Luiz Carlos da Cruz Carvalheira

Método Semi-Automático de Construção de Ontologias Parciais de Domínio com Base em Textos

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia.

São Paulo
2007

Luiz Carlos da Cruz Carvalheira

Método Semi-Automático de Construção de Ontologias Parciais de Domínio com Base em Textos

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia.

Área de concentração:
Sistemas Digitais

Orientador:
Prof. Dr. Edson Satoshi Gomi

São Paulo
2007

Ficha Catalográfica

Carvalheira, Luiz Carlos da Cruz
Método Semi-Automático de Construção de Ontologias Parciais de Domínio com Base em Textos. São Paulo, 2007. 143 p.

Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1. Gestão do Conhecimento. 2. Ontologias. 3. Processamento de Linguagem Natural. 4. Inteligência Artificial. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais. II. Título.

À Mariana, Laura e Mônica. Minhas flores, meus amores...

À memória de meu pai.

Agradecimentos

Agradeço ao caríssimo orientador, Prof. Dr. Edson Satoshi Gomi, pela busca permanente de excelência nas nossas colaborações e pelo contínuo estímulo, a um só tempo criativo e crítico. Sua seriedade e respeito foram fatores fundamentais para a elaboração e conclusão deste trabalho, nesta que foi uma jornada de dedicação e prazer aprendendo e descobrindo.

Agradeço aos professores Dr. Ricardo Luis de Azevedo da Rocha da Escola Politécnica e Dr. Marcelo Finger do Instituto de Matemática e Estatística da USP pelo interesse e pelas sugestões no meu exame de qualificação.

Obrigado aos queridos amigos do Laboratório de Engenharia de Conhecimento, o KNOMA, sempre dispostos a compartilhar as agruras e as alegrias da pesquisa científica acadêmica.

Agradeço aos meus gerentes na IBM, Gustavo Cameira, Fernando Nimer, Rodrigo Stanger e Roberto Mathias que ao longo destes anos permitiram que eu me dedicasse à realização deste trabalho.

Obrigado também aos meus colegas e mentores da IBM, em particular ao Dr. Paulo W. Costa, pelo estímulo e apoio durante esta e tantas outras iniciativas de inovação.

Sobretudo, agradeço à minha família pela compreensão, paciência e incentivo incansáveis.

Resumo

Os recentes desenvolvimentos relacionados à gestão do conhecimento, à web semântica e à troca de informações eletrônicas por meio de agentes têm suscitado a necessidade de ontologias para descrever de modo formal conceituações compartilhadas à respeito dos mais variados domínios. Para que computadores e pessoas possam trabalhar em cooperação é necessário que as informações por eles utilizadas tenham significados bem definidos e compartilhados. Ontologias são instrumentos viabilizadores dessa cooperação. Entretanto, a construção de ontologias envolve um processo complexo e longo de aquisição de conhecimento, o que tem dificultado a utilização desse tipo de solução em mais larga escala.

Este trabalho apresenta um método de criação semi-automática de ontologias a partir do uso de textos de um domínio qualquer para a extração dos conceitos e relações presentes nesses textos. Baseando-se na comparação da freqüência relativa dos termos extraídos com os escritos típicos da língua e na extração de padrões lingüísticos específicos, este método identifica termos candidatos a conceitos e relações existentes entre eles, apresenta-os a um ontologista para validação e, ao final, disponibiliza a ontologia ratificada para publicação e uso especificando-a na linguagem OWL.

Abstract

The recent developments related to knowledge management, the semantic web and the exchange of electronic information through the use of agents have increased the need for ontologies to describe, in a formal way, shared understanding of a given domain. For computers and people to work in cooperation it is necessary that information have well defined and shared definitions. Ontologies are enablers of that cooperation. However, ontology construction remains a very complex and costly process, which has hindered its use in a wider scale.

This work presents a method for the semi-automatic construction of ontologies using texts of any domain for the extraction of concepts and relations. By comparing the relative frequency of terms in the text with their expected use and extracting specific linguistic patterns, the method identifies concepts and relations and specifies the corresponding ontology using OWL for further use by other applications.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

API Application Programming Interface

CML Conceptual Modelling Language

CREOLE Collection of Reusable Objects for Language Engineering

DAML DARPA Agent Markup Language

DARPA US Defense Advance Research Project Agency

GATE General Architecture for Text Engineering

HTML Hyper Text Markup Language

JAPE Java Annotation Patterns Engine

KIF Knowledge Interchange Format

NLP Natural Language Processing

NIST United States National Institute of Standards and Technology

OIL Ontology Inference Layer

OWL Web Ontology Language

PLN Processamento de Linguagem Natural

PMBOK Project Management Body of Knowledge

PMI Project Management Institute

POS Part of Speech

PSG Phrase Structure Grammar

RDF Resource Description Framework

RDFS Resource Description Framework Schema

SGML Standard Generalized Markup Language

TF.IDF Term Frequency – Inverse Document Frequency

UIMA Unstructured Information Management Architecture

URI Universal Resource Identifier

XML Extensible Markup Language

1 Introdução

Os recentes desenvolvimentos relacionados à gestão do conhecimento, à web semântica e à troca de informações eletrônicas têm suscitado a necessidade de ontologias para descrever de modo formal conceituações compartilhadas à respeito dos mais variados domínios. Para que computadores e pessoas possam trabalhar em cooperação é necessário que as informações por eles utilizadas tenham significados bem definidos e comuns. Nesse contexto, as ontologias são instrumentos viabilizadores dessa cooperação.

Segundo Studer, Benjamins e Fensel (1998), uma ontologia é uma especificação formal e explicita de uma conceituação compartilhada. Por conceituação entende-se um modelo abstrato de algum fenômeno ou fato do mundo e cuja representação se torna explícita se todos os conceitos e restrições para seu uso forem claramente definidos. Essa conceituação torna-se compartilhada quando todas as partes envolvidas no seu uso concordam e aceitam tal abstração. O formalismo na definição dessa especificação permite a redução na ambigüidade da comunicação entre seres humanos e viabiliza a sua utilização por máquinas.

A construção de ontologias envolve, entretanto, um processo de aquisição de conhecimento bastante complexo e longo, o que vem dificultando a implementação desse tipo de solução. Em resposta a essa dificuldade, a comunidade científica vem explorando novas técnicas e abordagens que possam reduzir esse esforço, viabilizando assim a construção de ontologias de domínio.

1.1 Objetivos

Este trabalho tem como objetivo criar um método semi-automático de construção de ontologias parciais de domínio, baseado na análise de textos que expressam os principais conceitos desse domínio de conhecimento.

Esse método deve ser capaz de:

1. Processar os documentos selecionados com o intuito de formar um corpus básico de análise;
2. Identificar os conceitos mais importantes do domínio sobre o qual versa o texto;
3. Identificar as relações mais relevantes que possam existir entre esses conceitos;
4. Representar esses conceitos e relações em uma linguagem formal.

Ainda que este método necessite da intervenção de um ontologista durante a elaboração da ontologia, espera-se que o apoio oferecido pelo método facilite e acelere significativamente essa construção.

É ainda um objetivo deste trabalho fornecer um arcabouço que possa servir de base para aumentar o conhecimento e experiência nesse tema pelo laboratório do KNOMA na Escola Politécnica da Universidade de São Paulo e que permita a elaboração de trabalhos cada vez mais avançados na área.

1.2 Metodologia

Para alcançar os objetivos descritos na seção anterior foram realizadas cinco etapas.

A primeira etapa foi composta de uma pesquisa bibliográfica sobre os trabalhos recentes a respeito da definição e uso de ontologias, bem como da sua construção. Foram pesquisadas abordagens e métodos baseados, tanto em mecanismos automáticos ou semi-automáticos, quanto exclusivamente na capacidade de especialistas humanos. Na seqüência essas abordagens foram analisadas e a sua aplicabilidade em relação à proposta deste trabalho foi avaliada.

A segunda etapa, envolveu a análise das técnicas mais apropriadas para a implementação das abordagens escolhidas para tratamento dos vários subproblemas envolvidos neste trabalho. Nesta etapa foram também testadas, avaliadas e definidas algumas ferramentas de apoio ao desenvolvimento do método estabelecido no trabalho, tais como frameworks de desenvolvimento de aplicativos que envolvam processamento de linguagem natural e linguagens de representação de ontologias.

Na terceira etapa foi estabelecido um domínio para o qual uma ontologia foi construída segundo o método proposto no trabalho. Na seqüência definiu-

se o conjunto de textos que serviram de base para a confecção da ontologia. Nesta etapa foram também avaliadas possíveis abordagens para validação dos resultados obtidos.

Na quarta etapa foi constituido um ambiente de processamento lingüístico onde o método pudesse ser implementado e onde as diversas técnicas utilizadas pudessem ser constituídas, configuradas e analisadas.

Na quinta e última etapa os resultados obtidos foram avaliados de acordo com a abordagem estabelecida e analisados segunda a perspectiva deste e de potenciais trabalhos futuros sobre o tema.

1.3 Organização do Texto

Além desta introdução, este trabalho é composto por outros cinco capítulos. O próximos dois capítulos oferecem a fundamentação teórica na qual este trabalho foi baseado: o capítulo 2 apresenta o conceito de ontologia, possíveis usos e aplicações, além de métodos e ambientes para sua construção; o capítulo 3 apresenta o desafio do aprendizado de ontologias, algumas iniciativas recentes de estabelecimento de métodos de construção semi-automáticos de ontologia e, fechando o capítulo, as principais técnicas relacionadas ao processamento de linguagens naturais, base para desenvolvimento deste trabalho.

Os capítulos seguintes apresentam o método proposto e discute os resultados da sua implementação: o capítulo 4 descreve o método e as técnicas usadas no trabalho; o capítulo 5 apresenta os resultados obtidos durante a implementação do método. Finalmente, o capítulo 6 apresenta as considerações finais e as possíveis direções para trabalhos futuros na área.

2 Ontologias

Utilizando a perspectiva da Inteligência Artificial, este capítulo procura esclarecer questões como: O que são as ontologias? Como caracterizá-las? Quais são suas aplicações? Como construí-las? Para isso, são apresentadas as definições de diversos pesquisadores da área, métodos de construção, linguagens de definição e ambientes de desenvolvimento.

2.1 Definição

Originalmente o conceito de ontologia foi objeto de estudo de Platão e Aristóteles na antiga Grécia, quando tratavam da natureza do Ser, isto é, da realidade, da existência dos entes. Composta pelos termos gregos onto ("ser" no sentido de "existir") + logia ("conhecimento"), a palavra "ontologia" é associada, no contexto da filosofia ao exame da natureza da realidade, incluindo a relação entre mente e matéria, substância e atributo, fato e valor. Nesse sentido, ela procura descrever as categorias básicas e os relacionamentos do ser ou existir, estudando e analisando concepções da realidade.

Ainda nessa perspectiva da filosofia, a ontologia pode ser vista como a teoria dos objetos e de suas conexões. Seu estudo fornece critérios para a distinção dos vários objetos, concretos ou abstratos, existentes ou não, reais ou ideais, e suas conexões, relações, dependências e asserções.

Com uma longa história em filosofia, o termo ontologia foi recentemente incorporado ao vocabulário da Inteligência Artificial, da Engenharia de Conhecimento e da Representação do Conhecimento, entre outras áreas da Computação. Nas últimas duas décadas as pesquisas de Inteligência Artificial vêm se servindo de ontologias no processo de descrição formal das coisas do mundo, algo fundamental para que sistemas inteligentes possam atuar e ponderar sobre o mundo em que se propõem operar (WELTY; GUARINO, 2001).

Gruber (1995) define uma ontologia como sendo uma especificação explí-

cita de uma conceituação. Por conceituação entendem-se os objetos, os conceitos e outras entidades que se assume existirem em alguma área de interesse, bem como as relações que se verificam entre eles (GENESERETH, 1978). Em outras palavras, uma conceituação é uma visão abstrata e simplificada do mundo que se deseja representar para algum propósito. Assim, qualquer base de conhecimento, sistema baseado em conhecimento ou agente inteligente está comprometido com (*is committed to*) alguma conceituação, explícita ou implicitamente. Para sistemas baseados em conhecimento, o que "existe" é exatamente aquilo que consegue ser representado (GRUBER, 1995).

Na definição de Uschold e Gruninger (1996), uma ontologia limita e incorpora algum tipo de visão de mundo relacionado a um determinado domínio. Essa visão de mundo é comumente concebida como um conjunto de conceitos (ex.: entidades, atributos ou processos) e seus inter-relacionamentos. No mesmo trabalho, os autores mencionam a seguinte definição para o que vem a ser uma ontologia, suas formas e o contexto em que ela se insere:

> "Ontologias são acordos a respeito de conceituações compartilhadas. Conceituações compartilhadas incluem estruturas conceituais para a modelagem do conhecimento de um domínio; protocolos com conteúdo específico para a comunicação entre agentes interoperantes; e acordos a respeito das representação de teorias de um domínio particular. No contexto de compartilhamento de conhecimento, ontologias são especificadas na forma de definições de vocabulário representacional. Um caso muito simples seria uma hierarquia de tipos especificando classes e suas relações de subsunção. Esquemas de base de dados relacionais também servem como ontologias ao especificar as relações que podem existir entre bases de dados compartilhadas e as restrições de integridade que devem ser respeitadas entre elas."

Segundo Studer, Benjamins e Fensel (1998), uma ontologia é uma especificação explícita e formal de uma conceituação compartilhada. Ilustrando as restrições dessa definição:

- Por especificação explícita, entende-se a definição de conceitos, instâncias, relações, restrições e axiomas;
- Por formal, entende-se uma definição declarativa, utilizável por agentes e sistemas;

- Por conceituação, entende-se um modelo abstrato de uma área de conhecimento ou de um universo limitado de discurso;
- Por compartilhada, estabelece-se um conhecimento consensual, seja ele uma terminologia comum do domínio modelado, ou acordado entre os desenvolvedores dos agentes que inter-operam.

2.2 Categorias de Ontologia

Uma ontologia deve incluir necessariamente um vocabulário de termos e alguma definição para seus significados, mas pode variar bastante quanto ao formalismo dessas definições. Em relação a esse formalismo, Uschold e Gruninger (1996) definiram uma escala de avaliação de ontologias composta de quatro níveis de formalização:

- altamente informal: expressa em linguagem natural desestruturada;
- semi-informal: expressa em linguagem natural estruturada e restrita, reduzindo a ambigüidade e aumentando a clareza dos conceitos;
- semi-formal: expressa em linguagem artificial formalmente definida;
- rigorosamente formal: expressa com termos meticulosamente definidos, através de semântica formal, teoremas e provas de completude (*completeness*) e correção (*soundness*).

Em relação ao conteúdo e à natureza e abrangência dos conceitos definidos, ontologias podem ser classificadas nos seguintes tipos (adaptado de Guarino e Giaretta (1995)):

- Ontologias de representação: definem as conceituações que fundamentam os formalismos de representação de conhecimento, definindo as primitivas de representação;
- Ontologias genéricas: trazem definições gerais a respeito de conceitos abstratos, tais como, tempo, espaço, matéria, seres, coisas, etc., independentes de um problema ou domínio particular, mas necessárias para a compreensão de aspectos do mundo;

- Ontologias centrais (*core ontologies*) ou genéricas de domínio: descrevem os ramos de estudo de uma área. Um exemplo é a ontologia central de direito (VALENTE, 1995) que inclui conhecimentos normativos, de responsabilidade, comportamentos permitidos, etc. O objetivo das ontologias centrais é servir de base para a construção de ontologias de ramos mais específicos do domínio (por exemplo, ontologias de direito tributário, de família, etc);

- Ontologias de domínio: expressam conceituações de domínios particulares de uma área genérica de conhecimento (por exemplo, cardiologia);

- Ontologias de aplicação: descrevem conceitos dependentes do domínio e das tarefas que lidam com um problema específico desse domínio como, por exemplo, identificar doenças do coração;

- Ontologias de tarefas: expressam conceituações e descrevem o vocabulário relacionado às atividades e tarefas usadas na resolução de problemas (por exemplo, processos, planos, etc.);

Como se percebe, existe uma relação de generalidade entre as categorias de ontologia. As ontologias de nível mais genérico, são geralmente reutilizadas na construção de ontologias mais específicas.

2.3 Usos e Aplicações

Uschold e Gruninger (1996) consideram os usos para ontologias como sendo de três tipos básicos: comunicação, inter-operabilidade e engenharia de sistemas.

2.3.1 Comunicação

Ontologias podem prover uma estrutura unificada que reduz a confusão de uma organização a respeito de conceitos e terminologia. Assim, as ontologias podem ser usadas para promover a comunicação e o compartilhamento de conhecimento entre pessoas com diferentes pontos de vista e necessidades por estarem inseridas em diferentes contextos dentro da organização.

Uma ontologia pode ser usada para descrever uma rede de relacionamentos entre entidades, para registrar o que se conecta com o que, e para explorar

e navegar nessa rede de conexões. Ainda que esse tipo de rede possa ser implícita no sistema, as pessoas em geral têm perspectivas distintas e usam suposições diferentes, o que pode criar obstáculos ao entendimento das relações chaves dentro do sistema. Em aplicações que envolvem domínios diferentes, as ontologias são particularmente importantes já que explicitam as conexões lógicas entre os elementos dos vários modelos do sistema.

As pessoas que implementam e utilizam um sistema de software integrado de larga escala devem manter uma compreensão comum sobre o funcionamento e os objetivos desse sistema. Uma das maneiras de se prover essa compreensão é através da construção e uso de uma ontologia para estabelecer um modelo normativo desse sistema, isto é, um consenso por parte dos agentes envolvidos sobre os objetos e comportamentos relacionados a ele. Essa ontologia pode ser expandida e refinada, e eventualmente viabilizar a inferência e o raciocínio sobre os impactos de eventuais mudanças no sistema.

Em sistemas onde vários agentes se comunicam é imprescindível que a integração entre eles aconteça com base em uma interpretação compartilhada a respeito das informações trocadas. Por outro lado, recursos em diferentes áreas de uma companhia podem ter perspectivas diferentes a respeito do que ela faz, quais seus objetivos e como eles devem ser alcançados. Uma ontologia pode fornecer um modelo normativo da organização, apoiando a integração e a comunicação entre os vários participantes e fornecendo uma terminologia padronizada sobre todos os objetos e relações que existam nos seus domínios.

2.3.2 Inter-operabilidade

Outra aplicação importante para as ontologias é a de fornecer ambientes de integração em que diferentes ferramentas de software possam ser utilizadas em conjunto, em arquiteturas multi-agentes e modelos corporativos que envolvam atividades, produtos, serviços e organizações. Ontologias podem, nesses casos, funcionar como uma língua de inter-operação, permitindo a padronização de terminologia entre os diferentes usuários desses recursos e diminuindo a quantidade de conversões ou traduções necessárias entre cada par de componentes para que eles possam se comunicar. Essa idéia foi explorada com sucesso, por exemplo, no projeto da PSL - Process Specification Language - (SCHLENOFF et al., 1999) pela NIST [1] americana, onde uma ontologia atua como

[1] A PSL foi desenvolvida na Manufacturing Systems Integration Division do National Institute of Standards and Technology americano para ser uma representação "neutra" de dados dos processos de manufatura.

um tradutor entre duas aplicações (por exemplo, um processo de manufatura e um processo de monitoramento), eliminando as ambigüidades nas definições de termos usados por cada um dos sistemas. Assim, o termo "procedimento" usado por um sistema é traduzido para o termo "método" usado por um segundo sistema com a ajuda de uma ontologia, onde esse mesmo conceito é chamado de "processo".

Ainda ligado ao conceito de inter-operabilidade, é possível utilizar ontologias que integrem outras ontologias de domínios diversos. Por exemplo, uma ontologia que apóie um sistema de *workflow* precisa integrar ontologias de processos, recursos, produtos, serviços e mesmo organizações.

2.3.3 Engenharia de Sistemas

Além de suportar a operação de sistemas, ontologias podem também apoiar o desenho e o desenvolvimento de sistemas de software propriamente ditos. Ontologias facilitam o processo de identificação dos requisitos de um sistema e de entendimento das relações entre os seus vários componentes. De maneira mais formal, uma ontologia pode prover uma especificação declarativa do sistema, o que permite que se infira sobre o que o sistema foi projetado para fazer, ao invés de como ele implementa uma dada funcionalidade. A metodologia CommonKads (SCHREIBER et al., 1994), por exemplo, utiliza a linguagem CML (Conceptual Modelling Language) para construir ontologias de domínio e de tarefas para apoiar a especificação de sistemas baseados em conhecimento.

2.3.4 Web Semântica

A Web Semântica (BERNERS-LEE; HENDLER; LASSILA, 2001) é um projeto sob a direção do consórcio World Wide Web e que objetiva estabelecer uma infra-estrutura de troca de informações baseada no acréscimo de significado (isto é, "semântica") ao conteúdo dos documentos da Web. Para tanto, esse projeto estende a Web através do uso de padrões, de linguagens de marcação (*markup languages*) e de um ferramental de processamento conveniente. Tal como ela se apresenta atualmente, a Web é desenhada para ser utilizada e lida por pessoas, não por máquinas. A Web Semântica tenta justamente expandir o modo com que as páginas da Web são feitas de tal forma que computadores possam compreendê-las.

A Web é baseada fundamentalmente em documentos escritos em HTML

(HyperText Markup Language), uma linguagem utilizada para definir e representar textos e objetos como imagens e formulários e que foca na apresentação visual desses elementos. Ainda que o HTML permita uma definição bastante precisa do *layout* das informações nas páginas, essa linguagem não possui capacidade de expressar, por exemplo, que um determinado item da página de uma livraria é, em realidade, um livro cujo título é "Godel, Escher, Bach" e cujo preço é "R$ 63,20". Nessa página, o HTML usado consegue apenas indicar que uma cadeia de caracteres "Godel, Escher, Bach" deve ser mostrada ao lado de outra cadeia de caracteres "R$ 63,20". Em outras palavras, o HTML usado na Web não fornece a capacidade de exprimir que a página é de um catálogo, "Godel, Escher, Bach" é o título de um livro e que "R$ 63,20" é um preço, o preço daquela obra.

A Web Semântica tenta endereçar esse problema de representação com o uso combinado de componentes tecnológicos e linguagens tais como RDF (Resource Description Framework) (LASSILA; SWICK, 1999), OWL (Web Ontology Language) (SMITH; WELTY; MCGUINNESS, 2004) e XML (Extensible Markup Language) (BRAY; PAOLI; SPERBERG-MCQUEEN, 1998) para viabilizar o fornecimento de descrições que suplementam o conteúdo dos documentos da Web e expressas de forma que as máquinas possam lê-las. Este último requisito é o que permite que os autores dos documentos e dos sites possam associar significado ao conteúdo de suas páginas, facilitando a busca automática de informações e a pesquisa por computadores.

Dentre os usos concebidos para a Web Semântica podem-se destacar:

Agentes. A Web semântica pode capacitar agentes inteligentes a entender e integrar informações que um usuário procure a respeito de, por exemplo, entretenimento ou viagens. Para planejar uma atividade desse tipo, além de conhecer os interesses do usuário, esses agentes devem ter acesso a um ambiente rico de informações para que possam buscar e avaliar opções. Nesse contexto, uma ontologia que represente os principais conceitos ligados a restaurantes, hotéis e outros tipos de serviços pode permitir aos agentes que infiram a melhor combinação de opções, dadas as preferências e restrições do usuário. Assim, um agente, ao qual um usuário que gosta de frutos do mar tenha dado a tarefa de programar uma viagem rápida a Copacabana com pernoite em hotel, poderá considerar como interessantes opções para o usuário a estadia em um flat no Leme e um eventual jantar no Restaurante Shirley da Rua Gustavo Sampaio.

Computação ubíqua. A computação ubíqua ou *pervasive* é caracterizada pela migração do foco do usuário do aspecto físico do computador para o foco na capacidade computacional que ele necessita, esteja ela onde estiver. Idealmente, essa capacidade computacional deve perseguir o usuário de modo que o meio através do qual ele tem acesso a ela é muito menos relevante do que o uso que o utilizador faz da capacidade. Para isso, essa capacidade computacional embarcada (*embedded*) deve ser capaz de identificar o ambiente em que o usuário está operando e de se auto-configurar, descobrindo serviços e recursos disponíveis ao seu redor e fornecendo-os ao usuário. A inter-operabilidade de um dispositivo, algo que permite que ele funcione harmoniosamente com outros mesmo sob condições diversas e sem nunca ter sido explicitamente programado para tratar cada possível situação, depende da capacidade desse dispositivo de entender as características do ambiente ao seu redor. Uma ontologia pode ser usada para descrever as características desse ambiente e de outras máquinas, definir como acessar os serviços disponíveis, as políticas de uso dos recursos, etc., viabilizando assim a inter-operação desses dispositivos.

Portais Web. As ontologias podem ser usadas em portais da Web para prover terminologias específicas do domínio de interesse do portal e para descrever seu conteúdo. Como exemplo, em um portal de pesquisa científica, uma ontologia pode incluir definições como "uma revista é uma publicação", "um jornal é uma publicação", "todo autor de uma publicação é uma pessoa", etc. Essas definições podem então ser usadas em conjunto com fatos conhecidos para se inferir outros fatos. O conjunto desses fatos permitiria que buscas no portal pudessem resultar em informações que, usando formas mais tradicionais de busca, seriam mais difíceis de se conseguir.

2.3.5 Benefícios de uma ontologia

Como visto nas seções anteriores, a utilização de ontologias pode ser bastante vantajosa. Dentre os principais benefícios da sua utilização, destacam-se:

- A comunicação não ambígua de informações entre recursos. Uma ontologia permite o compartilhamento entre pessoas e entre agentes de software de um entendimento comum sobre uma estrutura de informação;

- O aumento na confiabilidade de sistemas. Ontologias podem ser usadas para aumentar a confiabilidade de sistemas através de verificações de consistência em relação às especificações declarativas do sistema e também explicitando as suposições nas quais se baseiam os vários componentes do sistema.

- A reutilização de conhecimento a respeito de um domínio. Através da caracterização de classes de domínio e de tarefas desses domínios, ontologias fornecem uma base para que se determine quais elementos são reutilizáveis em diferentes domínios e tarefas.

- A separação entre o conhecimento do domínio e o conhecimento operacional. Através da representação de conhecimento relacionado a uma determinada área pode-se separar os mecanismos usados para entender e avaliar uma situação ou um problema dos mecanismos utilizados para, efetivamente, corrigir ou atuar sobre ele;

- A capacidade de análise do conhecimento do domínio. As ontologias são também instrumentos para a organização daquilo que se sabe a respeito de um assunto e, como tal, pode capacitar a avaliação de conceitos e relacionamentos na medida em que eles evoluam no tempo ou em que mais usos se façam para seus conceitos;

2.4 Métodos e Critérios de Construção

Em seu trabalho de compilação, Corcho, Fernández-López e Gómez-Pérez (2003) relacionam algumas metodologias que foram propostas para a construção de ontologias. A seguir destaca-se a de Uschold e Gruninger (1996), assim como as recomendações básicas de Guarino e Giaretta (1995).

Segundo Uschold e Gruninger (1996), a construção de ontologias deve ser dividida em cinco processos: três de desenvolvimento e dois de suporte executados simultaneamente aos três anteriores. A seguir são sumarizados esses processos:

1. *Identificação do objetivo e do escopo da ontologia*: Neste processo estabelecem-se os motivos da construção da ontologia e as aplicações que se pretendem para ela. Define-se o escopo da ontologia através de "questões de competência", perguntas em linguagem natural que descrevem os requisitos que devem ser atendidos pela ontologia. Em outras palavras,

a ontologia deve criar componentes (como conceitos e relacionamentos) que permitam representar essas perguntas e suas respostas;

2. *Construção da ontologia*: Este processo é composto pelas atividades básicas de elaboração da ontologia:

   (a) A captura da ontologia, isto é, a identificação dos conceitos chaves e dos relacionamentos que existam no domínio, a descrição desses conceitos e relacionamentos e a escolha dos termos que identificarão cada um deles;

   (b) A codificação da ontologia, ou seja, a representação através de uma linguagem formal da conceituação identificada na atividade anterior;

   (c) A integração da ontologia com outras já existentes através da reutilização de conceitos.

3. *Avaliação da ontologia*: Nesta atividade utilizam-se as "questões de competência", agora expressas em linguagem formal, para verificar a consistência e a expressividade da ontologia criada;

4. *Definição de guias*: Aqui definem-se as técnicas e métodos que serão usadas como base para todas as demais fases da construção, as entradas e os produtos de cada fase, bem como a ordem em que elas devem ser executadas;

5. *Documentação da ontologia*: Nesta atividade definem-se as regras para que a construção seja documentada de modo apropriado, isto é, garantindo que as premissas utilizadas durante a construção sejam registradas.

Guarino estabelece alguns critérios que devem ser considerados na montagem de uma ontologia como forma de avaliação da sua consistência, especialmente importantes na fase de captura dos elementos da ontologia:

Particionamento: O que deve ser considerado como parte de uma entidade, isto é, algo de interesse que pode ser real ou conceitual e que se deseja estudar e descrever? Quais seriam as propriedades das partes? Quais os tipos de partes existentes?

Integridade: O que pode ser considerado como um inteiro, ou em outras palavras, algo que não pode ser dividido sem que se perca sua característica de ser algo completo? Em que sentido suas partes estão conectadas?

Identidade: Quais são as propriedades essenciais que definem uma entidade dando-lhe uma identidade? Como a entidade pode mudar mantendo a identidade? Em que circunstâncias a entidade perde a identidade? Como uma mudança de partes ou ponto de vista altera a identidade?

Dependência: Uma entidade pode existir sozinha? Depende de outras entidades?

2.5 Linguagens de Representação de Ontologias

Esta seção apresenta os elementos básicos que uma linguagem de representação de ontologias é concebida para representar. Em seguida, relacionam-se as linguagens que vêm sendo usadas para implementação de ontologias, destacando particularmente a linguagem OWL (Web Ontology Language), a escolhida para representar as ontologias criadas neste trabalho.

2.5.1 Elementos de uma Ontologia

Diferentes linguagens de ontologia apresentam diferentes capacidades de representação dos componentes de uma ontologia. A seguir são descritos os elementos e características que as linguagens de ontologia representam, com maior ou menor capacidade de expressão.

Conceitos

Um conceito é uma unidade básica de pensamento a respeito de um grupo de coisas ou idéias que tenham características comuns. Conceitos podem ser abstratos (como honestidade, reputação) ou concretos (como um livro, carros), fictícios ou reais, elementares ou compostos. Um conceito pode ser a descrição de uma tarefa, atividade, ação, estratégia, etc. Diferentes linguagens de representação de ontologias usam termos variados para nomear os conceitos: objetos, categorias, classes (tal como usado em RDF, OIL, DAML+OIL, OWL).

Um conceito pode ter atributos, que em algumas linguagens são chamados de funções, em outras de *slots* e, mais comumente, de propriedades. Esses atributos podem ser classificados em:

- *Atributos de classe*: são atributos cujo valor é ligado ao conceito e, portanto, esse valor é o mesmo para todas as instâncias da classe;

- *Atributos de instância*: são atributos que podem assumir valores diferentes para cada instância do conceito;
- *Atributos locais*: atributos que, apesar de possuírem o mesmo nome, pertencem à diferentes classes. Por exemplo, os conceitos *projeto* e *atividade* podem ter ambos um atributo chamado *data de início*, local a cada conceito;
- *Atributos globais*: são atributos que não tem um domínio especificado, isto é, se aplicam a qualquer conceito da ontologia.

Algumas linguagens de representação também permitem a especificação de outros elementos de um atributo, tais como

- *Valores default*: usado para definir valores *default* para casos em que não se defina o valor de um atributo explicitamente. Por exemplo, desconto no preço de um produto é 0 se não for mencionado;
- *Tipo*: usado para restringir o tipo do atributo. Por exemplo, o número de cilindros de um motor deve ser inteiro;
- *Cardinalidade*: restringe o número mínimo e máximo de valores para o atributo. Por exemplo, um produto pode ter de 1 a 5 cores, no máximo;
- *Documentação*: usado para manter uma definição em linguagem natural para o atributo.

Taxonomias

Taxonomias são usadas para organizar os conceitos de uma ontologia com o uso de relações de especialidade e generalidade, implementando o conceito de herança. Tipicamente as linguagens de implementação de ontologias fornecem primitivas para a definição de elementos tais como:

- *Subclasse*: especializa conceitos genéricos criando conceitos mais específicos;
- *Decomposição disjunta*: partição onde todos os conceitos são subclasses de um conceito comum e onde nenhum membro de uma subclasse é membro de outra subclasse;

- *Decomposição exaustiva de subclasse*: uma decomposição disjunta completa, isto é, todos os membros da classe ou conceito comum são membros de alguma de suas subclasses;

- *Não é subclasse de*: especifica que um conceito não é uma especialização de um outro conceito. Usado em algumas linguagens para representar a negação da primitiva de *subclasse*.

Relações

Relações representam interações entre os conceitos de um domínio. Algumas linguagens permitem a definição de relações com vários argumentos, como por exemplo uma relação de *compra* em que uma *pessoa*, compra um *produto* em uma *loja* em uma *data*. As linguagens podem também oferecer a capacidade de definir restrições de tipo para cada argumento e mesmo restrições de integridade para verificar a validade dos valores de cada argumento.

Axiomas

Os axiomas são sentenças aceitas como verdadeiras sem necessidade de prova. Em uma ontologia os axiomas são usados com diferentes objetivos mas sobretudo para verificar a validade de uma informação ou para restringí-la. Tipicamente, nas linguagens que os suportam, os axiomas são construídos com base em lógica de primeira-ordem.

Instâncias

A maior parte das linguagens fornece capacidade para definir instâncias e fatos em uma ontologia. Instâncias representam elementos do domínio de um conceito específico, enquanto fatos são relações que existem entre esses elementos. Em OWL, instâncias são chamadas de *indivíduos (individuals)*.

2.5.2 KIF

KIF - *Knowledge Interchange Format* - foi uma das primeiras linguagens criadas para representar ontologias. Idealizada por Genesereth e Fikes (1992), KIF é baseada no cálculo de predicados e possui alto poder de expressividade. Foi inicialmente desenhada para possibilitar a troca de conhecimento entre sistemas computacionais distintos e tem as seguintes características fundamentais:

- Possui semântica declarativa;
- Possui capacidade de expressar qualquer sentença lógica em cálculo de predicados de primeira ordem;
- Provê capacidade de representação de meta-conhecimento, ou seja, conhecimento sobre o conhecimento, o que permite que o usuário introduza novas construções de representação de conhecimento usando a própria linguagem.

2.5.3 RDF e RDFS

A *Resource Description Framework* - RDF - foi desenvolvido pelo W3C (*World Wide Web Consortium*) para permitir a descrição de recursos na Web e prover mecanismos para representação explícita de serviços, processos e modelos de negócio. Em outras palavras, o RDF (KLYNE; CARROLL, 2004) provê capacidade para descrever o metadados de documentos (por exemplo, autor, título, etc) e de outros recursos da Web.

O modelo de dados da RDF é composto por três tipos de objetos: i) recursos, ii) propriedades, usadas para definir características, atributos ou relações que descrevem um recurso e iii) sentenças (*statements*) usadas para atribuir valores para uma propriedade de um recurso específico.

A RDF não fornece mecanismos para descrever propriedades nem para descrever as relações entre essas propriedades e outros recursos. Essa é a função da RDFS, ou esquema RDF, que define classes e propriedades que podem ser usadas para descrever classes, propriedades e outros recursos, específicos de uma aplicação (como por exemplo os metadados que descrevem um filme ou um livro).

A RDF e RDFS são baseadas em XML (*Exchange Markup Language*). A linguagem XML (BRAY; PAOLI; SPERBERG-MCQUEEN, 1998) fornece uma sintaxe padrão para definição de linguagens de marcação, sendo portanto uma metalinguagem. Através dessas linguagens de marcação, pode-se definir estrutura, restrições e semântica de dados e documentos.

2.5.4 OIL

OIL - *Ontology Inference Layer* - é uma linguagem para representação de ontologias e uma camada de inferência. Fensel et al. (2000) criaram a OIL desenhan-

do-a em torno do XML e do RDFS, de uso comum na Web. A OIL combina as primitivas de modelagem das linguagens baseadas em *frames* com a semântica formal e os serviços de inferência da lógica descritiva. *Frames* são uma noção introduzida por Minsky (1975) para representação de conhecimento de um domínio. Um *frame* é uma estrutura para representar um conceito ou situação e que pode conter informações que descrevam ou definam o conceito. O objetivo dos *frames* é capturar a essência desses conceitos e situações através do agrupamento de todas as informações relevantes a essas situações.

A lógica descritiva é um sub-conjunto da lógica de primeira ordem que mantém um alto poder de expressividade, além de um mecanismo eficiente de inferência.

A OIL possui quatro camadas: i) *Core* Oil, que agrupa as primitivas que tem mapeamento direto com as primitivas RDF, ii) *Standard* Oil, o modelo completo OIL, iii) *Instance* Oil, usada para adicionar instâncias de conceitos e papéis aos modelos anteriores, e iv) *Heavy* Oil, reservada para futuras expansões.

2.5.5 DAML+OIL

DAML+OIL - *DARPA Agent Markup Language + Ontology Inference Layer* - é a linguagem criada por Horrocks (2002) baseada na integração da OIL com a DAML-ONT, definida pela DARPA para expressar definições de classes mais sofisticadas que as permitidas pelo RDFS. A DARPA, ou *Defense Advanced Research Projects Agency*, é a agência do Departamento de Defesa Americano responsável pelo desenvolvimento de novas tecnologias para uso militar.

2.5.6 OWL

Para representação das ontologias a serem criadas pelo método proposto neste trabalho, foi escolhida a linguagem OWL. A seguir serão apresentadas algumas das suas características e as razões que fundamentaram esta escolha.

A linguagem OWL (SMITH; WELTY; MCGUINNESS, 2004) é uma proposta do W3C para representação de ontologias e que foi concebida para viabilizar a Web Semântica e facilitar às máquinas o processamento e a integração de informações disponíveis na Web. Como parte dos requisitos básicos que orientaram essa concepção, a OWL tinha que contemplar as camadas de linguagens da Web Semântica, incluindo XML e RDF.

Baseada na DAML+OIL, a OWL estende o vocabulário de descrição de classes e propriedades da RDF, provendo assim funcionalidades como características de propriedades, relações entre classes e cardinalidade de relacionamento.

A escolha da linguagem OWL para uso neste trabalho foi baseada em algumas características que a distinguem das outras linguagens aqui relacionadas.

Primeiramente, a OWL foi uma linguagem concebida para o uso na Web. Enquanto as linguagens mencionadas anteriormente foram criadas para usos focados em comunidades de usuários e domínios específicos, a OWL foi definida para ser compatível com a arquitetura da World Wide Web e da Web Semântica. Através do uso de URIs (*Universal Resource Identifiers*) e RDF como base para nomeação e identificação de recursos, a OWL permite a distribuição de uma ontologia através de vários sistemas (fornecendo a escalabilidade necessária para uso na Web).

Outra consideração importante nesta escolha, é o equilíbrio entre o poder de expressividade da linguagem e a eficiência que ela oferece no suporte à inferência e raciocínio. De modo geral, quanto mais expressiva for uma linguagem, mais ineficiente é o suporte à inferência (tendendo, no limite, a não computabilidade).

No quesito de expressividade, a OWL parte da capacidade de declarar fatos do RDF e da capacidade de estruturar classes e propriedades do RDFS e as estende. Em OWL pode-se declarar classes e organizá-las em hierarquias de subsunções, isto é, subclasses, como no RDFS. No entanto, as classes OWL podem ser especificadas como combinações lógicas (intersecção, união e complemento) de outras classes ou enumerações de objetos específicos. Como em RDFS, a OWL permite declarar propriedades, organizá-las em hierarquias de sub-propriedades e especificar domínios e imagens (*ranges*) para essas propriedades. Indo além, em OWL os domínios das propriedades são classes e as imagens (*ranges*) podem ser ou classes ou tipos de dados definidos (tais como *string* ou *integer*). Além disso, a OWL pode exprimir quais objetos (também chamados de indivíduos) pertencem a quais classes e quais valores de propriedade podem ser de indivíduos específicos. Finalmente, a OWL permite que declarações de equivalência possam ser feitas para classes ou para propriedades, classes possam ser declaradas disjuntas e indivíduos possam ser declarados iguais ou desiguais.

Como exemplo de comparação entre a RDFS e a OWL, imaginemos que

com a RDFS é possível:

- declarar classes como `País`, `Pessoa`, `Estudante` e `Brasileiro`;
- afirmar que `Estudante` é uma subclasse de `Pessoa`;
- afirmar que `Brasil` e `Portugal` são instâncias da classe `País`;
- declarar `Nacionalidade` como uma propriedade que relaciona as classes `Pessoa` (o domínio) e `País` (a imagem ou *range*);
- afirmar que `idade` é uma propriedade de tipo de dado (*datatype*) onde o domínio é `Pessoa` é a imagem (*range*) é um número inteiro;
- afirmar que `João` é uma instância da classe `Brasileiro` e que sua `idade` tem valor 35.

Adicionalmente a essas declarações, com a OWL se poderia ainda:

- afirmar que `Brasil` e `Portugal` são indivíduos disjuntos;
- afirmar que `País` e `Pessoa` são classes disjuntas;
- declarar `TemCidadão` como sendo a propriedade inversa de `Nacionalidade`;
- afirmar que a classe `Brasileiro` é definida como sendo o conjunto de membros da classe `Pessoa` que tem `Brasil` como um dos valores da propriedade `Nacionalidade`;
- afirmar que a classe `Multinacional` é definida pelo conjunto de membros da classe `Pessoa` que possuem pelo menos 2 valores na propriedade `Nacionalidade`;
- afirmar que a classe `Apátrida` é definida pelo conjunto de membros da classe `Pessoa` que não possuem nenhum valor para a propriedade `Nacionalidade`;
- afirmar que `idade` é uma propriedade funcional.

Na OWL, o equilíbrio de expressividade e eficiência é ainda reforçado através da oferta de três sub-linguagens com capacidades distintas, como veremos a seguir.

2.5.6.1 As Três Espécies de OWL

A OWL fornece três sub-linguagens, cada uma com poder de expressividade diferente e progressivamente maior:

- OWL *Lite*: desenhada para suportar a representação de ontologias mais simples, compostas por uma hierarquia de classificação e restrições menos complexas. Ainda que ela suporte restrições de cardinalidade, por exemplo, só são permitidas cardinalidades 0 ou 1;
- OWL *DL*: com maior poder de expressividade que a OWL *Lite*, a OWL *DL* foi concebida para apoiar usuários que precisam de grande expressividade mas desejam manter a completude computacional (todas as conclusões são computáveis) e a decidibilidade (todas os cômputos terminam em um tempo finito). Baseada em lógica descritiva, a OWL *DL* inclui todas as construções da linguagem OWL mas com algumas restrições (como, por exemplo, uma classe não pode ser uma instância de uma outra classe);
- OWL *Full*: a sub-linguagem de maior poder de expressividade, a OWL *Full* provê a liberdade sintática do RDF, embora não ofereça garantias computacionais, fato que limita o seu uso por mecanismos de raciocínio automáticos (*reasoners*). Como exemplo disso, uma classe no OWL *Full* pode ser tratada simultaneamente como uma coleção de indivíduos e como um indivíduo separado.

Cada uma dessas sub-linguagens é uma extensão da sua antecessora mais simples, tanto do ponto de vista do que pode ser expresso, quanto do que pode ser concluído ou inferido. Assim, temos que: toda ontologia OWL *Lite* legal é também uma ontologia OWL *DL* legal; toda ontologia OWL *DL* legal é também uma ontologia OWL *Full* legal; toda conclusão válida em OWL *Lite* é também uma conclusão válida em OWL *DL*; toda conclusão válida em OWL *DL* é também uma conclusão válida em OWL *Full*;

2.5.6.2 Classes em OWL

Em OWL, conceitos são chamados de classes, instâncias são chamadas de indivíduos e as relações entre indivíduos são chamadas de propriedades. Todas as classes em OWL são definidas como subclasses da classe *owl:Thing*. Os seguintes construtores são oferecidos para definição de classes em OWL *Lite*:

- `rdfs:subClassOf`: usado para criar hierarquias de classes especificando que uma classe é subclasse de outra;
- `owl:equivalentClass`: utilizado para especificar que duas classes são equivalentes (isto é, possuem os mesmos indivíduos ou instâncias) ou são classes sinônimas;
- `owl:intersectionOf`: usado para especificar intersecções de classes e restrições. Por exemplo, `PessoaEmpregada` pode ser especificada como a intersecção entre `Pessoas` e `CoisasEmpregadas`.

Além desses construtores, OWL *DL* e OWL *Full* também provém axiomas de classes para a construção de classes:

- `owl:oneOf`: usado para descrever a classe com a enumeração de todos os indivíduos que dela fazem parte. Por exemplo, a classe `diasDaSemana` pode ser descrita com a simples enumeração dos indivíduos `Segunda, Terça, Quarta, Quinta, Sexta, Sábado` e `Domingo`;
- `owl:disjointWith`: utilizado para especificar que classes podem ser descritas como sendo disjuntas, isto é, não podem ter indivíduos comuns, por exemplo as classes `Homem` e `Mulher`;
- `owl:unionOf` e `owl:complementOf`: usado para permitir combinações de classes baseadas em operadores de união e complemento dos indivíduos das classes. Por exemplo, a classe de cidadãos da união européia pode ser descrita como a união dos cidadãos de todos os estados membros. Em um exemplo de complemento, pode-se dizer que Crianças são uma subclasse do complemento de PessoasIdosas.

Para exemplificar a definição de uma classe em OWL, a especificação a seguir declara a classe Mulher:

```
<owl:Class rdf:ID="Mulher">
    <owl:equivalentClass>
        <owl:Class rdf:ID="Pessoa_Sexo_Feminino"/>
    </owl:equivalentClass>
    <rdfs:subClassOf>
        <owl:Class rdf:ID="Pessoa"/>
    </rdfs:subClassOf>
```

```
    <owl:disjointWith>
        <owl:Class rdf:ID="Homem"/>
    </owl:disjointWith>
</owl:Class>
```

2.5.6.3 Propriedades em OWL

Em OWL propriedades são usadas para especificar o relacionamento entre dois indivíduos e podem ser classificadas nas seguintes categorias principais:

- Propriedade de tipo de dado (`datatype properties`): são propriedades estabelecidas entre instâncias de classes e o formato de dados. Esse formato pode ser de literais RDF ou como tipos baseados em esquemas XML. Entre os formatos recomendados para uso em OWL tem-se, por exemplo, strings (`xsd:string`), inteiros (`xsd:integer`), inteiros não negativos (`xsd:nonNegativeInteger`), URIs (`xsd:anyURI`);
- Propriedade de objeto (`object properties`): são propriedades que relacionam instâncias de classes, como por exemplo `temFilho` que pode relacionar dois indivíduos da classe `Pessoa`;

Além desses, os seguintes construtores podem ser usados na definição de propriedades:

- `rdfs:subPropertyOf`: usado para especificar hierarquias de propriedades (como, por exemplo, para especificar que `terIrmão` é uma subpropriedade de `terParente`);
- `rdfs:domain`: utilizado para restringir para quais indivíduos a propriedade pode ser aplicada;
- `rdfs:range`: usado para definir o conjunto de valores válidos para uma propriedade. Em outras palavras, se uma propriedade relaciona um indivíduo com outro indivíduo e a propriedade tem uma classe como a sua imagem (`range`) então o outro indivíduo tem que pertencer a essa classe;
- `rdfs:equivalentProperty`: usado para especificar que duas propriedades são equivalentes;

- `owl:inverseOf`: utilizado para declarar que uma propriedade é o inverso de outra, ou seja, se a propriedade P1 é a inversa de P2 e um individuo X é relacionado ao individuo Y pela propriedade P1, então Y é relacionado a X por P2;

- `owl:transitiveProperty`: usado para definir que uma propriedade é transitiva, isto é, se os pares (x,y) e (y,z) são ambos instâncias da propriedade transitiva P, então o par (x,z) também é uma instância de P. A propriedade `éUmAncestralDe` é um exemplo de uma propriedade transitiva;

- `owl:symmetricProperty`: define que uma propriedade é simétrica, isto é, se o par (x,y) é uma instância de uma propriedade simétrica P, então o par (y,x) também é uma instância de P. Por exemplo, a propriedade `éAmigoDe` é simétrica;

- `owl:functionalProperty`: define que a propriedade tem cardinalidades mínima zero e máxima 1. Um exemplo desse tipo de propriedade é `temTimePreferido`, onde nem todos teriam um time preferido, mas ninguém pode ter mais que um;

- `owl:inverseFunctionalProperty`: declara que a propriedade é inversamente funcional ou, de outra forma, que o inverso da propriedade é funcional. Por exemplo,

  `temUmNúmeroDeRG` é inversamente funcional, já que seu inverso, digamos, `éONúmeroDeRGDe` só tem no máximo um valor para cada indivíduo da classe de números de RG.

Exemplificando o uso da propriedade de objeto na linguagem OWL, se quiséssemos definir uma propriedade `temTio` que é sub-propriedade de `temFamiliar` e que possui a propriedade `temSobrinho` como propriedade inversa e a classe `Pessoa` como imagem, poderíamos utilizar a seguinte especificação:

```
<owl:ObjectProperty rdf:about="#temTio">
    <rdfs:subPropertyOf>
        <owl:ObjectProperty rdf:ID="temFamiliar"/>
    </rdfs:subPropertyOf>
    <owl:inverseOf rdf:resource="#temSobrinho"/>
    <rdfs:range rdf:resource="#Pessoa"/>
</owl:ObjectProperty>
```

2.5.6.4 Restrições de Propriedade em OWL

Através das `Property Restrictions` OWL permite a limitação de propriedades como forma de descrever uma classe, isto é, impondo restrições que devem ser obedecidas por todos os indivíduos que dela fazem parte. Essas restrições podem ser de valor (`value constraint`) ou de cardinalidade (`cardinality constraint`).

Os construtores a seguir podem ser usados na especificação desse tipo de restrição de propriedade:

- `owl:allValuesFrom`: utilizado para definir que a propriedade nessa classe em especial tem uma restrição local de imagem (`range`) de valores. Em outras palavras, se uma instância de uma classe está relacionada pela propriedade a um segundo indivíduo, então pode se inferir que essa segunda instância faz parte da classe de restrição local de valores. Por exemplo, a classe Pessoa apresenta a propriedade `temFilha` restringida a ter todos os seus valores (`allValuesFrom`) como sendo da classe Mulher;

- `owl:someValuesFrom`: usado para declarar que uma classe tem uma propriedade com uma restrição onde pelo menos um valor dessa propriedade é de um certo tipo. Por exemplo, a classe ArtigoSobreOntologia pode ter uma restrição na propriedade `temPalavraChave` de ter algum valor (`someValuesFrom`) dessa propriedade como sendo uma instância da classe `TópicoOntologia`.

Exemplificando o uso da restrição universal na linguagem OWL, temos abaixo a especificação da classe Gerente, especificada como sendo uma subclasse da classe anônima e que contem apenas indivíduos da classe Empregado e da classe Pessoa:

```
<owl:Class rdf:ID="Gerente">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty>
                <owl:ObjectProperty rdf:ID="temGerenciado"/>
            </owl:onProperty>
            <owl:\textbf{allValuesFrom}>
                <owl:Class rdf:ID="Empregado"/>
```

```
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Pessoa"/>
</owl:Class>
```

A restrição universal especifica que se o relacionamento `temGerenciado` existir então ele tem de ser com indivíduos da classe `Empregado`. Ao contrário do que pode parecer, essa definição não obriga a existência do relacionamento sobre a propriedade `temGerenciado` para todos os indivíduos da classe `Gerente`.

Abaixo, é exemplificado o uso da restrição existencial em uma situação similar a anterior mas onde se deseja especificar que um gerente tem pelos um de seus gerenciados como sendo um empregado (e que pode ter outros gerenciados que são sub-contratados, digamos):

```
<owl:Class rdf:ID="Gerente">
    <rdfs:subClassOf rdf:resource="#Pessoa"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:\textbf{someValuesFrom}
                rdf:resource="#Empregado"/>
            <owl:onProperty>
                <owl:ObjectProperty rdf:ID="temGerenciado"/>
            </owl:onProperty>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Deve-se notar que nesta definição não se espera que todos os indivíduos da classe `Gerente` tenham apenas gerenciados da classe `Empregado` mas sim que ao menos um dos gerenciados seja um indivíduo dessa classe.

Similarmente, OWL disponibiliza também a capacidade de declarar restrições de cardinalidade de uma propriedade sobre as instâncias de uma classe em particular, isto é, locais à classe. Os construtores a seguir são usadas para esse tipo de declaração:

- `owl:minCardinality`: utilizado para definir a quantidade mínima de relacionamentos que uma instância pode ter através de uma propriedade

em particular. Se, por exemplo, a `minCardinality` for 1, espera-se que qualquer instância da classe se relacione com no mínimo um indivíduo através dessa propriedade. A classe Pai deveria ter uma cardinalidade mínima de 1 na propriedade `temFilho`;

- `owl:maxCardinality`: usado para especificar o número máximo de relacionamentos que um indivíduo pode ter através de uma dada propriedade.

- `owl:cardinality`: empregado para definir o número exato de relacionamentos de uma dada propriedade que uma instância pode ter.

Notar que, diferentemente do OWL *DL* e OWL *Full*, as restrições de cardinalidade em OWL *Lite* são limitadas aos valores 0 e 1.

2.5.7 A Linguagem Escolhida para este Trabalho

Como mencionado durante a elucidação de suas características, a linguagem OWL será utilizada neste trabalho para a representação das ontologias criadas.

Dentre as principais razões para isso, destacam-se:

- o crescente interesse que a linguagem OWL tem suscitado na comunidade;

- a disponibilidade de um conjunto de construtores que permitirão a definição dos elementos que serão criados pelo método sugerido;

- a oferta de diferentes níveis de expressividade, desde um nível mais simples necessário aqui, até níveis mais complexos de construtores e de restrições, sofisticados o suficiente para permitir uma eventual evolução deste trabalho no futuro.

3 Abordagens para Criação de Ontologias

A construção e a manutenção de ontologias são atividades que requerem muito esforço e tempo. Além de cara, a modelagem feita à mão por um especialista humano pode apresentar erros e ser influenciada em demasia pela sua experiência própria, isto é, pode não representar a interpretação predominante entre a maioria dos especialistas do domínio analisado. Dadas essas dificuldades, é ainda bastante expressiva a quantidade de domínios para os quais não existem ontologias modeladas e reconhecidas pelos respectivos especialistas como adequadas para a representação do conhecimento da área. Esses fatores constituem inibidores bastante poderosos para uma utilização mais massiva e ubíqüa de ontologias.

Para endereçar esses problemas o campo da aprendizagem de ontologias surgiu para propor o desenvolvimento de técnicas e abordagens automáticas de construção de ontologias. Maedche e Staab (2004) sugerem uma classificação dessas abordagens segundo a base utilizada na construção da ontologia e que pode ser composta por dados estruturados (como aqueles que se mantém em bases de dados), semi-estruturados (como aqueles expressos em HTML e XML) ou textos em linguagem natural.

- **Criação de ontologias a partir de dados estruturados** objetiva a criação de ontologias baseada na extração de conceitos e relações contidos em bases de dados. Esse tipo de ontologia tem sido usado como uma forma de mediação ou integração entre bases de dados, mas a criação manual dessas ontologias é bastante complexa e tediosa;

- **Criação de ontologias a partir de dados semi-estruturados** consiste na utilização de novos padrões para a publicação de documentos na web, tais como HTML, XML, XML-Schemas. Esses novos padrões tem alterado a maneira como se disponibiliza informações na web e aumentado a disponibilidade de dados semi-estruturados, bem como as definições

formais para esses dados, o que incorpora algum nível de expressividade semântica aos documentos;

- **Criação de ontologias a partir de fontes não estruturadas** envolve a construção de ontologias a partir de textos utilizando-se técnicas de processamento de linguagem natural. Esses textos apresentam vários níveis de informação que são representadas através de características e restrições sintáticas, morfológicas, semânticas e pragmáticas, atributos que convergem para expressar significados particulares. Ferramentas que aprendem ontologias a partir de linguagem natural utilizam a interação entre essas características e restrições para identificar conceitos e estabelecer relações entre eles.

A seguir são discutidas as principais abordagens que têm sido estudadas para a construção semi-automática de ontologias de domínio. Todas elas são relacionadas ao processamento de linguagem natural embora difiram nas técnicas utilizadas e na forma de manipulação dos textos, o que permite uma distinção em abordagens baseadas em métodos lingüísticos, abordagens baseadas em métodos estatísticos e, finalmente, abordagens baseadas em métodos de aprendizado de máquina.

3.1 Abordagens baseadas em Técnicas Lingüísticas

As abordagens ligadas a técnicas lingüísticas baseiam-se tipicamente em análises sintáticas e morfológicas dos textos. Essas análises identificam elementos de linguagem que são eventualmente utilizados para a extração de padrões lingüísticos, para o cálculo de medidas de proximidade ou relacionamento semântico, por exemplo. Outra característica importante explorada por essas abordagens é a posição que os vários elementos ocupam nos padrões identificados e que outros elementos estão próximos a eles no texto.

Aussenac-Gilles, Biebow e Szulman (2000) estabeleceram um método de análise de um corpus através do uso de técnicas de Processamento de Linguagem Natural (NLP, na sigla em inglês) para criação de uma ontologia. Esse método prevê também o uso de ontologias já existentes, bem como outros recursos terminológicos, para essa construção. Em linhas gerais o método pode ser decomposto nos passos abaixo.

- **Constituição de um corpus**. Textos relacionados aos requisitos da on-

tologia a ser construída são selecionados para formar o corpus. Tipicamente, especialistas da área formam um glossário dos termos do domínio e selecionam textos que contenham esses termos para formação do corpus.

- **Análise Lingüística**. Nesta atividade extraem-se do corpus os termos do domínio, as relações léxicas existentes entre eles e possíveis sinônimos. Os termos são escolhidos baseados na freqüência de ocorrência no texto, enquanto a extração de relações é apoiada por alguns padrões lingüísticos pré-estabelecidos. As ferramentas e técnicas utilizadas são específicas da língua sendo tratada.

- **Normalização**. O ontologista, isto é, o especialista que controla a criação da ontologia, seleciona então as relações léxicas (hiperônimo[1], hipônimo[2], etc.) que deseja incluir na ontologia. Ele também adiciona definições em linguagem natural para os termos selecionados.

Alfonseca e Manandhar (2002b, 2002a) estabeleceram uma abordagem para estender uma ontologia já existente e se basearam na identificação de propriedades contextuais de palavras que co-ocorrem com cada grupo de conceitos. Essas propriedades são utilizadas para agrupar conceitos em uma ontologia ou para adicionar conceitos novos a uma ontologia já existente. A hipótese básica utilizada na abordagem é uma intitulada "Semântica Distributiva" e que afirma que o significado de uma palavra é altamente correlacionada com o contexto em que ela aparece. Cada conceito é representado por um conjunto de palavras que com ele co-ocorrem e pela freqüência com que elas aparecem. Esse contexto é codificado usando-se vetores de palavras, tal como feito por Lin e Hovy (2000) em sua assinatura de assunto (*topic signatures*). Algumas métricas de similaridade, tais como o TF.IDF (SALTON, 1991) são utilizadas para mensurar as distâncias entre os diversos conceitos. Uma das características desse método é que ele necessita de textos em que várias ocorrências dos conceitos ocorram para que exista suficiente informação contextual para construção dos vetores.

[1]Hiperônimo é o nome que se dá ao termo cujo sentido inclui aquele (ou aqueles) de um ou de vários outros termos. Ex. "Animal" é um hiperônimo de "cavalo", "baleia", etc.

[2]Hipônimo é uma palavra cujo significado é hierarquicamente mais específico que o de uma outra. Por exemplo, "abóbora" ou "tomate" estão em relação de hiponímia relativamente a "legume".

3.2 Abordagens baseadas em Técnicas Estatísticas

Abordagens baseadas em técnicas estatísticas se utilizam de métricas estatísticas para auxiliar na identificação de conceitos e relações. Tipicamente, as técnicas usadas calculam freqüências de repetição de palavras ou padrões de palavras e TF.IDF como medidas (*scores*) indicadoras da relevância dos elementos identificados. É muito comum que estas técnicas sejam complementadas ou complementem outras técnicas baseadas em lingüística.

Agirre et al. (2000) estipularam um método para enriquecimento dos conceitos de uma ontologia ao mesmo tempo em que controlam a proliferação de significados, algo que pode ser bastante comum em ontologias mais extensas, tais como o WordNet (FELLBAUM, 1989). Essa abordagem baseia-se em documentos que existam na web e pode ser sumarizada nos seguintes passos:

- Coletar na web documentos relevantes relacionados a cada um dos conceitos da ontologia. Utiliza-se a informação existente na ontologia na construção de consultas para cada conceito e cada significado, sejam sinônimos, hiperônimos, atributos, etc. Descartam-se documentos que se relacionam a mais de um significado e agrupam-se aqueles que se referem ao mesmo conceito, uma coleção para cada conceito/significado;

- Construir assinaturas de assunto (*topic signatures*). Processam-se os documentos em cada coleção extraindo-se as palavras e suas freqüências e calculando-se estatísticas que indiquem as palavras mais relacionadas ao conceito. Comparam-se então as palavras extraídas dessa coleção com as das outras coleções para se determinar aquelas que possuam uma freqüência distinta. Essa lista de palavras é usada então na construção das assinaturas de assunto (LIN; HOVY, 2000) de cada conceito;

- Agrupar significados das palavras. Comparam-se as diferentes assinaturas de assunto à procura de palavras compartilhadas para assim determinar a sobreposição entre as assinaturas. Utilizam-se nesta tarefa várias métricas de proximidade semântica e técnicas de agrupamento (*clustering*);

- Avaliação. O usuário verifica os resultados obtidos e os compara com um algoritmo de tratamento de desambigüação de sentido (*word sense disambiguation*).

Faatz e Steinmetz (2002) elaboraram uma abordagem de enriquecimento de uma ontologia já existente onde também se servem de documentos da web para construírem e compararem estatísticas de uso de palavras de um corpus com a estrutura da ontologia. Usando as palavras e frases associadas a cada conceito, eles calculam métricas de similaridade. Esse método pode ser sumarizado nas seguintes fases:

- Constituição do corpus. Buscam-se na web documentos que se relacionam à ontologia estudada;
- Detecção de candidatos a conceitos. Estatísticas são calculadas no corpus e uma lista de co-ocorrências (termos que ocorrem em conjunto) é gerada para cada palavra no corpus. Novas palavras são extraídas baseadas em função de distância semântica e consideradas então como candidatas a novos conceitos;
- Seleção de conceitos. A lista criada é proposta para um especialista do domínio que então decide a relevância dos novos conceitos para o domínio e se eles devem ou não ser incorporados na ontologia.

3.3 Abordagens baseadas em Aprendizado de Máquina

Pesquisadores da área de aquisição de conhecimento têm explorado possíveis conexões entre métodos tradicionais de aquisição de conhecimento e técnicas de aprendizado de máquina. Tipicamente essas técnicas são usadas para complementar abordagens baseadas em técnicas lingüísticas.

Hwang (1999) estabeleceu uma abordagem para tratamento e pesquisa de informações contidas em grandes bases de dados de texto que se baseia na organização de uma ontologia em taxonomias mais simples, a partir das quais conceitos são identificados e procurados nos documentos da base de dados. O método é composto basicamente das seguintes tarefas:

- Especialistas no domínio fornecem algumas palavras que servirão de sementes para representar os conceitos. Coletam-se da web documentos com consultas (*queries*) construídas com base nessas palavras;
- Os documentos resultantes das pesquisas são tratados e deles se extraem as frases que contenham as palavras sementes. Estabelecem-se conceitos

que são então inseridos na ontologia e oferecidos ao especialista humano como novos conceitos. Baseados nessa iteração, novos candidatos para palavras-sementes são então escolhidos automaticamente e novas iterações de busca de textos e descoberta de conceitos são feitas. Todos os documentos e as linhas de texto que serviram à construção de conceitos são mantidos indexados e disponíveis, juntamente com algumas métricas de freqüência;

- Executa-se a extração de relações. Através do uso de características (*features*) lingüísticas, relações do tipo 'is-a', 'part-of', 'owned-by', 'produced-by', etc. são extraídas. Utiliza-se então a relação 'assoc-with' (de *associated with*) para todas as relações descobertas que não são do tipo 'is-a';

- Ao final de cada iteração o especialista humano é consultado sobre a adequação dos conceitos descobertos.

Kietz, Volz e Maedche (2000) estabeleceram uma abordagem genérica para o refinamento de ontologias de domínio já existentes a partir de fontes heterogêneas em um processo semi-automático onde os algoritmos de aprendizado e o especialista do domínio cooperam. A premissa básica da abordagem é que os documentos escolhidos como base do processamento contém os conceitos a serem incluídos na ontologia. O método é sumarizado a seguir:

- Seleção das fontes de informação. A primeira tarefa consiste na seleção de uma ontologia genérica (também chamada de *top-level*) que deverá conter os conceitos genéricos e os conceitos de domínio que já tiverem sido elencados;

- Em seguida o usuário deve especificar os documentos que servirão de base ao processo de extensão e refinamento da ontologia fornecida. Apesar de parecer trivial, a decisão de quais documentos utilizar é a que mais influência tem sobre a qualidade do resultado final;

- Identificação de conceitos. Nesta atividade busca-se descobrir novos conceitos baseado na análise da freqüência dos termos, usando-se a hipótese de que os termos que são mais freqüentes no corpus específico do domínio do que em corpus genéricos devem ser propostos ao especialista como candidatos a novos conceitos;

- Poda de domínio. Nesta atividade a ontologia criada é podada para remoção de conceitos demasiadamente genéricos de maneira que o resultado seja focado no domínio estudado;

- Aprendizagem de relações. Utiliza-se uma análise de freqüências para identificação de relações existentes entre os conceitos do domínio, usando a hipótese de que a co-ocorrência freqüente de pares de conceitos pode expressar relações relevantes para a ontologia. Para essa identificação é utilizado o algoritmo de associação proposto por Agrawal e Srikant (1995);

- Avaliação. Nesta etapa o usuário especialista avalia a ontologia resultante e decide se uma nova iteração é necessária.

4 Processamento de Linguagem Natural

Este trabalho propõe predominantemente a utilização de processamentos lingüísticos nos textos como base para a construção semi-automática de uma ontologia para um domínio qualquer. Assim, são apresentados a seguir alguns dos tópicos de lingüística computacional e algumas das técnicas utilizadas para processamento de linguagem natural.

4.1 Áreas de Estudo das Linguagens Naturais

A seguir são apresentados diferentes áreas de estudo da linguagem: morfologia, sintaxe, semântica e pragmática.

4.1.1 Morfologia

A morfologia estuda a estrutura e a constituição das palavras que formam a expressão de uma língua. Por exemplo, palavras como *livro**s*** ou ***des**considerar* podem ser divididas em unidades menores, quando se identificam sufixos e prefixos. A essas unidades que compõem as palavras dá-se o nome de morfema.

A morfologia, além de analisar essas estruturas, também estuda a classificação das palavras em diferentes categorias, como substantivos, adjetivos, verbos, advérbios e preposições. Esse tipo de classificação é conhecida também como *part-of-speech* ou *POS*, por objetivar justamente classificar as palavras em partes do discurso.

Essas categorias de palavras são comumente associadas a processos morfológicos, tais como a formação do plural a partir da forma singular de um dado substantivo, ou a inflexão de um verbo na terceira pessoa do singular no presente. Inflexões são modificações sistemáticas da forma raiz de uma pala-

vra com a adição de prefixos, sufixos ou outras marcas. São esses processos morfológicos que nos permitem identificar palavras novas, que nunca havíamos visto antes. Os processos morfológicos são particularmente importantes para a PLN em línguas onde a inflexão é muita usada. Em Russo, Finlandês ou Latim, por exemplo, um verbo pode assumir uma grande quantidade de formas distintas. Todas as formas de inflexão de uma palavra são comumente agrupadas como manifestações de um mesmo lexema.

Além da inflexão, outros processos morfológicos comuns são a derivação e a composição. Através da derivação consegue-se uma mudança na categoria sintática e, em geral, uma mudança semântica. Um exemplo é a derivação do advérbio *rapidamente* a partir do adjetivo *rápido*. Já a composição acontece quando agrupamos duas ou mais palavras para formar outra. *Caminhão-pipa* é um exemplo desse processo.

A classificação morfológica é interessante na medida em que as palavras de uma mesma categoria tendem a contribuir de modo semelhante para o significado da frase. Substantivos (ou *nomes*), por exemplo, são usualmente associados a conceitos ou objetos, enquanto verbos demonstram ações feitas ou sofridas por tais objetos, relações entre esses conceitos. É importante notar que essas palavras podem também ser usadas como base de um grupo para formar os chamados sintagmas.

Esse tipo de análise é bastante importante e básica em um sistema que processe linguagem natural já que vários outros tratamentos subseqüentes dependem desse conhecimento a respeito das categorias das palavras.

4.1.2 Sintaxe

As línguas tem restrições a respeito da ordem em que as palavras aparecem nas sentenças. As palavras são organizadas em frases, conjuntos de palavras que funcionam juntas para expressar idéias. O estudo das regularidades e restrições na ordem das palavras e na estrutura da frase é chamada de sintaxe.

Essa estrutura sintática é normalmente regida por um conjunto finito de regras e princípios que possibilitam que escritores e leitores possam produzir e reconhecer sentenças que nunca foram criadas antes: a gramática.

Em português, tanto como em inglês, uma sentença normalmente se escreve como um sintagma nominal seguido de um sintagma verbal. Sintagmas nominais são unidades sintáticas nas quais se demonstram informações sobre

um nome (ou seja, um substantivo). Sintagmas nominais são normalmente os argumentos de verbos, os participantes de ações ou estados descritos pelos verbos. Ex.: "*O homem que eu ajudei ontem a subir no ônibus* ficou muito agradecido". Sintagmas verbais são unidades encabeçadas pelo verbo, que em geral organiza sintaticamente os elementos da sentença. Por exemplo: Ele *estava tentando manter a calma*.

Substantivos, verbos intransitivos, transitivos diretos, indiretos, orações subordinadas e complementos, entre outros, são partes integrantes da análise sintática a que se submetem as sentenças do texto em estudo. Esse tipo de avaliação é direcionado a situações em que estruturas são sintaticamente bem formadas, ou seja, são "gramaticais". A frase abaixo é um exemplo de uma sentença não gramatical.

"Crianças dormem as."

É importante, porém, distinguir sentenças não gramaticais de sentenças anormais. A famosa frase a seguir (em uma tradução livre do original inglês) é um exemplo de uma sentença gramatical que não faz sentido e não permite uma interpretação coerente.

"idéias verdes sem cor dormem furiosamente."

A análise sintática colabora, assim, também na interpretação da frase. O exemplo a seguir é clássico no estudo de ambigüidade em lingüística e demonstra como a estrutura da frase e a interpretação que damos a ela estão conectadas:

"O homem viu o menino com o telescópio."

A ambigüidade estrutural da frase permite uma primeira interpretação em que o telescópio caracteriza a ação de ver (*viu com o telescópio*) mas também uma segunda, onde o telescópio caracteriza o menino visto (*o menino com o telescópio*).

É importante notar que, da mesma forma que a análise sintática depende dos resultados da análise morfológica, subseqüentes análises de interpretação do texto dependem, por sua vez, dos resultados da análise sintática.

4.1.3 Semântica e Pragmática

A semântica está ligada ao estudo do significado das expressões da linguagem natural, e isso, independentemente de quem as utiliza. Por sua vez a pragmá-

tica estuda como o significado das expressões pode ser diferente, de acordo com o contexto em que são utilizadas, isto é, como elas se relacionam com o contexto de uso.

A semântica lexical tem seu foco no estudo do significado das palavras da língua e um dos seus principais desafios é a ambigüidade de sentido das palavras: a palavra *banco* possui vários significados completamente distintos (banco da praça, organização financeira, etc). Outra fonte de ambigüidade semântica é aquela oriunda da ambigüidade sintática, como na frase do menino com o telescópio.

Todas essas fontes de ambigüidade e o fato de que a interpretação do significado de uma frase está intimamente ligada ao conhecimento do mundo daqueles que se comunicam, acabam por tornar o estudo da semântica bastante desafiador. A pragmática procura analisar a convenção, o acordo mútuo estabelecido entre falantes e que, de certa forma, rege e orienta a interpretação da linguagem. É justamente esse mútuo entendimento da situação que faz florescer outras interpretações para as frases, menos ligadas ao seu significado literal que às circunstâncias e ao uso que se faz da linguagem.

Neste trabalho, espera-se que os textos que forem escolhidos para descrever domínios de conhecimento e que servirão de base para a construção de ontologias sejam de caráter tipicamente técnico e descritivo (ao contrário de, por exemplo, um diálogo transcrito) e onde poucas construções figurativas sejam utilizadas.

4.2 Componentes de um Tratamento Lingüístico

Nesta seção são apresentados os componentes básicos de um sistema de processamento de linguagem natural.

4.2.1 Tokens e Anotações

Do ponto de vista da computação, textos são representados por meio de arquivos que contém uma seqüência de caracteres. Em geral, no processamento lingüístico é necessário identificar e categorizar as partes do texto que nos interessam. As sentenças, parágrafos, títulos, palavras, conceitos, entidades, etc. presentes no texto precisam ser identificadas e tratadas.

Os tokens são blocos de caracteres normalmente associados a cada pala-

vra, ou item lexical que compõe o texto. A quebra de um texto em tokens não é tarefa tão trivial quanto parece a princípio, já que para isso não basta identificar cadeias de caracteres separadas por "espaços": as palavras podem ser seguidas por pontuação (vírgula, ponto, exclamações, etc.). Outras exceções como, por exemplo, "*aren't*" no inglês (uma contração das palavras *are* e *not* e portanto dois tokens) ou "UDP-Gal:glycosylphosphatidylinositol" (um só token) precisam ser previstas e tratadas durante a "tokenização" (ou etiquetagem, como também é chamado em português) do texto. Em outras línguas o problema pode ser ainda mais complexo: em chinês por exemplo os caracteres correspondem a morfemas monossilábicos; muitos morfemas constituem por si só uma palavra, mas a maioria das palavras são compostas por mais de um morfema.

Outros problemas contribuem para tornar a tokenização uma tarefa não trivial. Os pontos finais usados na abreviação de palavras, como por exemplo, "etc.", ou "S.P." (o estado brasileiro), "Wash." (o estado americano), devem em geral ser mantidos como parte da palavra e não sinalizam necessariamente o fim de uma sentença. O uso de hífens pode tanto estar ligado a quebras de palavras por razões estéticas e tipográficas quanto indicar conceitos compostos, como em "carbon-based life forms" ou "pro-Arab". Hífens são usados também em situações de citação (como na expressão inglesa "a final *take-it-or-leave-it* offer") ou em expressões de quantidade ("a 4-bedroom house" ou "the 12-year-old").

No outro lado do problema, algumas línguas (sobretudo no leste asiático) não costumam usar espaço entre as palavras. Mesmo em alemão, onde se mantém em geral espaços entre as palavras, encontram-se termos como *Dienstleistungsbetrieb* ('empresa do setor de serviços'). Ainda que possa fazer sentido do ponto de vista lingüístico, já que palavras compostas apresentam um conceito próprio, quebrá-las em seus componentes pode ser importante para alguns tipos de análises. Essa tarefa é conhecida como segmentação (ou divisão) de palavras.

Mudando de estratégia na análise da tarefa de tokenização, os espaços em branco podem não necessariamente representar quebras nas palavras. Números de telefone "(11) 3091 5262", cidades como "São Paulo" deveriam ser reconhecidas como um só token. Outros exemplos são os *phrasal verbs* do inglês "make up" ou "work out" (que ademais podem ser separados por outras palavras como em "he couldn't work the answers out") e algumas expressões do tipo "in spite of", "because of", etc. Um mecanismo de tokenização menos

sofisticado pode tentar incorretamente separar essas expressões (ou lexemas[1]).

O reconhecimento de sentenças é também uma tarefa pouco ordinária já que não basta procurar na cadeia de caracteres do texto um delimitador como o ".", "?" ou "!". Ainda que na grande maioria das ocasiões um ponto sinalize efetivamente o limite de uma sentença, existem vários tipos de situação em que isso não é verdade (abreviações são exemplos evidentes disso, como em "O Sr. Fernando telefonou".). As vezes ";", ":" e mesmo "-" podem sugerir a divisão entre sentenças. Outro fator complicador é o discurso direto em frases como " *'É incrível', disse ele, 'como você é bela'* ".

A tokenização e a identificação de sentenças estão entre as primeiras tarefas de uma aplicação de processamento lingüístico. Das informações que elas fornecem a respeito do texto vão depender outras tarefas típicas desse tipo de processamento.

Outro conceito importante no processamento de linguagem natural é o da anotação. Anotações são estruturas de dados simples que conectam blocos de caracteres de um texto com uma característica qualquer que se deseja "anotar". Por exemplo, no frase abaixo poderíamos ter as seguintes anotações:

```
Time flies like an arrow.
1...+....10...+....20...+
```

- anotação 1: tipo="Token", pos_inicial=1, pos_final=4, valor="Time"
- anotação 2: tipo="Token", pos_inicial=6, pos_final=10, valor="flies"
- anotação 3: tipo="Token", pos_inicial=12, pos_final=15, valor="like"
- anotação 4: tipo="Token", pos_inicial=17, pos_final=18, valor="an"
- anotação 5: tipo="Token", pos_inicial=20, pos_final=24, valor="arrow"
- anotação 6: tipo="Sentença", pos_inicial=1, pos_final=25, valor="Time flies like an arrow."

Outra questão importante no tratamento preliminar dos textos em aplicações de tratamento de linguagem é a decisão sobre reduzir ou não as diferentes formas de palavras em lexemas (por exemplo, *"analisar"*, *"análise"*, *"Análises"*,

[1]Um lexema é um conjunto de palavras de mesma classe morfológica que se distribuem de forma complementar e diferem morfologicamente entre si unicamente por sufixos flexivos. Por motivo de economia, os dicionários utilizam intensivamente o conceito de lexema.

"analisou", *"analisando"*, etc. seriam todos representados pelo lexema *"analis"*). Essa tarefa é comumente chamada na literatura de *stemming*, mas também pode ser referida como *lemmatization*. O *stemming* é uma questão bastante debatida já que, apesar de parecer fazer sentido eliminar pequenas diferenças nas palavras e focar no seu lexema, pesquisas empíricas feitas pela comunidade de Recuperação de Informações (*Information Retrieval*) têm demonstrado que basear sistemas apenas nos radicais das palavras não melhora o seu desempenho (HULL, 1996). A principal razão para isso está ligada à perda de informação resultante do processo, sobretudo em línguas que apresentam sistemas mais complexos de inflexão e derivação. Por outro lado, do ponto de vista da disciplina de Extração de Informações (*Information Extraction*), esse tipo de conclusão ainda não é dominante.

4.2.2 Análise Léxico-Morfológica

Um dos componentes fundamentais nas aplicações de tratamento de linguagem natural é o léxico. O léxico é um vocabulário, uma estrutura de dados que contém os ítens lexicais, isto é, as palavras (como *projeto, risco*) ou grupos de palavras (como *gerente de projeto*) e outras informações associadas à eles. Dentre essas informações geralmente encontram-se a categoria gramatical (*POS*), características morfo-sintáticas (gênero, número, pessoa, modo, tempo, etc) e características semânticas (como descrições semânticas e sinônimos, por exemplo).

Na abordagem tradicional, o analisador léxico-morfológico tipicamente quebra a sentença em ítens lexicais e trata em seguida cada um deles, decompondo-os em seus morfemas. Para isso, o analisador busca no léxico as diferentes descrições ligadas aos ítens, trazendo todas as possíveis formas de categorização e associando-as a cada morfema. A resolução de possíveis ambigüidades é deixada para uma etapa subseqüente. Esse tratamento pode ser implementado através de índices ou percurso em árvores, por exemplo.

Em uma abordagem mais recente um etiquetador gramatical (*POS tagger*) associa cada item lexical à apenas uma categoria gramatical (a etiqueta) de acordo com a posição que a palavra ocupa na frase. Essas etiquetas podem incluir artigos, substantivo, adjetivo, verbo, advérbio, preposição, conjunção, entre outros. Assim, ao encontrar a palavra *para* o etiquetador deverá decidir se a associa a etiqueta de preposição ou de verbo, efetivamente removendo a ambigüidade do item na sentença.

Esse processo de etiquetagem tem então como entrada a cadeia de ítens lexicais e como saída um conjunto com os ítens lexicais e a etiqueta apropriada associada a cada um deles. A decisão de melhor etiqueta para cada item pode ser baseada em algoritmos estocásticos ou de regras.

Os algoritmos baseados em métodos estocásticos resolvem situações de ambigüidade através de um modelo treinado com um corpus pré-etiquetado a mão. Um dos corpus pré-etiquetados mais utilizados para isso é o Corpus Brown. Esse corpus (FRANCIS; KUCERA, 1967) é provavelmente o corpus mais conhecido na disciplina de lingüística computacional. Ele contém mais de um milhão de palavras e foi estabelecido na década de 60 e 70 na Brown University com uma amostra bem equilibrada do inglês que se usava na América nesse tempo. Nele se encontram vários tipos de texto, tais como reportagens, ficção, textos científicos, legais, etc. Pelo seu pioneirismo, esse trabalho acabou estabelecendo etiquetas que são ainda muito comuns nos trabalhos de hoje em dia. Ainda que a maior parte dos sistemas do tipo *POS* considere a existência de 8 categorias básicas, alguns lingüístas terminam por refinar a análise de corpus de texto usando classificações mais detalhadas de classes de palavras. Com o passar do tempo essas análises foram estabelecendo uma série de abreviações para se referir às várias partes do discurso. No Corpus Brown, por exemplo, a tag "JJ" é usada para representar adjetivos, mas existem ainda mais de 80 outros marcadores ou tags nesse corpus.

Durante o treinamento desses modelos de etiquetagem, cada item do corpus é analisado e a probabilidade de cada palavra pertencer a uma etiqueta, dado um certo contexto (por exemplo, a etiqueta da palavra precedente) é computada. No momento de aplicação do modelo o analisador utiliza a probabilidade ligada à palavra sendo analisada e ao seu contexto para decidir sobre a melhor etiqueta para ela.

Os algoritmos baseados em regras utilizam bases de regras que são aplicadas contra cada palavra para identificar a categoria desse item lexical. Um exemplo desse tipo de regra poderia ser: se a palavra é composta pelo sufixo *-agem* então a etiqueta é "Substantivo".

Atualmente, um dos etiquetadores mais utilizados é o idealizado por Brill (1995) e que combina os dois métodos mencionados.

O método criado por Brill, ao qual ele deu o nome de "error-driven transformation-based tagger", é baseado em erro pois utiliza um mecanismo de aprendizado supervisionado para estabelecer, em primeiro lugar, a etiqueta

mais provável para cada palavra, e posteriormente, as regras de transformação que podem alterar cada etiqueta de acordo com algum contexto em que a palavra está inserida.

No momento do aprendizado, o algoritmo básico de Brill sugere uma inicialização da etiqueta de cada palavra. Essa inicialização pode ser baseada em um vocabulário construído com base em um corpus de treinamento pré-anotado e que contém a etiqueta que mais comumente está associada a cada palavra do corpus de treinamento. Se a palavra não é encontrada no vocabulário, é então associada a uma etiqueta padrão, como "`noun`".

Em seguida, já na fase de aprendizado, uma série de possíveis transformações são executadas contra o texto recém inicializado. Uma transformação é composta de dois componentes: o contexto de disparo da transformação e a regra de modificação. Um exemplo de contexto de disparo é:

`A palavra precedente é um "determiner" (artigo).`

E um exemplo de regra de modificação é:

`Mudar a etiqueta de "modal" (verbo modal) para "noun" (substantivo).`

Juntos, esses componentes corrigem o erro de etiquetagem associada à palavra **can** na frase:

The/*determiner* can/***modal*** rusted/*verb* ./.

para

The/*determiner* can/***noun*** rusted/*verb* ./.

Cada uma das transformações de um espaço de transformações candidatas é sucessivamente executada contra o texto anotado inicialmente e uma função objetivo (o número total de erros de etiquetagem, por exemplo) é calculada. A transformação associada ao menor valor da função objetivo é escolhida. O corpus inicializado é então submetido a ela e o resultado é usado em uma nova rodada para escolher a próxima transformação que minimiza o número de erros. Assim sucessivamente até que um limite mínimo na função objetivo tenha sido atingido (normalmente, até que não se encontrem mais erros). A fase de aprendizado termina então com uma lista ordenada de transformações.

No momento de etiquetagem de um novo corpus, o mecanismo de inicialização é executado e em seguida cada uma das regras de transformação escolhidas é executada na seqüência em que foram aprendidas.

Em seu trabalho, Brill menciona conseguir com seu método uma taxa de acerto de

96,6% na etiquetagem do Penn Treebank Tagged Wall Street Journal Corpus (MARCUS; SANTORINI; MARCINKIEWICZ, 1993) de 1,1 milhão de palavras.

4.2.3 Análise Sintática

O analisador sintático tem como objetivo reconhecer uma seqüencia de palavras que formam uma frase da língua e eventualmente construir uma árvore de derivação para apontar as relações entre as palavras componentes da sentença. Para isso, é necessário que o analisador ou *parser* tenha acesso ao léxico da língua e a uma gramática que estabeleça as regras de formação de frases.

Chomsky (1956) classifica as gramáticas em 4 tipos:

Regulares (ou tipo 3) são as gramáticas mais restritas. Embora de simples reconhecimento não são capazes de expressar as regras de formação de uma linguagem natural;

Livres de contexto (ou tipo 2) são gramáticas já mais expressivas, porém ainda não resolvem todas as dificuldades de especificação de algumas regras de dependência, como a da concordância verbal, por exemplo;

Sensíveis ao contexto (ou tipo 1) são gramáticas que conseguem tem expressividade suficiente para especificar dependências mas são bem mais complexas no momento do reconhecimento;

Irrestrita (ou tipo 0) é uma gramática que não tem restrições.

Existem vários formalismos que podem ser usados para expressar uma gramática: as gramáticas de constituintes imediatos (*PSG* ou *Phrase Structure Grammar*) (GAZDAR, 1980), as redes de transição (JURAFSKY; MARTIN, 2000), entre outras. A escolha depende sempre do equilíbrio que se busca entre expressividade e complexidade de reconhecimento da gramática para a aplicação desejada.

Como exemplo de representação, imaginemos as seguintes regras de formação de sentenças: uma frase (F) é composta de um sintagma nominal (SN) e de um sintagma verbal (SV); um sintagma nominal é composto de um artigo e um substantivo; um sintagma verbal é uma composição de verbo e de um sintagma nominal. Essa gramática poderia ser representada pelo seguinte formalismo:

$F \rightarrow SN, SV.$

$SN \rightarrow artigo, substantivo.$

$SV \rightarrow verbo, SN.$

Os analisadores sintáticos (ou *parsers*) utilizam diferentes abordagens durante o reconhecimento de uma sentença e para exemplificar algumas delas, tomemos a frase "*o menino olha a menina*".

$artigo \rightarrow [o]; [a].$

$subst \rightarrow [menino]; [menina].$

$verbo \rightarrow [olha].$

Os analisadores do tipo *top down* tem como estratégia identificar em primeiro lugar a frase. Para isso ele precisa associar os argumentos SN e SV aos constituintes da frase. Para obter um sintagma nominal ele irá então procurar um artigo e um substantivo, e assim por diante.

Já um analisador do tipo *bottom-up* inicialmente analisa as palavras e em seguida experimenta combinações para formar as constituintes: ele lê a palavra *o*, reconhecendo-a como um artigo, lê a palavra *menino* associando-a a um substantivo, junta o artigo ao substantivo formando um sintagma nominal *o menino* e assim prossegue até a constituição da frase. Em outras palavras, esse tipo de parser analisa a sentença tratando primeiro as palavras e só então utiliza as propriedades das palavras para deduzir as relações gramaticais e a estrutura da frase.

A combinação das duas estratégias anteriores é a base de um analisador *left-corner*: ao encontrar uma palavra ele usa o esquema *bottom-up* e tenta identificar qual constituinte pode começar pela tal palavra. Em seguida ele continua o restante da análise para esse constituinte de modo top-down.

Outro tipo de analisador importante é o tabular, ou *chart parser*, que possui a capacidade de reter as sub-estruturas já analisadas, o que evita a trabalho de analisar novamente as mesmas palavras em caso de um retrocesso na análise ser necessária, o que o distingue da abordagem *left-corner*.

4.2.4 Outros Tratamentos

Um importante problema no tratamento lingüístico é aquele relacionado as co-referências textuais. Nas linguagens naturais ocorre comumente um fenômeno em que duas ou mais expressões diferentes em um texto se referem a uma mesma entidade. A essa entidade normalmente dá-se o nome de entidade nomeada (ou mencionada).

Imaginando-se que essa entidade já foi mencionada alguma vez em algum ponto do texto, quando encontramos uma nova expressão que retoma a referência à mesma entidade, a essa expressão dá-se o nome de anáfora. Assim, a expressão anafórica e a expressão antecedente co-referenciam a entidade em questão.

As anáforas são constituídas de diversas formas e uma das mais comuns é através do uso de um pronome, como no exemplo "O chefe chegou mas ele não falou com ninguém.". Outro tipo de anáfora é aquele em expressões diferentes mas semanticamente relacionadas são usadas, como em "O presidente subiu a rampa. Lula acenou e entrou no avião. Lá se foi o mandatário do país.". Este exemplo mostra também a importância do reconhecimento de entidades nomeadas (ou mencionadas, como também são conhecidas): "Lula", "o presidente" e "o mandatário do país" são todas expressões que referenciam a mesma entidade.

É um dos desafios da Processamento de Linguagem Natural analisar a conexão entre as entidades mencionadas resolvendo e identificando as co-referências, ou seja, as expressões que se referem às mesmas entidades.

4.3 GATE: um Framework para Processamento Lingüístico

Para viabilizar a implementação do método proposto neste trabalho foi utilizado o GATE (CUNNINGHAM; GAIZAUSKAS; WILKS, 1996), o *framework* de desenvolvimento de sistemas para processamento de linguagem natural desenvolvido na Universidade de Sheffield. O GATE, ou *General Architecture for Text Engineering*, é um dos ferramentais mais utilizados pelos pesquisadores da área para o apoio à construção de sistemas de engenharia lingüística. Desde 1996, quando a sua primeira versão foi estabelecida, o GATE vem sendo usado em vários trabalhos acadêmicos na área do processamento de linguagens naturais (MAYNARD et al., 2000). Atualmente na versão 4, o GATE tem recebido constantes atualizações e melhoramentos para se adequar aos requisitos da comunidade e para integrar sua operação com outras ferramentas, tais como o Weka (WITTEN; FRANK, 2005), o UIMA (FERRUCCI; LALLY, 2004), o Wordnet (FELLBAUM, 1989), o Lucene (HATCHER; GOSPODNETIC, 2005), o API de busca do Google (www.google.com/api), entre outros.

A infraestrutura oferecida pelo GATE é composta de uma arquitetura que descreve os componentes que formam o sistema de processamento de linguagem, um *framework* com bibliotecas de classes Java e *API* usados construção desses sistemas e um ambiente de desenvolvimento gráfico que pode ser utilizado para definir os componentes.

Um ponto importante na escolha do GATE como instrumental para este trabalho foi a grande e ativa comunidade de usuários e colaboradores. O *mailing list* mantido pela comunidade (http://gate.ac.uk/mail/index.html) é bastante participativo, o que possibilita um desenvolvimento contínuo e aberto da ferramenta.

4.3.1 O GATE e seus Recursos

Como definido pelos seus autores no manual de utilização (CUNNINGHAM et al., 2006), o GATE possui três tipos básicos de recursos que são combinados para modelar os componentes de um sistema de processamento lingüístico:

- Recursos de Linguagem (*Language Resources*): são os documentos, dicionários, corpus, ontologias;
- Recursos de Processamento (*Processing Resources*): representam os componentes que operam nos recursos de linguagem criando novas informações;
- Recursos de Visualização (*Visual Resources*): são os componentes GUI usados para visualizar e editar os componentes de linguagem e de processamento.

Juntos, esses três tipos de recursos podem ser usados para modelar e compor sistemas de processamento de linguagem baseado no GATE. O conjunto de recursos que são disponibilizados pelo sistema é conhecido por CREOLE (*Collection of Reusable Objects for Language Engineering*). Quando conjuntos de recursos de processamento são agrupados e executados em seqüência contra algum conjunto de recursos de linguagem tem-se o que no ambiente do GATE se convencionou chamar de "aplicação".

O GATE também oferece formas persistentes de armazenagem de recursos de linguagens em "*datastores*". Um *datastore* pode ser "*serial*", quando se baseia no sistema de arquivos do sistema operacional, ou "*database*", se utiliza um gerenciador de base de dados.

4.3.2 Anotações

As informações produzidas a partir de recursos de processamento tais como analisadores sintáticos (*parsers*), etiquetadores, etc, são geralmente representadas através de anotações.

Esse tipo de representação pode ser feita basicamente com uma das seguintes maneiras:

- Uso de marcações (*markup*): inclusão dos dados da anotação no próprio texto com o uso de linguagens de marcação, tais como SGML ou XML. Um corpus que utiliza essa abordagem é o British National Corpus (BURNARD, 2000) com 100 milhões de palavras;
- Uso de referências: modelo em que as anotações são armazenadas em um arquivo separado do texto original e em forma de grafos que fazem referência a fragmentos do texto.

O GATE utiliza a segunda abordagem e para isso define os seguintes componentes de uma anotação:

- uma identificação, isto é, um número único para cada anotação;
- um tipo, que representa a categoria da anotação (já que diferentes recursos de processamento podem criar vários tipos de anotação);
- nós de início e fim, que denotam o fragmento do texto a que se refere a anotação;
- e um conjunto de *features*, pares de atributo e valor que fornecem informação adicional sobre a anotação.

Por exemplo, na frase:

O Sr. Carvalho é o novo reitor da Universidade de Sergipe.

um grafo como o da figura 4.1 pode ser criado para representar anotações.

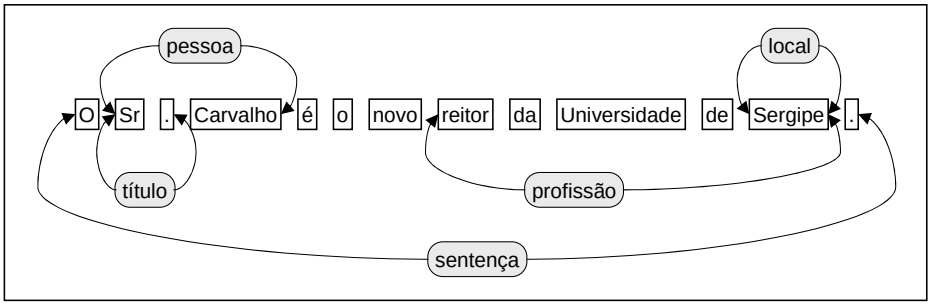

Figura 4.1: Grafo representando anotações.

Como se pode observar, a anotação "*título*" possui dois apontadores: o primeiro para o "S" da cadeia de caracteres "Sr." e o segundo para o "." dessa mesma cadeia. Já a anotação "*pessoa*" aponta para o início e o fim da cadeia "Sr. Carvalho". É importante notar que nada impede que haja uma sobreposição total ou parcial entre as anotações.

As anotações no Gate são mantidas em um *AnnotationSet*, ou conjunto de anotações, e que será permanentemente associado ao corpus original.

4.3.3 Recursos de Processamento Mais Comuns

O GATE fornece um conjunto de componentes que podem ser modificados e combinados de acordo com os objetivos da aplicação lingüística que deseja construir. Dentre os componentes do GATE de uso mais comum estão os tokenizadores, separadores de sentença, etiquetadores, identificadores de entidades mencionadas (ou nomeadas).

4.3.3.1 Tokenizador

Como visto na sessão 4.2.1, os *tokens* são blocos de caracteres associados a palavras, símbolos, números e pontuação e que constituem os ítens lexicais mais básicos do texto. O GATE fornece como parte dos seus componentes pré-definidos, um tokenizador para a lingua inglesa.

O *Tokeniser* é um tokenizador implementado por uma série de transdutores de estado finito especificados com gramáticas JAPE (um dos recursos do Gate que será definido a seguir na sessão 4.3.4).

Um *token* pode ser uma palavra, um número, um símbolo ('&','$'), pontuação (';', '(') ou um *token* de espaço (seja ele constituído de espaço em branco ou caracteres de controle do tipo *linefeed* ou *carriage return*). Uma palavra pode ser qualquer combinação de letras, incluindo hífens mas incluindo símbolos e pontuação. Números também são identificados como quaisquer combinação de dígitos

O tokenizador obedece uma série de regras definidas a maneira de um transdutor onde se especifica um padrão a ser identificado no texto de entrada e uma saída. Cada regra se divide em dois lados. No lado esquerdo é declarada uma expressão regular que vai eventualmente ser "casada"com a entrada representada pelo texto. No lado direito da regra se descrevem as anotações que serão adicionadas ao conjunto de anotações do corpus quando o lado esquerdo da regra disparar, isto é, houver um *match* entre a expressão regular e o texto lido. Os dois lados são separados pelo símbolo '>'.

Os seguintes operadores podem ser usados no lado esquerdo da regra:

'*' : indica 0 ou mais ocorrências (do padrão que precede o símbolo);

'?' : indica 0 ou 1 ocorrência (do padrão precedente);

'+' : indica 1 ou mais ocorrências (do padrão precedente);

'|' : é o operador lógico 'OU'.

O lado direito é definido seguindo o formato abaixo:

{lado esquerdo} > {tipo de anotação}; {$atributo_1$}={$valor_1$}; ...; {$atributo_n$}={$valor_n$}

Em exemplo baseado no manual de utilização do Gate, a seguir tem-se uma regra do tokenizador que identifica palavras que começam com uma letra maiúscula:

```
"UPPERCASE_LETTER" "LOWERCASE_LETTER"* >
 Token;orth=upperInitial;kind=word;
```

Essencialmente essa regra especifica que uma seqüência que comece com uma letra maiúscula e for opcionalmente seguida de uma ou mais letras minúsculas será

anotada como um `Token`. Essa anotação terá um atributo chamado `orth` que receberá o valor `upperInitial` e um segundo atributo `kind` que terá o valor `word`.

4.3.3.2 Separador de Sentenças

O separador de sentenças (ou *sentence splitter*) é mais um componente básico do Gate que gera anotações fundamentais para vários outros módulos de tratamento lingüístico. Composto por vários transdutores de estado finito que operam em seqüência, este módulo reconhece as principais marcas de pontuação para determinar aonde uma sentença acaba. Para evitar a quebra no meio de uma sentença, o separador de sentenças verifica uma lista de abreviaturas mais comuns da lingua. Cada sentença é então marcada com uma anotação do tipo *Sentence*.

4.3.3.3 Etiquetador Gramatical (POS Tagger)

O GATE oferece um etiquetador gramatical (*Part of Speech tagger*) que é uma implementação do algoritmo de Eric Brill mencionado na sessão 4.2.2 modificada para uso no GATE por Hepple (2000).

Esse etiquetador associa cada ítem lexical das sentenças a uma categoria gramatical (artigos, substantivo, adjetivo, verbo, etc.) usando um conjunto de regras e características tais como a posição que a palavra ocupa na frase, a categoria gramatical das palavras ao seu redor, e a morfologia da palavra (atributos como singular, plural, maiúscula, etc).

4.3.3.4 Analisador Morfológico

Vários componentes do GATE trabalham em cascata, isto é, usam o resultado do processamento de outros componentes para enriquecer o tratamento e análise do texto feito até então. O Analisador Morfológico (*Morphological Analyser* é mais um deles. Tendo como entrada as anotações do tipo `Token` e as demais anotações criadas pelo etiquetador gramatical (*POS tagger*), o analisador morfológico utiliza uma série de regras baseadas em expressões regulares para identificar e registrar na anotação de *Token* características de cada palavra tais como gênero, número, modo e tempo verbal, raiz das palavras e seus afixos. O conjunto preliminar de regras fornecido inicialmente no GATE pode ser modificado e novas regras podem ser criadas para adequar o tratamento a novas línguas ou mesmo a novos temas.

4.3.3.5 Gazetteer - Listas de Referências

Gazetteer é o nome que se dá a um dicionário geográfico, uma lista de referência sobre lugares, acidentes geográficos, características demográficas de um local, e outros elementos físicos como estradas, rios, pontes, construções, etc. Tipicamente, dado o nome de um lugar, um *gazetteer* fornece características e descrições associadas a ele.

O GATE estende o conceito de *gazetteer* e estabelece um mecanismo de busca (*lookup*) em listas de variados assuntos. Assim, uma lista pode conter nomes de cidades, outra fornece nomes de moedas, dias da semana, organizações importantes, etc. Abaixo tem-se um pequeno exemplo de uma lista com dias da semana (mantida, digamos no arquivo `DaysOfWeek.lst`):

```
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

Cada uma das listas é mantida em arquivos distintos e é associada a um tipo de anotação. Nesse exemplo, o tipo de anotação poderia ser "DiaDaSemana". Essas listas são então compiladas em máquinas de estado finito que, quando executadas contra uma entrada de texto, reconhece a menção a um dia da semana e associa à cadeia de caracteres correspondente a anotação "DiaDaSemana".

Esse mecanismo de especificação de reconhecedores baseado em grandes listas de palavras é bastante versátil permitindo a identificação de palavras e a sua associação com vários tipos de entidades. O GATE já traz inicialmente diversas listas, tais como cidades, países, companhias, organizações, títulos, nomes típicos, sobrenomes comuns, profissões, unidades de tempo, etc, mas permite ainda a edição e inclusão de outras com tantos temas quantos necessários.

4.3.3.6 Resolução de Co-referências e Entidades Nomeadas

O GATE oferece também a possibilidade de se modificar alguns componentes que identificam entidades nomeadas e fazem a resolução de co-referências. Os módulos conhecidos como *NE Transducer*, *OrthoMatcher* e *Pronominal Coreference* podem ser usados para identificar entidades (com base em listas mencionadas na sessão anterior) e relacionar as co-referências encontradas, criando anotações que demonstram essas associações, marcando anáforas e antecedentes.

4.3.4 JAPE e a Identificação de Padrões de Anotações

Além desses recursos, o GATE disponibiliza também uma *engine* de identificação de padrões de anotações chamada JAPE, *Java Annotation Patterns Engine* (CUNNINGHAM; MAYNARD; TABLAN, 2000) que provê transdutores de estado finito baseados em expressões regulares que atuam tendo como entrada o texto e as anotações definidas no *AnnotationSet* do GATE. Assim, a partir de uma gramática JAPE, cada um dos transdutores JAPE definidos são executados e, sempre que as regras definidas na gramática são satisfeitas pelas anotações existentes no corpus, uma saída é gerada (composta tipicamente de outras anotações) ou algum código Java que tenha sido especificado é disparado. Essa capacidade de geração de anotações e execução de código Java faz do JAPE um dos principais recursos de expansão do GATE, permitindo a construção de componentes de tratamento lingüístico completamente novos.

Uma gramática JAPE é composta de um conjunto de fases que são executadas seqüencialmente. Cada fase é composta por:

- Um nome único;
- Um identificador de entrada que define os tipos de anotação que serão considerados como entradas para a fase;
- Opções que modificam o comportamento da máquina JAPE durante a execução da fase;
- Macros para a definição de componentes da gramática (opcionalmente);
- Regras da gramática.

Por sua vez, uma regra é estabelecida com:

- Um nome único;
- Uma prioridade (opcional);
- Regras "à esquerda" (*left-hand-side*) que especificam expressões regulares que identificarão padrões de anotação. Pode conter operadores de expressão regular tais como "*" (zero ou mais ocorrências), "?" (zero ou uma ocorrência), "|" (ou) ou "+" (uma ou mais ocorrências);
- Regras "à direira" (*right-hand-side*) que definem as ações que devem ser executadas quando as regras "a esquerda" forem satisfeitas. Pode conter qualquer bloco de código Java válido.

A seguir, tem-se um exemplo da gramática JAPE:

```
1  Phase:Fase1
2  Input:Token
3  Options: control = appelt
4
5  Rule:IdentificaUniversidades
6  Priority: 25
7
8  (
9      {Token.string == "Universidade"}
10     ( {Token.string == "Federal"} )?
11     ( {Token.string == "de"} |
12       {Token.string == "da"} |
13       {Token.string == "do"}  )
14     ( {Token.category == "NNP"} )+
15
16 ):nomeOrg
17
18 --> :nomeOrg.Organizacao = {regra="IdentificaUniversidades"}
```

Esse exemplo abre com a definição de uma única fase, `Fase1`, e especifica na linha 2 que serão consideradas as anotações do tipo `Token` (todos os outros tipos de anotações presentes no *AnnotationSet* serão desprezadas). Em seguida define-se a opção que a engine do JAPE deve executar em modo `appelt`, o que significa que se várias regras a esquerda forem satisfeitas simultaneamente somente a regra mais longa dentre elas será considerada. Se houver mais de uma regra nessas condições então a regra com a prioridade mais alta será aplicada. A regra `IdentificaUniversidades` tem prioridade 25, como se observa na linha 6.

A regra "à esquerda" (o bloco entre as linhas 8 e 16 que antecede o '`->`') contém a expressão regular que será executada contra as anotações de entrada. A regra especifica que a seqüência de tokens `Universidade`, seguida opcionalmente ("?")por uma ocorrência de `Federal`, seguida de um entre os tokens (`de`, `da`, `do`) e de um ou mais ("+") substantivos próprios. Como se vê, cada `Token` tem pelo menos duas *features*: `string`, que contem a palavra ou símbolo do texto, e `category`, que contém a categoria gramatical (*POS tag*) atribuída pelo etiquetador gramatical em um passo anterior do processamento do texto. Essa seqüência de anotações identificada será então referenciada como `nomeOrg`.

A regra "à direita" (que sucede o `->` na linha 18) será executada para cada seqüência de anotações `nomeOrg` e irá criar uma nova anotação que chamará `Organizacao` e que se estenderá por toda a seqüência descoberta. Como exemplo, se a regra for aplicada contra a seqüência de tokens "Universidade", "de", "São" e "Paulo", uma nova anotação será criada e se estenderá pela seqüência "Universidade de São Paulo". Essa nova anotação terá apenas uma *feature*, `rule`, que receberá o valor `IdentificaUniversidades`, para caracterizar a regra usada em sua identificação e criação.

5 Método Semi-Automático para Construção de Ontologias de Domínio

Essencialmente, o método proposto neste trabalho deverá fornecer apoio a um ontologista na criação da ontologia, sugerindo conceitos, definições, relações e restrições capturadas a partir do texto. Esses componentes são então utilizados na geração de uma representação formal dessa ontologia com base na linguagem OWL.

Baseada nesse método, uma aplicação será implementada para automatizar o tratamento dos textos e a criação da ontologia. Em linhas gerais, as capacidades desse sistema podem ser descritas da seguinte forma:

1. Constituir um corpus de textos que sejam representativos do domínio cuja ontologia se pretende construir;
2. Utilizar técnicas de Processamento de Linguagem Natural para identificar vários componentes lingüísticos de interesse no corpus;
3. Identificar no corpus possíveis candidatos a conceitos;
4. Identificar no corpus relações entre esses conceitos;
5. Apresentar ao usuário os conceitos e relações descobertos para sua verificação e ajuste;
6. Representar esses conceitos e relações na linguagem formal escolhida (OWL).

5.1 Premissas e Escopo

O método proposto para a construção desse sistema é baseado na identificação de conceitos presentes no texto e nas relações expressas através de padrões lingüísticos entre esses conceitos.

Uma das premissas básicas do método é que os principais conceitos do domínio para o qual se deseja estabelecer a ontologia sejam comumente referenciados nos textos mais significativos desse domínio. Assim, a descoberta de conceitos no texto é

focada nos termos que mais freqüentemente ocorrem no corpus analisado. Por exemplo, uma análise dos termos do PMBOK, *Project Management Body of Knowledge*, (Project Management Institute, 1996) revela que o termo *project* é o mais comum do documento, como intuitivamente se esperaria.

Outra premissa fundamental é que sejam mencionadas nesse corpus as relações existentes entre os vários conceitos. As relações mais interessantes ao problema são aquelas do tipo "IsA" de classe-subclasse. Exemplos dessa relação extraídos literalmente do PMBOK são "a *project charter* **is a** *document* that formally recognizes the existence of a project", "the *project plan* **is a** formal, approved *document* used to manage and control project execution", "a *deliverable* **is a** tangible, verifiable *work product*". Esse mesmo tipo de relação pode ser representada com outros padrões lingüísticos, como em "a *program* **is a** *group of projects*", razão pela qual esses padrões devem ser cuidadosamente estabelecidos para maximizar a identificação dessas relações.

Ainda assim, não se espera que o corpus contenha todos os conceitos e relações de um domínio a que ele se refere, o que fundamenta a expectativa de que a ontologia criada por este método seja apenas parcial (e não completa).

5.1.1 Diretrizes do Trabalho

O desenvolvimento deste sistema se baseia na utilização de uma série de módulos, tais como separadores de sentença, tokenizadores, analisadores morfológicos, analisadores sintáticos, identificadores de conceitos e identificadores de relações, entre outros.

Para que se possa capacitar a expansão deste trabalho no futuro, seja no laboratório KNOMA ou fora dele, as seguintes diretrizes foram estabelecidas para este trabalho: o uso de componentes que possam ser facilmente reconfiguráveis, permitindo assim a expansão da aplicação e a reutilização dos módulos implementados; o uso de métodos que possuam bom desempenho e que possam expandir para tratamento de corpus que contenham milhões de palavras; a utilização de métodos de análise baseados em treinamento (como por exemplo, o método de POS tagging de Brill) ou que sejam facilmente parametrizáveis (como por exemplo, transdutores de estado finito); o uso de ferramentas abertas e de linguagens comumente empregadas pela comunidade científica e, em particular, pelo laboratório KNOMA (tal como Java).

Para viabilizar a construção desse sistema, foi utilizado o GATE, o *framework* de desenvolvimento de aplicações de processamento de linguagem natural citado na sessão 4.3.

5.2 Resumo do Método de Construção

As próximas sessões apresentam o método utilizado neste trabalho para tratamento dos textos e identificação dos conceitos e relações que possam existir no corpus de domínio.

A figura 5.1 mostra de modo esquemático os principais processos do método sugerido. Retângulos sinalizam processos enquanto as formas de canto arredondado demonstram repositórios de regras, especificações de padrões, textos ou listas de ítens específicos. As setas indicam a ordem lógica dos processos que, ao serem executados, incrementam o conjunto de informações extraídas dos textos processados e que é passado sucessivamente a cada novo processo.

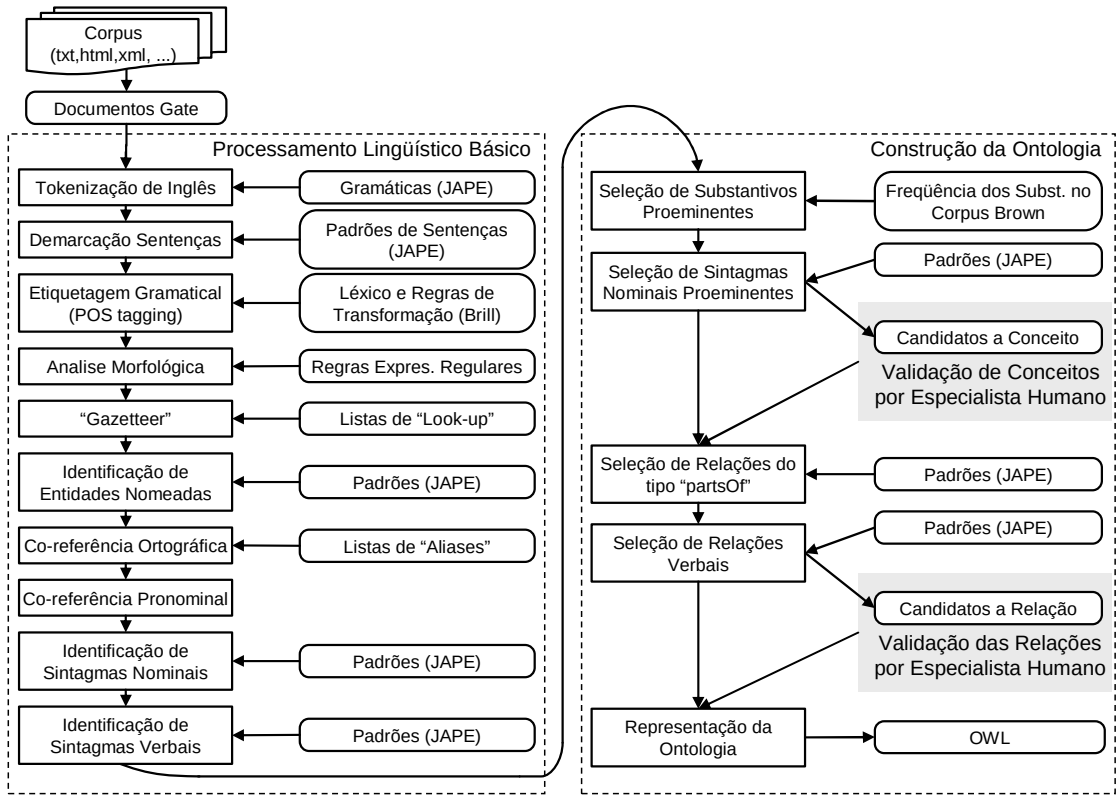

Figura 5.1: Sumário do método de construção de ontologias

Fundamentalmente, a abordagem é implementada com dois grandes grupos de tratamento.

No primeiro grupo vários componentes de Processamento de Linguagem Natural são combinados em um tratamento lingüístico que estabelece anotações para representar as unidades lingüísticas de interesse para os passos posteriores. Como exemplo dessas representações tem-se as sentenças, substantivos, verbos, características morfológicas (tais como singular ou plural) dos elementos, sintagmas nominais e sintagmas verbais. Além desses, outros componentes para resolução de anáforas e de recon-

hecimento de entidades mencionadas são utilizados para adicionar anotações a base de conhecimento relativo ao corpus sendo tratado. Esse tratamento fundamental será discutido em maior detalhe na sessão 5.3.

No segundo grupo vários componentes utilizam a base de anotações gerada anteriormente para buscar padrões, calcular métricas que indicam a proeminência dos termos encontrados, e identificar construções lingüísticas que sinalizem relações. Esses processos, principais responsáveis pela construção da ontologia per se, serão detalhados nas sessões 5.4 e 5.5.

5.3 Processamento Lingüístico Básico

Para que as etapas de identificação de conceitos e relações possam ser executadas, os documentos do corpus de domínio são inicialmente tratados com o uso de técnicas de processamento de linguagem natural para que as principais estruturas lexicais sejam descobertas, analisadas e anotadas. Os principais processos desse tratamento são listados abaixo.

1. Forma-se um corpus a partir da união dos textos selecionados do domínio e que servirão de base para construção da ontologia;

2. Quebra-se todo o texto do corpus em ítens lexicais simples tais como palavras, números e pontuação e estabelece-se uma anotação, aqui chamada '`Token`', para cada um deles (tal como discutido na sessão 4.2.1) em um processo conhecido como **tokenização**;

3. Os *tokens* são então varridos seqüencialmente a procura de marcas que indiquem o início ou fim de sentenças (tal como mencionado na sessão 4.2.1). Uma vez descobertos **os limites de cada sentença**, uma anotação do tipo 'Sentença' é criada para demarcá-la e disponibilizar informações sobre ela para os demais passos do processo;

4. Cada sentença descoberta é então tratada individualmente e todos os *tokens* são novamente analisados, desta vez com o intuito de associar uma categoria gramatical (substantivo, artigo, verbo, etc) a cada um deles. Esse tratamento, conhecido como **etiquetagem gramatical** ou POS *tagging* e mencionado na sessão 4.2.2, adiciona a informação da categoria gramatical de cada *token* na anotação 'Token' já existente;

5. Cada um dos *tokens* de cada sentença é analisado à luz de sua categoria gramatical em busca de suas características morfológicas tais como número, raiz da palavra, tempo e modo verbal (tal como mencionado na sessão 4.2.2. Esse

processo de **análise morfológica** adiciona então novas informações à anotação 'Token';

6. As sentenças são então varridas a busca de expressões que constem de listas de entidades pré-concebidas. Como exemplo, imagine-se que duas listas estejam disponíveis para processamento, uma lista de Profissões, onde aparece o ítem 'prefeito', e uma lista de Cidades, onde se menciona a expressão 'São Paulo'. Ao tratar a sentença "O prefeito afirmou que o transito de São Paulo é caótico" este processo adiciona duas novas anotações ao conjunto de anotações da sentença: a primeira, chamada 'Profissão' é associada à cadeia de caracteres 'prefeito' enquanto a segunda, chamada 'Cidade', é associada a cadeia 'São Paulo'. Esse processo discutido na sessão 4.3.3.5 é conhecido como ***Gazetteer*** e busca incorporar elementos semânticos às anotações do texto;

7. Cada sentença é analisada e os *tokens* que a compõem são verificados contra várias listas de nomes, organizações, títulos, e outras entidades comuns. Os eventuais casamentos (*matches*) entre os *tokens* e os ítens das listas são marcados com anotações de entidades nomeadas (vide sessão 4.2.4). Como exemplo deste processo de **identificação de entidades nomeadas**, na frase "O Sr. Bush foi agraciado com o título honorário da Universidade do Texas." os casamentos de 'Sr.' com um item da lista de títulos, de 'Bush' com um item da lista de nomes, de 'universidade' na lista de organizações, e de 'Texas' na lista de localidades são combinados de forma que as cadeias 'Sr. Bush' e 'Universidade do Texas' sejam marcadas como entidades nomeadas no texto em anotações específicas;

8. Uma vez que as entidades nomeadas sejam identificadas e marcadas, cada sentença é novamente varrida em busca de outras referências a elas mas com outra característica ortográfica. Neste tratamento de **co-referência ortográfica**, como é chamada aqui, são utilizados os mesmos mecanismos de busca e casamento de seqüência de *tokens* e listas específicas de sinônimos. Assim, é possível, por exemplo, marcar com uma nova anotação que 'IBM' e 'Big Blue' são co-referências a mesma entidade (tal como visto na sessão 4.2.4), assim como 'Coke' e 'Coca-Cola', 'New York' e 'Big Apple', ou 'National Aeronautics and Space Administration' e 'NASA', entre outros;

9. Na seqüência, outros exemplos de **co-referência** a entidades são reconhecidas nas sentenças, desta vez com o foco **em pronomes** que atuam como anáforas (sessão 4.2.4), como no exemplo em inglês "The McGill University is very old. It was founded in 1821". Nesse fragmento de texto, o *token* 'it' se refere à universidade McGill. Uma anotação demonstrando a conexão do *token* com esse seu antecendente é adicionada às anotações do corpus;

10. Baseado na categoria gramatical de cada *token* é feita então a **identificação de**

sintagmas nominais, isto é, grupos de palavras que exercem ou sofrem alguma ação verbal (tal como discutido na sessão). Aqui o foco é buscar a mais longa combinação de substantivos nessas condições, como por exemplo na frase "The project management discipline is relatively new", ainda que seja 'discipline' o foco principal da assertiva, é importante qualificar completamente que tipo de disciplina é nova, afinal. Anotações são adicionadas ao corpus para delinear os limites do sintagma nominal encontrado;

11. De forma similar ao processo anterior, cada sentença é varrida e seus *tokens* analisados quanto a sua categoria gramatical a busca de formações centradas ao redor de verbos (mas compostas também por preposições e advérbios) e que expressam as ações na sentença, os **sintagmas verbais**. Como no exemplo "Projects *are composed of* processes.", uma anotação é incluída no conjunto de anotações do corpus para delimitar o sintagma verbal encontrado (em negrito no exemplo).

Ao final dessa seqüência de processos o corpus possui não somente o texto original mas também todo o conjunto de anotações incluídas e refinadas ao longo de cada tratamento lingüístico específico.

5.4 Identificação de Conceitos

Com os resultados do processamento lingüístico básico, a próxima etapa do método é marcada pela identificação dos principais conceitos do domínio presentes no texto. A seguir são descritas as abordagens utilizadas para essa finalidade.

Seja para entendimento de um escrito qualquer, seja para construir uma base de conhecimento a respeito dele, uma das primeiras tarefas ligadas à análise desse texto é o estabelecimento da sua terminologia, isto é, o conjunto de termos usados. Um termo é um ótimo candidato a um conceito do domínio de que trata o texto, e portanto, o reconhecimento desses termos é uma das primeiras atividades necessárias à identificação de conceitos.

5.4.1 Análise dos Termos Proeminentes

Na busca de termos que mais se destacam no corpus de domínio, duas abordagens foram implementadas e avaliadas: a primeira é focada exclusivamente na freqüência absoluta de cada palavra, enquanto a segunda utiliza a freqüência relativa dos termos (substantivos, em particular) do corpus de domínio e a compara com a de um outro corpus composto de escritos típicos da língua inglesa.

5.4.1.1 Freqüência Absoluta das Palavras no Corpus

Um primeiro experimento para estabelecer termos candidatos a conceitos envolveu uma abordagem estatística de contagem composta fundamentalmente dos seguintes passos:

- As palavras (*tokens*) do corpus são filtradas com base em uma lista de *stop words*. O objetivo é remover dos passos subseqüentes as preposições, conjunções e outros ítens lexicais cuja ocorrência é muito freqüente na lingua. Esses tipos de palavras não são bons candidatos a conceitos, por em geral não serem substantivos e, sobretudo, por serem comuns em textos de qualquer domínio, isto é, pouco específicos do domínio sendo estudado;
- As ocorrências de cada palavra no corpus são contadas, considerando-se apenas a variação mais simples da palavra (por exemplo, diferentes combinações de caixa das letras são convertidas para letras minúsculas apenas). Na seqüência as palavras são ordenadas com base nessa freqüência, assim constituindo uma lista das mais freqüentes entre elas;
- Elabora-se com base no produto do primeiro passo, uma lista com todos os bigramas (isto é, grupos de duas palavras) do corpus, contando-se em seguida o número de ocorrências de cada um deles no corpus. Os bigramas de alta freqüência são bons candidatos a conceitos;
- Similarmente ao passo anterior, constrói-se uma lista com todos os trigramas (grupos de três palavras) do corpus e conta-se o número de ocorrências de cada um. Os trigramas mais freqüentes constituem, por sua vez, bons candidatos a conceitos.

O método é de simples implementação, o que confere ao processo rapidez e agilidade no processamento, algo que pode ser crucial em um corpus de grande volume. Como exemplo da aplicação dessa abordagem pode se observar na tabela 5.1 alguns dos termos (n-gramas) mais freqüentes no corpus do PMBOK:

Tabela 5.1: Unigramas, Bigramas e Trigramas mais freqüentes do corpus do PMBOK

Unigrama	**Bigrama**	**Trigrama**
project	project management	project management team
management	project plan	change control system
cost	management plan	work breakdown structure
section	management team	project plan execution

Tabela 5.1 – continuação da página anterior

Unigrama	Bigrama	Trigrama
risk	performing organization	project management process
plan	change control	overall change control
team	project team	project life cycle
planning	project scope	scope change control
work	scope statement	project plan development
scope	quality management	risk response development
process	general management	overall project plan
quality	cost estimate	activity duration estimate
control	quality control	project management software
product	breakdown structure	general management skill
information	project life	risk management plan
schedule	product description	project network diagram
process	project manager	communication management plan
performance	corrective action	quality management plan
organization	project performance	risk response control
activity	risk event	project cost management

É importante notar que esse processo não leva em conta a função das palavras na frase, já que simplesmente seleciona as combinações de palavras imediatamente sucessivas (nos casos de bigramas e trigramas) mais freqüentes. A seleção de termos é excessivamente influenciada pela freqüência de uso das palavras na lingua inglesa e não necessariamente pela sua relevância no domínio tratado no texto, característica que **foi julgada inadequada para o propósito deste trabalho**.

5.4.1.2 Comparação da Freqüência Relativa no Corpus Brown

A abordagem estabelecida neste trabalho para lidar com os problemas apresentados pelo processo anterior envolve a análise da freqüência relativa de cada conceito no texto em relação a freqüência típica desse conceito nos escritos comuns da língua. Para implementar essa abordagem foi utilizado aqui o Corpus Brown como uma amostra equilibrada de textos em inglês que versam sobre assuntos variados, reportagens, textos científicos, legais, etc. A idéia fundamental é que se um conceito aparece no texto de referência com a mesma freqüência relativa que aparece no Corpus Brown pode-se então afirmar que ele não é particularmente importante ou específico para o domínio sobre o qual versa o texto de referência. Por outro lado, se a sua freqüência relativa

no texto é significativamente mais alta que a observada no conjunto de escritos do Corpus Brown, então ele é um conceito importante para o domínio tratado no texto e deve ser considerado para efeitos de construção da ontologia.

Hofland e Johansson (1982) estabeleceram um dos primeiros estudos a comparar freqüências de palavras em corpora distintos, o corpus Brown de inglês americano e o corpus LOB - Lancaster/Oslo-Bergen - (JOHANSSON, 1978) de inglês britânico. Para cômputo da diferença relativa de freqüências de uma dada palavra p nos dois corpus foi usado um coeficiente definido por Yule (1944):

$$K_{Yule}(p) = \frac{Freq_{LOB}(p) - Freq_{Brown}(p)}{Freq_{LOB}(p) + Freq_{Brown}(p)} \tag{5.1}$$

Variando entre +1 e -1, essa métrica, chamada por Yule de coeficiente K, indica um uso mais intenso da palavra no corpus LOB se apresentar um valor positivo, enquanto um valor negativo mostra uma maior utilização da palavra no corpus Brown. Na prática essa métrica não apresenta bons resultados quando os corpus tem tamanhos muito distintos, como é o caso deste estudo, onde o Corpus Brown tem mais de um milhão de palavras e o Corpus do PMBOK tem pouco mais de 37 mil.

Uma alternativa proposta neste trabalho para a eliminar essa característica indesejada é utilizar as freqüências relativas da palavra nos dois corpus, tal como sugerido abaixo, onde se introduz uma métrica aqui chamada de `prevalência`, $Prev$:

$$Prev_{corpus1,2}(p) = \frac{\text{Freqüência Relativa}_{corpus1}(p)}{\text{Freqüência Relativa}_{corpus2}(p)} \tag{5.2}$$

onde

$$\text{Freqüência Relativa}_{corpus_i}(p) = \frac{\text{Freqüência Absoluta da palavra } p \text{ no corpus}_i}{\text{Quantidade Total de palavras no corpus}_i} \tag{5.3}$$

A métrica de `prevalência` indica portanto a prevalência da palavra p no $Corpus1$ comparativamente ao $Corpus2$, e reflete quantas vezes a freqüência relativa da palavra p no primeiro corpus é maior que no segundo corpus. Apesar de bastante simples essa métrica proporciona resultados bastante satisfatórios como se verá na tabela 5.3 abaixo.

Hofland e Johansson também propuseram um teste de ajuste estatístico baseado no teste não-paramétrico de Qui-Quadrado (χ^2) para comparação da freqüência de termos nos dois corpus. Esse teste mede a probabilidade das diferenças nas freqüências encontradas nos dois corpus serem devidas ao acaso, partindo da hipótese nula de que não há diferenças entre os dois grupos no tocante à freqüência dessa palavra.

Calcula-se o indicador de Qui-Quadrado do seguinte modo:

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{E_i} \tag{5.4}$$

onde

$$E_i = \frac{N_i \sum_i O_i}{\sum_i N_i} \tag{5.5}$$

Nesse cálculo O_i representa a freqüência observada da palavra no corpus i, E_i é a freqüência esperada dessa palavra e N_i é a freqüência total de palavras no corpus i.

Foi Pearson (1904) em um dos trabalhos seminais da Estatística quem sugeriu pela primeira vez o teste de χ^2 para verificar a independência de duas variáveis. A tabela de contingência bi-dimensional (tabela 5.2) representa as freqüências dos termos nos dois corpus, tal como freqüentemente se visualiza o problema e o cálculo da estatística de χ^2.

Tabela 5.2: Comparação de freqüências de palavras em dois corpus distintos

	Corpus 1	Corpus 2	Total
Freqüência da palavra p	a	b	$a+b$
Freqüência das outras palavras	c	d	$c+d$
Total	$a+c$	$b+d$	$a+b+c+d$

Usando a hipótese de que não há diferenças de freqüência relativa da palavra p nos dois corpus, pode-se calcular a sua freqüência esperada E_p da seguinte forma:

$$E_p = \frac{a+b}{a+b+c+d} \tag{5.6}$$

A distribuição de Qui-Quadrado com k graus de liberdade segue a distribuição Gama com parâmetros ($\frac{k}{2}$, $\frac{1}{2}$) o que nos permite determinar a probabilidade de se observar o valor χ^2 calculado consultando tabelas estatísticas como a de Barnett e Cronin (1986). Nesse tipo de tabela de contingência com c colunas e l linhas o número de graus de liberdade k é $(c-1) \times (l-1)$, neste caso, 1.

Na tabela 5.3 podem-se observar alguns dos termos encontrados e suas estatísticas correspondentes (no anexo A.1 uma lista mais completa de termos é apresentada). A coluna de *Prevalência no PMBOK*, demonstra quantas vezes a freqüência relativa da palavra no corpus do domínio é maior que no corpus Brown. Com base na hipótese de que as palavras que se destacam por aparecerem mais freqüentemente no corpus

do domínio do que no corpus de escritos típicos são justamente aquelas associadas aos conceitos mais importantes para o domínio, as palavras encontradas são apresentadas ao ontologista em ordem decrescente de *Prevalência no PMBOK*.

Valores de χ^2 superiores a 10,82 indicam um nível de confiança maior que 99,9% de que as freqüências são efetivamente diferentes. Sugere-se então que as palavras cuja *Prevalência no PMBOK* seja maior que 1 e que apresentem valores de χ^2 superiores a 10,82 sejam consideradas como os termos mais proeminentes do corpus de domínio. Como referência, o corpus Brown contém 1.015.945 palavras, enquanto o PMBOK convertido tem 37.238 palavras.

Tabela 5.3: Comparação de freqüências de termos com base no Corpus Brown

Termo	Freq. Absoluta Corpus PMBOK	Freq. Absoluta Corpus Brown	Prevalência no PMBOK	χ^2
project	1192	93	349,69	30.031,4
scope	164	27	165,72	3.796,3
management	367	91	110,03	7.881,5
schedule	134	36	101,55	2.825,7
procurement	78	21	101,33	1.643,9
risk	182	54	91,95	3.747,3
estimates	73	24	82,98	1.463,1
processes	160	57	76,58	3.135,7
team	171	84	55,54	3.017,6
projects	130	68	52,16	2.240,6
tools	63	34	50,55	1.072,7
documents	34	19	48,82	571,0
communications	49	28	47,74	815,5
assumptions	38	23	45,08	617,5
contract	98	60	44,56	1.585,0
product	138	87	43,28	2.204,2
phases	34	24	38,65	516,0
quality	156	114	37,33	2.329,7
planning	174	129	36,80	2.580,7
cycle	28	24	31,83	385,9

Como se pode observar, a métrica de `Prevalência` definida acima, quando corrobarada pela estatística de χ^2, fornece com elevado grau de confiança os substantivos

mais proeminentes no corpus de domínio. Esses substantivos são apresentados para a avaliação do ontologista e a lista dos substantivos ratificados por ele como efetivamente importantes para o domínio em estudo é chamada de substantivos do domínio e servirá de base para as próximas etapas do método.

5.4.2 Sintagmas Nominais que Expressam Conceitos

Em inglês, quando se quer criar novos objetos lingüísticos para expressar conceitos novos, é comum que se faça uso de uma composição de termos já conhecidos. No domínio de gestão de projetos, por exemplo, o conjunto de processos e mecanismos para gerir e controlar os limites dos produtos e atividades que devem ser criados, fornecidos ou executados em um projeto, convencionou-se chamar de "scope change control". É tipicamente com a utilização de uma composição de substantivos ou sintagmas nominais como esse que se representa a maior parte dos objetos importantes no estudo de um domínio, e portanto, os conceitos de uma ontologia desse domínio.

Para identificar os sintagmas nominais mais importantes do corpus de domínio, a referência de um corpus de inglês genérico como o Brown é menos conveniente, já que nele a freqüência de sintagmas nominais mais específicos do domínio em estudo será comumente próxima de zero. A abordagem proposta neste trabalho para lidar com essa característica do uso da língua para formação de conceitos é selecionar, dentre os sintagmas nominais mais freqüentes no corpus do domínio, aqueles que são compostos por substantivos que façam parte da lista de substantivos do domínio ratificada pelo ontologista na etapa anterior. Em outras palavras, apenas os sintagmas nominais compostos por pelo menos um substantivo proeminente no corpus de domínio serão considerados nesta fase.

Uma estratégia comum de identificação de sintagmas nominais consiste na busca de certas combinações de adjetivos, substantivos, particípios de verbos e artigos na frase. Os experimentos e análises preliminares realizados neste trabalho revelaram que os sintagmas nominais mais apropriados para a confecção de conceitos são aqueles compostos apenas por substantivos. A combinação de adjetivos (como em "... a **risky** project management activity ...") e particípios de verbos ("... the **related** management processes ...") nos padrões de identificação de sintagmas nominais foi inibido, portanto.

Uma representação simplificada de uma gramática usada para o reconhecimento desse tipo de sintagmas nominais compostos de pelo menos um substantivo pode ser vista abaixo:

```
( (SUBSTANTIVO)* (SUBSTANTIVO) )
```

onde `SUBSTANTIVO` é uma das anotações associadas aos *tokens* do corpus pelo

analisador gramatical (POS *tagging*). Como mencionado anteriormente, o símbolo '*' indica a ocorrência de 0 ou mais dos ítens que ele pós-fixa.

Na tabela 5.4 podem-se observar os resultados da aplicação desse mecanismo de reconhecimento no Corpus do PMBOK. Poucos sintagmas são mencionados aqui, porém uma lista mais ampla está disponível no anexo A.2.

Tabela 5.4: Sintagmas nominais que ocorrem com maior freqüência no corpus de domínio

Sintagma Nominal	Freq. Ocorrência	Sintagma Nominal	Freq. Ocorrência
project	488	system	48
process	146	contract	44
organization	129	need	44
cost	123	stakeholder	42
activity	90	constraint	39
information	87	estimate	39
product	80	assumption	37
technique	78	document	36
work	75	source	34
change	67	project team	33
output	66	project management	33
risk	65	risk event	32
result	65	knowledge	31
input	65	management	29
project plan	56	scope statement	29
project management team	56	cost estimate	28
tool	55	action	27
resource	55	team	27
application area	52	approach	27
phase	50	relationship	27

A lista completa dos sintagmas nominais selecionados dessa forma é apresentada ao ontologista para sua avaliação e ratificação. A lista ratificada contem o que aqui convencionou-se chamar de sintagmas nominais do domínio e que serão base para os próximos passos de construção de ontologia proposto pelo método.

5.4.3 Construção de uma Taxonomia de Conceitos

Uma vez um sintagma nominal aceito pelo ontologista como efetivo representante de um conceito importante da ontologia, é necessário situá-lo dentro da taxonomia de conceitos dessa ontologia. Como mencionado na sessão 2.5.1, uma taxonomia organiza conceitos ao redor da idéia de herança em que uma classe mais genérica repassa ou transmite suas características a subclasses que, além dessas, possuem também outras características que as tornam mais especializadas.

Para a criação da taxonomia de conceitos baseada na lista de sintagmas nominais do domínio, utiliza-se um mecanismo de decomposição dos sintagmas compostos por mais de um termo de modo que os componentes mais genéricos do sintagma são isoladamente inseridos na taxonomia como classes. Os componentes que progressivamente especificam o sintagma nominal são sucessivamente incorporados e inseridos como subclasses da classe incluída no passo anterior. No exemplo "*scope change control*", o substantivo que confere mais generalidade ao sintagma, "*Control*" é utilizado para criação de uma classe. Em seguida o primeiro componente a especificar essa classe,"change", é incorporado para a criação da subclasse "*ChangeControl*", e assim sucessivamente, de modo que cada classe criada é cada vez mais específica até que se complete o conceito expresso pelo sintagma original. Na figura 5.2 pode-se observar o resultado desse tratamento contra o exemplo mencionado e respectivas classes resultantes.

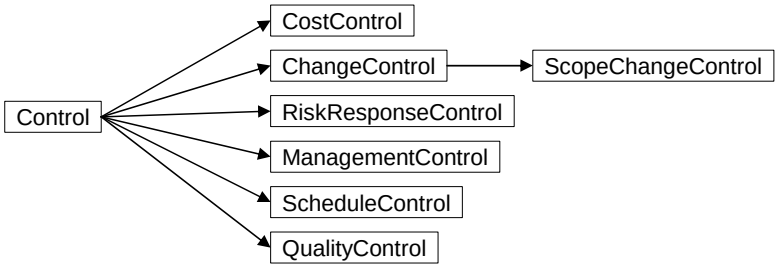

Figura 5.2: Exemplo de construção de relações taxonômicas

Esse tratamento para criação de relações taxonômicas entre os conceitos descobertos é implementado em Java e prevê a identificação e eventual inserção de todos as classes necessárias à inclusão de cada subclasse criada.

5.5 Identificação de Relações

A premissa principal deste método de extração de relações é a de que existe uma forte correlação entre as estruturas sintáticas verbais e as relações semânticas existentes entre as entidades representadas em uma sentença (MAEDCHE; STAAB, 2000).

Baseado nessa premissa, vários abordagens distintas foram estabelecidos e avaliadas quanto a sua efetividade na extração e identificação de relações entre conceitos presentes em textos, como se verá nas próximas sessões.

5.5.1 Padrões Lingüísticos Associados a Relações de Tipo "IsA"

Alguns padrões lingüísticos que costumam ocorrer com relativa freqüência (sobretudo em gêneros literários menos coloquiais) podem ser utilizados para identificação de relações do tipo "IS-A". Um dos primeiros trabalhos a estudar e a identificar esses padrões com o objetivo de descobrir esse tipo de relações em textos foi o de Hearst (1992). Nesse trabalho, componentes léxicos e sintáticos foram combinados para formar os padrões sugeridos pelo autor, alguns dos quais são apresentados a seguir:

- such $noun_0$ as $noun_1$, $noun_2$, ..., { and | or } $noun_n$
- $noun_0$ such as $noun_1$, $noun_2$, ..., { and | or } $noun_n$
- $noun_0$ { including | especially } $noun_1$, $noun_2$, ..., { and | or } $noun_n$
- $noun_1$, $noun_2$, ..., $noun_n$ { and | or } other $noun_0$
- $noun_1$ is a {kind | type} of $noun_0$
- $noun_1$ is a {term | word | concept} {[used] to *verb* | for *verb*-ing } $noun_0$
- $noun_0$ except $noun_1$, $noun_2$, ..., {and | or} $noun_n$

A ocorrência de um desses padrões no texto sugere que os $noun_i$, $1 < i <= n$, são hipônimos de $noun_0$. Um dos problemas que podem comprometer a precisão desse método é o fato de que o hiperônimo (ou seja, o $noun_0$) pode não estar imediatamente conectado a seus hipônimos, isto é, pode haver outras palavras entre ele e seus hipônimos.

Os resultados preliminares obtidos na experimentação de uso dessa abordagem foram desapontadores, sobretudo pelo baixo número de relações efetivamente descobertas. Como conseqüência dessa constatação, esse conjunto de padrões foi então abandonado.

A sessão 5.5.2 mostrará a definição e análise de um novo conjunto de padrões lingüísticos para a busca de relações verbais de interesse para a construção da ontologia.

5.5.2 Padrões Lingüísticos associados a tipos de Relações Específicas

De modo similar ao sugerido por Hearst para relações do tipo "IS-A", padrões léxico-sintáticos foram estabelecidos e implementados para reconhecer vários tipos diferentes de relação, julgadas a priori como sendo bastante comuns nos escritos da lingua:

- **Relação de Caracterização**: este tipo de relação envolvendo construções lingüísticas do tipo " X *is defined* by Y ", " X *is known as* Y ", " X *is referred to as* Y ", demonstra a noção de que um conceito é equivalente a um outro. Um exemplo extraído do corpus de domínio foi "**subprojects** *are* typically *referred to as* **projects** and managed as such.";

- **Relação de Composição**: este modelo de relação indica que um conceito é composto por outro conceito e ela pode ser expressa com construções lingüísticas do tipo " X *is divided into* Y ", " X *is composed of* Y ", " X *is formed by* Y " e outras. A frase "**projects** *are* frequently *divided into* more manageable **components** or **subprojects**" é um exemplo extraído do corpus de domínio e que demonstra esse tipo de relação;

- **Relação de Agrupamento**: relações deste tipo informam que um conceito é resultado de um agrupamento de outro conceito. Para identificá-las foram selecionadas construções como " X *is a group of* Y ", ou " X *is a set of* Y ". Em outro exemplo extraído literalmente do PMBOK, "a **change control system** *is a collection of* formal , documented **procedures** ...".

A representação simplificada abaixo, demonstra o tipo de gramática usada para reconhecimento de alguns desses tipos de padrão lingüístico:

```
((SINTAGMA_NOMINAL)
 (VERBO_MODAL)?
 (ADVERBIO)?
 (VERBO_TO_BE)
 (ADVERBIO)?
 (({Token.string == "divided"}  {Token.string == "into"})    |
 ({Token.string == "subdivided"} {Token.string == "into"}) |
 ({Token.string == "broken"} {Token.string == "down"} {Token.string == "into"}) |
 ({Token.string == "composed"} {Token.string == "of"})      |
 ({Token.string == "formed"}  {Token.string == "by"})       |
 ({Token.string == "constituted"} {Token.string == "of"}))
 (SINTAGMA_NOMINAL)
)
```

Nessa representação, o operador '?' indica que a presença do item que ele sucede é opcional. Além disso, '|' é o operador lógico 'OU'.

Seguindo essa especificação, fragmentos de sentenças, tais como

"... projects are divided into phases ..."

"... processes should always be constituted of activities ..."

"... programs are usually subdivided into projects ..."

podem ser reconhecidos e analisados quanto às relações que eles expressam (nesses exemplo, relações de composição e partes).

O anexo B.4 contém a especificação completa de uma implementação dessa abordagem. Mais de 40 construções verbais distintas foram utilizadas nessa especificação. Quando combinadas com as diferentes variações de artigos, advérbios e verbos auxiliares disponíveis, centenas de padrões verbais foram efetivamente definidos para reconhecimento.

Não obstante essa quantidade bastante considerável de padrões, não é possível afirmar com certeza que todos os principais padrões verbais necessários foram de fato especificados.

Ao analisarmos a aplicação dessa abordagem de busca de relações contra o corpus do PMBOK, várias relações são efetivamente apontadas. Alguns fragmentos de texto onde relações foram reconhecidas são listados a seguir (uma lista mais extensa está disponível no anexo A.3):

- **Project management** *is* a relatively young **profession**, and while there is substantial commonality around what is done, there is relatively little commonality in the terms used.
- A **project** *is* a temporary **endeavor** undertaken to create a unique product or service.
- Most **project life cycles** generally *involve* some form of **technology transfer** or handoff such as requirements to design, construction to operations, or design to manufacturing.
- **Building codes** *are* an example of **regulations**.
- The **design document** *defines* the **product description** for the ensuing implementation phase.

5.5.3 Relações Verbais

Outro processo de identificação de relações foi implementado baseado na análise de estruturas verbais. Nessas estruturas o sujeito e o objeto de um verbo são participantes de uma relação que é definida pelo verbo. Na frase "*gerente de projeto constrói cronograma*", por exemplo, fica estabelecido que entre os dois conceitos, *gerente de projeto* e *cronograma*, existe uma relação de construção.

Para implementação dessa abordagem de seleção de relações, os sintagmas verbais previamente anotados no corpus são extraídos juntamente com os sintagmas nominais referentes ao sujeito e ao objeto da ação verbal. Se ambos os sintagmas nominais foram selecionados na etapa de extração de conceitos (ou seja, se fazem parte da lista de sintagmas nominais do domínio) então a relação verbal é considerada relevante e é selecionada para ser apresentada ao ontologista e representada na ontologia.

Como exemplo de aplicação desse processo, na sentença "*A change control system is a collection of formal, documented procedures*" a relação "*isACollectionOf*" é identificada e associada aos conceitos "*ChangeControlSystem*" e "*Procedure*".

Em outro exemplo a sentença "*The project plan contains the various baselines that will be used to assess project performance.*" fornece evidências da relação *contain* entre os conceitos *ProjectPlan* e *Baseline*, como se observa na figura 5.3.

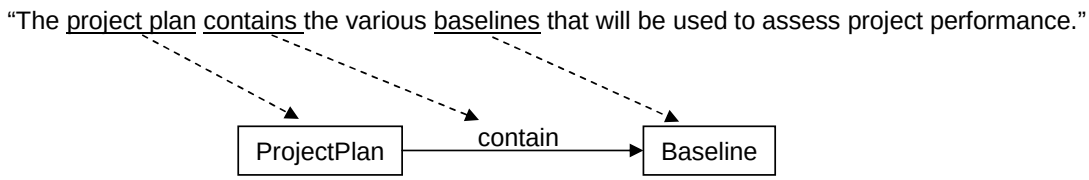

Figura 5.3: Exemplo de construção de relações baseada em estruturas verbais

Essa abordagem é suficientemente abrangente para identificar inclusive relações taxonômicas como na frase " a **project charter** *is a* **document** that formally recognizes the existence of a project. ".

A tabela 5.5 mostra algumas das relações encontradas ao se utilizar a abordagem sugerida contra as anotações do corpus do PMBOK.

Tabela 5.5: Exemplos de relações identificadas através de estruturas verbais envolvendo sintagmas nominais do domínio

Sintagma Nominal 1	**Relação Verbal**	**Sintagma Nominal 2**
Pmbok	include	Knowledge
Project Phase	isMarkedBy	Completion
Project	isComposedOf	Process
Management	encompass	Planning
Budget	is	Constraint
Project Plan	is	Document
Project Phase	isMarkedBy	Review
Product-Oriented Process	isDefinedBy	Project Life Cycle
Variance	isFedInto	Control Process
Project Planning Methodology	is	Approach
Project Plan Execution	is	Process
Work Authorization System	is	Procedure
Work Result	is	Outcome
Project Charter	is	Document
Contract	is	Relationship

5.5.4 Relações de Composição & Partes

Em uma abordagem complementar àquela vista na sessão 5.5.3, outras relações de composição podem ser extraídas, desta vez com base na premissa lingüística simples de que relações do tipo "isPartOf" ou "have" podem ser expressas com construções lingüísticas do tipo:

1. "... SN_1 **of the** SN_2 ..." ;
2. "... SN_1**'s** SN_2 ...".

onde SN_1 e SN_2 são sintagmas nominais.

Exemplos dessas construções são "departments **of the** organization" e "project team**'s** participants". Nesses exemplos específicos, pode-se intuir que um 'departamento' *é parte* de uma 'organização' (ou posto de outra forma, que uma 'organização' *tem* 'departamentos'), do mesmo modo que um 'time de projeto' *tem* 'participantes'.

Para identificar relações desse gênero, basta estabelecer as gramáticas de reconhecimento desses padrões lingüísticos e usá-las contra o corpus de domínio e suas anotações. Na representação simplificada abaixo é mencionada a essência dessas gramáticas.
Primeira construção:

```
(
 (((SUBSTANTIVO)* (SUBSTANTIVO)):parte )
 ({Token.string == "of"} | {Token.string == "OF"} | {Token.string == "Of"})
 ((ARTIGODEF) | (ARTIGOINDEF))
 (ADJETIVO)*
 (((SUBSTANTIVO)* (SUBSTANTIVO)):todo)
)
```

Segunda construção:

```
(
 (( (SUBSTANTIVO)* (SUBSTANTIVO) ):todo)
 (POSSESSIVO)
 (( (SUBSTANTIVO)* (SUBSTANTIVO) ):parte)
)
```

Nessa representação `SUBSTANTIVO`, `ARTIGODEF`, `ARTIGOINDEF`, `ADJETIVO` e `POSSESSIVO` são algumas das anotações associadas aos *tokens* do corpus pelo analisador gramatical (POS *tagging*). Em particular, `POSSESSIVO` é anotado pelo analisador quando o *token* " 's " é relacionado à forma gramatical do possessivo de substantivos (e não à simples contração do verbo "is").

Com respeito à relação estabelecida, o primeiro sintagma nominal do padrão reconhecido será associado à noção do "todo" (como "project team" no exemplo acima), enquanto o segundo sintagma nominal será associado à noção da 'parte' ("participants" no exemplo).

A tabela 5.6 mostra algumas das relações desse tipo encontradas ao se utilizar a abordagem sugerida. Uma lista mais ampla de relações reconhecidas é apresentada no anexo A.5.

Tabela 5.6: Exemplos de relações identificadas através de estruturas que indicam composição

Sintagma Nominal 1	**Relação Verbal**	**Sintagma Nominal 2**
Organization	has	Resource
Project	has	Scope
Project	has	Phase
Project	has	Scope
Resource	has	Cost
Project	has	Product
Activity	has	Outcome
Project Product	has	Description
Product Scope	has	Completion
Project Scope	has	Completion
Project	has	Scope Baseline
Activity	has	Duration
Project Plan	has	Component
Project	has	Cost
Project Product	has	Performance
Procurement Item	has	Cost
Project Stakeholder	has	Need
Project	has	Requirement
Project Participant	has	Expertise
System	has	Performance
Schedule Simulation	has	Result
Procurement Process	has	Review

A lista completa das relações selecionadas por esta e pela abordagem da sessão 5.5.3 é então apresentada ao ontologista para sua avaliação e validação.

5.6 Representação em Linguagem Formal

Uma vez criada uma base de conceitos e relações descobertas, apresentadas ao ontologista e confirmadas por ele, é chegado o momento de estabelecer uma representação da ontologia na linguagem OWL para que ela possa ser refinada em ambientes de edição de ontologias ou mesmo utilizada em aplicações que dela dependam. Ao conjunto de classes e relações efetivamente corroboradas pelo ontologista dá-se o nome a partir daqui de classes e relações aceitas.

Como visto na sessão 2.5.6, o processo de representação da ontologia em OWL envolve fundamentalmente o estabelecimento dos conceitos aceitos como *classes* OWL e das relações aceitas como *propriedades* OWL. Uma vez definida uma propriedade, resta estabelecer as *restrições* que definem quais classes são objetos da relação.

A seguir é visualizado um pequeno fragmento de ontologia representando na linguagem OWL os conceitos *ChangeControlSystem* e *Procedure*, além da relação *isACollectionOf* que existe entre eles (isto é, um *ChangeControlSystem* é uma coleção de *Procedures*).

```
1   <owl:Class rdf:ID="Procedure"/>
2
3   <owl:Class rdf:ID="System"/>
4
5   <owl:Class rdf:ID="ChangeControlSystem">
6     <rdfs:subClassOf>
7       <owl:Class rdf:ID="System"/>
8     </rdfs:subClassOf>
9   </owl:Class>
10
11  <owl:Class rdf:ID="ChangeControlSystem">
12    <rdfs:subClassOf>
13      <owl:Restriction>
14        <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
15        >A change control system is a collection of formal, documented procedures
16         that defines the steps by which official project documents may be
17         changed.</rdfs:comment>
18        <owl:onProperty>
19          <owl:ObjectProperty rdf:ID="isACollectionOf"/>
20        </owl:onProperty>
21        <owl:someValuesFrom rdf:resource="#Procedure"/>
22      </owl:Restriction>
23    </rdfs:subClassOf>
24  </owl:Class>
```

Desconsiderando um pequeno preâmbulo fixo de especificação do arquivo XML (que não aparece acima), o processo inicia com a representação de todas as classes correspondentes aos conceitos aceitos. Cada conceito gera construções como as das linhas 1 e 3. A taxonomia de conceitos aceitos é utilizada para definição de todas as sub-classes, como se vê nas linhas 5 a 9.

Em seguida é representada cada uma das relações aceitas, a exemplo das linhas 11 a 24, como propriedades de objeto (*ObjectProperty* na linha 19) e restrições de propriedade (*Property Restrictions* na linha 21). Essa construção efetivamente estabelece que a classe *ChangeControlSystem* tem uma restrição na propriedade *isACollectionOf* que obriga que pelo menos algum valor (*someValuesfrom*) dessa propriedade é uma instância da classe *Procedure*. De modo mais coloquial, um sistema de controle de mudanças é uma coleção de pelo menos um procedimento.

Como suporte e referência para o ontologista, é incluído como comentário da restrição (linhas 14 a 17), a sentença do corpus do domínio que deu origem ao reconhecimento da relação entre os conceitos. Essa representação pode então ser importada em um ambiente de edição de ontologias tal como o Protégé (STANFORD UNIVERSITY SCHOOL OF MEDICINE, 2007) para visualização e refinamento.

5.7 Descrição da Implementação

A implementação do método de construção de ontologias apresentado neste capítulo foi baseada na utilização do framework de desenvolvimento de sistemas de processamento de linguagem natural GATE, tal como mencionado na sessão 4.3.

O GATE provê uma infraestrutura composta por uma arquitetura e uma bibliotecas de classes que podem ser combinadas entre si e com outros módulos para possibilitar a implementação de capacidades de processamento de linguagem natural em outras aplicações. GATE é escrito em Java e está disponível como software *open source* sob uma licença do tipo GNU.

A grande maioria dos módulos do GATE permitem alguma latitude de customizaçao, seja com o auxílio de parâmetros, seja com a modificação ou criação de novos arquivos de regras, listas, léxicos, gramáticas. Para várias dos processos requeridos, porém, foi necessário a criação de novos programas Java ou mesmo a inclusão de programas Java em alguns dos módulos do GATE.

A seguir destacam-se os pontos mais importantes na implementação das abordagens e métodos citados neste capítulo até aqui.

5.7.1 Implementação do Processamento Lingüístico Básico

O processamento lingüístico básico, tal como mencionado na sessão 5.3, é implementado fundamentalmente com módulos tradicionais do GATE. Abaixo são descritas as características de implementação de cada processo.

- A constituição do corpus é feita com módulos próprios do GATE que identifica e

lê diversos formatos de documento, tais como HTML, SGML, XML, RTF e email. No entanto, como os documentos do PMBOK usados para testar a abordagem estão originalmente em formato PDF, é necessário que sofram uma pequena preparação sistêmica para limpeza de headers e footers que são repetidos a cada nova página para que possam então ser unidos em um corpus;

- A quebra do texto em *tokens* foi , implementada usando-se o módulo *Tokenizer* do GATE;
- A demarcação de sentenças no texto foi realizada com o módulo *Sentence Splitter* do GATE;
- Para executar a etiquetagem gramatical (*POS tagging*) foi usado o módulo de *Part of Speech Tagger* do GATE, que , como mencionado na sessão 4.3.3.3, é uma implementação do algoritmo de Brill;
- A análise morfológica que trata cada *token* do corpus é executada através do módulo *Morphological Analyser* do GATE;
- As anotações semânticas do gazetteer são adicionas pelo módulo de mesmo nome do GATE;
- A identificação de entidades nomeadas é feita com o módulo *NE Transducer* do GATE;
- Co-referências entre entidades nomeadas são resolvidas com o uso do módulo do GATE chamado *Orthographical Coreference*;
- Co-referências que se baseiam em pronomes é resolvida com o módulo de *Pronominal Coreference* do GATE;
- A **identificação de sintagmas nominais** é implementada através da especificação de padrões de anotações expressas com gramáticas JAPE, tal como mencionado na sessão 4.3.4, e fornecidas ao módulo *JAPE Transducer* do GATE. Esse módulo recebe essas gramáticas e constitui transdutores de estado finito que operam sobre as anotações de entrada. Ao ser reconhecido um padrão, as ações correspondentes especificadas em JAPE são disparadas. Essas ações são compostas essencialmente de código Java que usa o API do GATE para a criação/modificação de anotações. As gramáticas e o código Java que implementa as ações são apresentados no anexo B.1 tal como elaborados neste trabalho;
- A **identificação de sintagmas verbais** é feita através da modificação do módulo *VP Chunker* do GATE, um transdutor JAPE que opera com uma gramática específica. A gramática modificada é apresentada no anexo B.2;

5.7.2 Implementação da Identificação de Conceitos

A implementação desta parte do método onde se identifica conceitos candidatos a serem incorporados na ontologia é baseada inteiramente em Java. Os passos a seguir sumarizam essa implementação.

5.7.2.1 Comparação da Freqüência Relativa no Corpus Brown

A sessão 5.4.1 apresenta uma abordagem para a análise e seleção dos termos mais importantes ou proeminentes do corpus do domínio. A seguir são sumarizados os passos utilizados para implementar essa seleção.

- Inicialmente os *tokens* anotados como substantivos são separados e o número de ocorrências no corpus de domínio é calculado para cada um deles (juntamente com a sua freqüência relativa, tal como definido na equação 5.3);
- Cada substantivo tem então sua freqüência relativa no corpus comparada com a freqüência relativa de uso no Corpus Brown. Para isso são usadas a métrica de Prevalência (equação 5.2) e a estatística de χ^2 (equação 5.4);
- Substantivos cujo valor da estatística de χ^2 é superior a 10,82 e cuja Prevalência for superior a 2 são marcados e oferecidos ao ontologista como sendo termos relevantes para o domínio;
- Substantivos que não forem encontrados no Corpus Brown mas que tiverem freqüência relativa superior a 0,02% são também marcados na lista de termos oferecidos ao ontologista já que podem ser específicos do domínio;

Neste ponto do processo há uma interação com o ontologista para confirmação dos termos relevantes ao domínio da ontologia sendo construída. A lista de termos é oferecida em ordem decrescente de freqüência relativa no corpus para avaliação. Espera-se que o ontologista confirme a marcação de termos relevantes feita automaticamente, podendo inclusive marcar termos que não haviam sido pré-selecionados. Os termos apontados como importantes, aqui chamados de 'substantivos do domínio', são recebidos e prosseguem no tratamento automático de identificação de conceitos.

5.7.2.2 Sintagmas Nominais Importantes para a Ontologia

A sessão 5.4.2 justifica e analisa uma abordagem para a descoberta de sintagmas nominais como bons candidatos à conceitos do domínio. A implementação do conjunto de processos necessários à abordagem é apresentada a seguir.

Como visto na sessão 5.7.1 a identificação de sintagmas nominais é feito com a definição de uma gramática JAPE e a utilização do *JAPE Transducer*. O fragmento de

gramática JAPE usada no reconhecimento e anotação de sintagmas nominais é analisado abaixo:

```
1  Macro: SUBSTANTIVO (
2    {Token.category == NN, Token.kind == word} |
3    {Token.category == NNS, Token.kind == word}|
4    {Token.category == NNP, Token.kind == word} |
5    {Token.category == NNPS, Token.kind == word} |
6    {Token.category == NP, Token.kind == word} |
7    {Token.category == NPS, Token.kind == word}
8  )
9
10 Rule: SintNomimal
11 (
12   (SUBSTANTIVO)* (SUBSTANTIVO)
13 ):sn
14 -->
15 :sn.SintagmaNominal={}
```

Inicialmente, para aumentar a legibilidade da gramática, a macro `SUBSTANTIVO` é estabelecida (linhas 1 a 8) e define uma disjunção de várias condições baseadas nas anotações de token. A anotação `category` é criada originalmente pelo etiquetador gramatical seguindo as seguintes especificações:

NN : substantivo singular;

NNS : substantivo plural;

NNP : substantivo ou nome próprio no singular e maiúsculo;

NNPS : substantivo ou nome próprio no plural e maiúsculo;

NP : nome próprio singular;

NPS : nome próprio plural.

A regra de reconhecimento (linhas 10 a 13) especifica que um padrão composto por uma seqüência de pelo menos um `SUBSTANTIVO` seja encontrado para que a ação da regra seja disparada. A ação especificada a direita de `->` determina que uma anotação chamada `SintagmaNominal` seja criada e demarque toda cadeia que disparou a regra.

Com base nas anotações de sintagmas nominais do corpus os seguintes processos são executados por um programa Java usando o API do GATE:

- O conjunto dos chamados substantivos do domínio recebidos do ontologista anteriormente é indexado e posto a espera;

- O conjunto de anotações incorporadas ao corpus é varrido e as anotações de `SintagmaNominal` são selecionadas;
- A cada sintagma nominal é associada a quantidade de vezes em que ele ocorre no corpus de domínio;
- A freqüência relativa de ocorrência de cada sintagma é calculada com base em todas as ocorrências de sintagmas do corpus;
- Todo sintagma com freqüência relativa maior que 0.05% é verificado quanto a sua composição: se for composto por ao menos um dos substantivos do domínio então é marcado como um sintagma candidato a conceito.

Os sintagmas nominais candidatos a conceito são oferecidos ao ontologista para sua avaliação. Aqueles aceitos por ele são então chamados de 'sintagmas nominais do domínio' e prosseguem no tratamento para construção da ontologia.

5.7.2.3 A Construção da Taxonomia de Conceitos

A sessão 5.4.3 sugere uma abordagem para a transformação dos sintagmas nominais do domínio (que aqui, vale lembrar, são compostos exclusivamente de substantivos) em uma taxonomia de classe e subclasse. Para a implementação dessa abordagem um programa Java é utilizado.

Cada um dos sintagmas nominais do domínio recebidos do ontologista é tratado da seguinte forma:

- Se o sintagma nominal é composto de apenas um substantivo, ele é imediatamente incorporado a uma tabela de classes se ele nela ainda não tiver sido inserido;
- Se o sintagma nominal é composto por mais de um substantivo, como em "Sub_1 Sub_2 ... Sub_n", criam-se sucessivamente tantas novas classes quantos forem os substantivos que o compõe, n. Para tanto procede-se da seguinte forma:
  - Constrói-se um novo sintagma nominal composto apenas por Sub_n e adiciona-se à tabela de classes se ele ainda não existir nela;
  - Prefixa-se o sintagma nominal recém construído com o substantivo imediatamente anterior, Sub_{n-1}, de modo a formar um novo sintagma nominal: "Sub_{n-1} Sub_n";
  - Adiciona-se o novo sintagma nominal a tabela de classes, se ele ainda não existir nela, marcando-o como uma subclasse da classe anterior, Sub_n;

  - Repete-se a prefixação/formação e inclusão na tabela de classes dos sintagmas nominais que forem sendo construídos, todos cada vez mais específicos, subclasses dos anteriores imediatos:
    "Sub_{n-2} Sub_{n-1} Sub_n" é inserido na tabela como subclasse de "Sub_{n-1} Sub_n";
  - Para-se a repetição do processo assim que Sub_1 tiver sido prefixado e o sintagma resultante, o original sendo tratado, tenha sido inserido na tabela de classes.

- a tabela de classes (e subclasses) é então disponibilizada para o processo de representação da ontologia em linguagem OWL.

5.7.3 Implementação da Identificação de Relações

A implementação desta parte do método onde se identifica relações candidatas a serem incorporados na ontologia é baseada em uma combinação de processos expressos na linguagem JAPE e em Java. A seguir são sumarizados os passos dessa implementação.

5.7.3.1 Relações baseadas em Estruturas Verbais

Como visto na sessão 5.5.3, o primeiro método de reconhecimento de relações incorporado ao método proposto neste trabalho é baseado no uso de estruturas verbais, os sintagmas verbais, e os sintagmas nominais que colaboram para definição de autoria e foco da ação verbal.

Este processo de seleção de relações verbais utiliza como entrada para seu tratamento as anotações de sintagmas verbais e nominais reconhecidos durante o processamento lingüístico básico. Para sua implementação foi usada uma combinação de gramáticas JAPE executadas pelo *JAPE Transducer* e programas Java (detalhados no anexo B.5).

O cerne do reconhecimento das relações descritas nesta abordagem é representada pela gramática JAPE simplificada abaixo:

```
1 (
2  (ARTIGO) (ADJETIVO)*
3  (SINTAGMANOMINAL):RelNounHead
4
5  ((VERBGROUP) (EXAMPLEOF)? (GROUPOF)? (PREPOSITION)? ):VBPrep
6
7  (DETERMINER)? (ADJETIVO)? (({Token.string==","}) (ADJETIVO))*
8
```

```
9   (SINTAGMANOMINAL):RelNounObj
10  )
11  :def1 -->
12      :def1.RelacaoVerbal = {},
13      :RelNounHead1.RelNounHead = {rule = "REL1-RelNounHead"},
14      :VBPrep.RelVBPrep = {rule = "REL1-VBPrep"},
15      :RelNounObj.RelNounObj = {rule = "REL1-RelNounObj"}
```

Na linha 5 a construção verbal é reconhecida baseado na anotação `VERBGROUP` criada pela etapa de identificação de sintagmas verbais da sessão 5.7.1. Se houver uma preposição ao seu lado ou ainda uma construção que indique exemplos ou grupos, toda a cadeia desde o sintagma verbal será marcada com a anotação `VBPrep`. Exemplos desses fragmentos são:

```
... is a kind of ...
... are types of ...
... are examples of ...
... is the subset of ...
... is a group of ...
... are a collection of ...
```

Na linha 3 tem-se o padrão de busca do sintagma nominal que será o autor da ação verbal. Exemplos dos fragmentos de texto que podem ser reconhecidos por esse padrão são:

```
...  a project ...
...  the control document ...
...  the laborious planning activity ...
```

É importante notar que o texto em itálico não será incluído como limite da anotação.

Similarmente à busca do sintagma nominal anterior, a linha 9 descreve o padrão de reconhecimento do sintagma nominal que será o foco ou objeto da ação verbal.

As anotações acima demonstram um par ordenado de sintagmas nominais conectados por uma expressão verbal, um candidato a relação na ontologia do domínio. Se ambos os sintagmas nominais do par ordenado fizerem parte da lista de sintagmas nominais do domínio, então a relação é selecionada para ser apresentada ao ontologista para validação.

O anexo B.5 mostra de forma mais detalhada a implementação desta abordagem.

5.7.3.2 Relações de Composição & Partes

A outra abordagem de reconhecimento de relações de composição & partes no qual o método apresentado neste trabalho se baseia é a extração de relações do tipo '`isPartof`' ou '`have`' mencionada na sessão 5.5.4.

A gramática utilizada para implementação em JAPE é detalhada no anexo B.6.

5.7.3.3 Representação em Linguagem Formal

A implementação do mecanismo de representação da base de conceitos e relações reconhecidas automaticamente e aceitas pelo ontologista mencionado na sessão 5.6 é feita completamente com base em Java.

O programa Java utilizado, varre as estruturas onde são mantidos os conceitos e relações aceitos pelo ontologista e gera as especificações em linguagem OWL para representá-los. Um maior detalhamento do programa é fornecido no anexo B.3.

6 Avaliação dos Resultados

Para a avaliação da abordagem sugerida e de sua implementação computacional, esse sistema foi utilizado no apoio à construção de uma ontologia para o domínio de gerenciamento de projetos, tendo como corpus base o texto do PMBOK.

A escolha desse domínio foi baseada fundamentalmente no interesse que o laboratório KNOMA de Engenharia de Conhecimento da Escola Politécnica tem pelo tema do gerenciamento de projetos. Vários pesquisadores do laboratório possuem experiências práticas com a disciplina e, também por suas pesquisas na área, têm a capacidade de discutir e avaliar a qualidade dos conceitos e relações descobertos.

6.1 Avaliação do Método de Construção e da Ontologia Resultante

Ainda não existe um claro consenso sobre a melhor maneira de se avaliar uma ontologia, sobretudo quando ela é criada por um método automático ou semi-automático. Neste trabalho consideram-se como critérios importantes de avaliação o grau de cobertura que a ontologia criada tem sobre o domínio de estudo e o apoio que ela oferece ao ontologista na busca dos conceitos do domínio.

6.1.1 Avaliação da Extração e Criação de Conceitos

Como uma referência para o conjunto de conceitos importantes que deveriam ser incluídos na ontologia (ou seja, o conjunto de referência ou *golden standard*), utilizou-se o glossário de definições fornecido ao final do PMBOK pelos seus autores (239 definições são relacionadas no glossário da versão utilizada neste trabalho). Após análise dessas definições, alguns ítens lexicais, tais como "*as-of date*", "*link*", "*node*", "*subnet*" foram removidos da lista de conceitos para efeito de estabelecimento do conjunto de referência por terem sido julgados de pouca importância para o domínio de Gerenciamento de Projeto para merecerem uma definição. Da mesma forma, foram retirados ítens como "*accountability matrix*" ou "*contingency allowance*" que, apesar de serem referenciados no glossário, não são mencionados nos textos do PMBOK na versão utili-

zada neste trabalho.

Para a avaliação da ontologia criada em relação à cobertura do domínio que ela oferece, foi usada uma análise quantitativa dos conceitos baseada na tradicional métrica de cobertura (*recall*) do campo da Recuperação de Informações. No contexto de ontologias, cobertura pode ser considerada como o número de conceitos extraídos do corpus pelo método proposto e que também estão presentes no conjunto de referência ($ConceitosExtraidos_{Refer.}$) comparativamente à quantidade total de conceitos do conjunto de referência ($TodosConceitos_{Refer.}$):

$$Cobertura = \frac{ConceitosExtraidos_{Refer.}}{TodosConceitos_{Refer.}} \tag{6.1}$$

De modo similar à cobertura, também a precisão (*precision*), outra métrica da área de Recuperação de Informações, pode ser calculada no contexto de extração de conceitos. Aqui, precisão é definida como a razão entre a quantidade de conceitos extraídos que aparecem no conjunto de referência ($ConceitosExtraidos_{Refer.}$) e a quantidade total de conceitos extraídos ($TodosConceitosExtraidos$):

$$Precisão = \frac{ConceitosExtraidos_{Refer.}}{TodosConceitosExtraidos} \tag{6.2}$$

Na tabela 6.1 pode-se observar os resultados obtidos para cobertura da ontologia extraída. É importante notar que a métrica de precisão não demonstra um possível benefício do método que é o de oferecer ao ontologista novos conceitos que não faziam parte do conjunto de referência ou *golden standard*. Dentre eles estão conceitos fundamentais ao domínio, tais como *budget*, *assumption*, *change control*, *change request*. De fato, dentre os conceitos extraídos que não foram mencionados no conjunto de referência ($NovosConceitos$), 83% deles foram julgados pertinentes para compor a ontologia do domínio ($NovosConceitos_{Pertinentes}$).

Tabela 6.1: Métricas de Avaliação Quantitativa dos Conceitos Extraídos

$TodosConceitos_{Refer.}$	106
$ConceitosExtraidos_{Refer.}$	77
$TodosConceitosExtraidos$	279
$Cobertura$	73%
$Precisão$	28%
$NovosConceitos$	211
$NovosConceitos_{Pertinentes}$	176
% $NovosConceitos_{Pertinentes}$	83%

Do ponto de vista qualitativo, uma análise dos conceitos extraídos foi feita por 3 profissionais certificados na área de gestão de projetos que exerceram o papel de

ontologista e, por consenso, avaliaram cada conceito (classes e subclasses) automaticamente reconhecido e a eles oferecido. Para tanto, cada conceito foi apresentado ao grupo e, após uma breve discussão, o grupo atribuía por consenso um de 5 possíveis graus de relevância do conceito para o domínio de gestão de projetos: alta, boa, razoável, pouca ou nenhuma relevância. A tabela 6.2 sumariza o resultado dessa avaliação qualitativa;

Tabela 6.2: Avaliação qualitativa dos conceitos identificados por profissionais da área de Gestão de Projetos

Grau de Relevância	Qtde. de Conceitos	% de Conceitos
Alta	231	66,2%
Boa	75	21,5%
Razoável	36	10,3%
Pouca	4	1,1%
Nenhuma	3	0,9%

Como se vê, a avaliação qualitativa corrobora a avaliação quantitativa de que o método pode auxiliar o trabalho do ontologista na identificação dos conceitos de um domínio. Mais de 87% dos conceitos apresentados foram julgadas de alta ou boa relevância para o domínio da Gestão de Projetos. Em geral, foram julgados menos relevantes os conceitos mais genéricos que foram construídos para suportar a construção de subclasses, estas sim importantes. São exemplos dessa situação, os pares de classe e subclasse (`Chart`, `OrganizationChart`), (`Event`, `RiskEvent`), (`Need`, `BusinessNeed`), (`Request`, `ChangeRequest`), (`System`, `ChangeControlSystem`).

6.1.2 Avaliação da Extração e Criação de Relações

Como visto na sessão 5.5, várias abordagens foram estabelecidas e avaliadas para a extração de relações existentes entre os conceitos identificados.

As abordagens baseadas em padrões de relações verbais específicas, tais como estudadas nas sessões 5.5.1 e 5.5.2, apresentaram limitações de aspecto prático em sua implementação. Ainda que centenas de padrões verbais específicos tenham sido manualmente definidos para cobrir um grande número de expressões verbais de uso típico em textos não coloquiais, não se pode ter certeza de que eles sejam suficientes para identificar a maior parte das relações existentes em textos de qualquer domínio. Ainda assim, é importante notar que, quando utilizada contra o corpus do PMBOK, essas abordagens selecionaram do texto muitos fragmentos com expressões verbais que demonstravam relações consideradas relevantes para a ontologia de gerenciamento de projetos.

Foram então escolhidas para compor o método deste trabalho as abordagens apresentadas nas sessões 5.5.3 e 5.5.4 que focam em relações verbais e de composição & partes.

Dada a ausência de um "golden standard" com uma enumeração das relações fundamentais entre conceitos de gestão de projetos, a avaliação de performance da extração de relações proposta neste trabalho, é mais complexa. Assim, foi utilizado um método de avaliação qualitativa sobre a relevância das relações apresentadas ao ontologista envolvendo o mesmo painel de 3 profissionais que avaliou os conceitos discutidos na sessão 6.1.1. Os mesmos graus de relevância foram usados e os resultados obtidos são apresentados na tabela 6.3. A lista de relações avaliadas é apresentada no anexo A.6.

Tabela 6.3: Avaliação qualitativa das relações identificadas por profissionais da área de Gestão de Projetos

Grau de Relevância	Qtde. de Relações	% de Relações
Alta	48	29,4%
Boa	30	18,4%
Razoável	30	18,4%
Pouca	29	17,8%
Nenhuma	26	16,0%

Como se vê, pouco menos de 48% das relações foram julgadas de alta ou boa relevância para uma ontologia do domínio de gestão de projetos. Obviamente, esse resultado é muito dependente da relevância dos conceitos identificados. De forma geral, as relações menos relevantes foram as que se relacionavam a conceitos menos importantes, tais como *Design* ou *WorkAuthorizationSystem*.

Uma crítica a esta avaliação é a ausência de um mecanismo que verificasse as relações que faltam à ontologia, isto é, aquelas que não foram identificadas no texto, seja por falha do método, seja por elas simplesmente não serem mencionadas no corpus do domínio.

7 Considerações Finais

Este trabalho apresentou um método semi-automático de criação de ontologias de domínio, baseado na análise de textos que expressem os principais conceitos desse domínio de conhecimento e as relações que possam existir entre eles. O objetivo é apoiar o trabalho de um especialista humano (o ontologista) e assim contribuir para que a construção de ontologias seja mais rápida e menos onerosa, potencializando a disseminação do seu uso e aplicação.

Quando se observa que a quantidade de domínios para os quais ainda não existem ontologias especificadas e compartilhadas é vasta, fica evidente a importância do estabelecimento de métodos que suportem a construção de ontologias. Assim, o processo de construção se torna mais ágil, tanto em momento de aquisição e construção inicial, quanto em momentos de atualização e ajuste da ontologia.

Como detalhado na sessão 6, este método apresentou um desempenho satisfatório, dado a cobertura de conceitos alcançada, e a avaliação de especialistas do domínio para a relevância dos novos conceitos descobertos e as relações estabelecidas. Apesar das dificuldades inerentes ao tema, a implementação do método proposto conseguiu oferecer ao ontologista um conjunto de conceitos e relações de boa relevância para o domínio, na opinião daqueles especialistas. Além disso, as técnicas e as ferramentas utilizadas nesta implementação oferecem grande potencial de refinamento e expansão.

Ontologias são especificações complexas. Várias diferentes ontologias podem ser concebidas e existir para o mesmo domínio e ainda assim serem consideradas corretas ou apropriadas para representar o conhecimento nele contido. Essa característica torna pouco trivial a comparação entre ontologias e a análise de sua qualidade ou adequação a um determinado propósito, o que constitui um desafio atual da área.

Apesar dessas dificuldades inerentes ao tema, este método de extração de ontologias pode estabelecer uma base de trabalho importante para o especialista do domínio. Ainda que ele precise validar e completar a especificação da ontologia, estima-se que o suporte oferecido por este método reduza consideravelmente seu esforço com a aquisição de conceitos e relações. Esse tipo de apoio é especialmente valioso quando um glossário de definições a respeito do domínio não existir ou não estiver disponível.

7.1 Contribuições

Este trabalho fez uso de técnicas de processamento de linguagem natural para a idealização e implementação de um método de apoio à construção de ontologias.

As principais contribuições do trabalho estão associadas ao uso e combinação de diversos processos e mecanismos de tratamento de textos, recuperação de informação e avaliação estatística de importância ou relevância de ítens descobertos.

Uma nova abordagem de seleção de termos relevantes em um corpus através da comparação com outro corpus foi estabelecida. Uma estratégia de construção de conceitos e taxonomias foi criada e avaliada. Várias abordagens de reconhecimento de relações, algumas já abordadas na literatura e outras criadas durante o curso deste trabalho, foram implementadas e avaliadas.

Além disso, a utilização aqui de ferramentas abertas e de ampla utilização pela comunidade científica da área, tais como o GATE, e a linguagem Java, de uso padrão no laboratório de Engenharia de Conhecimento da Escola Politécnica, poderão proporcionar novas oportunidades de extensão deste trabalho pelos pesquisadores do laboratório.

7.2 Trabalhos Futuros

Dentre as possibilidades de trabalhos futuros já visualizados, está a expansão do método para a construção de ontologias no idioma português, usando corpus de domínio escritos nesse idioma.

Outra possibilidade de interesse é o estudo de mecanismos de aprendizagem para a identificação dos padrões lingüísticos mais importantes para o reconhecimento de relações, melhorando assim a qualidade do apoio oferecido ao ontologista.

Uma continuação interessante para este trabalho seria o estabelecimento de uma nova abordagem de avaliação de qualidade das ontologias criadas, abordagem essa que se basearia no uso das ontologias em aplicações variadas. Essa análise reforçaria o aspecto de aplicabilidade das ontologias criadas, estimulando assim critérios que favoreçam a efetiva disseminação de seu uso. Outra evolução possível na abordagem de avaliação seria aumentar o número de participantes do painel de avaliação qualitativa da importância de cada um dos conceitos e relações encontradas. Simultaneamente, se cada opinião for coletada isoladamente, será possível analisar a variabilidade na avaliação dos respondentes e assim determinar de modo mais eficaz as forças e fraquezas da abordagem: conceitos e relações cuja nível de importância tiver pequena variabilidade no painel de avaliadores serão mais importantes para a definição da eficácia do método e de sua implementação.

Anexo A – Resultados

A.1 Prevalência de Termos no Corpus do PMBOK

A comparação de freqüência dos 70 termos mais comuns do corpus do PMBOK com a freqüência correspondente no corpus Brown, tal como discutida na sessão 5.4.1 é mostrada a seguir.

Tabela A.1: Os 70 termos mais comuns no PMBOK

Termo	Freq. Absoluta Corpus PMBOK	Freq. Absoluta Corpus Brown	Prevalência no PMBOK	χ^2
project	1192	93	349,69	30.031,4
scope	164	27	165,72	3.796,3
management	367	91	110,03	7.881,5
schedule	134	36	101,55	2.825,7
procurement	78	21	101,33	1.643,9
risk	182	54	91,95	3.747,3
estimates	73	24	82,98	1.463,1
processes	160	57	76,58	3.135,7
team	171	84	55,54	3.017,6
projects	130	68	52,16	2.240,6
tools	63	34	50,55	1.072,7
documents	34	19	48,82	571,0
communications	49	28	47,74	815,5
assumptions	38	23	45,08	617,5
contract	98	60	44,56	1.585,0
product	138	87	43,28	2.204,2
phases	34	24	38,65	516,0
quality	156	114	37,33	2.329,7
planning	174	129	36,80	2.580,7
cycle	28	24	31,83	385,9
definition	44	38	31,59	604,0

Continua na próxima página

Tabela A.1 – continuação da página anterior

Termo	Freq. Absoluta Corpus PMBOK	Freq. Absoluta Corpus Brown	Prevalência no PMBOK	χ^2
section	211	189	30,46	2.841,6
skills	40	37	29,49	529,2
performance	128	122	28,62	1.665,6
phase	71	72	26,90	891,7
cost	222	229	26,45	2.761,4
detail	68	72	25,77	832,6
activity	109	116	25,64	1.330,7
techniques	92	99	25,35	1.115,7
application	63	68	25,28	762,6
objectives	36	39	25,18	434,8
organization	115	127	24,70	1.373,0
plan	178	205	23,69	2.071,2
responsibilities	21	25	22,92	239,2
customer	22	27	22,23	245,8
requirements	66	83	21,69	725,9
description	42	54	21,22	455,2
network	23	30	20,92	246,9
dates	22	30	20,01	229,2
resources	52	72	19,70	536,1
characteristics	37	52	19,41	377,6
activities	81	115	19,22	820,8
input	14	20	19,10	141,2
control	156	223	19,09	1.573,6
organizations	42	61	18,78	418,9
measurement	23	34	18,46	226,5
categories	16	24	18,19	155,9
process	129	196	17,96	1.246,1
relationships	24	38	17,23	224,9
identification	27	43	17,13	251,9
individuals	45	73	16,82	414,2
chapter	45	74	16,59	410,0
chart	13	22	16,12	115,9
estimate	23	39	16,09	204,8
changes	75	131	15,62	652,8
expert	17	30	15,46	146,8
completion	32	57	15,32	274,3

Continua na próxima página

Tabela A.1 – continuação da página anterior

Termo	Freq. Absoluta Corpus PMBOK	Freq. Absoluta Corpus Brown	Prevalência no PMBOK	χ^2
evaluation	17	31	14,96	143,0
manufacturing	13	24	14,78	108,3
information	142	269	14,40	1.159,5
probability	19	36	14,40	155,1
expectations	12	23	14,23	97,0
analysis	56	108	14,15	450,6
results	77	149	14,10	617,9
structure	47	91	14,09	377,0
items	36	72	13,64	281,2
engineering	23	46	13,64	179,6
adjustments	10	20	13,64	78,1
proposals	14	29	13,17	106,2
design	55	114	13,16	417,0
item	26	54	13,14	196,8
approaches	12	25	13,10	90,6

A.2 Sintagmas Nominais Reconhecidos

A tabela A.2 lista os sintagmas nominais mais importantes selecionados pela abordagem mencionada na sessão 5.4.2 quando ela é usada para tratamento do corpus do PMBOK.

Tabela A.2: Os 100 sintagmas nominais que ocorrem com maior freqüência no corpus de domínio

Sintagma Nominal	Freq. Ocorrência	Sintagma Nominal	Freq. Ocorrência
project	488	project objective	20
process	146	project scope	19
organization	129	procedure	19
cost	123	project schedule	19
activity	90	method	19
information	87	change request	18
product	80	design	18
technique	78	performance report	18
work	75	planning process	18
change	67	analysis	18
output	66	project plan execution	18
risk	65	control	18
result	65	documentation	18
input	65	work result	18
project plan	56	knowledge area	18
project management team	56	quality	17
tool	55	quality control	17
resource	55	customer	17
application area	52	area	16
phase	50	scope change control	16
system	48	resource planning	15
contract	44	scope change	15
need	44	project network diagram	15
stakeholder	42	project stakeholder	15
constraint	39	outcome	15
estimate	39	schedule development	15
assumption	37	procurement planning	14
document	36	decision	14
source	34	interface	14
project team	33	cost control	14
project management	33	statement	14
risk event	32	quality assurance	14
knowledge	31	activity list	14
management	29	schedule control	14
scope statement	29	corrective action	14
cost estimate	28	agreement	14
action	27	historical information	14
team	27	scope definition	14
approach	27	project activity	14
relationship	27	role	13

Continua na próxima página

Tabela A.2 – continuação da página anterior

Sintagma Nominal	Freq. Ocorrência	Sintagma Nominal	Freq. Ocorrência
performance	26	team development	13
project phase	26	quality planning	13
project manager	26	contract administration	13
proposal	25	procurement document	13
scope	25	solicitation planning	13
product description	25	project plan development	13
requirement	25	change control system	13
planning	24	staff	13
work breakdown structure	23	duration	13
deliverable	23	contract closeout	13
structure	23	technology	13
project performance	23	project deliverable	13
project life cycle	22	budget	13
completion	22	project cost	12
description	21	procurement	12
variance	21	objective	12
project management process	21	procurement item	12
schedule	20	process interaction	12
plan	20	communication planning	12
risk identification	20	resource requirement	12

A.3 Relações Específicas encontradas

Abaixo são enumaradas algumas das relações encontradas ao se utilizar a abordagem de reconhecimento de relações específicas discutida na sessão 5.5.2. Em parenteses menciona-se a regra ou padrão que disparou a seleção da frase). Estes resultados são intermediários e ilustrativos apenas.

1. **Project management** is a relatively **young profession**, and while there is substantial commonality around what is done, there is relatively little commonality in the terms used. (Rule=DEF1, relation=IsA)
2. a **project** is a **temporary endeavor** undertaken to create a unique product or service. (Rule=DEF1, relation=IsA)
3. **Temporary means** that every project has a definite beginning and a definite end. (Rule= CARACT1 , relation=Defines)
4. **Unique means** that the product or service is different in some distinguishing way from all similar products or services. (Rule=CARACT1 , relation=Defines)
5. **Projects involve doing something** which has not been done before and which is, therefore, **unique**. (Rule=PARTE1, relation=IsComposedOf)
6. **Project management** is the **application of knowledge, skills, tools, and techniques** to project activities in order to meet or exceed stakeholder needs and expectations from a project.(Rule=DEF1 , relation=IsA)
7. **project management** is a **subset of program management**. (Rule=DEF1, relation=IsA)
8. most **project life cycles** generally **involves** some form of **technology transfer** or handoff such as requirements to design, construction to operations, or design to manufacturing. (Rule=PARTE1, relation=IsComposedOf)
9. The **classic functional organization** is a **hierarchy** where each employee has one clear superior. (Rule=DEF1, relation=IsA)
10. **Matrix organizations** are a blend of **functional and projectized characteristics**. (Rule= DEF1, relation=IsA)
11. **General management** is a **broad subject** dealing with every aspect of managing an ongoing enterprise. (Rule=DEF1, relation=IsA)
12. **Negotiating involves conferring** with others in order to come to terms or reach an agreement. (Rule=PARTE1, relation=IsComposedOf)

13. **Communicating involves** the **exchange of information**. (Rule=PARTE1, relation=IsComposedOf)

14. **Building codes are** an example of **regulations**. (Rule=DEF1, relation=IsA)

15. the **design document defines** the **product description** for the ensuing implementation phase (Rule=CARACT1, relation=Describes)

16. **Controlling** also **includes taking preventive action** in anticipation of possible problems. (Rule=PARTE1, relation=IsComposedOf)

17. **Project Integration Management** includes the **processes** required to ensure that the various elements of the project are properly coordinated. (Rule=PARTE1, relation= IsComposedOf)

18. a **predefined budget** is **a constraint** that is highly likely to limit the team's options regarding scope, staffing, and schedule. (Rule=DEF1, relation=IsA)

19. **Assumptions** generally involve a **degree of risk**. (Rule=PARTE1, relation= IsComposedOf)

20. The **project plan** is a **formal, approved document** used to manage and control project execution. (Rule=DEF1, relation=IsA)

21. The **project plan** is a **document or collection of documents** that should be expected to change over time as more information becomes available about the project. (Rule=DEF1, relation=IsA)

22. A **work authorization system** is a **formal procedure** for sanctioning project work to ensure that work is done at the right time and in the proper sequence. (Rule=DEF1, relation=IsA)

23. **Work results** are the **outcomes of the activities** performed to accomplish the project. (Rule=DEF1, relation=IsA)

24. A **change control system** is a collection of **formal, documented procedures** that defines the steps by which official project documents may be changed. (Rule=DEF1, relation=IsA)

25. **configuration management** is a **subset of the change control system** and is used to ensure that the description of the project product is correct and complete. (Rule=DEF1, relation=IsA)

26. A **project charter** is a **document** that formally recognizes the existence of a project. (Rule=DEF1, relation=IsA)

27. **Scope planning** is the **process** of developing a written scope statement as the basis for future project decisions including, in particular, the criteria used to

determine if the project or phase has been completed successfully. (Rule=DEF1, relation=IsA)

28. **Inspection** includes **activities** such as measuring, examining, and testing undertaken to determine whether results conform to requirements. (Rule=PARTE1, relation= IsComposedOf)

29. A **scope change control system** defines the **procedures** by which the project scope may be changed. (Rule=CARACT1, relation=Describes)

30. **Decomposition** involves **subdividing project elements** into smaller, more manageable components in order to provide better management control. (Rule=PARTE1, relation=IsComposedOf)

A.4 Relações Verbais Reconhecidas

A tabela A.3 mostra algumas das relações encontradas ao se utilizar a abordagem sugerida na sessão 5.5.3 contra o corpus do PMBOK.

Tabela A.3: 35 exemplos de relações identificadas através de estruturas verbais
envolvendo sintagmas nominais do domínio

Sintagma Nominal 1	**Relação Verbal**	**Sintagma Nominal 2**
Pmbok	include	Knowledge
Project Phase	isMarkedBy	Completion
Management	encompass	Planning
Project Phase	isMarkedBy	Review
Project	isComposedOf	Process
ProductOriented Process	isDefinedBy	Project Life Cycle
Variance	isFedInto	Control Process
Budget	is	Constraint
Project Planning Methodology	is	Approach
Project Plan	is	Document
Performance Measurement Baseline	represent	Management Control
Project Plan Execution	is	Process
Work Authorization System	is	Procedure
Work Result	is	Outcome
Project Plan	provide	Baseline
Control	isOftenPerformedBy	Quality Control Department
Change Control System	isACollectionOf	Procedure
Project Scope	isMeasuredAgainst	Plan
Product Scope	isMeasuredAgainst	Requirement
Initiation	is	Process
Project Charter	is	Document
Scope Planning	is	Process
Scope Verification	is	Process
Scope Verification	differFrom	Quality Control
Activity Duration Estimate	is	Assessment
Project Calendar	affect	Resource
Resource Calendar	affect	Resource

Continua na próxima página

Tabela A.3 – continuação da página anterior

Sintagma Nominal 1	**Relação Verbal**	**Sintagma Nominal 2**
Cost Estimate	is	Assessment
Cost Baseline	is	Budget
Communication Management Plan	is	Document
Risk Response Development	isSometimesCalled	Response Planning
Procurement Planning	is	Process
Contract	is	Relationship
Contract Administration	is	Process
Procurement Audit	is	Review

A.5 Relações de Composição & Partes

Tal como discutido na sessão 5.5.4, a tabela A.4 mostra algumas das relações desse tipo encontradas ao se utilizar a abordagem sugerida contra o Corpus do PMBOK.

Tabela A.4: 25 exemplos de relações identificadas através de estruturas que indicam composição

Sintagma Nominal 1	**Relação Verbal**	**Sintagma Nominal 2**
Organization	has	Resource
Project	has	Objective
Project	has	Scope
Project	has	Phase
Project	has	Scope
Project	has	Duration
Process	has	Result
Resource	has	Cost
Project	has	Product
Activity	has	Outcome
Project Product	has	Description
Product Scope	has	Completion
Project Scope	has	Completion
Project	has	Scope Baseline
Activity	has	Duration
Project Plan	has	Component
Project	has	Cost
Project Product	has	Performance
Procurement Item	has	Cost
Project Stakeholder	has	Need
Project	has	Requirement
Project Participant	has	Expertise
System	has	Performance
Schedule Simulation	has	Result
Procurement Process	has	Review

A.6 Lista de Relações Identificadas no PMBOK

Como visto na sessão 5.7.3, a utilização do método sugerido neste trabalho contra o corpus do PMBOK revelou as relações da tabela A.5 para fazer parte da lista de relações apresentadas ao ontologista.

Tabela A.5: Relações identificadas e apresentadas ao ontologista

Conceito 1	**Relação**	**Conceito 2**
Activity	has	Completion
Activity	has	Duration
Activity	has	Outcome
ActivityDurationEstimate	is	Assessment
Budget	is	Constraint
ChangeControlSystem	isACollectionOf	Procedure
Checklist	is	Tool
CommunicationManagement_ Plan	is	Document
Contract	is	Relationship
ContractAdministration	is	Process
Control	isOftenPerformedBy	QualityControlDepartment
CostEstimate	is	Assessment
Initiation	is	Process
Management	encompass	Planning
PhaseDeliverable	has	Approval
Pmbok	include	Knowledge
ProcurementAudit	is	Review
ProcurementPlanning	is	Process
Product	has	Description
Product	has	Scope
ProductScope	isMeasuredAgainst	Requirement
Project	has	Duration
Project	has	Management
Project	has	Objective
Project	has	Phase
Project	has	Scope
Project	has	StaffingRequirement
ProjectCharter	is	Document
ProjectLifeCycle	has	Phase
ProjectParticipant	has	Expertise
ProjectPhase	isMarkedBy	Completion
ProjectPhase	isGenerallyMarkedBy	Review
ProjectPlan	contain	Baseline
ProjectPlan	is	Document

Continua na próxima página

Tabela A.5 – continuação da página anterior

Conceito 1	Relação	Conceito 2
ProjectPlanExecution	is	Process
ProjectPlanningMethodology	is	Approach
ProjectProduct	has	Quality
ProjectScope	has	Acceptance
ProjectScope	isMeasuredAgainst	Plan
ProjectStakeholder	has	Need
QualityAudit	is	Review
Resource	has	Cost
Scope	has	Description
ScopePlanning	is	Process
ScopeVerification	is	Process
Stakeholder	has	Need
WorkAuthorizationSystem	is	Procedure
WorkResult	has	Acceptance
Activity	has	Result
Contract	has	Requirement
ControlProcess	has	Output
Cost	has	Assessment
CostBaseline	is	Budget
Organization	has	Resource
PlanningProcess	has	Output
Process	has	Output
Process	has	Result
ProcurementDocument	has	Requirement
ProcurementItem	has	Cost
Project	has	Acceptance
Project	has	Completion
Project	has	Cost
Project	has	Product
Project	isComposedOf	Process
Project	has	Requirement
Project	has	ScopeBaseline
Project	has	Work
Project	has	WorkResult
ProjectManagementProcess	has	Output
ProjectPlan	has	Development
ProjectPlan	provide	Baseline
ProjectProduct	has	Description
ResourcePlanningProcess	has	Output
Result	has	Assessment
RiskIdentification	has	Result
RiskResponseDevelopment	isSometimesCalled	ResponsePlanning

Continua na próxima página

Tabela A.5 – continuação da página anterior

Conceito 1	Relação	Conceito 2
ScheduleVariation	require	Action
Variance	isFedInto	ControlProcess
Checklist	isTypicallyOrganizedBy	Source
CorrectiveAction	is	Output
EvaluationCriterion	has	Application
Inspection	isVariouslyCalled	Review
Plan	reflect	Resource
ProductDesignProject	has	Outcome
ProductionPhase	has	Cost
ProductScope	has	Completion
Project	has	Need
ProjectCalendar	affect	Resource
ProjectCommunication_ Management	include	Process
ProjectCostManagement	isPrimarilyConcerned_ With	Cost
ProjectCostManagement	include	Process
ProjectElement	has	Description
ProjectIntegrationManagement	include	Process
ProjectManagementApproach	has	Description
ProjectManagementProcess	has	Application
ProjectPlan	has	Component
ProjectProcurementManagement	include	Process
ProjectProduct	has	Performance
ProjectQualityManagement	include	Process
ProjectResult	include	ProductResult
ProjectRiskManagement	include	Process
ProjectScope	has	Acceptance
ProjectScopeManagement	include	Process
ProjectTimeManagement	include	Process
ResearchProject	has	Product
ResourceCalendar	affect	Resource
ScheduleSimulation	has	Result
WorkResult	is	Outcome
Agreement	has	Review
CommunicationPlanning	isOftentTightlyLinked_ With	Planning
Component	has	Delivery
Decision	has	Result
Estimate	have	Expertise
Information	has	Description
Inspection	include	Activity

Continua na próxima página

Tabela A.5 – continuação da página anterior

Conceito 1	Relação	Conceito 2
Organization	have	Checklist
Organization	has	System
OrganizationalPlanning	isOftenTightlyLinked_With	CommunicationPlanning
PerformanceMeasurement_Baseline	represent	ManagementControl
ProcurementProcess	has	Review
ProductOrientedProcess	isTypicallyDefinedBy	ProjectLifeCycle
Project	involve	PerformingOrganization
Project	consistOf	Product
ProjectCommunication_Management	is	Application
ProjectIssue	has	Product
ProjectManagement	is	Application
ProjectManagement	has	Quality
Proposal	has	Result
ResourcePlanningProcess	is	Description
Rework	is	Action
ScopeVerification	differFrom	QualityControl
Source	has	Description
Source	include	ProjectContext
System	has	Performance
TeamBuildingActivity	include	Management
Variance	require	Action
WorkBreakdownStructure	is	Input
ApplicationArea	isCategoryOf	Project
Approach	isOftenCalled	ProjectManagement_Methodology
Contract	become	Input
ContractAdministration	include	Application
Cost	is	Cost
Customer	has	OperatingCost
PerformanceMeasurement_Baseline	is	Input
Process	isIn	Control
ProcessGroup	isLinkedBy	Result
ProcurementAudit	has	Objective
ProcurementItem	require	Statement
Project	isCompletedWithin	Budget
Project	is	Organization
Project	is	Result
Project	is	ScheduleSimulation

Continua na próxima página

Tabela A.5 – continuação da página anterior

Conceito 1	**Relação**	**Conceito 2**
Project	is	ServiceProject
Project	has	Relationship
ProjectPlanDevelopment	usee	Output
ProjectSchedule	includeAt	Documentation
ProjectTeamMember	lack	Management
QualityAudit	has	Objective
QualityManagementPlan	provide	Input
RiskIdentification	isNot	Event
TelecommunicationsSystem	has	Design
WeightingSystem	is	Method
WorkAuthorizationSystem	has	Design

A.7 Resultado da Representação da Ontologia em OWL

A seguir listam-se as 200 primeiras linhas da representação em linguagem OWL da ontologia criada para o domínio de gestão de projetos. No total a representação contém 1930 linhas de código.

```
<?xml version="1.0"?> <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.owl-ontologies.com/Ontology1176060652.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.owl-ontologies.com/Ontology1176060652.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="ProjectProcurementManagement">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="include"/>
        </owl:onProperty>
        <owl:someValuesFrom>
          <owl:Class rdf:ID="Process"/>
        </owl:someValuesFrom>
        <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Project Procurement Management includes the processes required to acquire
         goods and services from outside the performing organization.</rdfs:comment>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Management"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Organization">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="have"/>
        </owl:onProperty>
        <owl:someValuesFrom>
          <owl:Class rdf:ID="Checklist"/>
        </owl:someValuesFrom>
        <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Many organizations have standardized checklists available to ensure
         consistency in frequently performed activities.</rdfs:comment>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="TechniqueOutput">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Output"/>
    </rdfs:subClassOf>
  </owl:Class>
```

```
<owl:Class rdf:ID="ActivityDefinition"/>
<owl:Class rdf:ID="ProductDevelopment">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Development"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ManagementTechnique">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Technique"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ProjectExecution"/>
<owl:Class rdf:ID="BarChart">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Chart"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ManagementControl">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Control"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ProjectProduct">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Product"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ProjectCompletion">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Completion"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="OrganizationalPlanning">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="isOftenTightlyLinkedWith"/>
      </owl:onProperty>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Organizational planning is often tightly linked with communications
       planning (described in Section 10.1) since the project's organizational
       structure will have a major effect on the project's communications
       requirements.</rdfs:comment>
      <owl:someValuesFrom>
        <owl:Class rdf:ID="CommunicationPlanning"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Planning"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Risk"/>
<owl:Class rdf:ID="Performance">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
```

```
        <owl:Class rdf:ID="System"/>
      </owl:someValuesFrom>
      <owl:someValuesFrom rdf:resource="#ProjectProduct"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="isPartOf"/>
      </owl:onProperty>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Simulation uses a representation or model of a system to analyze the
      behavior or performance of the system.</rdfs:comment>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >In many application areas predicting and analyzing the prospective
      financial performance of the project product is done outside the
      project.</rdfs:comment>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ProjectScopeManagement">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#include"/>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#Process"/>
      </owl:someValuesFrom>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Project Scope Management includes the processes required to ensure
      that the project includes all the work required, and only the work
      required, to complete the project successfully [1].</rdfs:comment>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Management"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="QualityControl">
  <rdfs:subClassOf rdf:resource="#Control"/>
</owl:Class>
<owl:Class rdf:ID="Input"/>
<owl:Class rdf:ID="ControlChart">
  <rdfs:subClassOf rdf:resource="#Chart"/>
</owl:Class>
<owl:Class rdf:ID="QualityStandard"/>
<owl:Class rdf:ID="PerformanceReporting"/>
<owl:Class rdf:ID="ScheduleChangeControl">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="ChangeControl"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Stakeholder"/>
<owl:Class rdf:ID="ProjectPhase">
  <rdfs:subClassOf>
    <owl:Restriction>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >The conclusion of a project phase is generally marked by a review
      of both key deliverables and project performance in order to (a)
      determine if the project should continue into its next phase and (b)
      detect and correct errors cost effectively.</rdfs:comment>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
      >Each project phase is marked by completion of one or more
      deliverables.</rdfs:comment>
      <owl:someValuesFrom>
        <owl:Class rdf:ID="Review"/>
      </owl:someValuesFrom>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#Completion"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="isGenerallyMarkedBy"/>
      </owl:onProperty>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="isMarkedBy"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Phase"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Subproject"/>
<owl:Class rdf:ID="Expertise"/>
<owl:Class rdf:ID="DiscretionaryDependency">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Dependency"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ProjectStaff">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Staff"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="MandatoryDependency">
  <rdfs:subClassOf rdf:resource="#Dependency"/>
</owl:Class>
<owl:Class rdf:ID="SolicitationPlanning">
  <rdfs:subClassOf rdf:resource="#Planning"/>
</owl:Class>
<owl:Class rdf:ID="Analysis"/>
<owl:Class rdf:ID="ProjectElement"/>
<owl:Class rdf:ID="ProjectIntegrationManagement">
  <rdfs:subClassOf>
    <owl:Restriction>
      <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Project Integration Management includes the processes required to
      ensure that the various elements of the project are properly
      coordinated.</rdfs:comment>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#Process"/>
      </owl:someValuesFrom>
      <owl:onProperty rdf:resource="#include"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Management"/>
  </rdfs:subClassOf>
</owl:Class>
```

Anexo B – Especificações em JAPE e Java Utilizadas

B.1 Padrões para Reconhecimento de Sintagmas Nominais

A gramática JAPE especificada abaixo é utilizada para reconhecimento de sintagmas nominais e criação das anotações correspondentes (usando ações expressas em Java).

```
//*********************************************************************
// Identifica Sintagmas Nominais simples (composto só de substantivos)
// Usa os POS tags (obtidos com o Hepple POS tagger).
// Cria anotações do tipo SN1 com as features seguintes:
//  * string - é a cadeia de caracteres anotada
//*********************************************************************
Phase:  SNs1
Input: Token
Options: control = appelt

//*********************************************************************
Macro: SUBSTANTIVO
(
  {Token.category == NN, Token.kind == word} |
  {Token.category == NNS, Token.kind == word}|
  {Token.category == NNP, Token.kind == word} |
  {Token.category == NNPS, Token.kind == word} |
  {Token.category == NP, Token.kind == word} |
  {Token.category == NPS, Token.kind == word} |
  {Token.category == CD, Token.kind == word}
)

Macro: ARTIGODEF
(
    {Token.string == "the"} | {Token.string == "The"} | {Token.string == "THE"}
)

Macro: ARTIGOINDEF
(
  {Token.string == "a"} | {Token.string == "an"} |
  {Token.string == "A"} | {Token.string == "An"}
)
```

```
Macro: ADJETIVO
(
  {Token.category == JJ, Token.kind == word} |
  {Token.category == JJR, Token.kind == word} |
  {Token.category == JJS, Token.kind == word}
)

Macro: POSSESSIVO
(
  {Token.category == POS, Token.kind == word}
)

Macro: ADVERBIO
(
  {Token.category == RB, Token.kind == word} |
  {Token.category == RBR, Token.kind == word} |
  {Token.category == RBS, Token.kind == word}
)

//*************************************************************
Rule: SNs1
Priority: 50

(
  (SUBSTANTIVO)* (SUBSTANTIVO)
):term

-->

{
    try{
        AnnotationSet termAnnot = (AnnotationSet)bindings.get("term");

        TreeMap termosMap = new TreeMap();
        Vector vTermos = new Vector();
        String termoComposto = "";

        //cria um ordered Map com os termos, ordenado pelas posições no texto
        Iterator termosIt = termAnnot.iterator();
        Annotation umTermo;
        FeatureMap features;
        while (termosIt.hasNext()){
            umTermo = (Annotation)termosIt.next();
            features = umTermo.getFeatures();
            termosMap.put(umTermo.getStartNode().getOffset(), features.get("root"));
        }

        //transforma o treemap no termo composto (pode ser um ou mais substantivos)
        Iterator mapIt = termosMap.keySet().iterator();
        Long key;
        while (mapIt.hasNext()){
            key = (Long)mapIt.next();
            termoComposto = termoComposto + " " + (String)termosMap.get(key);
            vTermos.add((String)termosMap.get(key));
            }
```

```
        termoComposto = termoComposto.trim();

        //cria a anotação
        FeatureMap term_features = Factory.newFeatureMap();
        term_features.put("string", termoComposto.toLowerCase());

        outputAS.add(termAnnot.firstNode().getOffset(),
            termAnnot.lastNode().getOffset(),
            "SN1",
            term_features);

    }catch(InvalidOffsetException ioe){
            ioe.printStackTrace();
    }
}
```

B.2 Padrões para Reconhecimento de Sintagmas Verbais

A gramática JAPE especificada abaixo é o resultado da modificação da gramática original utilizada no módulo VP Chunker do GATE. Apenas as macros e as 10 primeiras regras são listadas.

```
// Finite Verb Groups

Phase: FiniteVerbGroup
Input: Token
Options: control = appelt

Macro: MODALS (
 {Token.string == "can"} | {Token.string == "could"} | {Token.string == "may"} |
 {Token.string == "might"} | {Token.string == "must"} |
 ({Token.string == "ought"} {Token.string == "to"}) |
 {Token.string == "should"} | {Token.string == "would"}
)

Macro: TO ( ({Token.string == "to"})? )

Macro: ADVS (
 ({Token.category == RB})? | ({Token.category == RB} {Token.category == RB}) |
 ({Token.category == RB} {Token.category == RB} {Token.category == RB})
)

// Negative Verb Groups (i.e. 'not' and other adverbials)

Macro: NEGATION(
 ({Token.category == RB, Token.string == "not"}) |
 ({Token.category == RB, Token.string == "n't"})
)

////////////////////////////////////////////////////////////////
Rule: FVGSimPrePas
// Simple Present Pasive: is eaten or are eaten
// (is | are) VBN
 ( ({Token.string == "is" } | { Token.string == "are" }) (NEGATION)
 (ADVS) {Token.category == "VBN"}
):annotate -->
:annotate.VG = {type = "FVG",
  tense = "SimPre",
  voice = "passive",
  neg = "yes"}

Rule: FVGPreConAct
// Present Continuous Active: is eating or are eating
// (is | are) VBG
(
 ({Token.string == "is"} | {Token.string == "are"})
 (NEGATION) (ADVS)
 {Token.category == "VBG"}
```

```
):annotate -->
:annotate.VG  = { type = "FVG",  tense = "PreCon",
  voice = "active",  neg = "yes" }

Rule: FVGPreConPas
// Present Continuous Passive: is being  eaten or are being  eaten
// (is | are) being  VBN
(
 ({Token.string == "is"} | {Token.string == "are"})
 (NEGATION) (ADVS)
 {Token.string == "being"}
 {Token.string == "been"}
 {Token.category == "VBN"}
):annotate -->
:annotate.VG = { type = "FVG",  tense = "PreCon",
  voice = "active",  neg = "yes" }

Rule: FVGPrePerAct
// Present Perfect Active: has eaten
// (has | have) VBN
 ( ({Token.string == "has"} | {Token.string == "have"}) (NEGATION)
 (ADVS) {Token.category == VBN}
):annotate -->
:annotate.VG = { type = "FVG", tense = "PrePer",
  voice = "active", neg = "yes" }

Rule: FVGPrePerPas
// Present Perfect Active: has been eaten
// (has | have) been VBN
(
 ({Token.string == "has"} | {Token.string == "have"}) (NEGATION)
 (ADVS) {Token.string == "been"}
 {Token.category == VBN}
):annotate -->
:annotate.VG = { type = "FVG", tense = "PrePer",
  voice = "passive", neg = "yes" }

Rule: FVGPrePerConAct
// Present Perfect Continuous Active: has been eating
// (has | have) been VBG
(
 ({Token.string == "has"} | {Token.string == "have"})
 (NEGATION) (ADVS) {Token.string == "been"}
 {Token.category == VBG}
):annotate -->
:annotate.VG = { type = "FVG", tense = "PrePerCon",
  voice = "active", neg = "yes" }

Rule: FVGSimPasPas
// Simple Past Pasive: was eaten
// (was | were) VBN
(
 ({Token.string == "was"} | {Token.string == "were"}) (NEGATION)
 (ADVS) {Token.category == VBN}
):annotate -->
:annotate.VG = { type = "FVG", tense = "SimPas",
  voice = "passive", neg = "yes"}
```

```
Rule: FVGPasConAct
// Past Continuous Active: was eating
// (was | were) VBG
(
 ({Token.string == "was"} | {Token.string == "were"}) (NEGATION)
 (ADVS) {Token.category == VBG}
):annotate -->
:annotate.VG = { type = "FVG", tense = "PasCon",
  voice = "active", neg = "yes"}

Rule: FVGPasConPas
// Past Continuous Passive: was eating
// (was | were) being VBN
(
 ({Token.string == "was"} | {Token.string == "were"})
 (NEGATION) (ADVS) {Token.string == "being"}
 {Token.category == VBN}
):annotate -->
:annotate.VG = { type = "FVG", tense = "PasCon",
  voice = "passive", neg = "yes"}

Rule: FVGPasPerAct
// Past Perfect Active: had eaten
//  had VBN
(
 {Token.string == "had"} (NEGATION) (ADVS) {Token.category == VBN}
):annotate -->
:annotate.VG = { type = "FVG", tense = "PasPer",
  voice = "active", neg = "yes"}
```

B.3 Implementação da Construção da Taxonomia de Conceitos

Como analisada na sessão 5.7.2.3, a implementação da construção da taxonomia de conceitos da ontologia é baseada em Java. Como funcionalidade adicional, este programa usa a base de conhecimentos sobre conceitos e relações aceitas e faz sua representação em linguagem OWL.

Abaixo é mostrado um fragmento dessa implementação.

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.*;

import au.com.bytecode.opencsv.CSVReader;
import au.com.bytecode.opencsv.CSVWriter;

class Counter {
      int i = 1;
      String interesse = "N";
      String snClasse = "";
      String snInvertido = "";
      public String toString() {
        return Integer.toString(i) + ", " + interesse ;
      }
}

public class TrataSNs {

    private static final String Substantivos_FILE="SubstantivosEscolhidos.csv";
    private static final String SNs_FILE="SNsPotenciais.csv";
    private static final String Relacoes_FILE="GATE_Relacoes.txt";

    public static void main(String[] args) throws IOException {

        HashMap hmSubst = new HashMap();

        CSVReader reader = new CSVReader(new FileReader(Substantivos_FILE));
        String [] nextLine;
        //System.out.println("Substantivos de interesse lidos:");
        while ((nextLine = reader.readNext()) != null) {
            // se o substantivo está marcado como de interesse
            if (nextLine[1].equals("S")) {
                String subst = new String(nextLine[0]);
                Counter substCt = new Counter();
                //substCt.interesse = nextLine[1];
                //substCt.i = 1; //Integer.parseInt(nextLine[2].trim());
                hmSubst.put(subst, substCt);
            }
        }
```

```
TreeMap hmSNs = new TreeMap();
HashMap hmSN2 = new HashMap();

CSVReader reader2 = new CSVReader(new FileReader(SNs_FILE));

while ((nextLine = reader2.readNext()) != null) {
    String sN = new String(nextLine[0]);
    String snInvertido = "";

    String[] componentes = sN.split(" ");

    for(int i = 0; i < componentes.length; i++) {
        // verifica se alguns dos componentes do SN é um substantivo importante
        if(hmSubst.containsKey(componentes[i])){

            Counter sNct = new Counter();
            //sNCt.interesse = "S";
            for(int j = componentes.length - 1; j >= 0; j--) {
                snInvertido = snInvertido + " " + componentes[j];
            }
            System.out.print(sN + "\t"+ nextLine[1] + "\t" +
                Integer.toString(i+1));
            System.out.println("\t" + snInvertido);

            // guarda o SN invertido se a frequência for acima do threshold
            sNct.i = Integer.parseInt(nextLine[1].trim());
            if (sNct.i > 2){
                sNct.snInvertido = snInvertido.trim();
                sNct.snClasse    = formataClasse(sN);
                hmSNs.put(snInvertido.trim(), sNct);
                hmSN2.put(sN, sNct.snClasse);
            }
            break;
          } // fim do if de contain substantivo importante
    } // fim do for de componentes do sintagma nominal lido

    /*
    Counter substCt = new Counter();
    substCt.interesse = "N";
    substCt.i = Integer.parseInt(nextLine[2].trim());
    hmSubst.put(subst, substCt);

    System.out.println(nextLine[0] + ", " + nextLine[1] + ", " + nextLine[2]);
    */

} // fim do while que lê SNs

System.out.println("*************************************************");
System.out.println("********  CRIANDO TAXONOMIA  DE CONCEITOS  *********");
System.out.println("*************************************************");
System.out.println("");

outHeaderOWL(); // imprime o header do arquivo OWL

// trata todos os SNs Invertidos que foram coletados na TreeMap
String parteSN = "";
```

```
int aux;
for ( Iterator iter = hmSNs.values().iterator(); iter.hasNext(); ) {

    Counter snRef = (Counter) iter.next() ;
    parteSN = snRef.snInvertido;
    aux = parteSN.lastIndexOf(" ");

    while (aux != -1){

        // remove o último do SN invertido
        parteSN = parteSN.substring(0,aux);
        //System.out.println("      SNInvertido - pedaço: " + parteSN);

        if(hmSNs.containsKey(parteSN)){
            //achou pedaço do SN invertido no treemap - trata como subclasse
            Counter ct = (Counter)hmSNs.get(parteSN);

            outSubClasseOWL(snRef.snClasse, ct.snClasse); //subclasse,classe
            break;
        }

        aux = parteSN.lastIndexOf(" ");

    } // fim do while dos componentes do SN invertido

    // se não achou nenhuma parte do SN no TreeMap então trata
    // como uma classe isolada
    if (aux  == -1){
        //System.out.println("Classe   : " + snRef.snClasse);
        outClasseOWL(snRef.snClasse); // subclasse, classe
    }

} // fim do for de iteracao da treemap de SNs invertidos


//**********************************************************************
//**********************************************************************
// le as relações e cria subclasses e restricoes no arquivo OWL
System.out.println("\n");

HashMap hmRel = new HashMap();

CSVReader relReader = new CSVReader(new FileReader(Relacoes_FILE),
                                                    '\t', '~', 1);

while ((nextLine = relReader.readNext()) != null) {
    // verifica se existe o sujeito da relação na lista de SNs
    if(hmSN2.containsKey(nextLine[0]) & hmSN2.containsKey(nextLine[2])){
        String snSujeito = (String)hmSN2.get(nextLine[0]);
        String snObjeto  = (String)hmSN2.get(nextLine[2]);
        // String relacao   = nextLine[1];
        String relacao   = formataRelacao(nextLine[1].replace("_", " "));

        if (relacao.equals("is")) {
            // grava uma subclasse simplesmente
            outSubClasseOWL(snSujeito, snObjeto);
```

```
                } else {
                    // é uma relacao que tem ser implementada com propriedades
                    // e restrições
                    // se a relação ainda não foi gravada no arq OWL então grava
                    if (!hmRel.containsKey(relacao)) {
                        hmRel.put(relacao, relacao);
                        outObjectPropertyOWL(relacao);
                    }

                    // grava a restriçao e aproveita e põe a sentença como comentário
                    outRestrictionOWL(snSujeito, relacao, snObjeto,
                                      nextLine[3].replaceAll("", "_"));

                    // e complementa a subclasse do sujeito
                    outComplementaClasseOWL(snSujeito);

                } // else

            } // fim do if que verifica se os SNs existem
        } // fim do while

        outFooterOWL(); // fecha o arquivo OWL com últimos parametros

    } // fim do main
```

B.4 Padrões para Reconhecimento de Relações Específicas

Como visto na sessão 5.5.2, vários padrões foram utilizados no reconhecimento de relações específicas. A seguir é listado o código que implementa as gramáticas de reconhecimento expressas em JAPE e o código Java que gera as anotações correspondentes.

```
//*********************************************************************
//Identifica sentenças que contenham definições de conceitos,
//  classes e subclasses, definições de grupo / composição.
//Usa como insumo os POS tags do etiquetador (baseado em Hepple).
//Cria anotações do tipo "TERM" com as seguintes features:
//  * string - o fragmento de texto que foi anotado
//*********************************************************************
Phase:  AchaDefinicoes
Input:  Token
Options: control = appelt

//*********************************************************************
//      M A C R O S
//*********************************************************************
Macro: FULLSTOP (
 {Token.string=="."}
)

Macro: THREEDOTS (
 {Token.string=="."}
 {Token.string=="."}
 {Token.string=="."}
)

Macro: PUNCT (
 ({Token.string == "!"} |
  {Token.string == "?"} |
  {Token.string == ";"}
 )
)

Macro: SUBSTANTIVO (
  {Token.category == NN, Token.kind == word} |
  {Token.category == NNS, Token.kind == word}|
  {Token.category == NNP, Token.kind == word} |
  {Token.category == NNPS, Token.kind == word} |
  {Token.category == NP, Token.kind == word} |
  {Token.category == NPS, Token.kind == word} |
  {Token.category == CD, Token.kind == word}
)

Macro: ARTIGODEF (
    {Token.string == "the"} | {Token.string == "The"} | {Token.string == "THE"}
)
```

```
Macro: ARTIGOINDEF (
  {Token.string == "a"} | {Token.string == "an"} |
  {Token.string == "A"} | {Token.string == "An"}
)

Macro: ADJETIVO (
  {Token.category == JJ, Token.kind == word} |
  {Token.category == JJR, Token.kind == word} |
  {Token.category == JJS, Token.kind == word}
)

Macro: POSSESSIVO (
  {Token.category == POS, Token.kind == word}
)


Macro: ADVERBIO (
  {Token.category == RB, Token.kind == word} |
  {Token.category == RBR, Token.kind == word} |
  {Token.category == RBS, Token.kind == word}
)

Macro: BE (
 {Token.root == "be"}
)

Macro: MODAL (
   {Token.string == "can"} | {Token.string == "may"} |
   {Token.string == "should"} |
   {Token.string == "could"} | {Token.string == "must"}
)

Macro: CARACTERIZA1
// (relação: EH_DESCRITA_POR)
(   (BE)
    (ADVERBIO)?
    (({Token.string == "defined"}  {Token.string == "by"}) |
        ({Token.string == "described"} {Token.string == "by"}) |
        ({Token.string == "determined"} {Token.string == "by"}) |
        ({Token.string == "part"} {Token.string == "of"}) |
        ({Token.string == "considered"} {Token.string == "as"}) |
        ({Token.string == "referred"} {Token.string == "to"}
         {Token.string == "as"}) |
        ({Token.string == "thought"} {Token.string == "of"}
         {Token.string == "as"}) |
        ({Token.string == "known"} {Token.string == "as"})
        //  ({Token.string=="a"}|{Token.string=="an"}|
        //  {Token.string=="the"}|{Token.string=="many"}|{Token.string=="various"})
     )
)

Macro: DEFINE ( ({Token.string == "define"} | {Token.string ==
"defines"} |
   (({Token.string == "consist"} | {Token.string == "consists"})
    ({Token.string == "of"})) |
   {Token.string == "describe"} | {Token.string == "describes"} |
   {Token.string == "characterize"} | {Token.string == "characterizes"} |
```

```
   {Token.string == "determine"} | {Token.string == "determines"} |
   {Token.string == "mean"} | {Token.string == "means"} |
   {Token.string == "signify"} | {Token.string == "signifies"}            )
)

Macro: CARACTERIZA2 (   (BE)
    (ADVERBIO)?
    ({Token.string == "used"}  {Token.string == "to"})
    (DEFINE)
    //  ({Token.string=="a"}|{Token.string=="an"}|{Token.string=="the"}|
    //   {Token.string=="many"}|{Token.string=="various"})
)

Macro: CARACTERIZA3 (   (ADVERBIO)?
    (DEFINE)
    (ADVERBIO)?
    //  ({Token.string=="a"}|{Token.string=="an"}|{Token.string=="the"}|
    //    {Token.string=="many"}|{Token.string=="various"})
)

// achar composição ("part holonym"): "um trem é dividido em vagões"  (NA VOZ PASSIVA)
// exemplo: X (can) (usually) be divided into Y ou X... is (usually) divided into ...
//  para ser usado em: NP DIVIDIDO_EM NP
Macro: DIVIDIDO_EM (   (MODAL)?
    (ADVERBIO)?
    (BE)
    (ADVERBIO)?
    (({Token.string == "divided"}  {Token.string == "into"}) |
        ({Token.string == "subdivided"} {Token.string == "into"}) |
        ({Token.string == "part"} {Token.string == "of"}) |
        ({Token.string == "composed"} {Token.string == "of"}) |
        ({Token.string == "formed"}  {Token.string == "by"}) |
        ({Token.string == "constituted"} {Token.string == "of"}))
        //  ({Token.string=="a"}|{Token.string=="an"}|{Token.string=="the"}|
        //   {Token.string=="many"}|{Token.string=="various"})
)

// achar composição ("part holonym"): "um time inclue um lider e
//   vários trabalhadores"  (NA VOZ ATIVA)
// exemplo: X... includes ... para ser usado em: NP DIVIDE_SE_EM NP
Macro: DIVIDE_SE_EM (   (ADVERBIO)?
    ({Token.string == "involve"} | {Token.string == "involves"} |
     {Token.string == "include"} | {Token.string == "includes"}  )
    //  ({Token.string=="a"}|{Token.string=="an"}|{Token.string=="the"}|
    //  {Token.string=="many"}|{Token.string=="various"})
)

// achar hypernyms: "a project is a type of undertaking"
//      projeto é uma subclasse da classe undertaking
// exemplo: X... is generally a type of ... para ser usado em: NP UM_TIPO_DE NP
Macro: UM_TIPO_DE

( (BE)
  (ADVERBIO)?
  (ARTIGOINDEF)?
  ((({Token.string == "type"} | {Token.string == "types"}) ({Token.string == "of"})) |
   (({Token.string == "part"} | {Token.string == "parts"}) ({Token.string == "of"})) |
```

```
   (({Token.string == "kind"} | {Token.string == "kinds"}) ({Token.string == "of"})) |
   (({Token.string == "sort"} | {Token.string == "sorts"}) ({Token.string == "of"})) |
   (({Token.string == "class"} | {Token.string == "classes"})
    ({Token.string == "of"})) |
   (({Token.string == "category"} | {Token.string == "categories"})
    ({Token.string == "of"}))
  )
  //({Token.string=="a"}|{Token.string=="an"}|{Token.string=="the"}|
  // {Token.string=="many"}|{Token.string=="various"})
)

// achar GRUPOS : "a program is a group of projects" - projeto é uma
// subclasse da classe programa
// exemplo: X... is a set of ... para ser usado em: NP is GRUPO_DE NP
Macro: GRUPO_DE

(   (({Token.string == "group"} {Token.string =="of"}) |
     ({Token.string == "grouping"} {Token.string == "of"}) |
     ({Token.string == "set"} {Token.string == "of"}) |
     ({Token.string == "ensemble"} {Token.string == "of"}) |
     ({Token.string == "collection"} {Token.string == "of"})
    )
    //({Token.string=="a"}|{Token.string=="an"}|{Token.string=="the"}|
    // {Token.string=="many"}|{Token.string=="various"})
)

//*************************************************************************
//*************************************************************************
//                          R E G R A S
//*************************************************************************
Rule: DEF1
Priority: 50
(   ((ARTIGOINDEF) | (ARTIGODEF))?
    (ADJETIVO)? (ADJETIVO)?
    (SUBSTANTIVO) (SUBSTANTIVO)? (SUBSTANTIVO)?
    ({Token.string=="of"} | {Token.string=="from"} )?
    (SUBSTANTIVO)?  (SUBSTANTIVO)? (SUBSTANTIVO)?
    ({Token.string=="is"} | {Token.string=="are"} )
    ({Token.string=="a"}|{Token.string=="an"}|{Token.string=="the"})
    (ADJETIVO)? (ADJETIVO)? (SUBSTANTIVO)? (SUBSTANTIVO)?

    ({Token.kind == word}|{Token.string == ","}|{Token.kind == number})*
    ({Token.string == "."}|{Token.string == ";"})
):def1
//  (split)
--> {
    try{

        AnnotationSet defAnnot = (AnnotationSet)bindings.get("def1");

        TreeMap defsMap = new TreeMap();
        Vector vDefs = new Vector();
        String definicao = "";

        //cria um ordered Map com os termos, ordenado pelas posições no texto
        Iterator defsIt = defAnnot.iterator();
        Annotation umaDef;
```

```
        FeatureMap features;
        while (defsIt.hasNext()){
            umaDef = (Annotation)defsIt.next();
            features = umaDef.getFeatures();
            defsMap.put(umaDef.getStartNode().getOffset(), features.get("string"));
        }
        //transforma o treemap no termo composto (pode ser um ou mais substantivos)
        Iterator mapIt = defsMap.keySet().iterator();
        Long key;
        while (mapIt.hasNext()){
            key = (Long)mapIt.next();
            definicao = definicao + " " + (String)defsMap.get(key);
            vDefs.add((String)defsMap.get(key));
            }

        // definicao = definicao + " " + headnoun.getFeatures().get("string");
        definicao = definicao.trim();

        // cria uma estrutura de features e preenche com as características
        FeatureMap term_features = Factory.newFeatureMap();
        term_features.put("string", definicao.toLowerCase());
        term_features.put("relation", "IsA");
        term_features.put("rule", "DEF1");

        // salva a anotação e suas features no documento
        outputAS.add(defAnnot.firstNode().getOffset(),
            defAnnot.lastNode().getOffset(),
            "Definition",
            term_features);

    }catch(InvalidOffsetException ioe){
            ioe.printStackTrace();
    }
}

//*****************************************************************************
Rule: DEF2
Priority: 50
(   ((ARTIGOINDEF) | (ARTIGODEF))?
    (ADJETIVO)? (ADJETIVO)?
    (SUBSTANTIVO) (SUBSTANTIVO)? (SUBSTANTIVO)?
    ({Token.string=="of"} | {Token.string=="from"} )?
    (SUBSTANTIVO)?  (SUBSTANTIVO)? (SUBSTANTIVO)?
    ({Token.string=="is"} | {Token.string=="are"} )
    ({Token.string=="a"}|{Token.string=="an"}|{Token.string=="the"})
    (GRUPO_DE)  // MANDATóRIO !!!
    (ADJETIVO)? (ADJETIVO)? (SUBSTANTIVO)? (SUBSTANTIVO)?

    ({Token.kind == word}|{Token.string == ","}|{Token.kind == number})*
    ({Token.string == "."}|{Token.string == ";"})
):def2 --> {
    try{

        AnnotationSet defAnnot = (AnnotationSet)bindings.get("def2");

        TreeMap defsMap = new TreeMap();
        Vector vDefs = new Vector();
```

```
        String definicao = "";

        //cria um ordered Map com os termos, ordenado pelas posições no texto
        Iterator defsIt = defAnnot.iterator();
        Annotation umaDef;
        FeatureMap features;
        while (defsIt.hasNext()){
            umaDef = (Annotation)defsIt.next();
            features = umaDef.getFeatures();
            defsMap.put(umaDef.getStartNode().getOffset(), features.get("string"));
        }
        //transforma o treemap no termo composto (pode ser um ou mais substantivos)
        Iterator mapIt = defsMap.keySet().iterator();
        Long key;
        while (mapIt.hasNext()){
            key = (Long)mapIt.next();
            definicao = definicao + " " + (String)defsMap.get(key);
            vDefs.add((String)defsMap.get(key));
            }

        // definicao = definicao + " " + headnoun.getFeatures().get("string");
        definicao = definicao.trim();

        // cria uma estrutura de features e preenche com as características
        FeatureMap term_features = Factory.newFeatureMap();
        term_features.put("string", definicao.toLowerCase());
        term_features.put("relation", "IsACollectionOf");
        term_features.put("rule", "DEF2");

        // salva a anotação e suas features no documento
        outputAS.add(defAnnot.firstNode().getOffset(),
            defAnnot.lastNode().getOffset(),
            "Definition",
            term_features);

    }catch(InvalidOffsetException ioe){
            ioe.printStackTrace();
    }
}

//*********************************************************************
Rule: CARACT1
Priority: 40
(   ((ARTIGODEF)|(ARTIGOINDEF))?
    (ADJETIVO)? (ADJETIVO)?
    (SUBSTANTIVO) (SUBSTANTIVO)? (SUBSTANTIVO)?
    ({Token.string=="of"} | {Token.string=="from"} )?
    (SUBSTANTIVO)?  (SUBSTANTIVO)? (SUBSTANTIVO)?
    ((CARACTERIZA1) | (CARACTERIZA2) | (CARACTERIZA3))

    ({Token.kind == word}|{Token.string == ","}|{Token.kind == number})*
    ({Token.string == "."}|{Token.string == ";"})
):def3 --> {
    try{
        AnnotationSet defAnnot = (AnnotationSet)bindings.get("def3");

        TreeMap defsMap = new TreeMap();
```

```
        Vector vDefs = new Vector();
        String definicao = "";

        //cria um ordered Map com os termos, ordenado pelas posições no texto
        Iterator defsIt = defAnnot.iterator();
        Annotation umaDef;
        FeatureMap features;
        while (defsIt.hasNext()){
            umaDef = (Annotation)defsIt.next();
            features = umaDef.getFeatures();
            defsMap.put(umaDef.getStartNode().getOffset(), features.get("string"));
        }
        //transforma o treemap no termo composto (pode ser um ou mais substantivos)
        Iterator mapIt = defsMap.keySet().iterator();
        Long key;
        while (mapIt.hasNext()){
            key = (Long)mapIt.next();
            definicao = definicao + " " + (String)defsMap.get(key);
            vDefs.add((String)defsMap.get(key));
            }

        // definicao = definicao + " " + headnoun.getFeatures().get("string");
        definicao = definicao.trim();

        // cria uma estrutura de features e preenche com as características
        FeatureMap term_features = Factory.newFeatureMap();
        term_features.put("string", definicao.toLowerCase());
        term_features.put("relation", "Describes");
        term_features.put("rule", "CARACT1");

        // salva a anotação e suas features no documento
        outputAS.add(defAnnot.firstNode().getOffset(),
            defAnnot.lastNode().getOffset(),
            "Definition",
            term_features);

    }catch(InvalidOffsetException ioe){
            ioe.printStackTrace();
    }
}

//*****************************************************************************
Rule: PARTE1
Priority: 40
(   ((ARTIGODEF)|(ARTIGOINDEF))?
    (ADJETIVO)? (ADJETIVO)?
    (SUBSTANTIVO) (SUBSTANTIVO)? (SUBSTANTIVO)?
    ({Token.string=="of"} | {Token.string=="from"} )?
    (SUBSTANTIVO)?  (SUBSTANTIVO)? (SUBSTANTIVO)?
    ((DIVIDIDO_EM) | (DIVIDE_SE_EM))

    ({Token.kind == word}|{Token.string == ","}|{Token.kind == number})*
    ({Token.string == "."}|{Token.string == ";"})
):def4 --> {
    try{
        AnnotationSet defAnnot = (AnnotationSet)bindings.get("def4");
```

```
        TreeMap defsMap = new TreeMap();
        Vector vDefs = new Vector();
        String definicao = "";

        //cria um ordered Map com os termos, ordenado pelas posições no texto
        Iterator defsIt = defAnnot.iterator();
        Annotation umaDef;
        FeatureMap features;
        while (defsIt.hasNext()){
            umaDef = (Annotation)defsIt.next();
            features = umaDef.getFeatures();
            defsMap.put(umaDef.getStartNode().getOffset(), features.get("string"));
        }
        //transforma o treemap no termo composto (pode ser um ou mais substantivos)
        Iterator mapIt = defsMap.keySet().iterator();
        Long key;
        while (mapIt.hasNext()){
            key = (Long)mapIt.next();
            definicao = definicao + " " + (String)defsMap.get(key);
            vDefs.add((String)defsMap.get(key));
            }

        // definicao = definicao + " " + headnoun.getFeatures().get("string");
        definicao = definicao.trim();

        // cria uma estrutura de features e preenche com as características
        FeatureMap term_features = Factory.newFeatureMap();
        term_features.put("string", definicao.toLowerCase());
        term_features.put("relation", "IsComposedOf");
        term_features.put("rule", "PARTE1");

        // salva a anotação e suas features no documento
        outputAS.add(defAnnot.firstNode().getOffset(),
            defAnnot.lastNode().getOffset(),
            "Definition",
            term_features);

    }catch(InvalidOffsetException ioe){
            ioe.printStackTrace();
    }
}

//*****************************************************************************
Rule: PARTE2
Priority: 40
(   ((ARTIGODEF)|(ARTIGOINDEF))?
    (ADJETIVO)? (ADJETIVO)?
    (SUBSTANTIVO) (SUBSTANTIVO)? (SUBSTANTIVO)?
    ({Token.string=="of"} | {Token.string=="from"} )?
    (SUBSTANTIVO)?  (SUBSTANTIVO)? (SUBSTANTIVO)?
    (UM_TIPO_DE)

    ({Token.kind == word}|{Token.string == ","}|{Token.kind == number})*
    ({Token.string == "."}|{Token.string == ";"})
):def5 --> {
    try{
        AnnotationSet defAnnot = (AnnotationSet)bindings.get("def5");
```

```
        TreeMap defsMap = new TreeMap();
        Vector vDefs = new Vector();
        String definicao = "";

        //cria um ordered Map com os termos, ordenado pelas posições no texto
        Iterator defsIt = defAnnot.iterator();
        Annotation umaDef;
        FeatureMap features;
        while (defsIt.hasNext()){
            umaDef = (Annotation)defsIt.next();
            features = umaDef.getFeatures();
            defsMap.put(umaDef.getStartNode().getOffset(), features.get("string"));
        }
        //transforma o treemap no termo composto (pode ser um ou mais substantivos)
        Iterator mapIt = defsMap.keySet().iterator();
        Long key;
        while (mapIt.hasNext()){
            key = (Long)mapIt.next();
            definicao = definicao + " " + (String)defsMap.get(key);
            vDefs.add((String)defsMap.get(key));
            }

        // definicao = definicao + " " + headnoun.getFeatures().get("string");
        definicao = definicao.trim();

        // cria uma estrutura de features e preenche com as características
        FeatureMap term_features = Factory.newFeatureMap();
        term_features.put("string", definicao.toLowerCase());
        term_features.put("relation", "IsATypeOf");
        term_features.put("rule", "PARTE2");

        // salva a anotação e suas features no documento
        outputAS.add(defAnnot.firstNode().getOffset(),
            defAnnot.lastNode().getOffset(),
            "Definition",
            term_features);

    }catch(InvalidOffsetException ioe){
            ioe.printStackTrace();
    }
}
```

B.5 Padrões para Reconhecimento de Relações Verbais

A gramática JAPE especificada abaixo é utilizada para reconhecimento de relações verbais e criação das anotações correspondentes (o sintagma nominal que constitui a cabeça da relação verbal, o sintagma verbal e o sintagma nominal que representa o objeto ou foco da ação verbal).

```
//*********************************************************************
//Identifica sentenças que contenham definições de relações.
//Usa como insumo os POS tags do etiquetador,
//  a análise morfológica (root) e os sintagmas verbais
//  Verb Groups (VG) do VPChunker
//*********************************************************************
Phase:  AchaRelacoes Input:  Token SN1 VG Options: control = appelt

//*********************************************************************
//      M A C R O S
//*********************************************************************

Macro: SUBSTANTIVO (
  {Token.category == NN, Token.kind == word} |
  {Token.category == NNS, Token.kind == word}|
  {Token.category == NNP, Token.kind == word} |
  {Token.category == NNPS, Token.kind == word} |
  {Token.category == NP, Token.kind == word} |
  {Token.category == NPS, Token.kind == word}
//  | {Token.category == CD, Token.kind == word}
)

Macro: SUBSTANTIVOPluralMaiusculo (
  {Token.category == NNP, Token.kind == word, Token.orth == upperInitial}|
  {Token.category == NN, Token.kind == word, Token.orth == upperInitial}|
  {Token.category == NNS, Token.kind == word, Token.orth == upperInitial}
)

// artigos definidos e indefinidos, "any", "each", "every",
Macro: DETERMINER (
    {Token.category == DT, Token.kind == word}
)

Macro: PREPOSITION (
//  {Token.category == IN}
    {Token.category == IN, Token.root == "of"}
    | {Token.category == IN, Token.root == "in"}
    | {Token.category == IN, Token.root == "on"}
    | {Token.category == IN, Token.root == "by"}
    | {Token.category == IN, Token.root == "at"}
    | {Token.category == IN, Token.root == "from"}
    | {Token.category == IN, Token.root == "into"}
    | {Token.category == IN, Token.root == "onto"}
    | {Token.category == IN, Token.root == "over"}
    | {Token.category == IN, Token.root == "through"}
```

```
    | {Token.category == IN, Token.root == "within"}
    | {Token.category == IN, Token.root == "with"}
    | {Token.category == IN, Token.root == "against"}
    | {Token.category == IN, Token.root == "across"}
    | {Token.category == IN, Token.root == "off"}
    | {Token.category == IN, Token.root == "along"}
)

Macro: VERBGROUP (
    // busco o tempo verbal 'presente' aqui por que é o mais tipicamente
    // usado em definições de conceitos
    {VG.tense == SimPre}
    | {VG.tense == None, VG.type == MODAL}
)

Macro: ARTIGO (
    {Token.string == "the"} | {Token.string == "The"} | {Token.string == "THE"} |
    {Token.string == "a"} | {Token.string == "an"} |
    {Token.string == "A"} | {Token.string == "An"} |
    {Token.string == "any"} | {Token.string == "Any"} |
    {Token.string == "every"} | {Token.string == "Every"} |
    {Token.string == "each"} | {Token.string == "Each"}
)

Macro: ARTIGODEF (
    {Token.string == "the"} | {Token.string == "The"} | {Token.string == "THE"}
)

Macro: ARTIGOINDEF (
  {Token.string == "a"} | {Token.string == "an"} |
  {Token.string == "A"} | {Token.string == "An"}
)

Macro: EXAMPLEOF (
     ((ARTIGODEF) | (ARTIGOINDEF))?
     (({Token.string == "type"}     | {Token.string == "types"})
      ({Token.string == "of"})) |
     (({Token.string == "example"}  | {Token.string == "examples"})
      ({Token.string == "of"})) |
     (({Token.string == "part"}     | {Token.string == "parts"})
      ({Token.string == "of"})) |
     (({Token.string == "kind"}     | {Token.string == "kinds"})
      ({Token.string == "of"})) |
     (({Token.string == "sort"}     | {Token.string == "sorts"})
      ({Token.string == "of"})) |
     (({Token.string == "subset"}   | {Token.string == "subsets"})
      ({Token.string == "of"})) |
     (({Token.string == "class"}    | {Token.string == "classes"})
      ({Token.string == "of"})) |
     (({Token.string == "category"} | {Token.string == "categories"})
      ({Token.string == "of"}))
)

Macro: GROUPOF (
    ((ARTIGODEF) | (ARTIGOINDEF))?
    (({Token.string == "group"}     {Token.string == "of"}) |
     ({Token.string == "grouping"}  {Token.string == "of"}) |
```

```
     ({Token.string == "set"}       {Token.string == "of"}) |
     ({Token.string == "ensemble"}  {Token.string == "of"}) |
     ({Token.string == "collection"} {Token.string == "of"})
    )
)


Macro: ADJETIVO (   // usa também VBNs como em "documented"
procedures
  {Token.category == VBN, Token.kind == word} |
  {Token.category == JJ,  Token.kind == word} |
  {Token.category == JJR, Token.kind == word} |
  {Token.category == JJS, Token.kind == word}
)

Macro: ADJETIVOMaiusculo (
  {Token.category == JJ, Token.kind == word, Token.orth == upperInitial}
)

Macro: POSSESSIVO (
  {Token.category == POS, Token.kind == word}
)


Macro: ADVERBIO (
  {Token.category == RB, Token.kind == word} |
  {Token.category == RBR, Token.kind == word} |
  {Token.category == RBS, Token.kind == word}
)

Macro: BE (   {Token.root == "be"} )

Macro: MODAL (   {Token.string == "can"} | {Token.string == "may"} |
{Token.string == "should"} |
    {Token.string == "could"} | {Token.string == "must"}
)


//*********************************************************************
//                          R E G R A S
//*********************************************************************
Rule: REL1
Priority: 50
(
    (((ARTIGO)
      (ADJETIVO)*
      ((SUBSTANTIVO)* (SUBSTANTIVO)):RelNounHead1 ) |
     (((SUBSTANTIVOPluralMaiusculo) (SUBSTANTIVO)* ):RelNounHead2 ) |
     ((ADJETIVOMaiusculo) (ADJETIVO)*
      ((SUBSTANTIVO)* (SUBSTANTIVO)):RelNounHead3 )
    )

    ((VERBGROUP) (EXAMPLEOF)? (GROUPOF)? (PREPOSITION)? ):VBPrep

    (DETERMINER)? (ADJETIVO)? (({Token.string==","}) (ADJETIVO))*

    ((SUBSTANTIVO)* (SUBSTANTIVO)):RelNounObj
```

```
):def1 -->
    :def1.Relacao = {rule = "REL1"}
  , :RelNounHead1.RelNounHead = {rule = "REL1-RelNounHead1"}
  , :RelNounHead2.RelNounHead = {rule = "REL1-RelNounHead2"}
  , :RelNounHead3.RelNounHead = {rule = "REL1-RelNounHead3"}
  , :VBPrep.RelVBPrep = {rule = "REL1-VBPrep"}
  , :RelNounObj.RelNounObj = {rule = "REL1-RelNounObj"}
```

B.6 Padrões para Reconhecimento de Relações de Composição & Partes

A gramática JAPE especificada abaixo é utilizada para reconhecimento de relações do tipo Composição & Partes implementadas por formações verbais do tipo "isPartOf" ou "have".

```
//*********************************************************************
Phase:  PartsOf
Input:  Token
Options: control = appelt

Macro: SUBSTANTIVO (
  {Token.category == NN, Token.kind == word} |
  {Token.category == NNS, Token.kind == word}|
  {Token.category == NNP, Token.kind == word} |
  {Token.category == NNPS, Token.kind == word} |
  {Token.category == NP, Token.kind == word} |
  {Token.category == NPS, Token.kind == word}
)

Macro: ARTIGODEF (
    {Token.string == "the"} | {Token.string == "The"} |
    {Token.string == "THE"}
)

Macro: ARTIGOINDEF (
  {Token.string == "a"} | {Token.string == "an"} |
  {Token.string == "A"} | {Token.string == "An"}
)

Macro: ADJETIVO (
  {Token.category == JJ, Token.kind == word} |
  {Token.category == JJR, Token.kind == word} |
  {Token.category == JJS, Token.kind == word}
)

Macro: PARTIC_PASSADO (
  {Token.category == VBN, Token.kind == word}
)

Macro: POSSESSIVO (
  {Token.category == POS, Token.kind == word}
)

Macro: ADVERBIO (
  {Token.category == RB, Token.kind == word} |
  {Token.category == RBR, Token.kind == word} |
  {Token.category == RBS, Token.kind == word}
)


////////////////////////////////////////////////////////////////////
```

```
//             Tipo de padrão:  Business's objectives
////////////////////////////////////////////////////////////////////
Rule: PartOf1
Priority: 50
(
  (((SUBSTANTIVO)* (SUBSTANTIVO)):Todo)
   (POSSESSIVO)
  (((SUBSTANTIVO)* (SUBSTANTIVO)):Parte)
):PartOf

-->

{   try{

        AnnotationSet partOfAnnSet = (AnnotationSet)bindings.get("PartOf");
        Annotation partOf = (Annotation)partOfAnnSet.iterator().next();

        ////////////////////////////////////////////////////////////
        // MONTA a string de Parte lendo os tokens correspondentes
        AnnotationSet parteAnnSet = (AnnotationSet)bindings.get("Parte");
        String parte = "";
        TreeMap annotMap = new TreeMap();
        if (parteAnnSet != null && parteAnnSet.size() != 0) {

            //cria um ordered Map com modificadores ordenado por
            //  posição no texto
            Iterator iter = parteAnnSet.iterator();
            Annotation annot;
            FeatureMap features;
            while (iter.hasNext()){
                annot = (Annotation)iter.next();
                features = annot.getFeatures();
                // verifica se não é sigla
                if (features.get("orth").equals("allCaps")) {
                    annotMap.put(annot.getStartNode().getOffset(),
                        features.get("string"));
                } else {
                    annotMap.put(annot.getStartNode().getOffset(),
                        features.get("root"));
                }
            }
            Iterator mapIt = annotMap.keySet().iterator();
            Long key;
            while (mapIt.hasNext()){
                key = (Long)mapIt.next();
                parte = parte + " " + (String)annotMap.get(key);
            }
            parte = parte.trim();
        }

        ////////////////////////////////////////////////////////////////
        // MONTA a string de Todo lendo os tokens correspondentes
        AnnotationSet todoAnnSet = (AnnotationSet)bindings.get("Todo");
        String todo = "";
        annotMap.clear();
        if (todoAnnSet != null && todoAnnSet.size() != 0) {
```

```
            //cria um ordered Map com modificadores ordenado por
            //  posição no texto
            Iterator iter = todoAnnSet.iterator();
            Annotation annot;
            FeatureMap features;
            while (iter.hasNext()){
                annot = (Annotation)iter.next();
                features = annot.getFeatures();
                // verifica se não é sigla
                if (features.get("orth").equals("allCaps")) {
                    annotMap.put(annot.getStartNode().getOffset(),
                        features.get("string"));
                } else {
                    //corrige erro do POS tagger que marcou "cannot" como "NN"
                    if ( !features.get("root").equals("cannot") ) {
                    annotMap.put(annot.getStartNode().getOffset(),
                        features.get("root"));
                    //System.out.println("map todo <- " + features.get("root") );
                    }
                }
            }
            Iterator mapIt = annotMap.keySet().iterator();
            Long key;
            while (mapIt.hasNext()){
                key = (Long)mapIt.next();
                todo = todo + " " + (String)annotMap.get(key);
            }
            todo = todo.trim();
        }

        gate.FeatureMap features = Factory.newFeatureMap();
        features.put("todo", todo);
        features.put("relacao", "is_part_of");
        features.put("parte", parte);

        outputAS.add(partOfAnnSet.firstNode().getOffset(),
            partOfAnnSet.lastNode().getOffset(), "PartOf", features);

        } catch(InvalidOffsetException ioe){
                ioe.printStackTrace();
    }
}


////////////////////////////////////////////////////////////////////////////
//                  objectives of the business
////////////////////////////////////////////////////////////////////////////
Rule: PartOf2 Priority: 50 (
  (((SUBSTANTIVO)* (SUBSTANTIVO)):Parte )
   ({Token.string == "of"} | {Token.string == "OF"} | {Token.string == "Of"})
   ((ARTIGODEF) | (ARTIGOINDEF)) (ADJETIVO)? (ADJETIVO)? (ADJETIVO)?
  (((SUBSTANTIVO)* (SUBSTANTIVO)):Todo)
):PartOf

-->
```

```
{   try{

        AnnotationSet partOfAnnSet = (AnnotationSet)bindings.get("PartOf");
        Annotation partOf = (Annotation)partOfAnnSet.iterator().next();

        ////////////////////////////////////////////////////////////////////
        // MONTA a string de Parte lendo os tokens correspondentes
        AnnotationSet parteAnnSet = (AnnotationSet)bindings.get("Parte");
        String parte = "";
        TreeMap annotMap = new TreeMap();
        if (parteAnnSet != null && parteAnnSet.size() != 0) {

            //create an ordered Map with the modifiers, ordered by their
            // position in the text
            Iterator iter = parteAnnSet.iterator();
            Annotation annot;
            FeatureMap features;
            while (iter.hasNext()){
                annot = (Annotation)iter.next();
                features = annot.getFeatures();
                // verifica se não é sigla
                if (features.get("orth").equals("allCaps")) {
                    annotMap.put(annot.getStartNode().getOffset(),
                        features.get("string"));
                } else {
                    annotMap.put(annot.getStartNode().getOffset(),
                        features.get("root"));
                }
                //System.out.println("map parte <- " + features.get("string") );
            }
            Iterator mapIt = annotMap.keySet().iterator();
            Long key;
            while (mapIt.hasNext()){
                key = (Long)mapIt.next();
                parte = parte + " " + (String)annotMap.get(key);
            }
            parte = parte.trim();
        }

        ////////////////////////////////////////////////////////////////////
        // MONTA a string de Todo lendo os tokens correspondentes
        AnnotationSet todoAnnSet = (AnnotationSet)bindings.get("Todo");
        String todo = "";
        annotMap.clear();
        if (todoAnnSet != null && todoAnnSet.size() != 0) {

            //cria um ordered Map com modificadores ordenado por
            //  posição no texto
            Iterator iter = todoAnnSet.iterator();
            Annotation annot;
            FeatureMap features;
            while (iter.hasNext()){
                annot = (Annotation)iter.next();
                features = annot.getFeatures();
                // verifica se não é sigla
                if (features.get("orth").equals("allCaps")) {
                    annotMap.put(annot.getStartNode().getOffset(),
```

```
                    features.get("string"));
            } else {
                //corrige erro do POS tagger que marcou "cannot" como "NN"
                if ( !features.get("root").equals("cannot") ) {
                annotMap.put(annot.getStartNode().getOffset(),
                    features.get("root"));
                }
            }
        }
        Iterator mapIt = annotMap.keySet().iterator();
        Long key;
        while (mapIt.hasNext()){
            key = (Long)mapIt.next();
            todo = todo + " " + (String)annotMap.get(key);
        }
        todo = todo.trim();
    }

    gate.FeatureMap features = Factory.newFeatureMap();
    features.put("todo", todo);
    features.put("relacao", "is_part_of");
    features.put("parte", parte);

    outputAS.add(partOfAnnSet.firstNode().getOffset(),
    partOfAnnSet.lastNode().getOffset(),"PartOf", features);
    //System.out.println("PartOf: "+ todo + " tem " + parte);

    } catch(InvalidOffsetException ioe){
            ioe.printStackTrace();
  }
}
```

Anexo C – Recursos Utilizados

Aqui são apresentados alguns dos recursos utilizados neste trabalho.

C.1 O Corpus do PMBOK

Tabela C.1: Alguns dos primeiros parágrafos do PMBOK

Organizations perform work. Work generally involves either operations or projects, although the two may overlap. Operations and projects share many characteristics; for example, they are: performed by people; constrained by limited resources; planned, executed, and controlled. Operations and projects differ primarily in that operations are ongoing and repetitive while projects are temporary and unique. A project can thus be defined in terms of its distinctive characteristics: a project is a temporary endeavor undertaken to create a unique product or service. Temporary means that every project has a definite beginning and a definite end. Unique means that the product or service is different in some distinguishing way from all similar products or services. Projects are undertaken at all levels of the organization. They may involve a single person or many thousands. They may require less than 100 hours to complete or over 10,000,000. Projects may involve a single unit of one organization or may cross organizational boundaries as in joint ventures and partnering. Projects are often critical components of the performing organization's business strategy. Examples of projects include: Developing a new product or service; Effecting a change in structure, staffing, or style of an organization; Designing a new transportation vehicle; Developing or acquiring a new or modified information system; Constructing a building or facility; Running a campaign for political office; Implementing a new business procedure or process. Temporary means that every project has a definite beginning and a definite end. The end is reached when the project's objectives have been achieved, or when it becomes clear that the project objectives will not or cannot be met and the project is terminated. Temporary does not necessarily mean short in duration; many projects last for several years. In every case, however, the duration of a project is finite; projects are not ongoing efforts. In addition, temporary does not generally apply to the product or service created by the project. Most projects are undertaken to create a lasting result. For example, a project to erect a national monument will create a result expected to last centuries. Many undertakings are temporary in the sense that they will end at some point. For example, assembly work at an automotive plant will eventually be discontinued, and the plant itself decommissioned. Projects are fundamentally different because the project ceases when its declared objectives have been attained, while nonproject undertakings adopt a new set of objectives and continue to work.

C.2 O Corpus Brown

O corpus Brown tem 1.015.945 palavras e é composto de textos em inglês americano sobre temas variados. A seguir são listados alguns parágrafos do corpus.

Tabela C.2: Alguns parágrafos do Corpus Brown

East Providence should organize its civil defense setup and begin by appointing a full-time director, Raymond H. Hawksley, the present city CD head, believes.
Mr. Hawksley said yesterday he would be willing to go before the city council "or anyone else locally"to outline his proposal at the earliest possible time. East Providence now has no civil defense program.
Mr. Hawksley, the state's general treasurer, has been a part-time CD director in the city for the last nine years.
He is not interested in being named a full-time director. Noting that President Kennedy has handed the Defense Department the major responsibility for the nation's civil defense program, Mr. Hawksley said the federal government would pay half the salary of a full-time local director.
Mr. Hawksley said he believed there are a number of qualified city residents who would be willing to take the full-time CD job.
One of these men is former Fire Chief John A. Laughlin, he said. Along with a director, the city should provide a CD headquarters so that pertinent information about the local organization would be centralized. Mr. Hawksley said.
One advantage that would come to the city in having a full-time director, he said, is that East Providence would become eligible to apply to the federal government for financial aid in purchasing equipment needed for a sound civil defense program.
Matching funds also can be obtained for procurement of such items as radios, sirens and rescue trucks, he said.
Mr. Hawksley believes that East Providence could use two more rescue trucks, similar to the CD vehicle obtained several years ago and now detailed to the Central Fire Station.
He would assign one of the rescue trucks to the Riverside section of the city and the other to the Rumford area. Speaking of the present status of civil defense in the city, Mr. Hawksley said he would be willing to bet that not more than one person in a hundred would know what to do or where to go in the event of an enemy attack.
The Narragansett Race Track grounds is one assembly point, he said, and a drive-in theater in Seekonk would be another.
Mr. Hawksley said he was not critical of city residents for not knowing what to do or where to assemble in case of an air attack.
Rhode Island is going to examine its Sunday sales law with possible revisions in mind. Governor Notte said last night he plans to name a committee to make the study and come up with recommendations for possible changes in time for the next session of the General Assembly.
The governor's move into the so-called "blue law"controversy came in the form of a letter to Miss Mary R. Grant, deputy city clerk of Central Falls.
A copy was released to the press.
Mr. Notte was responding to a resolution adopted by the Central Falls City Council on July 10 and sent to the state house by Miss Grant.
The resolution urges the governor to have a complete study of the Sunday sales laws made with an eye to their revision at the next session of the legislature.

Referências

AGIRRE, E.; ANSA, O.; HOVY, E.; MARTINEZ, D. *Enriching very large ontologies using the WWW*. out. 17 2000. Disponível em: <http://arxiv.org/abs/cs/0010026>.

AGRAWAL, R.; SRIKANT, R. Mining generalized association rules. In: DAYAL, U.; GRAY, P. M. D.; NISHIO, S. (Ed.). *Proc. 21st Int. Conf. Very Large Data Bases, VLDB*. [S.l.]: Morgan Kaufmann, 1995. p. 407–419. ISBN 1-55860-379-4.

ALFONSECA, E.; MANANDHAR, S. *Improving an Ontology Refinement Method with Hyponymy Patterns*. jun. 27 2002. Disponível em: <http://citeseer.ist.psu.edu/540233.html; http://www.ii.uam.es/ ealfon/pubs/patterns.ps>.

ALFONSECA, E.; MANANDHAR, S. *An Unsupervised Method for General Named Entity Recognition And Automated Concept Discovery*. jan. 17 2002. Disponível em: <http://citeseer.ist.psu.edu/534491.html; http://www.ii.uam.es/ ealfon/pubs/generalne.ps>.

AUSSENAC-GILLES, N.; BIEBOW, B.; SZULMAN, S. Revisiting ontology design: A methodology based on corpus analysis. *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*, Springer-Verlag London, UK, p. 172–188, 2000.

BARNETT, S.; CRONIN, T. M. *Mathematical formulae for engineering and science students, fourth edition.* [S.l.]: Longman, London., 1986.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web. *Scientific American*, maio 2001. Disponível em: <http://www.sciam.com/2001/0501issue/0501berners-lee.html>.

BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C. M. *Extensible Markup Language (XML) 1.0 — W3C Recommendation 10-February-1998*. [S.l.], fev. 1998.

BRILL, E. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, v. 21, n. 4, p. 543–565, 1995.

BURNARD, L. *Reference Guide for the British National Corpus (World Edition)*. Oxford University Computing Services, 2000. Disponível em: <www.hcu.ox.ac.uk/BNC>.

CHOMSKY, N. Three models for the description of language. *IRE Transactions on Information Theory*, v. 2, n. 3, p. 113–124, 1956.

CORCHO, Ó.; FERNÁNDEZ-LÓPEZ, M.; GÓMEZ-PÉREZ, A. Methodologies, tools and languages for building ontologies: Where is their meeting point? *Data & Knowledge Engineering*, v. 46, n. 1, p. 41–64, 2003. Disponível em: <http://dx.doi.org/10.1016/S0169-023X(02)00195-7>.

CUNNINGHAM, H.; GAIZAUSKAS, R. J.; WILKS, Y. *A General Architecture for Language Engineering (GATE) - a new approach to Language Engineering R&D*. [S.l.], jan. 30 1996. Comment: 52 page technical report, LaTeX 2e source. Disponível em: <http://arxiv.org/abs/cmp-lg/9601009>.

CUNNINGHAM, H.; MAYNARD, D.; BONTCHEVA, K.; TABLAN, V.; URSU, C.; DIMITROV, M.; DOWMAN, M.; ASWANI, N.; ROBERTS, I. *Developing Language Processing Components with GATE Version 4 (a User Guide)*. [S.l.], 2006.

CUNNINGHAM, H.; MAYNARD, D.; TABLAN, V. *JAPE: a Java Annotation Patterns Engine*. nov. 01 2000.

FAATZ, A.; STEINMETZ, R. *Ontology Enrichment with Texts from the WWW*. jul. 03 2002. Disponível em: <http://citeseer.ist.psu.edu/558028.html; ftp://ftp.kom.e-technik.tu-darmstadt.de/pub/papers/FS02-1.pdf>.

FELLBAUM, C. (Ed.). *WordNet: An Electronic Lexical Database*. Cambridge, Massachusetts: The MIT Press, 1989.

FENSEL, D.; HORROCKS, I.; Van Harmelen, F.; DECKER, S.; ERDMANN, M.; KLEIN, M. OIL in a nutshell. In: DIENG, R. (Ed.). *Proceedings of the 12th. European Workshop on Knowledge Acquisition, Modeling and Management (EKAW-00)*. [S.l.]: Springer-Verlag, 2000, (Lecture Notes in Artificial Intelligence, 1937).

FERRUCCI, D.; LALLY, A. Building an example application with the unstructured information management architecture. In: *Unstructured Information Management*. [S.l.: s.n.], 2004, (IBM Systems Journal, 3).

FRANCIS, W. N.; KUCERA, H. *Computational analysis of present-day American English*. Providence: Brown University Press, 1967.

GAZDAR, G. Phrase structure grammar. Unpublished manuscript, Cognitive Studies Program, University of Sussex. 1980.

GENESERETH, M. *Automated Consultation for Complex Computer Systems*. Tese (Doutorado) — Harvard University, Cambridge, Mass, 1978.

GENESERETH, M.; FIKES, R. *Knowledge interchange format, version 3.0. Reference manual*. [S.l.], 1992.

GRUBER, T. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, v. 43, n. 5/6, p. 907–928, 1995.

GUARINO, N.; GIARETTA, P. Ontologies and knowledge bases: Towards a terminological clarification. In: MARS, N. (Ed.). *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*. [S.l.]: IOS Press, 1995. p. 25–32.

HATCHER, E.; GOSPODNETIC, O. *Lucene in Action*. [S.l.]: Manning Publications Company, 2005.

HEARST, M. A. Automatic acquisition of hyponyms from large text corpora. In: *Proc. of the 14th COLING*. Nantes, France: [s.n.], 1992. p. 539–545.

HEPPLE, M. Independence and commitment. assumptions for rapid training and execution of rule-based pos taggers. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS (ACL-2000). [S.l.], 2000.

HOFLAND, K.; JOHANSSON, S. *Word frequencies in British and American English*. [S.l.]: The Norwegian Computing Centre for the Humanities, Bergen, Norway, 1982.

HORROCKS, I. DAML+OIL: a reason-able web ontology language. In: JENSEN, C. S.; JEFFERY, K. G.; POKORNÝ, J.; SALTENIS, S.; BERTINO, E.; BÖHM, K.; JARKE, M. (Ed.). *Proceedings of the Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology, Prague, Czech Republic, March 25-27*. [S.l.]: Springer, 2002. (Lecture Notes in Computer Science, v. 2287). ISBN 3-540-43324-4.

HULL, D. A. *Stemming Algorithms - A Case Study for Detailed Evaluation*. set. 25 1996. Disponível em: <http://citeseer.ist.psu.edu/509573.html; http://www.xrce.xerox.com/people/hull/hull/./papers/jasis96.ps>.

HWANG, C. H. Incompletely and imprecisely speaking: Using dynamic ontologies for representing and retrieving information. *Proceedings of the 6th International Workshop on Knowledge Representation meets Databases (KRDB'99)*, p. 14–22, 1999.

JOHANSSON, S. *Some aspects of the vocabulary of learned and scientific English*. [S.l.]: Acta Universitatis Gothoburgensis, Göteburg, Sweden, 1978.

JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Englewood Cliffs, New Jersey: Prentice Hall, 2000. ISBN 0130950696.

KIETZ, J.-U.; VOLZ, R.; MAEDCHE, A. Extracting a domain-specific ontology from a corporate intranet. In: CARDIE, C.; DAELEMANS, W.; NEDELLEC, C.; SANG, E. T. K. (Ed.). *Proceedings of the Fourth Conference on Computational Natural Language Learning and of the Second Learning Language in Logic Workshop*. Lisbon, Portugal: [s.n.], 2000. p. 167–175.

KLYNE, G.; CARROLL, J. J. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. fev. 2004. World Wide Web Consortium, Recommendation REC-rdf-concepts-20040210.

LASSILA, O.; SWICK, R. R. *Resource Description Framework (RDF) Model and Syntax Specification*. 1999. W3C Recommendation. http://www.w3.org/TR/REC-rdf-syntax/.

LIN, C. yew; HOVY, E. *The Automated Acquisition of Topic Signatures for Text Summarization*. jun. 16 2000. Disponível em: <http://citeseer.ist.psu.edu/431638.html; http://nlp3.korea.ac.kr/proceeding/coling2000/COLING/ps/072.ps>.

MAEDCHE, A.; STAAB, S. Discovering conceptual relations from text. In: *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, 2000*. IOS Press, Amsterdam, 2000. Disponível em: <http://www.aifb.uni-karlsruhe.de/WBS/Publ/2000/ecai_amaetal_2000.pdf>.

MAEDCHE, A.; STAAB, S. Ontology learning. In: *Handbook on Ontologies*. [S.l.: s.n.], 2004. p. 173–190.

MARCUS, M.; SANTORINI, B.; MARCINKIEWICZ, M. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, v. 19, n. 2, p. 313–330, 1993.

MAYNARD, D.; CUNNINGHAM, H.; BONTCHEVA, K.; CATIZONE, R.; DEMETRIOU, G.; GAIZAUSKAS, R.; HAMZA, O.; HEPPLE, M.; HERRING, P.; AL. et. *A Survey of Uses of GATE*. 2000. Disponível em: <citeseer.ist.psu.edu/389665.html>.

MINSKY, M. A framework for representing knowledge. In: *Theoretical Issues in Natural Language Processing 1*. [S.l.: s.n.], 1975. See also: paper of the same name in Winston, P. H. (ed.) *The Psychology of Computer Vision*, (1975) New York: McGraw-Hill. Reprinted in Ronald J. Brachman and Hector J. Levesque (eds.) *Readings in Knowledge Representation*, (1985) Los Altos: Kaufman.

PEARSON, K. *On the theory of contingency and its relation to association and normal correlation. Biometric Series No. 1*. [S.l.]: Drapers Co. Memoirs, London, 1904.

Project Management Institute. *A Guide To The Project Management Body Of Knowledge (PMBOK Guides)*. [S.l.]: Project Management Institute, 1996. ISBN 1880410125.

SALTON, G. Developments in automatic text retrieval. *Science*, v. 253, p. 974–980, ago. 1991.

SCHLENOFF, C.; GRUNINGER, M.; CIOCOIU, M.; LEE, J. The essence of the process specification language. *Transactions of the Society for Computer Simulation International*, Society for Computer Simulation International, San Diego, CA, USA, v. 16, n. 4, p. 204–216, 1999. ISSN 0740-6797.

SCHREIBER, G.; WIELINGA, B.; AKKERMANS, H.; VELDE, W. van de; ANJEWIERDEN, A. Cml: The commonkads conceptual modelling language. *Proc. of EKAW'94*, 1994.

SMITH, M. K.; WELTY, C.; MCGUINNESS, D. L. *OWL Web ontology language guide*. [S.l.], 2004. Disponível em: <http://www.w3.org/TR/owl-guide/>.

STANFORD UNIVERSITY SCHOOL OF MEDICINE. 2007. Http://protege.stanford.edu.

STUDER, R.; BENJAMINS, R.; FENSEL, D. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, v. 25, n. 1-2, p. 161–198, MAR 1998.

USCHOLD, M.; GRUNINGER, M. Ontologies: principles, methods and applications. *Knowledge Engineering Review*, v. 11, p. 93–15, 1996.

VALENTE, A. *Legal Knowledge Engineering: A Modelling Approach*. Amsterdam: IOS Press, 1995.

WELTY, C.; GUARINO, N. Supporting ontological analysis of taxonomic relationships. *Data and Knowledge Engineering*, v. 39, p. 51–74, 2001.

WITTEN, I. H.; FRANK, E. *Data Mining: Practical machine learning tools and techniques*. 2nd. ed. [S.l.]: Morgan Kaufmann, 2005.

YULE, G. *The Statistical Study of Literary Vocabulary*. [S.l.]: Cambridge University Press, Cambridge, 1944.