UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

ONTOLOGIA APLICADA AO DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO

LUIZ CLÁUDIO HOGREFE

BLUMENAU
2013

2013/1-21

LUIZ CLÁUDIO HOGREFE

ONTOLOGIA APLICADA AO DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Roberto Heinzle, Doutor - Orientador

BLUMENAU
2013

2013/1-21

ONTOLOGIA APLICADA AO DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO

Por

LUIZ CLÁUDIO HOGREFE

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Roberto Heinzle, Doutor – Orientador, FURB
Membro:	Prof. Paulo Fernando da Silva, Mestre – FURB
Membro:	Prof. Maurício Capobianco Lopes, Doutor – FURB

Blumenau, 09 de julho de 2013

Dedico este trabalho aos meus pais, já que sempre estiveram ao meu lado, apoiando-me na a concepção do meu *Projeto de Vida*.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

Aos meus pais, peças fundamentais da minha vida.

Aos meus amigos, pelo apoio e incentivo na realização deste trabalho.

Ao meu orientador, Roberto Heinzle, pela confiança na conclusão deste trabalho.

Para fazer uma obra de arte não basta ter talento, não basta ter força, é preciso também viver um grande amor.

Wolfgang Amadeus Mozart

RESUMO

Este trabalho apresenta o desenvolvimento de uma plataforma para a geração de sistemas baseados em ontologias. Esta é composta por três ferramentas. Sendo a primeira utilizada na concepção das ontologias, fundamentada no guia *Ontology Development 101* e sob componentes gráficos. De tal modo, constitui-se um arquivo no formato OWL. A segunda ferramenta responsabiliza-se em converter os artefatos do modelo ontológico para o modelo orientado a objetos. Como resultado desta conversão, gera-se um arquivo no formato XMI. E por fim, a terceira ferramenta é baseada no modelo gerado. Esta se encarrega de conceber a estrutura de cadastros e consultas típicos em sistemas de informação. Sendo que, tanto as ferramentas propostas, como os cadastros originados utilizam conceitos de computação em nuvem. Quanto aos resultados alcançados, é possível afirmar a eficiência do mecanismo de desenvolvimento efetivo de sistemas automatizados por conhecimentos modelados juntos às ontologias.

Palavras-chave: Ontologia. Sistemas de informação. Computação em nuvem.

ABSTRACT

This paper presents the development of a platform based on ontologies for system generation. It is composed by three tools. The first tool is used in the design of ontologies based on Ontology Development 101 guide and under graphical components in three dimensions. So it constitutes a file in OWL format. The second tool is responsible for converting the artifacts from the ontological model to object-oriented model. As a result of this conversion, a file in XMI format is generated. And finally, the third tool is based on the generated model. It is responsible for the designing of typical structures of registers and consultations in information systems. So both, the proposed tools and the originated entries using concepts of cloud computing. As for the results, it is possible to state this mechanism can be efficient in developing effective systems automated by ontologies´ knowledge.

Key-words: Ontology. Information systems. Cloud computing.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

Amazon EC2 – *Amazon Elastic Compute Cloud*

BD – Banco de Dados

IA – Inteligência Artificial

IaaS – *Infrastructure as a Service*

IDE – *Integrated Development Environment*

JEE – Java *Enterprise Edition*

JSON – *JavaScript Object Notation*

MR – Modelo Relacional

MVC – *Model-View-Controller*

NoSQL – *Not Only Structured Query Language*

OMG – *Object Management Group*

OpenGL ES – *Open Graphics Library for Embedded Systems*

OWL – *Web Ontology Language*

PaaS – *Platform as a Service*

RC – Representação do Conhecimento

REST – *Representational State Transfer*

RF – Requisito Funcional

RNF – Requisito Não-Funcional

SaaS – *Software as a Service*

SE – *Standard Edition*

SGBD – Sistema de Gestão de Bases de Dados

SI – Sistema de Informação

SQL – *Structured Query Language*

UML – *Unified Modeling Language*

VTL – *Velocity Template Language*

W3C – *World Wide Web Consortium*

XMI – *XML Metadata Interchange*

XML – *eXtensible Markup Language*

LISTA DE SÍMBOLOS

& - e comercial

% - por cento

- sustenido

SUMÁRIO

1 INTRODUÇÃO

Ontologia é uma forma de conceitualização do conhecimento originada na filosofia, ainda nos pensamentos de Aristóteles, no qual se discutem questões metafísicas, a natureza do que existe e como a realidade é estruturada. Porém, a pesquisa sobre ontologias não é restrita à filosofia e à lógica e tem se caracterizado pela interdisciplinaridade, envolvendo áreas tão diversas como a Ciência da Computação, Ciência da Informação, entre outras (ALMEIDA et al., 2010, p. 33).

O termo ontologia passou a ser utilizado em Ciência da Computação, apenas na década de 1960, inicialmente nas pesquisas sobre Representação do Conhecimento (RC), subárea da Inteligência Artificial (IA). Neste contexto, o termo diz respeito a um artefato de software, uma linguagem formal que tem utilizações específicas em arquiteturas de sistemas inteligentes (VICKERY, 1997, p. 229).

Igualmente recente é a modelagem conceitual, que se refere a uma fase do desenvolvimento de sistemas, que tem por finalidade descrever, da melhor forma possível, parte da realidade e representar processos de interesse em um contexto social (SMITH; WELTY, 2001; GUARINO, 1998). Além disso, segundo Campus (2004, p. 24), “uma representação de conhecimento é um mecanismo usado para se raciocinar sobre o mundo, em vez de agir diretamente sobre ele. Neste sentido, ela é, fundamentalmente, um substituto para aquilo que representa”.

Sendo assim, considera-se o uso de ontologias junto à fase de modelagem conceitual, uma possibilidade de concepção de sistemas automatizados. Visto que, ao contrário do que possa parecer, a geração destes sistemas não privilegia exclusivamente questões da técnica computacional. Trata-se de uma atividade constituída por etapas distintas: em algumas prevalece o estudo de processos algorítmicos que vão ordenar as ações dos computadores, em outras a comunicação e a capacidade de abstração humana são essenciais para obter bons resultados, do ponto de vista sistêmico (ALMEIDA; BARBOSA, 2009, p. 2033).

Já a computação em nuvem é um modelo de computação distribuída, que difere dos tradicionais, em que é altamente escalável em termos de infraestrutura, fornece *frameworks* para desenvolvimento de aplicativos em nuvem, controle de transações e também hospedagem para os aplicativos desenvolvidos. Além disso, por ser impulsionada por economias de escala, a computação em nuvem permite que os serviços sejam configurados

dinamicamente e entregues sob demanda, possibilitando o consumidor pagar ao fornecedor com base no consumo (FOSTER et al., 2008).

Uma vez que o desenvolvimento de Sistemas de Informação (SI) seja uma atividade constituída por etapas distintas: os profissionais precisam criar as tabelas do Banco de Dados (BD), algoritmos em uma linguagem de programação, telas de apresentação e controlar a ligação entre as partes. Todos os passos tornam-se repetitivos e demorados, pois, além de serem executados de forma manual, simultaneamente também terão que se preocupar com a lógica de negócio, deixando o ambiente suscetível a erros.

Desta forma, tendo em vista o cenário descrito anteriormente, busca-se com este trabalho, reduzir os custos pertinentes às etapas da geração de sistemas. Sendo esta premissa alcançada pela junção da automatização estrutural destes sistemas baseados em ontologias, à automatização infraestrutural dos processos, sob um ambiente de computação em nuvem. Chegando a um estado de plena autonomia nas etapas do "roteiro clássico" de desenvolvimento de SI. Permitindo, além da redução dos gastos elevados com infraestrutura e manutenção, a melhoria na automação dos serviços e agilidade na resposta aos clientes.

Diante deste contexto, como ferramenta facilitadora do processo de desenvolvimento de sistemas, o presente trabalho constitui-se de um protótipo de plataforma para a construção automatizada das rotinas de cadastros e consultas típicos de SI. Esta plataforma será composta por três ferramentas: a primeira para a definição, conceitualização e formalização de ontologias, a segunda para a conversão de modelos ontológicos em orientados a objetos e a terceira, para a geração de artefatos de software inerentes às camadas do modelo *Model-View-Controller* (MVC). Sendo que a terceira ferramenta constituirá os sistemas informatizados.

Portanto, com a plataforma mencionada, pretende-se atender às várias etapas do desenvolvimento de SI sob uma perspectiva de alto nível, a partir da análise conceitual de cada base do conhecimento. Assim sendo, após a finalização do processo de construção automatizada dos sistemas a partir da definição das ontologias, os usuários já terão os mesmos prontos para uso, instalados em um ambiente no paradigma de computação em nuvem, permitindo acessá-los pela Internet.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é a geração de uma plataforma para desenvolvimento de SI, a partir de cada representação do conhecimento definida por ontologias, utilizando conceitos de computação em nuvem.

Os objetivos específicos do trabalho são:

a) fornecer os processos para especificação, conceitualização e formalização de ontologias para SI;
b) fornecer os mecanismo para converter a definição da linguagem de ontologia para a linguagem orientada a objetos;
c) permitir a geração de rotinas de cadastros e consultas típicos de SI, a partir da definição de classes, identificadas na linguagem orientada a objetos.

1.2 ESTRUTURA DO TRABALHO

O trabalho está desenvolvido em quatro capítulos. O segundo capítulo do presente trabalho refere-se à fundamentação teórica necessária para o seu entendimento.

O terceiro capítulo contém a descrição de como ocorreu a especificação e a implementação das três ferramentas que compõem o protótipo da plataforma almejada. Além de apresentar os resultados e discursões identificados durante e após o desenvolvimento.

Ao final, o quarto capítulo apresenta as conclusões do presente trabalho, apresentando também sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Primeiramente será detalhado o método em que o presente trabalho se baseia (ontologia) para representação conceitual do conhecimento. Em seguida se descreve a técnica para aplicação deste método: *Ontology Development 101*. Após, explana-se sobre modelagem de sistemas. Posteriormente serão explicitadas as principais tecnologias adotadas. Por fim, são listados dois trabalhos correlatos.

2.1 ONTOLOGIA E REPRESENTAÇÃO DO CONHECIMENTO

Segundo Russel e Norvig (2004, p. 253), "a palavra ontologia representa uma teoria específica sobre a natureza do ser ou existir". A ontologia permite representar conhecimento sobre qualquer domínio, por exemplo, ações, tempos, objetos físicos e crenças (RUSSELL; NORVIG, 2004, p. 309-310).

O termo ontologia na Engenharia do Conhecimento e na Ciência da Computação refere-se à representação de um vocabulário relacionado a certo domínio, onde a qualificação não está no vocabulário, mas sim nos conceitos expostos por ele. Desta forma, tem-se as chamadas ontologias de domínio (CHANDRASEKARAN; JOSEPHSON, 1999, p. 20-21).

A representação de conhecimento é a denominação dada aos métodos usados para modelar a informação relacionada a certo domínio. Trata-se de um conjunto de convenções sintáticas e semânticas, que torna possível descrever um mapeamento entre os objetos e as relações envolvidos neste domínio. A representação sintática especifica os símbolos que podem ser usados e as maneiras de defini-los, enquanto que a representação semântica especifica os significados incorporados a estes símbolos. Contudo, uma ontologia não é conceitualmente uma base de conhecimentos, mas pode tornar a ser quando é vinculada a uma aplicação (HEINZLE, 2011, p. 93-94).

De acordo com Heinzle (2011, p. 103), as ontologias empregam objetos básicos na formalização do conhecimento: classes, relacionamentos, axiomas e instâncias. As classes são as unidades básicas de toda ontologia. Elas representam coleções de termos que possuem atributos (características) e formulam conceitos. As ligações entre estas unidades se dão através dos relacionamentos ou relações. Os axiomas são regras relativas às relações e servem

para definir a semântica entre os termos. Já as instâncias representam os elementos da ontologia, ou seja, são os exemplares individuais das classes.

2.2 ONTOLOGY DEVELOPMENT 101

Ainda não existe um modo correto ou metodologia de desenvolvimento de ontologias em torno da qual haja consenso entre os autores. Por outro lado, alguns deles recomendam a adoção de um processo denominado *Ontology Development 101* (HEINZLE, 2011, p. 189). Este processo consiste em um guia de sete passos iterativos, que podem ser parcialmente ou integralmente executados, para a construção de uma ontologia (NOY, 2001, p. 4).

De acordo com Rautenberg et al. (2008, p. 243-244), resumidamente os sete passos do guia *Ontology Development 101* são:

a) determinar o domínio e o escopo da ontologia: deve-se identificar claramente o propósito e os cenários de utilização da ontologia a ser desenvolvida;
b) considerar o reuso de ontologias existentes: é aconselhável verificar a existência de ontologias que podem ser reutilizadas em um novo projeto, a fim de não se "reinventar a roda" e/ou proporcionar a interação da ontologia desenvolvida com outras aplicações;
c) enumerar termos importantes do domínio da ontologia: relacionar uma lista de termos presentes no discurso do domínio da ontologia. A relação de termos é necessária para os passos subsequentes do guia, como definir classes, definir propriedades e definir instâncias;
d) definir as classes do domínio e a hierarquia de classes: a partir da lista de termos, extrae-se aqueles que descrevem objetos, os quais genericamente especificam;
e) definir as propriedades das classes: a partir da lista remanescente de termos, deve ser observados se eles correspondem a propriedades de dados ou de relações de classe para uma determinada classe;
f) definir as restrições das propriedades: caso uma propriedade de classe seja de dados, observa-se o tipo de dado que a propriedade comporta (*string* ou número, por exemplo). Caso a propriedade seja uma relação, deve-se definir a que classes a relação aponta. Restrições sobre cardinalidade e valores válidos para as propriedades também devem ser considerados neste passo;

g) criar as instâncias do domínio: finalmente cria-se as instâncias da ontologia a partir da definição das classes, valorando suas propriedades de dados e relações.

2.3 MODELAGEM DE SISTEMAS

Considerando a representação do conhecimento na modelagem de sistemas, de acordo com Pressman (2006, p. 102-103), esta é uma abordagem fundamental das etapas de geração dos sistemas. Nesta se permite o foco sob a visão de mundo ou em visões mais detalhadas, criando padrões que definem:

a) os processos, condizendo com as necessidades em perceber o que está sendo avaliado;
b) o comportamento dos processos, identificando os pressupostos nos quais o comportamento está baseado;
c) explicitamente as entradas, tanto com visões conjuntas quanto individualizadas para cada modelo;
d) as ligações que permitirão os engenheiros de software entenderem melhor a visão do sistema.

> O modelo de sistema resultante (em qualquer visão) pode adotar uma solução completamente automatizada, uma solução semi-automatizada ou uma abordagem não automatizada. Na realidade, é frequentemente possível caracterizar modelos de cada tipo que servem como solução alternativa para o problema em mãos. (PRESSMAN, 2006, p. 103).

Contudo, segundo observado por Campos (2008, p. 119) sobre o ato de modelar, define-se que este deve subentender o conhecimento do objeto que se está modelando, considerando-se sua complexidade. Já que, não conhecer o objeto de modelagem, implica em não representá-lo adequadamente e utilizar inapropriadamente ferramentas e métodos possivelmente ineficazes na sua construção. Desta forma, ocasionando perda de qualidade, insatisfação e custos elevados na concepção dos sistemas.

2.4 OWL E XMI

Como forma de interpretação do conhecimento definido por ontologias, tem-se a *Web Ontology Language* (OWL). Esta é uma linguagem que integra as tecnologias recomendadas pelo *World Wide Web Consortium* (W3C) desde fevereiro de 2004. É baseada na sintaxe do *eXtensible Markup Language* (XML) (HEINZLE, 2011, p. 115).

Segundo a W3C (2004), a linguagem foi projetada para aplicações que necessitam processar o conteúdo das informações, não apenas apresentar estas aos seres humanos. Desta forma, seu uso é indicado quando se pretende:

a) formalizar um domínio por meio da definição de classes e suas propriedades;
b) definir instâncias e suas propriedades;
c) raciocinar a respeito destas classes e instâncias.

Já o XML *Metadata Interchange* (XMI) consiste em um padrão definido pela *Object Management Group* (OMG) para a troca de informações baseado em XML. Tendo como principal objetivo, a troca facilitada de metadados entre as ferramentas de modelagem e repositórios de metadados, baseados no *Unified Modeling Language* (UML) (KATZ; EBERHARDT, 2004, p. 5).

Assim sendo, de acordo com a OMG (2005, p. 8-9), um modelo XMI apresenta como elementos básicos: classes, atributos e tipos de dados. Além de permitir a definição associativa entre estes elementos. Onde conjuntamente, inclui-se o mecanismo de herança entre as classes.

2.5 JSON

Objetivando a troca de dados simplificada foi definida a formatação JavaScript *Object Notation* (JSON). Uma vez que, para seres humanos é fácil ler e escrever e, para máquinas é fácil interpretar e gerar. Além disso, está baseada em um subconjunto da linguagem de programação JavaScript (CROCKFORD, 2009).

Segundo Crockford (2009), as linguagens em sua maioria possuem elementos que se mapeiam facilmente para objetos JSON. Entre eles, pode-se destacar: objetos, estruturas, registros, dicionários, tabelas de hash, listas de propriedade e matrizes relacionais.

Desta forma, define-se o texto possibilitando a serialização de dados compostos. Sendo que estes conseguem representar, conjuntamente, quatro tipos primitivos (*strings*, números, booleanos, e nulos) e dois tipos estruturados (objetos e *arrays*). Ondo os objetos são referencias à própria estruta do modelo (CROCKFORD, 2006, P. 1).

2.6 COMPUTAÇÃO EM NUVEM

Quanto à hospedagem de sistemas de informação tem-se a computação em nuvem. Esta é um modelo onipresente, conveniente que, com acesso à rede sob demanda, permite o compartilhamento do *pool* de recursos computacionais configuráveis. Onde podem ser rapidamente provisionados e liberados com mínimo esforço de gerenciamento ou interação com o provedor de serviços (MELL; GRANCE, 2011, p. 2).

Além disso, ela oferece muitos benefícios para as organizações, como: a redução de custos, já que as empresas não precisam mais gastar com infraestrutura própria e também manutenção. Junto à melhora na automação dos serviços, flexibilidade e sustentabilidade (BAKSHI, 2011, p. 6).

De acordo com Mell e Grance (2011, p. 2-3), nuvens em geral prestam serviços em três níveis diferentes:

a) *Software as a Service* (SaaS) – a capacidade fornecida ao consumidor é usar o provedor de aplicações rodando em uma infraestrutura de nuvem. As aplicações são acessíveis a partir de dispositivos clientes, através de interface como um navegador web ou programa aplicativo;
b) *Platform as a Service* (PaaS) – a capacidade fornecida ao consumidor é implantar para a nuvem aplicações de infraestrutura criadas ou adquiridas usando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor;
c) *Infrastructure as a Service* (IaaS) – a capacidade fornecida ao consumidor é a disposição de processamento, armazenamento, redes e outros recursos de computação fundamentais onde o consumidor é capaz de implantar e executar software arbitrário, que pode incluir sistemas operacionais e aplicações.

2.7 NOSQL

Permitindo o melhor aproveitamento do processamento que o ambiente em nuvem permite, tem-se o BD *Not Only Structured Query Language* (NoSQL). O termo NoSQL surgiu em 1998, a partir de uma solução de BD que não oferecia uma interface *Structured Query Language* (SQL), mas este sistema ainda era baseado na arquitetura relacional. Posteriormente, o termo passou a representar soluções que promoviam uma alternativa ao Modelo Relacional (MR) (AMAZON INC., 2012).

Sendo que a arquitetura relacional trabalha muito bem com processos de validação, verificação e garantias de integridade dos dados, controle de concorrência, recuperação de falhas, segurança, controle de transações, dentre outros. Facilitando a rotina dos programadores, possibilitando que estes possam se preocupar exclusivamente com o foco da aplicação. (BRITO, 2011, p. 1).

Porém, o ponto em que o BD NoSQL apresenta as principais vantagens em relação aos BD relacionais é a questão do escalonamento, basicamente pelo fato de terem sido criados para este fim, enquanto os sistemas relacionais possuem uma estruturação menos flexível e menos adaptada para cenários em que o escalonamento faz-se necessário. Além disso, a questão da performance na busca dos dados também poderia ser apontada. Justificada pelo fato do NoSQL não fornecer todo o arcabouço de regras de consistência presentes nos MR (BRITO, 2011, p. 2-3).

Sendo assim, segundo Brito (2011, p. 2), o propósito das soluções NoSQL não é substituir o MR como um todo, mas apenas em casos nos quais seja necessária maior flexibilidade da estruturação das bases de dados.

2.8 TRABALHOS CORRELATOS

A seguir são relatados dois trabalhos correlatos: 1) Rautenberg et al. (2010) que apresenta a ferramenta OntoKEM, numa concepção que segue o princípio de definição de ontologia que este trabalho se propõe, e; 2) Gonda; Jodal (2011) que relata o desenvolvimento da ferramenta Genexus e que também compreende alguns preceitos deste trabalho.

2.8.1 OntoKEM

A OntoKEM é uma ferramenta *case* baseada na Web, de propósito acadêmico para documentação e construção de projetos de ontologias. Permite a especificação, conceitualização, formalização e documentação de ontologias (RAUTENBERG et al., 2010, p. 244).

A principal vantagem e justificativa de utilização desta ferramenta é a geração automática de artefatos customizados. Sendo que, estes artefatos privilegiam a atividade de documentação durante o processo de desenvolvimento de ontologias (RAUTENBERG et al., 2010, p. 248).

A ferramenta tem por base: o guia *Ontology Development 101*, que constitui uma visão clara do processo do desenvolvimento de ontologias e a metodologia *On-To-Knowledge*, que se preocupa principalmente com a viabilidade da ontologia. Como resultado, gera um arquivo no formato OWL (RAUTENBERG et al., 2010, p. 245-247).

2.8.2 Genexus

O Genexus é um produto de mercado que pretende automatizar o processo de gerar e manter os sistemas computacionais, prometendo uma mudança de paradigma: “descrever” em vez de “programar”. Portanto, define-se a partir de modelos externos de regras as bases de conhecimentos, permitindo associar um conjunto de deduções sobre os processos (GONDA; JODAL, 2011, p. 1, 3).

Segundo Ganda e Jodal (2011, p. 2), a essência do Genexus é um grande conjunto de subprodutos para:

a) projeção, geração e manutenção automática da base de dados e dos programas de aplicação necessários para cada empresa;
b) geração de componentes de software, a partir do mesmo conhecimento, para plataformas múltiplas;
c) integração do conhecimento de diversas fontes para atender a necessidades muito complexas, com custos em tempo e dinheiro bem inferiores aos habituais.

3 DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo são apresentadas as etapas do desenvolvimento do protótipo de uma plataforma composta por três ferramentas para a geração de sistemas baseados em ontologia, assim como a demonstração de uso destas ferramentas para: a definição de ontologias de domínio, a conversão do modelo ontológico para orientado a objetos e a geração de cadastros e consultas típicos de sistemas de informação. São abordados os principais requisitos, a especificação, a implementação e ao fim os resultados e discussão.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O protótipo da plataforma para desenvolvimento de SI, a partir de cada representação do conhecimento definida por ontologias deverá:

a) fornecer o mecanismo para a especificação do propósito de cada ontologia (Requisito Funcional - RF);
b) fornecer o mecanismo para a especificação do escopo de cada ontologia (RF);
c) fornecer o mecanismo para a definição hierárquica de classes inerentes a cada ontologia (RF);
d) fornecer o mecanismo para a definição relacional de classes inerentes a cada ontologia (RF);
e) fornecer o mecanismo para a definição de propriedades inerentes a cada classe própria de ontologias (RF);
f) converter o grafo ontológico para o formato JSON (RF);
g) gerar a definição da linguagem OWL, a partir das demarcações formuladas junto ao formato JSON (RF);
h) interpretar as definições da linguagem OWL (RF);
i) gerar a definição da linguagem XMI, a partir da linguagem OWL (RF);
j) gerar a partir da linguagem XMI, definições de classes do UML (RF);
k) gerar os componentes de software para o padrão de projeto arquitetural MVC (RF);
l) gerar as rotinas de cadastros sob conceitos de computação em nuvem (RF);

m) gerar as rotinas de consultas sob conceitos de computação em nuvem (RF);
n) gerar as rotinas de cadastros empregando-se a linguagem Java, na versão 7 (RF);
o) gerar as rotinas de consultas empregando-se a linguagem Java, na versão 7 (RF);
p) desenvolver as ontologias utilizando-se de recursos visuais em três dimensões (Requisito Não-Funcional - RNF);
q) desenvolver as ontologias utilizando-se de recursos de arrastar e soltar junto ao *mouse*. (RNF);
r) desenvolver as ontologias seguindo o processo iterativo *Ontology Development 101* (RNF);
s) persistir os dados inerentes as ontologias geradas, em BD NoSql (RNF);
t) desenvolver as aplicações utilizando-se da linguagem de programação Java, na versão 7 (RNF);
u) desenvolver as aplicações sob conceitos de computação em nuvem (RNF).

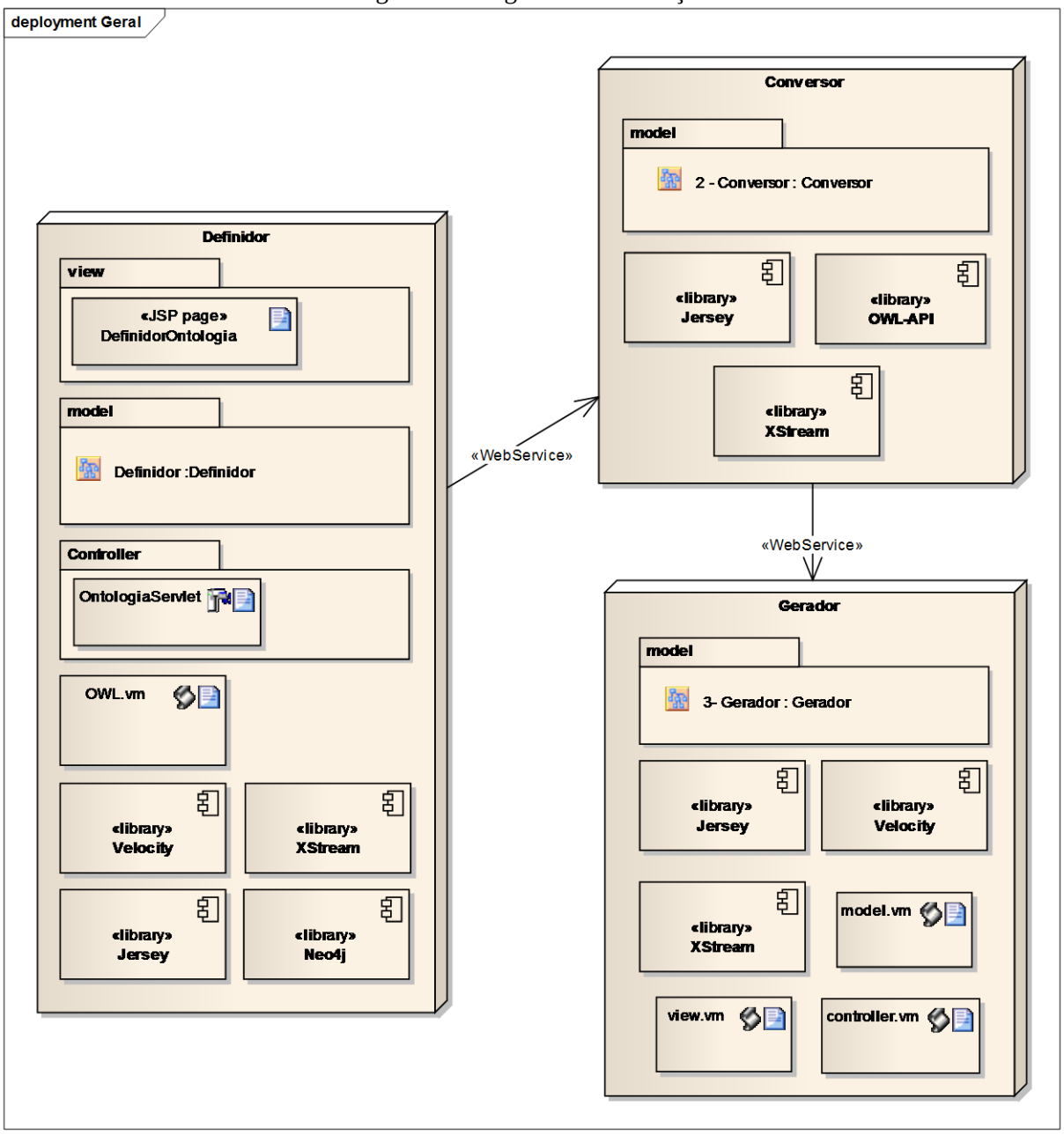
3.2 ESPECIFICAÇÃO

A especificação deste trabalho foi desenvolvida utilizando a ferramenta Enterprise Architect na versão 7.5. Empregou-se os conceitos do paradigma de orientação a objetos, por meio de diagramas do UML. Assim, nas seções seguintes são apresentados os diagramas de instalação, casos de uso, classes, sequência e comunicação.

3.2.1 Diagrama de instalação

Na Figura 1, expõe-se os componentes do protótipo de plataforma para geração de sistemas baseados em ontologias. Neste detalha-se os artefatos principais das três ferramentas objetivadas.

Figura 1 – Diagrama de instalação



Conforme pode ser observado na Figura 1, a primeira ferramenta, está intitulada `Definidor`, possui sua arquitetura modularizada sob o modelo MVC. Esta ferramenta é um sistema web. Desta forma, o artefato `DefinidorOntologia` compõe a camada de visão e o `OntologiaServlet` a camada de controle. Na mesma ferramenta, tem-se o artefato `OWL.vm`, utilizado na geração do código no formato OWL. Ainda se percebe quatro bibliotecas: `Velocity`, `XStream`, `Neo4j` e `Jersey`. A duas primeiras são assistenciais ao processo de geração de componentes ontológicos, a terceira ao processo de gravação da ontologia em BD e a última, ao processo de integração nesta ferramenta descrita.

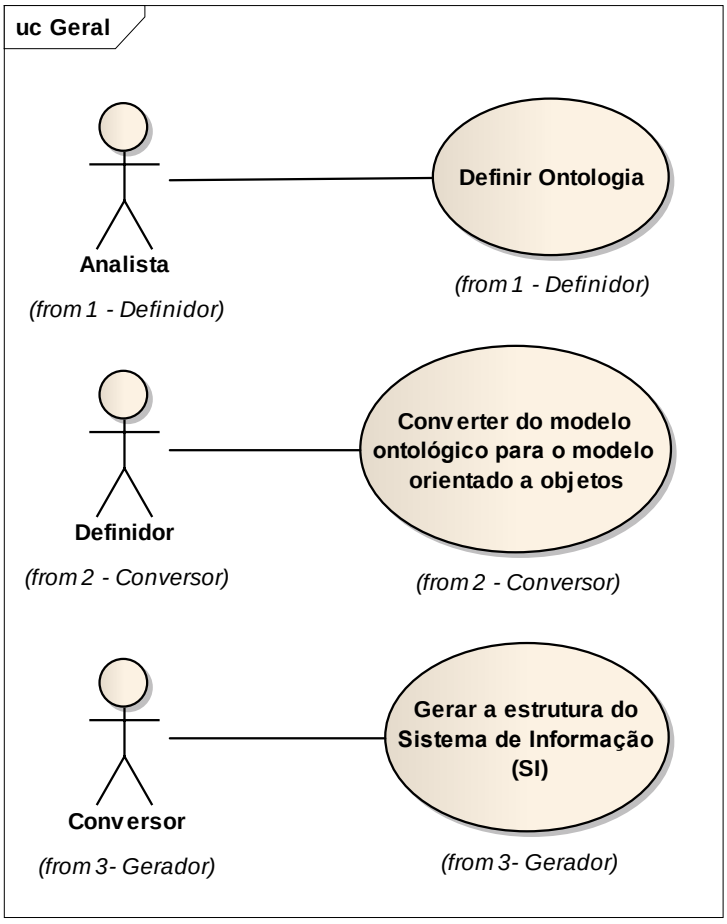
Na sequência, utiliza-se a comunicação via web *service*, da ferramenta descrita, a ferramenta denominada `Conversor`. Esta possui também o pacote de artefatos `model`. O mesmo tem classes responsáveis pelo processo de conversão de modelos definidos por arquivos do formato OWL em XMI. Portanto, utiliza-se, como apoio a este procedimento, as bibliotecas: `XStream` e `OWL-API`. E mais uma vez se vê a biblioteca `Jersey`, utilizada para a integração desta ferramenta.

Por último, tem-se a ferramenta chamada `Gerador`. Esta também se utiliza da tecnologia web *service* para a comunicação entre as ferramentas. Além disso, se constata novamente o pacote de artefatos: `model`. Este traz por objetivo a construção de componentes do modelo MVC. Para tal premissa, apoia-se nas bibliotecas: `Velocity` e `XStream`. Mas, além destas, utiliza-se dos artefatos de *script*: `model.vm`, `view.vm` e `controller.vm`, para a geração dos códigos inerentes ao modelo de três camadas. E outra vez, percebe-se a biblioteca `Jersey`. Esta é igualmente utilizada para a integração nesta ferramenta.

3.2.2 Diagrama de casos de uso

Nesta seção são descritos os casos de uso das funcionalidades principais das ferramentas propostas.

Figura 2 – Diagrama de casos de uso



Conforme mostrado na Figura 2, o ator que terá contato com a ferramenta de definição da ontologia é o `Analista`. Este ator utilizará a ferramenta para o desenvolvimento de ontologias de domínio. Os demais atores são as próprias ferramentas desenvolvidas neste trabalho: `Definidor` e `Conversor`.

A ação inicial do `Analista` deve ser realizada conforme descrito no caso de uso `Definir Ontologia`, apresentado no Quadro 1.

Quadro 1 – Detalhamento do caso de uso Definir Ontologia

Definir ontologia	
Pré-condições	O servidor deve estar operando com a ferramenta em execução.
Cenário Principal	1) O analista solicita a criação de uma nova ontologia. 2) O definidor fornece uma tela em branco, contendo o componente `Thing`. 3) O analista solicita a definição de uma nova classe a partir do elemento `Thing`. 4) O definidor fornece uma tela para definir o nome da classe. 5) O analista informa o nome da classe e confirma. 6) O definidor finaliza a tela. 7) O analista solicita a definição de atributos para a nova classe. 8) O definidor fornece uma tela para definir os nomes e tipos dos atributos. 9) O analista informa os nomes e tipos dos atributos, e confirma. 10) O definidor finaliza a tela. 11) O analista volta a executar a partir do passo 3 ou segue a execução no passo 12. 12) O analista informa as duas classes e solicita a definição de relação entre elas. 13) O definidor fornece uma tela para definir o nome da relação. 14) O analista informa o nome, e confirma. 15) O definidor finaliza a tela. 16) O analista solicita a finalização do processo. 17) O processo é encerrado e a tela finalizada.
Exceção 1	18) No passo 5 do Cenário Principal, caso o analista informe um nome de classe que já esteja determinado, o definidor não aceita a confirmação.
Exceção 2	19) No passo 9 do Cenário Principal, caso o analista informe um nome de atributo que já esteja determinado, o definidor não aceita a confirmação.
Pós-condições	Geração a partir da definição da ontologia do arquivo no formato OWL.

Na sequência, após a geração do artefato no formato OWL, a ferramenta de acepção, representada pelo ator `Definidor`, executa a ferramenta de conversão. Acompanha-se os passos conforme detalhado no caso de uso `Converter do modelo ontológico para o orientado a objetos`, visto no Quadro 2.

Quadro 2 – Detalhamento do caso de uso Converter do modelo ontológico para o orientado a objetos

Converter do modelo ontológico para o modelo orientado a objetos	
Pré-condições	Ter como base um arquivo no formato OWL.
Cenário Principal	1) O definidor envia um arquivo no formato OWL para a conversão. 2) O conversor recebe o arquivo no formato OWL. 3) O conversor transforma o arquivo para o formato XMI. 4) O conversor envia esse arquivo à ferramenta de geração de componentes de software.
Pós-condições	Geração do arquivo no formato XMI.

Finalmente, após a conversão, chama-se a partir do ator `Conversor` a ferramenta para a geração dos componentes de software inerentes ao modelo MVC. Os passos são detalhados no caso de uso `Gerar a estrutura do SI`, conforme o Quadro 3.

Quadro 3 – Detalhamento do caso de uso Gerar a estrutura do SI

Gerar a estrutura do SI	
Pré-condições	Ter por base um arquivo no formato XMI.
Cenário Principal	1) O conversor envia um arquivo no formato XMI para a geração dos componentes de software. 2) O gerador recebe o arquivo no formato XMI. 3) O gerador interpreta o arquivo recebido. 4) O gerador cria as estruturas de cadastros baseadas na interpretação. 5) O gerador cria as estruturas de consultas baseadas na interpretação.
Pós-condições	Geração de rotinas de software baseadas no arquivo no formato XMI.

3.2.3 Diagrama de classes

O diagrama de classes deste trabalho é dividido em três seções. Cada seção é responsável por detalhar as classes principais de cada ferramenta proposta.

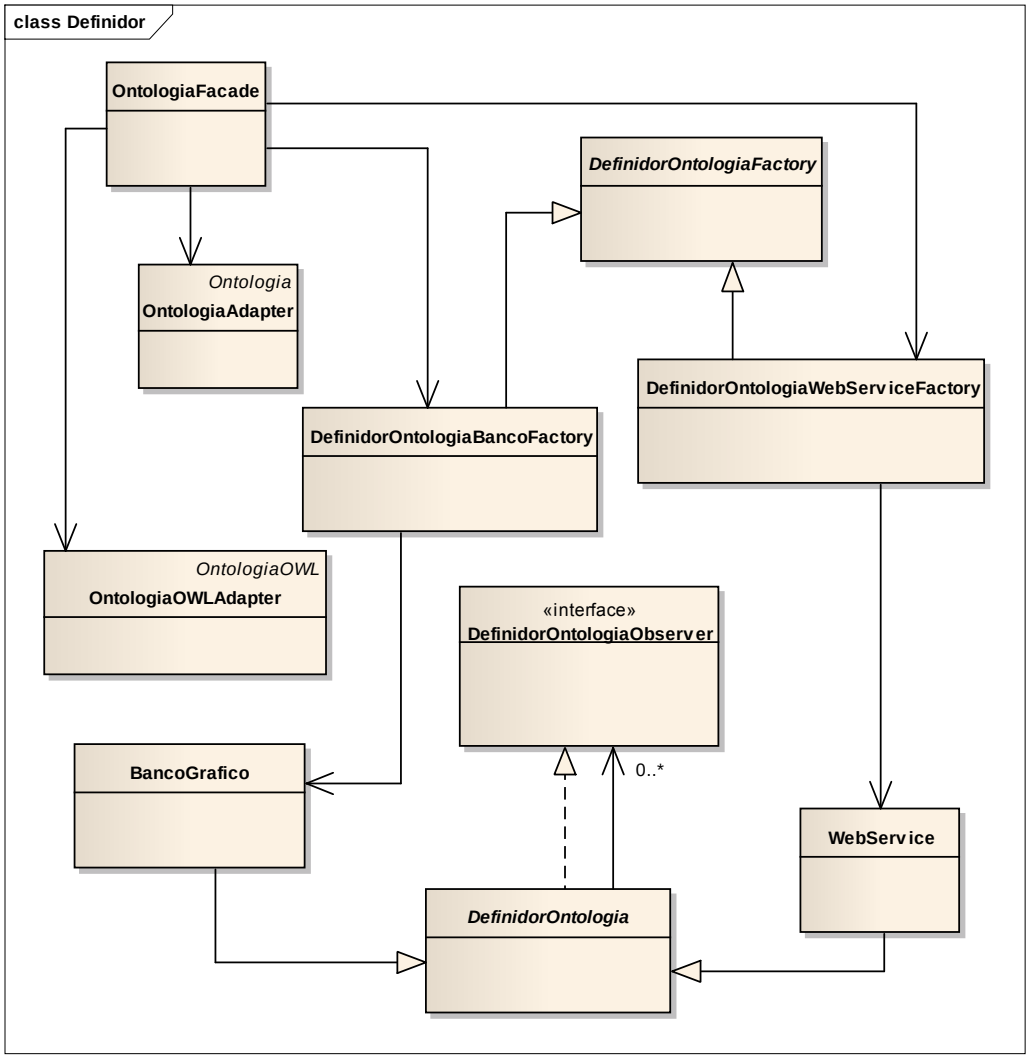
3.2.3.1 Ferramenta de definição, conceitualização e formalização de ontologias de domínio

Na Figura 3, exibe-se o diagrama de classe da ferramenta para a definição, conceitualização e formalização de ontologias de domínio.

A classe fundamental desta ferramenta é a `OntologiaFacade`. Esta é responsável pela execução das demais entidades. Considerando os papéis das classes, executa-se a classe `OntologiaAdapter`; onde esta se responsabiliza pela conversão da ontologia do formato JSON para objetos Java. A classe `DefinidorOntologiaBancoFactory` se responsabiliza pela execução da classe `BancoGrafico`, onde implementa-se o armazenamento da ontologia em um BD NoSql.

A classe `DefinidorOntologiaObserver` é responsável por observar o processo de gravação da ontologia e executar a classe `WebService`, definida a partir da entidade `DefinidorOntologiaWebServiceFactory`. Esta envia o código no formato OWL, gerado pela classe `OntologiaOWLAdapter`, para a ferramenta conversora.

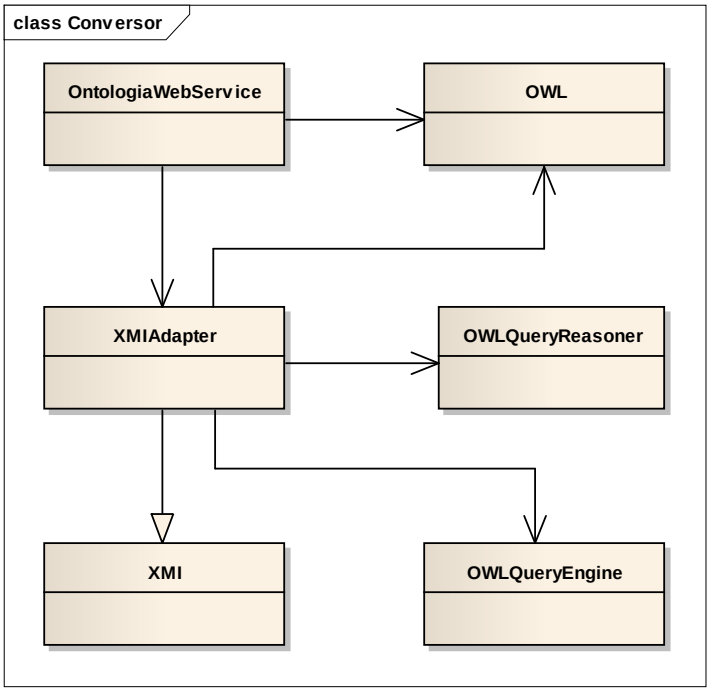
Figura 3 – Diagrama de classes da ferramenta para a geração de ontologias de domínio



3.2.3.2 Ferramenta para a conversão de modelos ontológicos em orientados a objetos

Na Figura 4, apresenta-se o diagrama de classe da ferramenta para a conversão de modelos ontológicos para orientados a objeto.

Figura 4 – Diagrama de classes da ferramenta de conversão



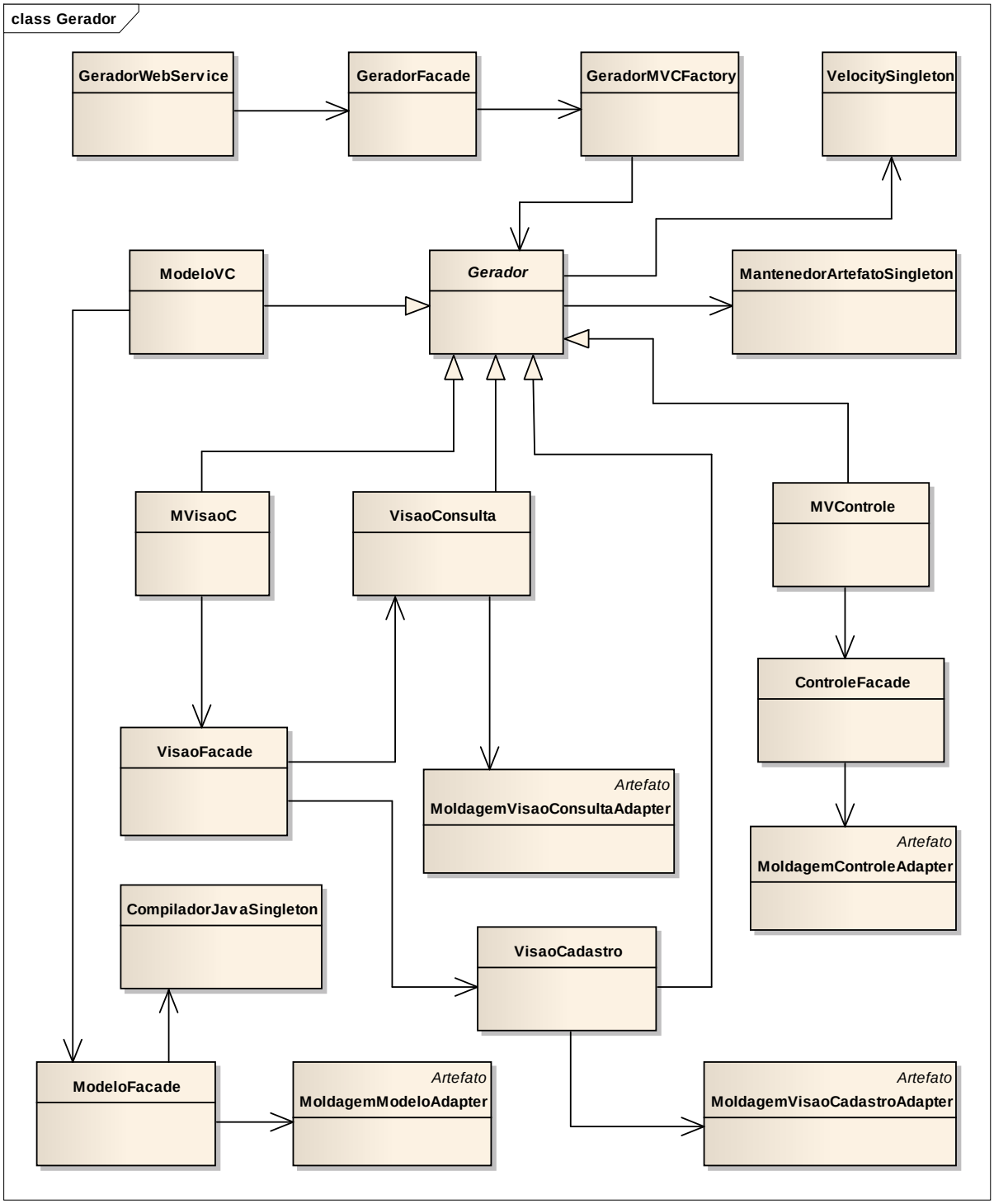
A partir da Figura 4, nota-se como classe principal a `OntologiaWebService`. A mesma, realiza a chamada à classe `XMIAdapter`. Esta permite a conversão da ontologia do formatado OWL, junto à classe `OWL`, para a ontologia no formato XMI, atrelada à classe `XMI`.

Para tal processo, possibilitado pela classe `OWLQueryReasoner`, executam-se as consultas à instância da classe OWL, a partir da classe `OWLQueryEngine`. Estas consultas permitem a busca de informações da ontologia, junto a um mecanismo de inferência, para a geração do modelo orientado a objetos. Este modelo é enviado à ferramenta de geração de componentes de software.

3.2.3.3 Ferramenta para a geração de componentes de software inerentes às camadas do modelo MVC

Na Figura 5, apresenta-se o diagrama de classe da ferramenta para a geração de componentes de software, baseados em modelos ontológicos.

Figura 5 – Diagrama de classes da ferramenta para a geração de componentes de software



Conforme mostra a Figura 5, a execução da ferramenta advém da classe `WebService`. Esta instancia a classe `GeradorMVCFactory`, para a execução das entidades: `ModeloVC`, `MVisaoC` e `MVControle`. Estas três entidades permitem, integralmente, a geração dos componentes do padrão MVC; a partir de suas respectivas classes executoras: `ModeloFacade`, `VisaoFacade` e `ControleFacade`. Exceto para a camada de Visão, que é formada por rotinas

de cadastros e consultas. Por este motivo, definiu-se, adicionalmente, as classes `VisaoCadastro` e `VisaoConsulta`.

Compondo também as classes da ferramenta, tem-se os processos de conversão das entidades, definidas no formato XMI, em elementos do padrão MVC. Sendo assim, a classe `MoldagemModeloAdapter` responsabiliza-se pela geração de elementos da camada de Modelo. Para a geração dos componentes inerentes à camada de Visão, fez-se uso das classes: `MoldagemVisaoCadastroAdapter` e `MoldagemVisaoConsultaAdapter`. Estas permitem, respectivamente, a construção das telas de cadastro e consulta. Por fim, para a constituição dos elementos de software responsáveis pela camada de Controle, utilizou-se a classe `MoldagemControleAdapter`.

Todo este "arcabouço de classes" é amparado por entidades coadjuvantes: `VelocitySingleton`, `MantenedorArtefatoSingleton` e `CompiladorJavaSingleton`. Estas permitem respectivamente, a geração dos artefatos a partir de modelos, a gravação dos mesmos no disco rígido e a compilação das classes Java.

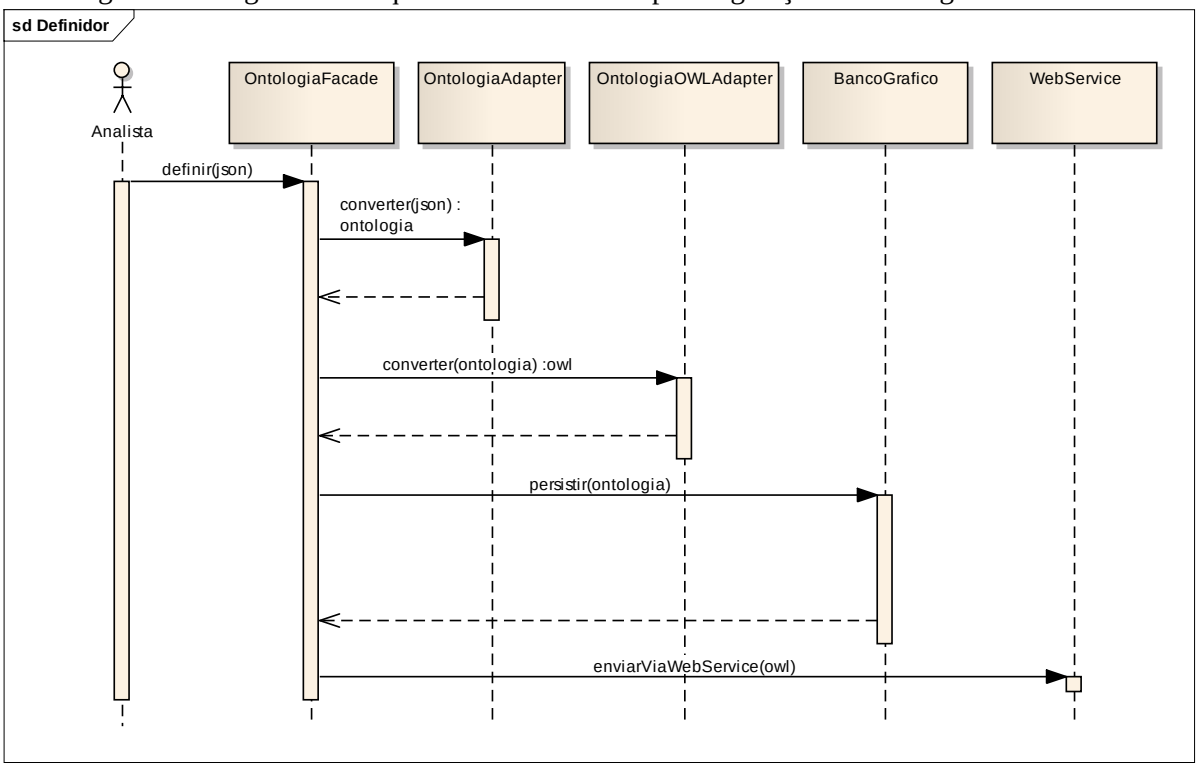
3.2.4 Diagrama de sequência

Nesta seção são apresentados os fluxos principais de execução das ferramentas para a definição das ontologias e a geração dos componentes de software.

3.2.4.1 Ferramenta de definição, conceitualização e formalização de ontologias

Na Figura 6 é apresentado o diagrama de sequência da ferramenta de definição, conceitualização e formalização de ontologias. Nele detalha-se as principais classes funcionais.

Figura 6 – Diagrama de sequência da ferramenta para a geração de ontologias de domínio

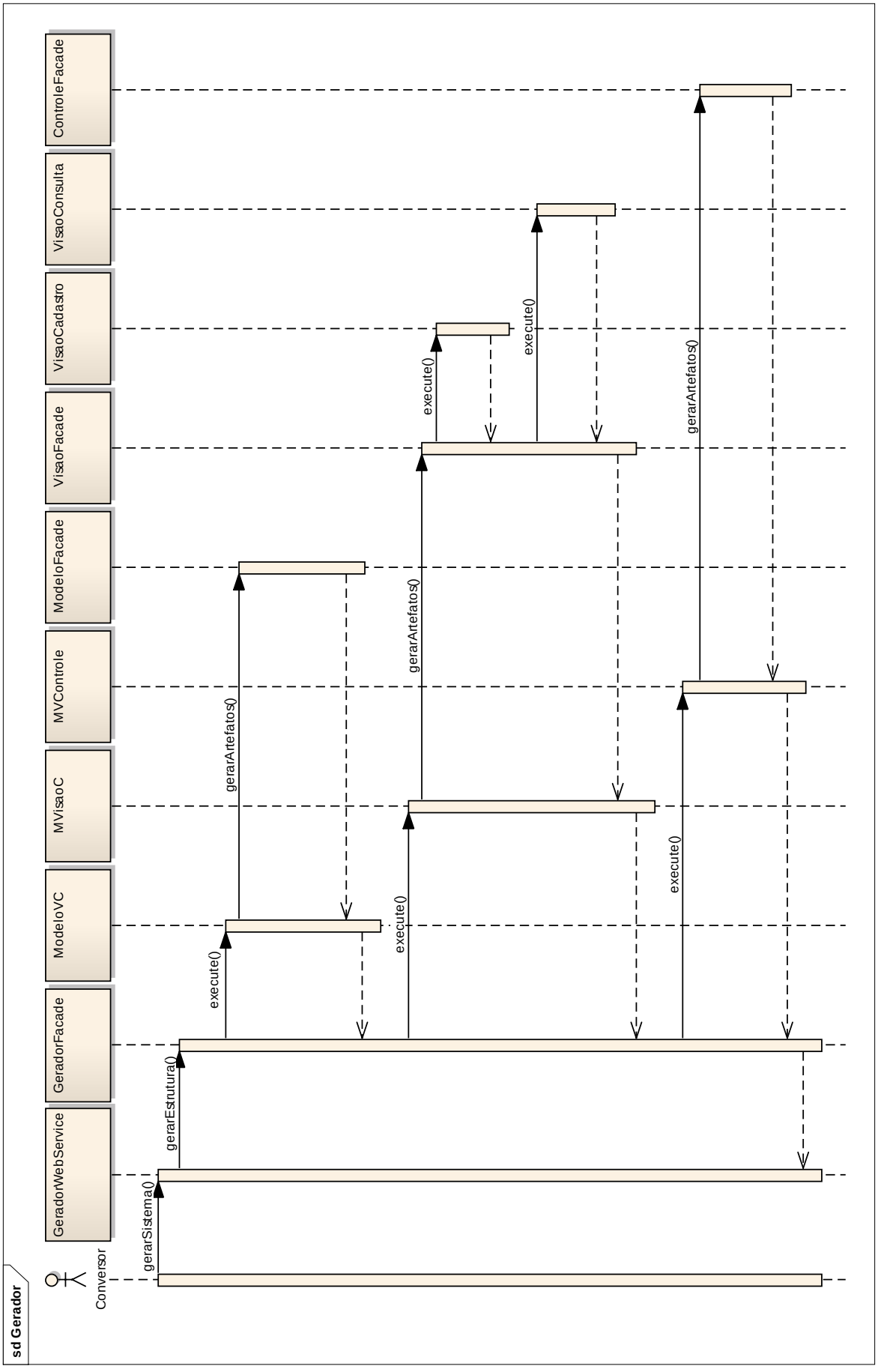


Conforme se mostra na Figura 6, primeiramente o ator `Analista` solicita a definição da ontologia. Esta definição é realizada a partir de um objeto da classe `OntologiaFacade` com o método `definir` e baseia-se em um artefato no formato JSON. Na sequência, chama-se o método `coverter`, de um objeto da classe `OntologiaAdapter`, para a definição de componentes ontológicos. Posteriormente, realiza-se outra conversão, a partir do método `converter`, junto a um objeto da classe `OntologiaOWLAdapter`. Nesta se gera o artefato ontológico no formato OWL. Então, finalmente se grava a ontologia em um BD, a partir de um objeto da classe `BancoGrafico` com o método `persistir`. Por último, envia-se o documento ontológico via web s*ervice* à ferramenta conversora. Este processo ocorre com o auxílio de um objeto da classe `WebService`, acoplado ao método `enviarViaWebService`.

3.2.4.2 Ferramenta para geração de componentes de software inerentes as camadas do modelo MVC

Na Figura 7, apresenta-se o diagrama de sequência da ferramenta para geração de componentes de software inerentes as camadas do modelo MVC. Neste detalha-se as principais classes operacionais da ferramenta.

Figura 7 – Diagrama de sequência da ferramenta para a geração de componentes de software



Conforme se detalha na Figura 7, primeiramente o ator `Conversor` solicita ao método `gerarSistema` de um objeto da classe `GeradorWebService`, a geração dos componentes de software. Esta geração é realizada a partir de um objeto da classe `GeradorFacade` com o método `gerarEstrutura`. Na sequência, chama-se os objetos das classes responsáveis pela definição das camadas do modelo MVC.

Desta forma, inicialmente invoca-se, sequencialmente, os métodos `execute` de objetos das classes: `ModeloVC`, `MVisaoC` e `MVControle`. Posteriormente, cada objeto das classes listadas anteriormente gera os componentes das camadas do modelo MVC; junto aos métodos `gerarArtefatos`. Estes implementados, respectivamente, sob as classes: `ModeloFacade`, `VisaoFacade` e `ControleFacade`. Entretanto, adicionamente à classe `VisaoFacade`, chama-se os métodos `execute` das classes `VisaoCadastro` e `VisaoConsulta`. Ambos se responsabilizam pela geração dos artefatos inerentes às telas de cadastro e consulta, respectivamente.

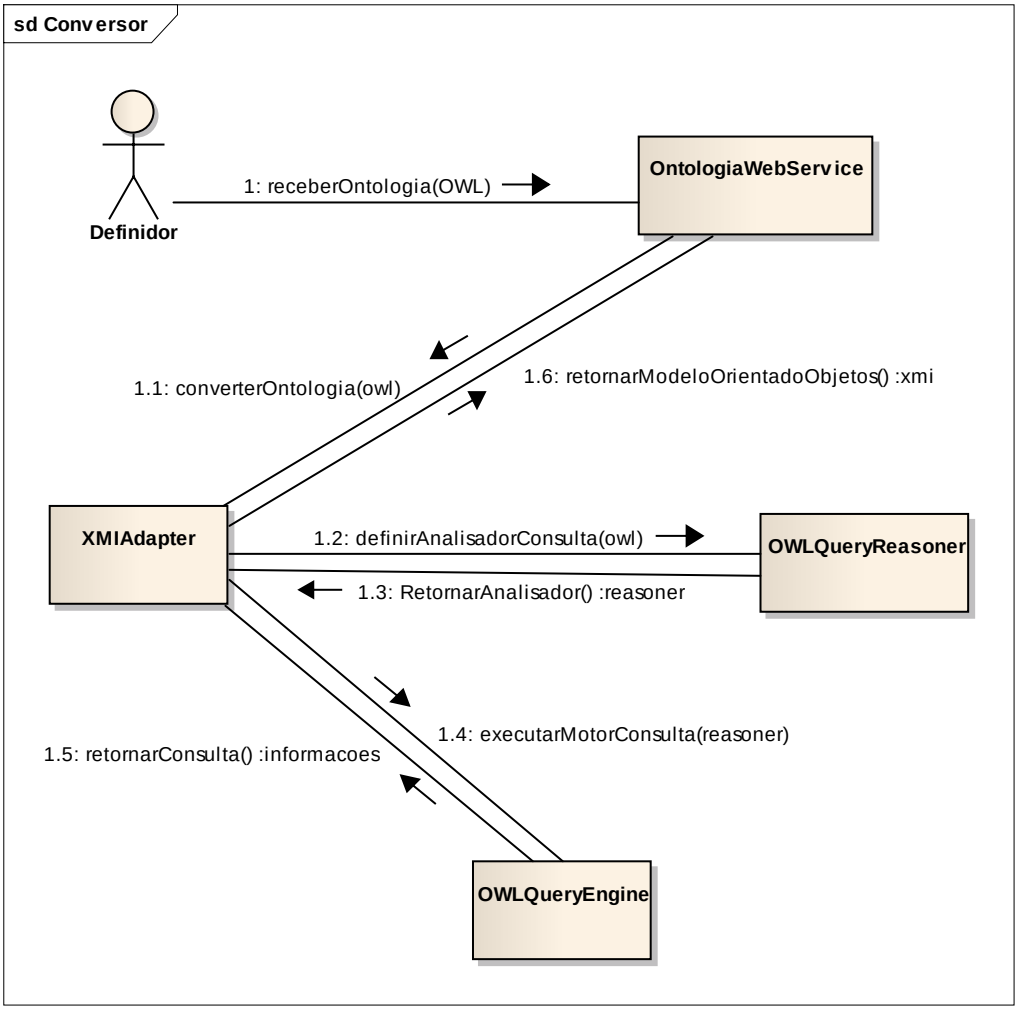
3.2.5 Diagrama de comunicação

Nesta seção, detalha-se junto à Figura 8, a iteração para conversão de modelos ontológicos para orientados a objetos, considerando-se o contexto da ferramenta. Nesta simulação, apresenta-se as classes principais da ferramenta.

Considerando-se a execução do processo de conversão, primeiramente o ator `Definidor` solicita o recebimento do artefato ontológico no formato OWL, junto à classe `ConversorWebService`, sob o método `receberOntologia`. Na sequência, executa-se a estrutura da conversão do modelo ontológico no artefato orientado a objetos no formato XMI.

Para tal processo, chama-se da classe `XMIAdapter` o método `converterOntologia`. Este invoca o método `definirAnalisadorConsulta` da classe `OWLQueryReasoner`, passando à chamada, o artefato ontológico. Este método retorna um elemento `reasoner`. O mesmo permite realizar consultas à classe `OWLQueryEngine`, com o procedimento `executarMotorConsulta`. Finalmente, a partir destas informações, cria-se o artefato no formato XMI.

Figura 8 – Diagrama de comunicação da ferramenta de conversão



3.3 IMPLEMENTAÇÃO

Esta seção descreve as técnicas e ferramentas utilizadas, junto à operacionalidade da implementação. Em seguida, são detalhadas as principais classes e apresentados os trechos de código-fonte para facilitar o entendimento. Mostrando-se brevemente como foram realizadas as implementações de funcionalidades principais, focadas em usabilidade. Por fim, é apresentada a operacionalidade do protótipo.

3.3.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento das três ferramentas propostas, empregou-se a linguagem de programação Java. Como *Integrated Development Environment* (IDE) utilizou-se o Eclipse.

Para o desenvolvimento gráfico da ferramenta de definição, conceitualização e formalização de ontologias de domínio, utilizou-se o WebGL (KHRONOS, 2012). Segundo Khronos (2012), esta biblioteca multiplataforma web, baseada no *Open Graphics Library for Embedded Systems* (OpenGL ES) tem por finalidade fornecer uma interface de programação de gráficos de três dimensões.

Para a troca de informações entre a camada de visão e a de controle da ferramenta de definição de ontologias de domínio, fez-se uso do formato JSON, juntamente com a biblioteca XStream (THE CODEHAUS, 2013). De acordo com The Codehaus (2013), esta possibilita a serialização de objetos, a partir da estrutura Java, para os formatos XML ou JSON e vice-versa.

A persistência destas informações ontológicas foi possível com o BD Neo4j (NEO TECHNOLOGY, 2013). Segundo Neo Technology (2013), este banco utiliza-se de recursos orientados a grafos. Estes, por sua vez, com propriedades transacionais, permitem a geração de um modelo de dados extremamente ágil em relação ao modelo definido sob o BD relacional.

Na sequência, a geração do arquivo no formato OWL foi viabilizada com o mecanismo de *templates* Apache *Velocity* (APACHE FOUNDATION, 2013). Desta forma, de acordo com a Apache Foundation (2013), este disponibiliza o *template-engine*, que possibilita a geração de artefatos baseados em modelos no formato *Velocity Template Language* (VTL).

Referente à ferramenta para conversão de modelos ontológicos para orientados a objetos, utilizou-se de forma conjunta à biblioteca XStream, a *Application Programming Interface* (API) Java OWL-API (SCHNEIDER, 2010). Segundo Horridge e Bechhofer (2009, p. 11), esta API oferece além do suporte à criação e manipulação de ontologias OWL, o mecanismo de consulta, em aplicações, às ontologias.

Na integração das ferramentas citadas anteriormente, utilizou-se o modelo Web *service* Jersey (ORACLE CORPORATION, 2013). Segundo a Oracle Corporation (2013), este fornece suporte para a criação de serviços web de acordo com o *Representational State Transfer* (REST). Além disso, para simplificar o desenvolvimento e implantação de clientes e

serviços web, faz-se uso de anotações introduzidas no Java *Standard Edition* (SE), na sua quinta versão.

Na última ferramenta, para geração de componentes de software inerentes as camadas do modelo MVC, utilizou-se novamente o mecanismo de *templates* Apache *Velocity* e a biblioteca XStream. Na integração da ferramenta de conversão com esta, aproveitou-se outra vez do web *service* Jersey.

Após o desenvolvimento das ferramentas, as mesmas foram disponibilizadas junto ao *Amazon Elastic Compute Cloud* (Amazon EC2). Segundo Oliveira Junior e Bonini (2013, p. 59), este é um serviço web que oferece capacidade computacional redimensionável em nuvem e possui uma rotina de gerenciamento simplificada. Além disso, ele permite a criação de instâncias de servidores em poucos minutos. O Apêndice A mostra a tela utilizada para a configuração deste servidor.

3.3.2 Implementação das ferramentas

Esta seção detalha em nível de código-fonte, o desenvolvimento das três ferramentas que compõem o protótipo de uma plataforma para a geração de sistemas baseados em ontologias.

3.3.2.1 Ferramenta de definição, conceitualização e formalização de ontologias de domínio

Tendo em vista o objetivo da ferramenta, optou-se em unir a definição ágil e visual de ontologias com recursos tridimensionais. Desta forma, permite-se o usuário cria as classes, associações, e instâncias da ontologia utilizando-se destes recursos.

Para tal estrutura, preferiu-se pelo desenvolvimento da ferramenta em ambiente web. Sendo assim, para o conhecimento dos recurso empregados, que permitem a geração da tela para definição de ontologias de domínio, no Quadro 4, exibe-se os trechos principais do código JavaScript. A implementação usa-se de recursos do WebGL.

Quadro 4 – Código JavaScript responsável pela renderização da tela para a definição de ontologias

```
1:    container = document.createElement( 'div' );
2:    document.body.appendChild( container );
3:
4:    camera = new THREE.PerspectiveCamera( 70, window.innerWidth /
                                        window.innerHeight, 1, 10000 );
5:    camera.position.z = 1200;
6:
7:    scene = new THREE.Scene();
8:
9:    scene.add( new THREE.AmbientLight( 0x505050 ) );
10:
11:   var light = new THREE.SpotLight( 0x999999, 1.5 );
12:   light.position.set( 0, 500, 2000 );
13:   light.castShadow = true;
14:
15:   light.shadowCameraNear = 200;
16:   light.shadowCameraFar = camera.far;
17:   light.shadowCameraFov = 50;
18:
19:   scene.add( light );
```

Primeiramente se cria a variável `container` (Quadro 4, linha 1), onde posteriormente insere-se os componentes visuais e acrescenta-os ao corpo da página (Quadro 4, linha 2). Em seguida se cria a câmera, o cenário e a luminosidade do ambiente, parametriza-os e então, os interliga (Quadro 4, linhas 4 a 19). Sendo estes, peças fundamentais para exibir os elementos gráficos em tela.

Após a definição da ontologia na camada de visão, as informações são transferidas para a camada de controle pelo formato JSON. O Apêndice B mostra um exemplo do código neste formato.

Na sequência, junto à camada de modelo, o principal processo da ferramenta de geração valeu-se do padrão de projetos *Facade*. O método base da classe `OntologiaFacade` é detalhado no Quadro 5.

Quadro 5 – Código-fonte do método responsável pela operacionalidade da definição de ontologias

```
1:    OntologiaAdapter ontologia = new OntologiaAdapter(ontologiaJSON);
2:    ontologia.converter();
3:
4:    OntologiaOWLAdapter ontologiaOWl = new OntologiaOWLAdapter(ontologia);
5:    ontologiaOWl.converter();
6:
7:    DefinidorOntologiaFactory factoryBanco = DefinidorOntologiaFactory.
                                    obterFactory(Mecanismo.BANCO, ontologia);
8:    DefinidorOntologiaFactory factoryWebService = DefinidorOntologiaFactory.
                               obterFactory(Mecanismo.WEB_SERVICE, ontologiaOWl);
9:
10:   DefinidorOntologia definidorBanco = factoryBanco.obterDefinidor();
11:   DefinidorOntologia definidorWebService = factoryWebService.obterDefinidor();
12:
13:   definidorBanco.cadastrarObserver(definidorWebService);
14:   definidorBanco.definirOntologia();
```

Inicialmente, necessita-se a conversão da ontologia do formato JSON para objetos Java (Quadro 5, linha 2), possibilitando-se a persistência e a conversão para o código OWL (Quadro 5, linha 5), procedendo-se o envio deste à ferramenta de conversão e geração da orientação a objetos. Ambas as conversões foram possibilitadas utilizando-se do padrão *Adapter* (Quadro 5, linha 1 e 4).

Para seleção entre a criação dos mecanismos de banco, permitindo a persistência (Quadro 5, linha 7), e o web *service*, este comportando o envio da ontologia ao conversor de códigos (Quadro 5, linha 8), utilizou-se o padrão de projetos *Factory*. Empregou-se também, o padrão *Observer*, junto à variável `definidorBanco`. Esta vinculada à rotina de gravação dos dados no banco, representada pela variável `definidorWebService` e permite o envio da OWL à ferramenta conversora (Quadro 5, linha 13). Desta forma, sob qualquer alteração no banco, estrategicamente, o web *service* é avisado. Porquanto, caso ocorra algum erro, o processo não se estenderá às outras ferramentas.

A seguir, nos Quadro 6 e Quadro 7, detalha-se os códigos de conversão dos adaptadores invocados no Quadro 5.

Quadro 6 – Código-fonte do método responsável pela conversão do código JSON para objetos Java

```
1:    XStream stream = new XStream(new JettisonMappedXmlDriver());
2:    Ontologia ontologia = (Ontologia) stream.fromXML(ontologiaJSON);
```

A classe `OntologiaAdapter` é responsável pela conversão do código do formato JSON em objetos Java. A mesma, utilizando-se da biblioteca XStream, com base no *driver*

`JettisonMappedXmlDriver` (Quadro 6, linha 1), permite a conversão da variável *String* `ontologiaJSON` para objetos Java (Quadro 6, linha 2).

Quadro 7 – Código-fonte do método responsável pela geração do código no formato OWL

```
1:    VelocityEngine engine = new VelocityEngine();
2:    engine.init();
3:    Template template = engine.getTemplate("templates/ontologia/OWL.vm");
4:    VelocityContext context = new VelocityContext();
5:    context.put("ontologia", ontologia);
6:    StringWriter writer = new StringWriter();
7:    template.merge(context, writer);
8:    String ontologiaOwl = writer.toString();
```

A classe `OntologiaOWLAdapter` é responsável por converter o código do formato JSON para o formato OWL, utilizando-se do mecanismo de *template Velocity* (Quadro 7, linhas 1 e 2). Para tal conversão, definiu-se o contexto (Quadro 7, linha 4) e inseriu-se os objetos Java vinculados ao atributo `ontologia` (Quadro 7, linha 5), permitindo-se o *merge* dos dados com o *template* no formato VTL (Quadro 7, linha 7). O arquivo `OWL.vm` encontra-se listado no Apêndice C. Por fim, tem-se como resultado, o código no formato OWL (Quadro 7, linha 8). Este consta no Apêndice D.

Para a persistência da ontologia utilizou-se o BD orientado a grafos Neo4j. No Quadro 8, segue a implementação de um dos métodos de inserção da classe `BancoGrafico`. Os demais métodos adotam estruturas semelhantes.

Quadro 8 – Código-fonte do método responsável de gravação das ontologias

```
1:    public void addClasse(Long classePai, Classe classe) {
2:        Transaction tx = graphDb.beginTx();
3:        try {
4:            Node outroNode = graphDb.createNode();
5:            outroNode.setProperty("id_classe", classe.getId());
6:            outroNode.setProperty("classe", classe);
7:
8:             if (classePai != null) {
9:                 ReadableIndex<Node> index = graphDb.index().
                                                 getNodeindexer().getindex();
10:                Node umNode = index.get("id_classe",
                                          classePai).getSingle();
11:                if (umNode != null) {
12:                    umNode.createRelationshipTo(outroNode,
                                           TipoRelacionamento.HERANCA);
13:                }
```

Quadro 8 – Código-fonte do método responsável de gravação das ontologias (continuação)

```
14:             }
15:
16:              tx.success();
17:          } catch ( Exception e )     {
18:              tx.failure();
19:          } finally {
20:              tx.finish();
21:          }
22:    }
```

No método `addClasse`, inicia-se uma nova transação com o banco (Quadro 8, linha 2), e então se cria um nodo para compor a classe ontológica (Quadro 8, linha 4). Em seguida, se verifica se a classe está vinculada a outra, junto à variável `classePai` (Quadro 8, linha 8). Encontrando-se o vínculo, busca-se o índice ligado à classe (Quadro 8, linha 9). Desta forma, encontra-se a classe superior (Quadro 8, linha 10) e cria-se o relacionamento entre os nodos (Quadro 8, linha 12). Por fim, com o sucesso em todos os passos anteriores, grava-se os dados no banco em forma de grafo (Quadro 8, linha 16).

O envio do código no formato OWL para a ferramenta de conversão dar-se conforme o código da classe `GeradorWebService`, demonstrado-se a seguir no Quadro 9.

Quadro 9 – Código-fonte do método responsável pelo envio do código para conversão

```
1:    String URL_CONVERSOR = "http://servidor:porta/Conversor/rest/ontologia/";
2:
3:    Client client = Client.create();
4:    WebResource webResource = client.resource(URL_CONVERSOR);
5:
6:    webResource.post(String.class, ontologiaOWL));
```

A variável `URL_CONVERSOR` possui o endereço da ferramenta de conversão de modelos ontológicos em orientados a objetos; por este princípio, cria-se um cliente Jersey (Quadro 9, linha 3) e associa-se à variável `webResource`, apontada a partir do cliente (Quadro 9, linha 4). Com este recurso, envia-se via *post* a *string* `ontologiaOWL` (Quadro 9, linha 6).

3.3.2.2 Ferramenta para conversão de modelos ontológicos para orientados a objetos

Para a integração da ferramenta de definição de ontologias com esta ferramenta conversora, criou-se um servidor Web *service* Jersey. Este recebe um artefato no formato OWL. A estrutura da classe será mostrada no Quadro 10.

Quadro 10 – Código-fonte da classe responsável pela integração das três ferramentas

```
1:    @Path("/ontologia/")
2:    public class OntologiaWebService {
3:        @POST
4:        @Consumes(MediaType.TEXT_PLAIN)
5:        public String receberOntologia(String owlStr) {
6:            String retorno = "S";
7:            try {
8:                OWL owl = new OWL(owlStr);
9:                XMIAdapter xmi = new XMIAdapter(owl);
10:               xmi.converter();
11:
12:               Client client = Client.create();
13:               final String URL_GERADOR =
                          "http://servidor:porta/Gerador/rest/ontologia/";
14:               WebResource webResource = client.resource(URL_GERADOR);
15:
16:               retorno =  webResource.post(String.class, xmi);
17:           } catch (Exception e) {
18:               retorno =  e.toString();
19:           }
20:           return retorno;
21:       }
22:    }
```

Na declaração da classe insere-se a anotação `@Path` (Quadro 10, linha 1). A mesma possibilita a publicação do web *service*. No método `receberOntologia`, a anotação `@POST` define a forma de recebimento dos dados (Quadro 10, linha 3) e a anotação `@Consumes` define o tipo de dado recebido por parâmetro (Quadro 10, linha 4). Na implementação do método em questão, percebe-se a conversão, a partir do padrão de projetos *Adapter*, do código no formato OWL (Quadro 10, linha 8), para o formato XMI (Quadro 10, linha 10). Finalmente, a constante `URL_GERADOR` possui o endereço da geração de componentes de software. Para este princípio, cria-se um cliente Jersey (Quadro 10, linha 14) e associa a variável `webResource`

apontado a partir do cliente (Quadro 10, linha 12). Então, com este recurso, envia-se via *post* a *String* `xmi` (Quadro 10, linha 16).

A seguir, no Quadro 11, estão os detalhes dos trechos de códigos-fontes principais da classe `XMIAdapter`.

Quadro 11 – Código-fonte do método responsável pela conversão do código OWL para a XMI

```
1:    final IRI ONTO_IRI = IRI.create(owl.getOwl());
2:
3:   OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
4:   OWLOntology ontology = manager.loadOntologyFromOntologyDocument(ONTO_IRI);
5:
6:   OWLReasoner reasoner = OWLQueryReasoner.createReasoner(ontology);
7:
8:   ShortFormProvider shortFormProvider = new SimpleShortFormProvider();
9:   OWLQueryEngine dlQueryEngine = new OWLQueryEngine(reasoner,
                                                    shortFormProvider);
10:
11:   Xmi xmi = converter(dlQueryEngine);
12:
13:   XStream stream = new XStream(new DomDriver());
14:   String xmi = stream.toXML(xmi);
```

Todo o mecanismo de conversão dar-se principalmente com o auxílio da OWL-API. Para uso de tal mecanismo, necessita-se definir as propriedades básicas: a constante `ONTO_IRI`, referenciando o código em formato OWL (Quadro 11, linha 1), a variável `manager` (Quadro 11, linha 3) e a própria `ontology` (Quadro 11, linha 4). Além destas, em posse da estrutura ontológica, criam-se as propriedades `reasoner` (Quadro 11, linha 6) e `dlQueryEngine` (Quadro 11, linhas 8 e 9), ambas conjuntamente possibilitam realizar as consultas à ontologia. Em seguida, chama-se o método `converter` para a definição do objeto da classe `Xmi`, contendo a estrutura obtida (Quadro 11, linha 11). Por último, utilizando-se novamente da biblioteca XStream (Quadro 11, linha 13), converte-se a variável `xmi` em código no formato XML (Quadro 11, linha 14).

A seguir, no Quadro 12, mostra-se para exemplificação, o código-fonte de um dos métodos de conversão da estrutura ontológica no formato OWL ao formato XMI. Os demais métodos seguem estruturas análogas.

Quadro 12 – Código-fonte de um dos métodos responsável pela conversão do formato OWL para XMI

```
1:    private void addEscopoClasses(List<PackagedElement> packagedElements,
              OWLOntology ontology, Map<String, PackagedElementClass> classes) {
2:        String propriedade = null, dominio = null,
                  referencia = null, tipo = null;
3:        TypePropertyHref typePropertyHref = null;
4:        PackagedElement packagedElement = null;
5:        boolean isAtributo = false;
6:
7:        for (OWLObjectProperty prop : ontology.getObjectPropInSignature()) {
6:            isAtributo = false;
8:            dominio = prop.getDomains(ontology).toArray()[0].toString();
10:            propriedade = prop.toStringID();
11:
12:           f: for (OWLAnnotationAssertionAxiom anAxiom :
                              prop.getAnnotationAssertionAxioms(ontology)) {
13:               try {
14:                   for (OWLDatatype aProperty :
                                anAxiom.getValue().getDatatypesInSignature()) {
15:                       try {
16:                           tipo = aProperty.toString();
17:                           typePropertyHref =
                                  TypePropertyHref.valueOf(tipo.toUpperCase());
18:                           isAtributo = true;
19:                           break f;
20:                       } catch (ArrayIndexOutOfBoundsException ie) {}
21:                   }
22:               } catch (IllegalArgumentException exception) {}
23:           }
24:
25:           if (isAtributo) {
26:               packagedElement = converterPropriedadeClasse(propriedade,
                                                    typePropertyHref);
27:           } else {
28:               referencia = prop.getRanges(ontology).toArray()[0].toString();
29:               packagedElement = converterAssociacao(dominio, referencia);
30:               converterLink(packagedElements, propriedade,
                                           dominio, referencia);
31:           }
32:           classes.get(dominio).getPackagedElements().add(packagedElement);
33:       }
34:   }
```

Inicialmente, são declaradas as variáveis auxiliadores do método `addEscopoClasses` (Quadro 12, linhas 2 a 5). Na sequência, itera-se sobre as propriedades da ontologia (Quadro 12, linha 7). Em cada propriedade, busca-se os objetos vinculados (Quadro 12, linha 12) e define-se os tipos dos mesmos na variável `typePropertyHref` (Quadro 12, linha 17). Após, verifica-se, se estes são atributos ou referências à classes (Quadro 12, linha 25 e 27). Caso um dos elementos sejam atributos, converte-se o mesmo à nova estrutura, informando-o à propriedade `packagedElement` (Quadro 12, linha 26), junto ao tipo (atributo `typePropertyHref`). Mas, caso um dos elementos seja uma associação, converte-se além da referência, junto ao atributo `referencia` (Quadro 12, linha 29), as ligações às outras classes (Quadro 12, linha 30), vinculando-as também à propriedade `packagedElement`. Finalmente, adiciona-se os elementos à nova estrutura, considerando as classes em evidência (Quadro 12, linha 32).

3.3.2.3 Ferramenta para geração de componentes para as camadas do modelo MVC

Em recepção ao código no formato XMI enviado pela ferramenta de conversão da ontologia, foi criado mais um servidor Web *service* Jersey. A estrutura da classe `GeradorWebService` é semelhante à classe `OntologiaWebService` exibida no Quadro 10 da seção 3.3.2.2 deste trabalho.

Na sequência, auxiliada pelo padrão de projetos *Factory*, executa-se a classe `GeradorFacade`, e chama-se o método `gerarArtefatos`, onde este se responsabiliza pela mecânica da ferramenta geradora. O código-fonte do método `gerarArtefatos` detalha-se a seguir, no Quadro 13.

Quadro 13 – Código-fonte do método responsável pela geração da estrutura do modelo MVC

```
1:    Sistema sistema = new MoldagemSistemaAdapter(xmi);
2:    GeradorMVCFactory factory = new GeradorMVCFactory();
3:    Gerador geradorModelo = factory.getGerador(MVC.MODEL, sistema);
4:    Gerador geradorVisao = factory.getGerador(MVC.VIEW, sistema);
5:    Gerador geradorControle = factory.getGerador(MVC.CONTROLLER, sistema);
6:    Gerador geradorModelo.execute(); geradorVisao.execute();
8:    geradorControle.execute();
```

Primeiramente se necessita a conversão do código do formato XMI para objetos Java (Quadro 13, linha 1). Em seguida, auxiliado pelo padrão de projetos *Factory* (Quadro 13,

linha 2), junto à classe `GeradorMVCFactory`, instancia-se os geradores do modelo MVC (Quadro 13, linha 3 a 5). Na sequência, os mesmo são executados, possibilitando-se a construção dos componentes (Quadro 13, linha 6 a 8).

Para o desenvolvimento da classe `MoldagemSistemaAdapter`, adota-se uma estrutura semelhante à implementada na classe `XMIAdapter`, detalhada no Quadro 11 da seção 3.3.2.2 deste trabalho. Exceto que neste caso, gera-se a estrutura de objetos Java a partir do código XMI. Para isto, utiliza-se o método `fromXML` da clase `XStream`.

Os geradores dos componentes do modelo MVC, implementados, respectivamente, a partir das classes `GeradorModelVC`, `GeradorMVisaoC` e `GeradorMVControle`, adotam o mesmo mecanismo de definição de código praticado na classe `OntologiaOWLAdapter` e detalhado no Quadro 7 da seção 3.3.2.1 deste trabalho. Assim sendo, os *templates Velocity* utilizados para a geração dos componentes, encontram-se listados nos apêndices E, F e G.

A seguir, no Quadro 14, exibe-se os trechos do código-fonte responsáveis pela compilação e geração dos arquivos com a extensão `.class`. Esta implementação está contida na classe `CompiladorJavaSingleton`.

Quadro 14 – Código-fonte do método responsável pela compilação das classes Java

```
1:    final String URL_APLICACAO = Utils.getCanonicalPath("WEB-INF",
                                this.getClass().getClassLoader().
                                getResource("br").getFile().substring(1));
2:    final String URL_PATH_SERVER_FILES = URL_APLICACAO+"lib\\";
3:    final String URL_JAVA_HOME = System.getProperty("java.home")+"\\lib\\";
4:
5:    StringBuilder classpath = new StringBuilder();
6:
7:    String[] jars = {"rt.jar", "resources.jar", "jsse.jar",
                           "jce.jar", "charsets.jar"};
8:
9:    for (int i = 0; i < jars.length; i++) {
10:       classpath.append((i == 0 ? "" : ";")+URL_JAVA_HOME+jars[i]);
11:   }
12:
13:   jars = {"hibernate3.jar", "hibernate-jpa-2.0-api-1.0.0.Final.jar",
                          "primefaces-3.2.jar", "javax.faces-2.1.7.jar"};
14:
```

Quadro 14 – Código-fonte do método responsável pela compilação das classes Java (continuação)

```
15:   for (int i = 0; i < jars.length; i++) {
16:       classpath.append((i == 0 ? "" : ";")+URL_PATH_SERVER_FILES+jars[i]);
17:   }
18:
19:   Process process = Runtime.getRuntime().exec(new String[] {"cmd.exe", "/C",
                        "javac -classpath "+classpath.toString()+arquivoCompilar});
20:   process.waitFor();
```

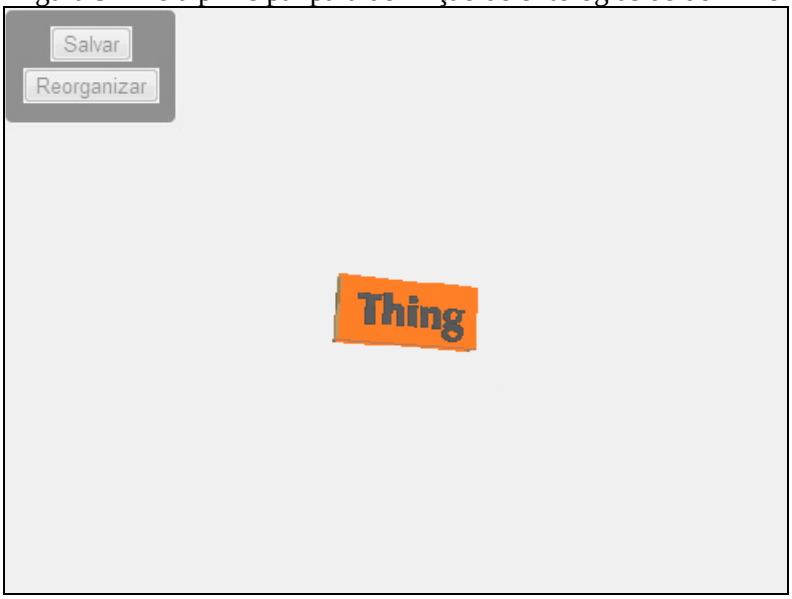
Fundamentalmente, as constantes `URL_APLICACAO`, `URL_PATH_SERVER_FILES`, `URL_JAVA_HOME` são definidas. Estas contêm, respectivamente, os caminhos da aplicação, das bibliotecas desta e do Java (Quadro 14, linhas 1 a 3); ambas as bibliotecas são necessárias para as compilações. Na sequência, cria-se o container `classpath` (Quadro 14, linha 5). Nesta variável, acrescenta-se as bibliotecas vinculadas ao Java (Quadro 14, linha 10) e a aplicação (Quadro 14, linha 16). Por fim, cria-se um processo e compila-se a classe definida na variável `arquivoCompilar` (Quadro 14, linhas 19 a 20).

3.3.3 Operacionalidade da implementação

Nesta seção é apresentado o processo principal para a definição das ontologias. Após, demonstra-se as funcionalidades das ferramentas integradas, a partir do desenvolvimento de um simplificado Sistema de Classificados de Emprego.

Ao todo são três ferramentas que agregam o protótipo. Porém, somente a ferramenta para a definição, conceitualização e formalização de ontologias terá a interação humana. Desta forma, cada pessoa que utiliza esta, chama-se de `Analista`. Estes indivíduos devem ter o conhecimento prévio sobre definição de ontologias. Segue na Figura 9 a tela principal da ferramenta.

Figura 9 – Tela principal para definição de ontologias de domínio



Tendo em vista o objetivo deste trabalho, optou-se em unir a definição ágil e visual de ontologias com recursos de arrastar e soltar em um plano tridimensional. Desta forma, percebe-se na Figura 9, a simplicidade da tela para a geração destas ontologias. Os dois botões vistos nesta figura, permitem além de salvar e/ou alterar a ontologia, a reorganização do grafo gerado pela rotina. Além dos botões citados, ao centro da tela, tem-se o objeto `Thing`. Este serve como ponto de partida na construção das ontologias.

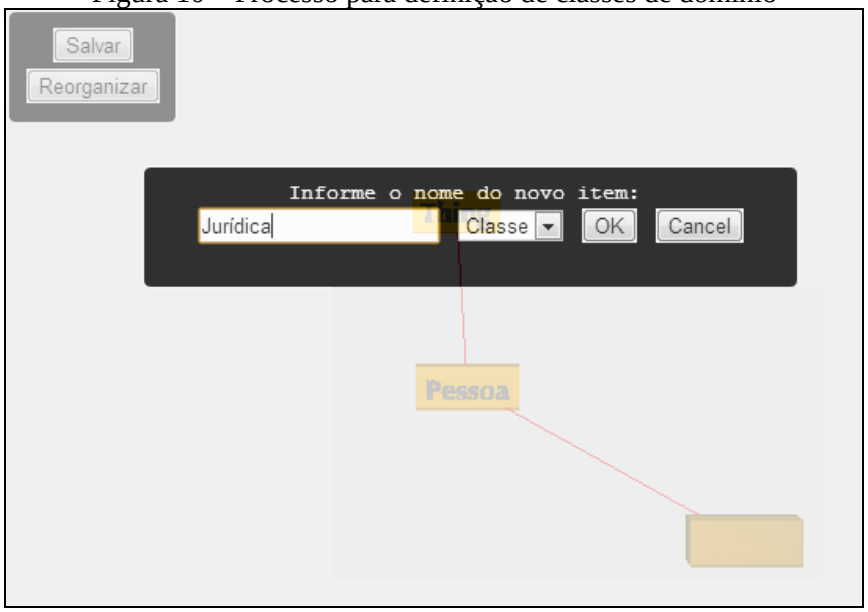
3.3.3.1 Implementação da ontologia para um Sistema de Classificados de Emprego

Evidenciando o uso das ferramentas, definiu-se uma ontologia modelo, seguindo os passos simplificados da metodologia *Ontology Development 101*. Desta forma, as seções seguintes exibem detalhadamente a execução das etapas preconizadas para construção da referida ontologia.

3.3.3.1.1 Definição das classes de domínio

Inicialmente se define as classes do domínio, considerando a hierarquia. A Figura 10 exibe deste processo.

Figura 10 – Processo para definição de classes de domínio

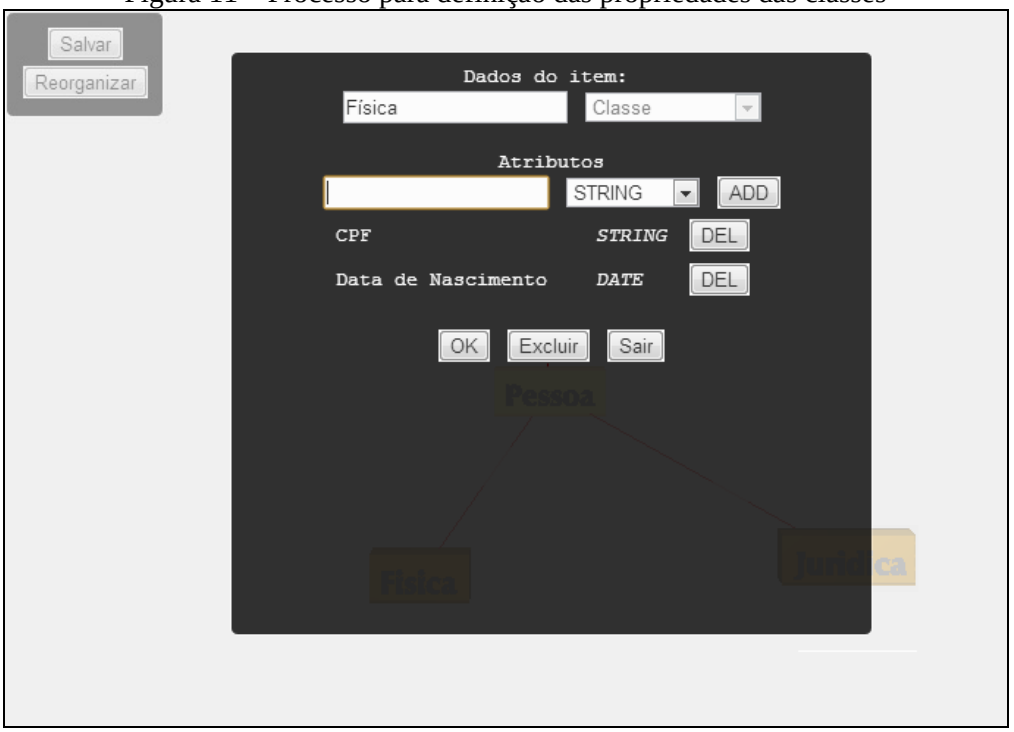


Conforme ilustrado na Figura 10, para a definição de uma nova classe, necessita-se clicar sobre a classe de origem com o botão direito e arrastar o cursor do *mouse* ao outro local da tela que esteja vazio. Após, pede-se o nome da nova classe e ao clicar no botão `OK`, gera-se a mesma e constrói-se a hierarquia entre as entidades.

3.3.3.1.2 Definição das propriedades das classes

Após a definição das classes, determina-se as propriedades destas classes. Este processo apresenta-se na Figura 11.

Figura 11 – Processo para definição das propriedades das classes

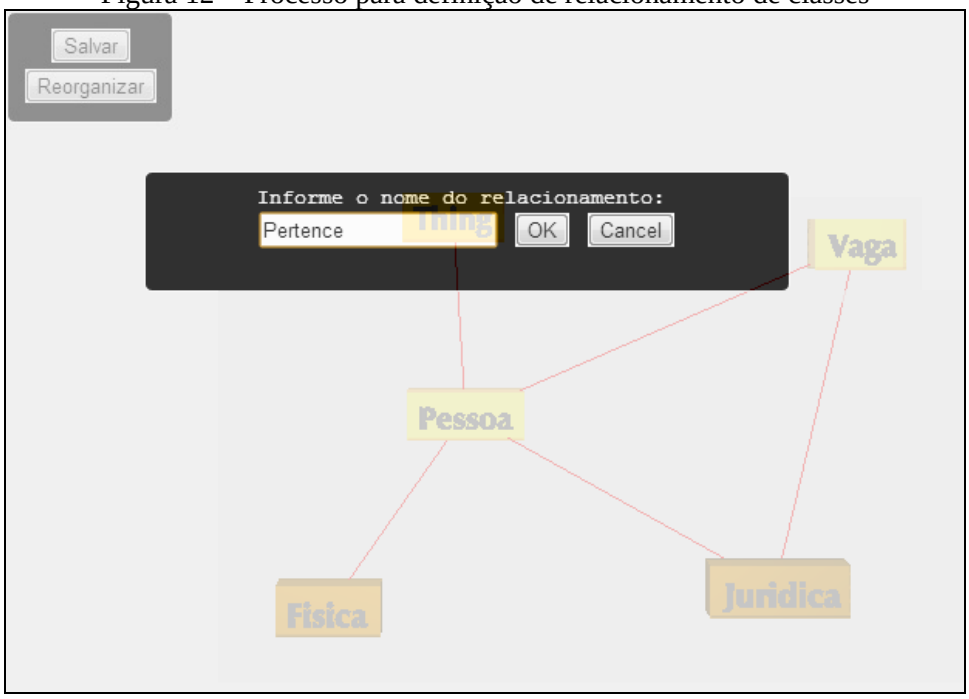


Para vincular as propriedades à classe, conforme a rotina exibida na Figura 11, é necessário clicar duas vezes sobre a classe. Então se exibe a tela do processo de definição. Nesta tela, para cada atributo, informa-se o tipo de dado. Por exemplo: alfanumérico, data, booleano, etc.

3.3.3.1.3 Definição dos relacionamentos de classes

Como passo importante do guia, são definidos os relacionamentos entre as classes. Esta demarcação exemplifica-se na Figura 12.

Figura 12 – Processo para definição de relacionamento de classes

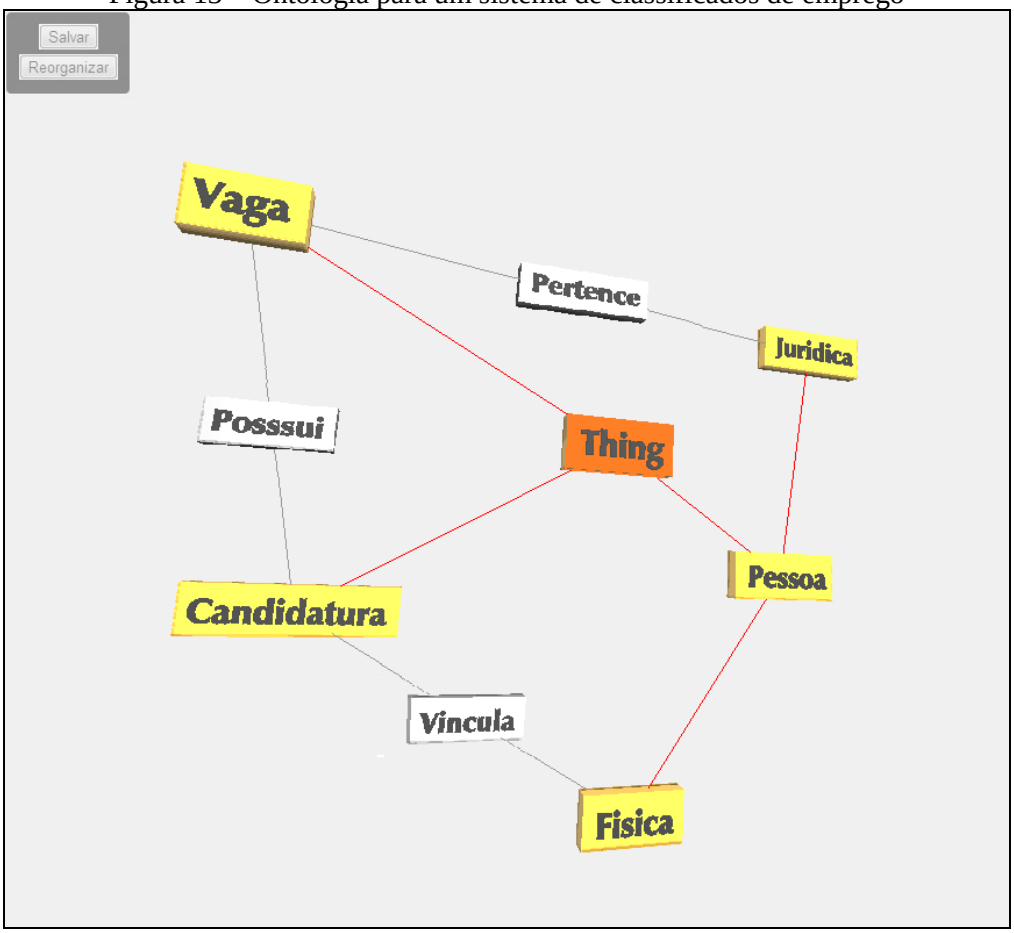


De acordo com o processo exibido na Figura 12, para a definição de um relacionamento entre classes de domínio deve-se: clicar sobre a classe de origem com o botão direito e arrastar o cursor do *mouse* ao local da tela onde se tenha a outra classe. Na sequência pede-se o nome do novo relacionamento. Após informá-lo, clica-se no botão `OK` e o elemento de relação estará definido.

3.3.3.1.4 Resultado da implementação da ontologia

Após os passos detalhados nas seções anteriores, a ontologia resulta no grafo exibido na Figura 13.

Figura 13 – Ontologia para um sistema de classificados de emprego



Notariamente, o grafo completo da ontologia mostrado na Figura 13, apresenta como componentes: o elemento base `Thing` na cor laranja, as classes (com seus atributos implícitos) na cor amarela e em formato retangular, os axiomas na cor cinza e também em formato retangular, e as relações dos elementos citados.

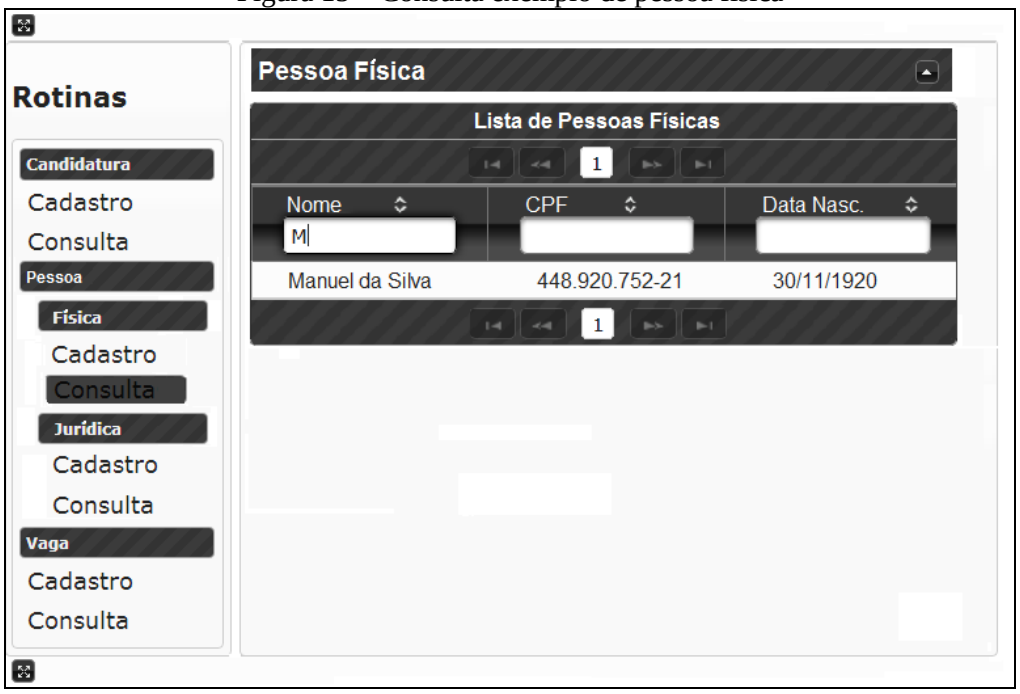
Posteriormente, grava-se a ontologia em uma instância do BD Neo4j. No Apêndice H, tem-se a imagem da ontologia gravada. Além da persistência em banco, a partir da definição da ontologia, gera-se o "espelhamento" das rotinas sob os cadastros de um sistema web. Permitindo a definição de instâncias de domínios às classes determinadas; valorizando-se suas propriedades e relações.

Portanto, para conhecimento do resultado, exibe-se na Figura 14, a rotina de cadastro e, na Figura 15, a rotina de consulta, definidas junto à classe `Fisica` da ontologia. Ambas geradas utilizando-se da mecânica determinada junto a este trabalho.

Figura 14 – Cadastro exemplo para definição de pessoa física

Nas telas geradas, como pode ser visto na Figura 14, tem-se no lado esquerdo, os *links* para acesso às rotinas do sistema. Além disso, no outro lado da tela, constata-se o processo de cadastro gerado sob a entidade Pessoa Física.

Figura 15 – Consulta exemplo de pessoa física



Complementando os resultados, na Figura 15, acompanha-se a tela de consulta, gerada para a busca das pessoas físicas definidas pelo procedimento de cadastro.

3.4 RESULTADOS E DISCUSSÃO

Nesta seção apresenta-se primeiramente as considerações acerca da realização deste trabalho. Na sequência se relata as ponderações sobre o uso de ontologias para a definição de sistemas. Finalmente, realiza-se os comparativos junto aos trabalhos correlatos que possuem objetivos semelhantes ao presente trabalho.

3.4.1 Considerações sobre o desenvolvimento do protótipo

Objetivando a efetivação deste projeto, buscou-se desenvolver as ferramentas utilizando conceitos de computação em nuvem. Além disto, para maior facilidade na manutenção das mesmas, procurou-se aprimorar o entendimento dos código-fontes, alcançados no desenvolvimento nestas ferramentas. Para tal premissa, utilizou-se em várias partes do código desenvolvido, padrões de projetos. Estes amplamente conhecidos no desenvolvimento de software.

Referente à interpretação de amostras ontológicos, a OWL-API mostrou-se útil, visto que conseguiu-se identificar, por meio de consultas, os elementos necessários da ontologia para reaproveitá-los em padrões do UML. Além desta API, a biblioteca XStream e os mecanismos de *templates* Apache *Velocity*, mostraram-se eficazes, respectivamente, na conversão dos modelos e geração de códigos-fontes, já que todos os componentes inerentes ao modelo MVC, para as novas rotinas, foram construídos automaticamente por meio deles.

3.4.2 Ponderações sobre o uso de ontologias para a definição de sistemas

Para o desenvolvimento de SI, através da concepção adquirida a partir de ontologias de domínio, usou-se modelos ontológicos, juntamente à linguagem OWL. Esta para a formalização das informações estruturais e semânticas. Por isso, considerando tal processo, os resultados obtidos se mostraram satisfatórios, visto que a partir da formalização de ontologias

de diversos domínios, permite-se a geração de cadastros característicos de sistemas informatizados.

Quanto à definição das ontologias, seguindo a simplicação do guia *Ontology Development 101*, foi perfeitamente possibilitada, usando-se de recursos gráficos. Estes oferecem um instrumento com o qual se possa formalizar o conhecimento de forma intuitiva, sem exigir maiores noções técnicas relacionadas à acepção de ontologias. Tendo o último passo do guia, a criação das instâncias do domínio, desacoplado dos demais, já que este será praticado nos próprios cadastros gerados, atrelados aos sistemas web.

Desta forma, os passos adaptados do guia *Ontology Development 101* e simplificados para a construção das ontologias são os seguintes:

a) definição das classes de domínio;
b) definição das propriedades das classes;
c) definição dos relacionamentos de classes.

Exceto o passo "considerar o reuso de ontologias existentes" do guia, que não foi contemplado neste trabalho e o passo “criar as instâncias do domínio” também do guia, que foi desacoplado da ontologia e definido junto a cada sistema gerado.

3.4.3 Comparativos dos trabalhos correlatos com este trabalho

Os comparativos deste trabalho entre os correlatos é realizado individualmente. Visto que cada trabalho correspondente possue funcionalidades distintas para efeito de conferência. Portanto, nas seções seguintes, detalha-se estes trabalhos sob elementos de análise.

3.4.3.1 OntoKEM

Para a apreciação comparativa, no Quadro 15, exibe-se as características principais levantadas sobre a ferramenta OntoKEM e a plataforma prototipada neste trabalho.

Quadro 15 – Comparativo deste trabalho com a ferramenta OntoKEM

Características	Este Trabalho	OntoKEM
Baseado no guia Ontology Development 101	X	X
Desenvolvimento visual da ontologia	X	
Documentação automática da ontologia		X
Geração do arquivo no formato OWL	X	X
Permite reusabilidade de ontologias		X
Gera sistemas de informação	X	

A partir do Quadro 15, observa-se que ambos os projetos posssuem como caracteríscas em comum: a definição de ontologias sob os passos do guia *Ontology Development 101* e a geração dos artefatos ontológicos em formato OWL. Porém, observa-se também, diferanças nos dois trabalhos na forma de definir ontologias, pois, este trabalho baseia-se na definição de cada ontologia sob um modelo visual simplificado e, a ferramenta OntoKEM, baseia-se em perguntas pré-formuladas. Desta forma, das respostas a estas perguntas, promoveu-se a documentação automática. Além disso, a ferramenta OntoKEM permite a reusabilidade de ontologias. Por outro lado, este trabalho permite a geração de sistemas de informação baseados em ontologias.

3.4.3.2 Genexus

Para o comparativo com o produto Genexus, no Quadro 16, exibem-se as qualidades observadas junto ao produto, considerando-se que o autor já fez o uso deste para fins profissionais, e as qualidades constatadas junto à plataforma prototipada no presente trabalho.

Quadro 16 – Comparativo deste trabalho com a ferramenta Genexus

Qualidades	Este Trabalho	Genexus
Especificação de sistemas baseada em ontologias	X	
Desenvolvimento de sistemas para diversas plataformas		X
Licença de software livre	X	
Reaproveitamento de bases de conhecimento		X
Uso da ferramenta em ambiente web	X	

Desta forma, conforme se expõe no Quadro 16, considera-se que o produto Genexus e o presente trabalho possuem qualidades distintas. Basicamente se considera como qualidade neste trabalho a simplificação do desenvolvimento dos sistemas, tendo-se como base, elementos vistos no cotidiano das pessoas. Estes amparados por conceitos definidos por ontologias.

As qualidades que o produto Genexus se destaca em relação ao presente trabalho são: o desenvolvimento de sistemas para diversas plataformas e o reaproveitamento de bases de conhecimento. Visto que, este produto possui uma arquitetura mais robusta. Contudo, apoia-se num plano de licenças pagas, que inviabiliza, em alguns casos, o seu uso.

Ainda como diferencial, este trabalho tem como característica importante, o fato de todas as suas ferramentas executarem em um ambiente de nuvem. Ou seja, desta forma, pode-se desenvolver os sistemas de vários locais, a partir de um navegador web.

4 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de uma plataforma para desenvolvimento de SI, a partir de cada representação do conhecimento definida por conceitos ontológicos.

Assim sendo, esta plataforma mencionada foi estruturada e constituída por três ferramentas. A primeira ferramenta concentrou-se na definição de ontologias, baseando-se na simplificação do guia *Ontology Development 101*. Como resultado foi gerado um artefato no formato OWL. A segunda ferramenta empregou-se para a conversão de modelos ontológicos para orientados a objetos, ou seja, fez-se uma via de comunicação entre as tecnologias OWL e XMI, gerando a definição de classes do UML. A terceira ferramenta foi responsável pela geração de artefatos de software inerentes às camadas do modelo MVC, possibilitando a geração de rotinas de cadastros típicas de SI.

Portanto, a partir do presente trabalho, verificou-se a empregabilidade da ontologia em modelar conhecimentos para definição de sistemas informatizados. Além disso, se permitiu uma maior interação entre analistas de sistemas e engenheiros do conhecimento. Possibilitando-se desta forma, a normatização e universalização do conhecimento.

4.1 EXTENSÕES

Durante o desenvolvimento deste trabalho, foram identificados alguns pontos que permitirão a otimização das ferramentas em trabalhos futuros. Pontos estes que não foram contemplados e estão descritos nesta seção.

À ferramenta de definição de ontologias, poderia se acrescentar o conceito de axiomas aos relacionamentos da ontologia. Permitindo, entre outros melhoramentos, determinar a cardinalidade na relação entre as classes. Além disso, também poderia ser vista junto a definição de ontologias, a possibilidade de definir instâncias de classes. Outra questão a ser analisada, seria prever os mecanismos para reutilização e extensão das ontologias. Permitindo-se desta forma, o reaproveitamento de estruturas ontológicas similares, na constituição e geração de sistemas com princípios semelhantes.

Com relação a ferramenta de conversão, a mesma poderia ser amparada tela técnica definida junto ao modelo de sistemas distribuídos: Middleware. Esta permitiria a integração

da ferramenta de definição de ontologias, a várias ferramentas de geração de sistemas e/ou vice-versa. Admitindo-se a construção de sistemas para outras plataformas além do Java *Enterprise Edition* (JEE), como por exemplo, para a plataforma .Net.

À ferramenta responsável pela geração de sistemas, poderia se aplicar os conceitos e mecanismos da ontologia, como a lógica de primeira ordem, para aprimoramento de restrições e regras. Permitindo-se desenvolver, além das rotinas de cadastros e consultas, as rotinas de processos inerentes aos sistemas informatizados. Além disso, se utilizando da linguagem OWL, seria possível aperfeiçoar as rotinas de consultas de cada sistema, baseando-as nos relacionamentos identificados junto aos artefatos ontológicos.

Por último, evidenciando-se a computação em nuvem, além de utilizá-la para hospedar as ferramentas, poderia valer-se da mesma de forma a explorar efetivamente o instrumental previsto em uma completa infraestrutura de nuvem. Utilizando-se dos recursos disponibilizados em cada um dos três níveis do modelo de computação em nuvem.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, Mauricio B.; BARBOSA, Ricardo R. Ontologies in knowledge management support - a case study. **Journal of American Society of Information Science and Technology,** Maryland, v. 60, n. 10, p. 2032-2047, ago. 2009.

ALMEIDA, Mauricio M.; OLIVEIRA, Viviane N. P.; COELHO, Kátia C. Estudo exploratório sobre ontologias aplicadas a modelos de sistemas de informação: perspectivas de pesquisa em Ciência da Informação. **Encontros Bibli: revista eletrônica de biblioteconomia e ciência da informação,** Florianópolis, v. 15, n. 30, p. 32-56, set. 2010. Disponível em: <http://www.periodicos.ufsc.br/index.php/eb/article/view/10987>. Acesso em: 18 set. 2012.

AMAZON INC. **Amazon elastic compute cloud.** Seattle, 2012. Disponível em: <http://aws.amazon.com/ec2/>. Acesso em: 16 set. 2012.

APACHE FOUNDATION. **User Guide –** Apache Velocity. Forest Hill, 2013. Disponível em: <http://velocity.apache.org/engine/devel/user-guide.html#What_is_Velocity>. Acesso em: 27 abr. 2013.

BAKSHI, Kapil. **Considerations for cloud data centers:** framework, architecture and adoption. Herndon: Cisco Systems Inc, 2011.

BRITO, Ricardo W. **Bancos de dados NoSQL x SGBDs relacionais:** análise comparativa. Fortaleza, 2011. Disponível em: <http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos de Dados NoSQL.pdf >. Acesso em: 16 set. 2012.

CAMPOS, Márcio F. **Modelagem, sistemas e informação.** Casos, conceitos e complexidades. Rio de Janeiro, 2008. Disponível em: <http://www.camposmf.eti.br/MSICCC/msiccc.pdf>. Acesso em: 18 set. 2012.

CAMPUS, Maria L. A. Modelização de domínios de conhecimento: uma investigação de princípios fundamentais. **Ciência da Informação**, Brasília, v. 33, n. 1, p. 22-32, 2004.

CHANDRASEKARAN, Bharath; JOSEPHSON, John R. What are ontologies, and why do we need them? **Intelligent systems and their applications,** Amsterdam, vol. 14, n. 1, p. 20-26, jan/fev 1999.

CROCKFORD, Douglas. **Introducing JSON.** São Francisco, 2009. Disponível em: <http://www.json.org/ >. Acesso em: 23 set. 2012.

CROCKFORD, Douglas. **The application/json media type for JavaScript Object Notation (JSON).** São Francisco, 2006. Disponível em: <https://tools.ietf.org/html/rfc4627>. Acesso em: 23 set. 2012.

FOSTER, Ian et al. **Cloud computing and grid computing 360-degree compared.** Chicago, 2008. Disponível em: <http://arxiv.org/ftp/arxiv/papers/0901/0901.0131.pdf>. Acesso em: 25 ago. 2012.

GONDA, Breogán; JODAL, Nicolás. **Desenvolvimento baseado no conhecimento.** Montevidéu, 2011. Disponível em: <http://www.genexus.com/files/desenvolvimento-baseado-no-conhecimento.pdf?pt>. Acesso em: 16 set. 2012.

GUARINO, Nicola. **Formal ontology in information systems.** Amsterdam, 1998. Disponível em: <http://www.loa.istc.cnr.it/Papers/FOIS98.pdf >. Acesso em: 18 ago. 2012.

HEINZLE, Roberto. **Um modelo de engenharia do conhecimento para sistemas de apoio a decisão com recursos para raciocínio abdutivo.** 2011. 251 f. Tese (Doutorado em Engenharia e Gestão do Conhecimento) - Programa de Pós-Graduação em Engenharia e Gestão do Conhecimento, Universidade Federal de Santa Catarina, Florianópolis.

HORRIDGE, Matthew; BECHHOFER, Sean. The OWL API: A Java API for OWL Ontologies. **Semantic Web Journal**, Dayton, v. 2, n. 1, p. 11-21, out. 2009.

KATZ, Susanne; EBERHARDT, Alexander. **XML metadata interchange.** Stuttgart, 2004. Disponível em: <http://www.kriha.de/dload/uni/generativecomputing/generation/xmi.pdf>. Acesso em: 22 set. 2012.

KHRONOS. **WebGL specification.** Beaverton, 2012. Disponível em: <https://www.khronos.org/registry/webgl/specs/1.0/>. Acesso em: 29 nov. 2012.

MELL, Peter; GRANCE, Timothy. **The NIST definition of cloud computing.** Recommendations of the National Institute of Standards and Technology. Gaithersburg: National Institute of Standards and Technology, 2011.

NEO TECHNOLOGY. **The world's leading graph database.** San Mateo, 2013. Disponível em: <http://www.neo4j.org/>. Acesso em: 20 abr. 2013.

NOY, Natalya F.; MCGUINNESS, Deborah L. **Ontology development 101:** a guide to creating your first ontology. Stanford, 2001. Disponível em: <http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf>. Acesso em: 09 set. 2012.

OLIVEIRA JUNIOR, Edson; BONINI, Rodrigo P. Desenvolvimento de sistemas para a nuvem. Como identificar as principais tecnologias de apoio ao ciclo de vida de software. **Revista Engenharia de Software Magazine**, São Paulo, v. 57, n. 1, p. 55-62, mar. 2013.

OMG. **MOF 2.0/XMI mapping specification.** Needham, 2005. Disponível em: <http://www.omg.org/spec/XMI/2.1/PDF/>. Acesso em: 23 set. 2012.

ORACLE CORPORATION. **Jersey –** RESTful web services in Java. Redwood City, 2013. Disponível em: <https://jersey.java.net/>. Acesso em: 28 abr. 2013.

PRESSMAN, Roger S. **Engenharia de software.** 6. ed., São Paulo: McGrawHill, 2006.

RAUTENBERG, Sandro et al. Uma metodologia para o desenvolvimento de ontologias. **Revista Ciências Exatas e Naturais**, Pitanga, v. 10, n. 2, p. 238-262, dez. 2008.

RAUTENBERG, Sandro et al. Ferramenta ontoKEM: uma contribuição a ciência da informação para o desenvolvimento de ontologias. **Perspectiva em Ciência da Informação,** Belo Horizonte, v. 15, n. 1, p. 239-258, abr. 2010.

SMITH, Barry; WELTY, Christopher. **Ontology:** towards a new synthesis. Buffalo, 2001. Disponível em: <http://www.cs.vassar.edu/~weltyc/papers/fois-intro.pdf>. Acesso em: 25 set. 2012.

RUSSELL, Stuart; NORVIG, Petter. **Inteligência artificial.** Tradução Vandenberg D. de Souza. Rio de Janeiro: Campus, 2004. 1021 p.

THE CODEHAUS. **About XStream.** Brisbane, 2013. Disponível em: <http://xstream.codehaus.org/>. Acesso em: 16 fev. 2013.

SCHNEIDER, Thomas. **Implementation in the OWL API.** Manchester, 2010. Disponível em: <http://owl.cs.manchester.ac.uk/modproj/#sec:implementation_owlapi>. Acesso em: 28 abr. 2013.

VICKERY, Brian C. Ontologies. **Journal of Information Science,** Londres, v. 23, n. 4, p. 227-286, jan. 1997.

W3C. **OWL web ontology language overview** - W3C recommendation 10 february 2004. Massachusetts, 2004. Disponível em: <http://www.w3.org/TR/owl-features>. Acesso em: 09 set. 2012.

APÊNDICE A – Tela responsável pela configuração do servidor Amazon EC2

Na Figura 16 apresenta-se a tela, responsável pela configuração do servidor Amazon EC2, onde foram embarcadas as ferramentas desenvolvidas neste trabalho.

Figura 16 – Tela responsável pela configuração do servidor

APÊNDICE B – Exemplo de código no formato JSON gerado a partir da ontologia

No Quadro 17, apresenta-se o exemplo `Empresa.owl` do código no formato JSON gerado pela ferramenta de definição, conceitualização e formalização de ontologias de domínio.

Quadro 17 – Código JSON exemplo da definição para ontologia de empresas

```
"Ontologia" : {
    "nomeOntologia" : "Empresa.owl",
    "classes" : [ {
      "Classe" : [ {
        "nomeClasse" : "Thing",
        "id" : 1,
        "subclasses" : [ {
          "nomeClasse" : "Pessoa",
          "id" : 2,
          "dados" : [ {
            "Dado" : {
              "codigo" : "nome",
              "extensao" : {
                "classe" : "Tipo",
                "tipo" : "STRING"
              },
              "tipo" : "FUNCIONAL",
            }
          } ],
          "subclasses" : [ {
            "nomeClasse" : "Empresa",
            "id" : 3,
            "dados" : [ {
              "Dado" : {
                "codigo" : "CNPJ",
                "extensao" : {
                  "classe" : "Tipo",
                  "tipo" : "STRING"
                },
                "tipo" : "FUNCIONAL",
              }
            } ]
          }, {
```

Quadro 17 – Código JSON exemplo da definição para ontologia de Empresas (continuação)

```
              "nomeClasse" : "Funcionario",
              "id" : 4,
              "dados" : [ {
                "Dado" : {
                  "codigo" : "CPF",
                  "extensao" : {
                    "classe" : "Tipo",
                    "tipo" : "STRING"
                  },
                  "tipo" : "FUNCIONAL",
                }
              } ]
            } ]
          } ]
        } ],
        "associacoes" : [ {
          "Associacao" : {
            "codigo" : "Possui",
            "extensao" : {
              "classe" : "Objeto",
              "classeId" : 4,
              "tipo" : "SIMETRICA"
            },
            "tipo" : "SIMETRICA",
            "dominiosId" : [ {
              "long" : 3,
            } ]
          }
        } ]
    }
```

APÊNDICE C – Código VTL responsável pela geração do código no formato OWL

No Quadro 18 é apresentado o código no formato VTL para a geração de código no formato OWL.

Quadro 18 – Código VTL responsável pela geração do código no formato OWL

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl = "http://www.w3.org/2002/07/owl#"
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema#"
>

#subclasses($ontologia.getClasses(), "")

#associacoes($ontologia.getAssociacoes())

#subclassesPropriedades($ontologia.getClasses())

</rdf:RDF>

#macro( subclassesPropriedades $subclasses)
#foreach( $classe in $subclasses )
#propriedades($classe.getDados(), $classe.getNomeClasse())
#subclassesPropriedades($classe.getSubclasses())
#end
#end

#macro( subclasses $subclasses)
#foreach( $classe in $subclasses )
<owl:Class rdf:ID="$classe.getNomeClasse()">
#if ($classeNome != "")
  <rdfs:subClassOf rdf:resource="#$classe.getClasseSuperNome()"/>
#end
 <rdfs:comment xml:lang="pt">$classe.getComentario()</rdfs:comment>
</owl:Class>
#subclasses($classe.getSubclasses())
#end
#end
```

Quadro 18 – Código VTL responsável pela geração do código no formato OWL (continuação)

```
#macro( propriedades $propriedades, $dominioNome)
#foreach( $propriedade in $propriedades )
<owl:ObjectProperty rdf:ID="$propriedade.getCodigo()">
<rdf:type rdf:resource="$propriedade.getTipo()" />
  <rdfs:comment xml:lang="pt">$propriedade.getComentario()</rdfs:comment>
  <rdfs:domain rdf:resource="#$dominioNome"/>
  <rdfs:range rdf:resource="#$propriedade.getExtensao()"/>
</owl:ObjectProperty>
#end
#end

#macro( associacoes $associacoes)
#foreach( $associacao in $associacoes )
<owl:ObjectProperty rdf:ID="$associacao.getCodigo()">
<rdf:type rdf:resource="$associacao.getTipo()" />
  <rdfs:comment xml:lang="pt">$associacao.getComentario()</rdfs:comment>
  <rdfs:domain  rdf:resource="#$associacao.getDominioNome()"/>
  <rdfs:range rdf:resource="#$associacao.getExtensao().getClasseNome()"/>
#end
#end
```

APÊNDICE D – Código OWL gerado pela ferramenta de definição das ontologias

No Quadro 19 é apresentado o exemplo `SistemaInformacao.owl` do código no formato OWL, gerado a partir da ferramenta de definição de ontologias.

Quadro 19 – Código OWL exemplo da definição para ontologia de sistema

```
<?xml version="1.0" encoding="iso-8859-1"?>
  <rdf:RDF
    xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl = "http://www.w3.org/2002/07/owl#"
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema#"
  >

  <owl:Class rdf:ID="SistemaInformacao">
   <rdfs:comment xml:lang="pt">[Não comentado]</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="Rotina">
   <rdfs:comment xml:lang="pt">[Não comentado]</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="Processo">
    <rdfs:subClassOf rdf:resource="#Rotina"/>
   <rdfs:comment xml:lang="pt">[Não comentado]</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="Cadastro">
    <rdfs:subClassOf rdf:resource="#Rotina"/>
   <rdfs:comment xml:lang="pt">[Não comentado]</rdfs:comment>
  </owl:Class>

  <owl:Class rdf:ID="Simples">
    <rdfs:subClassOf rdf:resource="#Cadastro"/>
   <rdfs:comment xml:lang="pt">[Não comentado]</rdfs:comment>
  </owl:Class>

  <owl:Class rdf:ID="Complexo">
    <rdfs:subClassOf rdf:resource="#Cadastro"/>
   <rdfs:comment xml:lang="pt">[Não comentado]</rdfs:comment>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="possui_telaFiltro">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#SymmetricProperty" />
```

Quadro 19 – Código OWL exemplo da definição para ontologia de sistema (continuação)

```
    <rdfs:comment xml:lang="pt">Cada processo possui uma tela simples
                               com campos de filtro.</rdfs:comment>
    <rdfs:domain rdf:resource="#Processo"/>
    <rdfs:range rdf:resource="#Simples"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="nome">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty" />
    <rdfs:comment xml:lang="pt">Dado nome pedido em tela.</rdfs:comment>
    <rdfs:domain rdf:resource="#Cadastro"/>
    <rdfs:range rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="idade">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty" />
    <rdfs:comment xml:lang="pt">Dado nome pedido em tela.</rdfs:comment>
    <rdfs:domain rdf:resource="#Cadastro"/>
    <rdfs:range rdf:datatype="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

APÊNDICE E – Código VTL responsável pela geração do código Java da camada modelo do padrão MVC

No Quadro 20 é apresentado o código no formato VTL para a geração de código Java da camada modelo do padrão MVC.

Quadro 20 – Código VTL responsável pela geração da camada modelo do padrão MVC

```
#macro( atributos $lista )
  #foreach( $atributo in $lista )
       @Column
    private $atributo.getTipoStr() $atributo.getNome();
  #end
#end

#macro( metodos $lista )
  #foreach( $atributo in $lista )
    public void set$atributo(String $atributo.getNome()) {
       this.$atributo.getNome() =  $atributo.getNome();
    }

    public $atributo.getTipo() get$atributo() {
       return this.$atributo.getNome();
    }
  #end
#end

package br.com.nemesis.gausten.arquivosgerados;

import javax.persistence.Column;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Entity;
import br.com.nemesis.gausten.modelo.Rotina;

@Entity
@Table(name="$entidade.getNome()")
public class $entidade.getNome() extends Rotina {
```

Quadro 20 – Código VTL responsável pela geração da camada modelo do padrão MVC (continuação)

```
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int id;

    #atributos($entidade.getAtributos())

    @Override
    public int getId() {
            return id;
        }

        public void setId(int id) {
            this.id = id;
        }

    #metodos($entidade.getAtributos())

    @Override
        public boolean equals(Object obj) {
            if(obj instanceof $entidade.getNome()){
                $entidade.getNome() entidade = ($entidade.getNome()) obj;
                return entidade.getId() == this.getId();
            }

            return false;
        }
    }
```

APÊNDICE F – Código VTL responsável pela geração do código Java da visão do padrão MVC

No Quadro 21 é apresentado o código no formato VTL para a geração de código JSF da camada visão do padrão MVC.

Quadro 21 – Código VTL responsável pela geração da camada visão do padrão MVC

```
#macro( atributos $lista )
  #foreach( $atributo in $lista )
    <p:column headerText="$atributo.getNome()"
                sortBy="#{rotina.$atributo.getNome()}"
                filterBy="#{rotina.$atributo.getNome()}"
                style="width:125px">
      <p:cellEditor>
        <f:facet name="output">
          <$atributo.getTipo() value="#{rotina.$atributo.getNome()}" />
        </f:facet>
        <f:facet name="input">
          ${tela.geraComponenteCampo($atributo)}
        </f:facet>
      </p:cellEditor>
    </p:column>
  #end
#end

<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:p="http://primefaces.org/ui">
<h:inputHidden id="inicio"
            value="#{${tela.getNomeEntidadeLower()}Bean.carregaTodas()}"/>
<f:view locale="pt_BR">
  <h:form>
    <p:messages />
    <p:dataTable id="rotinas" var="rotina"
                value="#{${tela.getNomeEntidadeLower()}Bean.rotinas}"
                emptyMessage="Sem registros cadastrados."
                editable="true">
    <f:facet name="header">
        ${tela.getNomeEntidade()}
```

Quadro 21 – Código VTL responsável pela geração da camada visão do padrão MVC (continuação)

```
            </f:facet>
          #atributos($entidade.getAtributos())
          <p:column headerText="" style="width:50px">
             <p:commandButton value="Gravar"
                     action="#{${tela.getNomeEntidadeBeanJSF()}.grava}"
                      icon="ui-icon-disk" update="@form"/>
             <p:commandLink update="@form"
                       actionListener="#{${tela.getNomeEntidadeLower()}Bean.
                       removeRegistro(rotina)}" style="text-decoration:none">
               <h:outputText title="Excluir" value="x" />
             </p:commandLink>
          </p:column>
           </p:dataTable>
        </h:form>

      </f:view>

      </html>
```

APÊNDICE G – Código VTL responsável pela geração do código Java da camada controle do padrão MVC

No Quadro 22 é apresentado o código no formato VTL para a geração de código Java da camada controle do padrão MVC.

Quadro 22 – Código VTL responsável pela geração da camada controle do padrão MVC

```
package br.com.nemesis.gausten.mb.gerados;

import java.util.List;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

import br.com.nemesis.gausten.arquivosgerados.$entidade.getNome();
import br.com.nemesis.gausten.modelo.Rotina;
import br.com.nemesis.gausten.mb.RotinaBean;

@ManagedBean
@SessionScoped
public class $entidade.getNome()Bean extends RotinaBean {
     @Override
     protected void defineRotina() {
        rotina = new $entidade.getNome()();
     }

     @Override
     protected void populateTable(List<Rotina> list) {
        list.addAll(DAO.findAll(${entidade.getNome()}.class));
     }

     @Override
     public void grava() {
        salvar(($entidade.getNome()) rotina);
     }
}
```

APÊNDICE H – Tela de consulta da ontologia gravada em BD orientado a grafos

Na Figura 17, apresenta-se a tela para a consulta à ontologia de Classificados de Emprego. A referida ontologia já está persistida na base de dados. Esta tela foi disponibilizada junto à composição de ferramentas do BD orientado a grafos: Neo4j.

Figura 17 – Tela do BD Neo4j para a consulta da ontologia de classificados de emprego