborhood << city << state
            println "Agent - lista de returns: ${cdsOutputList}"
            return cdsOutputList
          }
       }
    }
    def street = returnFactory.getInstance(streetMsg, "")
    def neighborhood = returnFactory.getInstance(neighborhoodMsg, "")
    def city = returnFactory.getInstance(cityMsg, "")
    def state = returnFactory.getInstance(stateMsg, "")

    cdsOutputList << street << neighborhood << city << state
}
```

Figura 5.22 – Trecho do código da lógica interna do agente de informações adicionais que obtêm informações complementares através do CEP.

Apesar da busca de informações complementares a partir do CEP ser uma atividade trivial, este agente comprova a viabilidade da realização de funcionalidades similares mais que poderiam agregar maior valor ao OpenCTI. Uma destas funcionalidades seria a partir de um determinado campo, a exemplo de diagnóstico clínico, buscar numa base de dados de saúde, informações relevantes, como protocolos de tratamento ou drogas com melhores respostas para combater uma enfermidade, fomentando assim a melhoria da qualidade do atendimento em saúde.

5.4 CDS Menu

Por fim, foi desenvolvido no OpenCTI uma interface gráfica para facilitar o acompanhamento e o gerenciamento dos agentes criados e em execução.

Para tanto, alguns usuários que possuem as devidas permissões para esse gerenciamento podem acessar o menu de CDS do OpenCTI, que fica localizado na barra de atalho do canto inferior direito, conforme está explicitado na Figura 5.23. A primeira opção, Ajuda, apresenta informações sobre as funcionalidade do menu de CDS. A segunda opção, Atualizar CDS, possibilita atualizar os agentes de CDS do MultiPersOn, por meio da chamada ao método *updateAgents*() da interface ICDSManager da Figura

3.3. Por último, a opção CDSConfig abre uma nova janela com as lista de agentes construídos e os agentes em execução do MultiPersOn.

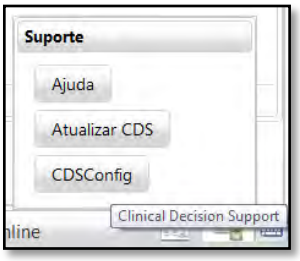

Figura 5.23 – Menu de CDS utilizado para facilitar a gerencia dos agentes do MultiPersOn.

Para obter essa lista de agentes construídos e em execução, o OpenCTI faz uso dos métodos *getLoadedAgents*() e *getRunningAgents*(), respectivamente, da interface ICDSManager da Figura 3.3. Através disto, então é possível montar essa lista, conforme pode ser visualizado na figura abaixo.

Agentes carregados

AgentUID	Hashcode	Description	Código	Linguagem	Escopo
1111111111	30686277	Agente criado para relembrar atividades diárias.	scheduling1.grvy	groovy	APPLICATION
1111111111	13092881	Agente criado para validação da frequencia cardíaca.	heartRateValidator.grvy	groovy	DOCUMENT
1111111111	2094154	Agente criado para calcular a escala de Glasgow.	glasgowScaleCalculator.grvy	groovy	DOCUMENT
1111111111	31407598	Agente criado para sugestão de valores.	residentList.grvy	groovy	USER
1111111111	22736605	Agente criado automaticamente, rever detalhes.	reasonForHospitalization.grvy	groovy	DOCUMENT
1111111111	16249322	Agente criado para a validação do CPF,	cpfValidator.grvy	groovy	DOCUMENT
1111111111	2835559	Agente criado para calcular o IMC.	bmiCalculator.grvy	groovy	DOCUMENT
1111111111	9951159	Agente criado automaticamente, rever detalhes.	tipoDeMorteNaoNatural.grvy	groovy	DOCUMENT

Agentes em execução

Usuário	AgentUID	Hashcode	Description	Linguagem	Escopo
Sem resultados.					

Figura 5.24 – Lista com os agentes construídos e em execução do MultiPerson gerada para facilitar a gerencia desses agentes.

Com esse recurso é possível gerenciar os agentes de maneira a verificar quais agentes são construídos e quais estão sendo executados num dado instante de tempo.

Capítulo

6

Considerações Finais

"Cada um é forjador do seu próprio sucesso."

Caio Salústio

Após visto os pormenores referentes aos aspectos arquiteturais e semânticos do MultiPerSon e os resultados obtidos com a utilização deste junto ao OpenCTI, podemos então analisar criticamente os pontos fortes e fracos da solução, bem como, comparar com as vantagens e e desvantagens das demais soluções similares. Feito isto, podemos então concluir quais as contribuições deste trabalho para a área e vislumbrar trabalhos futuros oriundos deste.

6.1 Discussão

O MultiPersOn é um *framework* que visa fomentar as soluções de apoio à decisão clínica durante o preenchimento do RES, através do uso de ontologias como modelo semântico para a geração dessas soluções. Como não foca num método específico de apoio à decisão e foi projetado para os mais variados fins, diz-se que ele é multipropósito. E ainda, tem a característica de ser personalizável, isto é, customizável através da associação das vantagens oriundas do uso conjunto de ontologias, eventos e da separação entre a lógica dos agentes e as informações acerca do próprio agente. Assim, ele pode ser utilizado em uma variedade de cenários, onde haja sistemas heterogêneos, culturas organizacionais diferentes ou mesmo tecnologias distintas envolvidas.

Para poder afirmar isto, devemos analisar primeiramente os resultados obtidos com o *framework,* suas características de maneira geral e por fim, compará-lo com as outras soluções existentes, a fim, de formular um juízo de valor sobre sua relevância.

Para mostrar seu propósito e sua aplicabilidade alguns agentes foram criados, conforme visto no capítulo anterior. Estes agentes servem para demonstrar que o MultiPersOn pode, de fato, auxiliar a melhoria da qualidade do atendimento em saúde, como ocorre com o agente de validação ao evitar equívocos de digitação, bem como, agilizar o processo de atendimento em saúde, como acontece com o agente de cálculo de fórmulas ao preencher campos automaticamente com bases em fórmulas previamente conhecidas. Isto quando não atuam de forma conjunta, melhorando e agilizando o processo de atendimento, conforme pôde ser evidenciado com o agente de sugestão de preenchimento, que pode utilizar técnicas de IA para sugerir o valor mais provável com base em outras informações, evitando um erro de digitação e acelerando o preenchimento do campo, através do fornecimento de informações personalizadas para o profissional de saúde. Ademais, o agente de notificação de atividades permite, além de agilizar o processo de atendimento em saúde, influenciar e alterar o fluxo de trabalho dos profissionais de saúde, bem como notificá-los sobre exames e atividades que podem ter relevância num dado momento. Por fim, o agente de leitura de dispositivo possibilita uma interação com diversos dispositivos, comprovando a abrangência da solução aqui proposta.

Como o MultiPersOn é fracamente acoplado com os dispositivos que interagem com ele, bem como, os da interface gráfica, faz-se necessário haver uma correta correspondência entre os eventos enviados para e a partir do *framework* com os eventos especificados no seu modelo semântico. Apesar dessa abordagem repassar a responsabilidade do envio e do tratamento dos eventos para a camada de interface gráfica , isto garante uma maior liberdade na utilização dos agentes de CDS do MultiPersOn, possibilitando que sejam usadas as mais variadas tecnologias de maneiras diferentes, conforme a necessidade específica de cada organização de saúde.

Apesar de tudo isso, o *framework* apresenta também desvantagens. A primeira deles diz respeito à dependência a tecnologia JBoss Seam 2.2. Toda a arquitetura do MultiPersOn foi baseada nas soluções oferecidas por essa tecnologia, assim, a instanciação dos seus componentes, o lançamento dos eventos e o escalonador de tarefas, todos são dependentes do JBoss Seam, necessitando que a instituição de saúde que fará uso do MultiPersOn utilize essa tecnologia, o que pode acarretar num aumento do grau de heterogeneidade dos sistemas usados por essa instituição.

Apesar dos agentes testados no MultiPersOn terem lógicas bastante simples, por lidarem com dados sensíveis, como os dados de saúde de pacientes, sua utilização pode, de fato, melhorar a qualidade e a agilidade no atendimento em saúde, evitando,

por exemplo, equívocos dos profissionais de saúde durante o preenchimento do RES e conseqüentemente reduzindo as chances de erros no cuidado ao paciente.

Além disso, duas características são marcantes no MultiPersOn. Seu cenário de utilização e a separação da lógica interna dos agentes das informações que os definem.

O cenário de utilização do MultiPersOn, através da comunicação por eventos descritos nas ontologias e o acesso direto aos dados de domínio das aplicações, trouxeram uma série de vantagens ao *framework*. Conseguindo-se obter uma ortogonalidade do *framework* com relação às aplicações, isto é, uma desassociação entre as partes, de maneira que seja possível, até mesmo, desativar o MultiPersOn sem prejuízos operacionais para o funcionamento das aplicações. Alcançando, também, um alto grau de flexibilidade de interação, devido à facilidade de comunicação com os outras aplicações. E ainda, o acesso direto aos dados das aplicações, viabiliza uma integração mais transparente, evitando a necessidade de replicar as informações contidas nas aplicações.

A separação entre a descrição das características dos agentes, por meio das ontologias, e a codificação das suas lógicas internas, favorece a manutenibilidade, a flexibilidade e a especificidade do *framework*. Visto que facilita a adaptação às constantes mudanças da área de saúde e às diferenças organizacionais, possibilitando seu uso de variadas formas, sem implicações na sua lógica interna e, ao mesmo tempo, permite a especialização das atividades de gerência dos mecanismos de CDS. Assim, profissionais distintos podem ser utilizados, com o intuito de aperfeiçoar a qualidade da prestação de saúde, tendo o especialista de domínio a responsabilidade de apenas desenvolver e validar novos métodos de CDS, enquanto que o especialista do MultiPersOn se encarregaria de realizar os ajustes para acrescentar esses novos métodos e manter o MultiPersOn funcionando.

E ainda, como o MultiPersOn foi projetado para ser de propósito geral, deixando para o especialista no MultiPersOn a tarefa de criar os agentes com propósitos específicos, a única amarração do *framework* com o domínio de saúde está na utilização dos escopos: *document*, *user*, *patient*, *application*. Porém, nada impede que o *framework* seja estendido para ser aplicado em outro domínio diferentemente do de saúde, necessitando para isto apenas realizar algumas alterações em relação a esses escopos de acordo com o novo domínio.

Assim, em suma, cada um dos princípios citados no item 3.1, foi buscado e trouxe consigo uma série de vantagens. A busca pela flexibilidade mitigou a elaboração do cenário de interação de maneira de fosse possível apoiar os mais variados ce-

nários dentro de uma organização de saúde. A portabilidade, por sua vez, fomentou a utilização de um mecanismo de execução genérico, ocasionando na construção de agentes que utilizam diferentes linguagens. A interoperabilidade foi responsável pelo uso de ontologias, pois facilitam e organizam a representação das informações específicas de cada organização de saúde, sem impedir a troca ou correlação com informações de outras organizações. E ainda, a manutenabilidade agrevou valor a solução através do uso de duas ontologias com finalidades bem distintas, uma para os recursos utilizados nas aplicações e outra para os agentes do MultiPersOn. O princípio da separação de propósitos resultou na separação entre a lógica interna dos agentes e as informações que decrevem cada um deles, possibilitando a especialização de cada uma das atividades, fomentando também assim a manutenabilidade. E por fim, a reatividade e relevância foram alcançados graças ao modelo de interação baseado em eventos que permite que a informação adequada seja enviada no momento oportuno.

Diante do exposto, podemos, então, comparar o MultiPerson com as outras soluções elencadas no item 2.7. Para fazer um paralelo entre elas, podemos consultar a Tabela 4, na qual as principais características das diferentes soluções de CDS estão expostas de forma simples e clara para facilitar e agilizar o entendimento.

Tabela 4 - Lista com as diversas características das diferentes soluções de CDS.

Característica	Arden Syntax	Gello	MultiPersOn
Licença	Código proprietário	Código proprietário	Código aberto
Objetivo	Recomendações	Guidelines e restrições	Multipropósito
Padrão	ANSI	ANSI Accredited	-
Status	Consolidado	Em desenvolvimento	Em homologação
Implementação	Compilador Open Source para Java (arden2bytecode)	Ambiente de desenvolvimento grátis limitado (MO-Gello)	Open Source
Modelo de Informação	Localmente definidos através de chaves	HL7 v3 RIM	Elementos descritos nas ontologias
Modelo Semântico	Gramática do Arden	Ontologia	Ontologia
Modelo de Inferência	Gramática do Arden	GLIF	Linguagem dos agentes
Limitações	Para cada estrutura de dados diferentes os dados deveriam ser redefinidos através de chaves	Não suporta todos os guidelines	Necessita da modelagem dos elementos estruturais na ontologia

A primeira diferença encontrada entre elas diz respeito à licença e ao objetivo de uso, enquanto o MultiPersOn possui seu código aberto e foi criado de manei-

ra genérica para dar suporte a vários propósitos, o Arden Syntax tem o código proprietário e foi criado para realizar apenas recomendações, similarmente ao Gello que também tem seu código proprietário, mas é útil na elaboração guidelines e restrições.

Em contrapartida, o Arden Syntax é um padrão consolidado, enquanto que o Gello e o MultiPersOn ainda não são. Apesar disso, acredita-se que o Gello será um padrão no futuro, por utilizar outras soluções bastante aceitas no mercado.

Em relação à implementação, pode-se encontrar uma solução de código aberto de um compilador de Arden Syntax para Java, entretanto esta não é uma solução criada e mantida pelo HL7, sendo assim sua confiabilidade questionável. Da mesma forma, pode-se encontrar uma solução grátis limitada para a linguagem Gello, sendo esta também não desenvolvida pelo HL7. Contrapondo-se a elas o MultiPersOn propõe-se a ser uma solução completamente livre, oferecendo os seus recursos abertamente.

A característica mais variante entre as três soluções é acerca do modelo de informação utilizado para representar as informações que serão usadas na lógica das soluções de CDS. O Arden Syntax usa uma estratégia simples e liberal de definição das informações localmente dentro do campo de dados do módulo lógico, através da utilização de chaves para especificar cada conceito. O Gello, por sua vez, utiliza uma solução mais robusta, o RIM (Reference Information Model), que nada mais é que um modelo ontológico proprietário do HL7. Semelhantemente, o MultiPersOn faz uso das suas ontologias para permitir a descrição das informações que serão utilizadas dentro da lógica dos agentes de CDS.

Por causa disto, o modelo semântico do Arden Syntax é sua própria gramática associado com a estruturação dos módulos lógicos, enquanto que no Gello e no MultiPersOn esse modelo é representado pelas ontologias. Assim, para inferir a lógica contida em cada um dos módulos o Arden Syntax usa a própria linguagem, enquanto que o Gello usa o GLIF, uma linguagem interpretável para orientações clínicas, e diferentemente dos demais, o MultiPersOn utiliza a linguagem específica de cada um dos agentes para realizar esse inferência da lógica que deve ser realizada.

Tudo isto, acarreta na limitação, no caso do Arden Syntax, de obrigar a descrição através de chaves da estrutura dos elementos de dados utilizados, conforme a plataform em uso, junto da definição do dado. E ainda, explicita no atributo *action* como e quando as mensagens de CDS devem ser construídas e enviadas ao usuário. Em contrapartida, o Gello, mesmo utilizando o GLIF, não suporta todos os tipos de guidelines, nem lida muito bem com matrizes por usar uma linguagem orientada a

objetos. Por fim, o MultiPersOn possui a limitação de necessitar saber a estrutura dos dados para manipulá-los corretamente dentro da lógica dos agentes de CDS e apesar de ser para um proposito geral, seu projeto foi com foco em agentes de baixa complexidade e inteligência.

Ademais, algumas soluções de CDS (Cho *et al.*, 2010) propõem utilizar a arquitetura orientada a serviços (SOA) na resolução dos problemas com foco no baixo acoplamento e na integração das diversas soluções entre as partes envolvidas, possibilitando oferecer o CDS como um serviço.

Entretanto, a realidade brasileira ainda está aquém da realidade em alguns países do exterior. Aqui a informatização dos sistemas de saúde ainda está num estágio rudimentar, sendo poucas as soluções de RES existentes em uso. Essas poucas que existem foram projetadas para se adequar às suas próprias realidades sem possibilitar de forma simples e clara ser incorporadas por outras instituições de sáude. Junta-se a isso, ainda, a falta de padronização dos termos clínicos utilizados para gerar o CDS, uma vez que a maior parte dos padrões está em inglês, necessitando portanto da sua tradução para poder sem amplamente utilizado no país.

Diante, disso falta um maior esforço conjunto a nível nacional para dar as condições básicas para soluções que visem a integração dos dados e a troca dessas informações entre as diferentes instituições de saúde.

6.2 Conclusão

Com a construção e implementação do MultiPersOn, foi possível elaborar uma solução que ajude a fomentar e consolidar a utilização do apoio a decisão no contexto clínico das instituições de saúde. Isto corrobora com o objetivo geral descrito no item 1.2, bem como, com os objetivos específicos 1 a 6 do mesmo item. Além disso, a integração com o OpenCTI e a criação dos agentes descritos no item 5.3, satisfazem os objetivos específicos 7 e 8 do item 1.2.

Assim, o MultiPersOn permite a criação de agentes multipropósitos de CDS, capazes de se adaptar às mudanças inerentes à área da saúde, tornando possível a disseminação desses agentes pelas organizações de saúde, devido a maior simplicidade de realizar alterações e customizações no *framework*. Como resultado, os esforços podem então ser concentrados na construção de agentes de CDS, sem se preocupar na adequação deles com as aplicações, tentando-se assim diminuir as barreiras tecnológicas para a utilização e disseminação das ferramentas de CDS nos RES.

Comparado ao Arden Syntax o MultiPersOn se destaca, pois é uma solução de código aberto e de propósito geral, com uma melhor modularização em relação ao modelo de informação e do modelo inferência, possibilitando uma melhor organização e separação entre as informações de quais dados serão utilizados das informações de como esses dados devem ser manipulados, bem como, das mensagens de CDS que devem ser tratadas e enviadas ao usuário.

Já comparado ao Gello o MultipersOn distingui-se por se mostrar como uma solução de código aberto e de propósito geral, possibilitando uma maior flexibilidade na criação das soluções tanto em relação a modelo de inferência para interpretar a lógica do agente, visto que o MultiPersOn dá suporte a uma gama delas, como em relação ao modelo de informação que pode ser mapeado para qualquer estrutura do domínio da aplicação.

6.3 Trabalhos Futuros

Apesar de terem sido criados quatro escopos, apenas três deles foram utilizados nos exemplos de criação de agentes. Isto se dá, pois o escopo de paciente foi elaborado pensando no acesso das informações antigas de um dado paciente, entretanto, como o OpenCTI foi desenvolvido em paralelo com o MultiPersOn, não houve possibilidade da utilização desse escopo. Entretanto, caso deseje-se utilizar este escopo, seu uso pode ser feito imediatamente, necessitando apenas para isso de um meio para acessar os dados já salvos de um paciente do OpenCTI.

Um aspecto não abordado aqui e, possivelmente, tema de trabalhos futuros é um estudo sobre a escalabilidade do MultiPersOn. Pois com o aumento do número de agentes seu desempenho pode variar consideravelmente, e também, esses agentes podem conflitar entre si, podendo ter comportamentos diretamente conflitantes ou indiretamente por meio de efeitos emergentes do *framework*.

Ontologias

Apêndice A

Neste apêndice encontra-se a descrição completa das ontologias do modelo semântico do MultiPersOn, isto é, as ontologias de configuração de recursos e de agentes. Vejamos primeiramente a ontologia de configuração.

```
<?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://larqss.di.ufpb.br/ontologias/UIComponent.owl#" >
    <!ENTITY cdsConfig "http://larqss.di.ufpb.br/ontologias/cdsConfig.owl#" >
    <!ENTITY                                                    LightBiomedicalConcept
"http://larqss.di.ufpb.br/ontologias/LightBiomedicalConcept.owl#" >
]>
<rdf:RDF xmlns="http://larqss.di.ufpb.br/ontologias/cdsConfig.owl#"
     xml:base="http://larqss.di.ufpb.br/ontologias/cdsConfig.owl"
    <owl:Ontology rdf:about="">
        <owl:imports
rdf:resource="http://larqss.di.ufpb.br/ontologias/LightBiomedicalConcept.owl"/>
        <owl:imports rdf:resource="http://larqss.di.ufpb.br/ontologias/UIComponent.owl"/>
    </owl:Ontology>

    <owl:ObjectProperty rdf:about="#ConfigRelationship"/>

    <owl:ObjectProperty rdf:about="#hasData">
        <rdfs:subPropertyOf rdf:resource="#ConfigRelationship"/>
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:about="#hasId">
        <rdfs:subPropertyOf rdf:resource="#ConfigRelationship"/>
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:about="#isA">
        <rdfs:subPropertyOf rdf:resource="#ConfigRelationship"/>
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:about="#isPartOf">
        <rdfs:subPropertyOf rdf:resource="#ConfigRelationship"/>
    </owl:ObjectProperty>

    <owl:DatatypeProperty rdf:about="#ConfigProperty"/>

    <owl:DatatypeProperty rdf:about="#className">
        <rdfs:subPropertyOf rdf:resource="#ConfigProperty"/>
    </owl:DatatypeProperty>

    <owl:DatatypeProperty rdf:about="#codeMapping">
        <rdfs:subPropertyOf rdf:resource="#ConfigProperty"/>
    </owl:DatatypeProperty>

    <owl:DatatypeProperty rdf:about="#description">
        <rdfs:subPropertyOf rdf:resource="#ConfigProperty"/>
    </owl:DatatypeProperty>

    <owl:DatatypeProperty rdf:about="#name">
        <rdfs:subPropertyOf rdf:resource="#ConfigProperty"/>
    </owl:DatatypeProperty>

    <owl:DatatypeProperty rdf:about="#value"/>
```

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

    <owl:Class rdf:about="#Concept">
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasId"/>
                <owl:onClass rdf:resource="#Identifier"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isPartOf"/>
                <owl:onClass rdf:resource="#Layer"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#name"/>
                <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="#hasData"/>
                <owl:onClass rdf:resource="#Data"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isA"/>
                <owl:onClass rdf:resource="#ResourceMapping"/>
                <owl:maxQualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxQualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>

    <owl:Class rdf:about="#Data"/>

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

    <owl:Class rdf:about="#Layer">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#isPartOf"/>
                <owl:onClass rdf:resource="#Application"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#name"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="&xsd;string"/>
            </owl:Restriction>
```

```
        </rdfs:subClassOf>
    </owl:Class>

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    <owl:Thing rdf:about="#application_null">
        <rdf:type rdf:resource="#Application"/>
        <name rdf:datatype="&xsd;string"></name>
    </owl:Thing>

    <Concept rdf:about="#concept_null">
        <rdf:type rdf:resource="&owl;Thing"/>
        <name rdf:datatype="&xsd;string">Conceito vazio</name>
        <hasData rdf:resource="#data_null"/>
        <hasId rdf:resource="#id_null"/>
        <isPartOf rdf:resource="#layer_null"/>
        <isA rdf:resource="#resourceMap"/>
    </Concept>

    <Data rdf:about="#data_applicationUserList">
        <rdf:type rdf:resource="&owl;Thing"/>
        <name rdf:datatype="&xsd;string">userList</name>
    </Data>

    <Data rdf:about="#data_cdsError">
        <rdf:type rdf:resource="&owl;Thing"/>
        <description rdf:datatype="&xsd;string"
            >Reportar erros nos campos</description>
        <name rdf:datatype="&xsd;string">cdsError</name>
    </Data>

    <owl:Thing rdf:about="#data_cdsFulfill">
        <rdf:type rdf:resource="#Data"/>
        <description rdf:datatype="&xsd;string"
            >Preenche automaticamente os campos</description>
        <name rdf:datatype="&xsd;string">cdsFulfill</name>
    </owl:Thing>

    <Data rdf:about="#data_cdsInfoButton">
        <rdf:type rdf:resource="&owl;Thing"/>
        <name rdf:datatype="&xsd;string">cdsInfoButton</name>
    </Data>
```

```
<owl:Thing rdf:about="#data_cdsMessage">
    <rdf:type rdf:resource="#Data"/>
    <name rdf:datatype="&xsd;string">cdsMessage</name>
</owl:Thing>

<Data rdf:about="#data_cdsSuccess">
    <rdf:type rdf:resource="&owl;Thing"/>
    <description rdf:datatype="&xsd;string"
        >Valida os campos</description>
    <name rdf:datatype="&xsd;string">cdsSuccess</name>
</Data>

<owl:Thing rdf:about="#data_cdsSuggestion">
    <rdf:type rdf:resource="#Data"/>
    <name rdf:datatype="&xsd;string">cdsSuggestion</name>
</owl:Thing>

<owl:Thing rdf:about="#data_cronInterval1">
    <rdf:type rdf:resource="#Data"/>
    <value rdf:datatype="&xsd;string">* 1 * * * ?</value>
</owl:Thing>

<owl:Thing rdf:about="#data_eventScheduler">
    <rdf:type rdf:resource="#Data"/>
    <name rdf:datatype="&xsd;string">cron</name>
</owl:Thing>

<Data rdf:about="#data_messagePane">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string">msgPane</name>
</Data>

<owl:Thing rdf:about="#data_null">
    <rdf:type rdf:resource="#Data"/>
</owl:Thing>

<Data rdf:about="#data_patient_CPF">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string">cpf</name>
</Data>

<Data rdf:about="#data_resident_list">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string">residentList</name>
</Data>

<Data rdf:about="#data_resident_selected">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string"
        >residentSelected</name>
</Data>

<owl:Thing rdf:about="#data_scopeApplication">
    <rdf:type rdf:resource="#Data"/>
    <name rdf:datatype="&xsd;string">application</name>
</owl:Thing>

<Data rdf:about="#data_scopeDocument">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string">document</name>
</Data>

<Data rdf:about="#data_scopePatient">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string">patient</name>
</Data>

<owl:Thing rdf:about="#data_scopeUser">
    <rdf:type rdf:resource="#Data"/>
    <name rdf:datatype="&xsd;string">user</name>
</owl:Thing>

<owl:Thing rdf:about="#data_user_name">
    <rdf:type rdf:resource="#Data"/>
    <name rdf:datatype="&xsd;string">userName</name>
</owl:Thing>

<owl:Thing rdf:about="#id_cron">
    <rdf:type rdf:resource="#Identifier"/>
    <className rdf:datatype="&xsd;string"
        >br.ufpb.larqss.multiperson.cds.conceptIdentifier.CronInterval</className>
</owl:Thing>

<owl:Thing rdf:about="#id_multiperson">
```

```
    <rdf:type rdf:resource="#Identifier"/>
    <className rdf:datatype="&xsd;string"
        >br.ufpb.larqss.multiperson.cds.conceptIdentifier.Mutiperson</className>
</owl:Thing>

<Identifier rdf:about="#id_null">
    <rdf:type rdf:resource="&owl;Thing"/>
    <className rdf:datatype="&xsd;string"></className>
</Identifier>

<owl:Thing rdf:about="#id_opencti_biomedical">
    <rdf:type rdf:resource="#Identifier"/>
    <className rdf:datatype="&xsd;string"
        >br.ufpb.larqss.multiperson.cds.conceptIdentifier.OpenctiBiomedical</className>
</owl:Thing>

<owl:Thing rdf:about="#id_opencti_events">
    <rdf:type rdf:resource="#Identifier"/>
    <className rdf:datatype="&xsd;string"
        >br.ufpb.larqss.multiperson.cds.conceptIdentifier.OpenctiEvents</className>
</owl:Thing>

<Identifier rdf:about="#id_opencti_tradicional">
    <rdf:type rdf:resource="&owl;Thing"/>
    <className rdf:datatype="&xsd;string"
        >br.ufpb.larqss.multiperson.cds.conceptIdentifier.OpenctiMBean</className>
</Identifier>

<owl:Thing rdf:about="#layer_null">
    <rdf:type rdf:resource="#Layer"/>
    <name rdf:datatype="&xsd;string"></name>
    <isPartOf rdf:resource="#application_null"/>
</owl:Thing>

<Application rdf:about="#multiperson_application">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string">MultiPerson</name>
</Application>

<Concept rdf:about="#multiperson_communication_concept_cdsError">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string"
        >Conceito do CDSError</name>
    <hasData rdf:resource="#data_cdsError"/>
    <hasId rdf:resource="#id_multiperson"/>
    <isPartOf rdf:resource="#multiperson_layer_communication"/>
</Concept>

<owl:Thing rdf:about="#multiperson_communication_concept_cdsFulfill">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Conceito do cdsFulfill</name>
    <hasData rdf:resource="#data_cdsFulfill"/>
    <hasId rdf:resource="#id_multiperson"/>
    <isPartOf rdf:resource="#multiperson_layer_communication"/>
</owl:Thing>

<owl:Thing rdf:about="#multiperson_communication_concept_cdsInfoButton">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Conceito para mostrar informa&#231;&#245;es adicionais</name>
    <hasData rdf:resource="#data_cdsInfoButton"/>
    <hasId rdf:resource="#id_multiperson"/>
    <isPartOf rdf:resource="#multiperson_layer_communication"/>
</owl:Thing>

<owl:Thing rdf:about="#multiperson_communication_concept_cdsMessage">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Conceito do envio de Menssagem do CDS</name>
    <hasData rdf:resource="#data_cdsMessage"/>
    <hasId rdf:resource="#id_multiperson"/>
    <isPartOf rdf:resource="#multiperson_layer_communication"/>
</owl:Thing>

<owl:Thing rdf:about="#multiperson_communication_concept_cdsSucess">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Conceito do evento cdsSucess</name>
    <hasData rdf:resource="#data_cdsSuccess"/>
    <hasId rdf:resource="#id_multiperson"/>
    <isPartOf rdf:resource="#multiperson_layer_communication"/>
</owl:Thing>
```

```
<owl:Thing rdf:about="#multiperson_communication_concept_cdsSuggestion">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Conceito para a sugest&#227;o de valores</name>
    <hasData rdf:resource="#data_cdsSuggestion"/>
    <hasId rdf:resource="#id_multiperson"/>
    <isPartOf rdf:resource="#multiperson_layer_communication"/>
</owl:Thing>

<owl:Thing rdf:about="#multiperson_domain_concept_eventScheduler">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Agendador de eventos</name>
    <hasData rdf:resource="#data_eventScheduler"/>
    <hasId rdf:resource="#id_multiperson"/>
    <isPartOf rdf:resource="#multiperson_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</owl:Thing>

<owl:Thing rdf:about="#multiperson_domain_concept_scope_application">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Conceito do escopo de aplica&#231;&#227;o</name>
    <hasData rdf:resource="#data_scopeApplication"/>
    <hasId rdf:resource="#id_multiperson"/>
    <isPartOf rdf:resource="#multiperson_layer_domain"/>
</owl:Thing>

<owl:Thing rdf:about="#multiperson_domain_concept_scope_document">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Conceito do escopo de documento</name>
    <hasData rdf:resource="#data_scopeDocument"/>
    <hasId rdf:resource="#id_multiperson"/>
    <isPartOf rdf:resource="#multiperson_layer_domain"/>
</owl:Thing>

<Concept rdf:about="#multiperson_domain_concept_scope_patient">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string"
        >Conceito do escopo do paciente</name>
    <hasData rdf:resource="#data_scopePatient"/>
    <hasId rdf:resource="#id_multiperson"/>
    <isPartOf rdf:resource="#multiperson_layer_domain"/>
</Concept>

<Concept rdf:about="#multiperson_domain_concept_scope_user">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string"
        >Conceito do escopo do usuario</name>
    <hasData rdf:resource="#data_scopeUser"/>
    <hasId rdf:resource="#id_multiperson"/>
    <isPartOf rdf:resource="#multiperson_layer_domain"/>
</Concept>

<Layer rdf:about="#multiperson_layer_communication">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string"
        >Camada de Comunica&#231;&#227;o</name>
    <isPartOf rdf:resource="#multiperson_application"/>
</Layer>

<owl:Thing rdf:about="#multiperson_layer_domain">
    <rdf:type rdf:resource="#Layer"/>
    <name rdf:datatype="&xsd;string"
        >Camada de Dominio do CDS</name>
    <isPartOf rdf:resource="#multiperson_application"/>
</owl:Thing>

<owl:Thing rdf:about="#opencti_application">
    <rdf:type rdf:resource="#Application"/>
    <name rdf:datatype="&xsd;string">OpenCTI</name>
</owl:Thing>

<owl:Thing rdf:about="#opencti_domain_concept_applicationUserList">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Lista dos usuarios logados da aplica&#231;&#227;o</name>
    <hasData rdf:resource="#data_applicationUserList"/>
    <hasId rdf:resource="#id_opencti_tradicional"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</owl:Thing>
```

```
<owl:Thing rdf:about="#opencti_domain_concept_bmi">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Indice de Massa Corporea</name>
    <hasData rdf:resource="&LightBiomedicalConcept;BMI"/>
    <hasId rdf:resource="#id_opencti_biomedical"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</owl:Thing>

<Concept rdf:about="#opencti_domain_concept_glasgowEyeResponse">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string"
        >Resposta ocular da escala de glasgow</name>
    <hasData rdf:resource="&LightBiomedicalConcept;bestEyeResponse"/>
    <hasId rdf:resource="#id_opencti_biomedical"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</Concept>

<Concept rdf:about="#opencti_domain_concept_glasgowMotorResponse">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string"
        >Resposta Motora da escala de Glasgow</name>
    <hasData rdf:resource="&LightBiomedicalConcept;bestMotorResponse"/>
    <hasId rdf:resource="#id_opencti_biomedical"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</Concept>

<owl:Thing rdf:about="#opencti_domain_concept_glasgowScore">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Valor da escala de glasgow</name>
    <hasData rdf:resource="&LightBiomedicalConcept;glasgowComaScore"/>
    <hasId rdf:resource="#id_opencti_biomedical"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</owl:Thing>

<owl:Thing rdf:about="#opencti_domain_concept_glasgowVerbalResponse">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Resposta Verbal da Escala de Glasgow</name>
    <hasData rdf:resource="&LightBiomedicalConcept;bestVerbalResponse"/>
    <hasId rdf:resource="#id_opencti_biomedical"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</owl:Thing>

<owl:Thing rdf:about="#opencti_domain_concept_heartRate">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Frequencia Cardiaca</name>
    <hasData rdf:resource="&LightBiomedicalConcept;heartRate"/>
    <hasId rdf:resource="#id_opencti_biomedical"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</owl:Thing>

<owl:Thing rdf:about="#opencti_domain_concept_height">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string">Altura</name>
    <hasData rdf:resource="&LightBiomedicalConcept;height"/>
    <hasId rdf:resource="#id_opencti_biomedical"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</owl:Thing>

<owl:Thing rdf:about="#opencti_domain_concept_idade">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string">Idade da pessoa</name>
    <hasData rdf:resource="&LightBiomedicalConcept;idade"/>
    <hasId rdf:resource="#id_opencti_biomedical"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</owl:Thing>

<owl:Thing rdf:about="#opencti_domain_concept_messagePane">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Local para mostrar mensagens</name>
    <hasData rdf:resource="#data_messagePane"/>
    <hasId rdf:resource="#id_opencti_tradicional"/>
```

```
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</owl:Thing>

<owl:Thing rdf:about="#opencti_domain_concept_patient_CPF">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string">CPF do paciente</name>
    <hasData rdf:resource="#data_patient_CPF"/>
    <hasId rdf:resource="#id_opencti_tradicional"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</owl:Thing>

<Concept rdf:about="#opencti_domain_concept_reasonForHospitalization">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string"
        >Motivo de interna&#231;&#227;o</name>
    <hasData rdf:resource="&LightBiomedicalConcept;reasonForHospitalization"/>
    <hasId rdf:resource="#id_opencti_biomedical"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</Concept>

<Concept rdf:about="#opencti_domain_concept_resident_list">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string"
        >Lista dos residentes</name>
    <hasData rdf:resource="#data_resident_list"/>
    <hasId rdf:resource="#id_opencti_tradicional"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</Concept>

<Concept rdf:about="#opencti_domain_concept_resident_selected">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string"
        >Nome do residente selecionado</name>
    <hasData rdf:resource="#data_resident_selected"/>
    <hasId rdf:resource="#id_opencti_tradicional"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</Concept>

<Concept rdf:about="#opencti_domain_concept_tipoDeMorteNaoNatural">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string"
        >Tipo da morte n&#227;o natural</name>
    <hasData rdf:resource="&LightBiomedicalConcept;tipoDeMorteNaoNatural"/>
    <hasId rdf:resource="#id_opencti_biomedical"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</Concept>

<Concept rdf:about="#opencti_domain_concept_user_name">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string">Nome do Usu&#225;rio</name>
    <hasData rdf:resource="#data_user_name"/>
    <hasId rdf:resource="#id_opencti_tradicional"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</Concept>

<Concept rdf:about="#opencti_domain_concept_weight">
    <rdf:type rdf:resource="&owl;Thing"/>
    <name rdf:datatype="&xsd;string">Peso</name>
    <hasData rdf:resource="&LightBiomedicalConcept;weight"/>
    <hasId rdf:resource="#id_opencti_biomedical"/>
    <isPartOf rdf:resource="#opencti_layer_domain"/>
    <isA rdf:resource="#resourceMap"/>
</Concept>

<owl:Thing rdf:about="#opencti_gui_concept_getFocus">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Obtem o Foco do componente</name>
    <hasData rdf:resource="&UIComponent;getFocus"/>
    <hasId rdf:resource="#id_opencti_events"/>
    <isPartOf rdf:resource="#opencti_layer_gui"/>
</owl:Thing>

<owl:Thing rdf:about="#opencti_gui_concept_lostFocus">
    <rdf:type rdf:resource="#Concept"/>
    <name rdf:datatype="&xsd;string"
        >Perda do Foco do componente</name>
```

```
        <hasData rdf:resource="&UIComponent;lostFocus"/>
        <hasId rdf:resource="#id_opencti_events"/>
        <isPartOf rdf:resource="#opencti_layer_gui"/>
    </owl:Thing>

    <owl:Thing rdf:about="#opencti_gui_concept_onClick">
        <rdf:type rdf:resource="#Concept"/>
        <name rdf:datatype="&xsd;string"
            >Lan&#231;ado ao clicar com o mouse</name>
        <hasData rdf:resource="&UIComponent;mouseEntered"/>
        <hasId rdf:resource="#id_opencti_events"/>
        <isPartOf rdf:resource="#opencti_layer_gui"/>
    </owl:Thing>

    <Concept rdf:about="#opencti_gui_concept_onKeyPressed">
        <rdf:type rdf:resource="&owl;Thing"/>
        <name rdf:datatype="&xsd;string"
            >Evento ocorrido ao digitar um valor</name>
        <hasData rdf:resource="&UIComponent;keyPressed"/>
        <hasId rdf:resource="#id_opencti_events"/>
        <isPartOf rdf:resource="#opencti_layer_gui"/>
    </Concept>

    <Layer rdf:about="#opencti_layer_domain">
        <rdf:type rdf:resource="&owl;Thing"/>
        <name rdf:datatype="&xsd;string"
            >Camada de Dominio</name>
        <isPartOf rdf:resource="#opencti_application"/>
    </Layer>

    <owl:Thing rdf:about="#opencti_layer_gui">
        <rdf:type rdf:resource="#Layer"/>
        <name rdf:datatype="&xsd;string"
            >Camada de Apresenta&#231;&#227;o</name>
        <isPartOf rdf:resource="#opencti_application"/>
    </owl:Thing>

    <Layer rdf:about="#opencti_layer_scheduling">
        <rdf:type rdf:resource="&owl;Thing"/>
        <name rdf:datatype="&xsd;string"
            >Camada de escalonamento</name>
        <isPartOf rdf:resource="#opencti_application"/>
    </Layer>

    <Concept rdf:about="#opencti_scheduling_concept_cronInterval1">
        <rdf:type rdf:resource="&owl;Thing"/>
        <name rdf:datatype="&xsd;string"
            >Intervalo de repeti&#231;&#227;o do cron</name>
        <hasData rdf:resource="#data_cronInterval1"/>
        <hasId rdf:resource="#id_cron"/>
        <isPartOf rdf:resource="#opencti_layer_scheduling"/>
    </Concept>

    <ResourceMapping rdf:about="#resourceMap">
        <rdf:type rdf:resource="&owl;Thing"/>
    </ResourceMapping>
</rdf:RDF>
```

Vejamos, então, a ontologia de agentes.

```
<?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 cdsConfig2 "http://opencti.ufpb.br/cdsConfig.owl#" >
         <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
         <!ENTITY cdsConfig "http://larqss.di.ufpb.br/ontologias/cdsConfig.owl#" >
         <!ENTITY cdsAgentsBETA "http://larqss.di.ufpb.br/ontologias/cdsAgents.owl#" >
]>
<rdf:RDF xmlns="http://larqss.di.ufpb.br/ontologias/cdsAgents.owl#"
          xml:base="http://larqss.di.ufpb.br/ontologias/cdsAgents.owl"
         <owl:Ontology rdf:about="">
                 <owl:imports rdf:resource="http://larqss.di.ufpb.br/ontologias/cdsConfig.owl"/>
         </owl:Ontology>

         <owl:ObjectProperty rdf:about="#agentRelationship"/>

         <owl:ObjectProperty rdf:about="#hasActivation">
                 <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
```

```
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasComponent">
        <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasContext">
        <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasEvent">
        <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasField">
        <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasLogic">
        <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasMetadata">
        <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasOutput">
        <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasProperty">
        <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasResource">
        <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasResult">
        <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasScope">
        <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasTarget">
        <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasTrigger">
        <rdfs:subPropertyOf rdf:resource="#agentRelationship"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:about="#activate">
        <rdfs:subPropertyOf rdf:resource="#logic"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#afterActivate">
        <rdfs:subPropertyOf rdf:resource="#logic"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#agentProperty"/>

<owl:DatatypeProperty rdf:about="#alias">
        <rdfs:subPropertyOf rdf:resource="#agentProperty"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#beforeActivate">
        <rdfs:subPropertyOf rdf:resource="#logic"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#component">
        <rdfs:subPropertyOf rdf:resource="#agentProperty"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#cron"/>

<owl:DatatypeProperty rdf:about="#description">
        <rdfs:subPropertyOf rdf:resource="#agentProperty"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="#ert"/>
```

```
                        <owl:DatatypeProperty rdf:about="#file">
                                <rdfs:subPropertyOf rdf:resource="#logic"/>
                        </owl:DatatypeProperty>

                        <owl:DatatypeProperty rdf:about="#infraClass">
                                <rdfs:subPropertyOf rdf:resource="#logic"/>
                        </owl:DatatypeProperty>

                        <owl:DatatypeProperty rdf:about="#language">
                                <rdfs:subPropertyOf rdf:resource="#component"/>
                        </owl:DatatypeProperty>

                        <owl:DatatypeProperty rdf:about="#logic">
                                <rdfs:subPropertyOf rdf:resource="#agentProperty"/>
                        </owl:DatatypeProperty>

                        <owl:DatatypeProperty rdf:about="#name">
                                <rdfs:subPropertyOf rdf:resource="#agentProperty"/>
                        </owl:DatatypeProperty>

                        <owl:DatatypeProperty rdf:about="#source"/>

                        <owl:Class rdf:about="#Activation">
                                <rdfs:subClassOf>
                                        <owl:Restriction>
                                                <owl:onProperty rdf:resource="#logic"/>
                                                <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="#hasTarget"/>
                                                <owl:onClass rdf:resource="&cdsConfig2;Concept"/>
                                                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                                        </owl:Restriction>
                                </rdfs:subClassOf>
                                <rdfs:subClassOf>
                                        <owl:Restriction>
                                                <owl:onProperty rdf:resource="#hasEvent"/>
                                                <owl:onClass rdf:resource="&cdsConfig2;Concept"/>
                                                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                                        </owl:Restriction>
                                </rdfs:subClassOf>
                        </owl:Class>

                        <owl:Class rdf:about="#Agent">
                                <rdfs:subClassOf>
                                        <owl:Restriction>
                                                <owl:onProperty rdf:resource="#hasTrigger"/>
                                                <owl:onClass rdf:resource="#Trigger"/>
                                                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                                        </owl:Restriction>
                                </rdfs:subClassOf>
                                <rdfs:subClassOf>
                                        <owl:Restriction>
                                                <owl:onProperty rdf:resource="#hasMetadata"/>
                                                <owl:onClass rdf:resource="#Metadata"/>
                                                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                                        </owl:Restriction>
                                </rdfs:subClassOf>
                                <rdfs:subClassOf>
                                        <owl:Restriction>
                                                <owl:onProperty rdf:resource="#hasScope"/>
                                                <owl:onClass rdf:resource="&cdsConfig2;Concept"/>
                                                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                                        </owl:Restriction>
                                </rdfs:subClassOf>
                                <rdfs:subClassOf>
                                        <owl:Restriction>
                                                <owl:onProperty rdf:resource="#hasOutput"/>
                                                <owl:onClass rdf:resource="#Output"/>
                                                <owl:maxQualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxQualifiedCardinality>
                                        </owl:Restriction>
                                </rdfs:subClassOf>
                                <rdfs:subClassOf>
                                        <owl:Restriction>
```

```
                                        <owl:onProperty rdf:resource="#description"/>
                                        <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="#hasContext"/>
                                        <owl:onClass rdf:resource="#Context"/>
                                        <owl:minQualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minQualifiedCardinality>
                                </owl:Restriction>
                        </rdfs:subClassOf>
                </owl:Class>

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

                <owl:Class rdf:about="#Context">
                        <rdfs:subClassOf>
                                <owl:Restriction>
                                        <owl:onProperty rdf:resource="#hasResource"/>
                                        <owl:onClass rdf:resource="#Resource"/>
                                        <owl:minQualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minQualifiedCardinality>
                                </owl:Restriction>
                        </rdfs:subClassOf>
                </owl:Class>

                <owl:Class rdf:about="#Logic">
                        <rdfs:subClassOf>
                                <owl:Restriction>
                                        <owl:onProperty rdf:resource="#afterActivate"/>
                                        <owl:maxQualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxQualifiedCardinality>
                                        <owl:onDataRange rdf:resource="&xsd;string"/>
                                </owl:Restriction>
                        </rdfs:subClassOf>
                        <rdfs:subClassOf>
                                <owl:Restriction>
                                        <owl:onProperty rdf:resource="#activate"/>
                                        <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="#beforeActivate"/>
                                        <owl:maxQualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxQualifiedCardinality>
                                        <owl:onDataRange rdf:resource="&xsd;string"/>
                                </owl:Restriction>
                        </rdfs:subClassOf>
                        <rdfs:subClassOf>
                                <owl:Restriction>
                                        <owl:onProperty rdf:resource="#file"/>
                                        <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                                        <owl:onDataRange rdf:resource="&xsd;string"/>
                                </owl:Restriction>
                        </rdfs:subClassOf>
                </owl:Class>

                <owl:Class rdf:about="#Metadata">
                        <rdfs:subClassOf>
                                <owl:Restriction>
                                        <owl:onProperty rdf:resource="#hasLogic"/>
                                        <owl:onClass rdf:resource="#Logic"/>
                                        <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                                </owl:Restriction>
                        </rdfs:subClassOf>
                        <rdfs:subClassOf>
                                <owl:Restriction>
```

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

                        <owl:Class rdf:about="#Output">
                                <rdfs:subClassOf>
                                        <owl:Restriction>
                                                <owl:onProperty rdf:resource="#hasResult"/>
                                                <owl:onClass rdf:resource="#Result"/>
                                                <owl:minQualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minQualifiedCardinality>
                                        </owl:Restriction>
                                </rdfs:subClassOf>
                        </owl:Class>

                        <owl:Class rdf:about="#Property"/>

                        <owl:Class rdf:about="#Resource">
                                <rdfs:subClassOf>
                                        <owl:Restriction>
                                                <owl:onProperty rdf:resource="#hasProperty"/>
                                                <owl:onClass rdf:resource="#Property"/>
                                                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                                        </owl:Restriction>
                                </rdfs:subClassOf>
                                <rdfs:subClassOf>
                                        <owl:Restriction>
                                                <owl:onProperty rdf:resource="#hasTarget"/>
                                                <owl:onClass rdf:resource="&cdsConfig2;Concept"/>
                                                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                                        </owl:Restriction>
                                </rdfs:subClassOf>
                                <rdfs:subClassOf>
                                        <owl:Restriction>
                                                <owl:onProperty rdf:resource="#alias"/>
                                                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                                                <owl:onDataRange rdf:resource="&xsd;string"/>
                                        </owl:Restriction>
                                </rdfs:subClassOf>
                        </owl:Class>

                        <owl:Class rdf:about="#Result">
                                <rdfs:subClassOf>
                                        <owl:Restriction>
                                                <owl:onProperty rdf:resource="#name"/>
                                                <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="#hasEvent"/>
                                                <owl:onClass rdf:resource="&cdsConfig2;Concept"/>
                                                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                                        </owl:Restriction>
                                </rdfs:subClassOf>
                                <rdfs:subClassOf>
                                        <owl:Restriction>
                                                <owl:onProperty rdf:resource="#hasTarget"/>
                                                <owl:onClass rdf:resource="&cdsConfig2;Concept"/>
                                                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
                                        </owl:Restriction>
                                </rdfs:subClassOf>
                        </owl:Class>

                        <owl:Class rdf:about="#Trigger">
                                <rdfs:subClassOf>
                                        <owl:Restriction>
                                                <owl:onProperty rdf:resource="#hasActivation"/>
                                                <owl:onClass rdf:resource="#Activation"/>
                                                <owl:minQualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minQualifiedCardinality>
                                        </owl:Restriction>
                                </rdfs:subClassOf>
```

```
                </owl:Class>

                <owl:Class rdf:about="&cdsConfig2;Concept"/>

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

                <owl:Thing rdf:about="#agent_bmi">
                        <rdf:type rdf:resource="#Agent"/>
                        <description rdf:datatype="&xsd;string"
                                >Agente criado para calcular o IMC.</description>
                        <hasContext rdf:resource="#bmi_context"/>
                        <hasMetadata rdf:resource="#bmi_metadata"/>
                        <hasOutput rdf:resource="#bmi_output"/>
                        <hasTrigger rdf:resource="#bmi_trigger"/>
                        <hasScope rdf:resource="&cdsConfig;multiperson_domain_concept_scope_document"/>
                </owl:Thing>

                <Agent rdf:about="#agent_glasgowScale">
                        <rdf:type rdf:resource="&owl;Thing"/>
                        <description rdf:datatype="&xsd;string"
                                >Agente criado para calcular a escala de Glasgow.</description>
                        <hasContext rdf:resource="#glasgowScale_context"/>
                        <hasMetadata rdf:resource="#glasgowScale_metadata"/>
                        <hasOutput rdf:resource="#glasgowScale_output"/>
                        <hasTrigger rdf:resource="#glasgowScale_trigger"/>
                        <hasScope rdf:resource="&cdsConfig;multiperson_domain_concept_scope_document"/>
                </Agent>

                <owl:Thing rdf:about="#agent_heartRate">
                        <rdf:type rdf:resource="#Agent"/>
                        <description rdf:datatype="&xsd;string"
                                >Agente criado para valida&#231;&#227;o da frequencia
card&#237;aca.</description>
                        <hasContext rdf:resource="#heartRate_context"/>
                        <hasMetadata rdf:resource="#heartRate_metadata"/>
                        <hasOutput rdf:resource="#heartRate_output"/>
                        <hasTrigger rdf:resource="#heartRate_trigger"/>
                        <hasScope rdf:resource="&cdsConfig;multiperson_domain_concept_scope_document"/>
                </owl:Thing>

                <owl:Thing rdf:about="#agent_patient_cpf">
                        <rdf:type rdf:resource="#Agent"/>
                        <description rdf:datatype="&xsd;string"
                                >Agente criado para a valida&#231;&#227;o do CPF,</description>
                        <hasContext rdf:resource="#patient_cpf_context"/>
                        <hasMetadata rdf:resource="#patient_cpf_metadata"/>
                        <hasOutput rdf:resource="#patient_cpf_output"/>
                        <hasTrigger rdf:resource="#patient_cpf_trigger"/>
                        <hasScope rdf:resource="&cdsConfig;multiperson_domain_concept_scope_document"/>
                </owl:Thing>

                <owl:Thing rdf:about="#agent_reasonForHospitalization">
                        <rdf:type rdf:resource="#Agent"/>
                        <description rdf:datatype="&xsd;string"
                                >Agente criado automaticamente, rever detalhes.</description>
                        <hasContext rdf:resource="#reasonForHospitalization_context"/>
                        <hasMetadata rdf:resource="#reasonForHospitalization_metadata"/>
                        <hasOutput rdf:resource="#reasonForHospitalization_output"/>
                        <hasTrigger rdf:resource="#reasonForHospitalization_trigger"/>
                        <hasScope rdf:resource="&cdsConfig;multiperson_domain_concept_scope_document"/>
                </owl:Thing>

                <owl:Thing rdf:about="#agent_residentSuggest">
                        <rdf:type rdf:resource="#Agent"/>
                        <description rdf:datatype="&xsd;string"
                                >Agente criado para sugest&#227;o de valores.</description>
                        <hasContext rdf:resource="#residentSuggest_context"/>
                        <hasMetadata rdf:resource="#residentSuggest_metadata"/>
                        <hasOutput rdf:resource="#residentSuggest_output"/>
                        <hasTrigger rdf:resource="#residentSuggest_trigger"/>
                        <hasScope rdf:resource="&cdsConfig;multiperson_domain_concept_scope_user"/>
                </owl:Thing>

                <Agent rdf:about="#agent_scheduling1">
                        <rdf:type rdf:resource="&owl;Thing"/>
                        <description rdf:datatype="&xsd;string"
                                >Agente criado para relembrar atividades di&#225;rias.</description>
                        <hasContext rdf:resource="#scheduling1_context"/>
                        <hasMetadata rdf:resource="#scheduling1_metadata"/>
                        <hasOutput rdf:resource="#scheduling1_output"/>
                        <hasTrigger rdf:resource="#scheduling1_trigger"/>
                        <hasScope
rdf:resource="&cdsConfig;multiperson_domain_concept_scope_application"/>
                </Agent>
```

```
<Agent rdf:about="#agent_tipoDeMorteNaoNatural">
        <rdf:type rdf:resource="&owl;Thing"/>
        <description rdf:datatype="&xsd;string"
                >Agente criado automaticamente, rever detalhes.</description>
        <hasContext rdf:resource="#tipoDeMorteNaoNatural_context"/>
        <hasMetadata rdf:resource="#tipoDeMorteNaoNatural_metadata"/>
        <hasOutput rdf:resource="#tipoDeMorteNaoNatural_output"/>
        <hasTrigger rdf:resource="#tipoDeMorteNaoNatural_trigger"/>
        <hasScope rdf:resource="&cdsConfig;multiperson_domain_concept_scope_document"/>
</Agent>

<owl:Thing rdf:about="#bmi_activation_1">
        <rdf:type rdf:resource="#Activation"/>
        <logic rdf:datatype="&xsd;string">activate</logic>
        <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_height"/>
        <hasEvent rdf:resource="&cdsConfig;opencti_gui_concept_lostFocus"/>
</owl:Thing>

<Activation rdf:about="#bmi_activation_2">
        <rdf:type rdf:resource="&owl;Thing"/>
        <logic rdf:datatype="&xsd;string">activate</logic>
        <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_weight"/>
        <hasEvent rdf:resource="&cdsConfig;opencti_gui_concept_lostFocus"/>
</Activation>

<owl:Thing rdf:about="#bmi_context">
        <rdf:type rdf:resource="#Context"/>
        <hasResource rdf:resource="#bmi_resource_1"/>
        <hasResource rdf:resource="#bmi_resource_2"/>
</owl:Thing>

<owl:Thing rdf:about="#bmi_logic">
        <rdf:type rdf:resource="#Logic"/>
        <beforeActivate rdf:datatype="&xsd;string"></beforeActivate>
        <afterActivate rdf:datatype="&xsd;string"></afterActivate>
        <file rdf:datatype="&xsd;string"
                >bmiCalculator.grvy</file>
        <infraClass rdf:datatype="&xsd;string">collectMed</infraClass>
        <activate rdf:datatype="&xsd;string">execut</activate>
</owl:Thing>

<Metadata rdf:about="#bmi_metadata">
        <rdf:type rdf:resource="&owl;Thing"/>
        <hasLogic rdf:resource="#bmi_logic"/>
        <hasComponent rdf:resource="#component_groovy"/>
</Metadata>

<Output rdf:about="#bmi_output">
        <rdf:type rdf:resource="&owl;Thing"/>
        <hasResult rdf:resource="#bmi_result_1"/>
</Output>

<owl:Thing rdf:about="#bmi_resource_1">
        <rdf:type rdf:resource="#Resource"/>
        <alias rdf:datatype="&xsd;string">altura</alias>
        <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_height"/>
</owl:Thing>

<Property rdf:about="#bmi_resource_1_property_345">
        <rdf:type rdf:resource="&owl;Thing"/>
        <ert rdf:datatype="&xsd;string">345</ert>
</Property>

<Property rdf:about="#bmi_resource_1_property_ert">
        <rdf:type rdf:resource="&owl;Thing"/>
        <ert rdf:datatype="&xsd;string"
                >http://larqss.di.ufpb.br/ontologias/cdsAgents.owl#345</ert>
</Property>

<Property rdf:about="#bmi_resource_1_property_null">
        <rdf:type rdf:resource="&owl;Thing"/>
</Property>

<owl:Thing rdf:about="#bmi_resource_2">
        <rdf:type rdf:resource="#Resource"/>
        <alias rdf:datatype="&xsd;string">peso</alias>
        <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_weight"/>
</owl:Thing>

<owl:Thing rdf:about="#bmi_resource_2_property_null">
        <rdf:type rdf:resource="#Property"/>
</owl:Thing>
```

```
                        <Result rdf:about="#bmi_result_1">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <name rdf:datatype="&xsd;string">fulfillMsg</name>
                                <hasEvent
rdf:resource="&cdsConfig;multiperson_communication_concept_cdsFulfill"/>
                                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_bmi"/>
                        </Result>

                        <Trigger rdf:about="#bmi_trigger">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <hasActivation rdf:resource="#bmi_activation_1"/>
                                <hasActivation rdf:resource="#bmi_activation_2"/>
                        </Trigger>

                        <Component rdf:about="#component_groovy">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <language rdf:datatype="&xsd;string">groovy</language>
                        </Component>

                        <owl:Thing rdf:about="#component_null">
                                <rdf:type rdf:resource="#Component"/>
                                <language rdf:datatype="&xsd;string"></language>
                        </owl:Thing>

                        <Component rdf:about="#component_ruby">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <language rdf:datatype="&xsd;string">ruby</language>
                        </Component>

                        <Activation rdf:about="#glasgowScale_activation_1">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <logic rdf:datatype="&xsd;string">activate</logic>
                                <hasTarget
rdf:resource="&cdsConfig;opencti_domain_concept_glasgowMotorResponse"/>
                                <hasEvent rdf:resource="&cdsConfig;opencti_gui_concept_lostFocus"/>
                        </Activation>

                        <Activation rdf:about="#glasgowScale_activation_2">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <logic rdf:datatype="&xsd;string">activate</logic>
                                <hasTarget
rdf:resource="&cdsConfig;opencti_domain_concept_glasgowVerbalResponse"/>
                                <hasEvent rdf:resource="&cdsConfig;opencti_gui_concept_lostFocus"/>
                        </Activation>

                        <Activation rdf:about="#glasgowScale_activation_3">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <logic rdf:datatype="&xsd;string">activate</logic>
                                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_glasgowEyeResponse"/>
                                <hasEvent rdf:resource="&cdsConfig;opencti_gui_concept_lostFocus"/>
                        </Activation>

                        <Context rdf:about="#glasgowScale_context">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <hasResource rdf:resource="#glasgowScale_resource_1"/>
                                <hasResource rdf:resource="#glasgowScale_resource_2"/>
                                <hasResource rdf:resource="#glasgowScale_resource_3"/>
                        </Context>

                        <Logic rdf:about="#glasgowScale_logic">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <beforeActivate rdf:datatype="&xsd;string"></beforeActivate>
                                <afterActivate rdf:datatype="&xsd;string"></afterActivate>
                                <infraClass rdf:datatype="&xsd;string">collectMed</infraClass>
                                <activate rdf:datatype="&xsd;string">execut</activate>
                                <file rdf:datatype="&xsd;string"
                                        >glasgowScaleCalculator.grvy</file>
                        </Logic>

                        <owl:Thing rdf:about="#glasgowScale_metadata">
                                <rdf:type rdf:resource="#Metadata"/>
                                <hasComponent rdf:resource="#component_groovy"/>
                                <hasLogic rdf:resource="#glasgowScale_logic"/>
                        </owl:Thing>

                        <owl:Thing rdf:about="#glasgowScale_output">
                                <rdf:type rdf:resource="#Output"/>
                                <hasResult rdf:resource="#glasgowScale_result_1"/>
                        </owl:Thing>

                        <owl:Thing rdf:about="#glasgowScale_resource_1">
                                <rdf:type rdf:resource="#Resource"/>
                                <alias rdf:datatype="&xsd;string">verbal</alias>
```

```
                                <hasTarget
rdf:resource="&cdsConfig;opencti_domain_concept_glasgowVerbalResponse"/>
                        </owl:Thing>

                        <Property rdf:about="#glasgowScale_resource_1_property_null">
                                <rdf:type rdf:resource="&owl;Thing"/>
                        </Property>

                        <owl:Thing rdf:about="#glasgowScale_resource_2">
                                <rdf:type rdf:resource="#Resource"/>
                                <alias rdf:datatype="&xsd;string">eye</alias>
                                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_glasgowEyeResponse"/>
                        </owl:Thing>

                        <owl:Thing rdf:about="#glasgowScale_resource_2_property_null">
                                <rdf:type rdf:resource="#Property"/>
                        </owl:Thing>

                        <Resource rdf:about="#glasgowScale_resource_3">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <alias rdf:datatype="&xsd;string">motor</alias>
                                <hasTarget
rdf:resource="&cdsConfig;opencti_domain_concept_glasgowMotorResponse"/>
                        </Resource>

                        <owl:Thing rdf:about="#glasgowScale_resource_3_property_null">
                                <rdf:type rdf:resource="#Property"/>
                        </owl:Thing>

                        <Result rdf:about="#glasgowScale_result_1">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <name rdf:datatype="&xsd;string">fulfillMsg</name>
                                <hasEvent
rdf:resource="&cdsConfig;multiperson_communication_concept_cdsFulfill"/>
                                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_glasgowScore"/>
                        </Result>

                        <owl:Thing rdf:about="#glasgowScale_trigger">
                                <rdf:type rdf:resource="#Trigger"/>
                                <hasActivation rdf:resource="#glasgowScale_activation_1"/>
                                <hasActivation rdf:resource="#glasgowScale_activation_2"/>
                                <hasActivation rdf:resource="#glasgowScale_activation_3"/>
                        </owl:Thing>

                        <owl:Thing rdf:about="#heartRate_activation_1">
                                <rdf:type rdf:resource="#Activation"/>
                                <logic rdf:datatype="&xsd;string">activate</logic>
                                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_heartRate"/>
                                <hasEvent rdf:resource="&cdsConfig;opencti_gui_concept_lostFocus"/>
                        </owl:Thing>

                        <Context rdf:about="#heartRate_context">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <hasResource rdf:resource="#heartRate_resource_1"/>
                                <hasResource rdf:resource="#heartRate_resource_2"/>
                                <hasResource rdf:resource="#heartRate_resource_3"/>
                        </Context>

                        <Logic rdf:about="#heartRate_logic">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <beforeActivate rdf:datatype="&xsd;string"></beforeActivate>
                                <afterActivate rdf:datatype="&xsd;string"></afterActivate>
                                <infraClass rdf:datatype="&xsd;string">collectMed</infraClass>
                                <activate rdf:datatype="&xsd;string">exec</activate>
                                <file rdf:datatype="&xsd;string"
                                        >heartRateValidator.grvy</file>
                        </Logic>

                        <owl:Thing rdf:about="#heartRate_metadata">
                                <rdf:type rdf:resource="#Metadata"/>
                                <hasComponent rdf:resource="#component_groovy"/>
                                <hasLogic rdf:resource="#heartRate_logic"/>
                        </owl:Thing>

                        <owl:Thing rdf:about="#heartRate_output">
                                <rdf:type rdf:resource="#Output"/>
                                <hasResult rdf:resource="#heartRate_result_1"/>
                        </owl:Thing>

                        <owl:Thing rdf:about="#heartRate_resource_1">
                                <rdf:type rdf:resource="#Resource"/>
                                <alias rdf:datatype="&xsd;string">frequencia</alias>
                                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_heartRate"/>
                        </owl:Thing>
```

```
<Property rdf:about="#heartRate_resource_1_property_null">
        <rdf:type rdf:resource="&owl;Thing"/>
</Property>

<Resource rdf:about="#heartRate_resource_2">
        <rdf:type rdf:resource="&owl;Thing"/>
        <alias rdf:datatype="&xsd;string">peso</alias>
        <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_weight"/>
</Resource>

<owl:Thing rdf:about="#heartRate_resource_3">
        <rdf:type rdf:resource="#Resource"/>
        <alias rdf:datatype="&xsd;string">altura</alias>
        <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_height"/>
</owl:Thing>

<Result rdf:about="#heartRate_result_1">
        <rdf:type rdf:resource="&owl;Thing"/>
        <name rdf:datatype="&xsd;string">errorMsg</name>
        <hasEvent rdf:resource="&cdsConfig;multiperson_communication_concept_cdsError"/>
        <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_heartRate"/>
</Result>

<owl:Thing rdf:about="#heartRate_trigger">
        <rdf:type rdf:resource="#Trigger"/>
        <hasActivation rdf:resource="#heartRate_activation_1"/>
</owl:Thing>

<Activation rdf:about="#patient_cpf_activation_1">
        <rdf:type rdf:resource="&owl;Thing"/>
        <logic rdf:datatype="&xsd;string">activate</logic>
        <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_patient_CPF"/>
        <hasEvent rdf:resource="&cdsConfig;opencti_gui_concept_lostFocus"/>
</Activation>

<Context rdf:about="#patient_cpf_context">
        <rdf:type rdf:resource="&owl;Thing"/>
        <hasResource rdf:resource="#patient_cpf_resource_1"/>
</Context>

<Logic rdf:about="#patient_cpf_logic">
        <rdf:type rdf:resource="&owl;Thing"/>
        <beforeActivate rdf:datatype="&xsd;string"></beforeActivate>
        <afterActivate rdf:datatype="&xsd;string"></afterActivate>
        <infraClass rdf:datatype="&xsd;string"
                >br.ufpb.larqss.multiperson.implementation.infra.NaiveInfra</infraClass>
        <file rdf:datatype="&xsd;string"
                >cpfValidator.grvy</file>
        <activate rdf:datatype="&xsd;string">exec</activate>
</Logic>

<Metadata rdf:about="#patient_cpf_metadata">
        <rdf:type rdf:resource="&owl;Thing"/>
        <hasComponent rdf:resource="#component_groovy"/>
        <hasLogic rdf:resource="#patient_cpf_logic"/>
</Metadata>

<owl:Thing rdf:about="#patient_cpf_output">
        <rdf:type rdf:resource="#Output"/>
        <hasResult rdf:resource="#patient_cpf_result_1"/>
        <hasResult rdf:resource="#patient_cpf_result_2"/>
</owl:Thing>

<Resource rdf:about="#patient_cpf_resource_1">
        <rdf:type rdf:resource="&owl;Thing"/>
        <alias rdf:datatype="&xsd;string">cpf</alias>
        <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_patient_CPF"/>
</Resource>

<Property rdf:about="#patient_cpf_resource_1_property_null">
        <rdf:type rdf:resource="&owl;Thing"/>
</Property>

<owl:Thing rdf:about="#patient_cpf_result_1">
        <rdf:type rdf:resource="#Result"/>
        <name rdf:datatype="&xsd;string">errorMsg</name>
        <hasEvent rdf:resource="&cdsConfig;multiperson_communication_concept_cdsError"/>
        <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_patient_CPF"/>
</owl:Thing>

<Result rdf:about="#patient_cpf_result_2">
        <rdf:type rdf:resource="&owl;Thing"/>
        <name rdf:datatype="&xsd;string">successMsg</name>
```

```
                                <hasEvent
rdf:resource="&cdsConfig;multiperson_communication_concept_cdsSucess"/>
                                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_patient_CPF"/>
                        </Result>

                        <Trigger rdf:about="#patient_cpf_trigger">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <hasActivation rdf:resource="#patient_cpf_activation_1"/>
                        </Trigger>

                        <Property rdf:about="#property_null">
                                <rdf:type rdf:resource="&owl;Thing"/>
                        </Property>

                        <owl:Thing rdf:about="#reasonForHospitalization_activation_1">
                                <rdf:type rdf:resource="#Activation"/>
                                <logic rdf:datatype="&xsd;string">activate</logic>
                                <hasTarget
rdf:resource="&cdsConfig;opencti_domain_concept_reasonForHospitalization"/>
                                <hasEvent rdf:resource="&cdsConfig;opencti_gui_concept_onClick"/>
                        </owl:Thing>

                        <owl:Thing rdf:about="#reasonForHospitalization_context">
                                <rdf:type rdf:resource="#Context"/>
                                <hasResource rdf:resource="#reasonForHospitalization_resource_1"/>
                        </owl:Thing>

                        <Logic rdf:about="#reasonForHospitalization_logic">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <afterActivate rdf:datatype="&xsd;string"></afterActivate>
                                <beforeActivate rdf:datatype="&xsd;string"></beforeActivate>
                                <infraClass rdf:datatype="&xsd;string"></infraClass>
                                <activate rdf:datatype="&xsd;string">execut</activate>
                                <file rdf:datatype="&xsd;string"
                                        >reasonForHospitalization.grvy</file>
                        </Logic>

                        <owl:Thing rdf:about="#reasonForHospitalization_metadata">
                                <rdf:type rdf:resource="#Metadata"/>
                                <hasComponent rdf:resource="#component_groovy"/>
                                <hasLogic rdf:resource="#reasonForHospitalization_logic"/>
                        </owl:Thing>

                        <owl:Thing rdf:about="#reasonForHospitalization_output">
                                <rdf:type rdf:resource="#Output"/>
                                <hasResult rdf:resource="#reasonForHospitalization_result_1"/>
                        </owl:Thing>

                        <Resource rdf:about="#reasonForHospitalization_resource_1">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <alias rdf:datatype="&xsd;string">reason</alias>
                                <hasTarget
rdf:resource="&cdsConfig;opencti_domain_concept_reasonForHospitalization"/>
                        </Resource>

                        <Property rdf:about="#reasonForHospitalization_resource_1_property_null">
                                <rdf:type rdf:resource="&owl;Thing"/>
                        </Property>

                        <Result rdf:about="#reasonForHospitalization_result_1">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <name rdf:datatype="&xsd;string">infoButtonMsg</name>
                                <hasEvent
rdf:resource="&cdsConfig;multiperson_communication_concept_cdsInfoButton"/>
                                <hasTarget
rdf:resource="&cdsConfig;opencti_domain_concept_reasonForHospitalization"/>
                        </Result>

                        <Trigger rdf:about="#reasonForHospitalization_trigger">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <hasActivation rdf:resource="#reasonForHospitalization_activation_1"/>
                        </Trigger>

                        <owl:Thing rdf:about="#residentSuggest_activation_1">
                                <rdf:type rdf:resource="#Activation"/>
                                <logic rdf:datatype="&xsd;string">activate</logic>
                                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_resident_selected"/>
                                <hasEvent rdf:resource="&cdsConfig;opencti_gui_concept_onKeyPressed"/>
                        </owl:Thing>

                        <Activation rdf:about="#residentSuggest_activation_2">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <logic rdf:datatype="&xsd;string">afterActivate</logic>
                                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_resident_selected"/>
```

```
                <hasEvent rdf:resource="&cdsConfig;opencti_gui_concept_onClick"/>
        </Activation>

        <owl:Thing rdf:about="#residentSuggest_context">
                <rdf:type rdf:resource="#Context"/>
                <hasResource rdf:resource="#residentSuggest_resource_1"/>
                <hasResource rdf:resource="#residentSuggest_resource_2"/>
                <hasResource rdf:resource="#residentSuggest_resource_3"/>
        </owl:Thing>

        <owl:Thing rdf:about="#residentSuggest_logic">
                <rdf:type rdf:resource="#Logic"/>
                <beforeActivate rdf:datatype="&xsd;string"></beforeActivate>
                <infraClass rdf:datatype="&xsd;string"
                        >br.ufpb.larqss.multiperson.implementation.infra.NaiveInfra</infraClass>
                <activate rdf:datatype="&xsd;string">load</activate>
                <file rdf:datatype="&xsd;string"
                        >residentList.grvy</file>
                <afterActivate rdf:datatype="&xsd;string">save</afterActivate>
        </owl:Thing>

        <Metadata rdf:about="#residentSuggest_metadata">
                <rdf:type rdf:resource="&owl;Thing"/>
                <hasComponent rdf:resource="#component_groovy"/>
                <hasLogic rdf:resource="#residentSuggest_logic"/>
        </Metadata>

        <owl:Thing rdf:about="#residentSuggest_output">
                <rdf:type rdf:resource="#Output"/>
                <hasResult rdf:resource="#residentSuggest_result_1"/>
        </owl:Thing>

        <owl:Thing rdf:about="#residentSuggest_resource_1">
                <rdf:type rdf:resource="#Resource"/>
                <alias rdf:datatype="&xsd;string">residentList</alias>
                <hasProperty rdf:resource="#residentSuggest_resource_1_property_document"/>
                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_resident_list"/>
        </owl:Thing>

        <Property rdf:about="#residentSuggest_resource_1_property_document">
                <rdf:type rdf:resource="&owl;Thing"/>
                <source rdf:datatype="&xsd;string">document</source>
        </Property>

        <owl:Thing rdf:about="#residentSuggest_resource_1_property_null">
                <rdf:type rdf:resource="#Property"/>
        </owl:Thing>

        <owl:Thing rdf:about="#residentSuggest_resource_2">
                <rdf:type rdf:resource="#Resource"/>
                <alias rdf:datatype="&xsd;string"
                        >residentSelected</alias>
                <hasProperty rdf:resource="#residentSuggest_resource_2_property_document"/>
                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_resident_selected"/>
        </owl:Thing>

        <owl:Thing rdf:about="#residentSuggest_resource_2_property_document">
                <rdf:type rdf:resource="#Property"/>
                <source rdf:datatype="&xsd;string">document</source>
        </owl:Thing>

        <owl:Thing rdf:about="#residentSuggest_resource_2_property_null">
                <rdf:type rdf:resource="#Property"/>
        </owl:Thing>

        <owl:Thing rdf:about="#residentSuggest_resource_3">
                <rdf:type rdf:resource="#Resource"/>
                <alias rdf:datatype="&xsd;string">userName</alias>
                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_user_name"/>
        </owl:Thing>

        <Property rdf:about="#residentSuggest_resource_3_property_null">
                <rdf:type rdf:resource="&owl;Thing"/>
        </Property>

        <Result rdf:about="#residentSuggest_result_1">
                <rdf:type rdf:resource="&owl;Thing"/>
                <name rdf:datatype="&xsd;string">suggestionMsg</name>
                <hasEvent
rdf:resource="&cdsConfig;multiperson_communication_concept_cdsSuggestion"/>
                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_resident_list"/>
        </Result>

        <owl:Thing rdf:about="#residentSuggest_trigger">
```

```
                        <rdf:type rdf:resource="#Trigger"/>
                        <hasActivation rdf:resource="#residentSuggest_activation_1"/>
                        <hasActivation rdf:resource="#residentSuggest_activation_2"/>
                </owl:Thing>

                <owl:Thing rdf:about="#scheduling1_activation_1">
                        <rdf:type rdf:resource="#Activation"/>
                        <logic rdf:datatype="&xsd;string">activate</logic>
                        <hasTarget rdf:resource="&cdsConfig;multiperson_domain_concept_eventScheduler"/>
                        <hasEvent rdf:resource="&cdsConfig;opencti_scheduling_concept_cronInterval1"/>
                </owl:Thing>

                <owl:Thing rdf:about="#scheduling1_context">
                        <rdf:type rdf:resource="#Context"/>
                        <hasResource rdf:resource="#scheduling1_resource_1"/>
                </owl:Thing>

                <owl:Thing rdf:about="#scheduling1_logic">
                        <rdf:type rdf:resource="#Logic"/>
                        <beforeActivate rdf:datatype="&xsd;string"></beforeActivate>
                        <afterActivate rdf:datatype="&xsd;string"></afterActivate>
                        <infraClass rdf:datatype="&xsd;string"></infraClass>
                        <activate rdf:datatype="&xsd;string">execut</activate>
                        <file rdf:datatype="&xsd;string"
                                >scheduling1.grvy</file>
                </owl:Thing>

                <Metadata rdf:about="#scheduling1_metadata">
                        <rdf:type rdf:resource="&owl;Thing"/>
                        <hasComponent rdf:resource="#component_groovy"/>
                        <hasLogic rdf:resource="#scheduling1_logic"/>
                </Metadata>

                <Output rdf:about="#scheduling1_output">
                        <rdf:type rdf:resource="&owl;Thing"/>
                        <hasResult rdf:resource="#scheduling1_result_1"/>
                </Output>

                <Resource rdf:about="#scheduling1_resource_1">
                        <rdf:type rdf:resource="&owl;Thing"/>
                        <alias rdf:datatype="&xsd;string">userList</alias>
                        <hasTarget
rdf:resource="&cdsConfig;opencti_domain_concept_applicationUserList"/>
                </Resource>

                <Property rdf:about="#scheduling1_resource_1_property_null">
                        <rdf:type rdf:resource="&owl;Thing"/>
                </Property>

                <owl:Thing rdf:about="#scheduling1_result_1">
                        <rdf:type rdf:resource="#Result"/>
                        <name rdf:datatype="&xsd;string">infoMsg</name>
                        <hasEvent
rdf:resource="&cdsConfig;multiperson_communication_concept_cdsMessage"/>
                        <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_messagePane"/>
                </owl:Thing>

                <owl:Thing rdf:about="#scheduling1_trigger">
                        <rdf:type rdf:resource="#Trigger"/>
                        <hasActivation rdf:resource="#scheduling1_activation_1"/>
                </owl:Thing>

                <Activation rdf:about="#tipoDeMorteNaoNatural_activation_1">
                        <rdf:type rdf:resource="&owl;Thing"/>
                        <logic rdf:datatype="&xsd;string">activate</logic>
                        <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_idade"/>
                        <hasEvent rdf:resource="&cdsConfig;opencti_gui_concept_lostFocus"/>
                </Activation>

                <owl:Thing rdf:about="#tipoDeMorteNaoNatural_context">
                        <rdf:type rdf:resource="#Context"/>
                        <hasResource rdf:resource="#tipoDeMorteNaoNatural_resource_1"/>
                        <hasResource rdf:resource="#tipoDeMorteNaoNatural_resource_2"/>
                </owl:Thing>

                <Logic rdf:about="#tipoDeMorteNaoNatural_logic">
                        <rdf:type rdf:resource="&owl;Thing"/>
                        <beforeActivate rdf:datatype="&xsd;string"></beforeActivate>
                        <afterActivate rdf:datatype="&xsd;string"></afterActivate>
                        <infraClass rdf:datatype="&xsd;string"></infraClass>
                        <activate rdf:datatype="&xsd;string">execut</activate>
                        <file rdf:datatype="&xsd;string"
                                >tipoDeMorteNaoNatural.grvy</file>
                </Logic>
```

```
                        <Metadata rdf:about="#tipoDeMorteNaoNatural_metadata">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <hasComponent rdf:resource="#component_groovy"/>
                                <hasLogic rdf:resource="#tipoDeMorteNaoNatural_logic"/>
                        </Metadata>

                        <owl:Thing rdf:about="#tipoDeMorteNaoNatural_output">
                                <rdf:type rdf:resource="#Output"/>
                                <hasResult rdf:resource="#tipoDeMorteNaoNatural_result_1"/>
                        </owl:Thing>

                        <Resource rdf:about="#tipoDeMorteNaoNatural_resource_1">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <alias rdf:datatype="&xsd;string">idade</alias>
                                <hasTarget rdf:resource="&cdsConfig;opencti_domain_concept_idade"/>
                        </Resource>

                        <owl:Thing rdf:about="#tipoDeMorteNaoNatural_resource_1_property_null">
                                <rdf:type rdf:resource="#Property"/>
                        </owl:Thing>

                        <Resource rdf:about="#tipoDeMorteNaoNatural_resource_2">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <alias rdf:datatype="&xsd;string">tipoDeMorte</alias>
                                <hasTarget
rdf:resource="&cdsConfig;opencti_domain_concept_tipoDeMorteNaoNatural"/>
                        </Resource>

                        <Property rdf:about="#tipoDeMorteNaoNatural_resource_2_property_null">
                                <rdf:type rdf:resource="&owl;Thing"/>
                        </Property>

                        <owl:Thing rdf:about="#tipoDeMorteNaoNatural_result_1">
                                <rdf:type rdf:resource="#Result"/>
                                <name rdf:datatype="&xsd;string"
                                        >cdsSuggestionMsg</name>
                                <hasEvent
rdf:resource="&cdsConfig;multiperson_communication_concept_cdsSuggestion"/>
                                <hasTarget
rdf:resource="&cdsConfig;opencti_domain_concept_tipoDeMorteNaoNatural"/>
                        </owl:Thing>

                        <Trigger rdf:about="#tipoDeMorteNaoNatural_trigger">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <hasActivation rdf:resource="#tipoDeMorteNaoNatural_activation_1"/>
                        </Trigger>

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

                        <owl:Thing rdf:about="&cdsConfig;opencti_domain_concept_heartRate"/>
```

```
                <owl:Thing rdf:about="&cdsConfig;opencti_domain_concept_height"/>

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

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

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

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

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

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

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

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

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

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

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

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

                <owl:Thing rdf:about="&cdsConfig;opencti_scheduling_concept_cronInterval1"/>
        </rdf:RDF>

                <Context rdf:about="#patient_cep_context">
                        <rdf:type rdf:resource="&owl;Thing"/>
                        <hasResource rdf:resource="#patient_cep_resource_country"/>
                </Context>

                <Activation rdf:about="#patient_cep_activation_cep">
                        <rdf:type rdf:resource="&owl;Thing"/>
                        <logic rdf:datatype="&xsd;string">activate</logic>
                        <hasTarget rdf:resource="&cdsConfig2;opencti_layer_domain_concept_patient_CEP"/>
                        <hasEvent rdf:resource="&cdsConfig2;opencti_layer_gui_concept_lostFocus"/>
                </Activation>

                <owl:Thing rdf:about="#patient_cep_logic">
                        <rdf:type rdf:resource="#Logic"/>
                        <file rdf:datatype="&xsd;string">cepSearch</file>
                        <activate rdf:datatype="&xsd;string">exec</activate>
                </owl:Thing>

                <Metadata rdf:about="#patient_cep_metadata">
                        <rdf:type rdf:resource="&owl;Thing"/>
                        <hasComponent rdf:resource="#component_groovy"/>
                        <hasLogic rdf:resource="#patient_cep_logic"/>
                </Metadata>

                <owl:Thing rdf:about="#patient_cep_output">
                        <rdf:type rdf:resource="#Output"/>
                        <hasResult rdf:resource="#patient_cep_result_city"/>
                        <hasResult rdf:resource="#patient_cep_result_neighborhood"/>
                        <hasResult rdf:resource="#patient_cep_result_state"/>
                        <hasResult rdf:resource="#patient_cep_result_street"/>
                </owl:Thing>

                <owl:Thing rdf:about="#patient_cep_resource_country">
                        <rdf:type rdf:resource="#Resource"/>
                        <alias rdf:datatype="&xsd;string">country</alias>
                        <hasTarget
rdf:resource="&cdsConfig2;opencti_layer_domain_concept_patient_country"/>
                </owl:Thing>

                <owl:Thing rdf:about="#patient_cep_result_city">
                        <rdf:type rdf:resource="#Result"/>
                        <name rdf:datatype="&xsd;string">cityMsg</name>
                        <hasEvent
rdf:resource="&cdsConfig2;multiperson_layer_communication_concept_cdsFulfill"/>
                        <hasTarget
rdf:resource="&cdsConfig2;opencti_layer_domain_concept_patient_city"/>
                </owl:Thing>

                <Result rdf:about="#patient_cep_result_neighborhood">
                        <rdf:type rdf:resource="&owl;Thing"/>
                        <name rdf:datatype="&xsd;string">neighborhoodMsg</name>
                        <hasEvent
rdf:resource="&cdsConfig2;multiperson_layer_communication_concept_cdsFulfill"/>
                        <hasTarget
rdf:resource="&cdsConfig2;opencti_layer_domain_concept_patient_neighborhood"/>
```

```
                        </Result>

                        <Result rdf:about="#patient_cep_result_state">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <name rdf:datatype="&xsd;string">stateMsg</name>
                                <hasEvent
rdf:resource="&cdsConfig2;multiperson_layer_communication_concept_cdsFulfill"/>
                                <hasTarget
rdf:resource="&cdsConfig2;opencti_layer_domain_concept_patient_state"/>
                        </Result>

                        <Result rdf:about="#patient_cep_result_street">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <name rdf:datatype="&xsd;string">streetMsg</name>
                                <hasEvent
rdf:resource="&cdsConfig2;multiperson_layer_communication_concept_cdsFulfill"/>
                                <hasTarget
rdf:resource="&cdsConfig2;opencti_layer_domain_concept_patient_street"/>
                        </Result>

                        <Trigger rdf:about="#patient_cep_trigger">
                                <rdf:type rdf:resource="&owl;Thing"/>
                                <hasActivation rdf:resource="#patient_cep_activation_cep"/>
                        </Trigger>

                        <owl:Thing rdf:about="&cdsConfig2;opencti_layer_domain_concept_patient_CEP"/>

                        <owl:Thing rdf:about="#agent_patient_cep">
                                <rdf:type rdf:resource="#Agent"/>
                                <description rdf:datatype="&xsd;string"
                                        >Agente que pesquisa o endere&#231;o a partir do cep</description>
                                <hasContext rdf:resource="#patient_cep_context"/>
                                <hasMetadata rdf:resource="#patient_cep_metadata"/>
                                <hasOutput rdf:resource="#patient_cep_output"/>
                                <hasTrigger rdf:resource="#       _trigger"/>
                                <hasScope
rdf:resource="&cdsConfig2;multiperson_layer_domain_concept_scope_stateless"/>
                                <hasField rdf:resource="&cdsConfig2;opencti_layer_domain_concept_patient_CEP"/>
                        </owl:Thing>
```

Código dos Agentes

Apêndice B

Neste apêndice encontra-se a descrição completa dos códigos da lógica interna dos agentes de CDS do MultiPersOn.

Código em Groovy do agente validador de CPF, exposto sucintamente na Figura 5.8.

```
public exec() {

    int     d1, d2;
    int     digito1, digito2, resto;
    int     digitoCPF;
    String  nDigResult;

    d1 = d2 = 0;
    digito1 = digito2 = resto = 0;

    cpf = cpf.replace(".", "");
    cpf = cpf.replace("-", "");
    cpf = cpf.replace(" ", "");
    cpf = cpf.toLowerCase();

    if( cpf.any{ it < '0' } || cpf.any{ it > '9'}) {
         println "Agent - ERROR"
                  def msg = "CPF inválido!";
                  def ret = returnFactory.getInstance(errorMsg, msg);
                  cdsOutputList << ret;
    }else if(cpf.length() == 11) {

         for (int nCount = 1; nCount < cpf.length() -1; nCount++)
         {
                  digitoCPF = Integer.valueOf (cpf.substring(nCount -1, nCount)).intValue();

                  //multiplique a ultima casa por 2 a seguinte por 3 a seguinte por 4 e assim por
                  diante.
                  d1 = d1 + ( 11 - nCount ) * digitoCPF;

                  //para o segundo digito repita o procedimento incluindo o primeiro digito calcu-
                  lado no passo anterior.
                  d2 = d2 + ( 12 - nCount ) * digitoCPF;
         };

         //Primeiro resto da divisão por 11.
         resto = (d1 % 11);

         //Se o resultado for 0 ou 1 o digito é 0 caso contrário o digito é 11 menos o resultado
         anterior.
         if (resto < 2)
                  digito1 = 0;
         else
                  digito1 = 11 - resto;

         d2 += 2 * digito1;

         //Segundo resto da divisão por 11.
         resto = (d2 % 11);

         //Se o resultado for 0 ou 1 o digito é 0 caso contrário o digito é 11 menos o resultado
         anterior.
         if (resto < 2)
                  digito2 = 0;
         else
                  digito2 = 11 - resto;
```

```
        //Digito verificador do CPF que está sendo validado.
        String nDigVerific = cpf.substring (cpf.length()-2, cpf.length());

        //Concatenando o primeiro resto com o segundo.
        nDigResult = String.valueOf(digito1) + String.valueOf(digito2);

        //comparar o digito verificador do cpf com o primeiro resto + o segundo resto.
        boolean ok = nDigVerific.equals(nDigResult);

        if(ok) {
                println "Agent - SUCCES"
                def ret = returnFactory.getInstance(successMsg);
                cdsOutputList << ret;
        }else {
                println "Agent - ERROR"
                def msg = "CPF inválido!";
                def ret = returnFactory.getInstance(errorMsg, msg);
                cdsOutputList << ret;
        }

        println "Agent - lista de returns: ${cdsOutputList}"
        return cdsOutputList

   }else {
                println "Agent - TAMANHO INVALIDO"
                def msg = "CPF inválido!"
                def ret = returnFactory.getInstance(errorMsg, msg)
                cdsOutputList << ret
                return cdsOutputList
   }
}
```

Código em Groovy do agente validador de frequencia cardíaca, exposto sucintamente na Figura 5.11.

```
public exec() {

    def valor = frequencia.properties.find { it.name=="value" }.value
    def exName = frequencia.properties.find { it.name=="exibitionName" }.value
    def name = frequencia.properties.find { it.name=="name" }.value

    println "Agent - ExibitionName: $exName - Valor: $valor";

    if(valor < 10 || valor > 220){
        println "Agent - ERROR"
        def msg = "Frequencia Cardiaca ${frequencia} fora dos valores limitrofes!!";
        def ret = returnFactory.getInstance(errorMsg, msg);
        cdsOutputList << ret;
        return cdsOutputList
    }
}
```

Código em Groovy do agente de cálculo automático do índice de massa corpórea (IMC), exposto sucintamente na Figura 5.14.

```
public execut() {

    def pesoValor = peso.properties.find { it.name=="value" }.value
    def alturaValor = altura.properties.find { it.name=="value" }.value

    println "Agent - PESO valor: "+pesoValor
    println "Agent - ALTURA valor: "+alturaValor

    if (pesoValor > 0 && alturaValor > 0) {
        msg = pesoValor / (alturaValor ** 2)
        println "Agent - FULFILL novoBMI: "+msg
        ret = returnFactory.getInstance(fulfillMsg, msg)
        cdsOutputList << ret
        return cdsOutputList
     }

}
```

Código em Groovy do agente de sugestão de valores, exposto sucintamente na Figura 5.17.

```
public load() {

    def anterior = cdsInfraStructure.select(userName, "residentList");
    anterior.each{ println "Agent INFRA - Valor: "+it.fieldAttribute+" Quantidade: "+it.counter}

    def residentList = [];
    anterior.each{ residentList.add(it.fieldAttribute)};

    def ret = returnFactory.getInstance(suggestionMsg, residentList);
    cdsOutputList << ret;
    return cdsOutputList;
}

public save() {

    cdsInfraStructure.update(userName,"residentList", residentSelected);
}
```

Código em Groovy do agente de notificação de atividades, exposto sucintamente na Figura 5.20.

```
public execut() {

   def msg = "Confira se todas as evoluções diárias foram realizadas!";
   def map = [:];
   userList.each{ map.put(it.username,msg); it.addMessage(msg); };

   def ret = returnFactory.getInstance(infoMsg, map);
   cdsOutputList << ret;
   return cdsOutputList
}
```

Código em Groovy do agente de informações adicionais, exposto sucintamente na Figura 5.22.

```
public exec() {

    if(country == "brasil" || country == "BRASIL") {
        if(cep.length() == 8) {
                println "Agent - País é BRASIL"
                def url = "http://viavirtual.com.br/webservicecep.php?cep=" + cep;
                println "Agent - cdsInfra"
                def stringInteira = cdsInfraStructure.getHttpRequest(url)
                def listaDeStrings = separa(stringInteira,'||')

                if(listaDeStrings.size() == 5) {
                        println "Agent - tamanho é 5!"
                        def street = returnFactory.getInstance(streetMsg, [listaDeStrings[0]])
                        def neighborhood = returnFactory.getInstance(neighborhoodMsg, [listaDe-
                        Strings[1]])
                       def city = returnFactory.getInstance(cityMsg, [listaDeStrings[2]])
                        def state = returnFactory.getInstance(stateMsg, [listaDeStrings[4]],
                        ['pollName':'zipCodePoll'])

                        cdsOutputList << street << neighborhood << city << state
                        return cdsOutputList
            }else {
        println "Agent - Não contem todas as informações de CEP"
        }
      }else {
        println "Agent - CEP inválido"
      }
    }else {
        println "Agent - País diferente do Brasil"
    }

    def street = returnFactory.getInstance(streetMsg, "")
    def neighborhood = returnFactory.getInstance(neighborhoodMsg, "")
    def city = returnFactory.getInstance(cityMsg, "")
    def state = returnFactory.getInstance(stateMsg, "", ['pollName':'zipCodePoll'])

    cdsOutputList << street << neighborhood << city << state
}

def separa(String input, token) {
        println "Agent - "+input
    return input.tokenize(token)
}
```

Referências

Ammenwerth E, Schnell-Inderst P, Machan C, et al. The effect of electronic prescribing on medication errors and adverse drug events: a systematic review. J Am Med Inform Assoc 2008 Sep;15(5):585-600.

ANS. [http://www.ans.gov.br]

Apkon M, Mattera JA, Lin Z, et al. A randomized outpatient trial of a decision-support information technology tool. Arch Intern Med 2005 Nov;165(20):2388-94.

Ash J, Sittig D, Campbell E, Guappone K, Dykstra R. Some unintended consequences of clinical decision support systems. In: Proc. AMIA. American Medical Informatics Association; 2007. p. 26-30.

Ash JS, Berg M, Coiera E. Some unintended consequences of information technology in health care: the nature of patient care information system-related errors. J Am Med Inform Assoc 2004;11(2):104–12.

Bates DW, Kuperman GJ, Rittenberg E, Teich JM, Fiskio J, Ma'luf N, et al. A randomized trial of a computer-based intervention to reduce utilization of redundant laboratory tests. Am J Med 1999;106(2):144–50.

Bemmel J; Van Bemmel V; Musen, M.A. .Handbook of Medical Informatics. 1st edition .Springer Verlag; 1997

Berlin A, Sorani M, Sim I. A taxonomic description of computer-based clinical decision support systems. Journal of Biomedical Informatics. 2006 ;39656-667.

Berner ES, La Lande TJ. Overview of Clinical Decision Support Systems. In: Berner ES. Clinical Decision Support Systems: Theory and Practice. Springer Verlag; 1999.

Berner ES. Clinical Decision Support Systems : State of the Art. Health (San Francisco). 2009 ;(09):

Blobel BG, Engel K, Pharow P. Semantic interoperability--HL7 Version 3 compared to advanced architecture standards. Methods of information in medicine. 2006 ;45(4):343-53.

Bouamrane M, Rector A, Hurrell M. Development of an ontology for a preoperative risk assessment clinical decision support system. 2009 22nd IEEE International Symposium on Computer-Based Medical Systems. 2009 ;1-6.

Carayon P, Schoofs HA, Karsh BT, et al. Work system design for patient safety: the SEIPS model. Qual Saf Health Care 2006 Dec;15 Suppl 1:i50-i58.

Chaudhry B, Wang J, Wu S, Maglione M, Mojica W, Roth E, et al. Systematic review: impact of health information technology on quality, efficiency, and costs of medical care. Ann Intern Med 2006;144(10):742–52.

Cho I, Kim J, Kim JH, Kim HY, Kim Y. Design and implementation of a standards-based interoperable clinical decision support architecture in the context of the Korean EHR. International journal of medical informatics. 2010 ;1-12.

Coen MH. SodaBot: A software agent environment and construction system. In Yannis Labrou and Tim Finin, editors, Proceedings of the CIKM Workshop on Intelligent Information Agents, Third International Conference on Information and Knowledge Management (CIKM 94), Gaithersburg, Maryland, December 1994.

Comité Européende Normalisation, TC 251 "Health informatics". http://www.centc251.org

Conselho Federal De Medicina. (2007) Resolução CFM Número 1.821/2007. Brasília, DF: CFM, 11 Jul. 2007. Disponível em: <http://www.portalmedico.org.br/resolucoes/cfm/2007/1821_2007.htm>. Acessado em: 26 abr. 2011.

Cote, R. A., Rothwell, R. S., Palotay, J. L., et al (1993) The Systematized Nomenclature of Human and Veterinary Medicine – SNOMED International. Northfield, IL: College of American Pathologists.

D. Pisanelly, Ontologies in Medicine, IOS Press, 2004.

DICOM: "Digital Imaging and Communication in Medicine", 2003. http://www.rsna.org.

Dini EF, Linkins RW, Sigafoos J. The impact of computer-generated messages on childhood immunization coverage. Am J Prev Med 2000;18(2):132–9.

Dinu V, Nadkarni P. Guidelines for the effective use of entity–attribute–value mod- eling for biomedical databases. International Journal of Medical Informatics. 2007; (76): 769–779.

Donabedian A. Evaluating the quality of medical care. 1966. Milbank Q 2005;83(4):691-729.

DUARTE, R. C. M. MedViewGen: Uma Ferramenta Baseada em Ontologias para Geração Automática de Interface com o Usuário para Documentos em Registros Eletrônicos em Saúde, 2011. 171p. Dissertação (Mestrado em Informática). Universidade Federal da Paraíba, UFPB, João Pessoa, 2011.

Field TS, Rochon P, Lee M, et al. Costs associated with developing and implementing a computerized clinical decision support system for medication dosing for patients with renal insufficiency in the long-term care setting. J Am Med Inform Assoc 2008 Jul;15(4):466-72.

Foster D, McGregor C, El-Masri S. A survey of agent-based intelligent decision support systems to support clinical management and research. In: Proceedings of the first international workshop on multi-agent systems for medicine, computational biology, and bioinformatics. Citeseer; 2005. p. 16–35.

Franklin S, Graesser A. Is it an Agent , or just a Program?: A Taxonomy for Autonomous Agents. Architecture. 1996 ;

Gilbert D, Aparicio M, Atkinson B, et al. IBM Intelligent Agent Strategy, White Paper, 1995.

Greenes RA. Definition, Scope And Challenges. In: Greenes RA. Clinical Decision Support: The Road Ahead. Boston: Academic Press; 2006.

Greenes RA. Features of Computer-based Clinical Decision Support. In: Greenes RA. Clinical Decision Support: The Road Ahead. Boston: Academic Press; 2006.

Gruber, T. (2010) What is a ontology? http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

Gruninger M, Lee J. Ontology applications and design. Commun ACM 2002;45:39–41.

Hafner M, Breu R. Security engineering for service-oriented architectures. Springer-Verlag New York Inc; 2008.

Health Level Seven, Inc. http://www.hl7.org

Huff SM.Ontologies, Vocabularies and Data Models. In: Greenes RA. Clinical Decision Support: The Road Ahead. Boston: Academic Press; 2006.

Institute of Medicine. Crossing the Quality Chasm: A New Health System for the 21st Century. Washington, DC: National Academy Press, 2001.

Jenders RA. Decision Rules and Expressions. In: Greenes RA. Clinical Decision Support: The Road Ahead. Boston: Academic Press; 2006.

Jha AK, DesRoches CM, Campbell EG et al. Use of electronic health records in U.S. hospitals. N Engl J Med. 2009; 360:1628- 38.

Johnson, S. (1996). Generic Data Modeling for Clinical Repositories. Journal of the American Medical Informatics Association, 3(5), 328. BMJ Publishing Group Ltd.

Kucher N, Koo S, Quiroz R, et al. Electronic alerts to prevent venous thromboembolism among hospitalized patients. N Engl J Med 2005 Mar;352(10):969-77.

Kuperman GJ, Teich JM, Tanasijevic MJ, Ma'Luf N, Rittenberg E, Jha A, et al. Improving response to critical laboratory results with automation: results of a randomized controlled trial. J Am Med Inform Assoc 1999;6(6):512–22.

Lanzola, G., Gatti, L., Falasconi, S., and Stefanelli, M., A framework for building cooperative software agents in medical applications. Artificial Intelligence in Medicine, 1999. 16(3): p. 224-249.

Larssan, J.E., and Hayes-Roth, B., Guardian: intelligent autonomous agent for medical monitoring and diagnosis. IEEE Intelligent Systems, 1998. 13(1): p. 58-64.

Lewis A. Health informatics : information and communication A common language :. Health (San Francisco). 2002 ;

Linder JA, Ma J, Bates DW, et al. Electronic health record use and the quality of ambulatory care in the United States. Arch Intern Med 2007 Jul;167(13):1400-5.

Marakas GM. Decision support systems in the 21st century. Upper Saddle River: Prentice Hall; 1999.

Massad E, Marin HF, Azevedo Neto RS, editores. O prontuário eletrônico na assistência, informação e conhecimento médico. São Paulo: H. de F. Marin; 2003.

Massad, E., Marin, H.F., Azevedo Neto, R. S. "O prontuário eletrônico do paciente na assistência, informação e conhecimento médico". Março 2003.

Medical Objects. [http://www.medical-objects.com.au].

Meigs JB, Cagliero E, Dubey A, et al. A controlled trial of web-based diabetes disease management: the MGH diabetes primary care improvement project. Diabetes Care 2003 Mar;26(3):750-7.

Metzger J, MacDonald K. Clinical Decision Support for the Independent Physician Practice. California HealthCare Foundation. 2002;

Meunier, J.A. A virtual machine for a functional mobile agent architecture supporting distributed medical information. in 12th IEEE Symposium on Computer-Based Medical Systems. 1999.

MICROMEDEX Healthcare Series. Mai. 2005. Disponível em: < http://naples.tju.edu/Ask/Help/handouts/micromedex.pdf >. Acessado em: 26 abr. 2011.

Miller RA, Masarie FE Jr.The demise of the "Greek Oracle" model for medical diagnostic systems. Methods Inf Med 1990;29:1–2.

Moreno A, Garbay C. Software agents in health care. Artificial Intelligence in Medicine. 2003 ;27(3):229-232.

Motta, G. H. M. B. (2004) Um modelo de autorização contextual para o controle de acesso ao prontuário eletrônico do paciente em ambientes abertos e distribuídos. São Paulo: USP. 213 p. Tese (Doutorado) – Programa de Pós-graduação em Engenharia Elétrica, Escola Politécnica da Universidade de São Paulo, São Paulo.

Murphy GF, Hanken MA, Waters KA Eletronic Health Records: Changing the Vision. 1999.

Oliveira, J. A shotgun wedding: business decision support meets clinical deci- sion support. J Healthc Inf Manage 2002;16:28–33.

OMG, 2007. Unified Modeling Language: Superstructure Version 2.1.1. <http:// www.omg.org/cgi-bin/doc?formal/07-02-03> (accessed 02.03.07).

OpenEHR Foundation. http://www.openehr.org

Osheroff JA, Teich JM, Middleton B, Steen EB, Wright A, Detmer DE. A roadmap for national action on clinical decision support. [Internet]. Journal of the American Medical Informatics Association : JAMIA. 2006 ;14(2):141-5.

Osheroff JA, Teich JM, Middleton BF, et al. A roadmap for national action on clinical decision support. American Medical Informatics Association; 2006 June 13. Available at: http://www.amia.org/inside/initiatives/cds/. Accessed March 20, 2009.

Paterno MD, Maviglia SM, Gorman PN, et al. Tiering drug-drug interaction alerts by severity increases compliance rates. J Am Med Inform Assoc 2009 Jan;16(1):40-6.

Pedersen CR, Gumpper KF. ASHP national survey on informatics: assessment of the adoption and use of pharmacy informatics in U.S. hospitals—2007. Am J Health-Syst Pharm. 2008; 65:2244- 64.

Peleg M, Keren S, Denekamp Y. Mapping computerized clinical guidelines to electronic medical records: knowledge-data ontological mapper (KDOM). Journal of biomedical informatics. 2008 ;41(1):180-201.

Perreault, L. E., and Metzger, J. B. "A Pragmatic Framework for Understanding Clinical Decision Support." Journal of the Healthcare Information and Management Systems Society, Summer 1999, 13(2).

PRC. [http://www.datasus.gov.br/prc].

Rosenbloom ST, Geissbuhler AJ, Dupont WD, et al. Effect of CPOE user interface design on user- initiated access to educational and patient information during clinical care. J Am Med Inform Assoc 2005 Jul;12(4):458-73.

SERAFIM, P. CollectMed: Extração e Reuso de Conhecimento Coletivo para o Registro Eletrônico em Saúde, 2011. 125p. Dissertação (Mestrado em Informática). Universidade Federal da Paraíba, UFPB, João Pessoa, 2011.

Shortliffe EH, Axline SG, Buchanan BG, Merigan TC, Cohen SN. An artificial intelligence program to advise physicians regarding antimicrobial therapy. Comput Biomed Res 1973;6(6):544–560.

Sim I, Gorman P, Greenes R, Haynes R, Kaplan B, Lehmann H, et al. Clinical decision support systems for the practice of evidence- based medicine. J Am Med Inform Assoc 2001;8(6):527–34.

Sirajuddin AM, Osheroff JA, Sittig DF, Chuo J, Velasco F, Collins DA. Implementation pearls from a new guidebook on improving medication use and outcomes with clinical decision support. Effective CDS is essential for addressing healthcare performance improvement imperatives. Journal of healthcare information management : JHIM. 2009 ;23(4):38-45.

Sittig DF, Wright A, Osheroff Ja, Middleton B, Teich JM, Ash JS, et al. Grand challenges in clinical decision support. Journal of biomedical informatics. 2008 ;41(2):387-92.

Slee, V.N.; Slee, D.A.; Schmidt, H.J. The Endangered Medical Record : Ensuring Its Integrity in the Age of Informatics , Tringa Press; 2000

Sociedade Brasileira De Informática Em Saúde. (2007) Manual de Certificação para Sistemas de Registro Eletrônico em Saúde, SBIS, Nov. 2007. Disponível em: <http://portal2.tcu.gov.br/portal/pls/portal/docs/769631.PDF>. Acessado em: 26 abr. 2011.

Sordo M, Ogunyemi O, Boxwala A, Greenes R. GELLO: An Object-Oriented Query and Expression Language for Clinical Decision Support: AMIA 2003 Open Source Expo. In: AMIA Annual Symposium Proceedings. American Medical Informatics Association; 2003. p. 1012.

Swain, M., and Kim, Y., Finding Family Relationship in Hospital Database Using Intelligent Agent. IEEE Proceedings, 2004: p. 194-199.

Tamblyn R, Huang A, Taylor L, et al. A randomized trial of the effectiveness of on-demand versus computer-triggered drug decision support in primary care. J Am Med Inform Assoc 2008 Jul;15(4):430-8.

Tierney WM, Overhage JM, Murray MD, et al. Effects of computerized guidelines for managing heart disease in primary care. J Gen Intern Med 2003 Dec;18(12):967-76.

Trowbridge R, Weingarten S. Clinical decision support systems. Making health care safer: a critical analysis of patient safety practices. Rockville, MD: Agency for Healthcare Research and Quality; 2001. Evidence Report/Technology Assessment, No. 43 AHRQ Publication No. 01- E058. p. 589-94.