UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA PARA CRIAÇÃO DE BASES DE CONHECIMENTO NA FORMA DE ONTOLOGIA OWL A PARTIR DE DADOS NÃO ESTRUTURADOS

ALLAN RENATO SABINO

BLUMENAU
2014

2014/1-01

ALLAN RENATO SABINO

FERRAMENTA PARA CRIAÇÃO DE BASES DE CONHECIMENTO NA FORMA DE ONTOLOGIA OWL A PARTIR DE DADOS NÃO ESTRUTURADOS

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
2014

2014/1-01

FERRAMENTA PARA CRIAÇÃO DE BASES DE CONHECIMENTO NA FORMA DE ONTOLOGIA OWL A PARTIR DE DADOS NÃO ESTRUTURADOS

Por

ALLAN RENATO SABINO

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:	Profa. Joyce Martins, Mestre – FURB
Membro:	Prof. Mauricio Capobianco Lopes, Doutor – FURB

Blumenau, 1 de Julho de 2014

Dedico este trabalho à minha família e amigos. Especialmente para minha mãe, pois sem ela a realização do mesmo não seria possível.

AGRADECIMENTOS

A Deus, por me possibilitar a conclusão de mais uma fase da minha vida.

À minha família, pela confiança e apoio durante a realização deste trabalho.

Aos meus amigos, pela paciência durante os longos momentos de ausência.

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

Ao meu colega de curso Leandro Vilson Battisti, pela ajuda com o tema Processamento de Linguagem Natural.

Ao professor José Roque Voltolini da Silva, por todo o apoio desde a elaboração da proposta, até a finalização desta monografia, por todos os puxões de orelha, por ser um pai acadêmico e por ser um modelo de pessoa e de professor. Minha maior meta na vida é conseguir ser um professor igual ele!

Ao professor Mauricio Capobianco Lopes, por todo o apoio durante elaboração da monografia.

À minha colega de turma Mayra Zanchett Manchein, pela ajuda com a tradução do resumo para inglês.

Algo só é impossível até que alguém duvide e resolva provar o contrário.

Albert Einstein

RESUMO

Este trabalho apresenta a especificação e implementação de uma ferramenta para a criação de bases de conhecimento na forma de uma ontologia OWL a partir de textos não estruturados. O processo se divide em duas etapas: descoberta do conhecimento e criação da base de conhecimento. Na primeira etapa são utilizados mineração de texto e processamento de linguagem natural para descobrir conhecimento a partir de uma coleção de documentos textuais. Por fim, é utilizada a especificação de *tags* OWL da ferramenta Protégé para criar uma ontologia no formato OWL. A ontologia criada foi aberta pela ferramenta Protégé, garantindo assim que ela segue o padrão de *tags* OWL utilizado pelo mesmo. Porém, por causa de algumas abordagens utilizadas durante a etapa de pré-processamento, existem algumas limitações para a identificação das estruturas ontológicas.

Palavras-chave: Mineração de texto. Representação do conhecimento. Processamento de linguagem natural.

ABSTRACT

This work presents the specification and development of a tool to creation of knowledge bases in the form of OWL ontology from non-structured texts. The process divides itself in two steps: knowledge discovery and creation of the knowledge base. In the first step, are used text mining and natural language processing to discover knowledge from a textual documents collection. Lastly, has been used the OWL tags specification from the Protégé tool. The created ontology was opened by Protégé tool, thus ensuring that it follows the pattern of OWL tags used by the tool. However, because of some approaches used during the pre-processing step, there are some limitations to the identification of ontological structures.

Key-words: Text mining. Knowledge representation. Natural language processing.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ACM - *Association for Computing Machinery*

AM - Aprendizado de Máquina

API - *Application Programming Interface*

DARPA - *Defense Advanced Research Projects Agency*

DCBD - Descoberta de Conhecimento em Bancos de Dados

DCT - Descoberta de Conhecimento em Texto

EA - Enterprise Architect

EI - Extração de Informação

LGPL - GNU *Lesser General Public License*

HTML - *Hyper Text Markup Language*

IA - Inteligência Artificial

IRI - *Internacionalized Resource Identifier*

IST - *Information Society Tecnologies*

MD - Mineração de Dados

MT - Mineração de Texto

MVC - *Model View Controller*

OIL - *Ontology Inference Layer*

OWL - *Web Ontology Language*

PLN - Processamento de Linguagem Natural

RC - Representação do Conhecimento

RDF - *Resource Description Framework*

RDFS - *Resource Description Framework Schema*

RF – Requisito Funcional

RN - Redes Neurais

RNF – Requisito Não Funcional

RI - Recuperação de Informação

SIGIR - *Special Interest Group on Information Retrival*

SGBD - Sistemas Gerenciadores de Bancos de Dados

TI - Tecnologia da Informação

TMSK - *Text Mining Software Kit*

UML - *Unified Modeling Language*

URI - *Uniform Resource Identifier*

URL - *Uniform Resource Locator*

XML - *eXtensive Markup Language*

W3C - *World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO

O avanço das tecnologias para aquisição e armazenamento de dados tem permitido que o volume de informação em formato digital aumente de forma significativa nas organizações. Estima-se que no período de 2003 a 2010 a quantidade de informação no universo digital passou de cinco hexabytes (aproximadamente cinco bilhões de gigabytes) para 988 hexabytes. Até o ano de 2008 contabilizou-se que a humanidade produziu cerca de 487 hexabytes de informação digital (MARCACINI; MOURA; REZENDE, 2011, p. 7).

Cerca de 80% destes dados estão em formato não estruturado, dos quais uma parte significativa são textos (KUECHLER, 2007, p. 86). Estes textos são considerados como dados brutos, pois não possuem estrutura definida. Constituem um importante repositório organizacional, que envolve o registro de histórico de atividades, memorandos, documentos internos, *e-mails*, projetos, estratégias e o próprio conhecimento adquirido (HAN; KAMBER, 2006, p. 5). Segundo Marcacini, Moura e Rezende (2011, p. 7), "A organização inteligente dessas coleções textuais é de grande interesse para a maioria das organizações, pois agiliza processos de busca e recuperação da informação".

A transformação de dados brutos e informação em conhecimento facilita sua análise e compreensão pelo gestor do conhecimento da organização. Esta transformação é uma reestruturação cognitiva dos dados, dependente da manipulação e aplicação de informações, ante uma atividade desenvolvida por um indivíduo (FIALHO et al., 2010, p. 19).

Na sociedade pós-industrial a transformação de dados em conhecimento é uma atividade importante para promoção do crescimento e da tomada de decisões eficazes nas organizações. Os bens e serviços que são produzidos e consumidos estão tornando-se cada vez mais intensivos em tecnologia e conhecimento. A competição é cada vez mais baseada na capacidade de transformar informação em conhecimento e conhecimento em decisões e ações de negócio (FIALHO et al., 2010, p. 53-54). Segundo Tarapanoff (2006, p. 26), esta competição embasada pelo conhecimento é chamada de inteligência competitiva e pode ser definida como um processo de aprendizado fundado sobre a informação, permitindo a otimização da estratégia corporativa em curto e longo prazo. Esse processo, para ser implementado, requer contínua utilização de dados e informações, sendo que no processo de análise, agregação de valor e criação a partir dos mesmos é utilizado o conhecimento individual do gestor do conhecimento.

Conforme destacado anteriormente, o volume de dados textuais produzidos e armazenados é tal que extrapola a capacidade humana de, manualmente, analisá-los e

compreendê-los por completo. Assim sendo, busca-se criar soluções para automatizar este processo, diminuindo a intervenção humana para a descoberta (também conhecida como extração) do conhecimento (MARCACINI; MOURA; REZENDE, 2011, p. 7). Um método utilizado para esse propósito é a Mineração de Texto (MT) definida como um conjunto de técnicas usadas para navegar, organizar e descobrir conhecimento em bases de texto. Camilo e Silva (2009, p. 11-19) descrevem algumas técnicas da MT para descobrir conhecimento, como árvores de decisão, classificação bayesiana, classificação baseada em regras, redes neurais, aprendizado tardio, algoritmos genéticos, conjuntos aproximados e conjuntos nebulosos.

A MT permite a transformação de um grande volume de dados textuais não estruturados em conhecimento útil, muitas vezes inovador para as organizações. O seu uso permite descobrir conhecimento a partir de dados textuais brutos (não estruturados), fornecendo elementos de suporte à gestão do conhecimento, que se refere ao modo de reorganizar como o conhecimento é criado, usado, compartilhado, armazenado e avaliado. Tecnologicamente, o apoio da MT à gestão do conhecimento se dá na transformação do conteúdo de repositórios de informação em conhecimento a ser analisado e compartilhado pela organização (MARCACINI; MOURA; REZENDE, 2011, p. 7).

Aranha et al. (2004, p. 105) afirmam que o Processamento de Linguagem Natural (PLN) é uma técnica muito importante para apoiar a MT. Utilizando conhecimento da área de linguística, o PLN permite aproveitar ao máximo o conteúdo do texto, extraindo entidades e seus relacionamentos, detectando sinônimos, corrigindo palavras escritas de forma errada e ainda desambiguizando-as.

A simples descoberta do conhecimento não resolve o problema da organização, sendo necessário estruturá-lo de forma que ele possa ser utilizado de forma eficiente, formando assim uma base de conhecimento. Esta estruturação é conhecida como Representação do Conhecimento (RC) que pode ser definida como um conjunto de sentenças em uma linguagem formal, para a qual foram definidas uma semântica e um conjunto de regras de inferência capazes de gerar novas sentenças a partir das sentenças disponíveis (HEINZLE, 2011, p. 93).

Um exemplo de formalismo com o propósito de representar o conhecimento é uma ontologia escrita com a linguagem *Web Ontology Language* (OWL). Uma ontologia escrita em OWL define formalmente um conjunto comum de termos que são usados para descrever e representar um domínio específico. Ela faz uso de alguns mecanismos para representar o conhecimento, como classes e relacionamentos. Assim, a OWL pode ser usada por

ferramentas automatizadas para melhorar os serviços avançados como buscas na *web* e gerenciamento de conhecimento (HEFLIN, 2004).

Um fator que mostra a importância da RC através das ontologias OWL é que a *World Wide Web Consortium* (W3C) definiu este formalismo como padrão para a criação de ontologias para a *web* (W3C, 2004). Um especialista em determinado domínio do conhecimento pode criar sua ontologia e compartilhá-la com terceiros, disseminando assim o conhecimento representado pela mesma.

Diante do exposto, este trabalho visa fornecer uma alternativa computacional que descubra e formalize o conhecimento de forma automática para o gestor do conhecimento. Este conhecimento dar-se-á na forma de uma ontologia OWL, com suas classes e relações. Este processo é útil ao gestor do conhecimento, pois muitas vezes as bases textuais são intratáveis manualmente devido ao seu tamanho demasiado, necessitando de uma solução automatizada.

1.1 OBJETIVOS DO TRABALHO

Este trabalho tem como objetivo disponibilizar uma ferramenta para a criação de bases de conhecimento na forma de uma ontologia OWL a partir de textos não estruturados.

Os objetivos específicos do trabalho são:

a) possibilitar a visualização dos documentos que compõem a base textual;
b) disponibilizar a estrutura morfológica da base textual;
c) apresentar as estruturas ontológicas a partir do conhecimento descoberto.

1.2 ESTRUTURA DO TRABALHO

O trabalho está organizado em quatro capítulos: introdução, fundamentação teórica, desenvolvimento e conclusão. No capítulo um foi apresentada a contextualização, justificativa e objetivos. No capítulo dois são elencados os aspectos teóricos estudados para o desenvolvimento do presente trabalho e, ao final, são apresentados os trabalhos correlatos e um quadro comparando suas características. O capítulo três apresenta o desenvolvimento da ferramenta, tratando temas como o detalhamento dos requisitos, a especificação e implementação, bem como resultados e discussões. No último capítulo são abordadas as conclusões, juntamente com sugestões de extensões para a ferramenta.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo descrever os principais assuntos estudados para o desenvolvimento da ferramenta. A seção 2.1 explana sobre era da informação e do conhecimento. Na seção 2.2 é discursado sobre RC. A seção 2.3 descreve a descoberta de conhecimento. A seção 2.4 descreve o PLN. Por fim, na seção 2.5 são descritos os trabalhos correlatos e um quadro comparando suas características.

2.1 ERA DA INFORMAÇÃO E DO CONHECIMENTO

A transição para o novo milênio foi marcada por mudanças rápidas e de grande impacto na sociedade, resultando no que se convencionou denominar de era da informação e do conhecimento. Neste novo ambiente, as facilidades proporcionadas pelas tecnologias da informação e da comunicação permitem que uma pessoa possa se comunicar ou acessar informações de qualquer outro lugar do planeta (HEINZLE, 2011, p. 49). Fialho et al. (2010, p. 53) compartilham deste entendimento e acrescentam que este fenômeno se deu a partir de 1980, com o advento da globalização.

Segundo Tarapanoff (2006, p. 215), na sociedade atual, quem deseja escrever sobre inteligência organizacional deve primeiramente escrever sobre conhecimento. Quem deseja abordar sobre conhecimento é levado a introduzir conceitos acerca de informação e tudo recai inevitavelmente sobre dados. Este fenômeno ocorre pelo fato dos dados brutos serem a matéria prima do conhecimento.

Dado é a representação simbólica de um objeto ou informação do domínio, sem considerações de contexto, significado ou aplicação. São sinais interpretados como bits em um computador, podendo ser entendidos como matéria-prima básica da informação e do conhecimento (MASTELLA, 2004, p. 15). Para Fialho et al. (2006, p. 71) dados são sinais desprovidos de interpretação. Os autores enfatizam que, a partir do momento que os dados são contextualizados, passam a ser considerados informação.

Informação é o reconhecimento dos objetivos do domínio, suas características, suas restrições e seus objetivos com outros objetos. É o dado com seu significado associado disposto em uma estrutura específica (MASTELLA, 2004, p. 15). A informação é um meio ou material necessário para extrair e construir o conhecimento, alterando-o. É vista de duas perspectivas: sintática (volume de informações) e semântica (significado). Esta última é a mais importante para a criação do conhecimento, pois envolve o significado transmitido, uma vez que o conhecimento está essencialmente relacionado com a ação humana (FIALHO et al., 2010, p. 54).

Conhecimento é o resultado da interpretação da informação e de sua utilização para gerar novas ideias, resolver problemas ou tomar decisões. Conhecimento inclui a informação sobre o domínio e a forma como esta informação é utilizada para resolver problemas, ou seja, pode ser descrito como tudo o que se usa para agir e criar novas informações (MASTELLA, 2004, p. 15). Fialho et al. (2010, p. 40) descrevem conhecimento como um conjunto completo de dados, informações e relações que levam pessoas a tomar decisões, desempenhar atividades e a criar novas informações ou conhecimentos. Segundo os autores, o conhecimento também pode ser conceituado como um conjunto de informações contextualizadas e dotadas de semânticas inerentes ao agente que o detém.

Para Fialho et al. (2006, p. 72) uma das principais causas da dificuldade de se especificar o que é conhecimento está no fato dele ser muito dependente do contexto. Segundo os autores, "a perspectiva com que se interpreta o conhecimento é importante na medida em que vai determinar a forma como a gestão em uma empresa é abordada".

Após contextualizar dado, informação e conhecimento, pode-se descrever o processo de conversão de dados em conhecimento. Segundo Fialho et al. (2006, p. 72), este processo se inicia no grande volume de dados disponíveis nas organizações sofrendo intervenção humana e tecnológica, recebendo assim alguma carga semântica, transformando-se em informação. Ainda segundo os autores, essa informação será submetida a uma análise humana e tecnológica mais acentuada que a etapa anterior, contextualizando as informações e criando ligações entre elas, gerando assim conhecimento.

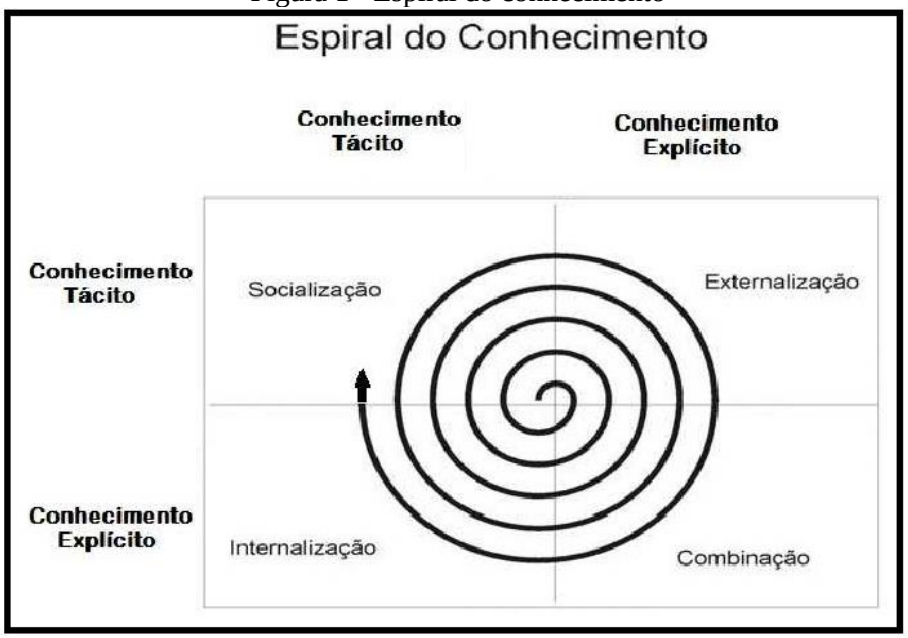
Segundo Nonaka e Takeuchi (1997, p. 15), o conhecimento é classificado em dois tipos:

a) tácito: é um tipo de conhecimento muito difícil de se expressar com palavras e é adquirido com a experiência, de maneira prática utilizando-se da intuição e da subjetividade. Envolve fatores intangíveis como crenças pessoais, perspectivas, sistemas de valor, intuições, emoções e habilidades individuais. Fialho et al. (2010, p. 44) partilham do mesmo entendimento, ao definir o conhecimento tácito como procedural, pessoal, específico ao contexto, difícil de ser formulado e comunicado;
b) explícito: é um tipo de conhecimento que pode ser expresso em palavras ou números e pode ser transmitido formal e sistematicamente entre pessoas. Envolve o conhecimento de fatos. É objetivo, teórico e articulado por meio da linguagem formal, com afirmações gramaticais, expressões matemáticas, manuais, entre outras. Segundo Fialho et al. (2010, p. 44), o conhecimento explícito permite o saber sobre determinados fatos e eventos.

O novo conhecimento começa sempre com o indivíduo. Um pesquisador tem uma ideia que leva a uma nova patente. Um operário extrai de anos de experiência uma inovação em um processo novo. Em cada caso, o conhecimento pessoal de um indivíduo é transformado em conhecimento organizacional para a empresa como um todo (NONAKA; TAKEUCHI, 2008, p. 41).

A facilidade com que uma organização alavanca o conhecimento está na razão direta da eficiência da conversão do conhecimento tácito em explícito. A simples combinação das informações explícitas não amplia a base de conhecimentos existentes na empresa. A interação contínua e dinâmica entre conhecimento explícito e tácito é que gera inovação (FIALHO et al., 2010, p. 48). Nonaka e Takeuchi (2008, p. 45) descrevem que converter conhecimento tácito em explícito significa encontrar uma forma de expressar o inexpressável. Segundo Nonaka e Takeuchi (1997, p. 18), o conhecimento tácito criado e acumulado em nível individual deve ser mobilizado e ampliado organizacionalmente pelos quatro modos de conversão, constituindo a chamada espiral do conhecimento, representada na Figura 1.

Figura 1 - Espiral do conhecimento



Fonte: Nonaka e Takeuchi (1997, p. 20).

Nonaka e Takeuchi (2008, p. 60) descrevem socialização como a conversão do conhecimento tácito em conhecimento tácito. Segundo os autores, é um processo de compartilhamento de experiências e, com isso, de criação de conhecimento tácito. Os autores ainda afirmam que o indivíduo pode adquirir conhecimento tácito diretamente dos outros sem usar a linguagem, como os aprendizes que trabalham com seus mestres e aprendem sua arte não através da linguagem, mas da observação, da imitação e da prática. Segundo Fialho et al.

(2006, p. 111), a mera transferência de informações muitas vezes faz pouco sentido se estiver desligada das emoções associadas a ela e dos contextos específicos nos quais as experiências compartilhadas são embutidas. Os autores ainda ressaltam o fato de que o segredo para a aquisição de conhecimento tácito é a experiência compartilhada. Eles concluem que sem a experiência compartilhada é extremamente difícil para uma pessoa projetar-se no processo de raciocínio de outro indivíduo.

Para Fialho et al. (2010, p. 47), externalização é o processo de articulação do conhecimento tácito em conhecimento explícito. Segundo os autores, é um processo de criação do conhecimento perfeito, considerando que a forma explícita é expressa por metáforas, analogias, conceitos, hipóteses ou modelos. Os autores citam a escrita como uma forma de converter o conhecimento tácito em conhecimento articulável. Segundo Fialho et al. (2006, p. 112), a explicitação da conversão do conhecimento é provocada pelo diálogo ou pela reflexão coletiva. É também comum a criação de conceitos por meio da combinação de conceitos de dedução e indução. Para os autores a explicitação é a chave para a criação de conhecimento, tendo em vista que cria conceitos novos e explícitos, a partir do conhecimento tácito.

Nonaka e Takeuchi (2008, p. 65) descrevem combinação como a conversão do conhecimento explícito em conhecimento explícito. Segundo os autores é um processo de sistematização de conceitos em um sistema de conhecimento. Este modo de conversão do conhecimento envolve a combinação de diferentes corpos de conhecimento explícito. Segundo Fialho et al. (2006, p. 112), é nesse ponto do processo de criação do conhecimento que surgiram os primeiros protótipos e modelos reais. Dessa forma, a combinação de um novo conhecimento explícito com informações e conhecimentos existentes geram e sistematizam o conhecimento explícito por toda a organização. Para os autores, os indivíduos trocam e combinam o conhecimento através de meios como documentos, reuniões, conversas telefônicas ou redes de comunicação computadorizadas.

Segundo Nonaka e Takeuchi (2008, p. 67), a internalização é o processo de incorporação do conhecimento explícito em conhecimento tácito. Está intimamente ligada ao "aprender fazendo". Ainda segundo os autores, quando as experiências através da socialização, externalização e combinação são internalizadas nas bases de conhecimento tácito do indivíduo, na forma de mapas mentais compartilhados, tornam-se um patrimônio valioso. Segundo Fialho et al. (2010, p. 47), a verbalização e diagramação do conhecimento sob a forma de documentos, manuais, ou histórias orais são fundamentais para que o

conhecimento explícito torne-se tácito. Para os autores, a documentação tem importante papel na internalização das experiências nos indivíduos.

2.2 REPRESENTAÇÃO DO CONHECIMENTO

Existem múltiplos formalismos e ferramentas para representar computacionalmente o conhecimento. Estes, originários da área da IA, tem como desafio central a construção de mecanismos computacionais capazes de simular, na máquina, algumas características da inteligência humana. Segundo Heinzle (2011, p.93), "no caso dos sistemas de informação, especificamente, buscam-se na IA, tecnologias que possam suprí-los com conhecimento formalizado, capaz de ser interpretado igualmente por homens e máquinas. São mecanismos que permitem representar o conhecimento e sobre ele raciocinar".

Esta formalização computacional do conhecimento é chamada de Representação do Conhecimento (RC). Heinzle (2011, p. 93) descreve a RC como sendo os métodos formais usados para modelar o conhecimento relacionado a um certo domínio de problema, através dos quais se busca o desenvolvimento de sistemas computacionais com certa faculdade de raciocínio sobre esta representação. Ainda segundo o autor, trata-se de um conjunto de convenções sintáticas e semânticas, que torna possível descrever um mapeamento entre os objetos e as relações envolvidas neste domínio. Segundo Araújo (2003, p. 50), representar conhecimento é o ato de documentar ou expressar, pela linguagem simbólica, textual ou algorítmica, os fatos e as ações, de modo que possam ser corretamente interpretados e reconstruídos por uma outra entidade.

Ladeira (1997, p. 26) classifica as abordagens para a RC em 3 categorias: declarativa (ou lógica), procedimental e estrutural. No sistema de representação declarativo o conhecimento é descrito com base em fatos, postulados como verdadeiros, sobre o domínio da aplicação e por um conjunto de procedimentos para manipulá-los. A base de conhecimento é vista como uma teoria ou um conjunto de teoremas numa teoria. Este sistema possui semântica bem definida e disponibilidade de uma teoria de demonstração, sendo adequados para representar porções de conhecimento passivas do domínio e em situações onde o raciocínio (monotônico) tenha importância. No sistema de representação procedimental o conhecimento é descrito através de procedimentos para a sua utilização, sendo adequado para a representação do conhecimento comportamental da aplicação. Sua base de conhecimento é representada como um conjunto de funções ou procedimentos como, por exemplo, regras ou árvores de decisão. Por fim, no sistema de representação estrutural, foco do presente trabalho, o conhecimento é representado com base em objetos e relações sobre as entidades a serem

modeladas. São utilizados conceitos, hierarquias de estruturas, descrições de classes de elementos e instâncias individuais ou componentes de objetos. Sua base de conhecimento é entendida como uma coleção de objetos e suas relações.

Ladeira (1997, p. 18) afirma que para resolver problemas, os programas de IA precisam de conhecimento e mecanismos para sua manipulação e que o conhecimento deve ser representado de uma forma que o programa possa manipulá-lo. Para Heinzle (2011, p. 94), representar conhecimento envolve fundamentalmente encontrar estruturas capazes de expressar o conhecimento do domínio de uma aplicação adequadas de tal forma que sobre elas possa ser realizado o raciocínio computacional por meio de mecanismos de inferência.

Diferente dos dados que são armazenados em bancos de dados, o conhecimento necessita de estruturas mais complexas para armazená-lo. Estas estruturas são chamadas de bases de conhecimento. Segundo Figueredo (2002, p. 2), bases de conhecimento são *containers* de conhecimento e informação, concebidas para armazenar, compartilhar e disseminar conhecimentos específicos. Elas resolvem questões de volume de armazenamento, organização, recuperação e herança do conhecimento. Também mantém e estimulam a colaboração e o compartilhamento de conhecimentos e informações relevantes dentro da organização. Bases de conhecimento são criadas fazendo-se uso de algum formalismo de RC, entre os quais destaca-se a ontologia.

2.2.1 Ontologia

A palavra ontologia é de origem grega e significa *ontos* (ser) + *logos* (conhecimento sobre), tendo sido criada entre os séculos XVII e XVIII por filósofos alemães para denominar o ramo da filosofia que trata da natureza e organização do ser. Muito antes, entretanto, Aristóteles já havia desenvolvido os primeiros estudos filosóficos do ser, relacionados à classificação dos seres vivos então conhecidos. A palavra foi posteriormente popularizada e difundida no meio filosófico por Christian Wolff, que a empregou em suas publicações, como no caso do título do trabalho "*Philosophia prima sive ontologia methodo scientifica pertractata, qua omnes cognitiones humanae principia continentur*," publicado em 1730 (MOREIRA; ALVARENGA; OLIVEIRA, 2004).

A ontologia, como subárea da filosofia estuda os seres, tratando da sua natureza, da realidade, da existência dos entes e também das questões metafísicas. Segundo Souza (2003, p. 13), é a "ciência do que é, das coisas e das estruturas dos objetos, das propriedades e das relações em toda área da realidade". Heinzle (2011, p. 106) aponta que uma ontologia trata do ser enquanto ser e objetiva compreender identidades, podendo ser vista como uma sistemática

descrição da existência. Para a filosofia, cada campo científico deve ter sua própria ontologia, definida pelo vocabulário do campo e pelas formulações de suas teorias.

Heinzle (2011, p. 106) descreve que, a partir do início da década de 90 do século passado, o termo foi adotado pelas áreas da Engenharia do Conhecimento, dos Sistemas de Informação e da Ciência da Computação, onde recebeu adaptações em sua definição e passou a ter outra interpretação. Gruber (1993, p. 210) afirma que uma ontologia é uma especificação formal, explícita e compartilhada de uma conceitualização. O autor esclarece a terminologia empregada. Para ele conceitualização indica que se trata de um modelo abstrato que representa algum fenômeno ou objeto do mundo real. Formal implica que pode ser compreendida por uma máquina. Explícita indica que os conceitos, suas restrições e relações precisam ser explicitamente definidas e compartilhadas.

Para Chandrasekaran e Josephson (1999, p. 22), o termo ontologia na Engenharia do Conhecimento e na Ciência da Computação, refere-se à representação de um vocabulário relacionado a um certo domínio, onde a qualificação não está no vocabulário mas sim nos conceitos expostos por ele. Adicionalmente, o autor afirma que uma ontologia pode também referir-se a um conjunto de conhecimentos que descreve algum domínio usando um vocabulário representativo. Segundo Novello (2003), a ontologia é descrita como um vocabulário de uma área que define, com diferentes níveis de formalismo, os significados de termos e dos relacionamentos entre eles. Para a autora, as ontologias objetivam capturar o conhecimento declarativo do domínio e fornecer uma compreensão deste, possibilitando o reuso e compartilhamento através das aplicações.

Alguns pesquisadores, embora com entendimento semelhante, dão ênfase à diferença da interpretação de ontologia nas áreas de filosofia e engenharia/computação para instrumentar seus argumentos. Segundo Guarino (1997, p. 300), no sentido filosófico pode-se referir a uma ontologia como um sistema particular de categorias que versa sobre uma certa visão do mundo. Desta forma, este sistema não depende de uma linguagem particular: a ontologia de Aristóteles é sempre a mesma, independente da linguagem usada para descrevê-la. Por outro lado, em seu uso mais prevalecente na IA, uma ontologia é referida como um artefato de engenharia, constituído de um vocabulário específico usado para descrever uma certa realidade e um conjunto de pressupostos explícitos relacionados com o significado pretendido para as palavras do vocabulário.

No contexto de engenharia, uma ontologia é descrita como um artefato concreto no nível simbólico que, assim, pode ser compartilhada e transmitida. Ela é um artefato que descreve conceitos, relações, restrições e axiomas de um domínio, usando uma organização

taxonômica baseada em generalização e especialização (HEINZLE, 2011, p. 107). Noy e Hafner (1997, p. 60) têm um entendimento semelhante. Eles descrevem a ontologia como um artefato de engenharia, constituído de um vocabulário de termos organizados em uma taxonomia, suas definições e um conjunto de axiomas formais usados para criar novas relações e para restringir as suas interpretações segundo um sentido pretendido.

Heinzle (2011, p. 107) explana sobre os componentes básicos das ontologias. Segundo ele, 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. As ligações entre estes conceitos ocorrem com base nos 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, sendo restrições. As instâncias representam os elementos ou objetos da ontologia, ou seja, são exemplares individuais das classes.

Supondo a criação de uma ontologia sobre uma concessionária de carros, duas classes que podem ser identificadas são `Comprador` e `Carro`. A classe `Comprador` é usada para modelar as características de uma pessoa que comprou um carro. A classe `Carro` modela um carro. Tem-se como exemplo o relacionamento `serDono`, existente entre as classes `Comprador` e `Carro`, onde uma pessoa será dona de um carro. Afirmar que todo `Comprador` tem um pai e uma mãe pode ser considerado um axioma, pelo fato disso ser uma regra que sempre é verdadeira. Por fim, nesse domínio, existem os objetos. Um `Fusca` é um objeto da classe `Carro`, pois o mesmo representa um carro do mundo real, assim como `João` é um objeto da classe `Comprador`.

Os interesses da Engenharia do Conhecimento e da Ciência da Computação no uso de ontologias estão relacionados ao desenvolvimento que envolvem o raciocínio computacional automático (viável através do formalismo declarativo da especificação), a interoperabilidade e reuso entre sistemas e agentes (garantido pela exigência de que o conhecimento representado deve ser formal e compartilhado) e suporte à integração e gestão do conhecimento, através de unificação e explicitação de termos, conceitos, categorias e relações relativas ao domínio e presentes na ontologia (eliminando ambiguidades) (HEINZLE, 2011, p. 108).

2.2.1.1 Linguagens de Representação

Segundo Heinzle (2011, p. 105), para construir ontologias existem algumas linguagens formais específicas. O autor também explana que elas distinguem-se pelas facilidades, expressividades e propriedades computacionais que oferecem, sendo que as principais são: *Resource Description Framework* (RDF), *Resource Description Framework Schema* (RDFS), *Ontology Inference Layer* (OIL), DAML+OIL e OWL.

RDF é um modelo padrão para troca de dados na *web*. Seu primeiro molde e especificação de sintaxe foram propostos em fevereiro de 1999 pela W3C. Atualmente, o RDF faz parte de um conjunto de recomendações publicado em 2004. O modelo mais simples de representação é um grafo dirigido rotulado, no qual dois nós representam os recursos e a aresta a propriedade que os relaciona (RDF WORKING GROUP, 2004).

São enumerados em Daconta, Obrst e Smith (2003, p. 149) algumas limitações da linguagem RDF. A primeira limitação é de que partes do RDF são complexas, por existirem representações de dados de várias comunidades diferentes, existindo muita flexibilidade na representação dos modelos, sendo sua árvore de difícil interpretação sem auxílio de ferramentas. Outra limitação é que os exemplos em RDF são fracos, não ilustrando de maneira satisfatória a capacidade da linguagem. Breitman (2005, p. 51) afirma que RDF fornece uma boa representação para o tratamento de metadados, mas não oferece os subsídios necessários para uma linguagem de ontologias. Assim surge a RDFS como uma linguagem de descrição de vocabulários, através de classes e propriedades para recursos RDF. Classes e propriedades em RDFS são similares ao conceito de classes e atributos em programação orientada a objetos.

O RDFS estende o RDF ao adicionar os conceitos de classe e hierarquia, permitindo a criação de ontologias simples. Ele considera que os recursos RDF formam classes e fornece instrumentos para criar modelos e representar sistemas hierárquicos destas classes, formando taxonomias. Adicionalmente, existem mecanismos para expressar relacionamentos entre as classes e suas instâncias, através dos quais é descrita a semântica e as características de um domínio de conhecimento e sobre o qual é possível a realização de inferências lógicas. Assim, cria-se um sistema de classes genérico, extensível e compartilhável que pode ser utilizado para esquemas num domínio específico (HEINZLE, 2011, p. 111).

A linguagem OIL nasceu do projeto On-To-Knowledge que foi patrocinado pelo programa *Information Society Tecnologies* (IST), um consórcio da comunidade europeia. Ela foi concebida para ampliar a capacidade semântica dos formalismos da linguagem RDFS e,

por isso, mantém compatibilidade com o padrão RDF. Os componentes principais da linguagem OIL são as classes, os relacionamentos e as instâncias. Entre as classes existe uma relação de hierarquia, além de outras relações binárias. Sobre estas relações podem ser atribuídas restrições de cardinalidade. Uma ontologia OIL é composta de dois elementos fundamentais: o *ontology container* e o *ontology definitions*. O primeiro contém as características da própria ontologia e no segundo está a definição do vocabulário da referida ontologia (HEINZLE, 2011, p. 112).

No início de 2001 foi lançada uma nova versão denominada de DAML+OIL, criada com o objetivo de combinar as propostas das linguagens DAML e OIL (BREITMAN, 2005, p. 52). Uma ontologia escrita em DAML+OIL é composta de cabeçalhos, classes, propriedades e instâncias. No cabeçalho estão informações relacionadas à versão, data, e referências a outras ontologias. Estas outras ontologias são referenciadas pela expressão *imports*, que faz referência a uma *Uniform Resource Identifier* (URI), que corresponde ao local onde a ontologia será importada. As declarações *imports* são tratadas como transitivas. Assim, se uma ontologia importar uma segunda e se esta segunda fizer referência a uma terceira através de uma cláusula *imports*, também a terceira fará parte da ontologia em questão. Quanto às classes, cada expressão *class* associa uma classe à sua definição e a relaciona com uma URI. Duas classes já estão predefinidas, a `daml:Thing` e a `daml:Nothing`. Por definição, todo objeto é membro de `daml:Thing` e não é membro de `daml:Nothing`, assim todas as classes são subclasses de `daml:Thing` e `daml:Nothing` é uma subclasse de todas as classes (HEINZLE, 2011, p. 113).

A OWL é uma linguagem para ontologias que integra as tecnologias recomendadas pelo consórcio W3C desde fevereiro de 2004 (W3C, 2004). É baseada na sintaxe do *eXtensive Markup Language* (XML) e RDFS, tratando-se de uma revisão da linguagem DAML+OIL. Segundo Heinzle (2011, p. 115), a OWL supera suas predecessoras ao disponibilizar instrumentos adicionais para explicitar o significado de termos e relacionamentos, porém mantendo em comum com elas a sintaxe e a lógica formal descritiva como mecanismo para expressar semântica por axiomas lógicos. Segundo a W3C (2004), a linguagem foi projetada para aplicações que necessitam processar o conteúdo das informações e não apenas apresentar informações. Seu uso é indicado quando se pretende:

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

Segundo Breitman (2005, p. 56), existem 3 versões ou sub-linguagens da OWL denominadas de Lite, DL e Full. Elas crescem gradativamente o nível ou capacidade de expressividade, sendo a primeira um subconjunto da segunda, que é por sua vez um subconjunto da terceira. Heinzle (2011, p. 116) registra esta mesma classificação, conceituando-as. Segundo o autor a OWL-Lite oferece um suporte mais básico e é indicada para aplicações que são caracterizadas por envolverem uma classificação hierárquica e que possuem definições simples de propriedades e restrições. Ela oferece restrições de cardinalidade para propriedades apenas com valores zero ou um. A OWL-DL disponibiliza construções adicionais que permitem mais expressividade, mas é ainda indicada para situações com poucas restrições envolvidas no modelo, em especial quanto aos tipos. A denominação DL deve-se à lógica de descrição, base formal da linguagem. Finalmente, a OWL-Full é a versão que permite máxima expressividade, com maior liberdade em relação ao RDF e à lógica formal. Ao aumentar o vocabulário RDF, ela pode levar a construção de ontologias com maior poder de expressividade, mas sobre a qual se tem menor capacidade de inferência computacional.

Os elementos básicos que compõem uma ontologia escrita em OWL são as classes, as propriedades, as instâncias destas classes e o relacionamento entre as instâncias. A classe é o conceito mais básico para descrever um domínio de aplicação de uma ontologia, pois trata-se do mecanismo de abstração para agrupar entidades com características afins. Por meio dela constrói-se uma árvore, que corresponde a taxonomia implícita na ontologia e na qual uma subclasse herda as propriedades da classe à qual está subordinada. A raiz desta árvore é a classe `owl:Thing`, que já é predefinida e da qual todas as outras, criadas no processo de modelagem do conhecimento, serão subclasses. Aos indivíduos de uma classe é dada a denominação de instâncias ou objetos (SMITH; WELTY; MCGUINNES, 2004).

As relações são descritas como propriedades binárias que descrevem características e relacionamentos entre classes ou indivíduos por meio dos quais é possível afirmar fatos sobre eles. Em OWL existem 2 categorias de propriedades: *Datatype Property* (associa indivíduos a valores de dados) e *Object Property* (empregada para associar classes ou indivíduos). Relacionadas às propriedades existem declarações para impor a elas restrições e caracterizações. *Range* (escopo) e *Domain* (domínio) são os elementos da linguagem para definir os conjuntos de elementos válidos envolvidos numa relação. O escopo é a classe que a propriedade parte e o domínio é a classe em que chega à propriedade. As propriedades conectam indivíduos de um escopo a indivíduos de um domínio (HEINZLE, 2011, p. 118).

No exemplo da seção 2.2.1 sobre a criação de uma ontologia de concessionária de carros, identifica-se, além das classes, suas propriedades. A propriedade `serDono` é uma *Object Property*. Ela relaciona as classes `Carro` e `Comprador`. Essa propriedade possui um domínio e um escopo. O escopo é a classe `Comprador` e o domínio é a classe `Carro`. Na classe `Comprador` existem algumas *Datatype Property*, representando os atributos de um comprador de carro. Podem ser citados `nome`, `telefone` e `endereço`. Estas propriedades são ligadas a algum dos tipos de dados descritos em RDF Working Group (2004).

A estrutura de uma ontologia OWL é composta de declarações de *namespace*, um cabeçalho opcional, as definições de classes, de propriedades e de indivíduos. O *namespace* é identificado por uma *Internationalized Resource Identifier* (IRI) e tem a finalidade de evitar ambiguidades no vocabulário. As informações relativas à própria ontologia recebem a denominação de *headers* (ou cabeçalhos). A estrutura padrão de uma ontologia OWL escrita com a ferramenta Protégé (PROTÉGÉ, 2013) é composta por duas *tags*: `<Ontology>` e `</Ontology>`. A primeira *tag* indica o início da ontologia, enquanto a segunda a finaliza. Após isso, a ontologia é descrita com um conjunto de declarações, cada qual indicando um ferramental utilizado para representar o conhecimento (classe ou relacionamento). As declarações são representadas através de blocos contidos entre as *tags* `<Declaration>` e `</Declaration>`. Um ferramental para se representar o conhecimento em uma ontologia OWL são as classes. Para se definir uma classe dentro de um arquivo OWL utiliza-se a *tag* `<Class IRI="#nomeDaClasse"/>`, onde `nomeDaClasse` é definida como uma IRI com o nome que se deseja denominar a classe (HEINZLE, 2011, p. 116).

Para Heinzle (2011, p. 116), outro ferramental utilizado para representar o conhecimento é a *Object Property*, que denota as relações entre classes. Para descrevê-la faz-se uso da *tag* `<ObjectProperty IRI="#nomeDaPropriedade"/>`, onde o `nomeDaPropriedade` é denotado através de uma IRI. A *Object Property* contém um domínio e um escopo. A definição do domínio ocorre colocando uma *Object Property* e uma classe entre as *tags* `<ObjectPropertyDomain>` e `</ObjectPropertyDomain>`. A definição de um escopo é envolta pelas *tags* `<ObjectPropertyRange>` e `</ObjectPropertyRange>`. No Quadro 1 é mostrada uma ontologia OWL escrita no Protégé. Nas linhas 3 e 6 são criadas as classes `bela` e `principe` respectivamente. Na linha 9 é criada a *Object Property* `matou`. O código entre as linhas 11 e 14 atribuem a classe `bela` como domínio da propriedade `matou` e entre as linhas 15 e 18 atribuem a classe `principe` como escopo da propriedade `matou`.

Quadro 1 - Estrutura de um arquivo OWL escrito com o Protégé

```
1  <Ontology>
2       <Declaration>
3            <Class IRI="#bela"/>
4       </Declaration>
5       <Declaration>
6            <Class IRI="#principe"/>
7       </Declaration>
8       <Declaration>
9            <ObjectProperty IRI="#matou"/>
10      </Declaration>
11      <ObjectPropertyDomain>
12           <ObjectProperty IRI="#matou"/>
13           <Class IRI="#bela"/>
14      </ObjectPropertyDomain>
15      <ObjectPropertyRange>
16           <ObjectProperty IRI="#matou"/>
17           <Class IRI="#principe"/>
18      </ObjectPropertyRange>
19 </Ontology>
```

2.3 DESCOBERTA DE CONHECIMENTO

Atualmente as organizações possuem grandes bases de dados e conglomerados de documentos textuais. Neles residem muitas informações que ainda não foram descobertas pelos gestores de conhecimento, tendo potencial para serem transformadas em conhecimento e servirem como vantagem competitiva. Segundo Wives (2000), descobrir conhecimento significa identificar, receber informações relevantes e poder processá-las e agregá-las ao conhecimento prévio de seu usuário, mudando o estado de seu conhecimento atual, a fim de que determinada situação ou problema possa ser resolvida. Neste sentido, observa-se que o processo de descoberta de conhecimento está fortemente relacionado à forma pela qual a informação é processada.

Tradicionalmente, o processo de análise de dados para a identificação de informações úteis baseia-se em trabalho manual. Por exemplo, especialistas em concessão de crédito de um banco podem periodicamente verificar determinadas estatísticas nos dados, como a média de idade dos clientes que conseguiram e dos que não conseguiram pagar um empréstimo, a inadimplência no período atual e no mesmo período dos anos anteriores, entre outras. A partir dessas análises, os especialistas podem gerar relatórios que serão utilizados para tomar futuras decisões de concessão de crédito para novos clientes (BATISTA, 2003, p. 24).

Segundo Baranauskas (2001, p. 130), o método clássico de análise de dados reside em um ou mais analistas humanos, familiares com os dados e atuando como uma interface entre os dados e os usuários. O autor ainda enfatiza que a forma manual de análise de dados é lenta, cara e altamente subjetiva. Batista (2003, p. 24) afirma que é comum o fato de que os dados residam em uma base de dados digital, facilitando a extração de muitos relatórios e consultas. Ainda segundo o autor, o analista humano deve comparar relatórios, cruzar informações e

utilizar seu conhecimento prévio sobre a área de atuação para tentar extrair algo novo dos dados existentes.

Na medida em que as bases de dados crescem cada vez mais, a abordagem manual torna-se impraticável em vários domínios, principalmente na descoberta de informações úteis. Bases de dados contendo registros na ordem de 10^9 estão se tornando cada vez mais comuns. De forma similar, o número de atributos dos registros pode atingir a ordem de 10^2 ou 10^3. É muito improvável que um ser humano consiga analisar e inferir informações em bases de dados com essas dimensões (BARANAUSKAS, 2001, p. 23).

Existe, portanto, a necessidade de uma nova geração de técnicas e ferramentas com a habilidade de assistir os analistas humanos de uma forma inteligente e automática na procura de conhecimentos previamente desconhecidos nos dados. Tais técnicas e ferramentas são objetos de estudo de uma área de pesquisa chamada de Descoberta de Conhecimento em Bancos de Dados (DCBD).

Segundo Soares (2013, p. 31), a tarefa da DCDB é extrair conhecimento de bases de dados através da identificação e exploração de padrões interessantes. Na DCDB, a obtenção de conhecimento ocorre a partir de bases de dados fortemente estruturadas, geralmente armazenadas em Sistemas Gerenciadores de Bancos de Dados (SGBD). Dados mantidos em SGBD apresentam uma estrutura de representação previamente definida. Muitas são as funções de um SGBD, porém uma das mais importantes é prover integridade às bases de dados gerenciadas pelo mesmo (DATE, 2005, p. 12).

Segundo Corrêa (2003), o processo de DCBD é composto de 6 etapas. A DCBD baseia-se no armazenamento de dados na forma estruturada. Assim, é necessário definir sobre qual domínio de dados trabalhar. A próxima etapa é selecionar e coletar o conjunto de dados necessários. A seguir existe a etapa de processamento, também conhecida como pré-processamento, que visa eliminar os dados que não estão no formato adequado, tais como dados incompletos ou repetidos. Na etapa de transformação os dados deverão ser armazenados adequadamente para facilitar sua utilização pelas técnicas de Mineração de Dados (MD). Na etapa de MD os dados são submetidos a algoritmos visando a descoberta de conhecimento. Entre as técnicas de MD pode-se destacar: Aprendizado de Máquina (AM), reconhecimento de padrões, estatística, classificação, clusterização, modelos gráficos, entre outros. Na etapa final, conhecida como interpretação/avaliação os resultados do processo de descoberta de conhecimento podem ser mostrados de diversas formas, porém devem ser apresentados de modo que o usuário possa entender e interpretar os resultados obtidos.

Chen (2001, p. 12) destaca que 80% dos dados de uma organização estão contidos em documentos textuais. Assim sendo, pode-se perceber que somente 20% de todo conhecimento das organizações é passível de ser descoberto através da DCBD. Segundo Baranauskas (2001, p. 145), a análise de dados armazenados em formato não estruturado (texto em formato livre) é considerada mais complexa em relação aos dados estruturados (presentes em bases de dados). Logo, são necessárias técnicas e ferramentas específicas para o tratamento desses tipos de dados. Com toda essa demanda de documentos textuais para serem analisados surgiu a técnica de Descoberta de Conhecimento em Texto (DCT).

Segundo Wives (2000), dentro da DCT existem métodos para a descoberta de conhecimento em bases textuais, como por exemplo, sumarização, agrupamento de documentos e MT. Esta é a aplicação de técnicas para extração de conhecimento de dados não estruturados. Ela é o método utilizado no presente trabalho e será abordada mais profundamente na seção a seguir.

2.3.1 Mineração de Texto

A MT é um processo de obtenção de conhecimento oriundo de bases de dados textuais, ou seja, documentos em linguagem natural, e que, portanto, possuem pouca ou nenhuma estruturação. Os documentos escritos em linguagem natural não possuem garantia nenhuma de integridade. Além disso, documentos deste tipo, ainda que dentro de um mesmo contexto, podem apresentar enormes diferenças estruturais entre si. Considerando, por exemplo, dados referentes a um curriculum vitae, pode-se ter um pequeno texto informal descrevendo dados pessoais e experiência profissional ou, no extremo oposto, um documento organizado em seções e subseções, com links para dados das empresas e instituições em que se trabalhou. A justificativa para tal fato decorre da própria natureza dos dados. Em geral, dados textuais referem-se a massas de informações, quase sempre digitalizadas e que não possuem rígida estruturação (SOARES, 2013, p. 32).

Segundo Konchady (2006, p. 50), o principal elemento da MT é a coleção de documentos, pois constitui o conjunto de dados sobre o qual este processo é realizado. O autor define um documento (elemento básico de uma coleção) como uma unidade discreta de dados em formato textual. A coleção de documentos constituí-se como a base textual onde o conhecimento será descoberto. Feldman e Sanger (2007, p. 79) descrevem o fato da MT lidar com coleções de documentos volumosos. Esta característica está relacionada com o fato da análise manual de uma base textual deste porte ser altamente onerosa.

Em alguns cenários, a coleção de documentos pode ser estática, ou seja, o conjunto de documentos selecionado permanece inalterado, tanto em elementos, como em conteúdo. Entretanto, em grande parte dos casos, a coleção de documentos é dinâmica, podendo ter elementos incluídos, excluídos e alterados, o que demanda desafios ainda maiores para um sistema de MT (ARANHA et al., 2004, p. 105).

Segundo Aranha (2007, p. 40), o processo de MT é dividido em cinco macro etapas: coleta, pré-processamento, indexação, mineração e análise da informação. Nas seções a seguir são detalhadas as atividades realizadas em cada etapa do processo.

2.3.1.1 Coleta de Dados

Segundo Konchady (2006, p. 70), a primeira etapa do processo de MT é a coleta de dados. Ela tem como função formar a base textual de trabalho. A atualização é feita pela simples adição de um novo conteúdo, remoção de conteúdos antigos, ou, substituição da base por uma totalmente nova (ARANHA, 2007, p. 50).

Conforme descrito em Soares (2013, p. 50), esta etapa envolve a seleção dos documentos que irão compor a base textual. Segundo o autor, é importante ressaltar que documentos devem ser relevantes ao domínio da aplicação do conhecimento a ser extraído, pois a seleção de documentos irrelevantes para fazer parte da base textual pode prejudicar o processo de MT, além de aumentar a dimensionalidade dos dados desnecessariamente. Aranha (2007, p. 41) complementa ao classificar essa fase como trabalhosa, pelo fato de muitas vezes os dados não estarem num formato adequado ao processo de MT.

Segundo Aranha (2007, p. 41), o principal problema em coletar dados para a MT é descobrir onde os dados estão armazenados. Depois disso, recuperar documentos relevantes ao domínio do conhecimento. Segundo o autor, esse procedimento se estabelece basicamente em três ambientes distintos: em tabelas de diferentes bancos de dados, nas pastas do disco rígido e na internet.

Os desafios encontrados na coleta em bases de dados podem ser diminuídos com o uso de um *data warehouse*. O *data warehouse* é um repositório de dados geralmente construído para dar suporte às pessoas que tomam decisões, tais como gerentes e diretores. Neste repositório os dados extraídos de diferentes bancos de dados são integrados e sua consistência é verificada na medida do possível, antes de serem carregados no data warehouse. Dessa forma, eles se tornam uma boa fonte de dados para o processo de MT (BATISTA, 2003, p. 38).

Segundo Aranha (2007, p. 41), para a coleta no disco rígido existe o sistema de arquivos do sistema operacional. Um sistema de arquivos é um conjunto de estruturas lógicas e de rotinas, que permitem ao sistema operacional o acesso ao disco rígido. Através dele é possível navegar através da estrutura de árvore formada pelos arquivos e pastas, realizando a coleta. Essa foi a estratégia adotada neste estudo para a coleta dos documentos.

Segundo Batista (2003, p. 50), na internet existe uma infinidade de páginas pessoais, institucionais, e diversas outras fontes disponíveis para a coleta de documentos, como livros e artigos. O autor ainda explica que para facilitar o acesso a esses documentos na internet, muitas ferramentas de apoio têm sido construídas, dentre elas podem ser citados os diretórios de assunto e os motores de busca baseados em robô.

Porém, não existem somente questões computacionais que podem dificultar o processo de coleta. Em Batista (2003, p. 38-39) são citadas algumas outras problemáticas. Podem existir barreiras legais ou éticas que impeçam que os dados sejam disponibilizados para análise. Por exemplo, instituições financeiras possuem barreiras legais que, sob algumas circunstâncias, impedem que dados referentes às movimentações financeiras de clientes sejam disponibilizados. Ainda, podem existir razões éticas que restrinjam o acesso aos dados, como ocorre, por exemplo, com dados que identificam pacientes na área médica e clientes na área legal. Ainda, alguns dados podem pertencer a pessoas ou departamentos que pelos mais diversos motivos não permitem que os dados sejam analisados.

2.3.1.2 Pré-Processamento

Uma vez realizada a coleta de dados, o próximo passo é a preparação dos documentos para que os mesmos possam ser manipulados pelos algoritmos de MT. Esta segunda etapa denomina-se pré-processamento e é responsável por criar uma representação do texto mais estruturada, capaz de alimentar algoritmos de descoberta de conhecimento (SOARES, 2013, p. 51).

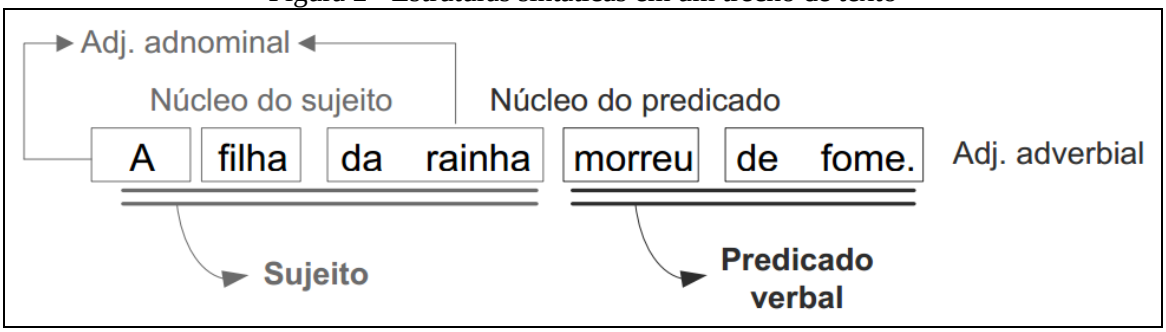
Aranha (2007, p. 42) descreve o pré-processamento como um conjunto de transformações realizadas sobre alguma coleção de textos com o objetivo de fazer com que esses passem a ser estruturados. Ainda segundo o autor, esta etapa tem por finalidade melhorar a qualidade dos dados disponíveis e organizá-los. As ações realizadas na etapa de pré-processamento visam preparar os documentos para serem submetidos a algum algoritmo de indexação ou de descoberta de conhecimento.

Durante a transformação do texto em formato estruturado, existe a possibilidade de que a informação intrínseca ao conteúdo dos textos seja perdida. Um desafio, nesse caso, é obter

uma boa representação, minimizando a perda de informação. A etapa de pré-processamento no processo de MT é, portanto, fundamental para o desempenho de todo o processo (MARTINS, 2003, p. 36).

Um documento textual pode ser representado através da linguística. Ele apresenta estruturas sintáticas e semânticas, ainda que estejam implícitas no texto. Essas estruturas fazem com que um documento textual seja interpretado como estruturado. Elementos tipográficos como pontuações, letras maiúsculas, números e outros caracteres especiais, ajudam a definir subcomponentes de um documento como parágrafos, títulos, autores e outras informações. Até mesmo a sequência das palavras pode definir características importantes de um documento (SOARES, 2013, p. 36). A Figura 2 ilustra a existência de várias estruturas sintáticas num pequeno trecho de texto.

Figura 2 - Estruturas sintáticas em um trecho de texto



Fonte: Soares (2013, p. 36).

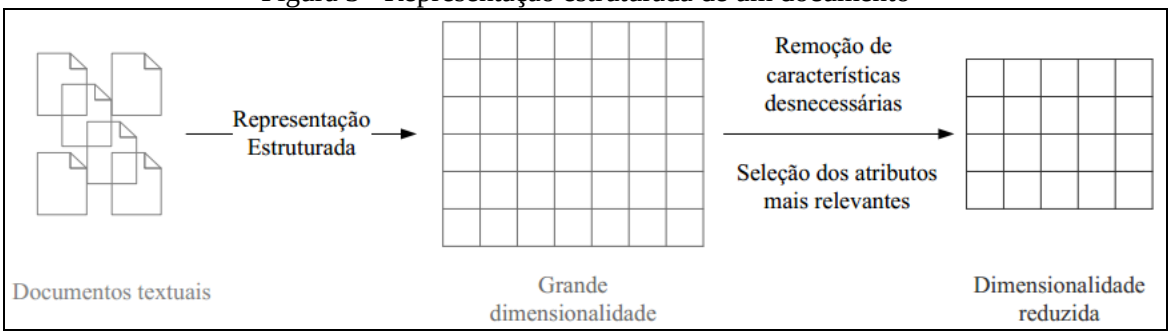
Segundo Feldman e Sanger (2007, p. 20), existem documentos que fornecem mais informações sobre a sua estrutura do que aquela provida pelo seu conteúdo. Documentos com extensivos e consistentes elementos de formatação (*tags*), estrutural ou visual, em que metadados podem ser facilmente inferidos, são denominados semiestruturados. Powel (2007, p. 150) classifica os documentos XML como semiestruturados quando eles apresentam diagramação estrutural que revela características adicionais sobre sua estrutura.

Para Feldman e Sanger (2007, p. 51), as operações de pré-processamento envolvidas em MT possuem como objetivo realçar os diferentes elementos presentes em um documento escrito em linguagem natural, com o intuito de transformá-lo de uma representação estrutural irregular e implícita, a uma representação explicitamente estruturada. Entretanto, Soares (2013, p. 37) afirma que dado o número potencialmente enorme das palavras, frases, sentenças e elementos tipográficos que até mesmo um simples e pequeno elemento pode apresentar, uma tarefa essencial para qualquer processo de MT é a identificação do conjunto mais simples de características de um documento que pode representá-lo como um todo. Segundo o autor, este conjunto de características recebe a denominação de modelo de

representação e cada documento em uma coleção é representado pelo conjunto de características que o seu modelo de representação possui.

Segundo Soares (2013, p. 51), uma vez criado o modelo de representação dos documentos, é necessário que esse seja computacionalmente tratável. Para isto são realizadas algumas operações de análise de dados, que visam selecionar somente as características que melhor expressam o conteúdo dos documentos. Esse processo é ilustrado na Figura 3.

Figura 3 - Representação estruturada de um documento



Fonte: Soares (2013, p. 52).

Para a transformação do texto não estruturado num modelo de representação, várias etapas necessitam ser realizadas. Segundo Soares (2013, p. 52), o primeiro passo do processo de pré-processamento é a tokenização ou atomização, tendo como objetivo dividir um documento textual em unidades mínimas, mas que exprimam a mesma semântica que o documento original. O autor utiliza o termo *token* para designar estas unidades que, geralmente, correspondem a somente uma palavra do texto. Porém, em alguns casos, os *tokens* podem ser formados por agrupamento de palavras chamados de termos.

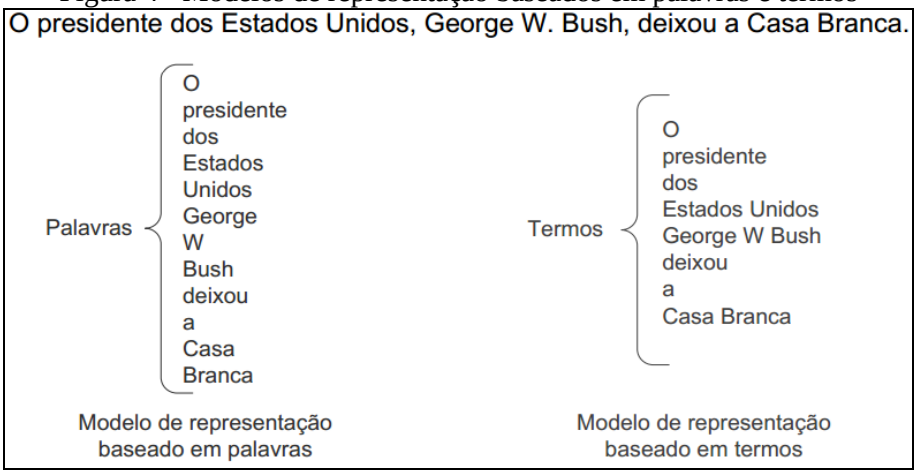
Segundo Feldman e Sanger (2007, p. 53), o processo de tokenização é auxiliado pelo fato das palavras serem separadas por caracteres de controle de arquivo ou de formatação, tais como, espaços ou sinais de pontuação, que podem ser considerados *tokens* delimitadores. Soares (2013, p. 53) afirma que a criação de *tokens* de um documento baseado em seus delimitadores é uma estratégia simples e que apresenta bons resultados. Entretanto, o autor enfatiza o fato de que a tarefa de identificação de *tokens*, que é relativamente simples para o ser humano, pode ser bastante complexa para ser executada pelo computador. Este fato, segundo Carrilho Junior (2007, p. 23), é atribuído ao grande número de papéis que os delimitadores podem assumir. Por exemplo, o ponto final pode ser utilizado para marcar o final de uma sentença, mas também é usado em abreviações e números.

Soares (2013, p. 38-39) classifica os *tokens* segundo sua expressividade semântica, sendo eles (em nível crescente de expressividade semântica): caracteres, palavras e termos. Os caracteres são componentes individuais responsáveis pela formação de blocos com um nível

semântico maior, como palavras e termos. Em geral, são utilizados junto com a posição em que ocorrem no texto. Abordagens que utilizam a combinação de um número predefinido de caracteres, tais como bigramas ou trigramas, são mais comuns do que a utilização de um único caractere. Embora modelos de representação que utilizem somente caracteres sejam considerados os mais próximos da realidade, a alta dimensionalidade, torna altamente custosa a utilização de diversas técnicas computacionais. A abordagem baseada em palavras possui certas vantagens em relação à sua predecessora. Geralmente as palavras possuem mais expressividade semântica do que os caracteres, expressando melhor ideias e conceitos. Outra vantagem é o fato de que ao invés de ter que se representar vários caracteres, se representa somente uma palavra. Isso faz com que o problema da alta dimensionalidade seja amenizado. Por esta razão, é muito utilizada para a construção de modelos de representação de um documento. Entretanto, termos multivocabulares, como "casa da moeda", podem perder seus valores semânticos quando separados. Os termos podem ser compostos por uma única palavra ou por um conjunto de palavras que exprimam, por completo, a semântica que era desejada no texto. A extração de termos, na maioria das vezes, é auxiliada por um dicionário de palavras que permite identificar os termos que são compostos por mais de uma palavra.

Na Figura 4 as diferenças na utilização do modelo baseado em palavras e baseado em termos são mostradas. O autor explana o fato que no modelo baseado em palavras houve perda semântica e um número maior de elementos foi encontrado. Por sua vez, o modelo baseado em termos exigiu que o sistema tivesse conhecimento prévio dos termos que são compostos por mais de uma palavra.

Figura 4 - Modelos de representação baseados em palavras e termos



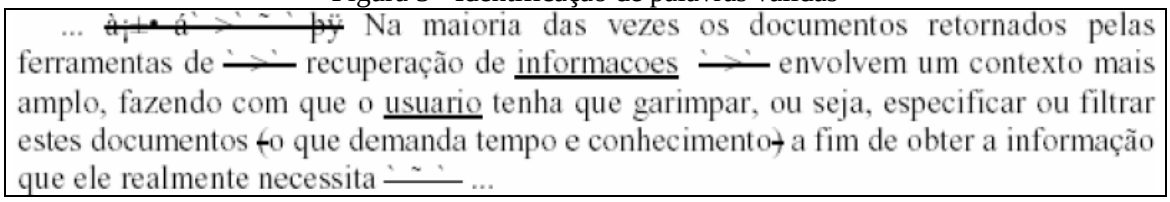
Fonte: Soares (2013, p. 40).

Em função de suas características, o presente estudo usa o modelo de representação baseado em palavras. Segundo Salton (1983, p. 51), a identificação das palavras nos documentos a serem indexados nada mais é do que a identificação de palavras analisando-se

as sequências de caracteres do texto. Salton (1983, p. 51) aconselha fazer um *dictionary lookup*, ou seja, comparar as sequências de caracteres retirados do texto com um dicionário a fim de validar se essas palavras realmente existem. Aranha (2007, p. 43) descreve esse processo de validação como sendo bastante útil, especialmente quando o documento apresenta muitos caracteres inválidos ou palavras com erros gramaticais. Ainda segundo o autor, as sequências de caracteres inválidas devem ser eliminadas e as palavras com erros corrigidas.

A Figura 5 apresenta um trecho com diversas sequências de caracteres. As sequências riscadas são inválidas, e não devem passar pela sequência de identificação de palavras. As demais sequências vão para a verificação de um dicionário. As palavras sublinhadas são palavras inexistentes no dicionário e devem ser corrigidas ou aprendidas. Os caracteres de pontuação são desprezados.

Figura 5 - Identificação de palavras válidas



Fonte: Aranha (2007, p. 44).

Conforme Soares (2013, p. 54), uma vez realizada a tokenização, o passo seguinte é a identificação do que pode ser desconsiderado nos passos posteriores do processamento dos textos, pois nem todas as palavras dos documentos devem ser adicionadas na estrutura de índice (ARANHA, 2007, p. 45). Em um documento, existem muitos *tokens* que possuem pouco valor semântico, sendo úteis apenas para o entendimento e compreensão geral do texto. Esses *tokens* são denominados *stopwords* e fazem parte de uma lista chamada *stoplist* (BASTOS, 2006, p. 37). Aranha (2007, p. 46) tem um entendimento semelhante. Segundo ele, é construída uma lista contendo todas as palavras que não devem ser indexadas.

As preposições são um exemplo de palavras com pouca expressividade semântica, pois são termos que servem para fazer o encadeamento de ideias e palavras. São termos inerentes à linguagem e não ao conteúdo dos documentos. Normalmente, as palavras que aparecem em muitos documentos não são indexadas, pois sua utilização compromete a precisão e a eficiência de um sistema de busca. O prejuízo semântico dessa estratégia é perder a busca exata por termos compostos, como "máquina de lavar", onde a preposição "de" não pode ser buscada (ARANHA, 2007, p. 46).

Segundo Soares (2013, p. 54), fazem parte de uma *stoplist* termos como conjunções, preposições, pronomes e artigos, pois são considerados termos de menor relevância, ou seja,

sua presença pouco contribui para a determinação do valor semântico de um documento. Conforme descrito em Silva (2004, p. 23), uma *stoplist* bem elaborada permite a eliminação de muitos termos irrelevantes, tornando mais eficiente o resultado obtido pelo processo de MT. O autor complementa com o fato que entre 40 e 50% do total de palavras de um texto são removidos com o uso dessa técnica.

A Figura 6 apresenta um documento após ser validado por uma *stoplist*. Neste caso, a lista de *stopwords* contém artigos, preposições, conjunções e algumas palavras que não devem ser adicionadas ao índice por possuir frequência elevada.

Figura 6 - Identificação de *stopwords*

... ~~Na~~ maioria ~~das~~ vezes ~~os~~ documentos retornados ~~pelas~~ ferramentas ~~de~~ recuperação ~~de~~ informações evolvem ~~um~~ contexto ~~mais~~ amplo fazendo ~~com~~ ~~que~~ ~~o~~ usuário tenha ~~que~~ garimpar ~~ou~~ ~~seja~~ especificar ~~ou~~ filtrar estes documentos ~~o~~ ~~que~~ demanda tempo ~~o~~ conhecimento ~~a~~ fim ~~de~~ obter ~~a~~ informação ~~que~~ ~~ele~~ realmente necessita ...

Fonte: Aranha (2007, p. 47).

O uso de técnicas de PLN no pré-processamento tem o objetivo de identificar a real importância de cada termo em determinados contextos, possibilitando ganho na qualidade dos resultados do processo de MT. PLN é utilizado para agregar valores semânticos que poderão beneficiar o processo de descoberta de conhecimento nas etapas posteriores. Embora muitas abordagens ao processo de MT não façam uso de PLN, sua utilização tem incrementado os resultados obtidos, justificando o esforço computacional adicional (SOARES, 2013, p. 55). Esse assunto será aprofundado na seção 2.4.

2.3.1.3 Indexação

Segundo Manning, Raghavan e Schutze (2007, p. 140), indexação é a fase responsável por criar estruturas de dados denominadas índices, capazes de permitir a consulta a uma informação gerada com base na etapa de pré-processamento, sem que seja necessário analisar toda a base textual. Semelhante ao sumário de um livro, que é uma lista detalhada com a indicação de localização no texto, dos principais tópicos abordados no interior deste, índices são utilizados para otimizar a velocidade e o desempenho da busca por um documento relevante em relação ao termo buscado. O custo pelo ganho de tempo durante a Recuperação de Informação (RI) é o espaço de armazenamento adicional necessário para armazenar o índice. Entende-se RI como o uso de técnicas computacionais para recuperar documentos relevantes para uma determinada expressão de busca (SOARES, 2013, p. 63).

Conforme descrito em Aranha (2007, p. 48), há bastante tempo cientistas estudam meios de catalogar informações textuais de forma automática. Entre eles destaca-se Gerard Salton (SALTON, 1983), que vem trabalhando nessa área desde a década de sessenta e já publicou mais de 150 artigos. Contudo, muitas pessoas acreditam que esta é uma área nova. Esta ideia talvez tenha surgido com a grande popularização das máquinas de busca que tornaram possível a pesquisa de conteúdo de páginas *web* (SOARES, p. 63).

Com o desdobramento da informática, catalogar informações se tornou cada vez mais rápido e produtivo. Técnicas bem definidas e testadas foram inseridas na literatura acadêmica, sendo reutilizadas em trabalhos científicos. O *Special Interest Group on Information Retrieval* (SIGIR) da *Association for Computing Machinery* (ACM) promove uma conferência internacional de pesquisa e desenvolvimento em RI que ocorre anualmente e é um dos meios de divulgação dos estudos na área (ACM, 1996, p. 21).

O resultado do processo de indexação é o arquivo de índices, sendo que existem diversas técnicas para implementar o mesmo. Segundo Konchady (2006, p. 50), a técnica mais utilizada são os índices invertidos. Salton (1983, p. 39) afirma que outros tipos de arquivos podem ser utilizados, mas sua experiência mostra que esse tipo de estrutura é um dos mais eficientes para a indexação de documentos.

Em sua apresentação básica, um índice invertido é uma estrutura de dados composta de uma lista ordenada, geralmente denominada vocábulo ou vocabulário, que armazena todas as palavras distintas encontradas nos textos e os documentos nos quais elas ocorrem. Informações adicionais como a frequência e posição de ocorrência das palavras no texto também podem ser armazenadas (SOARES, 2013, p. 64).

2.3.1.4 Mineração

Segundo Soares (2013, p. 65), é na etapa de mineração que ocorre a busca efetiva por conhecimentos novos e úteis a partir dos dados. O autor explicita que nesta fase não existe nenhum enriquecimento dos documentos, ou seja, todos os dados provenientes de documentos já foram computados pela etapa de pré-processamento.

Segundo Soares (2013, p. 65), a mineração compreende a aplicação de algoritmos sobre os dados de forma a extrair o conhecimento implícito neles. Segundo Aranha (2007, p. 58), nessa fase pode-se utilizar algoritmos provenientes de diversas áreas do conhecimento, tais como AM, estatística e MD. O autor define o objetivo desta fase como criar um modelo preditivo, ou seja, uma função matemática que é capaz de aprender o mapeamento entre um conjunto de variáveis de entrada de dados e uma variável de resposta ou destino. Contudo,

conforme descrito por Schaffer (1994, p. 262), decidir qual algoritmo é ótimo para o problema não é uma tarefa trivial. Esse fato ocorre pois nenhum algoritmo possui bom desempenho para todas as aplicações.

A escolha do algoritmo a ser utilizado está relacionada com o objetivo da tarefa de MT. Este objetivo, definido no início do processo, irá determinar quais as opções de algoritmos de descoberta de conhecimento que se aplicam ao problema. Além disso, outros detalhes devem ser considerados como, por exemplo, a necessidade ou não de que o conhecimento aprendido seja facilmente interpretável (SOARES, 2013, p. 66).

Goldschmidt e Passos (2005, p. 120) ressaltam que a dificuldade de escolha de um algoritmo de descoberta de conhecimento apropriado é intensificada na medida que surgem novos algoritmos com esse propósito, aumentando a diversidade de escolhas. Geralmente a escolha dos algoritmos se restringe às opções conhecidas pelo analista do processo que, pela falta de conhecimento, deixa de considerar muitas alternativas promissoras.

Uma abordagem utilizada para descobrir conhecimento é procurar por padrões contidos nos textos e agrupá-los em categorias, descobrindo assim conhecimentos que têm estreita relação entre si. Reconhecer padrões é uma tarefa simples e cotidiana para o cérebro humano. De forma contínua as pessoas estão reconhecendo padrões, seja de forma consciente ou mesmo inconsciente. Melo (2012, p. 28) afirma que quando o cérebro humano observa um determinado objeto previamente conhecido ele consegue imediatamente identificá-lo e classificá-lo, enquanto que, se um novo objeto lhe é apresentado, ele estabelece relações com outros objetos previamente conhecidos, para então classificá-lo.

Para uma máquina, tal processo de reconhecimento de padrões não é uma tarefa tão simples. Melo (2012, p. 36) afirma que “a automatização do reconhecimento de padrões, entre outras coisas, consiste da abstração de processos, identificação de características relevantes, manipulação e estabelecimento de relação entre as características”. Um problema de reconhecimento automático de padrões refere-se a uma tarefa de classificação ou categorização, onde as classes são definidas pelo projetista do sistema ou são aprendidas de acordo com a similaridade dos padrões.

Segundo Campos (2001, p. 60), há duas maneiras de reconhecer e/ou classificar um padrão: supervisionada e não supervisionada. Na primeira, o padrão de entrada é identificado como um membro de uma classe pré-definida, ou seja, a classe é definida pelo projetista do sistema. Na classificação não supervisionada o padrão é definido por uma fronteira de classe desconhecida. Durante a realização deste estudo, percebeu-se que sua natureza se encaixava na categoria supervisionada, sendo esta adotada para o reconhecimento de padrões.

Para Melo (2012, p. 40), somente após o avanço e a disponibilidade dos recursos computacionais, tornou-se viável o projeto e a utilização de elaborados métodos de análise e classificação de padrões. Em muitas aplicações não existe somente uma única abordagem para classificação que seja ótima e, por isso, a combinação de várias abordagens de classificadores pode ser usada.

Uma técnica computacional utilizada com esse propósito são as Redes Neurais (RN). Para Campos (2001, p. 65), uma RN é um sistema composto por vários neurônios. Estes neurônios estão ligados por conexões, chamadas conexões sinápticas. Alguns neurônios recebem excitações do exterior e são chamados neurônios de entrada e correspondem aos neurônios dos órgãos dos sentidos. Outros têm suas respostas usadas para alterar, de alguma forma, o mundo exterior e são chamados neurônios de saída e correspondem aos motoneurônios que são os neurônios biológicos que excitam os músculos. Os neurônios que não são nem entrada nem saída são conhecidos como neurônios internos.

Outra abordagem é o AM. Segundo Melo (2012, p. 22), o AM é uma área da IA responsável pelo desenvolvimento de teorias computacionais focadas na criação do conhecimento artificial. A partir do momento que é criado o conhecimento artificial, pode-se tornar possível que as máquinas aprendam sozinhas, sem que elas tenham sido programadas. Softwares desenvolvidos com esta tecnologia possuem a característica de tomarem decisões com base no conhecimento prévio acumulado através da interação com o ambiente.

Após analisar essas abordagens, elas foram desconsideradas no presente estudo como técnica para descoberta de conhecimento, principalmente pelo fato de necessitarem de um treinamento prévio para poder desempenhar sua função. O treinamento é a etapa mais onerosa da descoberta de conhecimento. Utilizar uma alternativa que não necessite de treinamento e ainda assim possibilite a descoberta do conhecimento é uma abordagem mais atraente.

Por isso, foram utilizados os chamados métodos linguísticos. Para Castro e Prado (2002, p. 136), os métodos linguísticos são úteis para padrões que não podem ser descritos convenientemente através de medidas numéricas. Utilizam-se as características linguísticas das frases do documento textual para encontrar padrões úteis. São utilizadas as estruturas sintáticas, semânticas e classificação gramatical das palavras para descobrir padrões e, assim, conhecimento. O presente estudo faz uso de dois padrões: todas as palavras que forem classificadas morfologicamente como substantivo ou nome próprio são consideradas classes da ontologia e verbos são considerados relacionamentos. Caso um verbo em uma sentença esteja entre duas classes ele é considerado um relacionamento, caso contrário ele é desconsiderado. Esse fato ocorre pois os relacionamentos devem ligar duas classes.

2.3.1.5 Análise da informação

Conforme descrito em Zhu e Davidson (2007, p. 150), a etapa de análise da informação, algumas vezes chamada de pós-processamento, abrange o tratamento do conhecimento obtido na etapa de mineração, através da análise, visualização e interpretação deste. Segundo os autores, esse tratamento tem como objetivo viabilizar a avaliação da utilidade do conhecimento descoberto.

Segundo Aranha (2007, p. 59), esta fase envolve todos os participantes do processo de MT, sendo eles o especialista no domínio, o usuário final e o analista de dados. O especialista no domínio irá verificar a compatibilidade dos resultados com o conhecimento disponível no domínio. O usuário é responsável por julgar a aplicabilidade dos resultados. Por fim, o analista de dados tenta descobrir se o processo de MT atingiu as espectatívas, avaliando os resultados de acordo com algumas métricas, tais como, taxa de erro e complexidade do modelo.

Existem diversas maneiras de se avaliar a mineração como um todo, seja de forma qualitativa ou quantitativa. A utilização de métricas de desempenho é considerada uma forma quantitativa, ao passo que a utilização do conhecimento de especialistas no domínio é considerada uma forma qualitativa. Os especialistas devem ser sempre consultados em todas as etapas da mineração, guiando a análise, ajudando a resolver as situações de conflito, indicando caminhos e complementando informações (CARRILHO JUNIOR, 2007, P. 56).

O objetivo de uma métrica de desempenho é graduar a execução de uma tarefa. As principais métricas de avaliação utilizadas em MT foram adotadas da área de RI e são baseadas na noção de relevância. Um documento é considerado relevante quando possui importância para o tópico considerado. Abrangência, precisão e *fall-out* são as métricas de desempenho mais utilizadas (SOARES, 2013, p. 66).

Soares (2013, p. 69-70) descreve brevemente as métricas citadas acima. A abrangência mede a habilidade do sistema em recuperar os documentos mais relevantes para seu usuário com base no termo ou expressão utilizada na formulação de sua busca. A precisão mede a habilidade do sistema de manter os documentos irrelevantes fora do resultado de uma consulta. O *fall-out* considera que a quantidade de documentos irrelevantes pode ser modificada pelo crescimento ou decrescimento da base textual. Assim sendo, ela mede a quantidade de documentos irrelevantes, permitindo que se identifique se a quantidade de documentos relevantes permanece a mesma quando o número de documentos varia.

Soares (2013, p. 67) ressalta que, de acordo com o objetivo do processo de DCT, métricas de avaliação de desempenho diferentes das citadas devem ser utilizadas. O autor exemplifica que uma tarefa de sumarização não será bem avaliada utilizando abrangência, precisão ou *fall-out*.

Muitas vezes, de forma a facilitar a análise do conhecimento obtido, podem ser utilizados métodos de transformação de dados, que consistem, basicamente, na conversão de uma forma de visualização para outra. Da mesma forma que as medidas de desempenho, diferentes estratégias de visualização podem ser empregadas, cada qual mais adequada ao objetivo de MT (KANTARDZIC, 2002, p. 210).

Segundo Spence (2001, p. 23), visualizar é tornar algo visual ou visível, ver uma imagem mental ou figurá-la mentalmente. Para o autor, o conceito de visualização é definido como a transformação de conceitos abstratos em imagens reais ou mentalmente visíveis, ou inserido no contexto computacional. O autor cita como exemplo a conversão de números ou dados para um formato gráfico que pode ser facilmente compreendido.

Segundo Spence (2001, p. 24), a visualização de informação pode ser definida como o uso de representações visuais, interativas e suportada por um computador, de dados abstratos objetivando a ampliação da cognição humana. Conforme descrito em Carvalho e Vaz (2004, p. 3), esta ideia atende a inúmeras áreas. Os autores ainda afirmam que a visualização de informação é uma área que dia após dia vem crescendo em importância e relevância nas tarefas humanas.

Branco (2003, p. 2) afirma que representações gráficas têm sido utilizadas como instrumento de comunicação desde os primórdios da humanidade. Para o autor, com o advento da ciência, as representações gráficas passaram a embutir significado cada vez mais regido por convenções, como por exemplo, gráficos matemáticos e cartas cartográficas. Normalmente, essas representações tem como propósito comunicar uma ideia que já existe. Entretanto, o autor conclui que tendo como propósito aproveitar as características da percepção visual humana para a resolução de problemas lógicos, uma segunda abordagem possível consiste em utilizar as representações gráficas para criar ou descobrir a própria ideia.

Segundo Carrilho Junior (2007, p. 56), uma das formas mais intuitivas de se analisar um resultado seria com a utilização de gráficos, tabelas e outras formas visuais. Assim, o usuário pode utilizar os resultados para a tomada de decisão. Segundo Soares (2013, p. 67), para melhor visualizar o conhecimento obtido no processo de MT é comum o uso de árvores de decisão ou regras.

2.4 PROCESSAMENTO DE LINGUAGEM NATURAL

Segundo Silva et al. (2007, p. 4), desde o início de 1940, o computador digital não só vem contribuindo para avanços substantivos nos diversos campos do conhecimento científico, como também tem sido responsável pelo desenvolvimento e pela abertura de novas frentes de pesquisas. Os autores afirmam que os computadores colocaram seus idealizadores diante de um enigma: como fazê-los entender as instruções necessárias para executar tarefas. A criação de linguagens de programação foi a resposta que os cientistas encontram para este enigma.

Porém, o sonho de construir um computador que se comunicasse com uma pessoa sempre foi buscado pela IA. Mesmo os mais antigos contos e filmes de ficção científica retratam robôs, computadores e outras máquinas conversando com pessoas através da linguagem natural. Com o intuito de que o computador interaja com um ser humano através da linguagem natural, surgiu a área de pesquisa denominada de PLN (MÜLLER, 2003, p. 2).

Segundo Müller (2003, p. 2), o processo de PLN é dividido em quatro etapas: análise morfológica, análise sintática, análise semântica e análise pragmática. A análise morfológica estuda a construção das palavras, com seus radicais e afixos, que correspondem às partes estáticas e variantes da palavra, além das classes gramaticais, com suas inflexões verbais.

A análise sintática diz respeito ao estudo das relações formais entre as palavras. Para a implementação de analisadores sintáticos é necessária a construção de um *parser* para que seja feita a verificação da posição das palavras na frase. Cada uma dessas palavras possui combinações coerentes com a sintaxe definida na linguagem. Não é coerente, por exemplo, a colocação de um adjetivo antes de um artigo. A verificação disto é tarefa da análise sintática.

A análise semântica é um processo de mapeamento de sentenças de uma linguagem visando a representação de seu significado, baseado nas construções obtidas na análise sintática. Apesar do extenso processamento realizado nas análises morfológicas e sintática, em alguns casos não é possível distinguir certas categorias de palavras e muito menos o objetivo da frase. Para tanto são acrescentados nas árvores de *parser* as chamadas ações semânticas, tendo como finalidade adicionar significado às palavras.

A análise pragmática diz respeito ao processamento da forma que a linguagem é utilizada para comunicar. Ela estuda como os significados obtidos na análise semântica agem sobre as pessoas e seu contexto. Este tipo de análise vai além da estrutura de somente uma frase. Ela busca nas demais frases a compreensão do contexto que falta ao texto em análise.

A seguir são descritas duas técnicas de PLN utilizadas no processo de MT. A primeira delas é a etiquetagem de classes gramaticais e faz parte da análise morfológica. Ela objetiva

classificar todas as palavras contidas no documento textual em sua classe gramatical. Segundo Cegalla (2005, p. 31), entende-se por classe gramatical a forma de classificação de uma palavra segundo seu significado e função. Na língua portuguesa existem dez classes gramaticais: substantivo, artigo, adjetivo, numeral, pronome, verbo, advérbio, preposição, conjunção e interjeição. A identificação da classe das palavras presentes em uma sentença facilita o entendimento desta e muitas vezes soluciona alguns problemas simples de ambiguidade (SOARES, 2013, p. 56).

Aranha (2007, p. 69) tem um entendimento semelhante sobre etiquetagem, porém com ênfase no uso de ontologias. Segundo ele, a etiquetagem é a rotulação das palavras de acordo com uma dada classificação que pode atender uma ontologia. Essa ontologia é uma estrutura básica de representação presente em quase todas as línguas, constituída de duas categorias principais: palavras funcionais e palavras de conteúdo. As palavras funcionais englobam os advérbios, preposições, conjunções e interjeições. Esse conjunto é estático, pois raramente surge uma nova palavra funcional em uma língua. Em contrapartida, essas palavras aparecem mais frequentemente em um texto. As palavras de conteúdo dizem respeito aos substantivos, artigos, adjetivos, numerais, pronomes e verbos. Este conjunto é dinâmico, pois a todo momento surge uma nova palavra de conteúdo em uma língua. Essas palavras aparecem menos frequentemente em um texto.

Qualquer documento textual apresenta muitas palavras flexionadas nas mais diversas formas. Na língua portuguesa, um substantivo pode ser flexionado em gênero, número e grau, e apresentar o mesmo valor semântico. O processo de formação de palavras é, na maior parte das vezes, realizado pela derivação de radicais, resultando na criação de palavras que também exprimem o mesmo significado (CEGALLA, 2005, p. 40).

Para resolver este problema, existe a segunda técnica de PLN denominada de lematização ou *stemming*. Lematização é o processo de reduzir ao radical original palavras derivadas ou flexionadas deste. O principal objetivo do processo de lematização é reduzir a grande dimensionalidade dos documentos durante o processo de MT. Com a remoção de prefixos e sufixos de palavras derivadas de um mesmo radical, que antes seriam considerados como palavras diferentes, obtém-se uma única palavra para representar todas elas (SOARES, 2013, p. 57).

Segundo Soares (2013, p. 58), o algoritmo de lematização mais utilizado é o algoritmo de Porter. Ainda segundo o autor, o funcionamento do algoritmo de Porter consiste na identificação e substituição das diversas inflexões e derivações de uma mesma palavra por um mesmo radical. Como termos que derivam de um mesmo radical possuem significados

semelhantes, consegue-se reunir em uma única palavra a importância de todas as suas derivações. Este algoritmo remove cerca de sessenta sufixos diferentes.

Segundo Silva et al. (2007, p. 8), o algoritmo de Porter foi criado em 1980, com suporte somente para a língua inglesa. Em 2007 ele recebeu uma versão para a língua portuguesa. Essa versão é composta de cinco passos:

a) passo um: remoção de sufixos comuns, como por exemplo, eza, ismos, ável, ível, oso, ações, mente, entre outros;
b) passo dois: remoção de sufixos verbais, caso a palavra não tenha sido alterada pelo passo um, como por exemplo, ada, ida, aria, ará, ava, isse, iriam, araram, endo, indo, arão, íreis, entre outras;
c) passo três: remoção do sufixo "i", se precedido de "c" e se a palavra foi alterada pelos passos um ou dois;
d) passo quatro: remoção de sufixos residuais (os, a, i, o, á, í, ó) se nenhum dos passos anteriores alterou a palavra;
e) passo cinco: remoção dos sufixos "e", "é" e "ê", tratamento do cedilha e das sílabas "gue", "gué" e "guê".

Existem outras técnicas de PLN utilizadas no processo de MT, tais como, classificação de entidades mencionadas, análise dos constituintes, correferência, nomes truncados, anáfora pronominal, sinônimos, erros ortográficos, mas não são objetos de estudo e aplicação neste trabalho.

2.5 TRABALHOS CORRELATOS

Esta seção descreve três trabalhos correlatos ao protótipo desenvolvido. O primeiro é o de Baségio (2007), que apresenta uma abordagem semiautomática para a identificação de estruturas ontológicas a partir de textos na língua portuguesa do Brasil. O segundo, de Faria e Girardi (2010), propõe um processo semiautomático para o povoamento de ontologias a partir de bases textuais. Por último, o trabalho de Silva (2004) que descreve a implementação de um sistema para extração de informações em referências bibliográficas baseado em AM. A seção também apresenta um quadro comparativo entre os trabalhos correlatos.

2.5.1 Uma abordagem semiautomática para a identificação de estruturas ontológicas a partir de textos na língua portuguesa do Brasil

Baségio (2007) definiu uma abordagem para suportar algumas fases do processo de aquisição de estruturas ontológicas, mais especificamente as fases de extração de conceitos e

relações taxonômicas, de modo a semiautomatizar os passos da construção de ontologias a partir de textos escritos na língua portuguesa do Brasil.

O resultado obtido serve como ponto de partida ao engenheiro de ontologia. Para avaliação da sua proposta, o autor desenvolveu um protótipo que incorpora mecanismos de importação de uma base textual, identificação de termos relevantes, identificação de relações taxonômicas entre estes termos e geração da estrutura ontológica em OWL (BASÉGIO, 2007, p. 59).

Segundo Baségio (2007, p. 59), a eficácia de seu protótipo depende da correta identificação das etiquetas gramaticais (substantivo, verbo, entre outras). Ele também afirma que existem ferramentas que realizam esta tarefa de forma automática, mas nenhuma delas oferece o nível de confiabilidade desejado em seu protótipo. Assim sendo, como entrada foram adotados textos já anotados linguisticamente, com as seguintes informações associadas a cada documento:

a) a palavra em seu formato original;
b) o lema da palavra original, ou seja, a palavra em sua forma singular e masculina;
c) a etiqueta gramatical da palavra (substantivo, adjetivo, entre outras).

Em seguida foram removidos os termos que não representam conceitos do domínio, foram identificados os termos compostos e as relações entre as classes e, por fim, foi realizada a criação da ontologia OWL (BASÉGIO, 2007, p. 60).

O protótipo foi utilizado num estudo de caso sobre o domínio do turismo, possibilitando a avaliação com relação a diferentes aspectos do processo de aquisição de conceitos e relações. Para a realização de um estudo de caso foram utilizados 294 documentos, contabilizando um total de 88.601 palavras. O autor afirma que nesse estudo de caso, 87,86% dos termos definidos como relevantes por sua ferramenta, foram considerados também relevantes para o especialista no domínio (BASÉGIO, 2007, p. 7).

2.5.2 Um processo semiautomático para o povoamento de ontologias a partir de fontes textuais

Faria e Girardi (2010) propuseram um processo semiautomático para o povoamento de ontologias a partir de fontes textuais utilizando técnicas de PLN e de extração de informação. O processo proposto consistia em duas fases: extração e classificação das instâncias e representação das instâncias.

A fase de extração e classificação das instâncias consiste em três tarefas: etiquetagem da base textual, construção de regras de extração e classificação e extração e classificação de

instâncias. A fase de construção de regras de extração e classificação é realizada de forma independente da primeira etapa. Ela busca a construção de regras de classificação e extração baseadas em padrões léxicos e sintáticos, além do conhecimento do domínio expresso na forma de uma ontologia (FARIA; GIRARDI, 2010, p. 4).

A fase de representação de instâncias consiste em duas tarefas: seleção e instanciação. A seleção de instâncias visa eliminar instâncias duplicadas. Para cada nova instância a ser inserida na ontologia, é realizada uma busca para ver se a mesma já não existe. Caso ela exista, será descartada, caso contrário, será inserida na ontologia. A etapa de instanciação visa a efetiva criação das instâncias. Para cada instância da ontologia, é realizada uma busca pela classe que ela pertence, e a ontologia é populada com essa nova instância (FARIA; GIRARDI, 2010, p. 6).

Um estudo de caso na área do direito de família foi conduzido para uma avaliação preliminar da efetividade do processo proposto. Para tanto, foi desenvolvido o protótipo de uma ferramenta para automatização do processo. Foi utilizado no estudo de caso a base textual denominada `FamilyJuris`, composto de 919 documentos textuais, capturados a partir do site "family.findlaw.com" (MACEDO, 2010, p. 50), contendo casos de jurisprudência do direito de família norte-americano. A ontologia `FamilyLaw` adotada no estudo de caso foi desenvolvida na ferramenta Protégé e descreve o conhecimento do direito de família. A taxa de precisão na identificação dos objetos foi de 95% (FARIA; GIRARDI, 2010, p. 5).

2.5.3 Um sistema para extração de informação em referências bibliográficas baseado em aprendizagem de máquina

A primeira etapa do trabalho de Silva (2004) foi estudar os conceitos básicos de Extração de Informação (EI) e suas principais técnicas. Essas técnicas incluem expressões regulares, autômatos finitos, técnicas de classificação de textos e modelos de Markov escondidos. Também foram analisados alguns sistemas de EI que utilizam AM.

O trabalho de Silva (2004) tem como objetivo a construção de um sistema para extrair informações a partir de textos contendo citações científicas (ou referências bibliográficas) através de uma abordagem baseada em aprendizagem automática. Dentre as diversas técnicas existentes, ele preferiu tratar o problema através de uma abordagem híbrida, que combina o uso de técnicas de classificação de textos com os modelos de Markov escondidos. Essa conclusão foi baseada em seu estudo sobre o tema (SILVA, 2004, p. 4).

Formalmente, um modelo de Markov escondido é definido como um conjunto de estados escondidos e a probabilidade de transição de um estado para outro. Os modelos de

Markov escondidos são capazes de explorar naturalmente a ocorrência de padrões em sequência no texto de entrada para classificá-los de uma só vez. Sua classificação maximiza a probabilidade de acerto para todo o conjunto de padrões e não para cada padrão isoladamente.

Segundo Silva (2004, p. 4), esta combinação mostrou resultados superiores aos obtidos usando exclusivamente as técnicas de classificação. Sua ideia básica é gerar com o uso das técnicas de classificação de textos para EI uma saída inicial para o sistema e refiná-la depois por meio de um modelo Markov escondido. Com sua abordagem híbrida, ele obteve um aumento entre 1,27 e 22,54 % na precisão. Experimentos realizados com um conjunto de teste contendo 3.000 referências resultaram em uma precisão de 87,48%.

3 DESENVOLVIMENTO DA FERRAMENTA

Neste capítulo são apresentadas as etapas do desenvolvimento de uma ferramenta para a criação de bases de conhecimento na forma de uma ontologia OWL a partir de textos não estruturados. Na seção 3.1 são enumerados os requisitos principais da ferramenta desenvolvida. A seção 3.2 apresenta sua especificação. A seção 3.3 detalha a implementação das principais técnicas e algoritmos utilizados na ferramenta desenvolvida. Por fim, a seção 3.4 apresenta o teste efetuado para a validação da ferramenta, os resultados obtidos e uma discussão sobre os mesmos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Para Pfleeger (2004, p. 115) existem dois tipos de requisitos para os sistemas: funcionais (RF) e não funcionais (RNF). Um RF descreve a interação entre o sistema e o ambiente. Ele descreve como o sistema deve se portar ao receber um estímulo do usuário. Ou seja, um RF é uma funcionalidade que o sistema disponibiliza para o seu usuário. Uma funcionalidade é descrita como um conjunto de entradas, o processamento das mesmas e as saídas. Um RNF não está ligado diretamente às funcionalidades do sistema, pois não descreve o que o sistema deve fazer e sim como deve fazer. Ele denota uma restrição no sistema que limita as opções para criar uma solução para o problema.

O Quadro 2 exibe os RNF elencados para a ferramenta.

Quadro 2 – RNF da ferramenta

Requisitos Não Funcionais
RNF 01: A ferramenta Enterprise Architect (EA) na versão 7.5 deverá ser utilizada para realizar a especificação.
RNF 02: A ferramenta deverá ser implementada utilizando a linguagem de programação Java na versão 7.0.
RNF 03: A ferramenta deverá ser implementada utilizando o ambiente de desenvolvimento Eclipse na versão Juno.
RNF 04: A ferramenta deverá utilizar ontologias OWL como formalismo de RC utilizado pela base de conhecimento.
RNF 05: A ferramenta deverá utilizar a biblioteca Cogroo (USP, 2011) na versão 4.0 para realizar a análise morfológica da base textual de trabalho.
RNF 06: A ferramenta deverá utilizar a biblioteca PTStemmer (OLIVEIRA, 2010) na versão 2.0 para lematizar as palavras contidas na base textual de trabalho atual.
RNF 07: A ferramenta analisará apenas documentos escritos em língua portuguesa.
RNF 08: A ferramenta deverá gerar uma ontologia OWL fazendo uso das *tags* definidas pelo Protégé, sendo assim possível manipulá-la através do mesmo.

O Quadro 3 exibe os RF elencados para a ferramenta.

Quadro 3 - RF da ferramenta

Requisitos Funcionais
RF 01: A ferramenta deverá permitir a manutenção de bases textuais.
RF 02: A ferramenta deverá permitir a definição da base textual de trabalho.
RF 03: A ferramenta deverá realizar a coleta de documentos com formato `txt`.
RF 04: A ferramenta deverá popular a base textual de trabalho com os documentos coletados.
RF 05: A ferramenta deverá exibir a estrutura da base textual.
RF 06: A ferramenta deverá exibir a base textual de trabalho anotada morfologicamente.
RF 07: A ferramenta deverá exibir a estrutura do arquivo índice invertido criado a partir da base textual de trabalho.
RF 08: A ferramenta deverá consultar documentos a partir de uma palavra fazendo uso do arquivo de índice invertido criado.
RF 09: A ferramenta deverá exibir a base textual de trabalho sem as *stopwords*.
RF 10: A ferramenta deverá exibir a base textual lematizada.
RF 11: A ferramenta deverá descobrir conhecimento da base textual de trabalho.
RF 12: A ferramenta deverá exibir as estruturas ontológicas (classes e seus relacionamentos) a partir do conhecimento extraído.
RF 13: A ferramenta deverá criar uma base de conhecimento com o conhecimento extraído.

3.2 ESPECIFICAÇÃO

A especificação deste trabalho foi desenvolvida fazendo uso da ferramenta EA na versão 7.5. Foram utilizados os conceitos do paradigma de análise e modelagem de sistemas orientados a objeto, por meios de diagramas da *Unified Modeling Language* (UML).

Segundo Bezerra (2002, p. 45), a UML é uma linguagem visual para modelar sistemas orientados a objetos. A UML é constituída de elementos gráficos (visuais) utilizados na modelagem, que permitem representar conceitos do paradigma da orientação a objetos. Para Larman (2002, p. 32), a notação UML, apesar de padronizar a criação de artefatos, não define um processo de desenvolvimento de software. Segundo o autor, esse fato aumenta a sua aceitação, pois a escolha de um processo de desenvolvimento de software depende de diversos fatores, tais como: equipe, pesquisa, domínio do problema, ferramentas, entre outros.

3.2.1 Diagrama de casos de uso

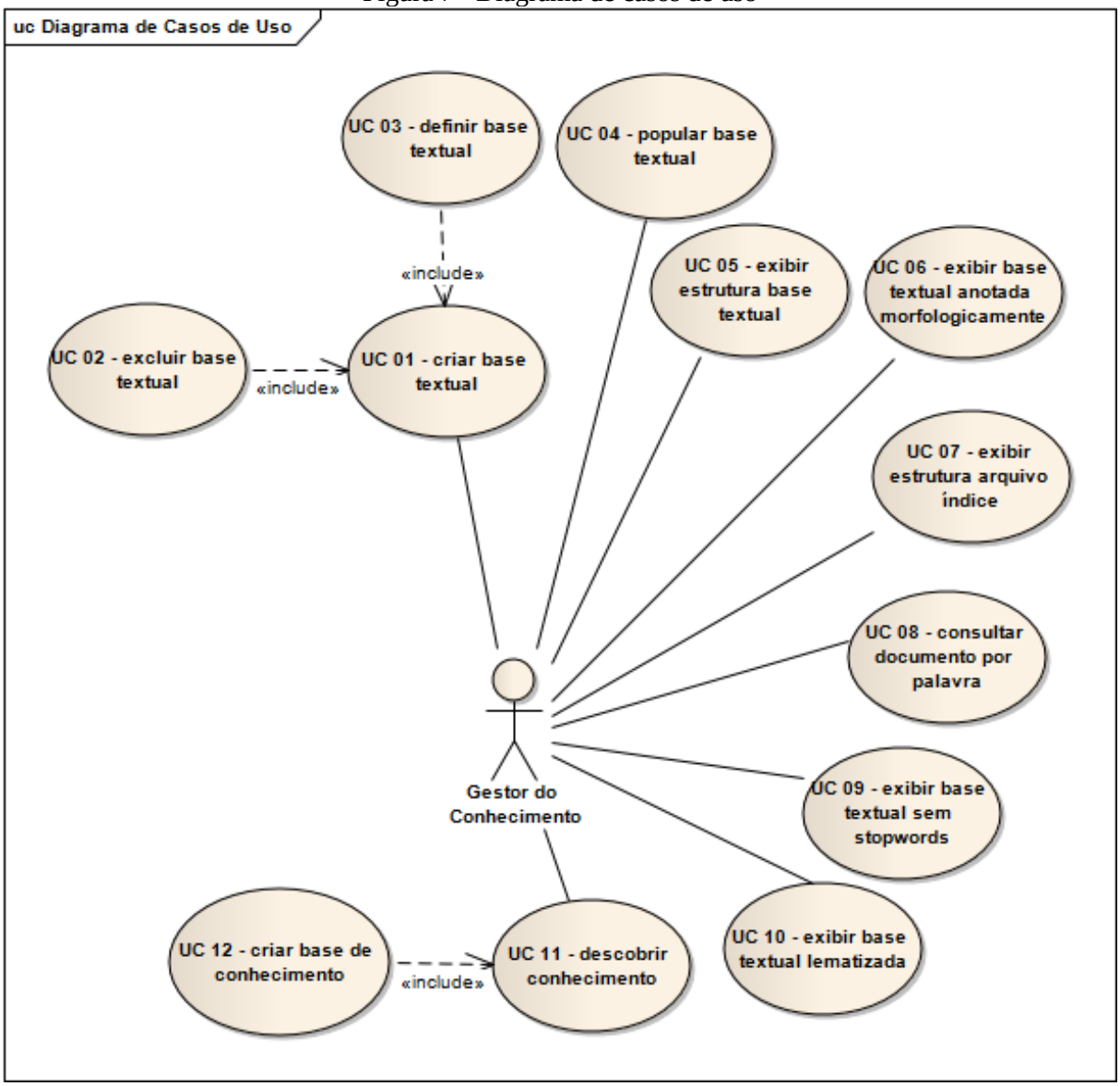
Conforme Bezerra (2002, p. 47), nos casos de uso não se considera comportamentos internos de um sistema, representando apenas agentes externos (atores). Para Larman (2002, p. 38), o diagrama de casos de uso ilustra a relação entre atores e os casos de uso, facilitando sua compreensão. Casos de uso são representados por elipses, atores por homens palito e as flechas indicam o fluxo de informações e as ações que ocorrem entre elas.

A Figura 7 ilustra o diagrama de casos de uso da ferramenta desenvolvida. Nele existe o ator `Gestor do Conhecimento` que fará uso de todas as funcionalidades da ferramenta.

Também foram identificados quatorze casos de uso, cada qual representando uma funcionalidade da ferramenta.

Nota-se a dependência entre alguns casos de uso. Tanto para a execução do caso de uso `UC 02 - excluir base textual`, quanto do `UC 03 - definir base textual` é necessário que o caso de uso `UC 01 - criar base textual` tenha sido executado, existindo assim no mínimo uma base textual criada. Existe também dependência entre os casos de uso `UC 12 - criar base de conhecimento` e `UC 11 - descobrir conhecimento`, sendo necessário primeiramente descobrir conhecimento, para então formalizá-lo.

Figura 7 - Diagrama de casos de uso



O primeiro passo de todo o processo é a criação da base textual atual de trabalho, ou seja, um diretório em disco que armazenará todos os documentos utilizados no processo de MT. Todo o processo é descrito no Quadro 4.

Quadro 4 - UC 01 - criar base textual

UC 01 – criar base textual	
Pré-condições	Ferramenta deve estar em execução.
Cenário principal	1) O gestor do conhecimento seleciona a opção responsável por criar uma nova base textual. 2) A ferramenta solicita o nome da base textual. 3) O gestor do conhecimento informa o nome da base textual. 4) A ferramenta cria um diretório com o nome informado dentro do diretório que armazena todas as bases textuais. 5) A ferramenta define o diretório criado como a base textual de trabalho. 6) A ferramenta exibe uma mensagem informando que a base textual foi criada com sucesso.
Exceção 1	No passo 4 do cenário principal, caso ocorra algum erro ao criar o diretório, a operação é abortada e uma mensagem de erro é exibida.
Pós-condições	Criado diretório em disco que irá armazenar os documentos da base textual.
RF associado	RF 01.
RNF associado	-

Após a criação da base textual pode-se realizar duas operações sobre elas que não fazem parte diretamente do processo de MT. Essas operações são: excluir uma base textual e definir a base de trabalho. A operação de exclusão deve ser implementada, pois uma base textual pode se tornar desnecessária ao `Gestor do Conhecimento`. Quando uma base textual é criada ela passa a ser a base atual de trabalho, como descrito anteriormente. Porém, caso se deseje trabalhar com outra base já existente, deve-se implementar uma funcionalidade que permita a troca da base de trabalho, por outra base textual. As referidas operações são descritas no Quadro 5 e Quadro 6 respectivamente.

Quadro 5 - UC 02 – excluir base textual

UC 02 – excluir base textual	
Pré-condições	Deve existir pelo menos uma base textual criada.
Cenário principal	1) O gestor do conhecimento seleciona a opção responsável por excluir uma base uma textual existente. 2) A ferramenta solicita a seleção da base textual a ser excluída. 3) O gestor do conhecimento seleciona a base textual que deseja excluir. 4) A ferramenta pede a confirmação da exclusão. 5) O gestor do conhecimento confirma a exclusão. 6) A ferramenta exclui o diretório selecionado do disco, juntamente com todos os arquivos textuais contidos pelo mesmo. 7) A ferramenta exibe uma mensagem informando que a base textual foi excluída com sucesso.
Exceção 1	No passo 6 do cenário principal, caso ocorra algum erro ao excluir o diretório, a operação é abortada e uma mensagem de erro é exibida.
Pós-condições	Diretório excluído do disco.
RF associado	RF 01.

RNF associado	-

Quadro 6 - UC 03 – definir base textual

UC 03 – definir base textual	
Pré-condições	Deve existir pelo menos uma base textual criada.
Cenário principal	1) O gestor do conhecimento seleciona a opção responsável por alterar a base textual. 2) A ferramenta solicita a seleção da base textual a ser definida como a nova base textual de trabalho. 3) O gestor do conhecimento seleciona a base textual. 4) A ferramenta define o diretório selecionado como a base textual de trabalho. 5) A ferramenta exibe uma mensagem informando que a base textual foi alterada com sucesso.
Pós-condições	Diretório definido como a nova base textual de trabalho.
RF associado	RF 02.
RNF associado	-

Após a criação ou definição da base textual de trabalho é necessário populá-la. Para isso são necessárias duas etapas: a coleta dos documentos e a cópia dos mesmos para a base textual. Essa é uma das etapas do processo de MT e está descrita no Quadro 7.

Quadro 7 - UC 04 – popular base textual

UC 04 – popular base textual	
Pré-condições	A base textual de trabalho foi definida. Documentos a serem coletados devem estar disponíveis no disco rígido do computador em um mesmo diretório.
Cenário principal	1) O gestor do conhecimento seleciona a opção responsável por popular a base textual. 2) A ferramenta solicita a seleção do diretório onde os arquivos `txt` a serem coletados estão armazenados. 3) O gestor do conhecimento seleciona o diretório. 4) A ferramenta realiza uma busca em profundidade no diretório selecionado, armazenando em uma lista os arquivos `txt` encontrados. 5) A ferramenta exibe as seguintes informações dos documentos encontrados: nome, caminho e tamanho (em bytes). 6) O gestor do conhecimento seleciona a opção para popular a base textual. 7) A ferramenta copia todos os documentos para a base de trabalho atual. 8) A ferramenta exibe uma mensagem informando que a coleta foi realizada com sucesso.
Exceção 1	No passo 4 do cenário principal, caso não exista nenhum documento `txt` no diretório, uma mensagem informando que não existem documentos neste formato é exibida.
Exceção 2	No passo 7 do cenário principal, caso ocorra algum erro ao copiar algum documento, a operação é abortada e uma mensagem de erro é exibida.
Pós-condições	Os documentos `txt` encontrados no diretório selecionado são copiados para a base textual.

RF associado	RF 03, RF 04.
RNF associado	-

Finalizada a coleta dos documentos, dá-se início a etapa de pré-processamento. Os casos de uso `UC 05 - exibir estrutura base textual`, `UC 06 - exibir base textual anotada morfologicamente`, `UC 09 - exibir base textual sem stopwords` e `UC 10 - exibir base textual lematizada` são funcionalidades contidas pela etapa de pré-processamento e são descritas a seguir no Quadro 8, Quadro 9, Quadro 10 e Quadro 11, respectivamente.

Quadro 8 - UC 05 - exibir estrutura base textual

UC 05 - exibir estrutura base textual	
Pré-condições	Base textual de trabalho populada.
Cenário principal	1) O gestor do conhecimento seleciona a opção responsável por exibir a estrutura da base textual. 2) A ferramenta exibe uma árvore composta de um nó com o nome "Base Textual". 3) O gestor do conhecimento seleciona a opção para mostrar a estrutura da base textual. 4) A ferramenta percorre a base textual de trabalho, inserindo cada documento contido nela em uma lista. 5) A ferramenta percorre essa lista, abrindo cada documento e copiando seu conteúdo para a memória. 6) A ferramenta particiona cada documento em frases. 7) A ferramenta particiona cada frase em palavras. 8) A ferramenta remove os caracteres especiais. 9) A ferramenta percorre a lista com todos esses documentos estruturados e monta a árvore da seguinte forma: cada documento será um nó filho de "Base Textual"; cada frase do documento será um nó filho de documento; cada palavra será um nó filho de frase.
Exceção 1	No passo 5 do cenário principal, caso ocorra algum erro ao ler o conteúdo do arquivo para a memória, a operação é abortada e uma mensagem de erro é exibida.
Exceção 2	No passo 8 do cenário principal, caso ocorra algum erro ao ler o conteúdo do arquivo que contém os caracteres especiais para a memória, a operação é abortada e uma mensagem de erro é exibida.
Pós-condições	Todos os documentos estruturados por frase e cada frase dividida pelas palavras.
RF associado	RF 05.
RNF associado	-

Quadro 9 – UC 06 - exibir base textual anotada morfologicamente

UC 06 - exibir base textual anotada morfologicamente	
Pré-condições	Base textual de trabalho populada.
Cenário principal	1) O gestor do conhecimento seleciona a opção responsável por analisar morfologicamente a base textual. 2) O gestor do conhecimento seleciona a opção para analisar a base textual.

	3) A ferramenta percorre a base textual de trabalho, inserindo cada documento contido nela em uma lista. 4) A ferramenta percorre essa lista, abrindo cada documento e copiando seu conteúdo para a memória. 5) A ferramenta particiona cada documento em frases. 6) A ferramenta particiona cada frase em palavras. 7) A ferramenta realiza a análise morfológica de todas as palavras. 8) A ferramenta remove os caracteres especiais. 9) A ferramenta exibe as seguintes informações dos documentos: lexema, lema, classe gramatical e inflexões.
Exceção 1	No passo 4 do cenário principal, caso ocorra algum erro ao ler o conteúdo do arquivo para a memória, a operação é abortada e uma mensagem de erro é exibida.
Exceção 2	No passo 7 do cenário principal, caso ocorra algum erro ao anotar morfologicamente as palavras, a operação é abortada e uma mensagem de erro é exibida.
Exceção 3	No passo 8 do cenário principal, caso ocorra algum erro ao ler o conteúdo do arquivo que contém os caracteres especiais para a memória, a operação é abortada e uma mensagem de erro é exibida.
Pós-condições	Todas as palavras anotadas morfologicamente.
RF associado	RF 06.
RNF associado	RNF 05, RNF 07.

Quadro 10 – UC 09 - exibir base textual sem *stopwords*

UC 09 - exibir base textual sem *stopwords*	
Pré-condições	Base textual de trabalho populada.
Cenário principal	1) O gestor do conhecimento seleciona a opção responsável por exibir a base textual sem as *stopwords*. 2) A ferramenta exibe uma árvore composta de um nó com o nome "Base Textual". 3) O gestor do conhecimento seleciona a opção para exibir a base textual sem *stopwords*. 4) A ferramenta percorre a base textual de trabalho, inserindo cada documento contido nela em uma lista. 5) A ferramenta percorre essa lista, abrindo cada documento e copiando seu conteúdo para a memória. 6) A ferramenta particiona cada documento em frases. 7) A ferramenta particiona cada frase em palavras. 8) A ferramenta remove os caracteres especiais. 9) A ferramenta remove as *stopwords*. 10) A ferramenta percorre a lista com todos esses documentos estruturados e monta a árvore da seguinte forma: cada documento será um nó filho de "Base Textual"; cada frase do documento será um nó filho de documento; cada palavra será um nó filho de frase.
Exceção 1	No passo 5 do cenário principal, caso ocorra algum erro ao ler o conteúdo do arquivo para a memória, a operação é abortada e uma mensagem de erro é exibida.
Exceção 2	No passo 8 do cenário principal, caso ocorra algum erro ao ler o conteúdo do arquivo que contém os caracteres especiais para a memória, a operação é abortada e uma mensagem de erro é exibida.

Exceção 3	No passo 9 do cenário principal, caso ocorra algum erro ao ler o conteúdo do arquivo que contém as *stopwords* para a memória, a operação é abortada e uma mensagem de erro é exibida.
Pós-condições	Base textual sem *stopwords*.
RF associado	RF 09.
RNF associado	-

Quadro 11 - UC 10 - exibir base textual lematizada

UC 10 - exibir base textual lematizada	
Pré-condições	Base textual de trabalho populada.
Cenário principal	1) O gestor do conhecimento seleciona a opção responsável por lematizar a base textual. 2) A ferramenta exibe uma árvore composta de um nó com o nome "Base Textual". 3) O gestor do conhecimento seleciona a opção para lematizar a base textual. 4) A ferramenta percorre a base textual de trabalho, inserindo cada documento contido nela em uma lista. 5) A ferramenta percorre essa lista, abrindo cada documento e copiando seu conteúdo para a memória. 6) A ferramenta particiona cada documento em frases. 7) A ferramenta particiona cada frase em palavras. 8) A ferramenta remove os caracteres especiais. 9) A ferramenta remove as *stopwords*. 10) A ferramenta lematiza as palavras. 11) A ferramenta percorre a lista com todos esses documentos estruturados e monta a árvore da seguinte forma: cada documento será um nó filho de "Base Textual"; cada frase do documento será um nó filho de documento; cada palavra será um nó filho de frase.
Exceção 1	No passo 5 do cenário principal, caso ocorra algum erro ao ler o conteúdo do arquivo para a memória, a operação é abortada e uma mensagem de erro é exibida.
Exceção 2	No passo 8 do cenário principal, caso ocorra algum erro ao ler o conteúdo do arquivo que contém os caracteres especiais para a memória, a operação é abortada e uma mensagem de erro é exibida.
Exceção 3	No passo 9 do cenário principal, caso ocorra algum erro ao ler o conteúdo do arquivo que contém as *stopwords* para a memória, a operação é abortada e uma mensagem de erro é exibida.
Exceção 4	No passo 10 do cenário principal, caso ocorra algum erro ao lematizar as palavras, a operação é abortada e uma mensagem de erro é exibida.
Pós-condições	Todas as palavras lematizadas.
RF associado	RF 10.
RNF associado	RNF 06, RNF 07.

Após a etapa de pré-processamento se inicia a indexação, visando melhorar o desempenho de consultas por documentos. O caso de uso `UC 07 - exibir estrutura arquivo índice` é responsável por indexar a base textual, criando assim um arquivo de índice invertido. Após a criação desse arquivo, sua taxonomia é exibida na forma de uma árvore. O caso de uso `UC 08 - consultar documento por palavra` faz uso do arquivo de

índice invertido para responder a seguinte consulta: quais documentos contêm determinada palavra? Esses casos de uso são descritos com mais ênfase no Quadro 12 e Quadro 13.

Quadro 12 - UC 07 - exibir estrutura arquivo índice

UC 07 - exibir estrutura arquivo índice	
Pré-condições	Pré-processamento concluído.
Cenário principal	1) O gestor do conhecimento seleciona a opção responsável por indexar a base textual. 2) A ferramenta exibe uma árvore composta de um nó com o nome "Base Indexada". 3) O gestor do conhecimento seleciona a opção para indexar a base textual. 4) A ferramenta cria o arquivo de índice invertido. 5) A ferramenta percorre todas as palavras resultantes do processo de pré-processamento, inserindo-as no arquivo de índice invertido, juntamente com uma lista dos documentos que contém aquela palavra. 6) A ferramenta grava o arquivo de índice invertido no disco. 7) A ferramenta percorre o arquivo de índice invertido e monta a árvore da seguinte forma: cada palavra será um nó filho de "Base Indexada"; cada documento da lista de documentos que contém aquela palavra será um nó filho de palavra.
Exceção 1	No passo 6 do cenário principal, caso ocorra algum erro ao gravar o arquivo de índice invertido no disco, a operação é abortada e uma mensagem de erro é exibida.
Pós-condições	Arquivo de índice invertido gravado em disco.
RF associado	RF 07.
RNF associado	-

Quadro 13 - UC 08 - consultar documento por palavra

UC 08 - consultar documento por palavra	
Pré-condições	Pré-processamento concluído. Arquivo de índice invertido criado.
Cenário principal	1) O gestor do conhecimento seleciona a opção responsável por pesquisar documento através de palavra. 2) A ferramenta exibe uma árvore composta de um nó com o nome "Documentos contém Palavra". 3) O gestor do conhecimento informa a palavra que deseja pesquisar e seleciona a opção para pesquisar no arquivo índice. 4) A ferramenta lematiza a palavra que será utilizada na busca. 5) A ferramenta abre o arquivo de índice invertido e copia seu conteúdo para a memória. 6) A ferramenta verifica se a palavra informada existe no arquivo de índice invertido. 7) A ferramenta recupera a lista de documentos que contém a palavra pesquisada. 8) A ferramenta monta a árvore da seguinte forma: a palavra lematizada será um nó filho de "Documentos contém Palavra"; cada documento da lista de documentos que contém aquela palavra será um nó filho de palavra.
Exceção 1	No passo 4 do cenário principal, caso ocorra algum erro ao lematizar a palavra a ser pesquisada, a operação é abortada e uma mensagem de

	erro é exibida.
Exceção 2	No passo 5 do cenário principal, caso ocorra algum erro ao ler o conteúdo do arquivo de índice invertido para a memória, a operação é abortada e uma mensagem de erro é exibida.
Exceção 3	No passo 6 do cenário principal, caso a palavra pesquisada não conste no arquivo de índice invertido, uma mensagem informando esse fato é exibida.
Pós-condições	-
RF associado	RF 08.
RNF associado	RNF 06.

Finalizada a indexação, pode-se efetivamente realizar a descoberta e formalização do conhecimento presente na base textual atual de trabalho. Esse processo é descrito através de dois casos de usos: `UC 11 - descobrir conhecimento` e `UC 12 - criar base de conhecimento`, descritos mais profundamente no Quadro 14 e Quadro 15. O primeiro descreve a etapa de mineração do processo de MT, onde serão aplicados os algoritmos de descoberta de conhecimento, para identificar as classes e relacionamento entre as mesmas. O segundo tem como entrada esse conhecimento descoberto e cria uma base de conhecimento, fazendo uso de uma ontologia escrita com a linguagem OWL como formalismo de RC.

Quadro 14 - UC 11 - descobrir conhecimento

UC 11 - descobrir conhecimento	
Pré-condições	Indexação concluída.
Cenário principal	1) O gestor do conhecimento seleciona a opção responsável por descobrir o conhecimento. 2) A ferramenta exibe uma árvore composta de um nó com o nome “Thing”. 3) O gestor do conhecimento seleciona a opção para mostrar o conhecimento descoberto. 4) A ferramenta cria uma ontologia. 5) A ferramenta percorre a lista que contém todos os documentos, acessando todas as frases dos mesmos. 6) A ferramenta percorre cada frase, analisando todas suas palavras. 7) A ferramenta verifica se a palavra analisada é um substantivo ou nome próprio, caso isso seja verdadeiro, ela é inserida na ontologia como uma classe. Caso a palavra for um verbo, ela é identificada como um relacionamento. Se esse relacionamento estiver entre duas classes, então ele é inserido na ontologia. 8) A ferramenta percorre a ontologia criada montando a árvore da seguinte forma: cada classe será um nó filho de “Thing”; cada relacionamento que parte da classe será um nó filho da mesma; cada classe em que esse relacionamento chega será um nó filho do mesmo.
Pós-condições	Ontologia representando o conhecimento descoberto é criada.
RF associado	RF 11, RF12.
RNF associado	-

Quadro 15 - UC 14 - criar base de conhecimento

UC 14 - criar base de conhecimento	
Pré-condições	Ontologia criada.
Cenário principal	1) O gestor do conhecimento seleciona a opção responsável por criar a ontologia OWL. 2) O gestor do conhecimento seleciona a opção para exibir a ontologia OWL. 3) A ferramenta cria uma variável textual para armazenar as *tags* que formam o arquivo OWL. 4) A ferramenta percorre a ontologia acessando todos os seus componentes. Para cada classe é concatenada a *tag* que a representa na variável textual. A partir da classe são acessados todos os seus relacionamentos, sendo concatenadas as *tags* que representam o relacionamento, seu domínio e escopo. 5) A ferramenta grava um arquivo em disco, com o conteúdo da variável textual, representando a base de conhecimento criada. Esse arquivo possui a extensão `owl`.
Exceção 1	No passo 5 do cenário principal, caso ocorra algum erro ao gravar o arquivo em disco, a operação é abortada e uma mensagem de erro é exibida.
Pós-condições	Base de conhecimento na forma de uma ontologia OWL criada e gravada em disco.
RF associado	RF 13.
RNF associado	RNF 04, RNF 08.

3.2.2 Diagrama de pacotes

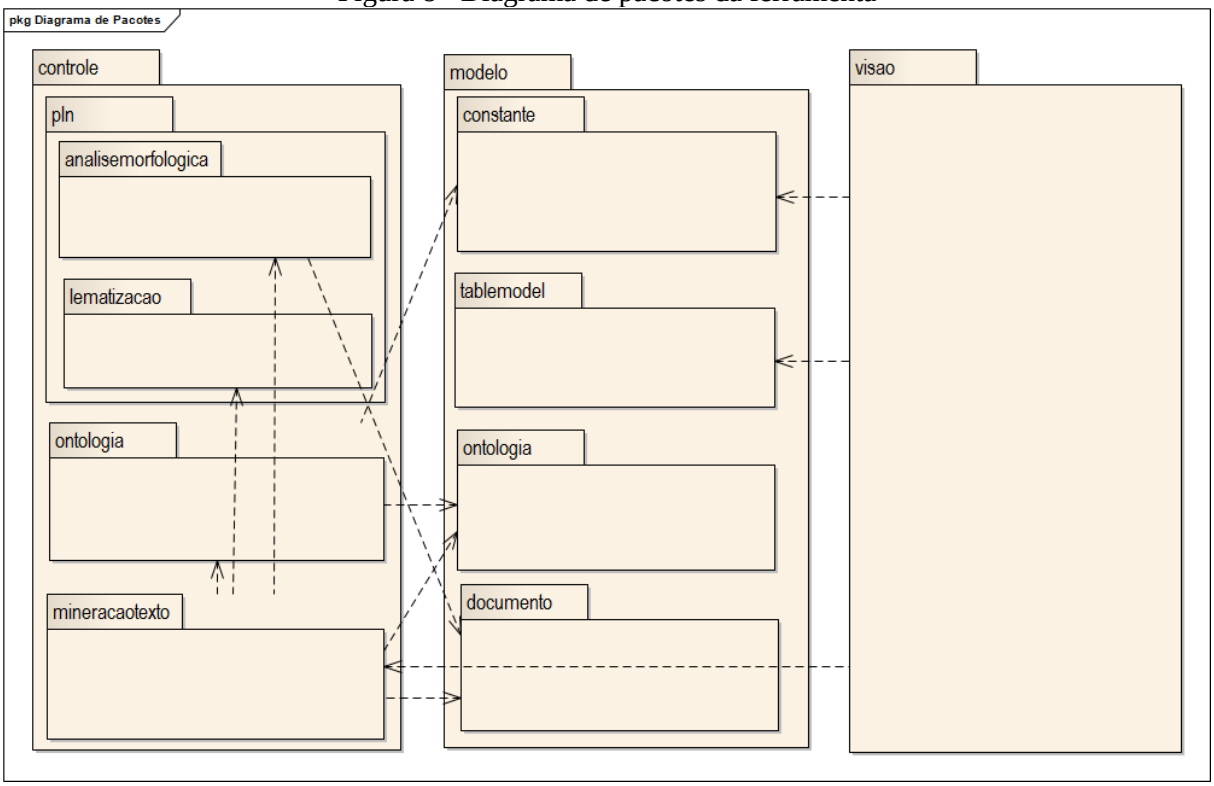
Para Pfleeger (2004, p. 50), o diagrama de pacotes descreve pedaços do sistema divididos em agrupamentos lógicos chamados de pacote. Um pacote é utilizado para agrupar elementos que guardam estreita relação entre si, sendo mais comumente utilizado para agrupar classes de um determinado domínio. Os pacotes se relacionam entre si através dos elementos contidos neles, estabelecendo assim relações de dependência.

O uso do diagrama de pacotes na modelagem do sistema ajuda a entender com mais facilidade a sua arquitetura e como os módulos se comunicam. Por isso, o sistema foi implementado através do padrão de projeto *Model View Controller* (MVC). Este consiste em dividir o software em três principais pacotes: *model* (modelo), *view* (visão) e *controller* (controle). No primeiro pacote estão inseridas as classes de modelo, sendo as classes que modelam os conceitos presentes no domínio da respectiva aplicação. O segundo pacote contém as classes de visão, ou seja, as classes que servem de interface entre o usuário e a aplicação. Por fim, o terceiro pacote armazena as classes de controle, que são aquelas responsáveis pelo fluxo do processamento da aplicação.

A Figura 8 ilustra a estrutura dos pacotes da ferramenta. Seguindo o padrão MVC, são criados três pacotes principais: `controle`, `modelo` e `visão`. O pacote `controle` se divide em

três pacotes: `pln`, `ontologia` e `mineraçãotexto`. O primeiro contém as classes responsáveis por implementar os algoritmos de PLN adotados pela ferramenta, sendo dividido em dois pacotes: `analisemorfologica` e `lematizacao`. O segundo armazena a classe responsável pela criação do arquivo OWL. O terceiro reúne as classes responsáveis pelo processo de MT, sendo considerado o principal pacote da ferramenta. O pacote `modelo` é composto de quatro pacotes: `constante`, `tablemodel`, `ontologia` e `documento`. No pacote `constante` estão reunidas as classes que armazenam valores constantes da ferramenta como, por exemplo, as *tags* utilizadas para criar o arquivo OWL. O pacote `tablemodel` contém as classes que são utilizadas como modelos de dados das tabelas presentes nas interfaces gráficas da ferramenta. O pacote `ontologia` armazena as classes que modelam os componentes de uma ontologia. No pacote `documento` estão reunidas as classes que modelam os documentos utilizados pela ferramenta. Por fim, No pacote `visao` estão alocadas as interfaces gráficas disponibilizadas pela ferramenta.

Figura 8 - Diagrama de pacotes da ferramenta



Na Figura 8 também podem ser observados os relacionamentos entre os pacotes. Como descrito anteriormente, o pacote `controle.mineracaotexto` é considerado o principal pacote da ferramenta. Esse fato é evidenciado analisando suas relações, sendo que o mesmo se relaciona com seis pacotes diferentes. Ele utiliza classes dos pacotes `controle.pln.analisemorfologica`, `controle.pln.lematizacao`, `controle.ontologia`,

`modelo.documento` e `modelo.ontologia` para realizar a descoberta e formalização do conhecimento. O pacote `controle.ontologia` utiliza uma classe do pacote `modelo.constante` para criar o arquivo OWL a partir do conhecimento descoberto contido nos objetos das classes do pacote `modelo.ontologia`. As classes do pacote `controle.pln.analisemorfologica` utilizam uma classe do pacote `modelo.documento` como entrada e outra como saída do seu processamento. Por fim, as interfaces gráficas do pacote `visao` fazem uso das classes do pacote `modelo.constante` para a formatação de alguns de seus textos, do pacote `modelo.tablemodel` como modelo de dados de suas tabelas e do `controle.mineracaotexto` para realizar o processo de MT.

3.2.3 Diagrama de classes

De acordo com Pfleeger (2004, p. 61), o diagrama de classes é utilizado para representar a estrutura estática de um sistema, sendo classificado como um diagrama estrutural. Ele descreve as classes e demais entidades que compõem o sistema (interfaces, enumerações, entre outras), juntamente com os relacionamentos entre elas. Ainda segundo o autor, o diagrama de classes é um dos mais importantes da UML, servindo como base para a construção de outros diagramas.

A seguir, são discutidos os diagramas de classe dos principais pacotes. A Figura 9 apresenta o diagrama de classes do pacote `modelo.documento`. Nele, pode-se verificar a existência de duas classes que modelam documentos: `DocumentoTextual` e `DocumentoProcessado`. A primeira classe modela um documento textual. Os objetos dessa classe são criados com o conteúdo dos documentos textuais presentes na base textual de trabalho. A segunda representa um documento textual que foi submetido ao processo de análise morfológica. Os objetos dessa classe possuem uma lista de objetos da classe `Sentence`. Esta representa uma sentença (frase) analisada morfologicamente, sendo composta de uma lista de objetos da classe `Token`, que modela uma palavra. As classes `Sentence` e `Token` fazem parte da *Application Programming Interface* (API) Cogroo (USP, 2011).

Na Figura 10 é exibido o diagrama de classes do pacote `modelo.ontologia`. Nele estão reunidas as classes responsáveis por representar os ferramentais utilizados para modelar o conhecimento através de uma ontologia. A classe `Ontologia` representa uma ontologia sobre um determinado domínio do conhecimento. Ela é composta por vários objetos de `Classe`, cada um representando um conceito do domínio da ontologia. Cada objeto `Classe`

possui uma lista de `Relacionamentos`. Cada objeto de `Relacionamento` dessa lista, representa o relacionamento entre duas classes (domínio e escopo).

Figura 9 - Diagrama de classes do pacote `modelo.documento`

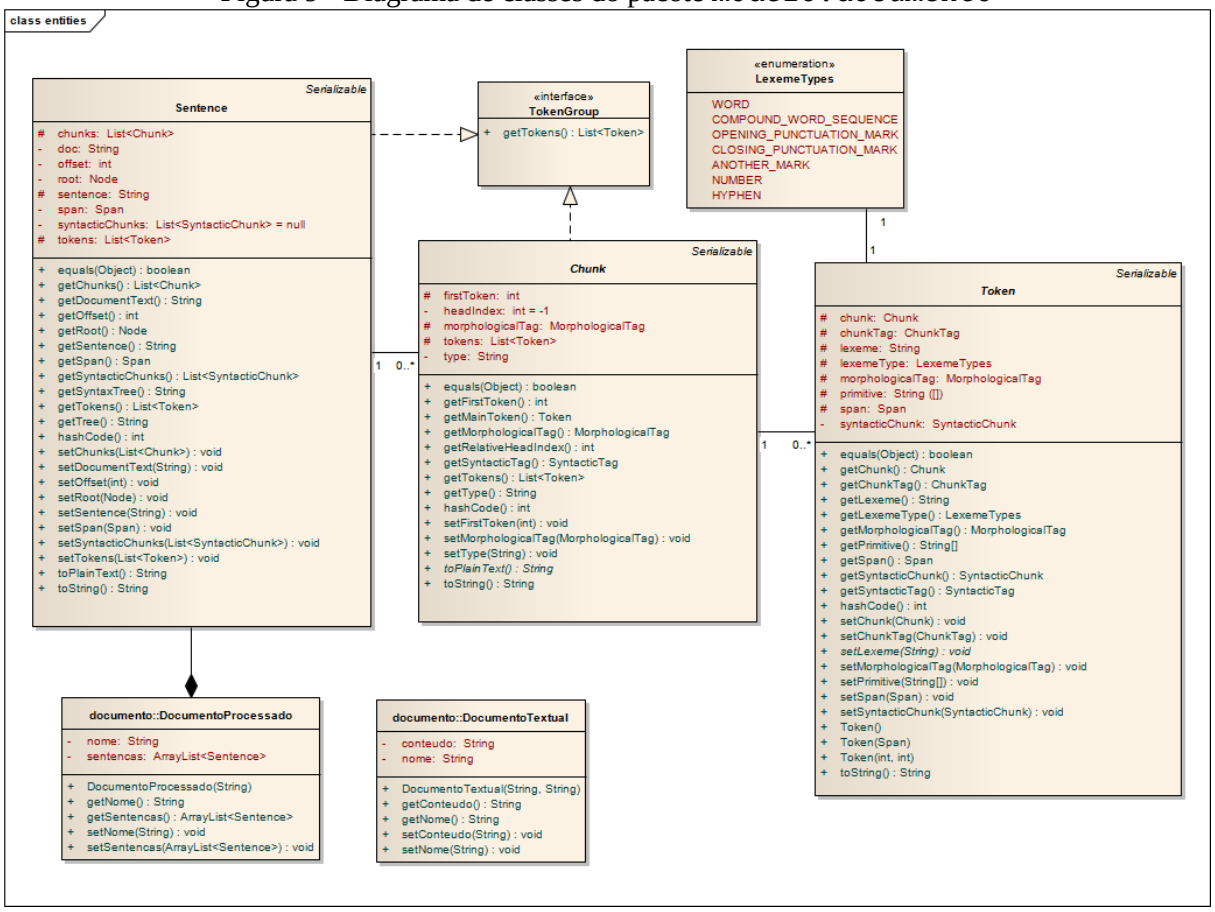
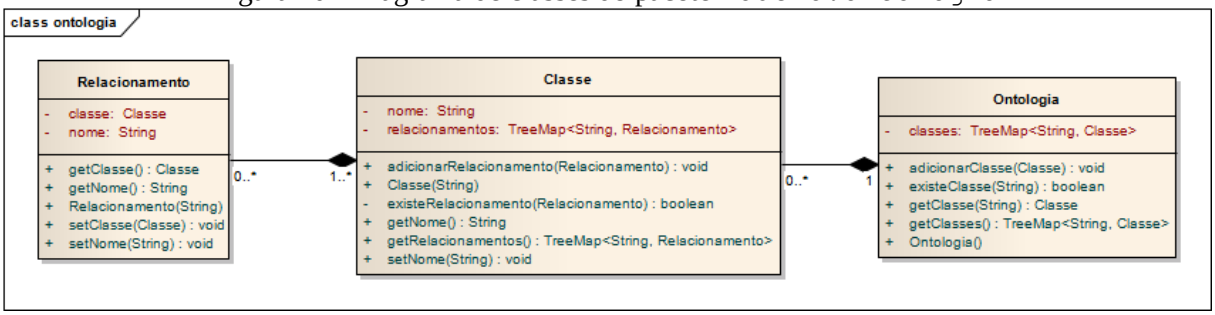


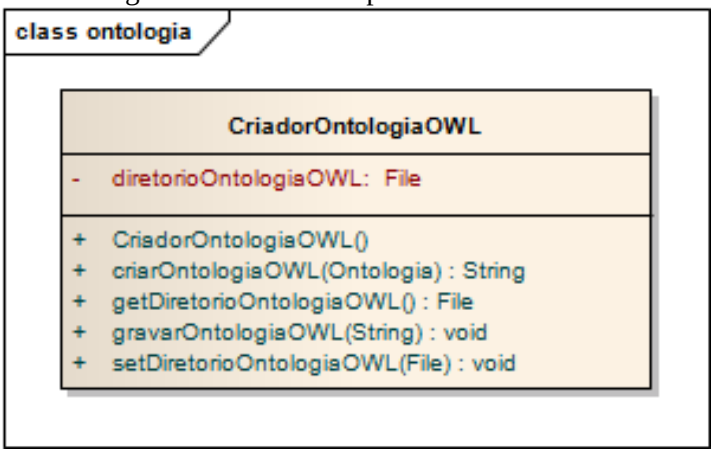
Figura 10 - Diagrama de classes do pacote `modelo.ontologia`



O diagrama de classes do pacote `controle.ontologia` é formado pela classe `CriadorOntologiaOWL`, sendo ilustrado pela Figura 11. Através dela, é criado o arquivo OWL que representa o conhecimento presente na base textual de trabalho. O método `criarOntologiaOWL` é responsável por implementar essa funcionalidade. Ele recebe como entrada um objeto da classe `Ontologia` contendo todo o conhecimento que foi descoberto através do processo de MT. Após isso, acontece uma iteração sobre todas as classes e relacionamentos gerando suas respectivas *tags* OWL, gerando o arquivo OWL que será

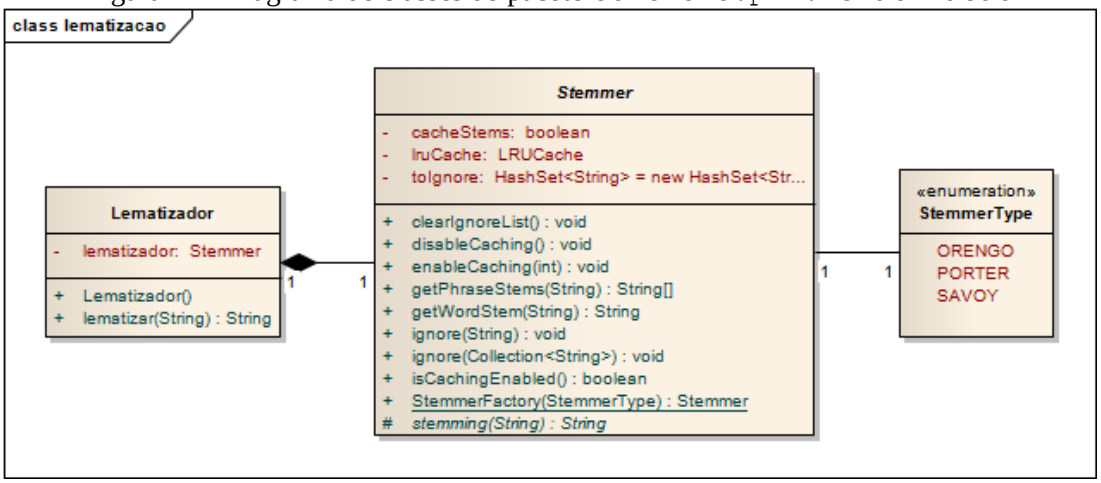
gravado em disco. Ao gravar-se o arquivo `.owl` no disco, configura-se a criação da base de conhecimento.

Figura 11 - Diagrama de classes do pacote `controle.ontologia`



O diagrama de classes do pacote `controle.pln.lematizacao` é descrito através da Figura 12. A classe `Stemmer` utiliza a enumeração `StemmerType` para definir qual dos algoritmos de lematização que a mesma implementa será utilizado. Os algoritmos implementados são: Orengo, Porter e Savoy. Tanto a classe `Stemmer` quanto a enumeração `StemmerType` fazem parte da API PTStemmer (OLIVEIRA, 2010). A classe `Lematizador` funciona como mediadora da troca de mensagens entre o PTStemmer e os demais módulos da ferramenta. Através de seu método `lematizar` é realizado o processo de lematização.

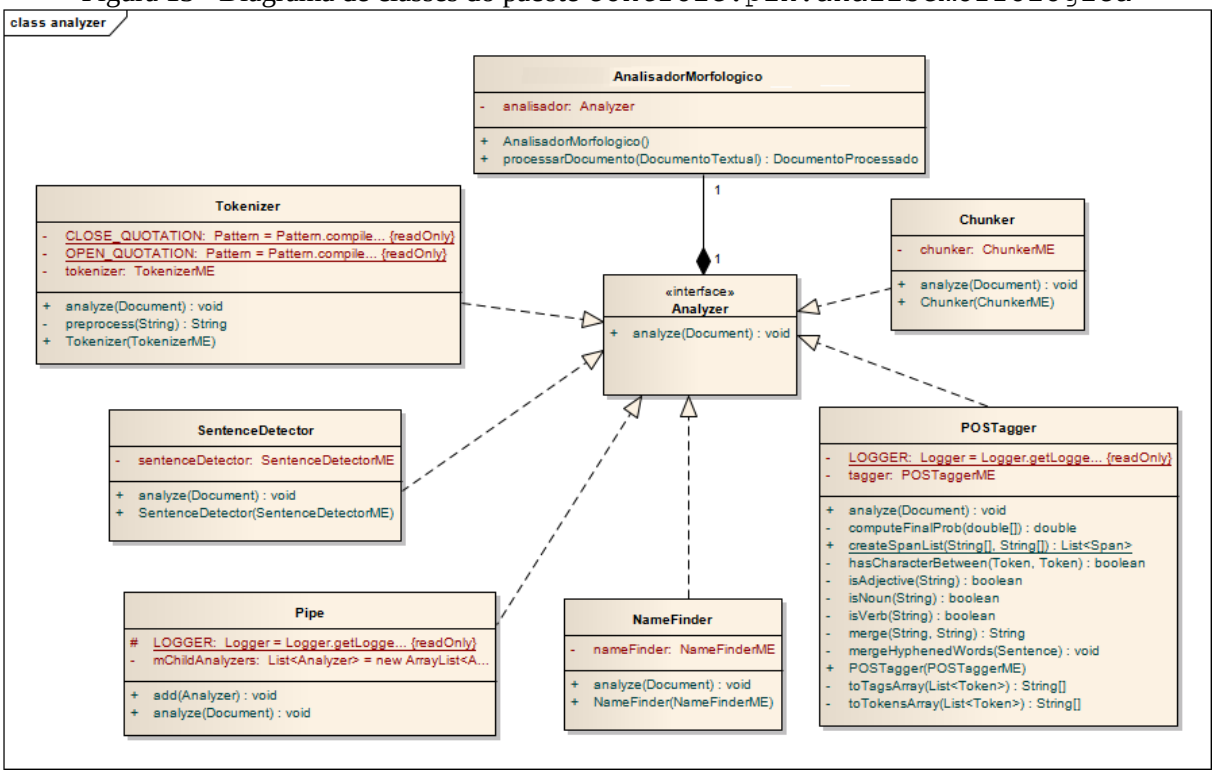
Figura 12 - Diagrama de classes do pacote `controle.pln.lematizacao`



O diagrama de classes do pacote `controle.pln.analisemorfologica` é apresentado na Figura 13. Ele é composto da classe `AnalisadorMorfologico`. Essa classe possui um atributo chamado `analisador` pertencente à classe `Analyser`. Esta é responsável por realizar a análise morfológica do texto apresentado à ela e faz parte do Cogroo (USP, 2011).

O processo de análise morfológica é iniciado após a chamada do método `processarDocumento` da classe `AnalisadorMorfologico`.

Figura 13 - Diagrama de classes do pacote `controle.pln.analisemorfologica`



Na Figura 14 é descrito o diagrama de classes do pacote `controle.mineracaotexto`. Ele foi modelado utilizando o padrão de projeto *Facade*. De acordo com Pfleeger (2004, p. 120), esse padrão objetiva prover uma interface única de acesso a várias funcionalidades de uma API ou subsistemas. A classe `MineradorTexto` é a interface única de acesso a todas as funcionalidades da ferramenta. Ela integra as chamadas de métodos das classes `Coletor`, `PreProcessador`, `Indexador` e `Minerador`.

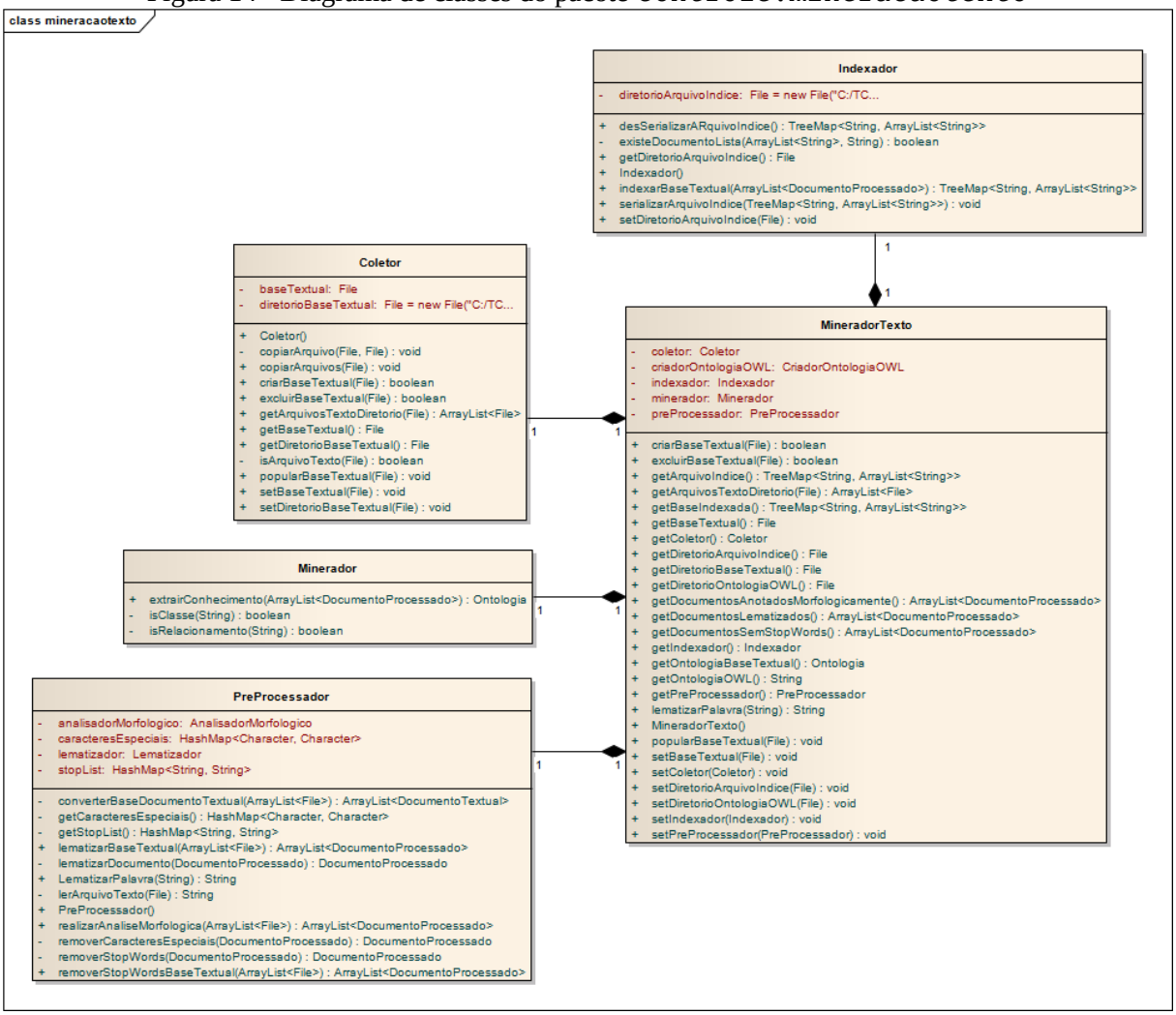
A classe `Coletor` implementa as funcionalidades da etapa de coleta do processo de MT. Quando se navega por um diretório buscando por documentos para compor a base textual de trabalho atual, é necessário filtrar os arquivos encontrados. Isso foi implementado através do método `isArquivoTexto`, onde é analisada a extensão do arquivo. Os arquivos que forem da extensão `txt` serão copiados para a base textual de trabalho atual através do método `copiarArquivos`.

Após a cópia, entra em execução a etapa de pré-processamento, modelada através da classe `PreProcessador`. Ela implementa sucessivas operações sobre os documentos textuais, resultando como saída desse processo, um objeto da classe `DocumentoProcessado` com seu conteúdo no formato ideal para as etapas de indexação e mineração.

A classe `Indexador` prove as funcionalidades do processo de indexação da base textual de trabalho atual. Seu método principal é o `indexarBaseTextual`, que recebe como entrada uma lista de arquivos pré-processados e gera como saída um arquivo de índice invertido.

Por fim, tem-se a etapa de mineração do conhecimento presente na base textual atual de trabalho. Essa função é desempenhada através da classe `Minerador` e do método `extrairConhecimento`, que se utiliza de análise morfológica e reconhecimento de padrões para descobrir o conhecimento.

Figura 14 - Diagrama de classes do pacote `controle.mineracaotexto`

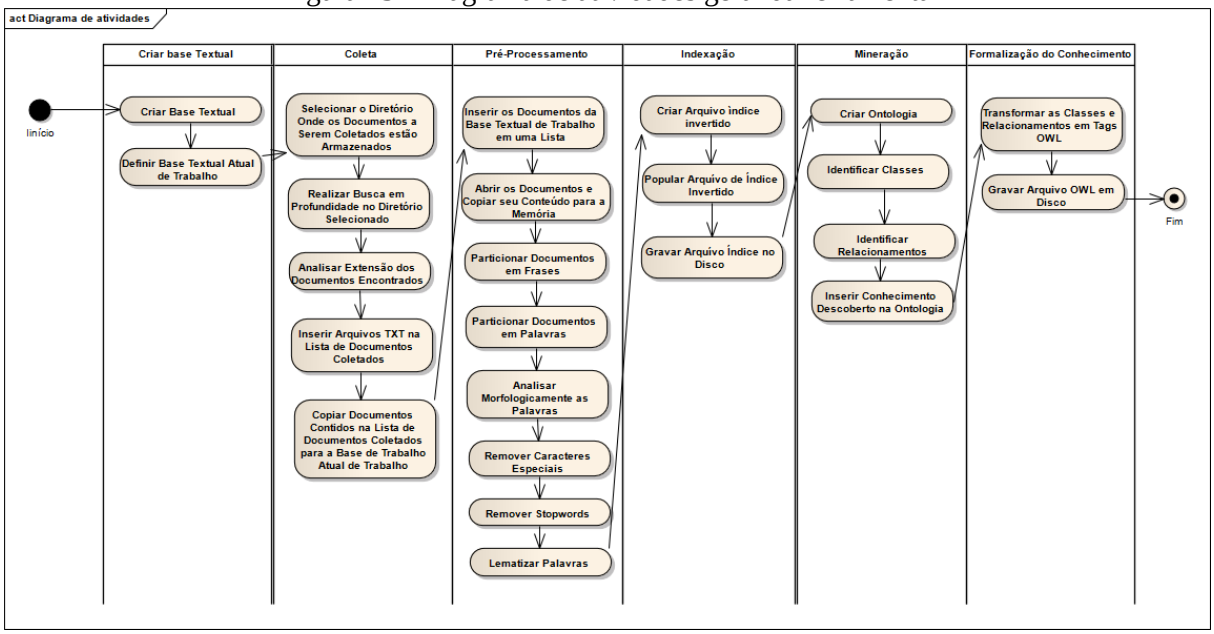


3.2.4 Diagrama de atividades

Conforme descrito por Pfleeger (2004, p. 70), o objetivo do diagrama de atividades é mostrar o fluxo de execução de um processo. Cada etapa desse processo é chamada de atividade. A Figura 15 ilustra o diagrama de atividades geral da ferramenta, ou seja, todas as atividades necessárias para a descoberta e formalização do conhecimento. As atividades são

divididas em raias, com o objetivo de demonstrar a que etapa do processo elas estão subordinadas.

Figura 15 - Diagrama de atividades geral da ferramenta



Todo o processo se inicia na raia `Criar base Textual`. Ela descreve a parte anterior ao processo de MT. Primeiramente é necessário `Criar Base Textual` para poder armazenar os documentos que farão parte da base textual de trabalho, servindo de entrada ao processo de descoberta e formalização do conhecimento. Uma vez criada, necessita-se `Definir Base Textual Atual de Trabalho`, sendo que todas as operações executadas pela ferramenta utilizarão a base que foi definida como base atual de trabalho.

Após isso, inicia-se o processo de MT com as atividades descritas na raia `Coleta`. Depois de `Selecionar o Diretório Onde os Documentos a Serem Coletados estão Armazenados` é necessário `Realizar Busca em Profundidade no Diretório Selecionado` com o intuito de procurar pelos documentos contidos no mesmo e em todos os seus diretórios filhos. Em seguida é necessário `Analisar Extensão dos Documentos Encontrados` e `Inserir Arquivos TXT na Lista de Documentos Coletados`. Com isso, é criada uma lista com todos os documentos com a extensão `txt` encontrados, sendo assim possível `Copiar Documentos Contidos na Lista de Documentos Coletados para a Base de Trabalho Atual de Trabalho`.

Com a coleta dos documentos concluída é realizado o pré-processamento dos mesmos, descrito pelas atividades da raia `Pré-Processamento`. Esse processo se inicia ao `Inserir os Documentos da Base Textual de Trabalho em uma Lista`. Com isso, é criada uma lista que armazena uma referência para os documentos da base atual de trabalho que estão

armazenados em disco. Na sequência, é preciso Abrir os Documentos e Copiar seu Conteúdo para a Memória para poder assim manipulá-los. Em seguida, são realizadas sucessivas operações sobre o conteúdo dos documentos visando aumentar a sua qualidade, sendo elas: Particionar Documentos em Frases, Particionar Documentos em Palavras, Analisar Morfologicamente as Palavras, Remover Caracteres Especiais, Remover Stopwords e Lematizar Palavras.

Com o pré-processamento concluído pode-se iniciar a etapa de indexação, descrita pela raia Indexação. Primeiramente é necessário Criar Arquivo índice Invertido e depois Popular Arquivo de Índice Invertido com todas as palavras resultantes do processo de pré-processamento, juntamente com a lista dos documentos em que elas estão contidas. Por fim, basta Gravar Arquívo Índice no Disco para a conclusão dessa etapa do processo de MT.

Em seguida, inicia-se a etapa de mineração, descrita pela raia Mineração. A primeira atividade desse processo é Criar Ontologia para armazenar o conhecimento que será descoberto. Para descobrir o conhecimento é preciso Identificar Classes e Identificar Relacionamentos. Ao Inserir Conhecimento Descoberto na Ontologia é caracterizado o fim do processo de MT.

Após a etapa de MT, o conhecimento foi descoberto e estruturado na forma de uma ontologia, mas ainda é necessário formalizá-lo através de uma ontologia OWL. Esse processo é descrito através da raia Formalização do Conhecimento, composta de duas etapas. Na primeira etapa é necessário Transformar as Classes e Relacionamentos em Tags OWL e na segunda Gravar Arquivo OWL em Disco.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas, como é realizada a coleta, o pré-processamento, a criação do arquivo de índice invertido, a descoberta e formalização do conhecimento, além da operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

As técnicas e ferramentas utilizadas para o desenvolvimento da ferramenta proposta foram as seguintes:

a) a linguagem de programação Java, na versão 7.0, para codificar a ferramenta;
b) a IDE Eclipse, na versão Juno, como ambiente de desenvolvimento;

c) a biblioteca PTStemmer (OLIVEIRA, 2010), na versão 2.0, para lematizar as palavras contidas na base textual de trabalho atual;

d) a biblioteca Cogroo (USP, 2011), na versão 4.0, para analisar morfologicamente as palavras contidas na base textual de trabalho atual.

A biblioteca PTStemmer implementa três algoritmos de lematização de palavras da língua portuguesa, sendo eles: Orengo, Porter e Savoy. É uma biblioteca de código livre tendo implementações em Java, Python e C#. O Quadro 16 mostra como utilizar os algoritmos Orengo, Porter e Savoy na linguagem de programação Java. Nas linhas 1, 4 e 7 são criados objetos do tipo `Stemmer` das classes `OrengoStemmer`, `PorterStemmer` e `SavoyStemmer`, cada uma representando o respectivo algoritmo de lematização. Por fim, nas linhas 2, 5 e 8 o texto do qual se deseja lematizar é passado como argumento para o método `getWordStem`, sendo sua saída exibida no console.

Quadro 16 - Exemplo de utilização dos algoritmos Orengo, Porter e Savoy em Java

```
1  Stemmer lematizador = new OrengoStemmer();
2  System.out.println(lematizador.getWordStem("Utilizando o algoritmo Orengo"));
3
4  Stemmer lematizador = new PorterStemmer();
5  System.out.println(lematizador.getWordStem("Utilizando o algoritmo Porter"));
6
7  Stemmer lematizador = new SavoyStemmer();
8  System.out.println(lematizador.getWordStem("Utilizando o algoritmo Savoy"));
```

Cogroo é uma biblioteca de código livre escrita em Java, sob licensa GNU *Lesser General Public License* (LGPL), sendo seus mantenedores oficiais o Centro de Competência em Software Livre (CCSL) da Universidade de São Paulo (USP). No Quadro 17 é exposta a estrutura principal para o uso da biblioteca. Na linha 1 é definido o idioma dos documentos a serem analisados, no caso o português do Brasil (`pt-BR`). Na linha 4 é criado um objeto da classe `Document` que representa o documento a ser analisado e na linha 5, através do método `setText`, é definido o texto a ser analisado. A análise ocorre através do método `cogroo.analyse` (linha 6), que recebe como argumento o objeto da classe `Document` criado anteriormente. Na linha 8 é descrito que para se acessar o conteúdo do objeto `Document` que foi analisado é utilizado o seu método `getSentences`, que retorna uma lista com todas as sentenças analisadas. A partir de cada sentença é possível acessar os seus *tokens* com o método `getTokens` (linha 9). Por fim, é possível imprimir as informações do *token*, sendo elas: nome (linha 11), lista com possíveis radicais (linha 12), *tag* denotando sua classificação morfológica (linha 13) e informações variadas, como gênero e tempo verbal (linha 14).

Quadro 17 - Exemplo da utilização da biblioteca Cogroo

```
1  ComponentFactory factory = ComponentFactory.create(new Locale("pt", "BR"));
2  Analyser  cogroo = factory.createPipe();
3
4  Document document = new DocumentImpl();
5  document.setText(documentText);
6  cogroo.analyze(document);
7
8  for (Sentence sentence : document.getSentences()) {
9    for (Token token : sentence.getTokens()) {
10
11       System.out.println(token.getLexeme());
12       System.out.println(Arrays.toString(token.getLemmas()));
13       System.out.println(token.getPOSTag());
14       System.out.println(token.getFeatures());
15    }
16  }
```

3.3.2 Coleta dos documentos

Para que seja possível realizar a descoberta do conhecimento a partir dos documentos contidos na base de trabalho atual, primeiro é necessário coletar os referidos documentos. Essa funcionalidade é implementada pelo método `copiarArquivos` da classe `Coletor`, apresentado no Quadro 18. Ao chamar o método `copiarArquivos` é passado como argumento o diretório em que será realizada a busca em profundidade pelos documentos `txt`.

Em seguida é verificado se o arquivo passado como argumento é um diretório (linha 4). Se essa condição for verdadeira, todos os seus arquivos são armazenados em um *array* (linha 6). Para cada arquivo, é recursivamente chamado o método `copiarArquivos` (linha 8).

Caso o resultado do teste da linha 4 seja falso, significa que está sendo manipulado um arquivo. O primeiro passo a se tomar é verificar a sua extensão, sendo para isso utilizado o método `isArquivoTexto` (linha 11) da classe `Coletor`. A partir do momento que o arquivo testado possuir a extensão `txt`, ele será copiado para a base textual de trabalho atual, através do método `copiarArquivo` (linha 12) passando como argumentos o arquivo a ser copiado e o diretório de destino.

Quadro 18 - Código fonte do método `copiarArquivos`

```
1  public void copiarArquivos(File diretorioOrigem) throws IOException {
2
3    File destino = getBaseTextual();
4    if (diretorioOrigem.isDirectory()) {
5
6      File[] arquivosDiretorioOrigem = diretorioOrigem.listFiles();
7      for (File arquivo : arquivosDiretorioOrigem)
8        copiarArquivos(arquivo);
9
10   } else {
11     if (isArquivoTexto(diretorioOrigem))
12       copiarArquivo(diretorioOrigem,new File(destino.getAbsolutePath() + "/"+
13       diretorioOrigem.getName()));
14   }
15 }
```

O Quadro 19 descreve a implementação do método `copiarArquivo`. Primeiramente deve-se abrir o arquivo de origem (linha 3) e o arquivo de destino (linha 4). Com eles abertos, é iniciada uma iteração onde, enquanto existir *bytes* a serem lidos no arquivo de origem, são lidos 1024 *bytes* do arquivo de origem (através do método `read`, presente na linha 9) e gravados no arquivo de destino (linha 10). Por fim, o arquivo de leitura e o arquivo de destino são fechados (linhas 12 e 13 respectivamente).

Quadro 19- Código fonte do método `copiarArquivo`

```
1  private void copiarArquivo(File origem, File destino) throws IOException {
2
3    FileInputStream leitor = new FileInputStream(origem);
4    FileOutputStream gravador = new FileOutputStream(destino);
5
6    int quantidadeBytesLida = 0;
7    byte conteudoLido[] = new byte[1024];
8
9    while ((quantidadeBytesLida = leitor.read(conteudoLido)) > 0)
10     gravador.write(conteudoLido, 0, quantidadeBytesLida);
11
12  leitor.close();
13  gravador.close();
14 }
```

3.3.3 Pré-processamento dos documentos

Com a base populada pode-se iniciar o pré-processamento dos documentos. Esse processo inicia com o método `converterBaseDocumentoTextual`, sendo descrito no Quadro 20. É passado como argumento uma lista contendo a referência para os arquivos contidos na base textual atual de trabalho. Após isso, é realizada uma iteração sobre essa lista (linha 10) e para cada arquivo contido nela é criado um objeto da classe `DocumentoTextual` (linha 15). Para ler o conteúdo do arquivo armazenado no disco é utilizado o método `lerArquivoTexto` (linha 12), da classe `PreProcessador`.

Quadro 20 - Código fonte do método `converterBaseDocumentoTextual`

```
1  private ArrayList<DocumentoTextual> converterBaseDocumentoTextual
2  (ArrayList<File> arquivosBaseTextual) throws IOException {
3
4    ArrayList<DocumentoTextual> documentos = new ArrayList<DocumentoTextual>();
5    DocumentoTextual documentoTextual;
6
7    String conteudoDocumento = null;
8    String nomeDocumento = null;
9
10   for (File documentoBaseTextual : arquivosBaseTextual) {
11
12     conteudoDocumento = lerArquivoTexto(documentoBaseTextual);
13     nomeDocumento = documentoBaseTextual.getName();
14
15     documentoTextual = new DocumentoTextual(conteudoDocumento, nomeDocumento);
16     documentosTextuais.add(documentoTextual);
17   }
18
19  return documentosTextuais;
20 }
```

No Quadro 21 é apresentada a implementação do método `lerArquivoTexto`. Inicialmente o arquivo é aberto para a leitura (linha 3), sendo criado um objeto da classe `BufferedReader`, chamado `leitor`, para ler o conteúdo do arquivo. Em seguida, enquanto a leitura não atingir o final do arquivo, uma linha é lida (linha 9) e concatenada com o conteúdo lido previamente (linha 11). Também é concatenada uma quebra de linha para manter a estrutura original do arquivo. Com a conclusão da leitura, é fechado o fluxo de leitura (linha 15) e o arquivo (linha 16). O método finaliza com o retorno do conteúdo lido (linha 18).

Quadro 21 - Código fonte do método `lerArquivoTexto`

```
1  private String lerArquivoTexto(File arquivoTexto) throws IOException {
2
3    FileReader arquivoAberto = new FileReader(arquivoTexto);
4    BufferedReader leitor = new BufferedReader(arquivoAberto);
5
6    String linhaLida;
7    StringBuilder conteudoArquivo = new StringBuilder();
8
9    while ((linhaLida = leitor.readLine()) != null) {
10
11     conteudoArquivo.append(linhaLida);
12     conteudoArquivo.append("\n");
13   }
14
15   leitor.close();
16   arquivoAberto.close();
17
18   return conteudoArquivo.toString();
19 }
```

Após transformar os arquivos da base textual atual de trabalho em objetos presentes na memória, são realizados sucessivos processamentos. O primeiro deles é a realização das análises sintática e morfológica. Isso dar-se-á através do método `processarDocumento` da classe `AnalisadorMorfologico`. O método `processarDocumento` utiliza a biblioteca Cogroo (descrita na seção 3.3.1) para realizar as análises, sendo descrito no Quadro 22. Na linha 4 é definido que o texto a ser analisado é proveniente do conteúdo do objeto `DocumentoTextual`, passado como argumento. Após a análise, é criado um objeto da classe `DocumentoProcessado`, que é populado com as sentenças analisadas (linha 9) e posteriormente retornado (linha 11) a quem chamou o método.

Quadro 22 - Código fonte do método `processarDocumento`

```
1 public DocumentoProcessado processarDocumento(DocumentoTextual documentoTextual){
2
3  Document documento = new DocumentImpl();
4  documento.setText(documentoTextual.getConteudo());
5  analisador.analyze(documento);
6
7  String nome = documentoTextual.getNome();
8  DocumentoProcessado documentoProcessado = new DocumentoProcessado(nome);
9  documentoProcessado.setSentencas((ArrayList<Sentence>)documento.getSentences());
10
11  return documentoProcessado;
12 }
```

Em seguida, o objeto da classe `DocumentoProcessado` é submetido ao método `removerCaracteresEspeciais`, descrito no Quadro 23. Este, como início de seu processamento, realiza uma iteração sobre todas as sentenças do documento (linha 5). Para cada sentença, é iterado sobre todas as suas palavras (linha 6), verificando a existência das mesmas em uma lista que contêm todos os caracteres especiais utilizados pela ferramenta (linha 8). Caso a palavra não esteja contida na lista, significa que ela não é um caractere especial, sendo assim, inserida em uma lista de palavras válidas (linha 9). Após o término da iteração sobre as palavras da frase (linha 10), a lista de palavras da frase é substituída pela lista sem os caracteres especiais (linha 12). Após a iteração sobre a última frase (linha 14) o documento processado é retornado (linha 16).

Quadro 23 - Código fonte do método `removerCaracteresEspeciais`

```
1  private DocumentoProcessado removerCaracteresEspeciais
2  (DocumentoProcessado documentoProcessado) {
3
4    ArrayList<Token> tokensSentenca = new ArrayList<Token>();
5    for (Sentence sentenca : documentoProcessado.getSentencas()) {
6      for (Token token : sentenca.getTokens()) {
7
8        if (!caracteresEspeciais.containsKey(token.getLexeme().charAt(0)))
9          tokensSentenca.add(token);
10      }
11
12     sentenca.setTokens(tokensSentenca);
13     tokensSentenca = new ArrayList<Token>();
14   }
15
16   return documentoProcessado;
17 }
```

Após a remoção dos caracteres especiais, o objeto da classe `DocumentoProcessado` é submetido ao método `removerStopWords`, descrito pelo Quadro 24.

Quadro 24 - Código fonte do método `removerStopWords`

```
1  private DocumentoProcessado removerStopWords
2  (DocumentoProcessado documentoProcessado) {
3
4    ArrayList<Token> tokensSentenca = new ArrayList<Token>();
5    for (Sentence sentenca : documentoProcessado.getSentencas()) {
6      for (Token token : sentenca.getTokens()) {
7
8        if (!stopList.containsKey(token.getLexeme().toLowerCase()))
9          tokensSentenca.add(token);
10     }
11     sentenca.setTokens(tokensSentenca);
12     tokensSentenca = new ArrayList<Token>();
13   }
14
15   return documentoProcessado;
16 }
```

Como início de seu processamento, realiza uma iteração sobre todas as sentenças do documento (linha 5). Para cada sentença, é iterado sobre todas as suas palavras (linha 6), verificando a existência das mesmas em uma lista que contêm todas as *stopwords* utilizadas

pela ferramenta (linha 8). Caso a palavra não esteja contida na lista, significa que ela não é uma *stopword*, sendo assim, inserida em uma lista de palavras válidas (linha 9). Após o término da iteração sobre as palavras da frase (linha 10), a lista de palavras da frase é substituída pela lista sem as *stopwords* (linha 12). Após a iteração sobre a última frase (linha 13) o documento processado é retornado (linha 15). A lista de *stopwords* utilizada foi disponibilizada no Anexo A.

A última etapa que o objeto da classe `DocumentoProcessado` é submetido é a lematização. Esta foi implementada através do método `lematizarDocumento` descrito no Quadro 25. O método `lematizarDocumento` tem como início de seu processamento iterar sobre todas as sentenças do documento (linha 3). Para cada sentença, é iterado sobre todas as suas palavras (linha 4), realizando a lematização das mesmas (linha 5). Para isso, é utilizado o método `lematizar` da classe `Lematizador`, que faz uso da biblioteca PTStemmer (descrita na seção 3.3.1).

Quadro 25 - Código fonte do método `lematizarDocumento`

```
1  private DocumentoProcessado lematizarDocumento(DocumentoProcessado documento) {
2
3    for (Sentence sentenca : documento.getSentencas()) {
4      for (Token token : sentenca.getTokens())
5        token.setLexeme(lematizador.lematizar(token.getLexeme()));
6    }
7
8    return documento;
9  }
```

3.3.4 Criação do arquivo de índice invertido

Com a etapa de pré-processamento concluída, a base textual teve sua dimensionalidade reduzida consideravelmente, fato crucial para se criar um arquivo de índice otimizado. O arquivo de índice invertido é criado através do método `indexarBaseTextual`, sendo o mesmo descrito pelo Quadro 26. O método `indexarBaseTextual` inicia seu processamento criando o objeto que representa o arquivo de índice em memória (linha 6). Em seguida, é iterado sobre a lista de `DocumentoProcessado` passada como argumento (linha 8). Para cada objeto contido na lista, é realizada uma iteração sobre todas as suas sentenças (linha 9). Em cada sentença é iterado sobre todas as suas palavras (linha 10).

Em seguida, é testado se a palavra existe no arquivo de índice (linha 12). Caso ela exista, é recuperada a lista de documentos que a contém (linha 14). Como uma palavra pode aparecer várias vezes em um mesmo documento, é necessário testar se o nome do documento da iteração atual existe na lista de documentos que contém a palavra da iteração atual (linha

17), isto dar-se-á através do método `existeDocumentoLista` (linha 17). Se o nome do arquivo não constar na lista, ele será inserido (linha 18), caso contrário será desconsiderado.

Em contrapartida, se a palavra não constar no arquivo de índice (linha 20), o documento da iteração atual será inserido na lista de documentos em que ela está contida (linha 23) e a palavra é inserida no arquivo de índice invertido (linha 24). Por fim, o arquivo de índice construído é retornado (linha 30), para que seja gravado em disco.

Quadro 26 - Código fonte do método `indexarBaseTextual`

```
1  public TreeMap<String, ArrayList<String>> indexarBaseTextual
2  (ArrayList<DocumentoProcessado> documentosLematizados){
3
4    ArrayList<String> documentosContemPalavra = new ArrayList<String>();
5    TreeMap<String, ArrayList<String>> arquivoIndice;
6    arquivoIndice = new TreeMap<String,ArrayList<String>>();
7
8    for(DocumentoProcessado documentoProcessado : documentosLematizados){
9      for(Sentence sentenca : documentoProcessado.getSentencas()){
10       for(Token token : sentenca.getTokens()){
11
12         if(arquivoIndice.containsKey(token.getLexeme())){
13
14           documentosContemPalavra = arquivoIndice.get(token.getLexeme());
15           String nomeDocumento = documentoProcessado.getNome();
16
17           if(!existeDocumentoLista(documentosContemPalavra,nomeDocumento))
18             documentosContemPalavra.add(documentoProcessado.getNome());
19
20         }else{
21
22           documentosContemPalavra = new ArrayList<String>();
23           documentosContemPalavra.add(documentoProcessado.getNome());
24           arquivoIndice.put(token.getLexeme(), documentosContemPalavra);
25         }
26       }
27     }
28   }
29
30   return arquivoIndice;
31 }
```

3.3.5 Descoberta do conhecimento

Com a conclusão da indexação inicia-se a etapa de descoberta de conhecimento, implementada através do método `extrairConhecimento`, descrito no Quadro 27 - Código fonte do método `extrairConhecimento`. O primeiro passo é criar a ontologia para armazenar o conhecimento descoberto (linha 4). Em seguida, é iterado sobre a lista de `DocumentoProcessado` passada como argumento (linha 10). Para cada objeto contido na lista, é realizada uma iteração sobre todas as suas sentenças (linha 11). Em cada sentença é iterado sobre todas as suas palavras (linha 16).

É testado se a palavra é uma classe (linha 18), através do método `isClasse`, analisando a sua etiqueta morfológica. Caso o resultado do teste for verdadeiro, é verificado

se foi achado algum relacionamento na frase (linha 19). O fato do resultado do teste anterior ser falso, significa que foi identificada a primeira classe do relacionamento. Antes da classe ser inserida na ontologia, é verificado se ela existe na mesma (linhas 21 e 22). Caso a mesma não contenha a classe identificada, um novo objeto de `Classe` é criado (linha 24) e inserido na ontologia (linha 25).

Quadro 27 - Código fonte do método `extrairConhecimento`

```
1  public Ontologia extrairConhecimento
2  (ArrayList<DocumentoProcessado> documentosProcessados) {
3
4    Ontologia ontologia = new Ontologia();
5    Classe classe1 = null;
6    Classe classe2 = null;
7    Relacionamento relacionamento = null;
8    boolean achouRelacionamento, achouClasse;
9
10   for (DocumentoProcessado documento : documentosProcessados) {
11     for (Sentence sentenca : documento.getSentencas()) {
12
13       achouRelacionamento = false;
14       achouClasse = false;
15
16       for (Token token : sentenca.getTokens()) {
17
18         if (isClasse(token.getPOSTag())) {
19           if (!achouRelacionamento) {
20
21             classe1 = ontologia.getClasse(token.getLexeme());
22             if (classe1 == null){
23
24               classe1 = new Classe(token.getLexeme());
25               ontologia.adicionarClasse(classe1);
26              }
27           achouClasse = true;
28
29         } else {
30
31              classe1.adicionarRelacionamento(relacionamento);
32              classe2 = new Classe(token.getLexeme());
33              relacionamento.setClasse(classe2);
34              ontologia.adicionarClasse(classe2);
35              achouRelacionamento = false;
36           }
37        }
38
39         if (isRelacionamento(token.getPOSTag())) {
40           if (achouClasse) {
41
42             relacionamento = new Relacionamento(token.getLexeme());
43             achouRelacionamento = true;
44             achouClasse = false;
45             }
46           }
47        }
48      }
49   }
50
51  return ontologia;
52 }
```

Se por ventura o resultado do teste da linha 19 for verdadeiro (linha 29), significa que anteriormente foi encontrado um relacionamento e a classe identificada foi a segunda do

mesmo. Então, é inserido o relacionamento na primeira classe da relação (linha 31), criado um objeto de `Classe` que representa a classe identificada (linha 32), inserindo-o como a segunda classe do relacionamento (linha 33) e em seguida na ontologia (linha 34).

Na linha 39 é verificado se a palavra da iteração atual é um relacionamento, através do método `isRelacionamento`. Sendo classificado como tal, é verificado se na sentença já foi encontrada alguma classe (linha 40). Se esse fato se confirmar, é então criado um objeto de `Relacionamento` (linha 42), caso contrário ele é descartado. Por fim, após o processamento do último documento, a ontologia contém todo o conhecimento descoberto da base textual, sendo assim retornada para que possa ser criado o arquivo OWL.

3.3.6 Construção da base de conhecimento

A ontologia criada no processo de descoberta de conhecimento é transformada em um arquivo OWL, gravado em disco, caracterizando assim a construção da base de conhecimento. Esse processo consiste em transformar os componentes que modelam o conhecimento (classes e relacionamentos), contidos na ontologia, em *tags* OWL (seguindo o padrão de *tags* utilizado pelo Protégé e descrito na seção 2.2). Para tal, é utilizado o método `criarOntologiaOWL` descrito no Quadro 28. É realizada uma iteração sobre as classes da ontologia (linha 8), criando as *tags* que as representam (linhas 10 a 14). Para cada classe, é iterado sobre seus relacionamentos (linha 18), criando assim as *tags* para representar: o mesmo (linhas 20 a 24), seu domínio (linhas 26 a 33) e seu escopo (linhas 35 a 43). Por fim, o arquivo com o conteúdo da ontologia OWL é retornado, para que possa ser gravado em disco (linha 48).

Quadro 28 - Código fonte do método `criarOntologiaOWL`

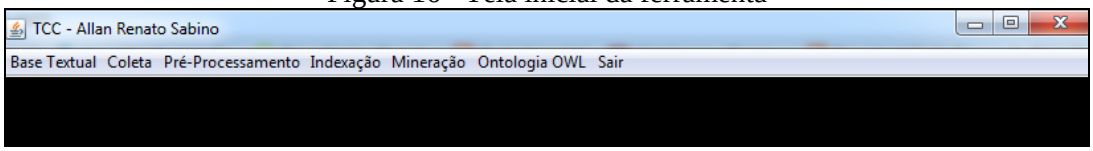
```
1  public String criarOntologiaOWL(Ontologia ontologia) {
2
3    StringBuilder ontologiaOWL = new StringBuilder();
4    Classe classeOntologia1,classeOntologia2;
5    TreeMap<String, Relacionamento> relacionamentosClasse;
6
7    ontologiaOWL.append(ConstantesOntologiaOWL.ABRIR_ONTOLOGIA);
8    for (String classe : ontologia.getClasses().keySet()) {
9
10    ontologiaOWL.append(ConstantesOntologiaOWL.ABRIR_DECLARACAO);
11    ontologiaOWL.append(ConstantesOntologiaOWL.ABRIR_CLASSE);
12    ontologiaOWL.append(classe);
13    ontologiaOWL.append(ConstantesOntologiaOWL.FECHAR_CLASSE);
14    ontologiaOWL.append(ConstantesOntologiaOWL.FECHAR_DECLARACAO);
15
16    classeOntologia1 = ontologia.getClasse(classe);
17    relacionamentosClasse = classeOntologia1.getRelacionamentos();
18    for(String relacionamento : relacionamentosClasse.keySet()){
19
20      ontologiaOWL.append(ConstantesOntologiaOWL.ABRIR_DECLARACAO);
21      ontologiaOWL.append(ConstantesOntologiaOWL.ABRIR_OBJECT_PROPRIETY);
22      ontologiaOWL.append(relacionamento);
23      ontologiaOWL.append(ConstantesOntologiaOWL.FECHAR_OBJECT_PROPRIETY);
24      ontologiaOWL.append(ConstantesOntologiaOWL.FECHAR_DECLARACAO);
```

```
25
26      ontologiaOWL.append(ConstantesOntologiaOWL.ABRIR_OBJECT_PROPRIETY_DOMAIN);
27      ontologiaOWL.append("\t"+ConstantesOntologiaOWL.ABRIR_OBJECT_PROPRIETY);
28      ontologiaOWL.append(relacionamento);
29      ontologiaOWL.append(ConstantesOntologiaOWL.FECHAR_OBJECT_PROPRIETY);
30      ontologiaOWL.append("\t"+ConstantesOntologiaOWL.ABRIR_CLASSE);
31      ontologiaOWL.append(classe);
32      ontologiaOWL.append(ConstantesOntologiaOWL.FECHAR_CLASSE);
33      ontologiaOWL.append(ConstantesOntologiaOWL.FECHAR_OBJECT_PROPRIETY_DOMAIN);
34
35      classeOntologia2 = relacionamentosClasse.get(relacionamento).getClasse();
36      ontologiaOWL.append(ConstantesOntologiaOWL.ABRIR_OBJECT_PROPRIETY_RANGE);
37      ontologiaOWL.append("\t"+ConstantesOntologiaOWL.ABRIR_OBJECT_PROPRIETY);
38      ontologiaOWL.append(relacionamento);
49      ontologiaOWL.append(ConstantesOntologiaOWL.FECHAR_OBJECT_PROPRIETY);
40      ontologiaOWL.append("\t"+ConstantesOntologiaOWL.ABRIR_CLASSE);
41      ontologiaOWL.append(classeOntologia2.getNome());
42      ontologiaOWL.append(ConstantesOntologiaOWL.FECHAR_CLASSE);
43      ontologiaOWL.append(ConstantesOntologiaOWL.FECHAR_OBJECT_PROPRIETY_RANGE);
44    }
45  }
46
47  ontologiaOWL.append(ConstantesOntologiaOWL.FECHAR_ONTOLOGIA);
48  return ontologiaOWL.toString();
49 }
```

3.3.7 Operacionalidade da implementação

Ao se executar a ferramenta, a tela descrita na Figura 16 é apresentada ao usuário. As funcionalidades providas estão distribuídas entres os seus menus, de acordo com a etapa do processo que a mesma representa.

Figura 16 - Tela inicial da ferramenta

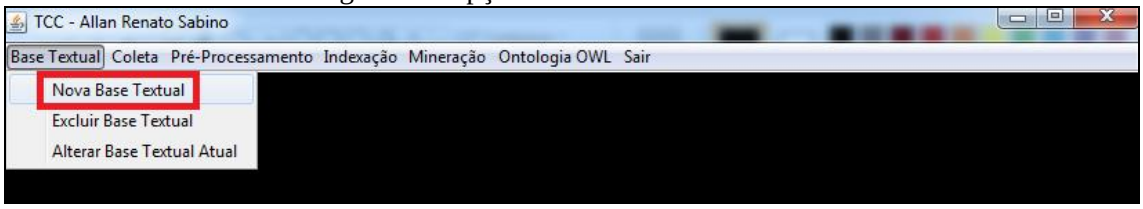


Logo quando entra em execução a ferramenta cria um diretório com o endereço `C:/TCC`. Dentro do mesmo são criados três diretórios: `Arquivos de Índice`, `Bases Textuais` e `Ontologias OWL`. Esses diretórios são utilizados para armazenar os arquivos de índices gerados, as bases textuais e as ontologias OWL, respectivamente.

3.3.7.1 Criar base textual

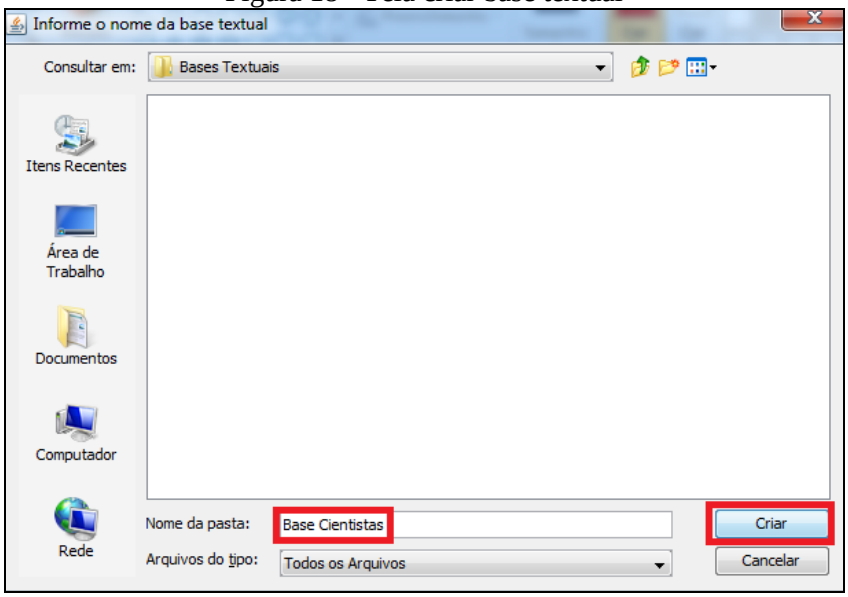
Para se criar uma base textual deve-se clicar no menu `Base Textual` e em seguida na opção `Nova Base Textual` (Figura 17).

Figura 17- Opção `Nova Base Textual`



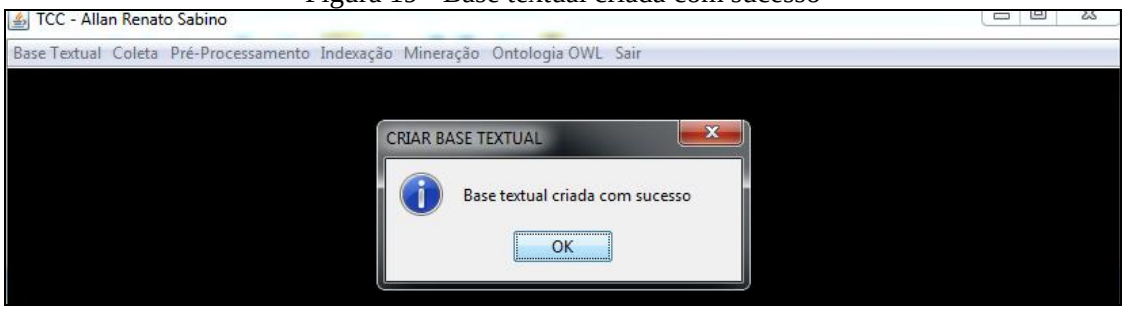
Em seguida, deve-se informar um nome para a base textual a ser criada e clicar no botão `Criar` (Figura 18).

Figura 18 - Tela criar base textual



É então criado um diretório com o nome escolhido dentro do diretório `C:/TCC/Bases Textuais` e uma mensagem informando o sucesso da criação da base textual é exibida (Figura 19).

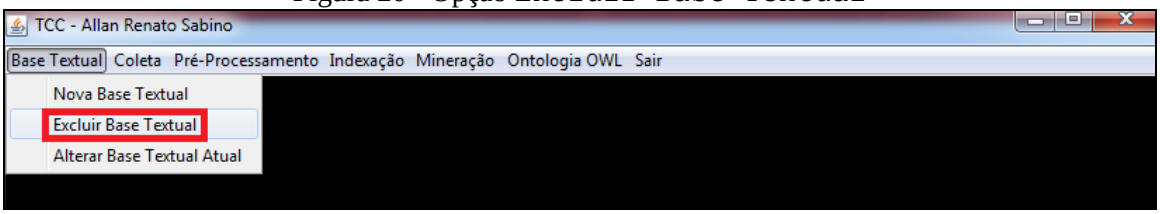
Figura 19 - Base textual criada com sucesso



3.3.7.2 Excluir Base Textual

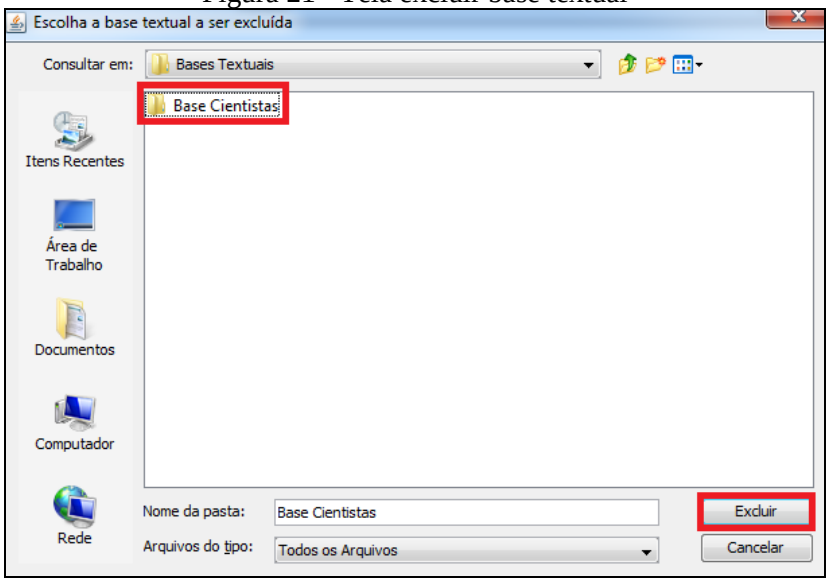
Para se criar uma base textual deve-se clicar no menu `Base Textual` e em seguida na opção `Excluir Base Textual` (Figura 20).

Figura 20 - Opção `Excluir Base Textual`



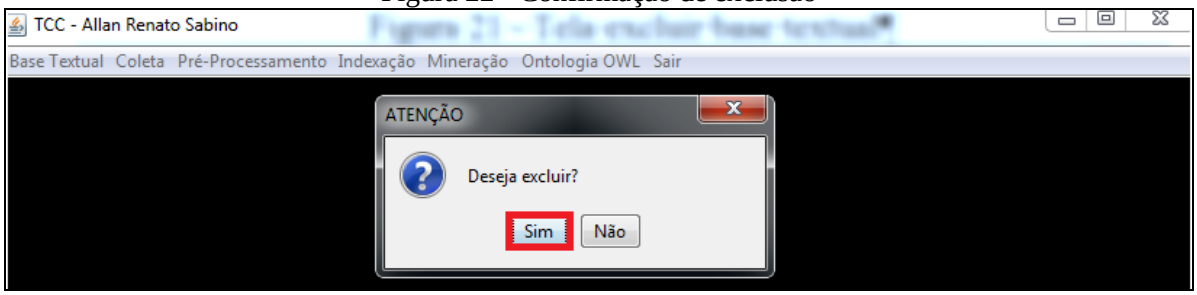
Em seguida, deve-se selecionar a base textual a ser excluída e clicar no botão `Excluir` (Figura 21).

Figura 21 - Tela excluir base textual



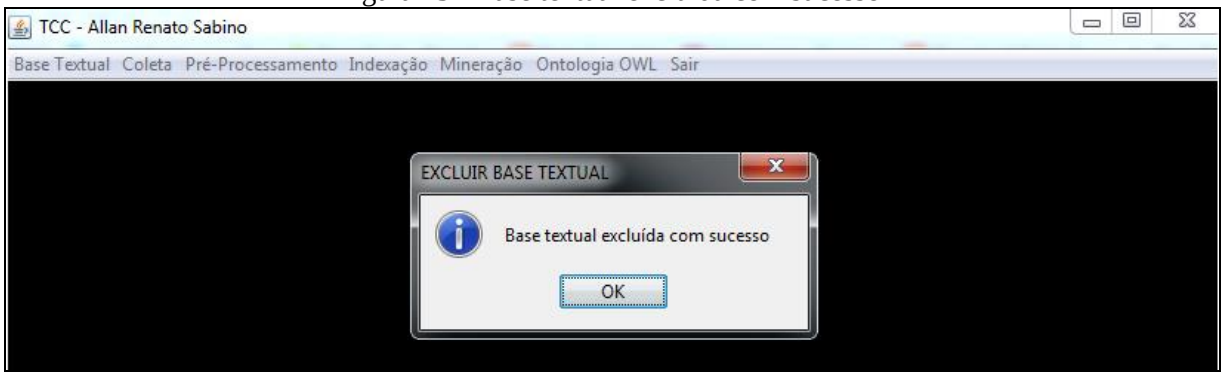
É exibida uma mensagem solicitando a confirmação da exclusão (Figura 22). Deve-se então clicar na opção `Sim`.

Figura 22 - Confirmação de exclusão



A base textual é deletada do disco, juntamente com todos os documentos textuais contidos por ela, sendo exibida uma mensagem afirmando o sucesso do processo (Figura 23).

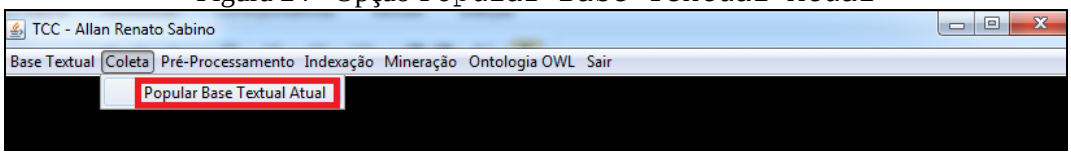
Figura 23 - Base textual excluída com sucesso



3.3.7.3 Coletar documentos

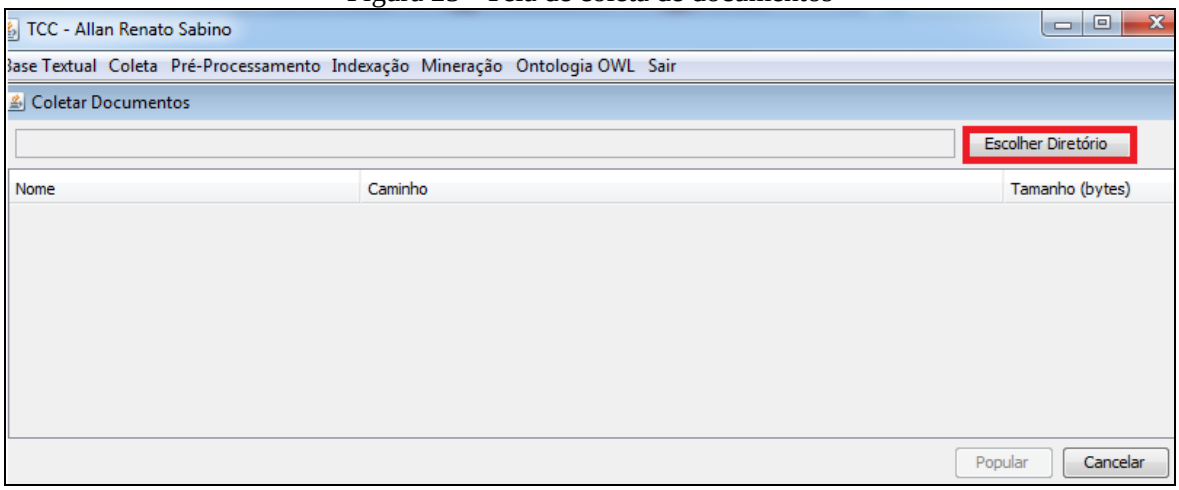
Para se coletar os documentos que compõem uma base textual deve-se clicar no menu `Coleta` e em seguida na opção `Popular Base Textual Atual` (Figura 24).

Figura 24 - Opção `Popular Base Textual Atual`



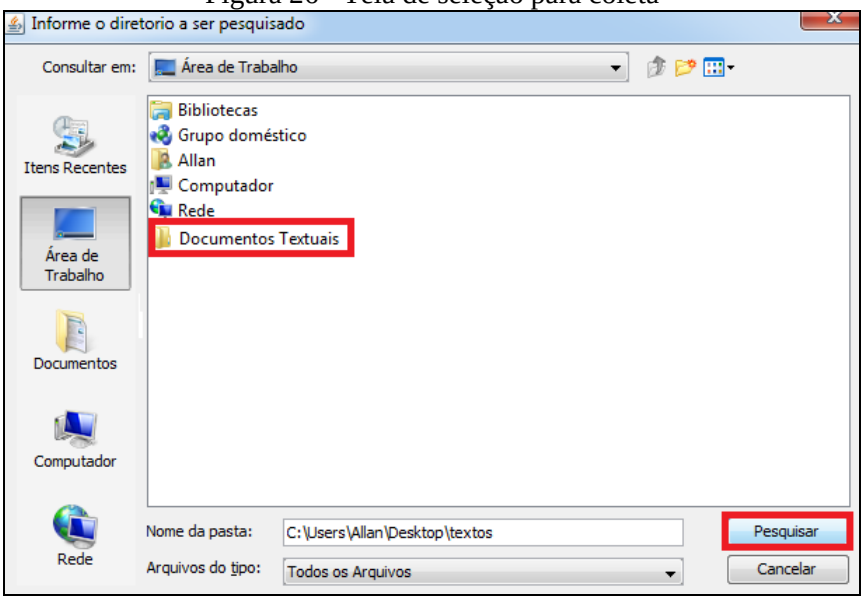
Após isso deve-se clicar no botão `Escolher Diretório` (Figura 25), sendo aberta uma tela (Figura 26) para escolher o local em que serão procurados os arquivos para compor a base textual.

Figura 25 - Tela de coleta de documentos



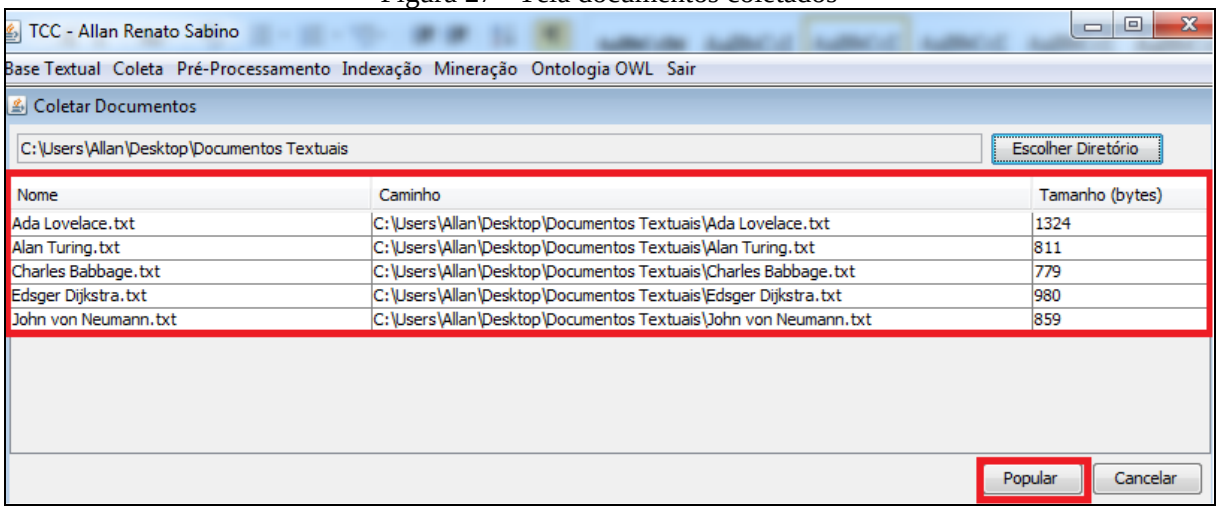
Deve-se selecionar um diretório e clicar no botão `Pesquisar` (Figura 26).

Figura 26 - Tela de seleção para coleta



A partir do diretório selecionado é realizada uma busca em profundidade coletando todos os documentos textuais e armazenando-os em uma tabela (Figura 27), exibindo três informações: nome, caminho e tamanho (em *bytes*). Por fim, clica-se no botão `Popular` (Figura 27) para copiar todos os documentos coletados para a base textual.

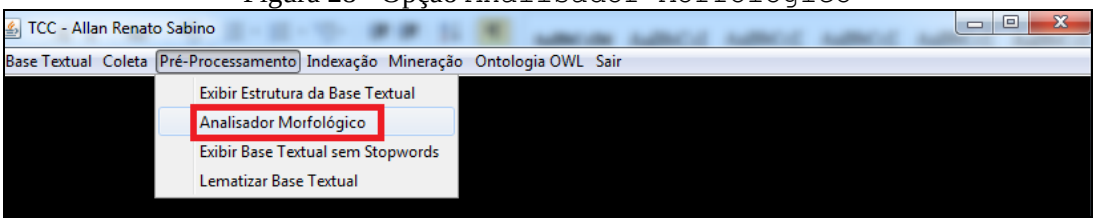
Figura 27 - Tela documentos coletados



3.3.7.4 Exibir estrutura morfológica da base textual

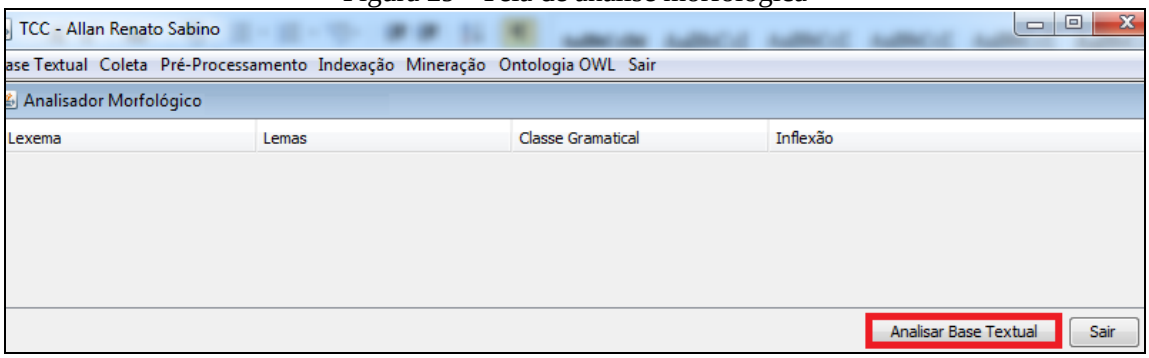
Para se exibir a estrutura morfológica das palavras que compõem a base textual deve-se clicar no menu `Pré-Processamento` e em seguida na opção `Analisador Morfológico` (Figura 28).

Figura 28 - Opção `Analisador Morfológico`



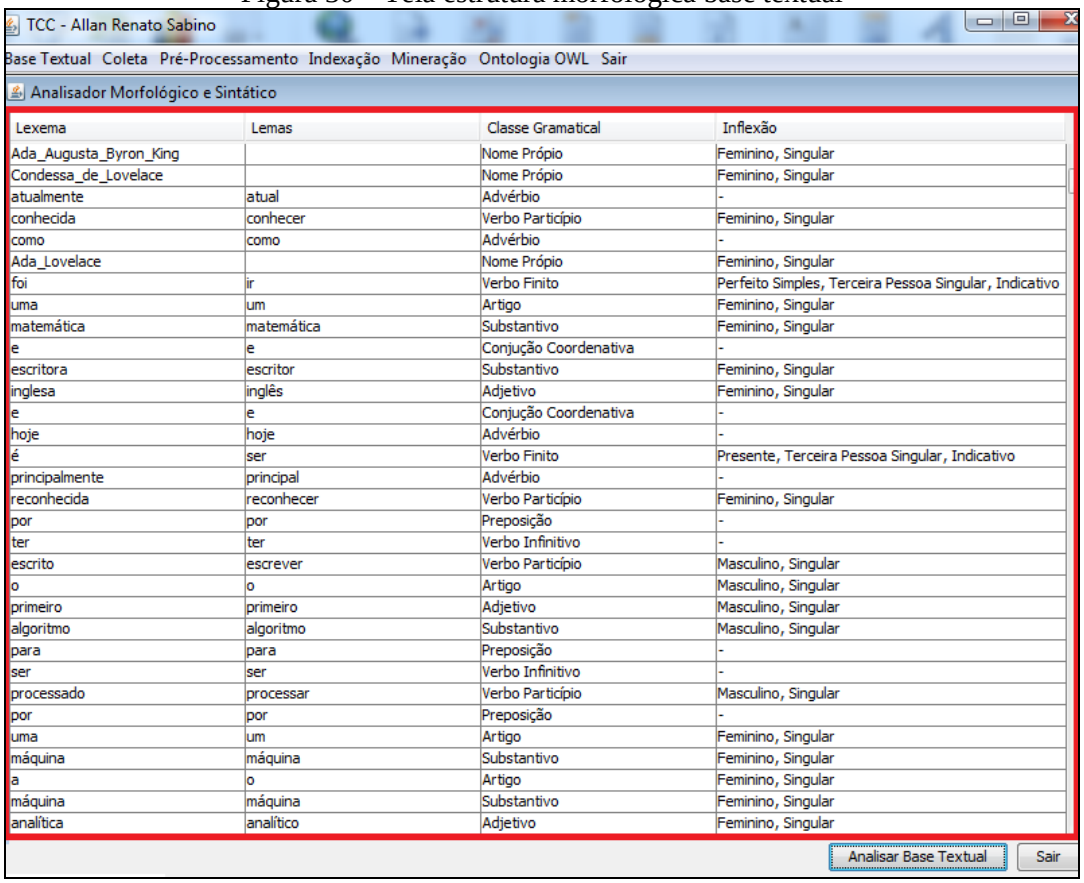
Em seguida deve-se clicar no botão `Analisar Base Textual` (Figura 29).

Figura 29 - Tela de análise morfológica



É então iniciada a análise e todas as palavras são inseridas em uma tabela (Figura 30), onde são mostradas quatro informações: a palavra, seu possível lema, a classe gramatical e suas inflexões.

Figura 30 – Tela estrutura morfológica base textual

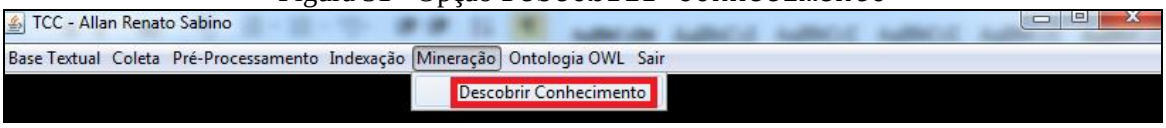


Lexema	Lemas	Classe Gramatical	Inflexão
Ada_Augusta_Byron_King		Nome Própio	Feminino, Singular
Condessa_de_Lovelace		Nome Próprio	Feminino, Singular
atualmente	atual	Advérbio	-
conhecida	conhecer	Verbo Particípio	Feminino, Singular
como	como	Advérbio	-
Ada_Lovelace		Nome Próprio	Feminino, Singular
foi	ir	Verbo Finito	Perfeito Simples, Terceira Pessoa Singular, Indicativo
uma	um	Artigo	Feminino, Singular
matemática	matemática	Substantivo	Feminino, Singular
e	e	Conjução Coordenativa	-
escritora	escritor	Substantivo	Feminino, Singular
inglesa	inglês	Adjetivo	Feminino, Singular
e	e	Conjução Coordenativa	-
hoje	hoje	Advérbio	-
é	ser	Verbo Finito	Presente, Terceira Pessoa Singular, Indicativo
principalmente	principal	Advérbio	-
reconhecida	reconhecer	Verbo Particípio	Feminino, Singular
por	por	Preposição	-
ter	ter	Verbo Infinitivo	-
escrito	escrever	Verbo Particípio	Masculino, Singular
o	o	Artigo	Masculino, Singular
primeiro	primeiro	Adjetivo	Masculino, Singular
algoritmo	algoritmo	Substantivo	Masculino, Singular
para	para	Preposição	-
ser	ser	Verbo Infinitivo	-
processado	processar	Verbo Particípio	Masculino, Singular
por	por	Preposição	-
uma	um	Artigo	Feminino, Singular
máquina	máquina	Substantivo	Feminino, Singular
a	o	Artigo	Feminino, Singular
máquina	máquina	Substantivo	Feminino, Singular
analítica	analítico	Adjetivo	Feminino, Singular

3.3.7.5 Exibir relações ontológicas

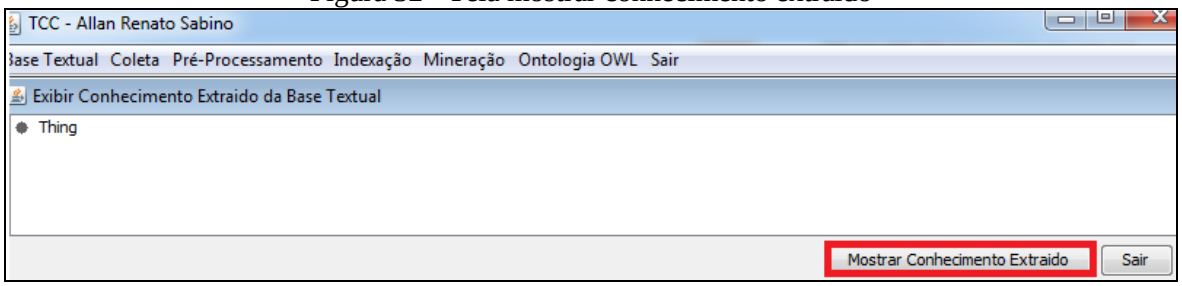
Para se exibir a estrutura ontológica do conhecimento descoberto deve-se clicar no menu `Mineração` e em seguida na opção `Descobrir Conhecimento` (Figura 31).

Figura 31 - Opção `Descobrir Conhecimento`



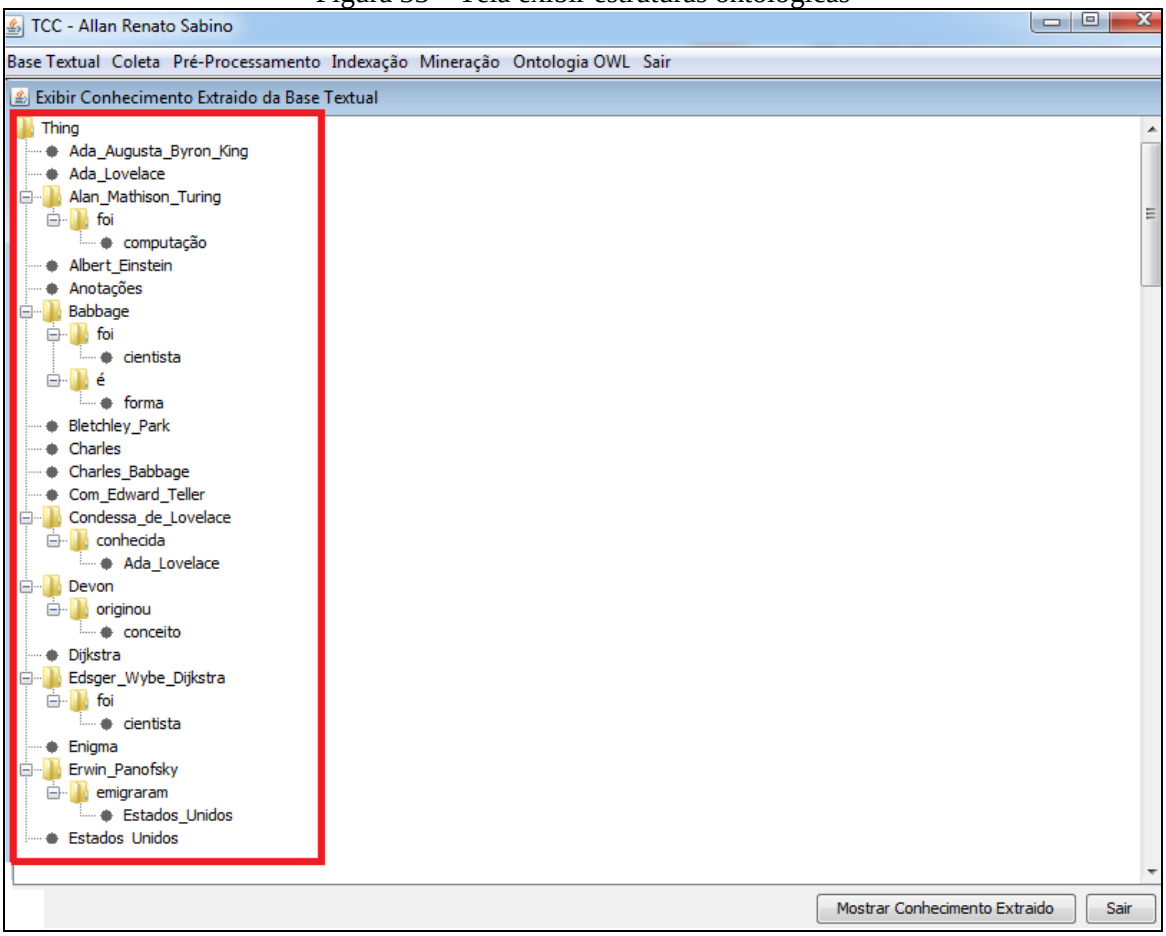
Em seguida deve-se clicar no botão `Mostrar Conhecimento Extraído` (Figura 32).

Figura 32 - Tela mostrar conhecimento extraído



Então é iniciada a descoberta do conhecimento, sendo as estruturas ontológicas (classes e seus relacionamentos) dispostas na forma de uma árvore (Figura 33).

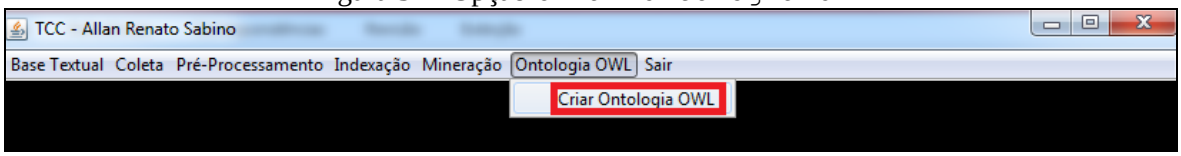
Figura 33 – Tela exibir estruturas ontológicas



3.3.7.6 Exibir ontologia OWL criada

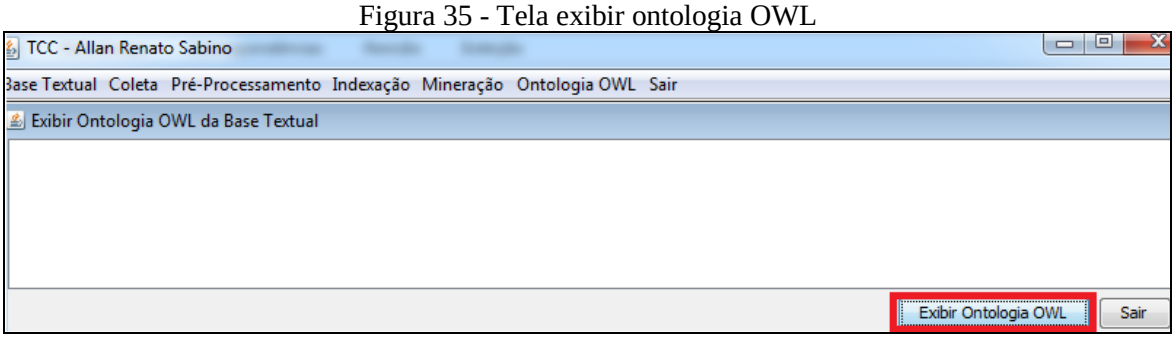
Para se exibir a estrutura ontológica do conhecimento descoberto deve-se clicar no menu `Ontologia OWL` e em seguida na opção `Criar Ontologia OWL` (Figura 34).

Figura 34 - Opção `Criar Ontologia OWL`



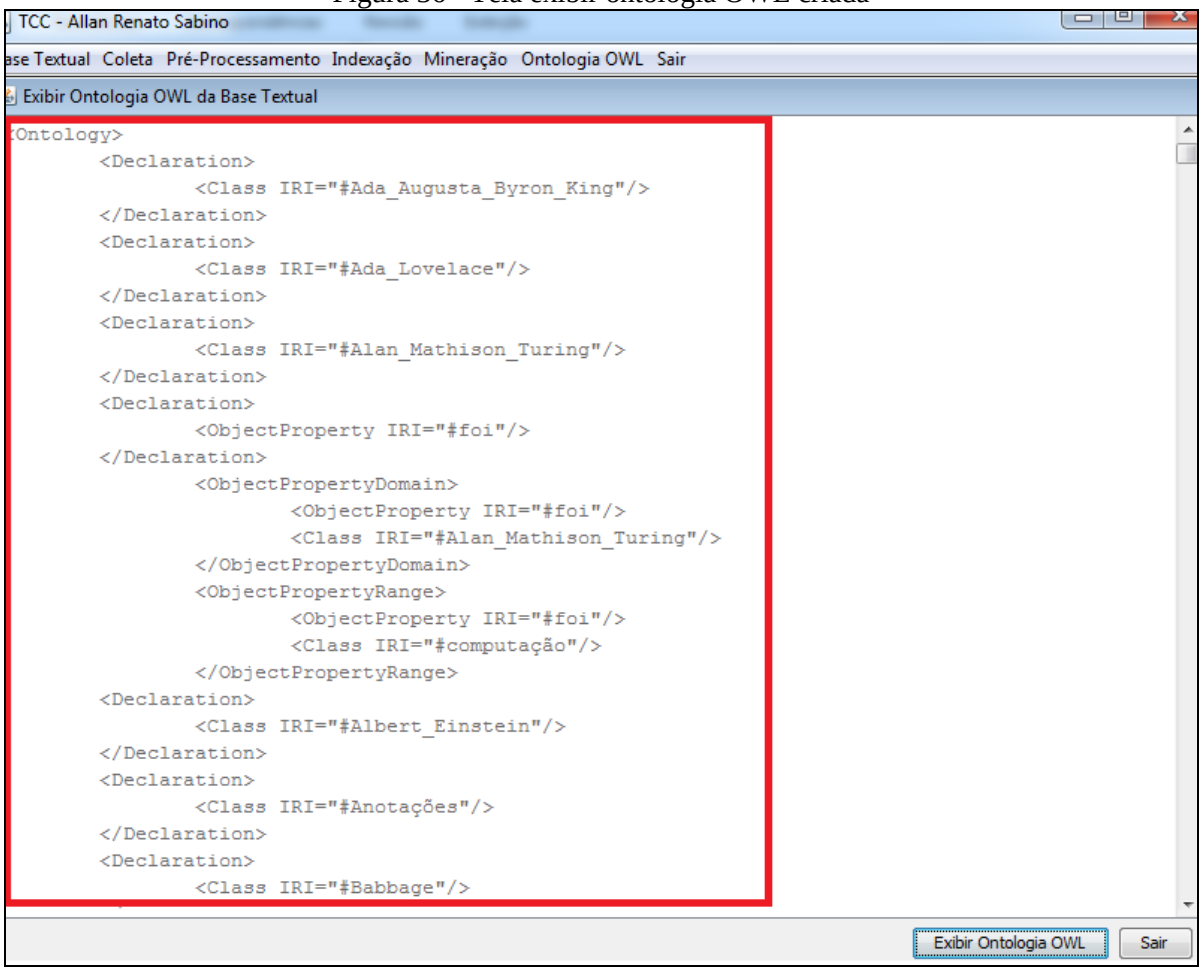
Em seguida deve-se clicar no botão `Exibir Ontologia OWL` (Figura 35).

Figura 35 - Tela exibir ontologia OWL



Um arquivo OWL é criado e gravado em disco, tendo seu conteúdo exibido (Figura 36). Ao exibir a estrutura do arquivo OWL, o gestor do conhecimento pode analisá-lo, caracterizando a etapa de análise do processo de MT.

Figura 36 - Tela exibir ontologia OWL criada



3.4 RESULTADOS E DISCUSSÃO

O presente trabalho apresentou o desenvolvimento de uma ferramenta para criação de bases de conhecimento a partir de dados não estruturados. Para a validação do mesmo, foi realizado um estudo de caso utilizando textos sobre alguns cientistas da computação. A base textual utilizada (disponibilizada no Anexo B) possui cinco arquivos, totalizando seiscentos e noventa e sete palavras.

Todas as classes foram identificadas corretamente. Porém, alguns relacionamentos não foram corretamente identificados. Na frase "Alan Mathinson Turing foi um cientista da computação", o correto seria identificar `Alan Mathinson Turing` e `cientista da computação` como classe e `foi` como relacionamento. Todavia, foi identificado o seguinte: `Alan Mathinson Turing` (identificação do nome composto realizada pelo Cogroo), `cientista` e `computação` foram identificados como classe, ou seja, não foram agrupados os

termos “cientista” e “computação” como sendo uma classe única. A palavra `foi` foi identificada como relacionamento entre `Alan Mathinson Turing` e `computação`.

Esse fato se deu pelo pré-processamento simplista utilizado. Optou-se pelo modelo de representação de documento baseado em palavras ao invés do baseado em termos. Isso fez com que termos compostos fossem separados, gerando assim palavras independentes. Para a solução desse problema, é necessário alterar o modelo de representação de documentos utilizado, passando a fazer uso do modelo baseado em termos

Outra frase que apresentou problema na identificação das estruturas ontológicas foi “Charles Babbage foi um cientista, matemático, filósofo, engenheiro, mecânico e inventor”. As classes (`Charles Babbage`, `cientista`, `matemático`, `filósofo`, `engenheiro`, `mecânico` e `inventor`) e o relacionamento (`foi`) foram identificados com corretude, mas a estrutura ontológica gerada apresentou uma falha, uma vez que `foi` foi identificado como relacionamento apenas entre as classes `Charles Babbage` e `cientista`. Isso se deu pelo padrão que foi utilizado para buscar pelas estruturas ontológicas contidas nas frases. Ele faz uso somente da primeira classe identificada após o relacionamento. Para mitigar isso deve-se colocar todas as classes identificadas em uma lista e, após o final da frase, inserí-las ao relacionamento.

A ontologia gerada (disponibilizada no Apêndice A) foi aberta com a ferramenta Protégé para sua validação. Com isso foi constatada a corretude do arquivo `owl` criado. Na Figura 37 são descritas algumas classes da ontologia e na Figura 38 alguns relacionamentos.

Figura 37 - Classes abertas no Protégé

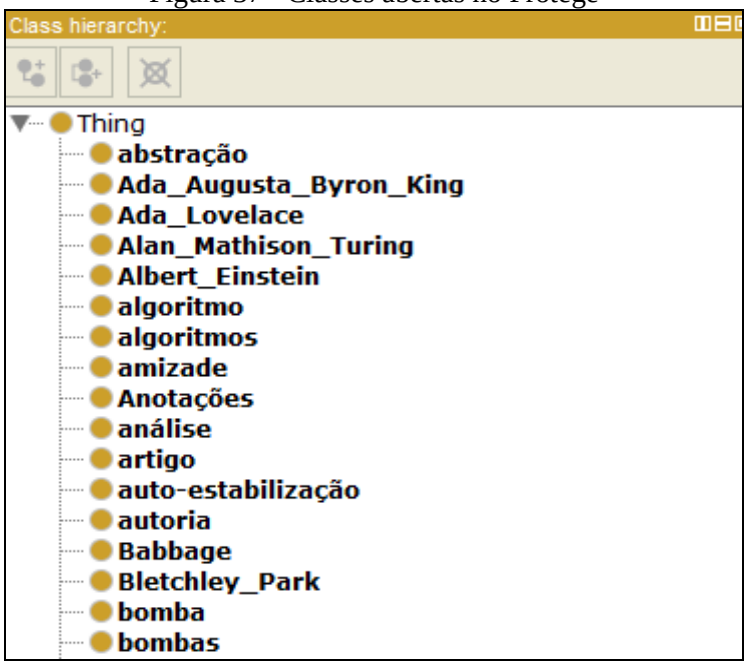
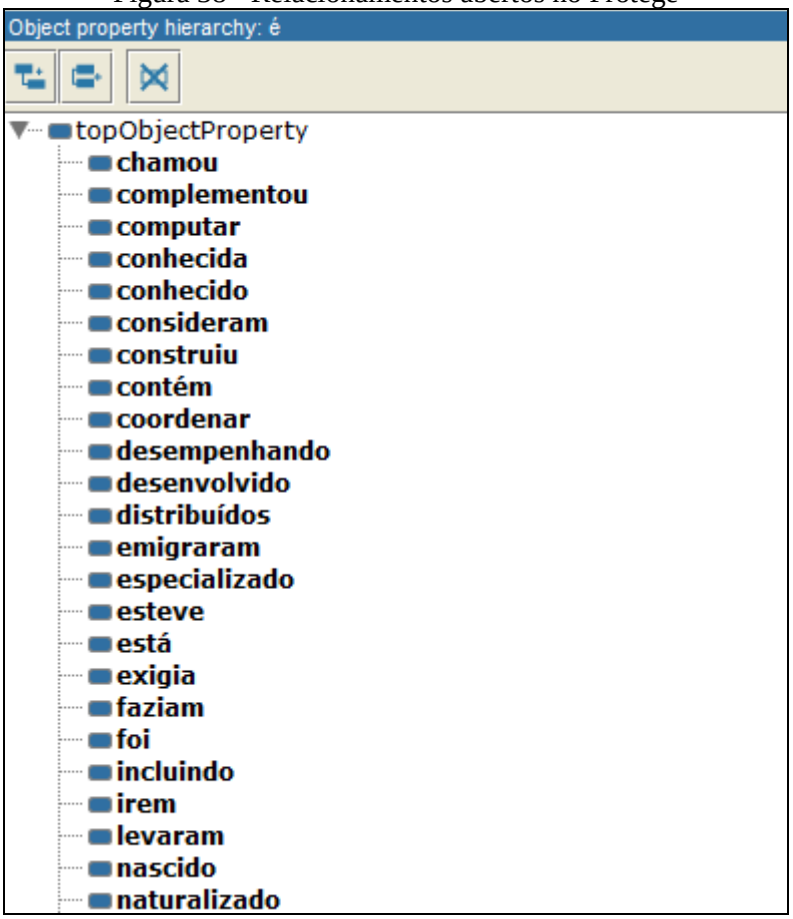


Figura 38 - Relacionamentos abertos no Protégé



Com a ontologia aberta no Protégé pode-se tanto visualizar seus componentes como editá-los, podendo, assim, agregar recursos não suportados pela ferramenta desenvolvida, como por exemplo, características nas propriedades. A Figura 39 exibe as seguintes informações da propriedade `nascido`: características (*characteristics*), domínio (*domain*) e escopo (*range*).

Figura 39 - Propriedades do relacionamento `nascido`

O Quadro 29 descreve as semelhanças e diferenças entre os trabalhos correlatos e trabalho desenvolvido.

Quadro 29 - Comparando os trabalhos correlatos com o trabalho desenvolvido

Características	**Trabalho Desenvolvido**	**Baségio (2007)**	**Faria e Girardi (2010)**	**Silva (2004)**
Identificar estruturas ontológicas	X	X	-	-
Popular ontologia OWL	-	-	X	-
Criar Ontologia OWL	X	X	-	-
AM	-	-	-	X
Remover *stopwords*	X	X	-	X
Lematizar palavras	X	X	-	X
Realizar mineração	X	X	X	X
Realizar análise morfológica	X	-	X	X
Realizar coleta	X	-	-	-
Criar arquivo de índice	X	X	-	-

Pode-se perceber que o trabalho de Faria e Girardi (2010) dispõe de uma funcionalidade única que é popular uma ontologia OWL existente com dados provenientes de uma base textual. O trabalho de Silva (2004) é o único que faz uso de algoritmos de AM. A saída inicial gerada pelo processo de MT é utilizada como entrada para estes algoritmos, que através de sucessivas iterações refinam o resultado, até torná-lo aceitável à sua finalidade

O trabalho de Baségio (2007) é o que mais se aproxima da ferramenta desenvolvida. Ambos visam identificar as estruturas ontológicas em uma base textual e, a partir das mesmas, criar uma ontologia OWL. Porém, seu trabalho não realiza a coleta dos documentos que compõem a base textual e nem a análise morfológica. Ele parte de uma base de documentos organizada e anotada morfologicamente de forma manual.

Nota-se que todos os trabalhos utilizam técnicas de MT para a extração de conhecimento, fazendo uso de técnicas de PNL como apoio. As técnicas de PLN utilizadas foram: remoção de *stopwords*, lematização e análise morfológica.

4 CONCLUSÕES

O avanço das tecnologias para aquisição e armazenamento de dados permitiu que o volume de informação em formato digital tenha aumentado significativamente nas organizações, sendo que 80% dos dados encontram-se em formato textual. Porém, dados não provêm insumos suficientes, ao gestor da organização para a tomada de decisão, devido sua baixa semântica. A transformação de dados brutos e informação em conhecimento facilita sua análise e compreensão. Entretanto, os repositórios de dados foram se tornando cada vez maiores, alcançando grandezas que tornam inviável o modelo de análise manual, necessitando de uma solução computacional.

Com essa premissa, foi desenvolvida uma ferramenta que possibilita a descoberta de conhecimento em uma base textual, formalizando-o através de uma ontologia OWL. Esse fato faz com que o conhecimento formalizado possa ser utilizado por soluções computacionais que detém faculdade de raciocínio, ajudando assim o gestor do conhecimento na tomada de decisão na organização. A ontologia gerada pela ferramenta, pode ser aberta pelo Protégé, facilitando sua análise. Assim sendo, é possível editar a ontologia, adicionando componentes da linguagem OWL que não são suportados pela ferramenta desenvolvida. Esta atualmente possui somente suporte para identificar e formalizar classes e relacionamentos (com seu devido domínio e escopo).

Para implementar a ferramenta foram estudados métodos de descoberta de conhecimento. A escolha pela MT se deu por dois motivos. O primeiro é por ela ser uma especialização da DCBD, criada especialmente para lidar com um cenário onde a quantidade de informações armazenadas em formato textual são de 80% e somente 20% são dados estruturados presentes em bases de dados. O segundo motivo é por ela ser uma das técnicas de DCT mais utilizadas em trabalhos acadêmicos, fato esse detectado ao efetuar a revisão bibliográfica sobre descoberta de conhecimento em bases textuais.

A escolha pela MT fez com que houvesse a necessidade de estudar PLN. Fazendo uso do conhecimento linguístico, o PLN permite extrair ao máximo as características do texto, sendo utilizada durante duas etapas da MT: pré-processamento e mineração. Na primeira etapa ela foi utilizada na remoção das *stopwords*, lematização e análise morfológica das palavras presentes na base textual de trabalho. Na segunda foi utilizada para buscar pelos padrões de descoberta de conhecimento adotados para a implementação da ferramenta (descritos na seção 2.2).

Por fim, estudou-se sobre formalismos de RC, sendo elencado para este trabalho a ontologia escrita com a linguagem de representação OWL. Esta foi escolhida pois a W3C, órgão regulador da internet, adotou esse formalismo como padrão para as ontologias na *web*.

As ferramentas utilizadas para o desenvolvimento da ferramenta se mostraram eficientes, facilitando a implementação de algumas funcionalidades. O analisador morfológico Cogroo foi imprescindível para o desenvolvimento da ferramenta. Ele possui a vantagem de não necessitar conexão com a internet. O uso da linguagem de programação Java facilitou o uso do Cogroo. Como ele é implementado nessa linguagem, basta baixar os fontes e integrá-lo ao projeto. Também foi utilizada a biblioteca PTStemmer para realizar a lematização das palavras da base textual. A mesma também possui implementação em Java, seguindo a mesma ideia do Cogroo de baixar os fontes e inserí-los no projeto.

O objetivo principal do presente trabalho, disponibilizar uma ferramenta para a criação de bases de conhecimento na forma de uma ontologia OWL a partir de textos não estruturados, foi atendido. Os objetivos específicos também foram atendidos, assim sendo, a ferramenta possibilita a visualização dos documentos que compõem a base textual, disponibiliza a estrutura morfológica da base textual e apresenta as estruturas ontológicas a partir do conhecimento extraído.

Porém, duas escolhas feitas durante a revisão bibliográfica tiveram impacto sobre a qualidade do conhecimento descoberto. A primeira delas foi a adoção de um pré-processamento simples, baseado no modelo de representação de palavras. A partir dela, foi concluído que quanto mais elaborado o pré-processamento, melhor são os resultados obtidos. Outro fator foi a escolha do padrão para descoberta do conhecimento. Ele se mostrou efetivo para a descoberta das classes da ontologia, porém simplista demais para os relacionamentos. Tem-se como alternativa utilizar estruturas sintáticas para melhorar a qualidade do conhecimento descoberto.

4.1 EXTENSÕES

Como extensões para a continuação do presente trabalho tem-se:

a) permitir o cadastro de *stopwords*;
b) permitir o cadastro de caracteres especiais;
c) permitir a escolha do algoritmo de lematização utilizado;
d) utilizar o modelo de representação baseado em termos;
e) permitir a identificação de abreviações;
f) permitir a identificação de símbolos da internet;

g) utilizar sinônimos durante o pré-processamento para diminuir a dimensionalidade dos documentos;
h) utilizar análise sintática para a descoberta de conhecimento;
i) implementar um analisador morfológico próprio;
j) identificar herança de classes;
k) popular ontologia com objetos a partir do conhecimento da base textual;
l) identificar propriedades das propriedades (transitiva, simétrica, entre outras);
m) identificar restrições nas propriedades;
n) identificar equivalências entre classes, ou seja, classes com nomes diferentes, mas mesma semântica;
o) identificar equivalência de objetos.

REFERÊNCIAS

ACM. Association of Computation Machinery. In: INTERNATIONAL CONFERENCE ON RESEARCH AND DEVELOPMENT IN INFORMATION RETRIVAL, 18., 1996, Zurich. **Proceedings...** Zurich: ACM, 1996. p.18-22.

ARANHA, Christian. et al. Um modelo de desambiguização de palavras e contextos. In: TIL: WORKSHOP DE TECNOLOGIA DA INFORMAÇÃO E DA LINGUAGEM HUMANA, 2., 2004, São Carlos. **Anais...** São Carlos: Sociedade Brasileira de Computação, 2004. p. 101-110.

ARANHA, Christian N. **Uma abordagem de pré-processamento automático para mineração de textos em português:** sob o enfoque da inteligência computacional. 2007. 144 f. Tese (Doutorado em Engenharia Elétrica) – Programa de Pós-Graduação em Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro. Disponível em: <http://www.maxwell.lambda.ele.puc-rio.br/10081/10081_1.pdf>. Acesso em: 09 abr. 2012.

ARAÚJO, Paulo H.M**. Utilização de Redes Bayesianas na Representação do Conhecimento Empírico.** 2003. 105 f. Dissertação (Mestrado em Gestão do Conhecimento e da Tecnologia da Informação) - Programa de Pós-Graduação em Gestão do Conhecimento e da Tecnologia da Informação, Universidade Católica de Brasília, Brasília.

BASÉGIO, Túlio L**. Uma abordagem semiautomática para identificação de estruturas ontológicas a partir de textos na língua portuguesa do brasil.** 2007. 124 f. Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre. Disponível em: <http://tede.pucrs.br/tde_arquivos/4/TDE-2009-06-09T170445Z-1994/Publico/403014.pdf>. Acesso em: 12 ago. 2013.

BASTOS, Vitor M. **Ambiente de descoberta de conhecimento na web para a língua portuguesa**. 2006. 144 f. Tese (Doutorado em Engenharia Civil) – Programa de Pós-Graduação em Engenharia Civil, Universidade Federal do Rio de Janeiro, Rio de Janeiro.

BARANAUSKAS, José A. **Extração automática de conhecimento por múltiplos indutores**. 2001. 201 f. Tese (Doutorado em Ciência da Computação e Matemática Computacional) – Programa de Pós-Graduação em Ciência da Computação e Matemática Computacional, Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo, São Paulo.

BATISTA, Gustavo E. A. P. A. **Pré-processamento de dados em aprendizado de máquina supervisionado**. 2003. 204 f. Tese (Doutorado em Ciência da Computação e Matemática Computacional) – Programa de Pós-Graduação em Ciência da Computação e Matemática Computacional, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Paulo.

BRANCO, Vinícius M. A. **Visualização de informação e mineração de dados visuais**. 2003. 150 f. Dissertação (Mestrado em Ciência da Computação e Matemática Computacional) – Programa de Pós-Graduação em Ciência da Computação e Matemática Computacional, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Paulo.

BREITMAN, Karin K. **Web semântica:** a internet do futuro. Rio de Janeiro: LTC, 2005.

BREITMAN, Karin K.; LEITE, Júlio C. S. P. Ontologias – como e porquê criá-las. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 24., 2004, Salvador. **Anais...** Salvador: Sociedade Brasileira de Computação, 2005. p. 3-53.

CAMILO, Cassio O.; SILVA, João C. **Mineração de dados**: conceitos, tarefas, métodos e ferramentas. Goiânia, 2009. Disponível em: <http://www.inf.ufg.br/sites/default/files/uploads/relatorios-tecnicos/RT-INF_001-09.pdf>. Acesso em: 22 mar. 2014.

CAMPOS, Teófilo E. **Técnicas de seleção de características com aplicações em reconhecimento de faces.** 2001. 136 f. Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação**,** Universidade de São Paulo, São Paulo.

CARRILHO JUNIOR, João R. **Desenvolvimento de uma metodologia para mineração de textos**. 2007. 96 f. Dissertação (Mestrado em Engenharia Elétrica) – Programa de Pós-Graduação em Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

CARVALHO, Cedric L.; VAZ, Fernando R. **Visualização de informações**. Goiânia, 2004. Disponível em: < http://www.inf.ufg.br/sites/default/files/uploads/relatorios-tecnicos/RT-INF_003-04.pdf >. Acesso em: 8 maio 2014.

CASTRO, Armando A. M.; PRADO, Pedro P**.** L. Algoritmos para reconhecimento de padrões. **Revista de Ciências Exatas**, Taubaté, v. 5, n. 3, p. 129-145, dez. 2008.

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

CEGALLA, Domingos P. **Novíssima gramatica da língua portuguesa**. São Paulo: Nacional, 2005.

CHEN, Hsinchun. **Knowledge management systems:** a text mining perspective. Tucson: University of Arizona, 2001.

CORRÊA, Adriana C. G. **Recuperação de documentos baseada em informação semântica no ambiente AMMO.** São Carlos, 2003. Disponível em: <http://www.bdtd.ufscar.br/tde_busca/arquivo.php?codArquivo=485>. Acesso em: 20 mar. 2014.

DACONTA, Michael C.; OBRST, Leo J.; SMITH, Kevin T. **The semantic web**. San Francisco: Wiley, 2003.

DATE, Christopher J. **Introdução a sistemas de bancos de dados**. 8 ed. São Paulo: Campus, 2005.

FARIA, Carla; GIRARDI, Rosario. **Um processo semiautomático para o povoamento de ontologias a partir de fontes textuais.** São Luís, 2010. Disponível em: <http://www.seer.unirio.br/index.php/isys/article/viewFile/1258/1178> **.** Acesso em: 15**.** ago**. 2013.**

FELDMAN, Ronen ; SANGER, John. **The text mining handbook**: advanced approaches in analyzing unstructured data. Cambridge: Cambridge Press, 2007.

FIALHO, Francisco et al. **Gestão do conhecimento e aprendizagem**: as estratégias competitivas da sociedade pós-industrial**.** Florianópolis: Visual Books, 2006.

FIALHO, Francisco et al. **Gestão do conhecimento organizacional.** Florianópolis: UFSC, 2010.

GOLDSCHMIDT, Ronaldo; PASSOS, Emmanuel L. **Data mining**: um guia prático. Rio de Janeiro: Campus, 2005.

GRUBER, Tom R. A translation approach to portable ontologies. **Knowledge Acquisition**, New York, v. 5, n. 2, p. 199-220, fev.1993.

GUARINO, Nicola. Understanding, building and using ontologies: a commentary to using explicit ontologies in KBS development. **International Journal of Human and Computer Studies**, Toronto, v. 10, n. 3, p.293-310, out. 1997.

HAN, Jiawei; KAMBER, Micheline. **Data mining**: concepts and techniques. 2 nd. ed. New Jersey: Morgan Kaufmann, 2006.

HEFLIN, Jeff. **OWL web ontology language use cases and requirements** - W3C recommendation 10 February 2004. Massachusetts, 2004. Disponível em: <http://www.w3.org/TR/2004/REC-webont-req-20040210/>. Acesso em: 16 mar. 2014.

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.

KANTARDZIC, Mehmed. **Data mining**: concepts, models, methods and algorithms. San Francisco: Wiley, 2002.

KUECHLER, Willian L. Business applications of unstructured text. **Communications of ACM**, New York, v. 50, n. 10, p. 86–93, out. 2007.

KONCHADY, Manu. **Text mining application programming**. Newton: Charles River Media, 2006.

LADEIRA, Marcelo**. Representação de conhecimento e redes de decisão**. 1997. 150f. Tese (Doutorado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre.

MARCACINI, Ricardo M.; MOURA, Maria F.; REZENDE, Solange O. O uso da mineração de textos para extração e organização não supervisionada de conhecimento. **Revista de Sistemas de Informação da FSMA**, Macaé, v. 1, n.7, p. 7-21, nov. 2011.

MACEDO, Marcelo J. C. **Processamento da linguagem natural para identificação de classes e instâncias de uma ontologia.** 2010. 90 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Tecnologia, Universidade Federal do Maranhão, São Luís.

MANNING, Christopher D.; RAGHAVAN, Prabhakar; SCHUTZE, Hinrich. **Introduction to information retrival.** Cambridge: University Press, 2007.

MARTINS, Claudia A. **Uma abordagem para pré-processamento de dados textuais em algoritmos de aprendizado**. 2003. 208 f. Tese (Doutorado em Ciência da Computação e Matemática Computacional) – Programa de Pós-Graduação em Ciência da Computação e Matemática Computacional, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Paulo.

MASTELLA, Laura S. **Técnicas de aquisição de conhecimento para sistemas baseados em conhecimento**. Porto Alegre, 2004. Disponível em: <http://www.inf.ufrgs.br/gpesquisa/bdi/publicacoes/files/TI1LSM.pdf>. Acesso em: 1 mar.2014.

MELO, Francisco R. **Modelo neural por padrões proximais de aprendizagem para automação personalizada de conteúdos didáticos**. 2012. 157 f. Tese (Doutorado em Engenharia Elétrica) – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Uberlândia, Uberlândia.

MOREIRA, Alexandra; ALVARENGA, Lidia; OLIVEIRA, Alcione P. O nível do conhecimento e os instrumentos de representação: tesauros e ontologias**. DataGramaZero – Revista de Ciência da Informação**, Rio de Janeiro, v. 5, n. 6,dez. 2004**.** Disponível em: <http://www.datagramazero.org.br/dez04/Art_01.htm>. Acesso em: 12 mar. 2014.

NONAKA, Ikujiro; TAKEUCHI, Hirotaka. **Criação de conhecimento na empresa**. Rio de Janeiro: Campus, 1997.

NONAKA, Ikujiro; TAKEUCHI, Hirotaka. **Gestão do conhecimento.** Porto Alegre: Bookman, 2008.

NOVELLO, Taisa C. **Ontologias, sistemas baseados em conhecimento e modelos de banco de dados**. Porto Alegre, 2003. Disponível em: <http://www.inf.ufrgs.br/~clesio/cmp151/cmp15120021/artigo_taisa.pdf>. Acesso em: 12 mar. 2014.

NOY, Natasha F.; HAFNER, Carole D. The state of the art in ontology design: a survey and comparative Review. **AI Magazine**, New Orleans, v. 18, n. 3, p. 53-74, set. 1997.

OLIVEIRA, Kevin. **Modelo para construção de ambientes de desenvolvimento de software orientados a domínio**. 1999. 222f. Tese (Doutorado em Engenharia de Sistemas e Computação) - Programa de Pós-Graduação em Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro.

PROTÉGÉ. W**hat is protégé-owl?** [Califórnia], [2013]. Disponível em: <http://protege.stanford.edu/overview/protege-owl.html>. Acesso em: 4 maio 2014.

PFLEEGER, Shari L. **Engenharia de software:** teoria e prática. São Paulo: Person, 2004.

POWEL, Gavin. **Beginning XML databases**. San Francisco: Wiley, 2007.

RDF WORKING GROUP. **Resource Description Framework (RDF)**. [S.l.], 2004. Disponível em: <http://www.w3.org/RDF/>. Acesso em: 15 mar. 2014.

SPENCE, Robert. **Information visualization**. Boston: Addison-Wesley, 2001.

SALTON, Gerard. **Introduction to modern information retrieval**. New York: McGraw-Hill, 1983.

SCHAFFER, Christian. A conservation law for generalization performance. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 11., 1994, New Jersey. **Proceedings...** New Jersey: Morgan Kaufmann, 1994. p. 259-265.

SILVA, Eduardo F. do A. **Um sistema para extração de informação em referências bibliográficas baseado em aprendizado de máquina.** 2004. 95 f. Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Pernambuco, Recife. Disponível em: <http://www.cin.ufpe.br/~rbcp/artigos/tese_efas_versao_corrigida.pdf>. Acesso em: 10 ago. 2013.

SMITH, Michael K.; WELTY, Chris; MCGUINNESS, Deborah L. **OWL web ontology language guide** – W3C recommendation 10 February 2004. Massachusetts, 2004. Disponível em: <http://www.w3.org/TR/owl-guide/>. Acesso em: 3 ago. 2013.

SOARES, Fabio de A. **Categorização automática de textos baseada em mineração de texto**. 2013. 158 f. Tese (Doutorado em Engenharia Elétrica) – Programa de Pós-Graduação em Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

SOUZA, Lucas C. G. **Regras de raciocínio aplicadas a ontologias por meio de sistema multi agente para apoio a decisões organizacionais**. 2003. 173 f. Dissertação (Mestrado em Informática Aplicada) - Centro de Ciências Exatas e de Tecnologia, Universidade Católica do Paraná, Curitiba.

TARAPANOFF, Kira. **Inteligência, informação e conhecimento.** Brasília: Instituto Brasileiro de Informação em Ciência e Tecnologia, 2006.

W3C. **DAML+OIL reference description** - W3C note 18 December 2001**.** Massachusetts, 2001. Disponível em: <http://www.w3.org/TR/daml+oilreference>. Acesso em: 15 mar. 2014.

W3C. **OWL web ontology language overview** - W3C recommendation 10 February 2004. Massachusetts, 2004. Disponível em: <http://www.w3.org/TR/owl-features>. Acesso em: 4 ago. 2013.

WIVES, Leandro. **Recursos de text mining**. Porto Alegre, 2000. Disponível em: <http://www.inf.ufrgs.br/~wives/portugues/textmining.html>. Acesso em: 19 mar. 2014.

ZHU, Xingquan; DAVIDSON, Ian. **Knowledge discovery and data mining**: challenges and realities. New York: Hershey, 2007.

APÊNDICE A – Ontologia gerada com a ferramenta proposta

O Quadro 30 apresenta a ontologia gerada pela ferramenta desenvolvida no estudo de caso apresentado na seção 3.4.

Quadro 30 - Ontologia gerada com a ferramenta desenvolvida

```
<Ontology>
<Declaration>
	<Class IRI="#Ada_Augusta_Byron_King"/>
</Declaration>
<Declaration>
	<Class IRI="#Ada_Lovelace"/>
</Declaration>
<Declaration>
	<Class IRI="#Alan_Mathison_Turing"/>
</Declaration>
<Declaration>
	<ObjectProperty IRI="#foi"/>
</Declaration>
	<ObjectPropertyDomain>
		<ObjectProperty IRI="#foi"/>
		<Class IRI="#Alan_Mathison_Turing"/>
	</ObjectPropertyDomain>
	<ObjectPropertyRange>
		<ObjectProperty IRI="#foi"/>
		<Class IRI="#computação"/>
	</ObjectPropertyRange>
<Declaration>
	<Class IRI="#Albert_Einstein"/>
</Declaration>
<Declaration>
	<Class IRI="#Anotações"/>
</Declaration>
<Declaration>
	<Class IRI="#Babbage"/>
</Declaration>
<Declaration>
	<ObjectProperty IRI="#foi"/>
</Declaration>
	<ObjectPropertyDomain>
		<ObjectProperty IRI="#foi"/>
		<Class IRI="#Babbage"/>
	</ObjectPropertyDomain>
	<ObjectPropertyRange>
		<ObjectProperty IRI="#foi"/>
		<Class IRI="#cientista"/>
	</ObjectPropertyRange>
<Declaration>
	<ObjectProperty IRI="#é"/>
</Declaration>
	<ObjectPropertyDomain>
		<ObjectProperty IRI="#é"/>
		<Class IRI="#Babbage"/>
	</ObjectPropertyDomain>
	<ObjectPropertyRange>
		<ObjectProperty IRI="#é"/>
		<Class IRI="#forma"/>
	</ObjectPropertyRange>
<Declaration>
	<Class IRI="#Bletchley_Park"/>
</Declaration>
<Declaration>
```

```
            <Class IRI="#Charles"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Charles_Babbage"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Com_Edward_Teller"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Condessa_de_Lovelace"/>
        </Declaration>
        <Declaration>
            <ObjectProperty IRI="#conhecida"/>
        </Declaration>
            <ObjectPropertyDomain>
                <ObjectProperty IRI="#conhecida"/>
                <Class IRI="#Condessa_de_Lovelace"/>
            </ObjectPropertyDomain>
            <ObjectPropertyRange>
                <ObjectProperty IRI="#conhecida"/>
                <Class IRI="#Ada_Lovelace"/>
            </ObjectPropertyRange>
        <Declaration>
            <Class IRI="#Devon"/>
        </Declaration>
        <Declaration>
            <ObjectProperty IRI="#originou"/>
        </Declaration>
            <ObjectPropertyDomain>
                <ObjectProperty IRI="#originou"/>
                <Class IRI="#Devon"/>
            </ObjectPropertyDomain>
            <ObjectPropertyRange>
                <ObjectProperty IRI="#originou"/>
                <Class IRI="#conceito"/>
            </ObjectPropertyRange>
        <Declaration>
            <Class IRI="#Dijkstra"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Edsger_Wybe_Dijkstra"/>
        </Declaration>
        <Declaration>
            <ObjectProperty IRI="#foi"/>
        </Declaration>
            <ObjectPropertyDomain>
                <ObjectProperty IRI="#foi"/>
                <Class IRI="#Edsger_Wybe_Dijkstra"/>
            </ObjectPropertyDomain>
            <ObjectPropertyRange>
                <ObjectProperty IRI="#foi"/>
                <Class IRI="#cientista"/>
            </ObjectPropertyRange>
        <Declaration>
            <Class IRI="#Enigma"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Erwin_Panofsky"/>
        </Declaration>
        <Declaration>
            <ObjectProperty IRI="#emigraram"/>
        </Declaration>
            <ObjectPropertyDomain>
                <ObjectProperty IRI="#emigraram"/>
                <Class IRI="#Erwin_Panofsky"/>
            </ObjectPropertyDomain>
            <ObjectPropertyRange>
                <ObjectProperty IRI="#emigraram"/>
```

```
                <Class IRI="#Estados_Unidos"/>
            </ObjectPropertyRange>
        <Declaration>
            <Class IRI="#Estados_Unidos"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Física_Nuclear"/>
        </Declaration>
        <Declaration>
            <ObjectProperty IRI="#relacionados"/>
        </Declaration>
            <ObjectPropertyDomain>
                <ObjectProperty IRI="#relacionados"/>
                <Class IRI="#Física_Nuclear"/>
            </ObjectPropertyDomain>
            <ObjectPropertyRange>
                <ObjectProperty IRI="#relacionados"/>
                <Class IRI="#reações"/>
            </ObjectPropertyRange>
        <Declaration>
            <Class IRI="#George_F"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Hermann_Weyl"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Instituto_de_Estudos_Avançados_de_Princeton"/>
        </Declaration>
        <Declaration>
            <Class IRI="#John"/>
        </Declaration>
        <Declaration>
            <ObjectProperty IRI="#von"/>
        </Declaration>
            <ObjectPropertyDomain>
                <ObjectProperty IRI="#von"/>
                <Class IRI="#John"/>
            </ObjectPropertyDomain>
            <ObjectPropertyRange>
                <ObjectProperty IRI="#von"/>
                <Class IRI="#Neumann"/>
            </ObjectPropertyRange>
        <Declaration>
            <Class IRI="#Kennan"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Kurt_Gödel"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Londres"/>
        </Declaration>
        <Declaration>
            <ObjectProperty IRI="#construiu"/>
        </Declaration>
            <ObjectPropertyDomain>
                <ObjectProperty IRI="#construiu"/>
                <Class IRI="#Londres"/>
            </ObjectPropertyDomain>
            <ObjectPropertyRange>
                <ObjectProperty IRI="#construiu"/>
                <Class IRI="#invenções"/>
            </ObjectPropertyRange>
        <Declaration>
            <Class IRI="#Luigi_Menabrea"/>
        </Declaration>
        <Declaration>
            <Class IRI="#Museu_de_Ciência"/>
        </Declaration>
```

```
        <Declaration>
               <Class IRI="#Máquina_Analítica"/>
        </Declaration>
        <Declaration>
               <Class IRI="#Neumann"/>
        </Declaration>
        <Declaration>
               <ObjectProperty IRI="#trabalhou"/>
        </Declaration>
               <ObjectPropertyDomain>
                      <ObjectProperty IRI="#trabalhou"/>
                      <Class IRI="#Neumann"/>
               </ObjectPropertyDomain>
               <ObjectPropertyRange>
                      <ObjectProperty IRI="#trabalhou"/>
                      <Class IRI="#desenvolvimentos"/>
               </ObjectPropertyRange>
        <Declaration>
               <Class IRI="#Nova_Jérsei"/>
        </Declaration>
        <Declaration>
               <ObjectProperty IRI="#faziam"/>
        </Declaration>
               <ObjectPropertyDomain>
                      <ObjectProperty IRI="#faziam"/>
                      <Class IRI="#Nova_Jérsei"/>
               </ObjectPropertyDomain>
               <ObjectPropertyRange>
                      <ObjectProperty IRI="#faziam"/>
                      <Class IRI="#parte"/>
               </ObjectPropertyRange>
        <Declaration>
               <Class IRI="#Projeto_Manhattan"/>
        </Declaration>
        <Declaration>
               <Class IRI="#Prêmio_Turing"/>
        </Declaration>
        <Declaration>
               <Class IRI="#Robert_Oppenheimer"/>
        </Declaration>
        <Declaration>
               <Class IRI="#Segunda_Guerra_Mundial"/>
        </Declaration>
        <Declaration>
               <Class IRI="#Stanislaw_Ulam"/>
        </Declaration>
        <Declaration>
               <Class IRI="#Teignmouth"/>
        </Declaration>
        <Declaration>
               <Class IRI="#Turing"/>
        </Declaration>
        <Declaration>
               <ObjectProperty IRI="#desempenhando"/>
        </Declaration>
               <ObjectPropertyDomain>
                      <ObjectProperty IRI="#desempenhando"/>
                      <Class IRI="#Turing"/>
               </ObjectPropertyDomain>
               <ObjectPropertyRange>
                      <ObjectProperty IRI="#desempenhando"/>
                      <Class IRI="#papel"/>
               </ObjectPropertyRange>
        <Declaration>
               <ObjectProperty IRI="#trabalhou"/>
        </Declaration>
               <ObjectPropertyDomain>
                      <ObjectProperty IRI="#trabalhou"/>
```

```
                <Class IRI="#Turing"/>
            </ObjectPropertyDomain>
            <ObjectPropertyRange>
                <ObjectProperty IRI="#trabalhou"/>
                <Class IRI="#inteligência"/>
            </ObjectPropertyRange>
        <Declaration>
            <Class IRI="#abstração"/>
        </Declaration>
        <Declaration>
            <Class IRI="#algoritmo"/>
        </Declaration>
        <Declaration>
            <Class IRI="#algoritmos"/>
        </Declaration>
        <Declaration>
            <Class IRI="#amizade"/>
        </Declaration>
        <Declaration>
            <Class IRI="#análise"/>
        </Declaration>
        <Declaration>
            <Class IRI="#artigo"/>
        </Declaration>
        <Declaration>
            <Class IRI="#auto-estabilização"/>
        </Declaration>
        <Declaration>
            <Class IRI="#autoria"/>
        </Declaration>
        <Declaration>
            <ObjectProperty IRI="#chamou"/>
        </Declaration>
            <ObjectPropertyDomain>
                <ObjectProperty IRI="#chamou"/>
                <Class IRI="#autoria"/>
            </ObjectPropertyDomain>
            <ObjectPropertyRange>
                <ObjectProperty IRI="#chamou"/>
                <Class IRI="#Anotações"/>
            </ObjectPropertyRange>
        <Declaration>
            <Class IRI="#bomba"/>
        </Declaration>
        <Declaration>
            <Class IRI="#bombas"/>
        </Declaration>
        <Declaration>
            <Class IRI="#caminho"/>
        </Declaration>
        <Declaration>
            <ObjectProperty IRI="#conhecido"/>
        </Declaration>
            <ObjectPropertyDomain>
                <ObjectProperty IRI="#conhecido"/>
                <Class IRI="#caminho"/>
            </ObjectPropertyDomain>
            <ObjectPropertyRange>
                <ObjectProperty IRI="#conhecido"/>
                <Class IRI="#algoritmo"/>
            </ObjectPropertyRange>
        <Declaration>
            <Class IRI="#capacidade"/>
        </Declaration>
        <Declaration>
            <Class IRI="#capacidades"/>
        </Declaration>
        <Declaration>
```

```
        <Class IRI="#caras"/>
    </Declaration>
    <Declaration>
        <Class IRI="#centro"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="#especializado"/>
    </Declaration>
        <ObjectPropertyDomain>
            <ObjectProperty IRI="#especializado"/>
            <Class IRI="#centro"/>
        </ObjectPropertyDomain>
        <ObjectPropertyRange>
            <ObjectProperty IRI="#especializado"/>
            <Class IRI="#quebra"/>
        </ObjectPropertyRange>
    <Declaration>
        <Class IRI="#chave"/>
    </Declaration>
    <Declaration>
        <Class IRI="#chefe"/>
    </Declaration>
    <Declaration>
        <Class IRI="#cientista"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="#foi"/>
    </Declaration>
        <ObjectPropertyDomain>
            <ObjectProperty IRI="#foi"/>
            <Class IRI="#cientista"/>
        </ObjectPropertyDomain>
        <ObjectPropertyRange>
            <ObjectProperty IRI="#foi"/>
            <Class IRI="#ensaios"/>
        </ObjectPropertyRange>
    <Declaration>
        <Class IRI="#ciência"/>
    </Declaration>
    <Declaration>
        <Class IRI="#colega"/>
    </Declaration>
    <Declaration>
        <Class IRI="#coleção"/>
    </Declaration>
    <Declaration>
        <Class IRI="#complexidade"/>
    </Declaration>
    <Declaration>
        <Class IRI="#computador"/>
    </Declaration>
    <Declaration>
        <Class IRI="#computadores"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="#irem"/>
    </Declaration>
        <ObjectPropertyDomain>
            <ObjectProperty IRI="#irem"/>
            <Class IRI="#computadores"/>
        </ObjectPropertyDomain>
        <ObjectPropertyRange>
            <ObjectProperty IRI="#irem"/>
            <Class IRI="#mero"/>
        </ObjectPropertyRange>
    <Declaration>
        <Class IRI="#computação"/>
    </Declaration>
```

```
	<Declaration>
		<ObjectProperty IRI="#conhecido"/>
	</Declaration>
		<ObjectPropertyDomain>
			<ObjectProperty IRI="#conhecido"/>
			<Class IRI="#computação"/>
		</ObjectPropertyDomain>
		<ObjectPropertyRange>
			<ObjectProperty IRI="#conhecido"/>
			<Class IRI="#contribuições"/>
		</ObjectPropertyRange>
	<Declaration>
		<ObjectProperty IRI="#está"/>
	</Declaration>
		<ObjectPropertyDomain>
			<ObjectProperty IRI="#está"/>
			<Class IRI="#computação"/>
		</ObjectPropertyDomain>
		<ObjectPropertyRange>
			<ObjectProperty IRI="#está"/>
			<Class IRI="#algoritmo"/>
		</ObjectPropertyRange>
	<Declaration>
		<Class IRI="#computação.5"/>
	</Declaration>
	<Declaration>
		<Class IRI="#conceito"/>
	</Declaration>
	<Declaration>
		<ObjectProperty IRI="#desenvolvido"/>
	</Declaration>
		<ObjectPropertyDomain>
			<ObjectProperty IRI="#desenvolvido"/>
			<Class IRI="#conceito"/>
		</ObjectPropertyDomain>
		<ObjectPropertyRange>
			<ObjectProperty IRI="#desenvolvido"/>
			<Class IRI="#cientista"/>
		</ObjectPropertyRange>
	<Declaration>
		<Class IRI="#confiança"/>
	</Declaration>
	<Declaration>
		<Class IRI="#conjunto"/>
	</Declaration>
	<Declaration>
		<Class IRI="#conjuntos"/>
	</Declaration>
	<Declaration>
		<Class IRI="#construção"/>
	</Declaration>
	<Declaration>
		<Class IRI="#contribuições"/>
	</Declaration>
	<Declaration>
		<Class IRI="#criadores"/>
	</Declaration>
	<Declaration>
		<Class IRI="#criação"/>
	</Declaration>
	<Declaration>
		<Class IRI="#criptoanálise"/>
	</Declaration>
	<Declaration>
		<Class IRI="#cálculo"/>
	</Declaration>
	<Declaration>
		<Class IRI="#códigos"/>
```

```
	</Declaration>
	<Declaration>
		<Class IRI="#definições"/>
	</Declaration>
	<Declaration>
		<Class IRI="#desenvolvimento"/>
	</Declaration>
	<Declaration>
		<Class IRI="#desenvolvimentos"/>
	</Declaration>
	<Declaration>
		<Class IRI="#economia"/>
	</Declaration>
	<Declaration>
		<Class IRI="#engenheiro"/>
	</Declaration>
	<Declaration>
		<Class IRI="#ensaios"/>
	</Declaration>
	<Declaration>
		<Class IRI="#escritora"/>
	</Declaration>
	<Declaration>
		<ObjectProperty IRI="#é"/>
	</Declaration>
		<ObjectPropertyDomain>
			<ObjectProperty IRI="#é"/>
			<Class IRI="#escritora"/>
		</ObjectPropertyDomain>
		<ObjectPropertyRange>
			<ObjectProperty IRI="#é"/>
			<Class IRI="#algoritmo"/>
		</ObjectPropertyRange>
	<Declaration>
		<Class IRI="#estadunidense"/>
	</Declaration>
	<Declaration>
		<Class IRI="#estatística"/>
	</Declaration>
	<Declaration>
		<Class IRI="#explosões"/>
	</Declaration>
	<Declaration>
		<Class IRI="#filósofo"/>
	</Declaration>
	<Declaration>
		<Class IRI="#forma"/>
	</Declaration>
	<Declaration>
		<Class IRI="#formalização"/>
	</Declaration>
	<Declaration>
		<Class IRI="#frota"/>
	</Declaration>
	<Declaration>
		<Class IRI="#funções"/>
	</Declaration>
	<Declaration>
		<ObjectProperty IRI="#publicar"/>
	</Declaration>
		<ObjectPropertyDomain>
			<ObjectProperty IRI="#publicar"/>
			<Class IRI="#funções"/>
		</ObjectPropertyDomain>
		<ObjectPropertyRange>
			<ObjectProperty IRI="#publicar"/>
			<Class IRI="#coleção"/>
		</ObjectPropertyRange>
```

```
	<Declaration>
		<Class IRI="#hidrodinâmica"/>
	</Declaration>
	<Declaration>
		<Class IRI="#hidrogênio"/>
	</Declaration>
	<Declaration>
		<Class IRI="#história"/>
	</Declaration>
	<Declaration>
		<Class IRI="#inteligência"/>
	</Declaration>
	<Declaration>
		<Class IRI="#invento"/>
	</Declaration>
	<Declaration>
		<ObjectProperty IRI="#exigia"/>
	</Declaration>
		<ObjectPropertyDomain>
			<ObjectProperty IRI="#exigia"/>
			<Class IRI="#invento"/>
		</ObjectPropertyDomain>
		<ObjectPropertyRange>
			<ObjectProperty IRI="#exigia"/>
			<Class IRI="#técnicas"/>
		</ObjectPropertyRange>
	<Declaration>
		<Class IRI="#inventor"/>
	</Declaration>
	<Declaration>
		<ObjectProperty IRI="#nascido"/>
	</Declaration>
		<ObjectPropertyDomain>
			<ObjectProperty IRI="#nascido"/>
			<Class IRI="#inventor"/>
		</ObjectPropertyDomain>
		<ObjectPropertyRange>
			<ObjectProperty IRI="#nascido"/>
			<Class IRI="#Teignmouth"/>
		</ObjectPropertyRange>
	<Declaration>
		<ObjectProperty IRI="#projetou"/>
	</Declaration>
		<ObjectPropertyDomain>
			<ObjectProperty IRI="#projetou"/>
			<Class IRI="#inventor"/>
		</ObjectPropertyDomain>
		<ObjectPropertyRange>
			<ObjectProperty IRI="#projetou"/>
			<Class IRI="#computador"/>
		</ObjectPropertyRange>
	<Declaration>
		<Class IRI="#invenção"/>
	</Declaration>
	<Declaration>
		<Class IRI="#invenções"/>
	</Declaration>
	<Declaration>
		<Class IRI="#jogos"/>
	</Declaration>
	<Declaration>
		<Class IRI="#juventude"/>
	</Declaration>
	<Declaration>
		<Class IRI="#linguagens"/>
	</Declaration>
	<Declaration>
		<Class IRI="#matemática"/>
```

```
	</Declaration>
	<Declaration>
		<Class IRI="#mecânica"/>
	</Declaration>
	<Declaration>
		<Class IRI="#membro"/>
	</Declaration>
	<Declaration>
		<Class IRI="#mero"/>
	</Declaration>
	<Declaration>
		<Class IRI="#motor"/>
	</Declaration>
	<Declaration>
		<ObjectProperty IRI="#complementou"/>
	</Declaration>
		<ObjectPropertyDomain>
			<ObjectProperty IRI="#complementou"/>
			<Class IRI="#motor"/>
		</ObjectPropertyDomain>
		<ObjectPropertyRange>
			<ObjectProperty IRI="#complementou"/>
			<Class IRI="#conjunto"/>
		</ObjectPropertyRange>
	<Declaration>
		<Class IRI="#máquina"/>
	</Declaration>
	<Declaration>
		<ObjectProperty IRI="#computar"/>
	</Declaration>
		<ObjectPropertyDomain>
			<ObjectProperty IRI="#computar"/>
			<Class IRI="#máquina"/>
		</ObjectPropertyDomain>
		<ObjectPropertyRange>
			<ObjectProperty IRI="#computar"/>
			<Class IRI="#valores"/>
		</ObjectPropertyRange>
	<Declaration>
		<ObjectProperty IRI="#poderia"/>
	</Declaration>
		<ObjectPropertyDomain>
			<ObjectProperty IRI="#poderia"/>
			<Class IRI="#máquina"/>
		</ObjectPropertyDomain>
		<ObjectPropertyRange>
			<ObjectProperty IRI="#poderia"/>
			<Class IRI="#definições"/>
		</ObjectPropertyRange>
	<Declaration>
		<ObjectProperty IRI="#usando"/>
	</Declaration>
		<ObjectPropertyDomain>
			<ObjectProperty IRI="#usando"/>
			<Class IRI="#máquina"/>
		</ObjectPropertyDomain>
		<ObjectPropertyRange>
			<ObjectProperty IRI="#usando"/>
			<Class IRI="#técnicas"/>
		</ObjectPropertyRange>
	<Declaration>
		<ObjectProperty IRI="#é"/>
	</Declaration>
		<ObjectPropertyDomain>
			<ObjectProperty IRI="#é"/>
			<Class IRI="#máquina"/>
		</ObjectPropertyDomain>
		<ObjectPropertyRange>
```

```
                <ObjectProperty IRI="#é"/>
                <Class IRI="#pioneiro"/>
        </ObjectPropertyRange>
    <Declaration>
        <Class IRI="#máquinas"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="#consideram"/>
    </Declaration>
        <ObjectPropertyDomain>
                <ObjectProperty IRI="#consideram"/>
                <Class IRI="#máquinas"/>
        </ObjectPropertyDomain>
        <ObjectPropertyRange>
                <ObjectProperty IRI="#consideram"/>
                <Class IRI="#programa"/>
        </ObjectPropertyRange>
    <Declaration>
        <Class IRI="#método"/>
    </Declaration>
    <Declaration>
        <Class IRI="#múltiplos"/>
    </Declaration>
    <Declaration>
        <Class IRI="#notas"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="#contém"/>
    </Declaration>
        <ObjectPropertyDomain>
                <ObjectProperty IRI="#contém"/>
                <Class IRI="#notas"/>
        </ObjectPropertyDomain>
        <ObjectPropertyRange>
                <ObjectProperty IRI="#contém"/>
                <Class IRI="#algoritmo"/>
        </ObjectPropertyRange>
    <Declaration>
        <Class IRI="#números"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="#incluindo"/>
    </Declaration>
        <ObjectPropertyDomain>
                <ObjectProperty IRI="#incluindo"/>
                <Class IRI="#números"/>
        </ObjectPropertyDomain>
        <ObjectPropertyRange>
                <ObjectProperty IRI="#incluindo"/>
                <Class IRI="#Babbage"/>
        </ObjectPropertyRange>
    <Declaration>
        <Class IRI="#origem"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="#naturalizado"/>
    </Declaration>
        <ObjectPropertyDomain>
                <ObjectProperty IRI="#naturalizado"/>
                <Class IRI="#origem"/>
        </ObjectPropertyDomain>
        <ObjectPropertyRange>
                <ObjectProperty IRI="#naturalizado"/>
                <Class IRI="#estadunidense"/>
        </ObjectPropertyRange>
    <Declaration>
        <Class IRI="#pai"/>
    </Declaration>
```

```
        <Declaration>
               <Class IRI="#papel"/>
        </Declaration>
        <Declaration>
               <Class IRI="#parte"/>
        </Declaration>
        <Declaration>
               <Class IRI="#partes"/>
        </Declaration>
        <Declaration>
               <Class IRI="#período"/>
        </Declaration>
        <Declaration>
               <ObjectProperty IRI="#esteve"/>
        </Declaration>
               <ObjectPropertyDomain>
                      <ObjectProperty IRI="#esteve"/>
                      <Class IRI="#período"/>
               </ObjectPropertyDomain>
               <ObjectPropertyRange>
                      <ObjectProperty IRI="#esteve"/>
                      <Class IRI="#projeto"/>
               </ObjectPropertyRange>
        <Declaration>
               <Class IRI="#pioneiro"/>
        </Declaration>
        <Declaration>
               <Class IRI="#problema"/>
        </Declaration>
        <Declaration>
               <Class IRI="#processamento"/>
        </Declaration>
        <Declaration>
               <Class IRI="#programa"/>
        </Declaration>
        <Declaration>
               <Class IRI="#programadora"/>
        </Declaration>
        <Declaration>
               <Class IRI="#programadores"/>
        </Declaration>
        <Declaration>
               <ObjectProperty IRI="#precisam"/>
        </Declaration>
               <ObjectPropertyDomain>
                      <ObjectProperty IRI="#precisam"/>
                      <Class IRI="#programadores"/>
               </ObjectPropertyDomain>
               <ObjectPropertyRange>
                      <ObjectProperty IRI="#precisam"/>
                      <Class IRI="#abstração"/>
               </ObjectPropertyRange>
        <Declaration>
               <Class IRI="#programas"/>
        </Declaration>
        <Declaration>
               <Class IRI="#programação"/>
        </Declaration>
        <Declaration>
               <Class IRI="#projeto"/>
        </Declaration>
        <Declaration>
               <Class IRI="#quebra"/>
        </Declaration>
        <Declaration>
               <Class IRI="#reações"/>
        </Declaration>
        <Declaration>
```

```
        <Class IRI="#relação"/>
    </Declaration>
    <Declaration>
        <Class IRI="#semáforos"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="#coordenar"/>
    </Declaration>
        <ObjectPropertyDomain>
            <ObjectProperty IRI="#coordenar"/>
            <Class IRI="#semáforos"/>
        </ObjectPropertyDomain>
        <ObjectPropertyRange>
            <ObjectProperty IRI="#coordenar"/>
            <Class IRI="#múltiplos"/>
        </ObjectPropertyRange>
    <Declaration>
        <Class IRI="#seção"/>
    </Declaration>
    <Declaration>
        <Class IRI="#sistema"/>
    </Declaration>
    <Declaration>
        <Class IRI="#sistemas"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="#distribuídos"/>
    </Declaration>
        <ObjectPropertyDomain>
            <ObjectProperty IRI="#distribuídos"/>
            <Class IRI="#sistemas"/>
        </ObjectPropertyDomain>
        <ObjectPropertyRange>
            <ObjectProperty IRI="#distribuídos"/>
            <Class IRI="#forma"/>
        </ObjectPropertyRange>
    <Declaration>
        <Class IRI="#sucesso"/>
    </Declaration>
    <Declaration>
        <Class IRI="#série"/>
    </Declaration>
    <Declaration>
        <Class IRI="#talentos"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="#levaram"/>
    </Declaration>
        <ObjectPropertyDomain>
            <ObjectProperty IRI="#levaram"/>
            <Class IRI="#talentos"/>
        </ObjectPropertyDomain>
        <ObjectPropertyRange>
            <ObjectProperty IRI="#levaram"/>
            <Class IRI="#relação"/>
        </ObjectPropertyRange>
    <Declaration>
        <Class IRI="#tempo"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="#foi"/>
    </Declaration>
        <ObjectPropertyDomain>
            <ObjectProperty IRI="#foi"/>
            <Class IRI="#tempo"/>
        </ObjectPropertyDomain>
        <ObjectPropertyRange>
            <ObjectProperty IRI="#foi"/>
```

```
                    <Class IRI="#chefe"/>
             </ObjectPropertyRange>
      <Declaration>
             <Class IRI="#teoria"/>
      </Declaration>
      <Declaration>
             <Class IRI="#trabalho"/>
      </Declaration>
      <Declaration>
             <ObjectProperty IRI="#é"/>
      </Declaration>
             <ObjectPropertyDomain>
                    <ObjectProperty IRI="#é"/>
                    <Class IRI="#trabalho"/>
             </ObjectPropertyDomain>
             <ObjectPropertyRange>
                    <ObjectProperty IRI="#é"/>
                    <Class IRI="#programadora"/>
             </ObjectPropertyRange>
      <Declaration>
             <Class IRI="#técnicas"/>
      </Declaration>
      <Declaration>
             <Class IRI="#uso"/>
      </Declaration>
      <Declaration>
             <ObjectProperty IRI="#utilizando"/>
      </Declaration>
             <ObjectPropertyDomain>
                    <ObjectProperty IRI="#utilizando"/>
                    <Class IRI="#uso"/>
             </ObjectPropertyDomain>
             <ObjectPropertyRange>
                    <ObjectProperty IRI="#utilizando"/>
                    <Class IRI="#partes"/>
             </ObjectPropertyRange>
      <Declaration>
             <Class IRI="#valores"/>
      </Declaration>
      <Declaration>
             <Class IRI="#visão"/>
      </Declaration>
      <Declaration>
             <Class IRI="#von"/>
      </Declaration>
      <Declaration>
             <Class IRI="#área"/>
      </Declaration>
      <Declaration>
             <Class IRI="#áreas"/>
      </Declaration>
      <Declaration>
             <Class IRI="#época"/>
      </Declaration>
      <Declaration>
             <ObjectProperty IRI="#foi"/>
      </Declaration>
             <ObjectPropertyDomain>
                    <ObjectProperty IRI="#foi"/>
                    <Class IRI="#época"/>
             </ObjectPropertyDomain>
             <ObjectPropertyRange>
                    <ObjectProperty IRI="#foi"/>
                    <Class IRI="#invenção"/>
             </ObjectPropertyRange>
</Ontology>
```

ANEXO A – Lista de *stopwords* removidas pela ferramenta

A lista de *stopwords* utilizada foi retirada de Baségio (2007, p. 112), sendo ilustrada pelo Quadro 31.

Quadro 31 - *Stopwords* removidas pela ferramenta

Classe gramatical	Palavra
Conjunção subordinativa	porque, que, pois, como, porquanto, visto que, visto como, já que, uma vez que, desde que, como, qual, tal qual, tal e qual, assim como, como, tal como, tão, como, tanto como, que, mais que, mais do que, menos que, menos do que, quanto, tanto quanto, que nem, feito, o mesmo que, embora, conquanto, que, ainda que, mesmo que, posto que, por mais que, por menos que, se bem que, nem que, dado que, se caso, contanto que, desde que, salvo se, sem que, a não ser que, a menos que, como, conforme, segundo, consoante, de sorte, que, de modo que, de forma que, de maneira que, sem que, para que, a fim de que, à proporção que, à medida que, ao passo que, quando, enquanto, logo que, mal, sempre que, assim que, desde que, antes que, depois que, até agora, agora que, que, se.
Artigo	o, a, os, as, um, uma, uns, umas.
Pronome pessoal	eu, tu, ele, ela, nós, vós, eles, elas, me, mim, comigo, te, ti, contigo, se, si, lhe, o, a, nos, conosco, vos, lhes, os, as.
Pronome possessivo	meu, minha, meus, minhas, teu, tua, teus, tuas, seu, sua, seus, suas, nosso, nossa, nossos, nossas, vosso, vossa, vossos, vossas.
Pronome demonstrativo	este, estes, esta, estas, esse, esses, essa, essas, aquele, aqueles, aquela, aquelas, mesmo, mesmos, mesma, mesmas, próprio, próprios, própria, próprias, tal, tais, isto, isso, aquilo, o, a, os, as.

Pronome interrogativo	que, quê, quem, qual, quantos, quantas.
Advérbio	sim, deveras, talvez, quiçá, acaso, porventura, decerto, muito, pouco, assaz, bastante, mais, menos, tão, demasiado, meio, todo, demais, que, quão, quanto, quase, como, abaixo, acima, acolá, cá, lá, aqui, ali, aí, além, aquém, algures, alhures, nenhures, atrás, fora, afora, dentro, perto, longe, adiante, diante, onde, avante, através, defronte, aonde, donde, bem, mal, assim, depressa, devagar, debalde, alerta, melhor, pior, não, tampouco, agora, hoje, amanhã, depois, ontem, anteontem, já, sempre, amiúde, nunca, jamais, ainda, logo, antes, cedo, tarde, ora, outrora, então, absolutamente, breve, calmamente, certamente, corretamente, efetivamente, fielmente, levemente, possivelmente, primeiramente, provavelmente, quiçá, realmente, tanto, tarde, ultimamente.
Preposição	a, ante, após, até, com, contra, de, desde, em, entre, para, per, perante, por, sem, sob, sobre, trás, conforme, segundo, durante, mediante, visto, como.
Pronome relativo	cujo, cujos, cuja, cujas, quanto, quantos, quanta, quantas, quem, que, onde.
Locução denotativa	eis, exclusive, menos, exceto, fora, salvo, senão, sequer, inclusive, também, mesmo, ainda, até, ademais, além disso, de mais a mais, só, apenas, é que, sobretudo, embora, aliás, ou, melhor, isto é, ou antes, a saber, por exemplo, ou seja, afinal, agora, então, mas.
Conjunção	e, nem, mas também, mas ainda, senão, também, como também, bem como, mas, porém, todavia, contudo, entretanto, senão, ao passo que, antes, no entanto, não obstante, apesar disso, em todo caso, ou, logo, portanto, por conseguinte, pois, por isso, que, porque, porquanto.
Pronome indefinido	algo, alguém, nada, ninguém, outrem, quem, tudo, cada, certo, certos, certa, certas, algum, alguns, alguma, algumas, bastante,

	demais, mais, menos, muito, muitos, muita, muitas, nenhum, nenhuns, nenhuma, nenhumas, outro, outros, outra, outras, pouco, poucos, pouca, poucas, qualquer, quaisquer, qual, que, quanto, quantos, quanta, quantas, tal, tais, tanto, tantos, tanta, tantas, todo, todos, toda, todas, um, uns, uma, umas, vários, várias.

ANEXO B – Base textual utilizada

No Quadro 32, Quadro 33, Quadro 34, Quadro 35 e Quadro 36 é exibido o conteúdo dos arquivos que compõem a base textual utilizado pelo estudo de caso descrito na seção 3.4.

Quadro 32 - Conteúdo do arquivo Alan Turing

Nome	Conteúdo
Alan Turing	Alan Mathison Turing foi um matemático, lógico, criptoanalista e cientista da computação britânico. Foi influente no desenvolvimento da ciência da computação e na formalização do conceito de algoritmo e computação com a máquina de Turing, desempenhando um papel importante na criação do computador moderno. Ele também é pioneiro na inteligência artificial e na ciência da computação. Durante a Segunda Guerra Mundial, Turing trabalhou para a inteligência britânica em Bletchley Park, num centro especializado em quebra de códigos. P

Quadro 33 - Conteúdo do arquivo Charles Babbage

Nome	Conteúdo
Charles Babbage	Charles Babbage foi um cientista, matemático, filósofo, engenheiro mecânico e inventor inglês nascido em Teignmouth, Devon que originou o conceito de um computador programável. Charles Babbage é mais conhecido e, de certa forma, referenciado como o inventor que projetou o primeiro computador de uso geral, utilizando apenas partes mecânicas, a máquina analítica. Ele é considerado o pioneiro e pai da computação. Seu invento, porém, exigia técnicas bastante avançadas e caras na época, e nunca foi construído. Sua invenção também não era conhecida dos criadores do computador moderno. Mais recentemente, entre 1985 e 1991, o Museu de Ciência de Londres construiu outra de suas invenções inacabadas, a máquina diferencial, usando apenas técnicas disponíveis na época de Babbage.

Quadro 34 - Conteúdo do arquivo Edsger Dijkstra

Nome	Conteúdo
Edsger Dijkstra	Edsger Wybe Dijkstra foi um cientista da computação holandês conhecido por suas contribuições nas áreas de desenvolvimento de algoritmos e programas, de linguagens de programação, sistemas operacionais e processamento distribuído. Entre suas contribuições para a ciência da computação está incluído o algoritmo para o problema do caminho mínimo e a construção de semáforos para coordenar múltiplos processadores e programas. Outro conceito desenvolvido pelo cientista foi a auto-estabilização na área de sistemas distribuídos, uma forma alternativa de garantir a confiança de um sistema. O cientista também foi conhecido por seus ensaios sobre programação, tendo sido o primeiro a alegar que programação é tão inerentemente difícil e complexa que os programadores precisam realizar qualquer abstração possível para gerenciar a complexidade com sucesso.

Quadro 35 - Conteúdo do arquivo John von Neumann

Nome	Conteúdo
John von Neumann	John von Neumann, foi um matemático húngaro de origem judaica, naturalizado estadunidense. Contribuiu na teoria dos conjuntos, análise funcional, teoria ergódica, mecânica quântica, ciência da computação, economia, teoria dos jogos, análise numérica, hidrodinâmica das explosões, estatística e muitas outras as áreas da matemática. Foi membro do Instituto de Estudos Avançados de Princeton, Nova Jérsei, do qual também faziam parte Albert Einstein e Erwin Panofsky, quando emigraram para os Estados Unidos, além de Kurt Gödel, Robert Oppenheimer, George F. Kennan e Hermann Weyl. Com Edward Teller e Stanislaw Ulam, trabalhou em desenvolvimentos chave da Física Nuclear, relacionados com reações termonucleares e com a bomba de hidrogênio. Participou também do Projeto Manhattan, responsável pelo desenvolvimento das primeiras bombas atômicas.

Quadro 36 - Conteúdo do arquivo Ada Lovelace

Nome	Conteúdo
Ada Lovelace	Ada Augusta Byron King, Condessa de Lovelace, atualmente conhecida como Ada Lovelace, foi uma matemática e escritora inglesa e hoje é principalmente reconhecida por ter escrito o primeiro algoritmo para ser processado por uma máquina, a máquina analítica de Charles Babbage. Durante o período em que esteve envolvida com o projeto de Babbage, ela desenvolveu os algoritmos que permitiriam à máquina computar os valores de funções matemáticas, além de publicar uma coleção de notas sobre a máquina analítica. Por esse trabalho é considerada a primeira programadora de toda a história. Na juventude seus talentos matemáticos levaram-na a uma relação de trabalho e de amizade com o colega matemático britânico Charles Babbage e, em particular, o trabalho de Babbage sobre a Máquina Analítica. Entre 1842 e 1843, ela traduziu um artigo do engenheiro militar italiano Luigi Menabrea sobre o motor, e complementou com um conjunto de sua própria autoria, que ela chamou de Anotações. Essas notas, contém um algoritmo criado para ser processado por máquinas, o que muitos consideram ser o primeiro programa de computador. Ela também desenvolveu uma visão sobre a capacidade dos computadores de irem além do mero cálculo ou processamento de números, enquanto outros, incluindo o próprio Babbage, focavam apenas nessas capacidades.