UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

UMA FERRAMENTA PARA CONVERSÃO DE MER DE UM BANCO DE DADOS RELACIONAL EM UMA ONTOLOGIA OWL

JOHANN NEHRING FRITZ

BLUMENAU
2013

2013/2-11

JOHANN NEHRING FRITZ

UMA FERRAMENTA PARA CONVERSÃO DE MER DE UM BANCO DE DADOS RELACIONAL EM UMA ONTOLOGIA OWL

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

2013/2-11

UMA FERRAMENTA PARA CONVERSÃO DE MER DE UM BANCO DE DADOS RELACIONAL EM UMA ONTOLOGIA OWL

Por

JOHANN NEHRING FRITZ

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. Cláudio Ratke, Mestre – FURB
Membro:	Prof. Alexander Roberto Valdameri, Mestre – FURB

Blumenau, 27 de novembro de 2013

Dedico este trabalho a minha namorada, que por mais distante fisicamente, sempre esteve ao meu lado quando para me apoiar durante o desenvolvimento deste trabalho.

AGRADECIMENTOS

A todos os professores, pelos longos anos de ensino.

Aos colegas que ficaram pelo caminho, mas nunca deixaram de apoiar.

Aos amigos, pela dura caminhada até aqui.

Ao meu orientador, Roberto Heinzle, pelo apoio desde a ideia até a apresentação deste.

A minha família por sempre estar presente.

A minha namorada pela compreensão e carinho demonstrados durante essa fase difícil.

You miss 100% of the shots you don't take.

Wayne Gretzky

RESUMO

O presente trabalho desenvolveu um conversor de um MER de um Banco de Dados em uma ontologia no formato OWL. Para esse fim foi utilizado o software DBDesigner para gerar o MER inicial, e o software Protégé-OWL para leitura do OWL final, bem como a linguagem XSL e a API Xalan. O conversor desenvolvido possibilita a conversão com sucesso de qualquer MER de entrada, e gera uma ontologia no formato OWL, capaz de ser executada em qualquer software do gênero.

Palavras-chave: Linguagem OWL. Banco de dados. Ontologia.

ABSTRACT

The present work developed a converter to convert a database ERM to an OWL ontology. To archive this, the software DBDesigner was used to generate the ERM, the software Protégé-OWL to read the generated OWL, and the language XSL with the Xalan API. The developed converter offer the possibility of convert, with success, any ERM and generates an OWL ontology, capable of being executed in any software of that specie.

Key-words: OWL language. Database. Ontology.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

API – *Application Programming Interface*

BD – Banco de Dados

DOM – *Document Object Model*

JAXP – *Java API for XML Processing*

MER – Modelo Entidade-Relacionamento

ODBC – *Open DataBase Connectivity*

OWL – *Web Ontology Language*

RDF – *Resource Description Framework*

SAX – *Sample API for XML*

SG – Sistemas Gerenciadores

SGBD – Sistemas de Gerenciamento de Bancos de Dados

SPARQL – *SPARQL Protocol And RDF Query Language*

SQL – *Structured Query Language*

XML – *eXtensible Markup Language*

XPath – *XML Path Language*

XSL – *eXtensible Stylesheet Language*

XSLT – *XSL Transformation*

XSL-FO – *XSL Formatting Objects*

SUMÁRIO

1 INTRODUÇÃO

Com a evolução tecnológica dos últimos anos, muitos dos paradigmas acabaram sendo revistos, reestruturados ou substituídos. O que antes representava um avanço tecnológico, com o passar do tempo vem a ser substituído por algo novo. Um exemplo, na área de computação, foi a evolução dos arquivos indexados pelos Sistemas de Gerenciamento de Bancos de Dados (SGBD).

O aumento da capacidade de processamento e de armazenamento de dados proporcionado pelos computadores atuais tornaram os SGBD a opção preferencial para substituir seu antecessor na tarefa de armazenar os dados dos sistemas informatizados. Esta mudança trouxe consigo ainda outras importantes tecnologias relacionadas, tais como a *Structured Query Language* (SQL) e o Modelo Entidade-Relacionamento (MER), entre outras (FLORENTINO, 2009).

O MER é um modelo baseado na percepção do mundo real, que consiste em um conjunto de objetos básicos chamados entidades e nos relacionamentos entre eles. Ele possui informações de todos os atributos e relações do mesmo, independente dos dados. É comumente representado por um composto de quadros informativos chamados de entidades e seus relacionamentos (MELLO, 2005). Ao se extrair um MER de um Banco de Dados (BD), o formato pode variar, sendo um deles o *eXtensible Markup Language* (XML).

Posteriormente, já no contexto da Internet e também dos esforços para o desenvolvimento dos sistemas baseados em conhecimento, surge outra tecnologia que vem recebendo atenção destacada dos pesquisadores, as ontologias (HEINZLE, 2011, p. 29). As ontologias são vocabulários compartilhados que podem ser usadas para modelar um domínio – um tipo de objetos e/ou conceitos existentes – e suas relações e propriedades (MALUSZYNSKI, 2002). Elas tiveram origem na filosofia, onde foram criadas para tentar responder perguntas como “O que é um ser?” ou “Quais são as características comuns de todos os seres?”. Uma ontologia trata do ser enquanto ser, e tem como objetivo compreender identidades individuais ou agrupadas, podendo ser vista como uma descrição sistemática da existência das mesmas (HEINZLE, 2011, p. 106).

Já na área de computação, este termo é usado para representar ou descrever conceitos ou áreas do conhecimento (GUIMARÃES, 2002). O conceito de ontologia veio para a área de computação por volta da década de noventa do século passado, onde ganhou uma nova interpretação. Esta interpretação prevê a montagem de uma especificação formal ou de um conjunto de regras sobre um determinado assunto (HEINZLE, 2011, p. 106).

Ao ser adicionada na área de computação, a ontologia pode ser um artefato de software, para ser criada, manipulada ou utilizada pelo mesmo. Para este fim, foram criadas linguagens para representa-las, sendo uma delas a *Web Ontology Language* (OWL) (W3C, 2013).

Pode-se afirmar que existe certa convergência entre as ontologias e os MERs, sobretudo no que diz respeito às suas estruturas, já que ambos representam algo real, do qual se tem interesse em armazenar características como atributos e relações. Um MER é uma representação de um BD qualquer, cujo objetivo final é o armazenamento de informações. Uma ontologia, entre muitas coisas, serve para criar relações entre entidades, e atribuir valores a estas entidades. Com isso em mente, é possível que sejam feitas análises sobre as estruturas dos arquivos XML e OWL, e que seja traçado um padrão entre atributos de ambos os modelos.

A estrutura interna de um arquivo OWL é baseada em XML, fazendo com que possa ser comparado a um MER que também esteja em algum formato baseado em XML. Existem *frameworks* e softwares que permitem criar ontologias no formato OWL, porém nenhum deles suporta a conversão MER – OWL, o que torna o trabalho apresentado um complemento. Converter um MER em um OWL é o primeiro passo para migrar um BD para uma ontologia, explorando-se assim as vantagens das mesmas.

Uma forma de fazer esta conversão é através de uma linguagem chamada *eXtensible Stylesheet Language* (XSL). Esta linguagem serve para, através de *parsing*, fazer o *matching* dos elementos XML, e fazer as transformações desejadas com o mesmo. Através deste processo, é possível converter um XML para outro formato desejado.

É neste contexto, que surge este trabalho, o qual visa o desenvolvimento de uma ferramenta que ofereça as funcionalidades necessárias para a conversão destas estruturas. Notadamente de um MER representado em XML para uma Ontologia-OWL.

1.1 OBJETIVOS DO TRABALHO

O trabalho consiste no desenvolvimento de uma ferramenta para conversão de um diagrama MER de um SGBD relacional representado no formato XML para uma ontologia no padrão OWL.

Os objetivos específicos deste trabalho são:

a) identificar os elementos e as construções oferecidas pelas linguagens XML e OWL;
b) estabelecer as regras de equivalência de conversão dos elementos XML para OWL;
c) especificar e implementar as regras de conversão.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está dividido em três partes: a fundamentação teórica apresenta as ferramentas e tecnologias utilizadas durante a realização deste, bem como trabalhos correlacionados com o assunto; o desenvolvimento apresenta trechos do software desenvolvido bem como explicações e resultados obtidos após o desenvolvimento e por último as conclusões obtidas ao fim do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está organizado em quatro áreas: a primeira trata de banco de dados relacionais, MER e a ferramenta DBDesigner, a segunda aborda ontologias, o formato OWL e o software Protégé-OWL, o terceiro fala da linguagem XSL e da *Application Programming Interface* (API) Xalan e o último deles mostra trabalhos relacionados à ferramenta desenvolvida.

2.1 BANCO DE DADOS RELACIONAL

Um sistema de banco de dados é um sistema para armazenar registos; é um sistema para permitir que usuários mantenham informações, e depois as acessem e as modifiquem, baseando-se em regras de seleção. Esta informação pode ser de qualquer natureza que seja importante para um indivíduo ou organização; tudo o que seja necessário para o processo geral de administração. Essa informação fica armazenada em tabelas, e estas por sua vez possuem linhas e colunas, onde cada linha representa um novo registro e cada coluna uma característica que se deseja guardar sobre tal (DATE, 2004, p. 5).

É necessário ressaltar que as tabelas contidas em um BD não são simplesmente arquivos convencionais. A diferença é que nos métodos de acesso a arquivos convencionais a recuperação das informações se faz registro a registro enquanto no ambiente relacional faz-se uso dos fundamentos da teoria das relações na construção e operação do SGBD. Dentre estes fundamentos, destaca-se um conjunto de operadores que compõem a álgebra relacional (KEMCZINSKI, 2008).

Um SGBD é um conjunto de programas que incorpora as funções definição, recuperação e alteração de dados em um BD, além de gerenciar e garantir a integridade das informações nele armazenadas. Esta modularização tem inúmeras vantagens, tais como facilitação da manutenção, pois as funções estão separadas e o aumento da produtividade dos desenvolvedores, uma vez os programas ficam menores já que existem menos coisas para se preocuparem (HEUSER, 2008, p. 4).

2.1.1 MERs

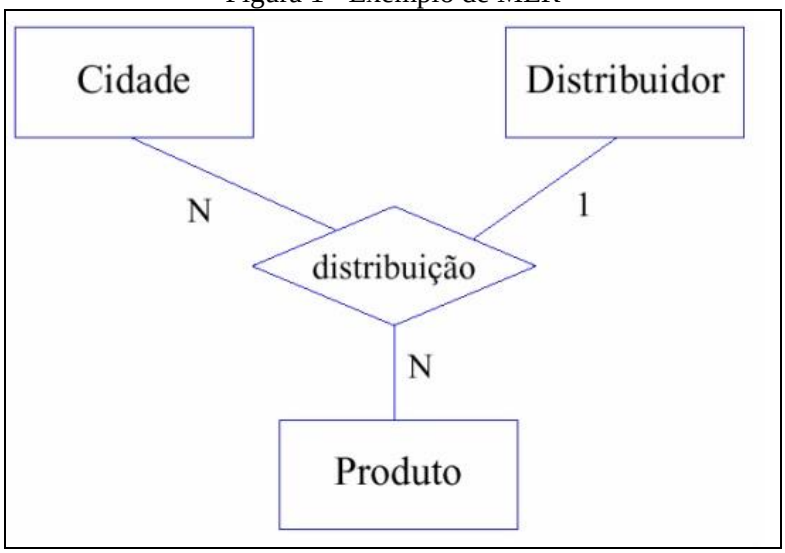
É a representação abstrata e simplificada de um sistema real com o qual se pode explicar o mesmo. É um modelo baseado na percepção do mundo real, que consiste em um conjunto de objetos básicos chamados entidades e nos relacionamentos entre estes objetos (MELLO, 2005).

O MER foi criado em 1976 por Peter Chen para servir como padrão de modelagem conceitual de um BD. Esta modelagem é uma descrição abstrata, que independe da implementação em um computador e dos dados que serão armazenados. Os modelos e técnicas de modelagem mais recentes tem sido baseado nos conceitos e abordagem do MER (HEUSER, 2008, p. 11).

Neste modelo, as entidades são um conjunto de objetos da realidade modelada sobre os quais se deseja manter informações no BD, enquanto os relacionamentos são conjuntos de associações entre estas entidades modeladas (HEUSER, 2008, p. 12, 13). Nesta modelagem é importante tambem a cardinalidade das relações. Esta indica o número minimo e máximo de ocorrencias de entidades associadas por um relacionamento, podendo ser elas 1:1 (um para um), 1:n (um para muitos) ou n:n (muitos para muitos) (HEUSER, 2008, p. 16).

O MER pode ser extraído de um SGBD através de ferramentas de reengenharia, podendo ser salvo em diferentes formados, entre eles o XML, que pode ser interpretado em outros softwares que farão uso deste arquivo. Um exemplo de software usado para extrair um MER é o DB Designer. Um exemplo de um MER pode ser visto na Figura 1. Nela se pode verificar a forma como se relacionam as diversas entidades propostas. É visto que um produto pode possuir mais de uma distribuição, assim como apenas um distribuidor poderá distribuir vários produtos para várias cidades.

Figura 1 - Exemplo de MER



Fonte: Heuser (2008, p. 19).

2.1.2 DBDesigner

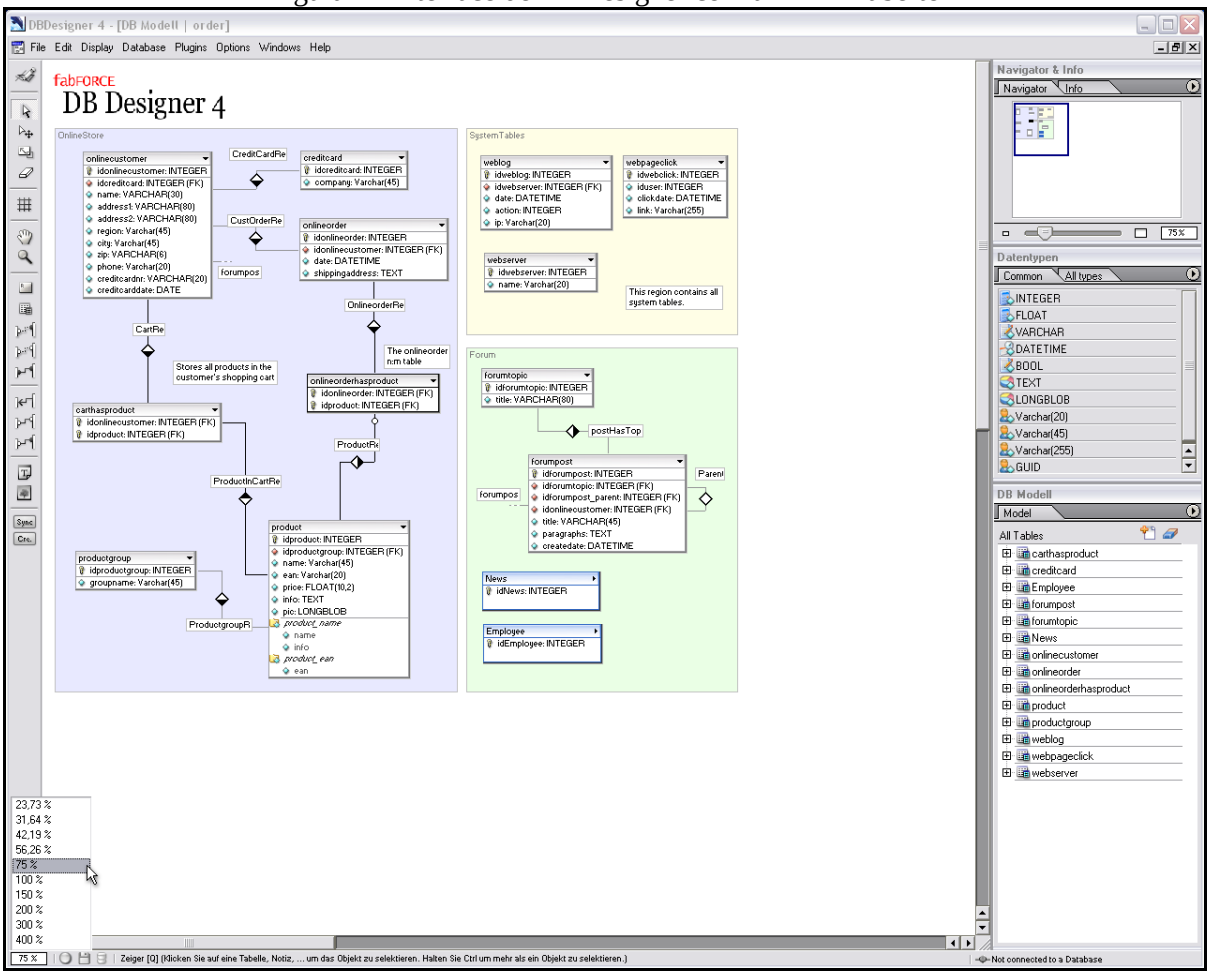
O DBDesigner é um software que envolve todas as etapas de modelagem de um BD. Desde o seu desenvolvimento inicial, até manutenções, tudo de uma forma que facilite o entendimento de quem o está desenvolvendo. Ele combina ferramentas que antes eram distribuídas separadamente, de modo a facilitar para todos os que queiram desenvolver um BD. Além disso, ele conta com uma função de *reverse engineering* permitindo que através de uma coneção com um BD existente, o software faça a extração do MER do mesmo (FABFORCE, 2003).

O DBDesigner é utilizado em softwares de pequena e média escala devido a sua facilidade, mas é uma ferramenta que possui facilidades existentes em outros grandes nomes na area de desenvolvimento de SGBDs. O software foi desenvolvido e otimizado para servir principalmente o BD MySQL por ser *open source* assim como o proprio DBDesigner. Apesar disso, a versão 4 do software dá suporte para BDs com acesso por *Open DataBase Connectivity* (ODBC) (FABFORCE, 2003).

Outra funcionalidade oferecida do software é o fato de permitir salvar um MER no modelo XML. Isso faz com que o MER gerado possa ser lido por algum outro software de modelagem ou qualquer linguagem com suporte a XML para usos gerais (FABFORCE, 2003). Na Figura 2 pode-se ver a interface de modelagem de um MER. Nela é possivel identificar alguns diferentes módulos presentes. Além disso, pode ser visto como cada tabela

se relaciona com as demais. É possivel ainda identificar o tipo da relação (1 para 1, 1 para n) entre as mesmas.

Figura 2 - Interface do DBDesigner com um MER aberto



Fonte: Fabforce (2003).

2.2 ONTOLOGIAS

Cada vez mais se tem buscado alternativas sobre a organização de informações e suas técnicas, estando às ontologias entre estas (BAX; ALMEIDA, 2003). Levando em conta a necessidade de uma boa definição estrutural de uma ontologia, a mesma geralmente é feita por um profissional com conhecimento aprofundado na área para qual a mesma será aplicada (comercial, industrial, etc.).

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. Adicionalmente, é dito que uma ontologia pode também referir-se a um conjunto de conhecimentos que descreve algum domínio usando um vocabulário representativo (HEINZLE, 2011, p. 107).

Depois de criada, a ontologia passa a servir para diversos propósitos, dentre eles, como um molde para armazenamento de dados. Na tecnologia atual, existem diversas linguagens para a representação de uma ontologia, sendo uma delas a OWL.

2.2.1 OWL

Uma ontologia no formato OWL é composta por uma estrutura baseada em um XML onde, com um objeto (modelo) de estudo, são dados os seus atributos e referências, montando uma representação estrutural do mesmo (W3C, 2013). As ontologias OWL são uma variação derivada de outro formato conhecido como *Resource Description Framework* (RDF).

O RDF é um *framework* reconhecido pela W3C e utilizado desde 1999 com o objetivo de criar um conjunto de informações básicas e resumidas para serem usadas em endereços Web (W3C, 2004). Com uma estrutura criada é possível ainda aplicar outras tecnologias sobre ela, como operações similares as de um banco de dados, como por exemplo, consultas ou alterações utilizando o *SPARQL Protocol And RDF Query Language* (SPARQL) (HERMAN, 2008).

As ontologias no formato OWL são regulamentadas pela W3C, responsável por gerenciar cada nova versão da mesma, ministrar cursos e técnicas, disponibilizar tutoriais e

suporte aos interessados no assunto (W3C, 2013). Elas empregam alguns componentes básicos na formalização do conhecimento, sendo eles: classes, relacionamentos, axiomas e instâncias.

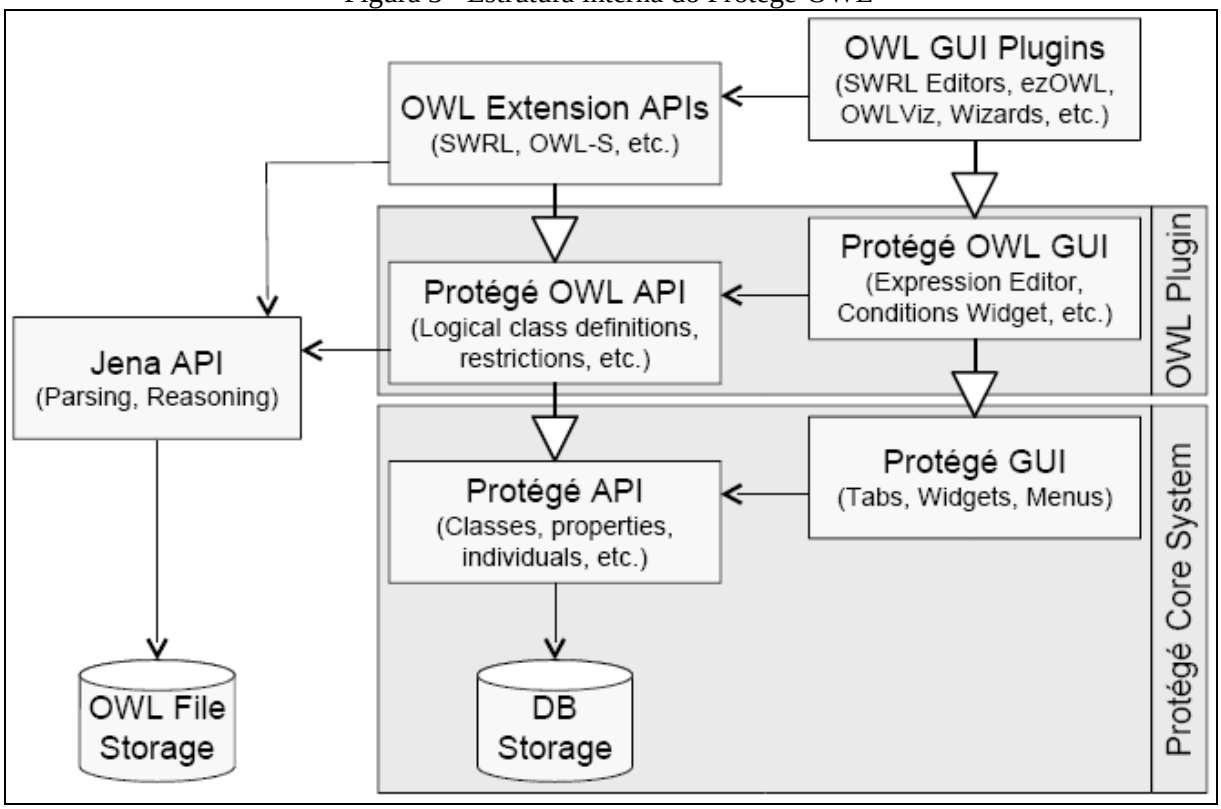
As classes são as unidades básicas de toda ontologia. Elas representam coleções de elementos que possuem atributos iguais e formam conceitos que definem um determinado objeto. Os conceitos representam todas as coisas relacionadas ao domínio que se pretende modelar, incluindo objetos, tarefas, ações etc. As ligações entre estes conceitos se dão através dos relacionamentos ou relações. As relações descrevem as interações entre os conceitos, as quais representam os relacionamentos semânticos envolvidos no domínio. Os axiomas são regras relativas às relações que devem obrigatoriamente ser cumpridas pelos elementos de uma ontologia, que são restrições. Já as instâncias representam os elementos ou objetos da ontologia, que são os exemplares individuais das classes (HEINZLE, 2011, p. 107).

2.2.2 Protégé-OWL

O Protégé-OWL é um é uma extensão do software Protégé, com suporte a ontologias OWL. Este software permite que sejam manipuladas ontologias no formato OWL e RDF. Pode-se visualizar e editar suas propriedades, definir expressões lógicas para o ambiente OWL, aplicar *reasoners* para a sua completa definição logica e editar todos os indivíduos do formato OWL para deixá-los no formato dos *markups* da Web Semântica (PROTÉGÉ, 2013).

O Protégé-OWL está ligado ao software Jena que será visto na seção de trabalhos relacionados. Essa ligação faz com que chamadas do Jena possam ser feitas em tempo de execução, sem ser necessaria uma longa recompilação e reexecução a cada mudança na ontologia (PROTÉGÉ, 2013). Na Figura 3 pode ser visto o *overview* do fucionamento da ferramenta. Nela é possivel identificar os modulos que fazem parte do software Protégé (na area *Protégé Core System*) e as extensões especificar para o funcionamento das ontologias OWL (area *OWL Plugin*). Com isso, é identificado o percurso feito pela informação quando uma ontologia é criada. A mesma passa pelo editor de expressões, pela definição de classes lógicas e suas restrições da qual pode ou partir para um reasoner para ser trabalhada, ou ter suas definições ampliadas.

Figura 3 - Estrutura interna do Protégé-OWL



Fonte: Protégé (2013).

2.3 XSL

Pertencente a família da *XSL Transformation* (XSLT), o XSL é uma linguagem de transformação reconhecida e mantida pela W3C. Ela serve, juntamente com os dois outros membros da familia, o *XML Path Language* (XPath) e o *XSL Formatting Objects* (XSL-FO), para converter documentos XML em outros formatos (W3C, 2013).

O XSL é utilizado para descrever uma folha de estilos – um padrão que deverá ser seguido – e o XSLT irá utilizar essa folha de estilos para realizar a transformação. A premissa básica para que o XSLT funcione é que a linguagem de origem da transformação seja baseada em XML, ou seja, contenha elementos e atributos. A partir disso, com uma folha de estilos, é possível transformar a linguagem de origem em qualquer outra linguagem que for desejada, desde SQL, Java, outra linguagem baseada em XML, ou até mesmo texto (W3C, 2013). No Quadro 1 pode ser visto um exemplo de codigo XSL.

O trecho de código demonstrado no Quando 1 é um exemplo que irá gerar como resultado um bloco através do codigo `fo:block` com textos em negrito devido ao atributo

`font-weight="bold"`. Após isso ele irá – com a chamada `<xsl:apply-templates/>` - continuar para o próximo *template*. Este segundo irá selecionar todos os `speech` e executar com a condição que o atritubo `speaker` seja `Arthur`. Isso pode ser visto no trecho `match="speech[@speaker='Arthur']"`. Em seguida ele irá criar um bloco com o fundo azul com o codigo `<fo:block background-color="blue">` e nele irá colocar o valor do atributo `speaker`.

Quadro 1 - Exemplo de XSL

```
<xsl:template match="FX">
   <fo:block font-weight="bold">
       <xsl:apply-templates/>
   </fo:block>
</xsl:template>

<xsl:template match="speech[@speaker='Arthur']">
   <fo:block background-color="blue">
       <xsl:value-of select="@speaker"/>
       <xsl:apply-templates/>
   </fo:block>
</xsl:template>
```

Fonte: W3C (2013).

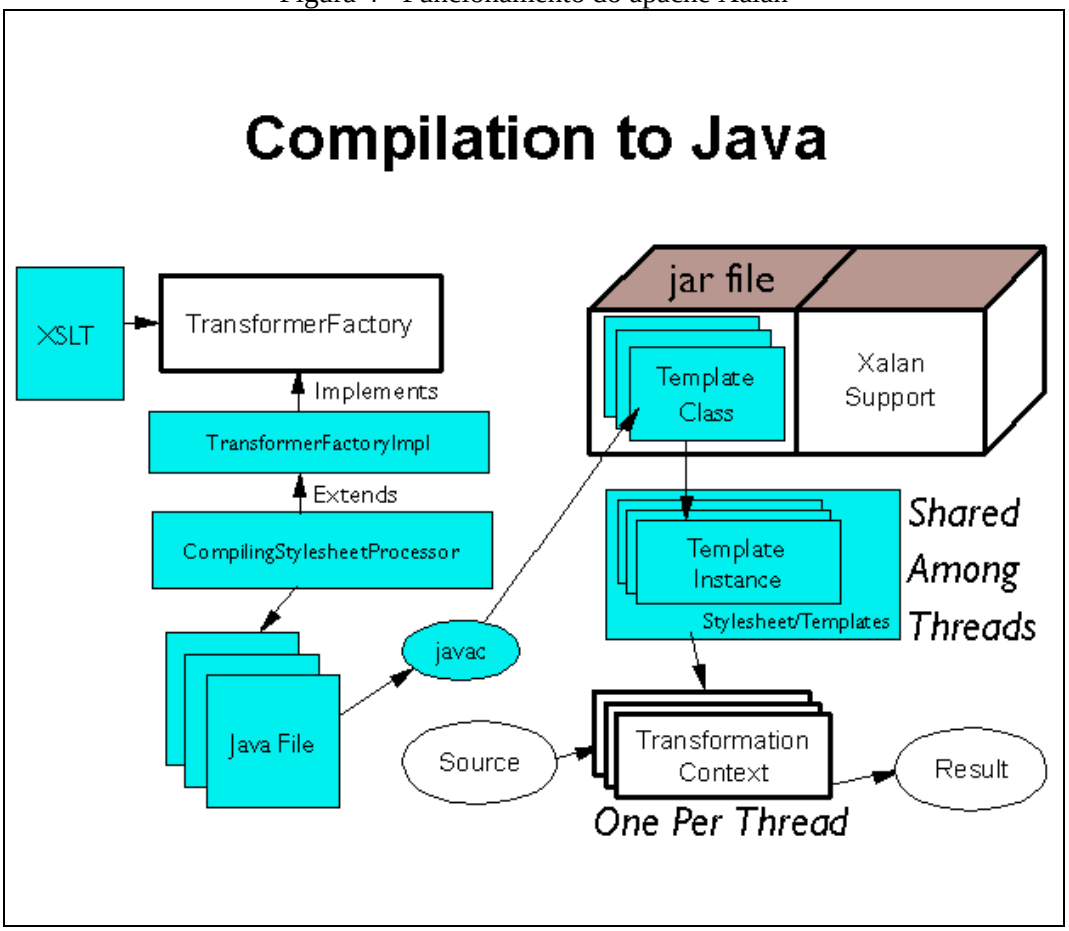
2.3.1 Xalan

O Apache Xalan foi criado com o instinto de promover o uso da linguagem XSLT e seus componentes. Atualmente está dividido em 2 projetos, um de C++ e outro de Java. O Xalan é um conjunto de bibliotecas cujo principal objetivo é o de executar uma transformação XSLT junto com as demais linguagens suportadas tais como o XML e o XPath (APACHE, 2012).

O projeto Xalan Java serve para a a conversão de documentos XML em documentos HTML, textos ou outros formatos que derivem do XML. Ele pode gerar um arquívo físico, bem como um *Sample API for XML* (SAX) ou um *Document Object Model* (DOM) de nivel 3. Alem disso, ele implementa tambem uma interface de processamento de XML chamada de *Java API for XML Processing* (JAXP) (APACHE, 2012). A estrutura do Xalan pode ser visto na Figura 4. Nela é visto como o Xalan faz a transformação utilizando a JVM. Quando é chamado o método `Transform` da API, o XSLT é lido pela classe `CompilingStylesheetProcessor` que é responsavel pela execução do mesmo sobre o

arquivo de entrada. Após isso, em caso de sucesso, o resultado é enviado para o contexto de transformação, onde é então preparado e o resultado fica disponível para que possa ser acessado ou gravado.

Figura 4 - Funcionamento do apache Xalan



Fonte: Xalan (2012).

2.4 TRABALHOS CORRELATOS

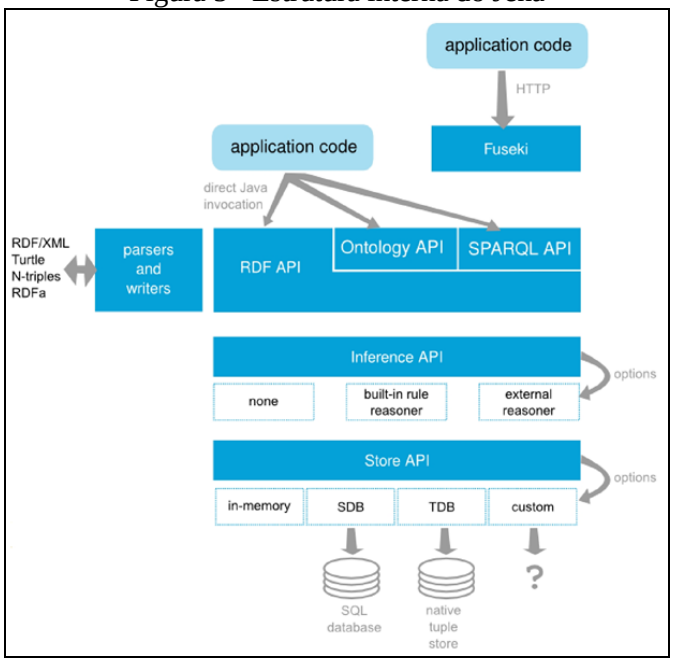
A seguir serão apresentados três (3) trabalhos correlatos. Todos apresentam alguma semelhança com o trabalho apresentado. As ferramentas apresentadas consistem no Jena (APACHE, 2013), Sesame (OPENRDF, 2012) e 4store (GARLIK, 2009).

2.4.1 Jena

O Jena é um *framework* Java para a construção de aplicações de web semântica. Ele oferece uma extensa biblioteca Java para auxiliar os desenvolvedores em conjunto com publicações recomendadas pela W3C (APACHE, 2013).

O Jena inclui um motor de inferência (ou *reasoner*) baseado em regras para percorrer ontologias baseadas em OWL e RDFs, e uma variedade de soluções de armazenamento de triplas no formato RDF em memória ou em disco. Internamente, o Jena possui diversas funcionalidades adicionais. A Figura 5 resume a forma de trabalho do Jena.

Figura 5 - Estrutura interna do Jena



Fonte: Apache (2012).

Como se pode observar na Figura 5, o Jena permite trabalhar de 2 formas:

a) iniciando uma ontologia nova, escrita pelo desenvolvedor;

b) importando um arquivo para utilizar como base para a ontologia.

Em ambos os casos, o funcionamento é o mesmo. É possível interagir com a ontologia carregada com o motor de inferência, ou através de instruções de SPARQL. Depois de manipulada a informação desejada, pode-se armazenar a ontologia através de diversos dispositivos ou formas, seja diretamente em disco, um disco removível, na memória, dentro de uma base de dados, ou qualquer forma pretendida pelo desenvolvedor. Para o armazenamento, o Jena gera arquivos no padrão OWL para serem reimportados e reutilizado nas mais diversas formas e aplicações que uma ontologia pode ter (APACHE, 2013).

No software desenvolvido foi utilizada a sequência de processamento do Jena das ontologias no caso de uma manipulação. Em um software que interprete ontologias, quando uma é importada, o software faz uma leitura do arquivo completo e só após esta começa a importar os elementos. Dessa maneira a geração dos mesmos não precisa ser sequencial.

2.4.2 Sesame

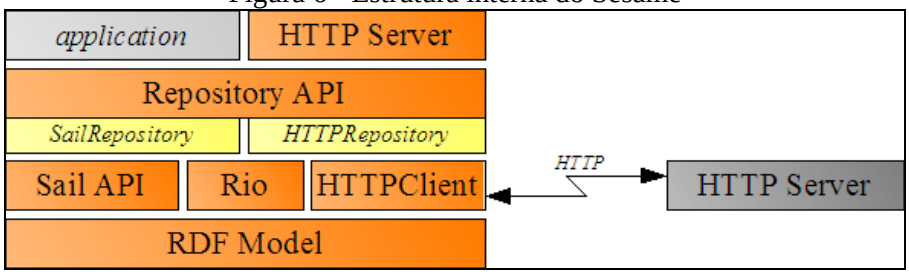
O Sesame é um *framework* para processamento de arquivos RDF que conta com funcionalidades desde a entrada (*parsing*) até o armazenamento (*storage*), passando por motores de inferência e consultas aos dados armazenados. Ele oferece uma API que pode ser conectada a todos os principais dispositivos de armazenamento (OPENRDF, 2012).

O *framework* foi desenvolvido com o objetivo de ser flexível. Pode ser gravado em vários dispositivos, desde arquivos, BDs ou até na própria memória. Além disso, oferece uma variedade de ferramentas para o desenvolvedor trabalhar com o RDF por inteiro.

Outra vantagem do Sesame é a interação com o SPARQL como linguagem de interação, inclusive para servidores remotos, utilizando a mesma API do acesso local. A Figura 6 mostra a estrutura interna e o funcionamento do Sesame. Nela se pode ver como funciona a comunicação de uma aplicação local e uma aplicação remota atraves das camadas do Sesame. Partindo de uma solicitação inicial, o pedido passa pela API do repositório principal para normatização, seja local ou web. Após isso, a API encaminha o pedido para a próxima camada de coneção. Essa por sua vez aplica o SPAQRL sobre o modelo RDF e devolve o resultado.

No conversor desenvolvido, o Sesame serviu de auxilio na área de *parsing* do XML, uma vez que o XSLT necessitava de algumas regras para poder ser preciso sem perder eficiência.

Figura 6 - Estrutura interna do Sesame



Fonte: OpenRDF (2012).

2.4.3 4store

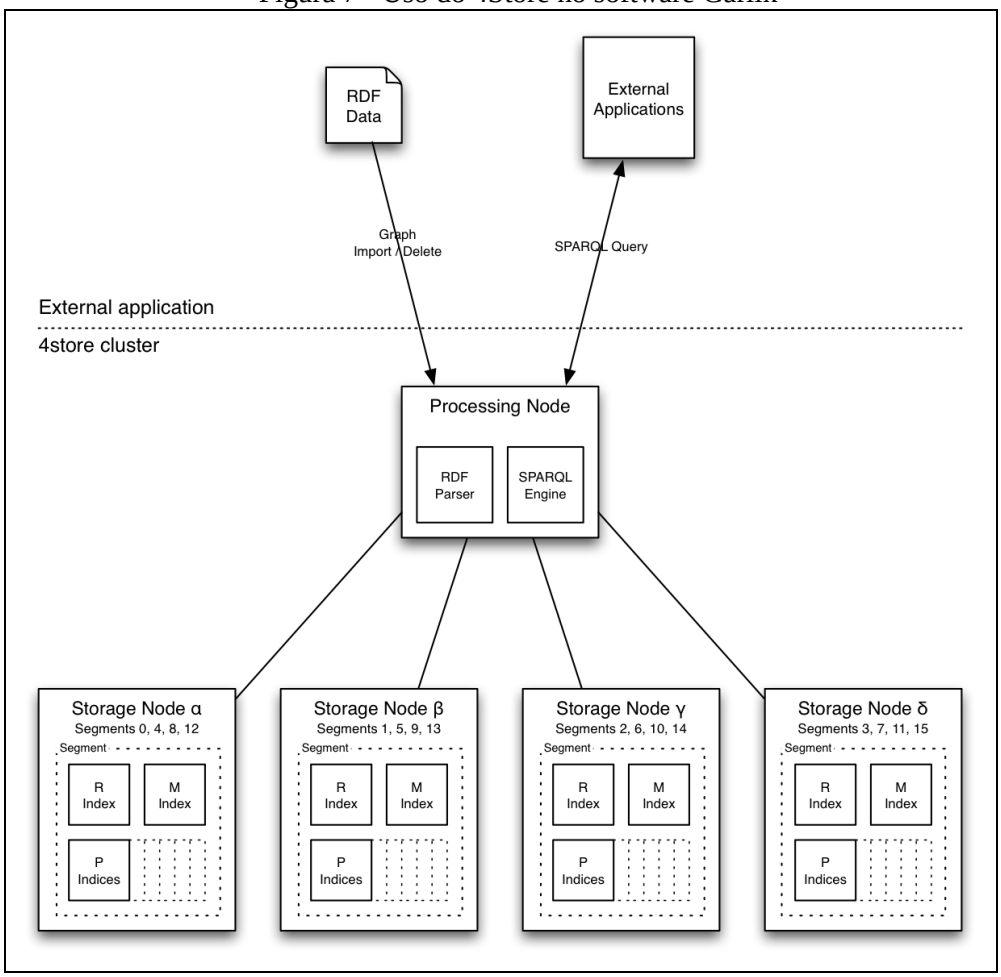
O 4store é uma ferramenta de armazenamento e consulta de arquivos RDF. Foi usado em larga escala no software Garlik para proteção familiar de conteúdos *online* (GARLIK, 2009).

Seus principais pontos fortes são o desempenho, a escalabilidade e a estabilidade. Além de armazenamento e consultas, o 4store não proporciona muitas outras funcionalidades. Dentro das fornecidas, a performance fornecida pelo 4store pode facilmente chegar a 120 kT/s (*kiloteras* por segundo) de importação quando em *cluster*. Além disso, o 4store oferece um módulo de segurança, área ainda pouco explorada por outras ferramentas semelhantes, fazendo com que o 4store seja um *storage* de RDF seguros (GARLIK, 2009). O funcionamento do 4store pode ser visto na Figura 7.

Quando uma aplicação faz uma solicitação no formato de SPARQL para a API do 4store, a mesma é responsavel por enviar esta para os *clusters* de dados, e quando a resposta for recebida, ela faz o *parsing* do resultado, montando um arquivo RDF e o devolvendo para a aplicação. Além disso, a API ainda é responsável pela montagem de instruções de armazenamento de novos RDFs no cluster de dados.

No conversor implementado, o 4store serviu, junto com o Sesame, na parte de *parsing* auxiliando na criação das regras de otimização do mesmo.

Figura 7 - Uso do 4Store no software Garlik



Fonte: Garlik (2011).

3 DESENVOLVIMENTO

Essa seção é dividida em 4 partes: a primeira mostra os requisitos do software, a segunda a especificação, a terceira a implementação contemplando as técnicas, as ferramentas, os métodos e ao final do mesmo o uso da ferramente proposta, e a última apresenta os resultados e discussões.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O software proposto deverá ser capaz de converter um MER (no formato XML) em uma ontologia no formato OWL. Para tal deverá ser capaz de receber como entrada um aquivo XML previamente extraído pelo software DBDesigner versão 4. A partir desse arquivo, ele irá aplicar uma transformação XSLT utilizando a linguagem XSL. Após a transformação, o software deverá exibir um arquivo no formato OWL para ser salvo e, assim, possível de ser importado na ferramenta Protégé-OWL.

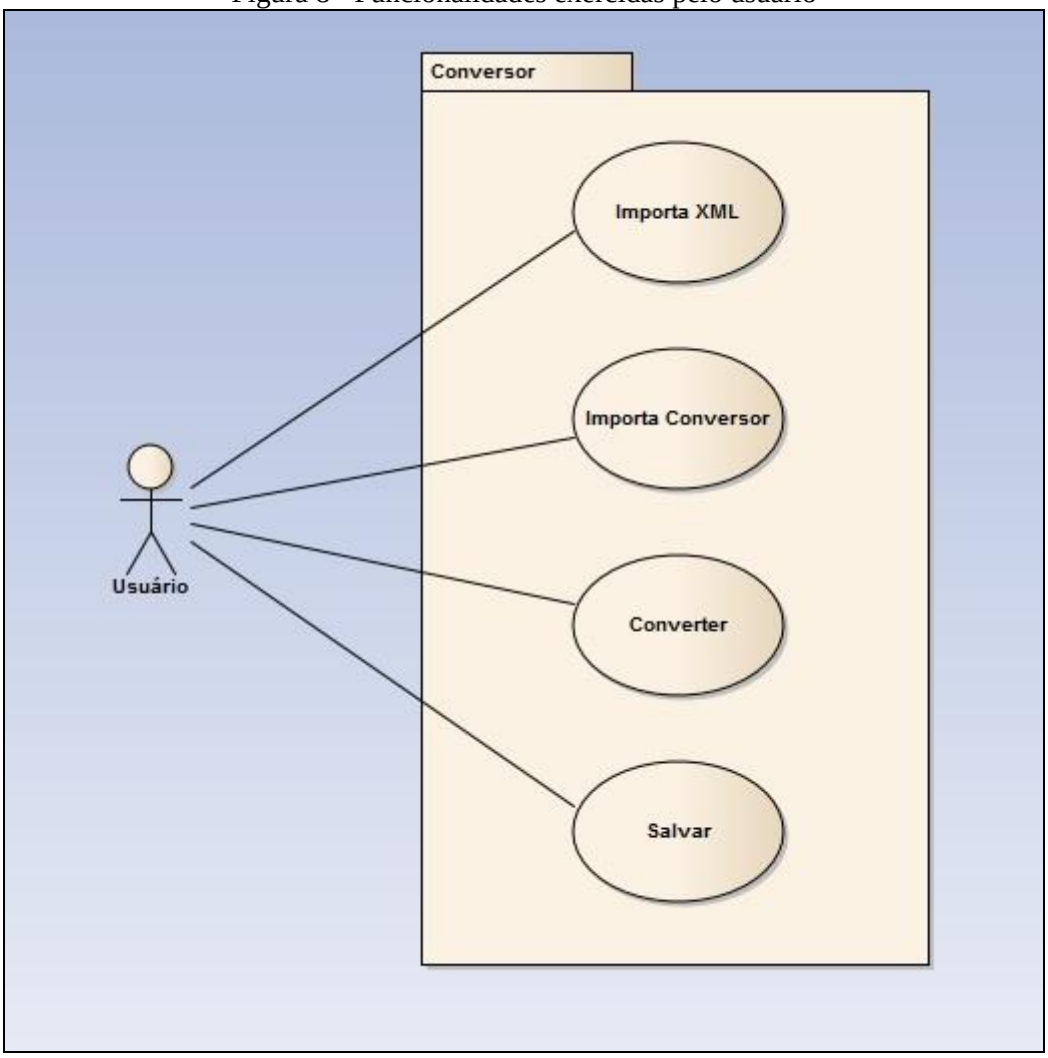
3.2 ESPECIFICAÇÃO

Esta seção apresenta duas figuras para diferentes modelos. Na Figura 8, são apresentadas as funcionalidades que o usuário final poderá executar no software desenvolvido, enquanto na Figura 9 é apresentado um modelo mostrado a interação das diferentes linguagens dentro do conversor.

A Figura 8 mostra o diagrama de casos de usos e as funcões que o usuário poderá exercer 4 atividades durante o funcionamento do software. O primeiro caso de uso é importar um arquivo XML. Este deverá ter sido previamente extraído de um SGBD com o software DBDesigner. No segundo caso de uso o usuário poderá carregar o conversor desejado, no formato XSL. No terceiro, acionar o botão de conversão do arquivo. Isso executará o XSL desenvolvido e converterá o XML de entrada em um arquivo OWL. Esse arquivo ficará salvo em uma pasta temporária caso o software seja fechado de forma inexperada. No último caso de uso o usuário poderá salvar o arquivo para uma pasta que lhe convém. Por se tratar de um

conversor, o software não contará com analizador de sintaxe XML ou OWL, portanto, para prevenir qualquer erro, em momento algum será possível editar os arquivos por dentro do software.

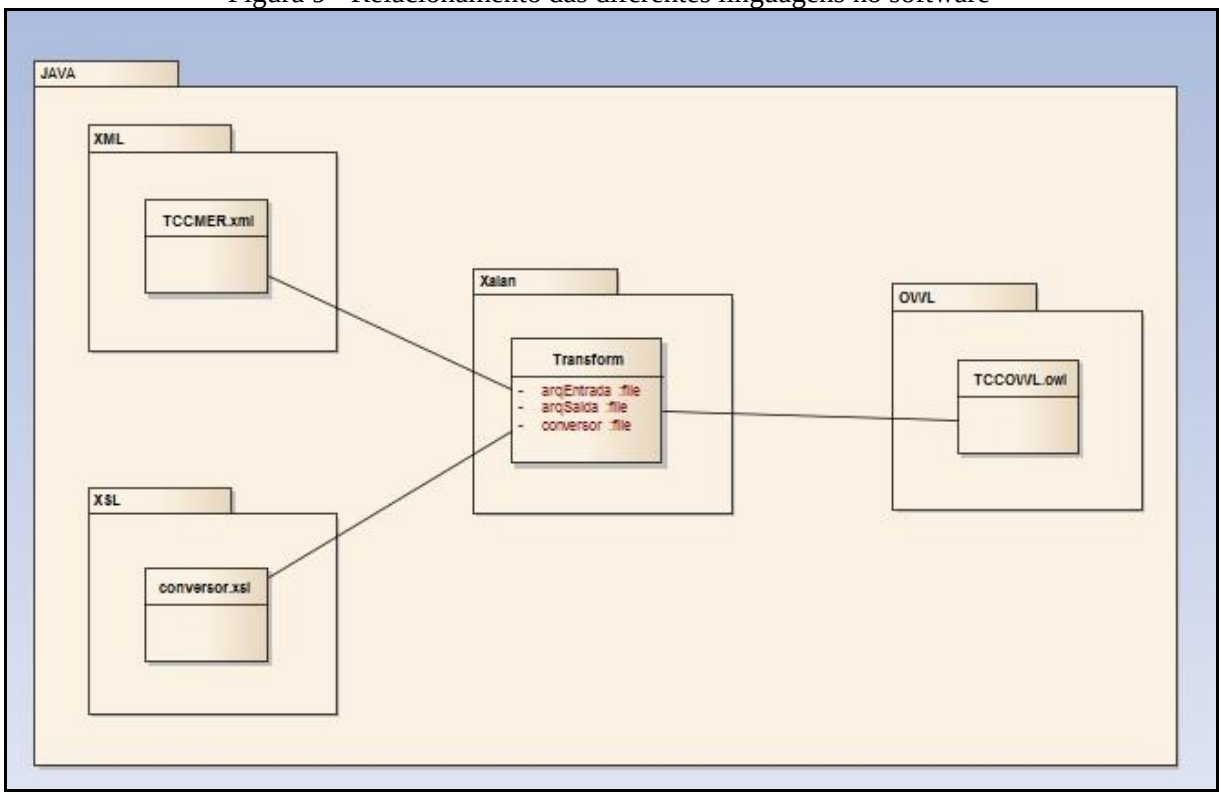
Figura 8 - Funcionalidades exercidas pelo usuário



A Figura 9 mostra a divisão interna e a interconeção das diferentes linguagens e formatos. Tudo acontecerá dentro do ambiente Java, como pode ser visto na representação a seguir. Para tal, existem 4 diferentes “módulos”: a entrada, a saída, o arquivo XSL e o arquivo Java. O arquivo Java será o responsável, com a API do Xalan, por executar a transformação do arquivo de entrada, no arquivo de saída, através do XSL.

O arquivo de entrada será um XML, o arquivo de saída será um OWL, o arquivo de transformação será um XSL e o arquivo que irá juntar todos esses será um arquivo Java. A Figura 9 mostra a representação do funcionamento do software implementado.

Figura 9 - Relacionamento das diferentes linguagens no software



3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

Inicialmente, o arquivo de origem é um XML puro que continha a descrição do MER de um BD. O mesmo foi desenvolvido no DBDesigner. A maioria dos atributos eram apenas controle interno do próprio DBDesigner, então puderam ser ignorados pelo conversor. A estrutura principal utilizada no conversor é apresentada no Quadro 2.

Quadro 2 - Estrutura XML utilizada no XSL

```
<DBMODEL>
      <TABLES>
            <TABLE Tablename="AAA">
                  <COLUMNS>
                        <COLUMN ColName="BBB" idDatatype="000"/>
                        ...

                  </COLUMNS>
            </TABLE>
            ...

      </TABLES>
      <RELATIONS>
            <RELATION FKFields="CCC=DDD\n" DestTable="111" SrcTable="222"/>
            ...

      </RELATIONS>
</DBMODEL>
```

Após essa análise, foi criado manualmente um arquivo no software Protégé-OWL que possuísse uma representação semelhante a de um BD. Com isso feito, o arquivo foi aberto e sua estrutura interna pode ser analisada. A estrutura verificada no arquivo OWL pode ser vista no Quadro 3.

Quadro 3 - Estrutura interna de um OWL do software Protégé-OWL

```
<Ontology>
      <Declaration>
            <Class IRI="#AAA"/>
      </Declaration>
      ...
      <Declaration>
            <DataProperty IRI="#BBB"/>
      </Declaration>
      ...
      <DataPropertyDomain>
            <DataProperty IRI="#CCC"/>
            <Class IRI="#DDD"/>
      </DataPropertyDomain>
      ...
      <DataPropertyRange>
            <DataProperty IRI="#EEE"/>
            <Datatype abbreviatedIRI="FFF"/>
      </DataPropertyRange>
      ...
      <SubClassOf>
            <Class IRI="#GGG"/>
            <Class IRI="#HHH"/>
      </SubClassOf>
      ...
      <EquivalentDataProperties>
            <DataProperty IRI="#III"/>
            <DataProperty IRI="#JJJ"/>
      </EquivalentDataProperties>
      ...
</Ontology>
```

Os quadros 2 e 3 compoem o objetivo específico “a”. Após essa análise, foi iniciado uma tabela de equivalência entre os elementos do XML e os elementos do OWL. Além da comparação entre os elementos, foi criada ainda uma outra tabela para os tipos básicos presentes nos dois softwares. A primeira tabela dos elementos XML pode ser encontrada no Apêndice A, e a de tipos pode ser encontrada no Apêndice B. Estes dois apêndices compoem o objetivo específico “b”.

Com a tabela em mãos, o próximo passo era escrever o XSL. Este teria que percorrer os elementos do XML, ignorando os que não são utilizados e gerar um arquivo de saída. Para isso, a estrutura do XSL escrito deveria percorrer os elementos `DBMODEL`, `TABLES`, `TABLE`, `COLUMNS`, `COLUMN`, `RELATIONS` e `RELATION` e ignorar todos os demais. A estrutura base montada para percorrer o XML pode ser vista no Quadro 4 e o XSL completo pode ser visto no Apêndice C. O objetivo específico “c” é composto do software completo desenvolvido e do Apêndice C que contém o código do conversor.

Quadro 4 - Estrutura base do XSL

```
<xsl:stylesheet>

      <xsl:template match="/">
            ...
            <xsl:apply-templates select="DBMODEL"/>
            ...
      </xsl:template>

      <xsl:template match="DBMODEL">
            ...
            <xsl:apply-templates select="METADATA/TABLES" />
            ...
            <xsl:apply-templates select="METADATA/RELATIONS" />
            ...
      </xsl:template>

      <xsl:template match="METADATA/TABLES">
            ...
          <xsl:apply-templates select="COLUMNS"/>
            ...
      </xsl:template>

      <xsl:template match="COLUMNS">
            ...
      </xsl:template>

      <xsl:template match="METADATA/RELATIONS">
            ...
            <xsl:apply-templates select="COLUMNS"/>
            ...
      </xsl:template>

</xsl:stylesheet>
```

Com esse XSL, foi possível percorrer todos os elementos de interesse, ignorando tanto elementos quando atributos desconsiderados no levantamento inicial. Além disso, foram criados `templates` extras para processamentos mais minunciosos que podem ser vistos no Apêndice C.

A interface do conversor e o código utilizado para iniciar o processamento do XSL e do XML de entrada foi escrito em Java. O Quadro 5 contem a chamada da classe `Transformer` da API Xalan para dar inicio a este processo.

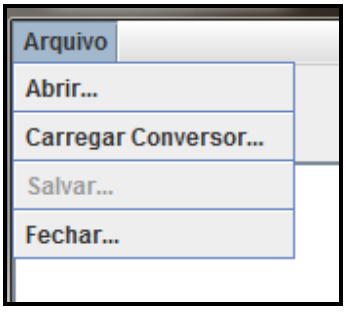
Quadro 5 - Código Java para a API Xalan

```
TransformerFactory tFactory = TransformerFactory.newInstance();
Transformer transformer =
tFactory.newTransformer(new StreamSource(arquivoConversor));
transformer.transform(new StreamSource(arquivoMEREntrada),
                      new StreamResult(tmp));
```

3.3.2 Operacionalidade da implementação

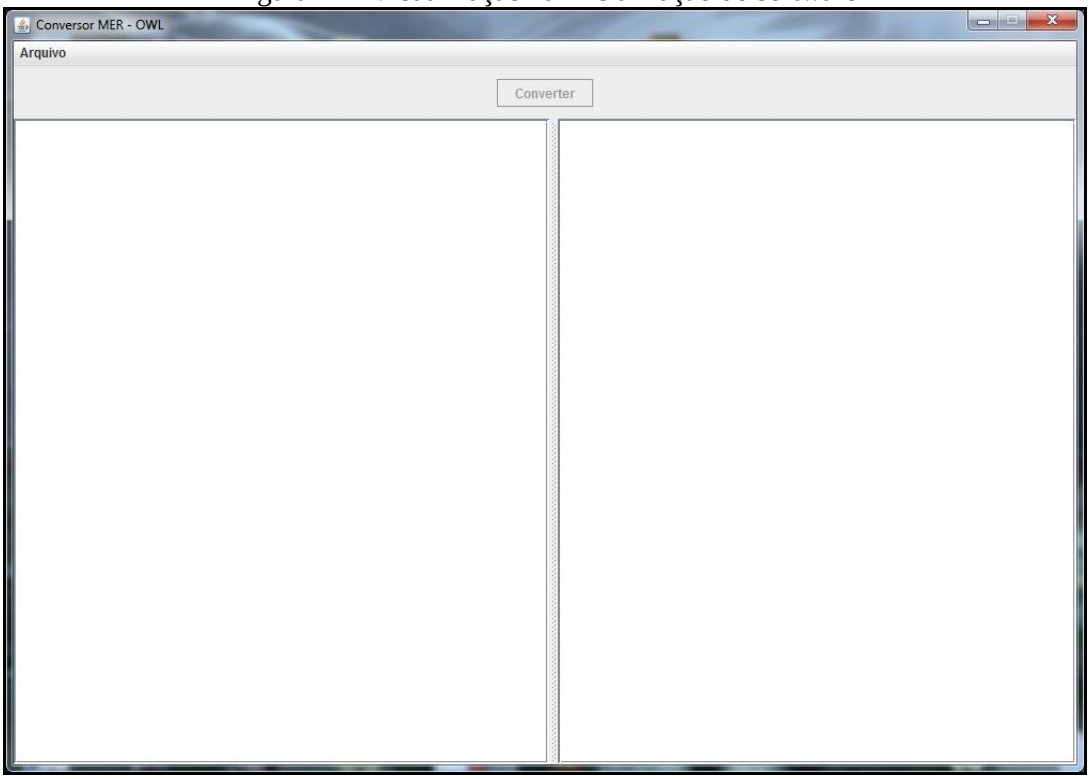
O software desenvolvido será apresentado em duas frentes: a inicial abordará o procedimento antes da conversão, enquanto a final abordará o necessário após a mesma. Para isso, basta que seja analisado o menu da aplicação desenvolvida. O mesmo pode ser visto na Figura 10. Ela mostra as opções disponíveis para o usuário antes da conversão.

Figura 10 - Menu inicial antes da conversão



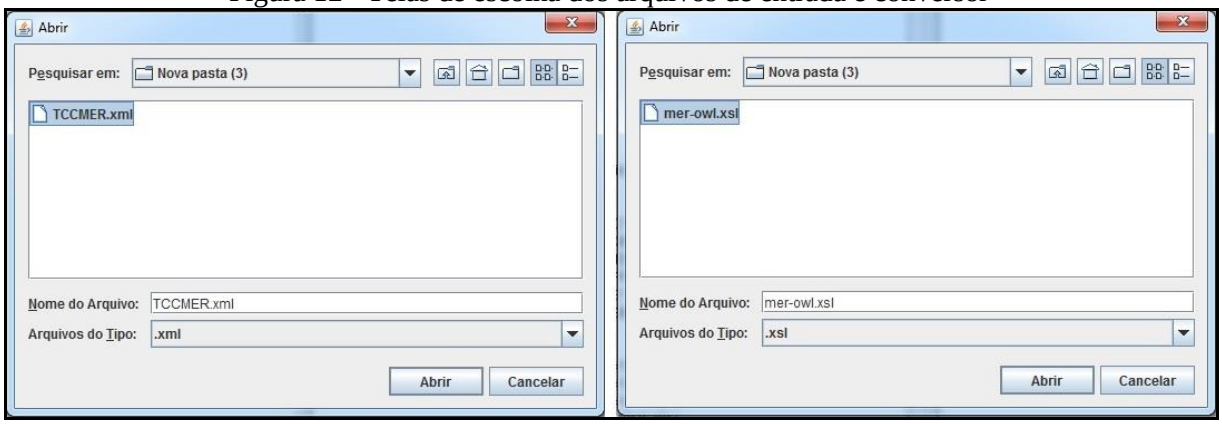
Além de verificar o menu, é importante observar a parte superior da aplicação. Nela poderá ser visto um botão inativo com o texto `Converter`. Esse botão servirá, após todos os procedimentos iniciais terem sido executados, para chamar a execução do conversor carregado. O estado inicial do aplicativo pode ser visto na Figura 11.

Figura 11 - Visualização na inicialização do software



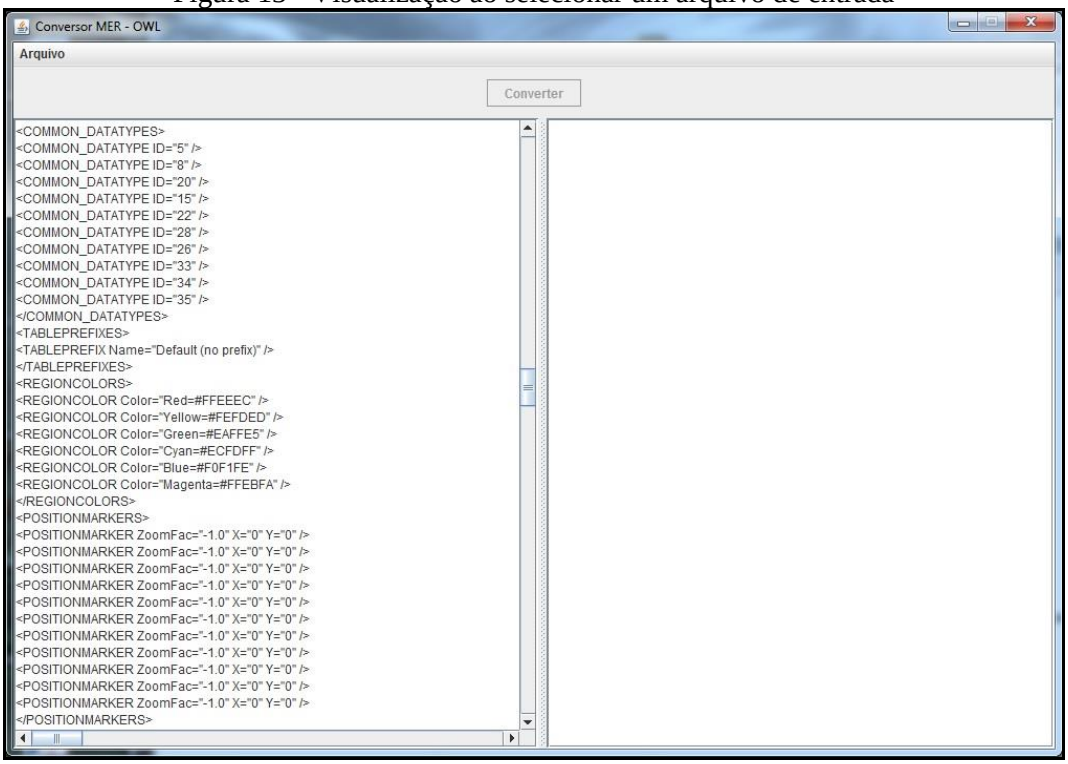
Como pode ser visto, as opções disponíveis são `Abrir` e `Carregar Conversor`. A opção `Abrir` irá permitir que seja escolhido um arquivo no formato `.xml`, enquanto a opção `Carregar Conversor` permite que o usuário escolha um arquivo do formato `.xsl` para que sirva de conversor para o processo. Ambas as telas de seleção de arquivos podem ser vistas na Figura 12 abaixo.

Figura 12 - Telas de escolha dos arquivos de entrada e conversor



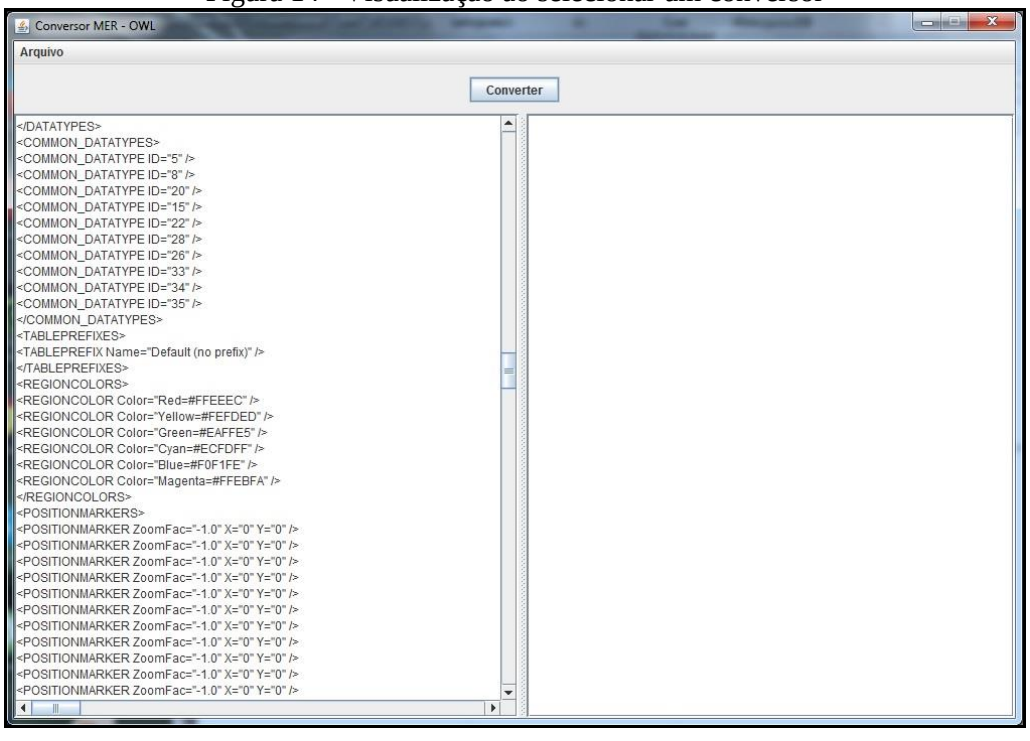
Após a escolha de um arquivo `.xml`, a área de texto a esquerda da divisória será preenchida com o texto contido no mesmo. Esta área de texto não permite edição por questões de segurança, uma vez que o software não faz a validação do texto antes do mesmo ser convertido. Um exemplo pode ser visto na Figura 13.

Figura 13 - Visualização ao selecionar um arquivo de entrada



O próximo passo é a escolha de um conversor. Isto se dá devido ao fato de que o usuário pode querer carregar outras versões do conversor, ou manipulá-lo para gerar um OWL de acordo com suas premissas. Após carregado, o botão `Converter` será liberado. Um exemplo do fim da primeira etapa com o botão `Converter` liberado pode ser visto na Figura 14.

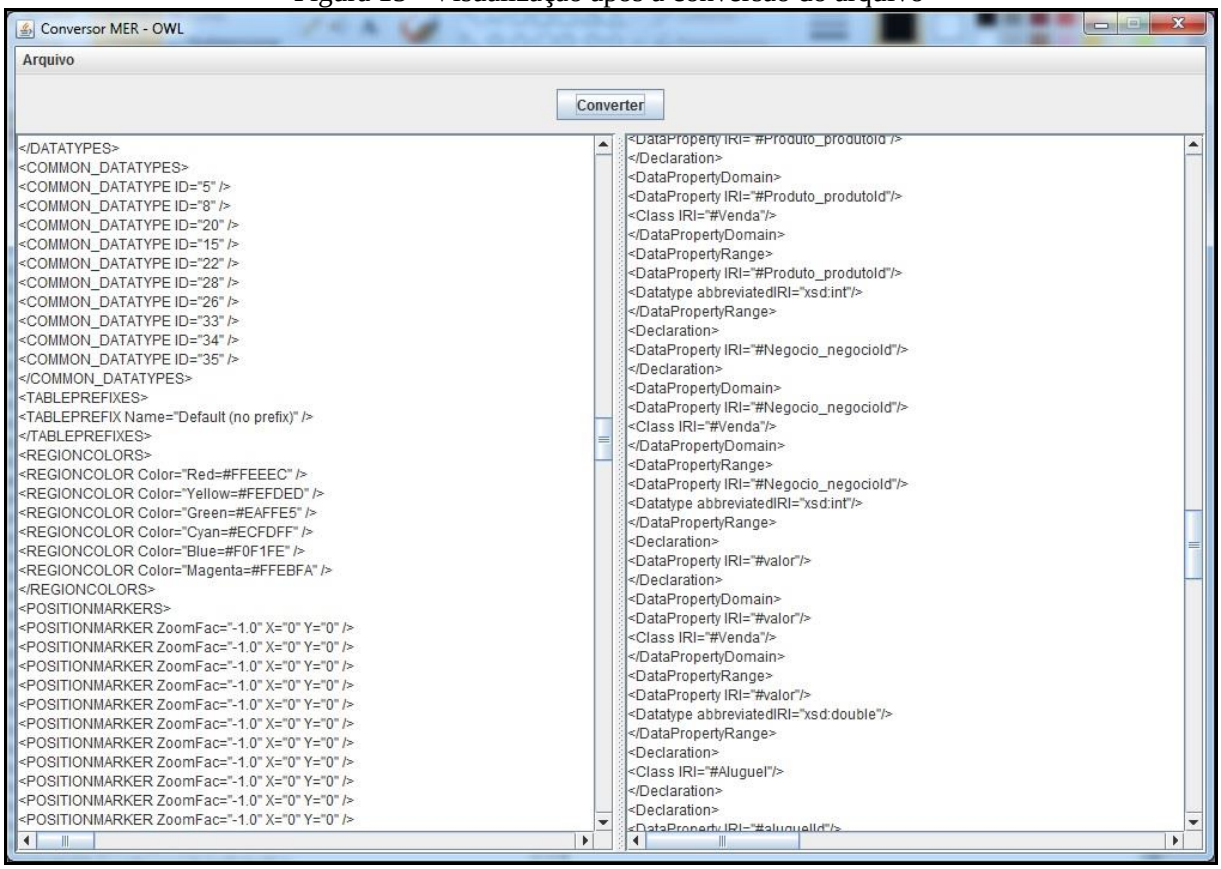
Figura 14 - Visualização ao selecionar um conversor



Após a finalização dos passos iniciais, com o botão `Converter` ativo, tudo o que é necessário fazer é pressioná-lo. Isso irá iniciar o processo de conversão interno do software. O tempo necessário para a finalização da conversão irá variar de acordo com o tamanho do arquivo de entrada.

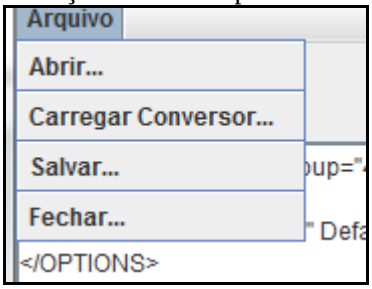
Quando o processo for finalizado, o resultado será exibido na área de texto na direita. É válido observar que esta área de texto também não permite alterações por também não possuir validação. Um exemplo do processo finalizado pode ser visto na Figura 15.

Figura 15 - Visualização após a conversão do arquivo



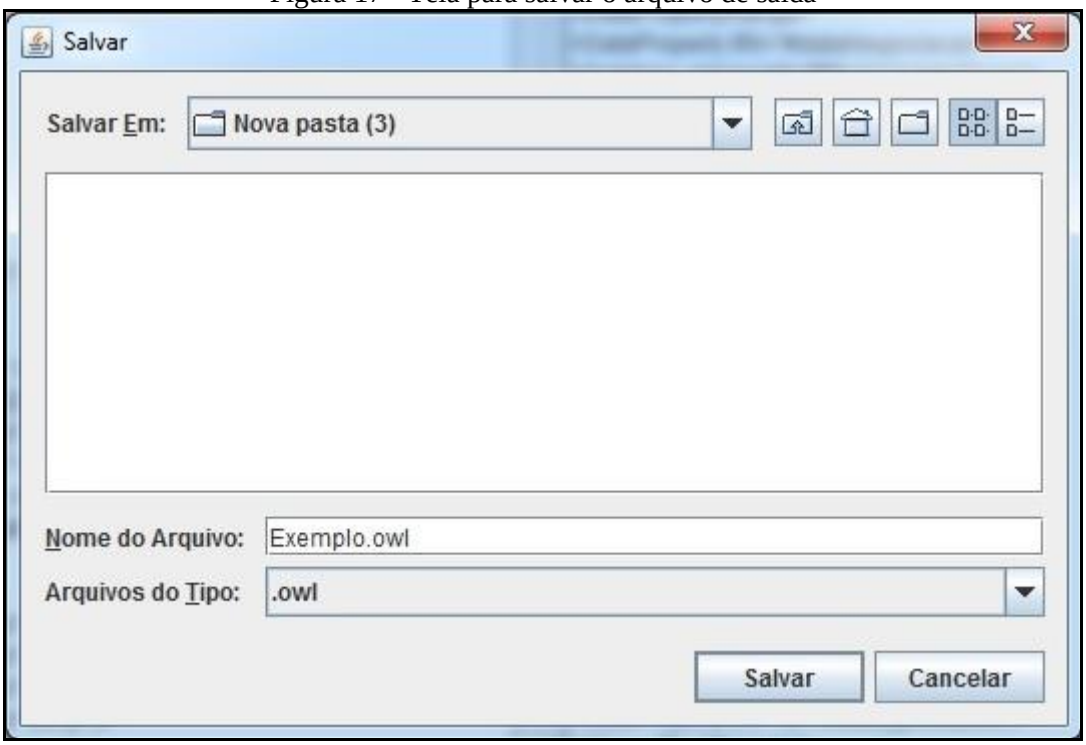
Além do arquivo ser visualizado na área de texto da direita, caso a conversão aconteça com sucesso, a opção `Salvar` do menu `Arquivo` se torna ativa, permitindo que o resultado que está sendo visualizado possa ser salvo. Isso pode ser visto na Figura 16.

Figura 16 - Visualização do menu após a conversão do arquivo



Quando a opção `Salvar` for selecionada, uma nova janela abrirá na qual se poderá escolher o local onde deseja salvar seu arquivo, bem como o seu nome. Na digitação do nome é necessário digitar também a extensão `.owl` para que o mesmo possa ser utilizado corretamente. Um exemplo da opção salvar pode ser visto na Figura 17.

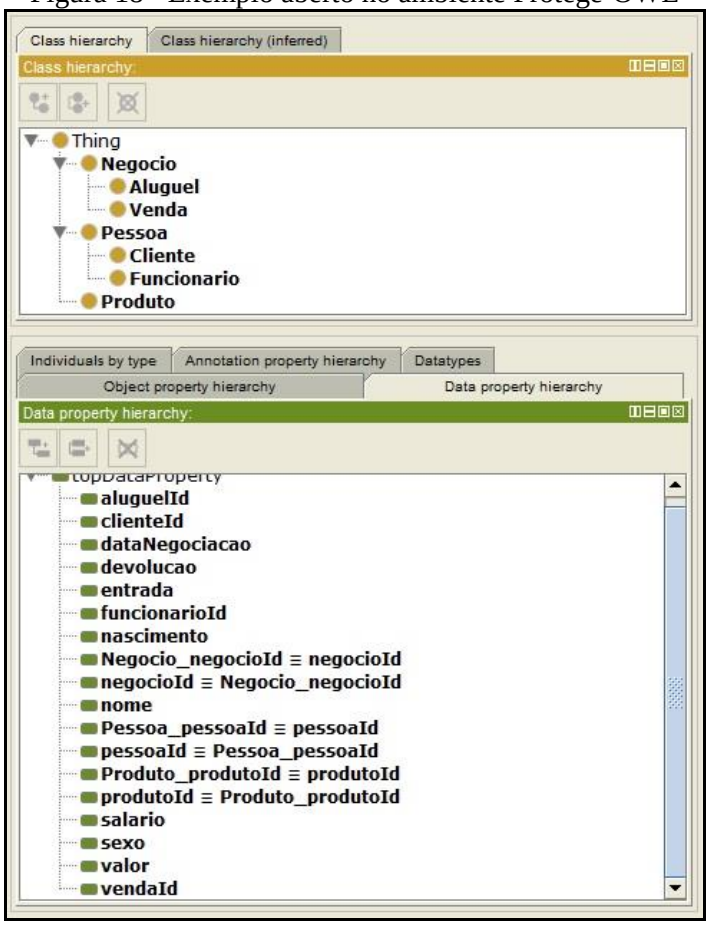
Figura 17 - Tela para salvar o arquivo de saída



Relizado todos os passos até aqui, para verificar se o arquivo de saída é valido, é necessário que o mesmo seja aberto em um editor OWL. Caso o mesmo consiga exibir um resultado, a conversão foi executada com sucesso.

A Figura 18 mostra o arquivo de exemplo aberto no ambiente Protégé-OWL. Nela se pode ver a representação de um MER dentro de uma ontologia. Essa representação se dá da seguinte forma: as tabelas se tornam uma `Class`, enquanto as colunas se tornam `DataProperty`. As relações entre as tabelas são analisadas pelo conversor, e as tabelas envolvidas em uma são transformadas uma a uma em `SubClassOf`. Da mesma forma, as colunas envolvidas em relacionamentos possuem um tratamento de equivalência, de modo a demonstrar que seus valores deverão ser idênticos aos da coluna original. No ambiente de ontologia isso é representado por uma `EquivalentDataProperty`. O resultado final pode ser visto na figura a seguir. Maiores informações sobre equivalências entre os formatos podem ser vistas no Apêndice A.

Figura 18 - Exemplo aberto no ambiente Protégé-OWL



3.4 RESULTADOS E DISCUSSÃO

Nesta seção, serão utilizados 2 exemplos exemplos para mostrar os resultados. O primeiro será um MER simples, com apenas 2 tabelas, contendo apenas 1 campo em cada uma delas. Esse teste servirá para tirar o tempo de execução do conversor em um MER pequeno. Em seguida, será executado o conversor em um MER um pouco mais complexo, com 7 tabelas relacionadas entre sí. Esse teste servirá para testar complexidade, e para testar todas as possibilidades cobridas pelo conversor.

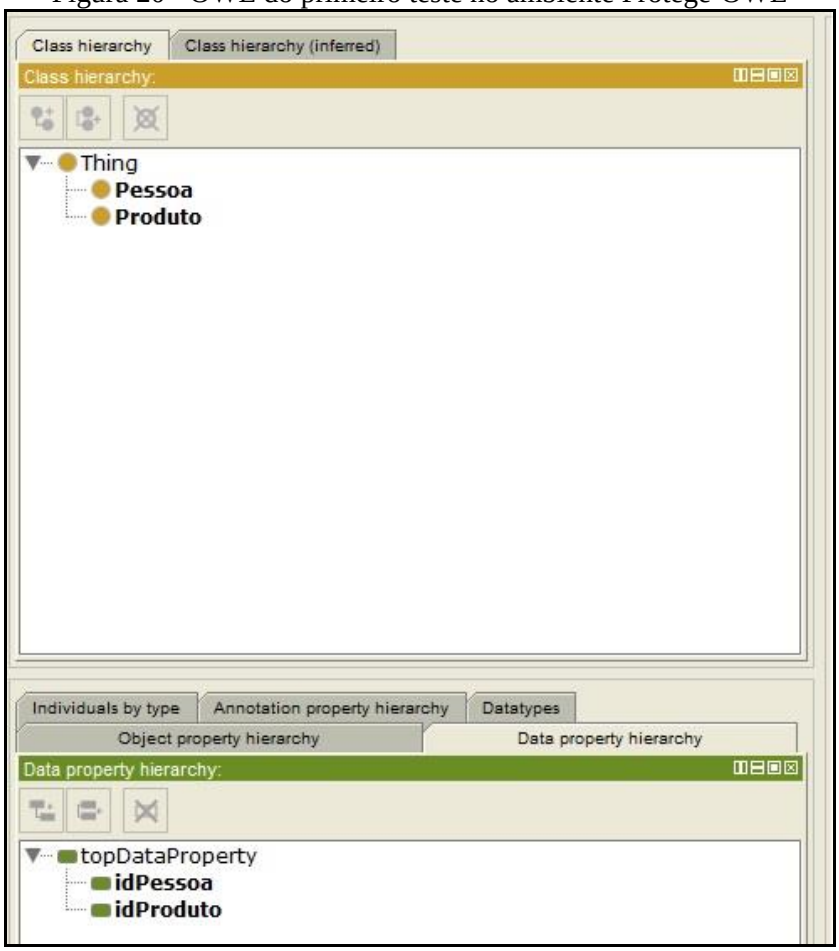
Após a demonstração dos testes executados, serão apresentadas as observações de cada um deles. Para tais demonstrações existem as 4 figuras a seguir. A Figura 19 e Figura 20 dizem respeito ao primeiro teste: elas são, respectivamente, a visão geral do MER no ambiente DBDesigner, e a ontologia gerada por ela pelo software desenvolvido, no ambiente

Protégé-OWL. Para o segundo teste, as duas figuras seguintes, Figura 21 e Figura 22, possuem a mesma representação em ambos os ambientes.

Figura 19 - MER do primeiro teste no ambiente DBDesigner

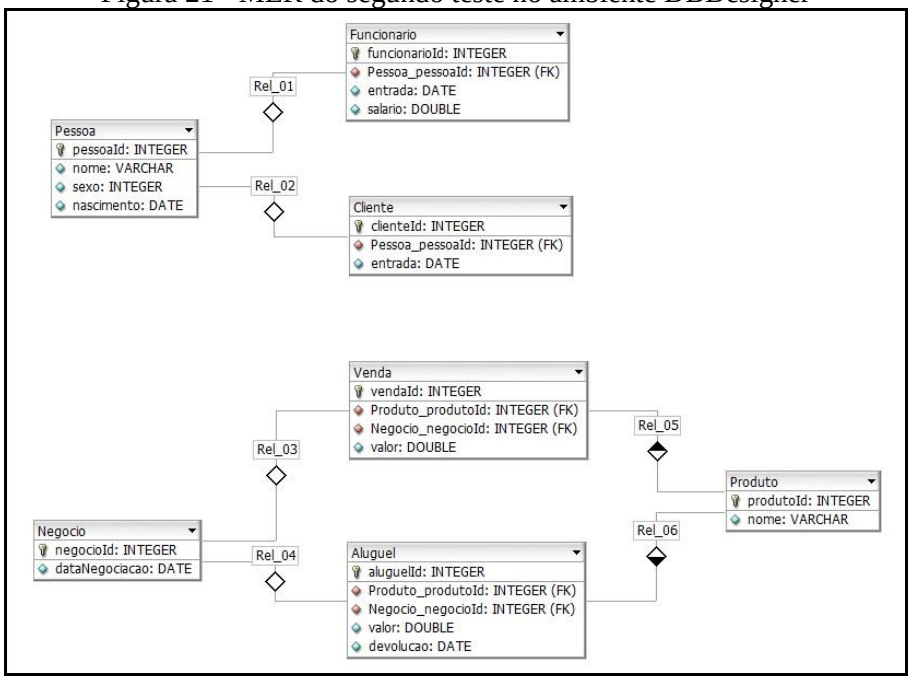
Na figura acima podem-se identificar duas tabelas sem relações entre elas, contendo apenas um campo gerado automaticamente pelo software DBDesigner para identificação das mesmas. Esse caso de teste foi submetido ao conversor, e o resultado obtido pode ser visto na Figura 20.

Figura 20 - OWL do primeiro teste no ambiente Protégé-OWL



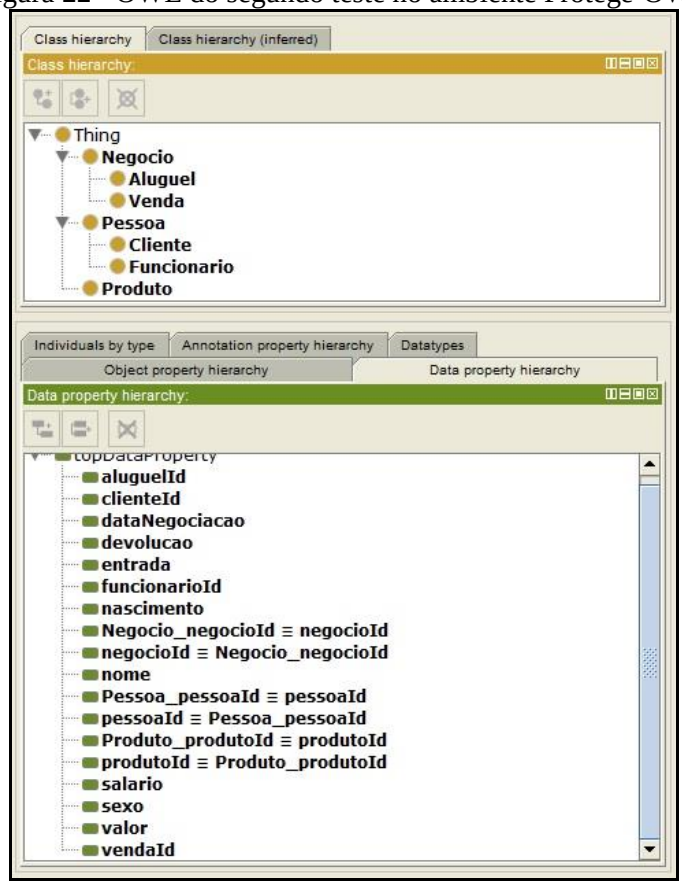
Após a conversão do caso de testes simples foi criado um caso mais complexo, com relacionamentos e campos de tipos diferentes para que pudesse ser determinado o tempo de execução da conversão em um MER com mais objetos. O caso de testes utilizado pode ser visto na Figura 21.

Figura 21 - MER do segundo teste no ambiente DBDesigner



De forma semelhante a primeira, o resultado da comparação foi aberto no software Protégé-OWL e pode ser encontrado na Figura 22. É valido ressaltar que o software utilizado para visualização da ontologia não faz a separação das propriedades por indivíduo selecionado, deixando todos visíveis em uma mesma lista.

Figura 22 - OWL do segundo teste no ambiente Protégé-OWL



Para título de comparação, o primeiro MER foi convetido em 0.437ms, enquanto o segundo precisou de 1.154ms para terminar. Isso mostra que a complexidade interfere no tempo de execução de forma expressiva, principalmente quando são utilizados casos mais próximos dos encontrados nos softwares comerciais. Um MER de cerca de 50 tabelas e 100 relações deve levar cerca de 10 segundos para terminar de acordo com uma regra de três simples. Comparando isso ao tempo que um ser humano com conhecimento na área de ontologia e no software Protégé-OWL levaria para criar os mesmos resultados, o software executa em tempo hábil.

Outro aspecto que deve ser observado é o tratamento dado para colunas com nomes iguais, em tabelas diferentes. Estas serão convertidas para um único `DataProperty` na ontologia, que possuirá duas referências para as respectivas `Class`.

Em relação aos softwares relacionados, embora não sejam conversores, todos fazem a leitura de arquivos e o *parsing* dos mesmos de alguma forma. A Tabela 1 abaixo mostra o tempo necessário para os softwares relacionados e o software desenvolvido fazerem a leitura e o *parsing* de um arquivo de 1000 linhas (superior ao segundo caso de teste).

Tabela 1 - Tempo necessário para leitura e *parsing* de arquivos XML

TEMPO DE LEITURA E *PARSING* DE XML		
Software	Tempo	Variação (em relação ao Conversor)
Jena	2.142 ms	- 59 ms
Sesame	2.217 ms	+ 16 ms
4store	1.871 ms	- 330 ms
Conversor	2.201 ms	0 ms

A tabela acima mostra que o conversor desenvolvido faz o *parsing* de um XML em tempo próximo ao das ferramentas relacionadas. Durante as versões preliminares do desenvolvimento o desempenho era superior ao dos softwares relacionados, porém devido a introdução de um algoritmo para a quebra de relacionamentos multiplos entre tabelas, o mesmo acabou sofrendo uma perda drástica.

Mesmo que no arquivo de entrada existam apenas relacionamentos simples, ainda assim o desempenho foi comprometido, uma vez que o algoritmo que causou a perda irá executar independente do número de relacões entre as tabelas. Vale ressaltar ainda que em arquivos maiores, com mais relações, o conversor irá perder ainda mais desempenho, podendo apresentar tempos de *parsing* superiores aos softwares relacionados.

Seguindo o quadro encontrada no Apêndice A e o código encontrado no Apêncide C, o conversor executou uma conversão completa do MER fornecido como entrada. Após essa conversão, foram analisados os códigos fonte da ontologia gerada para que pudesse ser criado um quadro para demonstrar o resultado da conversão. O Quadro 6 apresenta algumas das contruções do MER original e suas conversões respectivas como prova da conversão.

Quadro 6 - Tabela comparativa entre entrada e saída

`<TABLE Tablename="Pessoa">`
`...`
`</TABLE>`
	`<Declaration>`
	`<Class IRI="#Pessoa"/>`
	`</Declaration>`

`<COLUMN ColName="pessoaId">`
`...`
`</COLUMN>`
	`<Declaration>`
	`<DataProperty IRI="#pessoaId"/>`
	`</Declaration>`

`<COLUMN ColName="pessoaId"`
`idDatatype="5">`
`...`
`</COLUMN>`
	`<DataPropertyRange>`
	`<DataProperty IRI="#pessoaId"/>`
	`<Datatype`
	`abbreviatedIRI="xsd:int"/>`
	`</DataPropertyRange>`

`<TABLE ID="1000" Tablename="Pessoa">`
`...`
`</TABLE>`
`<TABLE ID="1007"`
`Tablename="Funcionario">`
`...`
`</TABLE>`
`<RELATION SrcTable="1000"`
`DestTable="1007"/>`
	`<SubClassOf>`
	`<Class IRI="#Funcionario"/>`
	`<Class IRI="#Pessoa"/>`
	`</SubClassOf>`

`<RELATION FKFields=`
`"pessoaId=Pessoa_pessoaId\n" />`
	`<EquivalentDataProperties>`
	`<DataProperty`
	`IRI="#Pessoa_pessoaId"/>`
	`<DataProperty IRI="#pessoaId"/>`
	`</EquivalentDataProperties>`

O padrão visto no quadro acima é repetido para cada nova tabela, coluna ou relação encontrada, gerando um arquivo de saída contendo a a representação do MER em uma ontologia no formato OWL. O conversor desenvolvido foi capaz de converter com êxito todos os arquivos de entrada submetidos, gerando uma ontologia válida para cada um deles.

Os softwares apresentados como relacionados são softwares que de alguma forma se relacionam com ontologias ou *parsers*, porém não são conversores, por tal motivo foram utilizados apenas para a otimização do algoritmo e para a comparação do desempenho entre o software desenvolvido e outros existentes que possuem alguma semelhança com o mesmo.

4 CONCLUSÕES

O software apresentado faz uma conversão de um MER no formato XML extraído pelo software DBDesigner em uma ontologia no formato OWL que pode ser lida pelo software Protégé-OWL. Até a presente data, no universo analisado, não foi possivel encontrar um outro software que faça tal conversão.

Os resultados obtidos são excelentes, caso as condições sejam respeitadas. O XML inicial precisa ser extraído pelo DBDesigner 4, e gerará uma ontologia que será lida no software Protégé-OWL versão 4.3. Mantendo-se essas condições, e levando em conta que o arquivo XML não tenha sido manualmente modificado para que seja apagado alguma informação necessária para o processo de conversão, ou que o arquivo OWL tenha sofrido alguma alteração manual, a mesma executará com sucesso em 100% dos casos.

Para os objetivos propostos, o presente trabalho atende satisfatóriamente a todos. Dentre os mesmos, o primeiro, que se refere a identificar os elementos e as contruções oferecidas pelas linguagens XML e OWL, na seção de desenvolvimento deste trabalho podem ser encontrados quadros que mostram quais foram utilizados ao longo da execução deste. O segundo, que diz respeito a estabelecer as regras de equivalência de conversão dos elementos XML para OWL, o Apêndice A e B tratam das conversões que foram levantadas. E por último, para especificar e implementar as regras de conversão, o que foi desenvolvido pode ser visto por completo no Apêndice C bem como na seção de desenvolvimento. Em ambos estão presentes, ou por completo, ou em trech/os, o código que foi utilizado.

O trabalho apresentado se mostra bastante útil quando se deseja uma forma diferente de representação. Enquanto um BD é uma forma de representar informações, uma ontologia é uma forma de representar conhecimento. Por isso, caso se tenha a necessidade de estudar ou analisar um problema em um ponto de vista diferente, ou caso se queira saber como uma informação estruturada ficará quando convertida em conhecimento, este conversor foi desenvolvido. O mesmo serve para que possa ser feita uma mudança de pontos de vista ou de tecnologias caso seja necessário.

Durante o desenvolvimento do mesmo, as escolhas de tecnologia para a execução do mesmo se mostraram certas. O XML gerado pelo DBDesigner é bastante simples de ser lido, e pode ser facilmente interpretado, o que facilitou a implementação do conversor. Este, por sua vez, por ter como entrada um XML, foi naturalmente levado para a escolha da XSLT. Por trabalhar nativamente com a conversão de documentos XML, por ter uma sintaxe simples, e

por possuir bibliotecas Java que façam a sua interpretação, foi a linguagem correta a ser utilizada.

Quanto a linguagem Java, embora apenas esteja presente para fazer a interpretação da XSLT, se provou de grande utilidade devido ao uso da API Xalan. Esta permitiu que, com poucas linhas de código, fosse executado um interpretador com mais funcionalidades que o nativo da linguagem.

Por último, a escolha do software Protégé-OWL como software para leitura do OWL gerado, foi uma das mais impactantes. Além de possuir uma interface simples, o software tambem conta, com os devidos *plugins* com uma visualização em formato árvore, fazendo com que a análise do arquivo gerado pelo conversor pudesse ser mais rápida e precisa.

4.1 EXTENSÕES

Para evoluir a uma total conversão de um BD para uma ontologia o software proposto deveria ainda levar os dados. O software atual faz a conversão de estruturas entre um BD e uma ontologia, sendo necessário importar os dados manualmente. Uma extensão seria a conversão dos dados de um BD para uma ontologia.

Outra opção é melhorar o software, ampliando a variedade de formatos aceitos pelo conversor, ou a versão do software Protégé-OWL no qual o arquivo convertido poderá ser executado.

Além da versão do software de leitura da saida, uma extensão tambem seria a ampliação dos formatos de entrada, passando a aceitar XMLs gerados em outros softwares que não o DBDesigner.

É necessário tambem um validador das entradas e saídas, para que o software possa permitir que o usuário manipule ou crie as informações dentro do mesmo sem riscos de cometer alguma falha que comprometa o processo.

Por último é necessário que sejam levadas em conta as *constraints* das colunas durante a conversão.

REFERÊNCIAS BIBLIOGRÁFICAS

APACHE. **Jena.** [S.l.], [2013]. Disponível em: <http://jena.apache.org/index.html>. Acesso em: 07 ago. 2013

_____. **Xalan.** [S.l.], [2012]. Disponível em: <http://xalan.apache.org/index.html>. Acesso em: 01 out. 2013

_____. **Xalan-J.** [S.l.], [2012]. Disponível em: <http://xalan.apache.org/xalan-j/index.html>. Acesso em: 01 out. 2013

BAX, Marcelo P.; ALMEIDA, Mauricio B. **Uma visão geral sobre ontologias:** pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção. [Minas Gerais], [2003]. Disponível em: <http://www.scielo.br/pdf/ci/v32n3/19019>. Acesso em: 02 set. 2012.

DATE, Christopher J. **Introdução a sistemas de bancos de dados**. 8. ed. Rio de Janeiro: Campus, 2004.

FLORENTINO, Pablo V. **Banco de dados I**. [Bahia], [2009]. Disponível em: <http://www.ifba.edu.br/professores/pablovf/repositorio/NotasdeAulas1.pdf>. Acesso em: 01 nov. 2012.

FABFORCE. **DBDesigner 4**. [S.l.], [2003]. Disponível em: <http://www.fabforce.net/dbdesigner4/>. Acesso em: 05 out. 2013.

GARLIK. **4store**. [Inglaterra], [2009]. Disponível em: <http://4store.org/about>. Acesso em: 01 nov. 2012.

_____. **4store**. [Inglaterra], [2011]. Disponível em: <http://www.garlik.com/home>. Acesso em: 01 nov. 2012.

GUIMARÃES, Francisco J. Z. **Utilização de ontologias no domínio B2C**. [Rio de Janeiro], [2002]. Disponível em: <http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0024134_02_cap_04.pdf>. Acesso em: 31 out. 2012.

HEINZLE, Roberto. **Um modelo de engenharia do conhecimento para sistemas de apoio a decisão com recursos para raciocínio abdutivo**. 2011. 247 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. Disponível em: <http://btd.egc.ufsc.br/wp-content/uploads/2011/05/Roberto_Heinzle.pdf>. Acesso em: 07 nov. 2012.

HERMAN, Ivan. **SPARQL is a recommendation**. [Amsterdã], [2008]. Disponível em: <http://www.w3.org/blog/SW/2008/01/15/sparql_is_a_recommendation/>. Acesso em: 01 nov. 2012.

HEUSER, Carlos A. **Projeto de banco de dados**. 6. ed. Porto Alegre: Bookman, 2008.

KEMCZINSKI, Avanilde. **Sistema de Banco de Dados Relacional.** [Santa Catarina], [2008]. Disponível em: <http://www2.joinville.udesc.br/~dcc2efcm/SGBDprofaAvanilde.pdf>. Acesso em 03 set. 2012.

MALUSZYNSKI, Jan. **Ontologies I**. [Suécia],[2002]. Disponível em: <http://www.ida.liu.se/~janma/SemWeb/Slides/ontologies1.pdf>. Acesso em: 01 nov. 2012.

MELLO, Mauricio C. F. de. **O modelo entidade-relacionamento (MER)**. [Paraná], [2005]. Disponível em: <http://www.las.pucpr.br/mcfmello/BD/BD-Aula02-MER.pdf>. Acesso em: 01 nov. 2012.

OPENRDF. **Sesame**. [S.l.], [2012]. Disponível em: <http://www.openrdf.org/index.jsp>. Acesso em: 07 ago. 2013.

PROTÉGÉ. W**hat is protégé-owl?** [Califórnia],[2013]. Disponível em: <http://protege.stanford.edu/overview/protege-owl.html>. Acesso em: 05 out. 2013.

W3C. **Ontologies**. [S.l.], [2013]. Disponível em: <http://www.w3.org/standards/semanticweb/ontology>. Acesso em: 07 ago. 2013.

_____. **Ontologies**. [S.l.], [2004]. Disponível em: <http://www.w3.org/RDF/>. Acesso em: 26 ago. 2013.

_____. **The Extensible Stylesheet Language family (XSL)**. [S.l.], [2013]. Disponível em: <http://www.w3.org/Style/XSL/>. Acesso em: 05 out. 2013.

_____. **What is XSL?** [S.l.], [2013]. Disponível em: <http://www.w3.org/Style/XSL/WhatIsXSL.html>. Acesso em: 05 out. 2013.

APÊNDICE A – Quadro de equivalência entre elementos XML e OWL

Neste apêndice poderá ser visto o quadro de equivalência entre elementos XML e OWL que foi utilizada no desenvolvimento deste trabalho. Este consiste em duas colunas: a primeira a esquerda irá conter os elementos do arquivo XML enquanto a segunda a direita ira conter os elementos OWL. O Quadro 7 pode ser visto abaixo.

Quadro 7 - Quadro de equivalência entre elementos XML e OWL

`<TABLE Tablename="AAA">`
`    ...`
`</TABLE>`
	`<Declaration>`
	`    <Class IRI="#AAA"/>`
	`</Declaration>`

`<COLUMN ColName="BBB">`
`    ...`
`</COLUMN>`
	`<Declaration>`
	`    <DataProperty IRI="#BBB"/>`
	`</Declaration>`

`<COLUMN ColName="CCC"`
`idDatatype="DDD">`
`    ...`
`</COLUMN>`
	`<DataPropertyRange>`
	`    <DataProperty IRI="#CCC"/>`
	`    <Datatype abbreviatedIRI="DDD"/>`
	`</DataPropertyRange>`

`<TABLE Tablename="EEE">`
`    <COLUMNS>`
`        <COLUMN ColName="FFF"/>`
`    </COLUMNS>`
`</TABLE`
	`<DataPropertyDomain>`
	`    <DataProperty IRI="#FFF"/>`
	`    <Class IRI="#EEE"/>`
	`</DataPropertyDomain>`

`<RELATION SrcTable="GGG"`
`DestTable="HHH">`
`    ...`
`</RELATION`
	`<SubClassOf>`
	`    <Class IRI="#GGG"/>`
	`    <Class IRI="#HHH"/>`
	`</SubClassOf>`

`<RELATION FKFields="III=JJJ\n">`
`    ...`
`</RELATION>`
	`<EquivalentDataProperties>`
	`    <DataProperty IRI="#III"/>`
	`    <DataProperty IRI="#JJJ"/>`
	`</EquivalentDataProperties>`

O quadro comparativo apresenta como foi feita a tomada de decisões durante a montagem do arquivo de saída para o conversor. Para cada uma das entradas encontradas no arquivo XML, o correspondente nesta tabela foi gerado pelo XLS para o arquivo OWL.

APÊNDICE B – Quadro de equivalência entre tipos no XML e OWL

Para uma total conversão, é necessário que os tipos presentes em cada uma das colunas possam também ser convertidos. Para esse fim, o presente quadro mostra qual foram as equivalências levadas em conta. Por questões de diferenças entre as tecnologias, os tipos *built in* eram diferentes entre os softwares. Para solucionar esses casos, os tipos mais próximos foram utilizados. Caso algum tipo não possuísse correspondente próximo, foi utilizado a estratégia de gerar um tipo `xsd:string` para que o software pudesse gerar todos os registros com um tipo. O Quadro 8 mostra as decisões tomadas pelo software.

Quadro 8 - Equivalência de tipos entre o XML e o OWL

TINYINT	xsd:int
SMALLINT	xsd:int
MEDIUMINT	xsd:int
INT	xsd:int
INTEGER	xsd:int
BIGINT	xsd:integer
FLOAT	xsd:float
FLOAT	xsd:float
DOUBLE	xsd:double
DOUBLE PRECISION	xsd:double
REAL	owl:real
DECIMAL	xsd:decimal
NUMERIC	xsd:decimal
DATE	xsd:dateTime
DATETIME	xsd:dateTime
TIMESTAMP	xsd:dateTimeStamp
TIME	xsd:string
CHAR	xsd:string
VARCHAR	xsd:string
BIT	xsd:string
BOOL	xsd:boolean
TINYBLOB	xsd:string
BLOB	xsd:string
MEDIUMBLOB	xsd:string
LONGBLOB	xsd:string
TINYTEXT	xsd:string
TEXT	xsd:string
MEDIUMTEXT	xsd:string
LONGTEXT	xsd:string
ENUM	xsd:int
SET	xsd:string
Varchar(20)	xsd:string
Varchar(45)	xsd:string
Varchar(255)	xsd:string

Todos os outros tipos que não estão nesta lista serão considerados como `xsd:string` para o software, mesmo que sejam tipos não existentes e informados manualmente no código fonte do XML.

APÊNDICE C – Código XSL desenvolvido para conversão de XML em OWL

O quadro abaixo contém o código utilizado para a conversão de um XML em um OWL. Ele percorre, a partir da raiz do documento (identificada sempre dentro do XSL como "/"), todos os elementos descritos na seção de desenvolvimento, e durante sua execução vai gerando o arquivo OWL de saída. Este arquivo é então armazenado temporariamente para depois ser exibido para o usuário. O código pode ser visto no Quadro 9.

Quadro 9 - Código XSL para conversão de XML em OWL

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.semanticweb.org/johann/TCC/2013/9/TCCOWL"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace">
      <xsl:output method="xml" indent="yes" />

      <xsl:template match="/">


<!-- Inicio do documento gerado contendo os cabeçalhos e namespaces -->
<xsl:text disable-output-escaping="yes">
&lt;!DOCTYPE Ontology [
      &lt;!ENTITY xsd "http://www.w3.org/2001/XMLSchema#"&gt;
      &lt;!ENTITY xml "http://www.w3.org/XML/1998/namespace"&gt;
      &lt;!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#"&gt;
      &lt;!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#"&gt;
]&gt;
</xsl:text>
<Ontology
ontologyIRI="http://www.semanticweb.org/johann/TCC/2013/9/TCCOWL">
      <Prefix name="" IRI="http://www.w3.org/2002/07/owl#"/>
      <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
      <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"/>
      <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
      <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>

      <!-- Chamada do template para matching com elementos DBMODEL -->
      <xsl:apply-templates select="DBMODEL"/>

      <!-- Fim do documento OWL gerado -->
</Ontology>
      </xsl:template>

      <xsl:template match="DBMODEL">

      <!-- Chamada do template para matching dos elementos TABLES -->
      <xsl:apply-templates select="METADATA/TABLES" />
      <!-- Chamada do template para matching dos elementos RELATIONS -->
      <xsl:apply-templates select="METADATA/RELATIONS" />
```

```
	</xsl:template>

	<xsl:template match="METADATA/TABLES">

	<!-- Chamada do template para matching dos elementos TABLE -->
	<xsl:for-each select="TABLE">
	<Declaration>
		<Class IRI="#{@Tablename}"/>
	</Declaration>
		<xsl:apply-templates select="COLUMNS"/>
	</xsl:for-each>

	</xsl:template>

	<xsl:template match="COLUMNS">

		<xsl:for-each select="COLUMN">

	<!-- Declaração das DataPropertiy/Columns no OWL -->
	<Declaration>
		<DataProperty IRI="#{@ColName}"/>
	</Declaration>

	<!-- Ligação entre DataProperty/Class ~~ Columns/Table -->
	<DataPropertyDomain>
		<DataProperty IRI="#{@ColName}"/>
		<Class IRI="#{../../@Tablename}"/>
	</DataPropertyDomain>

	<!-- Ligação entre DataProperty/Tipo -->
	<DataPropertyRange>
		<DataProperty IRI="#{@ColName}"/>
		<Datatype>
			<xsl:attribute name="abbreviatedIRI">
			<xsl:call-template name="getTipo">
			<xsl:with-param name="idTipo" select="@idDatatype"/>
			</xsl:call-template>
			</xsl:attribute>
		</Datatype>
	</DataPropertyRange>

		</xsl:for-each>
	</xsl:template>

	<xsl:template match="METADATA/RELATIONS">

		<xsl:for-each select="RELATION">

	<!-- Declaração das subclasses/relações -->
			<xsl:if test="@Kind = '5'">
	<SubClassOf>
		<Class>
			<xsl:attribute name="IRI"><xsl:text>#</xsl:text>
			<xsl:call-template name="getNomeTabelaPai">
			<xsl:with-param name="idPai" select="@DestTable"/>
			</xsl:call-template>
			</xsl:attribute>
		</Class>
		<Class>
			<xsl:attribute name="IRI"><xsl:text>#</xsl:text>
			<xsl:call-template name="getNomeTabelaPai">
```

```
                <xsl:with-param name="idPai" select="@SrcTable"/>
                </xsl:call-template>
                </xsl:attribute>
          </Class>
    </SubClassOf>
                </xsl:if>

    <!-- Declaração das relações entre DataProperty(Columns) -->

    <xsl:call-template name="getRelacoes">
    <xsl:with-param name="stringCampos" select="@FKFields"/>
    </xsl:call-template>


          </xsl:for-each>
    </xsl:template>

    <xsl:template name="getTipo"><xsl:param name="idTipo"/>

          <xsl:choose>

                <!-- TINYINT -->
                <xsl:when test="$idTipo = '1'">
                      <xsl:text>xsd:int</xsl:text>
                </xsl:when>
                <!-- SMALLINT -->
                <xsl:when test="$idTipo = '2'">
                      <xsl:text>xsd:int</xsl:text>
                </xsl:when>
                <!-- MEDIUMINT -->
                <xsl:when test="$idTipo = '3'">
                      <xsl:text>xsd:int</xsl:text>
                </xsl:when>
                <!-- INT -->
                <xsl:when test="$idTipo = '4'">
                      <xsl:text>xsd:int</xsl:text>
                </xsl:when>
                <!-- INTEGER -->
                <xsl:when test="$idTipo = '5'">
                      <xsl:text>xsd:int</xsl:text>
                </xsl:when>
                <!-- BIGINT -->
                <xsl:when test="$idTipo = '6'">
                      <xsl:text>xsd:integer</xsl:text>
                </xsl:when>
                <!-- FLOAT -->
                <xsl:when test="$idTipo = '7'">
                      <xsl:text>xsd:float</xsl:text>
                </xsl:when>
                <!-- FLOAT -->
                <xsl:when test="$idTipo = '8'">
                      <xsl:text>xsd:float</xsl:text>
                </xsl:when>
                <!-- DOUBLE -->
                <xsl:when test="$idTipo = '9'">
                      <xsl:text>xsd:double</xsl:text>
                </xsl:when>
                <!-- DOUBLE PRECISION -->
                <xsl:when test="$idTipo = '10'">
                      <xsl:text>xsd:double</xsl:text>
                </xsl:when>
```

```
                    <!-- REAL -->
                    <xsl:when test="$idTipo = '11'">
                          <xsl:text>owl:real</xsl:text>
                    </xsl:when>
                    <!-- DECIMAL -->
                    <xsl:when test="$idTipo = '12'">
                          <xsl:text>xsd:decimal</xsl:text>
                    </xsl:when>
                    <!-- NUMERIC -->
                    <xsl:when test="$idTipo = '13'">
                          <xsl:text>xsd:decimal</xsl:text>
                    </xsl:when>
                    <!-- DATE -->
                    <xsl:when test="$idTipo = '14'">
                          <xsl:text>xsd:dateTime</xsl:text>
                    </xsl:when>
                    <!-- DATETIME -->
                    <xsl:when test="$idTipo = '15'">
                          <xsl:text>xsd:dateTime</xsl:text>
                    </xsl:when>
                    <!-- TIMESTAMP -->
                    <xsl:when test="$idTipo = '16'">
                          <xsl:text>xsd:dateTimeStamp</xsl:text>
                    </xsl:when>
                    <!-- TIME -->
                    <xsl:when test="$idTipo = '17'">
                          <xsl:text>xsd:string</xsl:text>
                    </xsl:when>
                    <!-- YEAR -->
                    <xsl:when test="$idTipo = '18'">
                          <xsl:text>xsd:string</xsl:text>
                    </xsl:when>
                    <!-- CHAR -->
                    <xsl:when test="$idTipo = '19'">
                          <xsl:text>xsd:string</xsl:text>
                    </xsl:when>
                    <!-- VARCHAR -->
                    <xsl:when test="$idTipo = '20'">
                          <xsl:text>xsd:string</xsl:text>
                    </xsl:when>
                    <!-- BIT -->
                    <xsl:when test="$idTipo = '21'">
                          <xsl:text>xsd:string</xsl:text>
                    </xsl:when>
                    <!-- BOOL -->
                    <xsl:when test="$idTipo = '22'">
                          <xsl:text>xsd:boolean</xsl:text>
                    </xsl:when>
                    <!-- TINYBLOB -->
                    <xsl:when test="$idTipo = '23'">
                          <xsl:text>xsd:string</xsl:text>
                    </xsl:when>
                    <!-- BLOB -->
                    <xsl:when test="$idTipo = '24'">
                          <xsl:text>xsd:string</xsl:text>
                    </xsl:when>
                    <!-- MEDIUMBLOB -->
                    <xsl:when test="$idTipo = '25'">
                          <xsl:text>xsd:string</xsl:text>
                    </xsl:when>
                    <!-- LONGBLOB -->
```

```
                  <xsl:when test="$idTipo = '26'">
                        <xsl:text>xsd:string</xsl:text>
                  </xsl:when>
                  <!-- TINYTEXT -->
                  <xsl:when test="$idTipo = '27'">
                        <xsl:text>xsd:string</xsl:text>
                  </xsl:when>
                  <!-- TEXT -->
                  <xsl:when test="$idTipo = '28'">
                        <xsl:text>xsd:string</xsl:text>
                  </xsl:when>
                  <!-- MEDIUMTEXT -->
                  <xsl:when test="$idTipo = '29'">
                        <xsl:text>xsd:string</xsl:text>
                  </xsl:when>
                  <!-- LONGTEXT -->
                  <xsl:when test="$idTipo = '30'">
                        <xsl:text>xsd:string</xsl:text>
                  </xsl:when>
                  <!-- ENUM -->
                  <xsl:when test="$idTipo = '31'">
                        <xsl:text>xsd:int</xsl:text>
                  </xsl:when>
                  <!-- SET -->
                  <xsl:when test="$idTipo = '32'">
                        <xsl:text>xsd:string</xsl:text>
                  </xsl:when>
                  <!-- Varchar(20) -->
                  <xsl:when test="$idTipo = '33'">
                        <xsl:text>xsd:string</xsl:text>
                  </xsl:when>
                  <!-- Varchar(45) -->
                  <xsl:when test="$idTipo = '34'">
                        <xsl:text>xsd:string</xsl:text>
                  </xsl:when>
                  <!-- Varchar(255) -->
                  <xsl:when test="$idTipo = '35'">
                        <xsl:text>xsd:string</xsl:text>
                  </xsl:when>
                  <xsl:otherwise>
                        <xsl:text>xsd:string</xsl:text>
                  </xsl:otherwise>
            </xsl:choose>
      </xsl:template>

      <xsl:template name="getNomeTabelaPai"><xsl:param name="idPai"/>

            <xsl:value-of
select="/DBMODEL/METADATA/TABLES/TABLE[@ID=$idPai]/@Tablename"/>

      </xsl:template>

      <xsl:template name="getRelacoes"><xsl:param name="stringCampos"/>

            <xsl:if test="string-length($stringCampos) > 2">
            <xsl:text disable-output-escaping="yes">
&lt;EquivalentDataProperties>
</xsl:text>
            <DataProperty IRI="#{substring-before($stringCampos,'=')}"/>
            <xsl:call-template name="getRelacoesIgual">
            <xsl:with-param name="stringCampos" select="substring-
```

```
after($stringCampos,'=')"/>
            </xsl:call-template>
            </xsl:if>
      </xsl:template>

      <xsl:template name="getRelacoesIgual"><xsl:param
name="stringCampos"/>

            <xsl:if test="string-length($stringCampos) > 0">
            <DataProperty IRI="#{substring-before($stringCampos,'\n')}"/>
            <xsl:text disable-output-escaping="yes">
&lt;/EquivalentDataProperties>
</xsl:text>
            <xsl:call-template name="getRelacoes">
            <xsl:with-param name="stringCampos" select="substring-
after($stringCampos,'\n')"/>
            </xsl:call-template>
            </xsl:if>
      </xsl:template>

</xsl:stylesheet>
```