UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA (UESB)
PROGRAMA DE PÓS-GRADUAÇÃO EM LINGUÍSTICA (PPGLIN)

LUIZ FERNANDO CARDEAL DE SOUZA

EASSIGNER: CONCEPÇÃO E MODELAGEM DE UM *SOFTWARE* PARA A AUTOMATIZAÇÃO DE ANOTAÇÕES FILOLÓGICO-LINGUÍSTICAS EM *CORPORA* ELETRÔNICOS USANDO XML

VITÓRIA DA CONQUISTA – BAHIA
2017

LUIZ FERNANDO CARDEAL DE SOUZA

EASSIGNER: CONCEPÇÃO E MODELAGEM DE UM *SOFTWARE* PARA A AUTOMATIZAÇÃO DE ANOTAÇÕES FILOLÓGICO-LINGUÍSTICAS EM *CORPORA* ELETRÔNICOS USANDO XML

Dissertação apresentada ao Programa de Pós-graduação em Linguística, Universidade Estadual do Sudoeste da Bahia, como parte dos requisitos para a obtenção do título de Mestre em Linguística.

Área de Concentração: Linguística

Linha de Pesquisa: Descrição e Análise de Línguas Naturais

Orientadora: Profª. Drª. Cristiane Namiuti Temponi

VITÓRIA DA CONQUISTA – BA

2017

Souza, Luiz Fernando Cardeal de.

S716e Eassigner: concepção e modelagem de um software para a automatização de anotações filológico-linguísticas em corpora eletrônicos usando XML./ Luiz Fernando Cardeal de Souza, 2017.
109f.

Orientador (a): Dra. Cristiane Namiuti Temponi.
Dissertação (mestrado) – Universidade Estadual do Sudoeste da Bahia, Programa de Pós-Graduação em Linguística – PPGLin, Vitória da Conquista, 2017.
Inclui referência F. 75 – 78.

1. Linguística computacional. 2. eAssigner - Software. 3. Revoluções Tecnolinguísticas. 4. Corpora Digitais. I. Temponi, Cristiane Namiute. II. Universidade Estadual do Sudoeste da Bahia, Programa de Pós- Graduação em Linguística. T. III

CDD: 410

Catalogação na fonte: Juliana Teixeira de Assunção – CRB 5/1890
UESB – Campus Vitória da Conquista - BA

Título em inglês: Eassigner: design and modeling software for automating philological-linguistic annotations in electronic corpus using XML

Palavras-chave em inglês: Annotation. Corpus. eAssigner. Computational Linguistics.

Área de concentração: Linguística

Titulação: Mestre em Linguística

Banca examinadora: Profa. Dra. Cristiane Namiute Temponi (Presidente-Orientadora); Prof. Dr. Jorge Viana Santos (UESB); Prof. Dr. Pablo Picasso Feliciano de Faria (UNICAMP)

Data da defesa: 29 de agosto de 2017

Programa de Pós-Graduação: Programa de Pós-Graduação em Linguística.

FOLHA DE APROVAÇÃO

LUIZ FERNANDO CARDEAL DE SOUZA

EASSIGNER: CONCEPÇÃO E MODELAGEM DE UM SOFTWARE PARA A AUTOMATIZAÇÃO DE ANOTAÇÕES FILOLÓGICO-LINGUÍSTICAS EM CORPORA ELETRÔNICOS USANDO XML

Dissertação apresentada ao Programa de Pós-Graduação em Linguística, da Universidade Estadual do Sudoeste da Bahia, como requisito parcial e obrigatório para a obtenção do título de Mestre em Linguística.

Data da aprovação: 29 de agosto de 2017.

Banca Examinadora:

Profa. Dra. Cristiane Namiuti Temponi (Presidente) Ass.:
Instituição: UESB

Prof. Dr. Jorge Viana Santos Ass.:
Instituição: UESB

Prof. Dr. Pablo Picasso Feliciano de Faria Ass.:
Instituição: UNICAMP

AGRADECIMENTOS

Agradeço à minha esposa Izabel por estar sempre presente, pela compreensão e aceitação das minhas incontáveis horas de estudo e de pesquisa e pelo apoio incondicional, principalmente nos momentos mais difíceis.

Agradeço à minha família, pelo carinho, pela compreensão e por ter sempre apoiado minhas escolhas, e ter investido desde cedo na minha formação.

Agradeço à minha professora e orientadora, **Prof. Dra. Cristiane Namiuti Temponi**, pela experiência e conhecimentos transmitidos. Obrigado especialmente pela paciência, pela compreensão, pela disponibilidade, e pelas palavras de incentivo durante toda minha caminhada no mestrado. Obrigado também pela força e pelos incontáveis: "vamos, estamos juntos!".

Agradeço a todos os professores do PPGLin pelos conhecimentos transmitidos, pela compreensão e pela orientação, em especial os professores **Jorge Viana, Adriana Lessa** e **Vera Pacheco**.

Agradeço ao professor **Pablo Faria** pela sugestão de trabalho, pelo incentivo e colaboração em toda a pesquisa.

Agradeço ao colega de mestrado e de trabalho, **Sinval Araújo** pela sugestão do nome do *software* eAssigner e pela sua preciosa ajuda em vários momentos, especialmente na infindável busca por referências.

Agradeço a todos os colegas de Mestrado, em particular a **Lorenna Oliveira e Warley Campos** pelo apoio e carinho e por transmitirem a força e a luz da sua juventude.

Agradeço também aos meus amigos e colegas do IFBA pelo apoio incondicional, palavras de incentivo, compreensão nos momentos em que tive que me dividir entre trabalho e estudo.

Finalmente, e acima de tudo, agradeço a Deus e à Espiritualidade Superior, pela força, pela luz e por me erguerem sempre que precisei.

RESUMO

Este trabalho pretende discutir e ajudar a resolver o problema das inconsistências geradas pela falta de eficiência do processo de edição filológica manual que podem causar ruídos na anotação linguística do corpus, podendo comprometer sua etiquetagem morfossintática e a anotação sintática, causando prejuízo para a pesquisa linguística. Para resolver esse problema, este trabalho discute a importância da utilização da Tecnologia da Informação como ferramenta de auxílio aos trabalhos dos linguistas, particularmente no que se refere à Linguística de Corpus. Assim, ao longo desse trabalho se define a Linguística de Corpus, a Linguística Computacional e são descritas utilizações atuais da Inteligência Artificial (IA) enfatizando os problemas relacionados à Linguística e as estratégias modernas em busca de soluções. A partir daí, descreve a importância do uso de *softwares* de anotação em *corpora* eletrônicos e a decorrente necessidade de automatizar algumas dessas operações através do projeto para desenvolvimento do software. Assim, também foi feita a concepção e modelagem da ferramenta eAssigner, apresentando as suas características, limitações e estágio de desenvolvimento. Para ilustrar a necessidade de uso do *software* são apresentados resultados de alguns testes realizados em amostra de documentos do Corpus DoViC.

PALAVRAS-CHAVE

Anotação. *Corpus*. eAssigner. Linguística Computacional.

ABSTRACT

This research work intends to discuss and help solve the problem of inconsistencies generated by the lack of efficiency of the manual philological editing process that can cause noise in the linguistic corpus annotation, which may compromise its morphosyntactic labeling and syntactic annotation, causing a loss of linguistic research. To solve this problem, this research discusses the importance of the use of Information Technology as a tool to help the work of linguists, particularly with regard to Corpus Linguistics. Thus, throughout this work, we define the terms Corpus Linguistics, Computational Linguistics and current uses of Artificial Intelligence are described emphasizing the problems related to Linguistics and the modern strategies in search of solutions. From there, the research describes the importance of the use of annotation softwares in electronic corpora and the consequent need to automate some of these operations through the project for software development. Thus, the design and modeling of the eAssigner tool was also made, presenting its characteristics, limitations and stage of development. To illustrate the need to use the software are presented results of some tests performed on Corpus DoViC document samples.

KEYWORDS

Annotation. Corpus. eAssigner. Computational Linguistics.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

DOViC	Corpus de Documentos Oitocentistas de Vitória da Conquista
IA	Inteligência Artificial
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Os manuscritos antigos são fontes importantes para o estudo da história de um povo, de uma civilização, pois podem guardar registros de acontecimentos ocorridos em determinada sociedade, conforme enfatizam Almiro Filho e Ximenes (2014). Estudar a escrita antiga é, portanto, de grande interesse para historiadores. De forma similar, os estudiosos da Linguística podem utilizar esses mesmos tipos de documento para compreender, acompanhar a história da língua, a evolução do uso das palavras, alterações de sentido e forma que podem, inclusive, revelar mudanças no plano gramatical, nos diferentes níveis linguísticos: fonológico, morfológico, sintático. De acordo com Namiuti-Temponi, Viana Santos, Leite (2011), a garantia de fidelidade às formas originais dos textos antigos é o pilar de sustentação que qualquer estudo linguístico deve pressupor. Os autores, ao tratar da construção de um corpus de documentos antigos do sudoeste baiano, explicam que:

> A integração entre o tratamento filológico e o computacional na elaboração de corpus para o estudo do português brasileiro é especialmente importante para a preservação e divulgação do patrimônio histórico-linguístico do sudoeste baiano, da Bahia e do Brasil (NAMIUTI-TEMPONI; VIANA SANTOS; LEITE, 2011, p. 1).

As pesquisas manuais nesses textos enfrentam muitas dificuldades, entre elas a dificuldade de acesso aos documentos, a fragilidade e raridade dos documentos e a lentidão para percorrer e comparar um volume representativo de documentos. Possivelmente vem daí a necessidade de unir duas grandes ciências, a Linguística e a Informática, surgindo a Linguística Computacional. Sobre isso, Othero (2014) afirma:

> A Linguística Computacional é a parte da ciência linguística que se preocupa com o tratamento computacional da linguagem, e suas aplicações englobam programas como tradutores automáticos, *chatterbots*, corretores ortográficos e gramaticais, *parsers*, entre outros. O conhecimento linguístico é indispensável para desenvolver tais aplicativos, e somente a interação entre pesquisadores das duas áreas – Linguística e Informática – pode conquistar bons resultados (OTHERO, 2014, p. 1)

A construção de *corpora* eletrônicos para fins de pesquisa linguística envolve tratamento especial dos documentos para viabilizar seu processamento eletrônico. Conforme Aluísio e Almeida (2006), esse tratamento é chamado de anotação e pode ocorrer em nível sintático, semântico, discursivo etc. As autoras acrescentam que essa anotação pode ser feita manualmente (por linguistas), automaticamente (por Processamento de Linguagem Natural – PLN) ou semi-automaticamente, quando é feita por PLN e complementada por linguistas.

Em oposição ao procedimento de anotação, Leech (2004) afirma que existem pesquisadores que preferem não o adotar, pois segundo tais pesquisadores, as anotações podem adulterar os *corpora* com informações suspeitas e mesmo erros do anotador, sendo os *corpora* não anotados representações dos documentos em seu estado "puro", fidedigno. Apesar do problema da "impureza", muitos pesquisadores (possivelmente a maioria) preferem utilizar corpora anotados por sua potência na busca de dados. Para contornar o problema da fidedignidade, fica evidente a necessidade de utilizar mecanismos de anotação que mantenham/resgatem as formas originais dos textos, ao mesmo tempo em que agregam valor aos documentos. Uma solução viável para essa situação, conforme explica Paixão de Sousa (2006), foi o desenvolvimento de uma técnica de anotação/marcação baseada na linguagem XML que permite editar os documentos e, ao mesmo tempo, recuperar a forma original dos textos. Sobre isso, Namiuti-Temponi e Costa (2014), afirmam que a linguagem XML permite descrever qualquer tipo de dado por ter um padrão aberto e permitir a interoperabilidade, facilitando o tratamento dos documentos tanto por linguistas quanto por programas de computador.

Um dos principais *corpora* em uso no Brasil, o *Corpus Tycho Brahe*[1]. As anotações de edição no *Corpus Tycho Brahe* são feitas atualmente através da ferramenta eDictor que foi desenvolvida a partir de demandas observadas na construção desse mesmo *corpus* (PAIXÃO DE SOUZA; KEPLER; FARIA, 2009). Apesar da eficiência do eDictor, a ferramenta tem alguns limites, sendo a principal limitação do eDictor localizada no fato da "edição" ser manual, contando apenas com um simples procedimento de "aplicar a todos" para os casos em que a edição de um dado *token* é sempre a mesma. A falta de capacidade de treinamento do eDictor torna a tarefa do editor humano repetitiva.

A principal motivação do presente trabalho é a falta de uma ferramenta de análise, um *software* capaz de aprender as decisões de edição feitas através da ferramenta eDictor (ou outra que utilize etiquetas padrão XML) e que seja capaz de, utilizando técnicas de Inteligência Artificial, transferir as marcações e decisões de edição aprendidas para todo o acervo, reduzindo o trabalho repetitivo e as possibilidades de enganos/erros na edição do texto.

Para ilustrar a necessidade de desenvolver uma ferramenta que aprenda as decisões e marcações das edições filológicas de textos antigos no meio eletrônico, foram extraídos e estudadas as edições de oito cartas de alforria emitidas em favor de escravos, documentos do

[1] Disponível em http://www.tycho.iel.unicamp.br/corpus/

Corpus de Documentos Oitocentistas de Vitória da Conquista e região (DOViC). Os documentos selecionados foram coletados e transpostos para o meio digital seguindo o método LAPELINC descrito em Namiuti e Santos 2015. Em seguida foram editados e revisados por linguistas utilizando o *software* eDictor. Os seguintes resultados foram observados:

No documento 1 pode-se observar que o nome “Jose” foi modernizado para “José” na edição número 13, mas não foi modernizado na edição número 23 do mesmo documento, conforme demonstrado a seguir:

```
<w id="13">
<o>Jose</o>
<e t="mod">José</e>          Feita uma MODERNIZAÇÃO
<m v="NPR"/>
</w>

.....

<w id="23">
<o>Jose</o>
<m v="NPR"/>
</w>
```

Observem o que acontece ao observar vários documentos diferentes:

Documento 1

```
<w id="28">
<o>assi-<bk t="l" id="bk_3"/> gnado</o>
<e t="jun">assi-gnado</e>
<e t="gra">assignado</e>
<e t="mod">assinado</e>
<m v="VB-AN"/>
</w>
```

Documento 3

```
<w id="35">
<o>assignado</o>
</w>
```

Documento 4

```
<w id="275">
<o>assignado</o>
<e t="mod">assinado</e>
<m v="VB-AN"/>
```

```
</w>
```

Documento 6

```
<w id="310">
<o>assignado</o>
<e t="mod">assinado</e>
<m v="VB-AN"/>
</w>
```

A palavra “assignado” teve a sua grafia modernizada manualmente para “assinado” em três documentos diferentes (1, 4 e 6), dentro da nossa amostra de oito documentos. Observe também que no documento 3 a grafia não foi modernizada. Em uma pequena amostra do *corpus*, já é possível perceber a incosistência gerada pela tarefa manual de edição de documentos e a importância do desenvolvimento e utilização de uma ferramenta de edição automática, capaz de aprender as decisões de edição dos linguistas e aplicá-las em novos documentos. Essa é a motivação para a criação do eAssigner, uma ferramenta capaz de aprender padrões de edição em marcações XML e aplicá-los a novos documentos.

É preciso ressaltar que uma ferramenta de edição automática, como o eAssigner, precisa lidar com casos de ambiguidade, em que uma mesma forma arcaica pode ter duas ou mais formas modernizadas possíveis, a depender do contexto de ocorrência. Digamos, como exemplo hipotético, que se encontre a forma “õde” cuja modernização seja, em alguns casos, “onde” e noutros casos “aonde”. Neste caso, é preciso levar o contexto em consideração para determinar qual é a edição mais provável.

Assim, pode-se determinar o problema de pesquisa a ser resolvido.

1.1 PROBLEMA DE PESQUISA

O processo de edição filológica manual, além de lento, é mais sujeito a produção de inconsistências.

As inconsistências geradas pela falta de eficiência do processo de edição filológica manual podem causar ruídos na anotação linguística do corpus, podendo comprometer sua etiquetagem morfossintática e a anotação sintática, causando prejuízo para a pesquisa linguística.

1.2 HIPÓTESE

Automatizar o processo de edição filológica em corpora digitais, tornará a tarefa de edição mais rápida e eficiente, diminuindo as inconsistências e consequentemente a incidência de erros na anotação linguística automática aplicada ao texto editado, gerada pelas inconsistências e erros na edição filológica.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

O objetivo da pesquisa aqui relatada é conceber e modelar uma ferramenta para automatizar o processo de anotação de *corpora* digitais, utilizando técnicas de Inteligência Artificial e *Softwares* Livres. Denominamos tal ferramenta de eAssigner.

1.3.2 Objetivos Específicos

I. Descrever e avaliar o processo de edição filológica de corpora, focalizando as dificuldades e avanços na área.
II. Descrever as características das ferramentas de edição e anotação em *corpora*, fazendo uma análise comparativa de características e limitações.
III. Comparar as tecnologias de mineração de texto e *parsing* nas ferramentas existentes.
IV. Pesquisar o estado da arte da Inteligência Artificial, particularmente o aprendizado de máquina.
V. Modelar o anotador automático eAssigner para documentos em formato XML.

1.4 METODOLOGIA

1.4.1 Dimensões da Pesquisa

A pesquisa foi realizada inicialmente a partir de estudo bibliográfico nas fontes referenciadas e fontes acrescentadas ao longo do trabalho.

O método de pesquisa utilizado é predominantemente qualitativo, embora esteja prevista a abordagem de alguns aspectos quantitativos. Segundo Lakatos e Marconi (2006) e Appolinário (2012), uma pesquisa predominantemente qualitativa prevê a coleta de dados a partir de interações sociais do pesquisador com o fenômeno pesquisado, enquanto a pesquisa predominantemente quantitativa prevê a mensuração de variáveis predeterminadas.

Quanto à finalidade, a pesquisa é classificada como aplicada, por gerar um produto aplicável ao atendimento de necessidades do mercado, conforme Appolinário (2012).

Quanto ao tipo, de acordo com Wazlawick (2009) e Appolinário (2012), a pesquisa pode ser classificada como experimental por analisar aspectos da realidade interferindo deliberadamente em alguns eventos em busca de explicações ou de melhoria de resultados.

Quanto à estratégia em relação à fonte de informação, a pesquisa é classificada como pesquisa de campo, pois, segundo Lakatos e Marconi (2006), envolve levantamento de dados não bibliográficos, especialmente na fase de pesquisa do estado da arte em Linguística Computacional, particularmente a área de *software* de anotação. Uma pesquisa de campo também tem a sua fase documental, conforme Appolinário (2012), o que não a torna pesquisa documental.

1.4.2 Procedimentos Metodológicos

O processo de desenvolvimento do presente de trabalho de pesquisa envolveu os seguintes procedimentos aqui apresentados como etapas:

- Levantamento bibliográfico inicial;
- Pesquisa do estado da arte em desenvolvimento de *softwares* de tratamento de *corpora*;
- Pesquisa de *softwares* de anotação, similares ao eDictor;
- Projeto e modelagem do *software* eAssigner:
    - Definição de ferramentas e de plataforma de desenvolvimento;
    - Definição e construção do Diagrama de Casos de Uso;
    - Construção de Diagramas de Classe;
    - Definição da forma de utilização;
    - Construção de protótipo funcional;
    - Avaliação de protótipo e ajustes

1.5 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está organizado em seis seções com o seguinte conteúdo. A presente seção, seção 1 – Introdução - contextualiza o projeto e apresenta os seus objetivos, além dos aspectos metodológicos. A seção 2 - A Linguística na Era Digital – descreve importantes aspectos das Humanidades Digitais, descreve conceitos de *corpora* digitais, além de definir a Linguística Computacional e as suas áreas de atuação. A seção 3 – *Softwares* de Anotação – conceitua esses *softwares* e apresenta uma comparação entre quatro *softwares* testados e o eDictor. A seção 4 – Modelagem do *Software* eAssigner – descreve a estrutura do *software*, apresentando a sua modelagem, aspectos de desenvolvimento de protótipo parcialmente funcional. A seção 5 – Testes e Discussão dos Resultados preliminares - apresenta resultados de testes preliminares do protótipo feitos com uma base resuduzida. A seção 6 – Considerações Finais – discute resultados e apresenta sugestões de continuação do projeto em trabalhos futuros. O trabalho também inclui dois apêndices para discutir mais profundamente alguns aspectos tecnológicos: Apêndice A – Inteligência Artificial – apresenta o estado da arte dessa importante área da computação, fazendo a sua relação com a Linguística Computacional. O Apêndice B – Noções sobre Software Livre – caracteriza esse tipo de *software*, discutindo as principais vantagens e desvantagens na sua utilização.

2 A LINGUÍSTICA NA ERA DIGITAL

A presente seção descreve a importância das influências dos recursos computacionais da era digital na Linguística. Inicia caracterizando o que se decidiu chamar de Humanidades Digitais e depois apresenta um breve histórico sobre as revoluções tecnolinguísticas, que culminaram nas tecnologias e processos envolvidos na construção de *corpora* digitais, enfatizando a aplicação de técnicas da Inteligência Artificial (IA) na construção de ferramentas de anotação de *corpora* e outros usos na Linguística.

2.1 HUMANIDADES DIGITAIS

Dacos (2011) no Manifesto das Humanidades Digitais, define:

> A opção da sociedade pelo digital altera e questiona as condições de produção e divulgação dos conhecimentos;
> [...] as *digital humanities* referem-se ao conjunto das Ciências humanas e sociais, às Artes e às Letras. As humanidades digitais não negam o passado, apoiam-se, pelo contrário, no conjunto dos paradigmas, *savoir-faire*[2] e conhecimentos próprios dessas disciplinas, mobilizando simultaneamente os instrumentos e as perspectivas singulares do mundo digital;
> As *digital humanities* designam uma transdisciplina, portadora dos métodos, dos dispositivos e das perspectivas heurísticas ligadas ao digital no domínio das Ciências humanas e sociais. (DACOS, 2011).

O referido manifesto prega o livre acesso aos dados e metadados que devem ser registrados de tal forma que possam ser utilizados de forma livre e interoperável. Essa declaração aponta para a tendência a registrar os dados utilizando-se formatos abertos. Veja a subseção *Software* Livre para melhor compreender esse conceito.

Outra importante conclamação do Manifesto também parece apontar para as características dos *Softwares* Livres:

> Chamamos à construção de ciber-infra-estruturas evolutivas que respondam a necessidades reais. Estas ciber-infra-estruturas construir-se-ão de maneiras iterativas, apoiando-se sobre a constatação de métodos e de abordagens comprovadas nas comunidades de pesquisa (DACOS, 2011).

Uma das características dos *Softwares* livres é o desenvolvimento comunitário de *softwares*, sem proprietários, o que parece harmonizar perfeitamente com o Manifesto.

---

[2] *Savoir-faire*: conhecimento para realizar uma tarefa, *know-how*.

Girão (2016), definindo as Humanidades Digitais, explica:

> [..] alia as disciplinas tradicionais das áreas humanas (história, filosofia, linguística, literatura, arte, arqueologia, música etc.) com as ferramentas oferecidas pela ciência da computação (hipertexto, hipermídia, visualização de dados, realidade virtual, recuperação de informações, mineração de dados, estatística, mapeamento geográfico etc.) (GIRÃO, 2016, p. 2).

O mesmo autor afirma que ainda não existe uma clara definição da abrangência e do escopo das Humanidades Digitais. Entretanto, prossegue afirmando que fica claro que os seus projetos dependem fortemente da colaboração entre os pesquisadores das disciplinas mencionadas.

O autor afirma que o conceito de Humanidades Digitais (não com esse nome), surgiu em 1946 quando o padre jesuíta Roberto Busa decidiu indexar toda a obra de São Tomás de Aquino, palavra por palavra. A tecnologia computacional da época não poderia viabilizar o projeto, mas o padre tomou como lema uma frase de um famoso pôster da IBM: "As dificuldades a gente resolve de imediato; o impossível leva um pouco mais de tempo" [3].

Paixão de Sousa (2013, p. 114), também destaca esse trabalho, o *Corpus Thomisthicum*[4], datando-o de 1942, portanto contemporâneo à invenção do computador. A autora acrescenta que, além do trabalho de indexação, também foi iniciado um trabalho de anotação linguística na mesma obra. A autora menciona que os trabalhos realizados nessa área entre os anos 1900 e 2000 levaram alguns pesquisadores a afirmarem o surgimento do campo da Filologia Digital (ou *e-Philology*), ou seja, a "[...] filologia realizada por meio de ferramentas computacionais." (PAIXÃO DE SOUSA, 2013, p. 114).

Girão (2016), sobre uma importante aplicação das Humanidades Digitais, menciona:

> Uma aplicação, somente possível com os atuais recursos computacionais, cada vez mais utilizados nessa área, é a denominada Leitura Remota. Trata do uso de enormes acervos de documentos, livros e similares, para correlacionar suas informações e obter daí conclusões sobre perguntas que tenham utilidades específicas (GIRÃO, 2016, p. 3).

Essa aplicação *Leitura Remota* mencionada pelo autor, pelas características e utilização mencionadas, necessita utilizar *Corpora* digitais. O autor complementa o raciocínio:

> Os historiadores cada vez mais fazem uso dessas ferramentas. Por exemplo, poderíamos rastrear o processo de libertação dos escravos e suas consequências

[3] Essa frase também era usada em um pôster das Forças Especiais do Exército Norte-Americano.
[4] Disponível em <www.corpusthomisticum.org> – acesso em 02 maio 2017.

> analisando e correlacionando os jornais, livros e revistas, os contratos de trabalho que vieram logo depois, os registros escolares, as certidões de nascimento para acompanhar sua migração pelo país, o censo e assim por diante (GIRÃO, 2016, p. 4)

Para executar o tipo de pesquisa indicado pelo autor, além dos *Corpora* digitais, é preciso dispor de ferramentas de anotação e ferramentas de pesquisa em corpora anotados. Mais uma vez, pode-se perceber a forte relação entre as Humanidades Digitais e a Linguística de *Corpus*.

2.2 *CORPORA* DIGITAIS

Essa subseção descreve a importância da utilização de *corpora* nos estudos linguísticos. Para isso, descreve as revoluções tecnolinguísticas e os principais processos de criação de *corpora* digitais.

2.2.1 As Revoluções Tecnolinguísticas

De acordo com Auroux (1998):

> A linguagem humana é um fenômeno profundamente ligado à evolução corporal dos hominídeos. Embora se trate de uma manifestação eminentemente social, ela é irredutivelmente, enquanto tal, uma manifestação do comportamento individual que coloca em jogo o corpo e o domínio de um grande número de controles psicomotores. Ora, a linguagem humana não parece, no entanto, irredutivelmente ligada à especificidade do canal áudio-oral. O fenômeno mais espantoso é, com efeito, a existência de suportes transpostos (AUROUX, 1998, p. 63).

O autor define suporte transposto como "qualquer substituição do suporte áudio-oral em proveito de um outro, enquanto permanece a identidade da linguagem" (AUROUX, 1998, p. 63). Dessa forma, o homem tem desenvolvido suportes transpostos como forma de comunicação com e sem a presença do corpo humano.

2.2.1.1 A Primeira Revolução Tecnolinguística: A Escrita

De acordo com Auroux (1998), dentre os diversos suportes transpostos criados, o mais importante foi o "suporte gráfico, não somente porque utiliza a bidimensionalidade do espaço plano, mas ainda porque é o primeiro suporte que permitiu à fala humana subsistir sem a presença de som emissor". (AUROUX, 1998, p. 64).

Assim, o autor defende que a escrita foi a primeira revolução tecnolinguística da humanidade. Não iremos aqui nos deter na cronologia dos sistemas de escrita por estar fora do escopo desse trabalho, entretanto, vale a pena apontar três estados do desenvolvimento da escrita:

> Estado pictográfico, em que a escrita representa diretamente o aspecto exterior dos objetos do mundo; o estado ideográfico, em que ela codificaria as ideias e não as palavras; o estado fonético, enfim, e que são os próprios sons das linguagens que são codificados (AUROUX, 1998, p. 65).

Auroux (1998, p. 70-73) menciona que é incontestável que o escrito só aparece e se mantém em sociedades fortemente hierarquizadas. Que a escrita apareceu primitivamente na esfera econômica, e que a escrita esteve inicialmente ligada à contagem e repartição dos bens; mas esclarece que nem sempre isso traduz a verdade, uma vez que existiram sociedades que não usavam a escrita como forma de contagem, que é o caso da sociedade asteca. Mas pode-se concluir que a língua escrita esteve e está fortemente ligada a todos os aspectos significativos da sociedade. Transita pelo mítico e cultural, passa pelo econômico e político, associa-se ao religioso, interage com o social e tudo sem perder a ligação e relação direta com o poder. Saber escrever é sinônimo de domínio em quase todo o mundo se não for nele todo. Ao observar com mais clareza, esse conceito de domínio pela escrita se modificou com o passar da história. Abandonou o domínio físico (força bruta), adotou uma postura de domínio intelectual derivando para o tipo mais sutil de domínio, porém mais significativo que é o ideológico (aquele visto na nossa sociedade).

2.2.1.2 A Segunda Revolução Tecnolinguística: A Gramatização das Diferentes Línguas

De acordo com Auroux (2001), na sua obra "A revolução tecnológica da gramatização", o processo de gramatização das línguas modificou de forma significativa a comunicação humana e deu ao Ocidente um meio de conhecimento e de dominação sobre as outras culturas do planeta.

O estudo de Auroux inicialmente descreve o nascimento das metalinguagens. De acordo com Houaiss (2009), "metalinguagem é uma linguagem (natural ou formalizada) que serve para descrever ou falar sobre uma outra linguagem, natural ou artificial". Nessa reflexão sobre as metalinguagens, Auroux distingue dois tipos de saberes sobre a linguagem: um saber epilinguístico e um saber metalinguístico. O saber epilinguístico, segundo o autor, "é o saber

inconsciente que todo locutor possui de sua língua e da natureza da linguagem". É este saber que nos permite, por exemplo, entender e produzir piadas e jogos de linguagem. Já o saber metalinguístico é "construído e manipulado enquanto tal com a ajuda de uma metalinguagem". É este saber, conforme Auroux (2001), que permite que possamos não apenas entender e produzir piadas e jogos de linguagem, mas também refletir sobre o funcionamento das piadas e dos jogos de linguagem.

O surgimento da escrita foi decisivo para a passagem dos saberes epilinguísticos para os saberes metalinguísticos. Um ponto importante que o autor observa é que não se trata de considerar a escrita como a origem de uma tradição de saber linguístico. Isso porque o que se tem não é uma origem, mas um processo. Processo esse que pode ser longo e que passa, necessariamente, pela escrita. Segundo Auroux (2001), o que faz deslanchar verdadeiramente uma reflexão linguística é a qualidade que se constitui através de relações de contraste (a alteridade), considerada do ponto de vista da escrita.

Devido à dificuldade de se ler textos antigos ou textos estrangeiros tornou-se importante compreender o funcionamento dessas línguas para poder traduzi-las. A esse respeito, o autor observa que a primeira análise gramatical nasceu da necessidade de se ler e compreender textos e não da necessidade de se falar uma língua qualquer. De uns tempos para cá, a gramática passou a ser uma técnica escolar destinada "às crianças que dominam mal sua língua" ou que aprendem uma língua estrangeira. Isso se deve tanto ao desenvolvimento do sistema escolar quanto ao da gramática. Em tempos remotos, nunca se teve espontaneamente a ideia de fazer uma gramática para aprender a falar.

Auroux apresenta duas causas da gramatização das línguas. A primeira delas é a necessidade de aprendizagem de uma língua estrangeira em um contexto onde já existe uma tradição linguística. A segunda concerne essencialmente à política de uma língua dada, e pode se reduzir a dois interesses: organizar e regular uma língua literária; e desenvolver uma política de expansão linguística de uso interno ou externo. Auroux faz uma interessante afirmação sobre isso: que durante o desenvolvimento das concepções linguísticas europeias desde o século V de nossa era até o fim do século XIX, tem-se o desenrolar de um processo único em seu gênero: a gramatização massiva, a partir de uma só tradição linguística inicial (a tradição greco-latina), das línguas do mundo. Um marco importante durante esse período, destacado pelo autor, é o Renascimento Europeu (XIV a XVI), que é um momento em que são produzidos, em uma quantidade extremamente significativa, dicionários e gramáticas de diversas línguas do mundo e não somente dos vernáculos europeus, na base da tradição greco-latina.

Essa gramatização constitui a segunda revolução técnico-linguística, que tem consideráveis consequências práticas para a organização das sociedades humanas. Trata-se propriamente de uma revolução tecnológica sem dúvida muito importante para a história da humanidade. A gramatização é definida por Auroux como um processo que conduz a descrever e a instrumentar uma língua na base de duas tecnologias, que são ainda hoje os pilares de nosso saber metalinguístico: a gramática e o dicionário. Desse ponto de vista, a gramática e o dicionário não são vistos como simples descrições da linguagem natural. Eles são concebidos também como instrumentos linguísticos. O aparecimento dos instrumentos linguísticos não deixa intactas as práticas linguísticas humanas.

A gramatização modificou profundamente a ecologia da comunicação e o estado do patrimônio linguístico da humanidade. As línguas, pouco ou menos "não-instrumentalizadas", foram mais expostas ao que se convém chamar "linguicídio", quer seja ele voluntário ou não. O processo de gramatização corresponde a uma transferência de tecnologia de uma língua para outras línguas, transferência essa que não é totalmente independente de uma transferência cultural mais ampla. Essa transferência pode ser de dois tipos. Ela pode ser uma endotransferência ou uma exotransferência. O autor especifica esses dois tipos de transferência pelos termos endogramatização (correspondente a uma endotransferência) e exogramatização (correspondente a uma exotransferência). A gramatização espontânea (fora de transferência) corresponde para o autor a uma endogramatização.

Um caso de endogramatização é a transferência de tecnologia das tradições linguísticas gregas para a língua latina, pelos latinos. A gramatização dos vernáculos europeus a partir das tradições latinas pelos próprios europeus também é um caso de endogramatização. Um caso de exogramatização é a transferência de tecnologia do português para as línguas indígenas, pelos portugueses e não pelos indígenas.

Outro ponto extremamente importante a ser destacado no estudo do autor é a sua observação de que a gramática não surgiu de uma necessidade didática. Ele afirma, a esse respeito, que as crianças gregas ou latinas, que frequentavam a escola, já sabiam sua língua, sendo a gramática só uma etapa do acesso à cultura escrita. Já para um europeu do século IX, o latim era antes de tudo uma segunda língua que ele devia aprender. Somente com a constituição das nações europeias é que a gramática passou a ser utilizada para fins de aprendizagem da própria língua. Naquele momento, houve uma profunda transformação das relações sociais. A expansão das nações implicou uma situação de luta entre elas, o que se traduziu, ao final, por uma concorrência entre as línguas.

Uma vez que o surgimento da escrita foi o pilar da construção de saberes metalinguisticos e por consequência a base do processo de criação de uma tradição linguística (AUROUX, 2001), vale ressaltar que o suporte gráfico da estrita necessitou e necessita de aparatos/suportes materiais, que caracterizam os documentos históricos, seus limites e possibilidades. O aparato material físico, a exemplo do papel, por ser físico, traz algumas complexidades como a tridimensionalidade, a imutabilidade e não padronização. Com o advento da tecnologia eletrônica e computacional, surgem novos suportes para as fontes documentais, a exemplo do digital, trazendo com ele novas complexidades, possibilidades e limites (NAMIUTI; SANTOS, 2015). Ao relacionar os limites e as possibilidades dos aparatos/suportes materiais que figuraram na história da humanidade com as revoluções tecnolinguísticas, notamos que, diferentemente das duas primeiras revoluções a terceira revolução tecnolinguistica, que exploraremos a seguir, a mecanização da linguagem, foi possibilitada pela evolução do aparato/suporte material.

2.2.1.3 A Terceira Revolução Tecnolinguística: A Mecanização da Linguagem

De acordo com Auroux (1998, p. 289), “a terceira grande revolução nesse domínio é a do tratamento eletrônico da informação apresentada em linguagem natural”. O autor acrescenta que essa revolução se apoia nas demais e as complementa.

Ainda segundo Auroux (1998, p. 290), “a importância presente e futura dos efeitos dessa revolução tecnolinguística foi objeto de avaliações contrastadas, até violentamente opostas sobre o tema das possibilidades e limites daquilo que chamamos a Inteligência Artificial (IA)”.

Um dos problemas apontados pelo autor é a questão da intencionalidade, pois, por trás da tela de um computador não há uma pessoa que possa se preocupar ou se empenhar para que algo seja realizado. A intencionalidade implica em consciência, pelo que se pode compreender da argumentação apresentada.

Auroux (1998) aponta a área de tradução automática como uma das primeiras onde se tentou utilizar o computador em Linguística. A teoria vigente na época do início dos trabalhos (1949) é que seria possível fazer uma tradução palavra por palavra de uma língua para outra. Sabemos que esse processo é inviável, pois as palavras podem ter significados diferentes de acordo com o contexto e marcas culturais devem ser consideradas; assim, um processo de tradução implica necessariamente na análise linguística do texto. Hoje dispomos de tradutores

automáticos e sabemos que, apesar terem evoluído bastante, não é possível ainda confiar 100% nas traduções.

A evolução dos algoritmos computacionais e do poder de processamento dos computadores, aliados às necessidades de processamento automático na área de Linguística gerou uma nova área de pesquisa: a Linguística Computacional.

2.2.2 Linguística Computacional

Conforme Othero e Menuzzi (2005, p. 22), "a área responsável pela investigação do tratamento computacional da linguagem e das línguas naturais é conhecida como 'Linguística Computacional'". Vieira e Lima assim a definem:

> A Linguística Computacional pode ser entendida como a área de conhecimento que explora as relações entre linguística e informática, tornando possível a construção de sistemas com capacidade de reconhecer e produzir informação apresentada em linguagem natural (VIEIRA; LIMA, 2001, p. 1).

Uma área da Linguística Computacional que tem despertado muito interesse dos linguistas é a Linguística de *Corpus*. De acordo com Othero e Menuzzi (2005, p. 22).

> A linguística de *corpus* preocupa-se basicamente com o trabalho a partir de *corpora* eletrônicos, isto é, grandes bancos de dados que contenham amostras de linguagem natural. Essas amostras podem ser de diferentes fontes como linguagem falada, linguagem escrita literária, textos de jornal, falas de crianças em fase de desenvolvimento etc. Esses corpora podem ser utilizados para o estudo de fenômenos linguísticos e sua ocorrência em grandes amostras de uma determinada língua (OTHERO; MENNUZZI, 2005, p. 22).

Ainda segundo Othero e Menuzzi, outra área de grande importância é o Processamento de Linguagem Natural (PLN):

> A área de PLN preocupa-se com o estudo da linguagem voltado para a construção de softwares e sistemas computacionais específicos, como tradutores automáticos, chatterbots, parsers, reconhecedores automáticos de voz, geradores automáticos de resumos etc (OTHERO; MENUZZI, 2005, p. 22-23).

Os sistemas computacionais que envolvem PLN, mencionados pelos autores, estão presentes no nosso dia a dia, muitas vezes passando despercebidos. Seguem alguns exemplos e descrições desses sistemas:

Chatterbot: Um *chatterbot* ou *chatbot* é um robô de conversação, ou seja, um programa destinado a simular uma conversação com um cliente através de reconhecimento de

palavras chave. Alguns são utilizados como atendentes virtuais em bancos, operadoras de telefonia etc. A tecnologia de reconhecimento de voz está atualmente em um estágio excelente, mas para que o programa funcione bem, é preciso que o cliente fale com alguma objetividade de forma que o robô reconheça as palavras chave. Um bom programa deve ser capaz de perceber um eventual aumento de volume na voz do cliente e passar a ligação para um atendente real, uma vez que isso pode indicar que o cliente está gritando. De acordo com Chatbots (2015) o primeiro *chatterbot* foi o programa Eliza[5] que simulava uma conversa entre um paciente e a psicóloga (Eliza). Utilizava técnicas de Inteligência Artificial para formular perguntas utilizando as frases do interlocutor.

Parser: De acordo com Othero e Menuzzi (2005, p. 39), "um *parser* é um programa capaz de classificar morfossintaticamente as palavras e expressões de sentenças em determinada língua e, principalmente, de atribuir às sentenças a sua estrutura de constituintes, baseando-se em um modelo formal de gramática". Esse tipo de programa é de grande utilidade para a anotação morfossintática automática ou semiautomática em *corpora* digitais.

Reconhecimento de voz: O Sistema de Reconhecimento de Voz também está presente em muitos aparelhos celulares, pois dispomos de aplicativos instalados que nos permitem ditar textos e eles reconhecem as palavras, fazem buscas na Internet ou enviam mensagens sem que precisemos digitar. Esse tipo de ferramenta é particularmente útil para pessoas que têm deficiência visual ou alguma deficiência motora.

Outra aplicação bastante popular da IA é o Sistema de Posicionamento Global, mais conhecido pela sigla GPS. Esses permitem localizar endereços, traçar rotas para veículos (ou a pé) com orientações verbais e visuais.

Pelas aplicações citadas, pode-se perceber a evolução do processo de Mecanização da Linguagem como uma das aplicações da Inteligência Artificial, o que faz com que as suas técnicas sejam de grande importância na construção da ferramenta eAssigner. Por isso, esse assunto é detalhado no Apêndice B - INTELIGÊNCIA ARTIFICIAL.

[5] O programa Eliza foi o primeiro robô de conversação e foi desenvolvido em 1966 no MIT pelo pesquisador Joseph Weizenbaum (Chatbots, 2015).

2.2.3 A Construção De *Corpora*

Um dos problemas de grande interesse para os estudos em linguística histórica está ligado ao manuseio e tratamento de *corpora*: trata-se da necessidade de construir bibliotecas de documentos com anotações conforme Faria e Galves (2016) explicam:

> *Corpora* anotados são importantes em todos os ramos da linguística, uma vez que constituem bases de dados perenes sobre as quais se podem efetuar análises qualitativas e quantitativas de vários tipos, que complementam outras abordagens como o recurso à intuição dos falantes ou ainda estudos baseados em experimentos, prática corrente em aquisição da linguagem e cada vez mais em análises sintáticas. Em linguística histórica, uma vez que não há falantes nativos disponíveis, os *corpora* são indispensáveis (FARIA; GALVES, 2016, p. 303).

A construção desses *corpora* anotados deve ser feita de forma a torná-los acessíveis através de programas de busca e, para tanto, investe-se na tarefa de transcrição e edição das fontes documentais. Faria e Galves (2016, p. 302), enfatizam que "[...] esta construção, atualmente, é semiautomática e envolve, geralmente, programas de computador que implementam algoritmos de aprendizagem (de máquina) ".

O processo de construção de corpora eletrônicos mediante a edição de textos antigos envolve várias operações como: junção ou segmentação de termos, modernização de grafia, expansão de abreviaturas, correção etc. Como cada operação dessas ocorre várias vezes ao longo de um texto, o processo tende a se tornar repetitivo e sujeito a erros. Além disso, operações feitas em um documento possivelmente terão que ser repetidas em outros documentos do mesmo *corpus*, o que já justifica a necessidade de ferramentas computacionais para auxílio dos pesquisadores.

A fidelidade e a confiabilidade dos *corpora* eletrônicos em relação aos documentos originais são alguns dos pilares de sustentação dos estudos linguísticos, conforme enfatizam Paixão de Sousa (2006), Namiuti-Temponi, Viana Santos e Leite (2011):

> [...]os estudos históricos realizados com base em textos antigos dependem, antes de tudo, da garantia da fidelidade às formas originais dos textos – sendo este o pilar de sustentação que qualquer estudo linguístico, em qualquer quadro teórico, deve pressupor. No caso dos *corpora* eletrônicos, esse pressuposto fundamental precisa ser integrado com requerimentos impostos pela vertente computacional e linguística dos estudos – tais sejam: o arquivo virtual/digital, a confiabilidade e durabilidade do código, a necessidade de quantidade, agilidade e automação no trabalho de organização e seleção de dados (NAMIUTI-TEMPONI; VIANA SANTOS; LEITE, 2011, p. 2833).

Para os autores, é necessário produzir os *corpora* de forma a facilitar o tratamento computacional, mas o processo deve ser feito de tal maneira que seja mantida a forma original de cada documento, ou de uma forma que seja possível recuperar a forma original, conforme enfatiza Paixão de Sousa (2006).

Sobre o processo de construção de *corpora*, Aluísio e Almeida (2006), propuseram uma metodologia com três estágios principais: **projeto do *corpus***, **compilação e tratamento dos textos** e **anotação**, descritos em seguida.

2.2.3.1 Projeto do *Corpus*

A etapa inicial do projeto do *corpus* é a seleção dos textos, conforme explicam Aluísio e Almeida (2006):

> Inicialmente, procede-se à seleção dos textos pertinentes e relevantes para a pesquisa. Para esta etapa, a definição do tipo de corpus que está se compilando é importante; outras decisões dizem respeito ao seu tamanho e à sua composição em termos dos textos existentes bem como dos gêneros aos quais eles pertencem (ALUÍSIO; ALMEIDA, 2006, p. 160).

Ainda nessa etapa, prosseguem as autoras, é importante decidir sobre aspectos como gênero dos textos (informativo, científico, religioso etc.) e tamanho do *corpus*. O tipo de *corpus* também deve ser considerado: oral, escrito ou ambos.

Aluísio e Almeida (2006) levantam importantes aspectos que devem ser considerados durante a seleção de textos e projeto de um *corpus*: autenticidade, representatividade, balanceamento, amostragem, diversidade e tamanho, descritos a seguir:

Autenticidade: Aluísio e Almeida (2006) definem autenticidade como: devem ser escritos em linguagem natural e devem ter sido produzidos por falantes nativos da língua. Sardinha (2000, p. 336), acrescenta que os textos autênticos são "aqueles que existem na linguagem e que não foram criados com o propósito de figurarem no *corpus*".

Representatividade: Sinclair (2005, p. 12), corroborado por Aluísio e Almeida (2006), afirma que, embora o conceito de representatividade seja vago, um *corpus* será representativo se os textos incluídos espelharem as características linguísticas da comunidade cuja língua está sendo analisada.

Balanceamento: embora esse conceito também seja vago, Aluísio e Almeida assim especificam um *corpus* balanceado:

> [...] deve ter um equilíbrio de gêneros discursivos (informativo, científico, religioso, etc.), ou de tipos de textos (artigo, editorial, entrevista, dissertação, carta, etc.), ou de títulos, ou de autores, ou de todos esses itens juntos, desde que as escolhas sejam adequadas à pesquisa que se pretende realizar, demonstrando que os textos foram escolhidos criteriosamente [...] (ALUÍSIO; ALMEIDA, 2006 p. 159).

Amostragem: Conforme Sinclair (2005, p. 10), um *corpus* deve incluir uma quantidade suficiente de amostras de seus componentes de tal forma que as suas características se tornem evidentes. Um *corpus* com amostras pequenas pode ocultar a presença de algum componente; por outro lado, muitas amostras podem tornar o *corpus* desnecessariamente grande.

Diversidade: Sinclair (2005, p. 15) afirma que qualquer controle sobre o tema de um *corpus* deve ser definido pelos critérios de uso externo e não interno. Namiuti-Temponi, Viana Santos e Leite afirmam: "[...] é necessário se colocar na posição de um pesquisador formador de corpora (PFC) e não apenas de um Pesquisador pragmático, aquele que se contenta apenas com seu objeto de pesquisa" (NAMIUTI-TEMPONI; VIANA SANTOS; LEITE, 2011, p. 7).

> [...] se um corpus se presta para estudos de variação ou procura representar uma língua, ele deve se preocupar com a diversidade de gêneros e tipos de textos, com a variação de dialetos e, por último, com uma diversidade de tópicos que é de fundamental importância para estudos lexicográficos, pois a frequência de muitas palavras varia de acordo com a variação de tópicos. Este último tipo de diversidade deve ser considerado para todos os tipos de estudos (ALUÍSIO; ALMEIDA, 2006, p. 159);

Tamanho: Sardinha afirma que "O corpus é uma amostra de uma população cuja dimensão não se conhece (a linguagem como um todo) " (SARDINHA, 2000; p. 342). Por esse motivo não é possível antecipar o tamanho ideal da amostra. De acordo com Sinclair (2005, p. 15), o tamanho de um *corpus* deve considerar dois fatores: o tipo de consulta que será feita pelos usuários e a metodologia dos usuários para o estudo dos dados coletados. Por esse motivo, prossegue Sinclair, não existe um tamanho máximo e, para evitar o risco de baixa representatividade, a amostra deve ser tão grande quanto possível.

2.2.3.2 Compilação e Tratamento dos Textos

Uma vez projetado o *corpus* e seus textos sido selecionados de acordo com os critérios estabelecidos na subseção anterior, é preciso armazenar e tratar os textos de forma a viabilizar

o processo de anotação e o processo de pesquisa. Aluísio e Almeida (2006) afirmam que a compilação é o armazenamento dos textos em arquivos digitais previamente determinados.

O tratamento dos textos (para a criação do *corpus*), conforme explicam as autoras, consiste basicamente em duas etapas: a) converter os documentos para o formato de texto simples; e b) limpar e formatar os textos de forma a eliminar imagens, tabelas, números de páginas, cabeçalhos etc. Esse processo pretende viabilizar o tratamento dos textos por ferramentas computacionais. Sobre isso, Namiuti-Temponi e Viana Santos afirmam:

> Os textos antigos possuem características gráficas e grafemáticas que dificultam o processamento computacional posterior à etapa de transcrição. Por essa razão, os textos precisam ser editados, mas as características do texto original devem ser preservadas, devido à sua importância para estudos linguísticos e filológicos (NAMIUTI-TEMPONI; VIANA SANTOS, 2015, p. 14).

Isso deixa claro que é preciso utilizar um procedimento de limpeza e, conforme será visto adiante, de anotação, que permita viabilizar o processamento computacional sem perder as características dos textos originais.

2.2.3.3 Anotação

Conforme Leech (2004), anotação é a adição de informação linguística interpretativa ao corpus. Namiuti-Temponi e Santos (2015) acrescentam que as anotações são interferências na edição que podem ser "[...] das mais restritas, próprias das edições paleográficas (desdobramento de abreviaturas; decisões de leitura), às mais amplas, próprias das edições modernizadas (atualização de grafia) " (NAMIUTI-TEMPONI, SANTOS, 2015, p. 15).

Faria e Galves (2016)[6], retomando o trabalho de pesquisa de Hovy e Lavid (2010), apresentam uma sugestão de roteiro para o processo de anotação de *corpus*:

> 1. **Seleção de textos representativos**. O primeiro passo para a construção de um *corpus* de treinamento envolve a seleção dos textos representativos segundo alguma hipótese ou interesse de pesquisa.
> 2. **Especificação do sistema de anotação**. Uma vez definidos os textos que constituirão o *corpus*, passa-se à definição da teoria linguística que determinará a forma de interpretação dos dados e orientará a especificação do sistema de anotação. Nesta fase, começa-se também a produção do manual de anotação, em

[6] Faria e Galves (2016) citam originalmente o trabalho apresentado por LAVID, Julia em 2013: The Impact of Corpus Annotation on Linguistic Research, apresentado no evento *36th AEDEAN Conference*. Como não se encontrou o texto citado por Faria e Galves, optou-se por citar um texto que trata do mesmo assunto, que foi publicado por HOVY, Eduard; LAVID, Julia no International Journal of Translation em 2010, e que está disponível para consulta.

que se descreve o sistema, suas motivações e assunções, e que será utilizado para treinamento de anotadores/revisores humanos.

3. **Teste do sistema de anotação**. Antes de proceder a uma utilização definitiva do sistema de anotação, é preciso determinar sua executabilidade e a clareza do manual, o que é feito anotando-se um fragmento do material selecionado no passo 1. Preferencialmente, é interessante que pelo menos duas pessoas façam essa anotação paralelamente.
4. **Avaliação do teste**. Envolve comparar as decisões dos anotadores, além de decidir pelas medidas apropriadas sobre concordância na anotação e sobre como aplicá-las. É preciso ter em mente que o que se busca são anotações confiáveis, isto é, que sejam estáveis e reproduzíveis. A estabilidade está relacionada à concordância intra-anotadores, isto é, ao quanto um mesmo anotador é consistente na anotação. Já a reprodutibilidade está relacionada à concordância entre anotadores, isto é, ao quanto os anotadores concordam na anotação dos mesmos fenômenos. Estabilidade e reprodutibilidade são fundamentais para que o treinamento de algoritmos de aprendizagem de máquina seja eficiente. É preciso estabelecer o nível (mínimo) satisfatório de concordância entre anotadores. Enquanto o teste de anotação não atingir o mínimo satisfatório, volta-se ao passo 2 para redefinir o sistema de anotação e o manual.
5. **Anotação manual de grande parte do material**. Uma vez determinado que o sistema de anotação é satisfatoriamente executável, passa-se à anotação de grande parte do material, processo que pode levar meses ou anos.
6. **Treinamento de um analisador**. Quando for acumulada uma certa quantidade de material anotado manualmente (p.e., 100 mil palavras), pode-se começar a avaliar se um analisador pode ser eficientemente treinado para que a anotação semiautomática possa começar. Para isso, este material acumulado deve ser dividido em duas partes, uma para treinamento (p.e., 90% do material) e outra para teste, de modo que o analisador seja treinado com base na porção de treinamento e testado sobre a porção "inédita" de teste. Uma vez que para a porção teste há uma anotação manual correta (em princípio) disponível, o desempenho do analisador pode ser avaliado, quanto à acurácia em relação à anotação alvo.
7. **Anotação semiautomática**. Se o desempenho do analisador se mostrar satisfatório, passa-se a utilizá-lo em novo material, em conjunto com a correção/revisão por anotadores humanos. Caso não seja satisfatório, podem ser necessários ajustes no sistema de anotação ou pode ser necessário mais material para treinamento. No primeiro caso, pode ser preciso voltar ao passo 2, a depender do quanto o sistema precisará ser alterado (em alguns casos, certas modificações podem ser aplicadas automaticamente sobre o corpus, não requerendo revisão manual). No segundo caso, volta-se ao passo 5, para produção manual de mais material anotado (FARIA; GALVES, 2016, p. 302).

Sobre o processo de anotação, Aloísio e Almeida (2006) afirmam que existem dois tipos de anotação: estrutural e linguística.

Anotação estrutural: inclui o registro de dados internos e externos dos textos. São dados externos: metadados estruturais que descrevem o texto, como cabeçalho, autor, tipologia textual etc. Já os dados internos são marcações de estrutura como título, subtítulo, capítulo, notas de rodapé etc. O objetivo dessas informações estruturais, segundo Aloísio e Almeida (2006) é facilitar a recuperação posterior do texto

Anotação linguística: Conforme Leech (2004), esse tipo de anotação visa agregar valor ao texto em aspectos semânticos, sintáticos etc.[7]

Conforme já foi anteriormente mencionado, seja qual for o tipo de anotação feito, é fundamental a utilização de uma linguagem que seja facilmente implementável em sistemas computacionais, que seja interoperável para facilitar o intercâmbio de informações e que seja extensível para permitir a adaptação às necessidades específicas dos Linguistas. Namiuti-Temponi e Costa (2014) sugerem a utilização da linguagem XML[8], pois:

> XML é uma linguagem que permite descrever qualquer tipo de dado e é um padrão aberto para interoperabilidade e intercâmbio de informações. A existência de uma ampla variedade de tecnologias para esse padrão permite a criação de recursos padronizados, favorecendo a reutilização tecnológica e facilitando a extração de dados de *corpora* (NAMIUTI-TEMPONI; COSTA, 2014, p. 83).

Os documentos XML são documentos de texto simples que utilizam marcadores para delimitar e representar os dados de forma estruturada, permitindo fazer essa representação em múltiplos níveis configuráveis de acordo com a necessidade do utilizador.

A linguagem XML vem sendo utilizada por linguistas na criação de *corpora* como o *corpus* Documentos Oitocentistas de Vitória da Conquista (DOViC) que é um *corpus* digital de documentos manuscritos do século XIX, desenvolvido no âmbito do projeto "Memória conquistense: implementação de um *corpus* digital" (NAMIUTI; SANTOS, 2013) em parceria com o projeto de pesquisa "Sintaxe diacrônica em *corpus* eletrônico: do português pré-clássico às variantes modernas" (NAMIUTI; SANTOS, 2010).

Como consequência das vantagens apresentadas pelas suas características, a linguagem XML vem sendo adotada por desenvolvedores de *softwares*, conforme analisado na seção *SOFTWARES* DE ANOTAÇÃO.

De acordo com a pesquisa realizada, a utilização de recursos digitais como ferramentas de suporte aos estudos na área da Linguística é um caminho a ser seguido para se conseguir obter mais rapidez e precisão nas pesquisas dessa área. A Linguística Computacional vem se consolidando como área de pesquisa, colocando definitivamente a área de Linguística na Era Digital.

---

[7] Será discutido em detalhes na seção *SOFTWARES* DE ANOTAÇÃO.
[8] XML – eXtensible Markup Language – Ou linguagem de marcadores extensíveis (tradução livre).

3 *SOFTWARES* DE ANOTAÇÃO

Nessa seção, é mencionada a importância dos *softwares* de anotação e apresenta alguns *softwares* de anotação disponíveis no mercado, enfatizando o eDictor que é a ferramenta alvo dessa pesquisa.

A utilização de *corpora* de dados linguísticos para estudos sobre a linguagem é crescente, especialmente nos estudos diacrônicos sobre a sua sintaxe, conforme explicam Faria e Galves (2016). Essa utilização de *corpora* torna de vital importância a realização do processo de anotação, assim definido por Faria e Galves: "[...] o processo de anotação de um *corpus*, do ponto de vista de processamento de linguagem natural (PLN), consiste na transformação do texto puro em texto marcado (e, de certo modo, interpretado) " (FARIA; GALVES, 2016, p. 302). Leech (2004), assim define anotação de *corpus*: "*Corpus annotation is the practice of adding interpretative linguistic information to a corpus. For example, one common type of annotation is the addition of tags, or labels, indicating the word class to which words in a text belong*"[9] (LEECH, 2004, p. 25).

Conforme já foi mencionado na Introdução deste trabalho, Aluísio e Almeida (2006) comentaram sobre o processo de anotação linguística:

> A anotação linguística pode ser em qualquer nível que se queira, isto é, nos níveis morfossintático, sintático, semântico, discursivo, etc., sendo inserida de três formas: manualmente (por linguistas), automaticamente (por ferramentas de Processamento de Língua Natural – PLN) ou semiautomaticamente (correção manual da saída de outras ferramentas). Essa última é comprovadamente mais eficiente, pois revisar é mais rápido e gera dados mais corretos do que anotar pela primeira vez (ALUÍSIO; ALMEIDA, 2006, p. 161).

As anotações linguísticas feitas manualmente têm o inconveniente de serem demasiado trabalhosas, repetitivas e lentas. Não raro, exige do linguista o domínio de alguma linguagem computacional. Já as anotações feitas de forma automática ou semiautomática exigem a utilização de *softwares* de anotação. Esses *softwares* devem permitir que os linguistas façam correções controladas das anotações linguísticas automáticas concentrando-se na sua atividade fim.

Também a tarefa manual das edições filológicas pode exigir do pesquisador o domínio da linguagem utilizada na marcação da edição (XML, por exemplo), o que pode trazer

[9] Anotação de *corpus* é a prática de adicionar informação linguística interpretativa a um *corpus*. Por exemplo, um tipo comum de anotação é a adição de etiquetas indicando as classes gramaticais a que as palavras de um texto pertencem. (Livre tradução do autor).

dificuldades como: curva de aprendizagem da linguagem, o que pode atrasar o processo por exigir dos linguistas um conhecimento não usual nas suas áreas de conhecimento e de atuação; marcações malformadas ou erros de marcação, o que pode inutilizar parte de um *corpus* e, a depender do tamanho desse, dificultar a localização e correção.

Para conhecer o estado da arte nessa área, foram pesquisados *softwares* especialistas em anotação de corpus e as suas características são comentadas adiante.

Para comparar as características de cada *software* foram analisados os seguintes aspectos:

Desenvolvedor: nome e nacionalidade do(s) desenvolvedor(es).

Ano de desenvolvimento.

Licença de uso: de que forma é disponibilizado para os pesquisadores, taxas cobradas (se for o caso) etc.

Local onde fazer o *download*.

Facilidade de uso da interface: para apontar se a interface de operação é intuitiva, se pode ser configurada para exibição em mais de uma língua, especialmente a Língua Portuguesa.

Disponibilidade de documentação: para verificar se existe documentação, se é incorporada ao *software* ou se está disponível em outro local e se engloba os comandos de operação.

Local de execução: para indicar se o *software* é executado no *desktop* (no próprio computador), se é baseada na WEB através de navegador ou baseada na WEB e iniciada por Java Web Start ou similar.

Plataforma(s) de uso: disponibilidade para qual plataforma de Sistema Operacional (Windows, Linux, MAC OS etc.).

Estágio do desenvolvimento: se está disponível em versão estável, se está em desenvolvimento e número da versão disponível.

Software adicional: que *software* necessita para funcionar.

Armazenamento dos metadados: de que forma os dados são armazenados.

Compatibilidade com XML: se permite ou não tratar *corpus* e fazer as anotações em formato XML.

Anotações em múltiplas camadas: se permite fazer anotações em diversas camadas de forma consistente.

Edições filológicas: se permite fazer edições filológicas de forma nativa e se pode estender essa capacidade.

Acoplamento de outras ferramentas: se permite acoplar outras ferramentas como *plugins* e quais existem disponíveis.

Garantia de suporte técnico: se tem suporte técnico oferecido de alguma forma.

Os seguintes *softwares* foram analisados: ATOMIC, SACODEYL, MULTEXT, CLARK e eDICTOR. A escolha desses *softwares* foi feita a partir de indicações do site corpus-tools.org e corpus-analysis.com. Após análise da lista de *softwares* indicados, foram selecionados aqueles que apresentaram recursos de tratamento multiníveis.

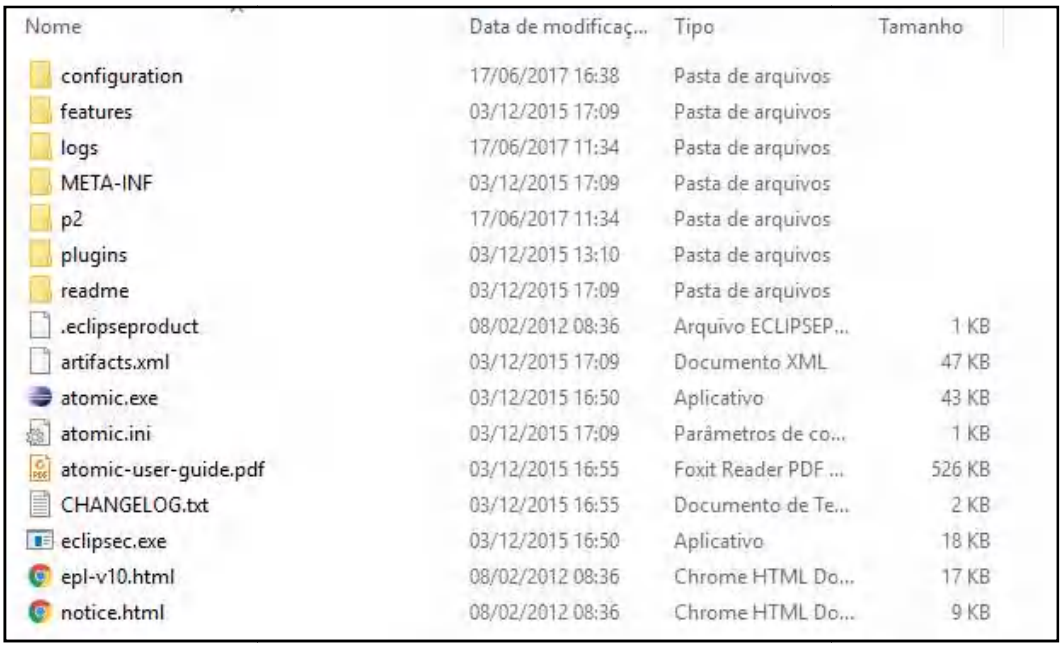
3.1 ATOMIC

Segundo Stephan et al (2014), o *software* ATOMIC é uma aplicação projetada para fazer anotações multiníveis em *corpus*. Foi desenvolvido em Java, utilizando Eclipse Rich Client.

3.1.1 Características

- Desenvolvedores: Stephan Druskat, Lennart Bierkandt, Volker Gast, Christoph Rzymski e Florian Zipser nas Universidades de Jena (Friedrich Schiller University Jena - FSU) e de Zurique – Alemanha.
- Ano de desenvolvimento: 2014.
- Licença de uso: Apache 2.0 e disponível sob licença Creative Commons, atribuição não comercial. Isso significa que pode ser livremente utilizado, desde que seja para fins não comerciais.
- Local onde fazer o download: http://linktype.iaa.uni-jena.de/atomic/#download. É fornecido em formato ZIP e sua instalação requer apenas a descompactação do arquivo fornecido. Após a descompactação, é gerada a estrutura de pastas mostrada na Figura 1.

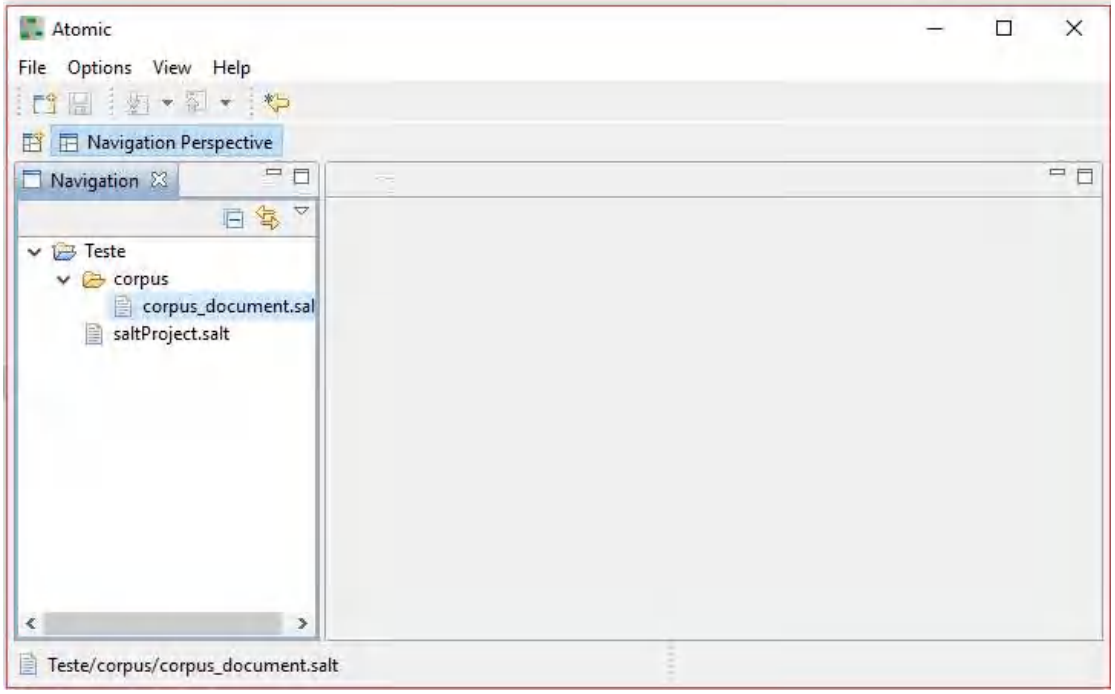
Figura 1 - Estrutura de Pastas do ATOMIC



Fonte: print screen no Sistema Operacional Windows 10.

- Facilidade de uso da interface: Disponível apenas em Inglês, a interface de uso é simples, porém intuitiva, conforme é ilustrado na Figura 2.

Figura 2 - Interface do Atomic



Fonte: *print screen* da aplicação no Sistema Operacional Windows 10

- Disponibilidade de documentação: o Atomic dispõe de ajuda incorporada ao programa (help) e inclui todos os seus comandos. Além disso, existe um manual disponível no site do programa e um manual em formato PDF juntamente com o *download (atomic-user-guide.pdf)*.

- Local de execução: trata-se de uma aplicação *desktop*, executando localmente no computador onde está instalado. Necessita de acesso à Internet apenas para incluir *plug-ins*.
- Plataforma(s) de uso: no momento, está disponível para as plataformas Windows, Linux e Mac OS X, todas em versões para 32 e para 64 bits.
- Estágio do desenvolvimento: versão mais recente é a Atomic 0.2.1 de dezembro de 2015.
- *Software* adicional: Para execução, necessita apenas do Java Runtime Environment 1.6 ou mais nova.
- Armazenamento dos metadados: de acordo com Stephan et al (2014), os metadados são baseados na biblioteca (Application Programming Interface[10] – API) Salt, disponível em http://corpus-tools.org/salt.
- Compatibilidade com XML: permite importar dados em formato XML, entretanto isso requer a instalação do plugin Eclipse XMLEspresso, o que pode ser feito diretamente do Atomic.
- Anotações em múltiplas camadas: permite fazer anotações em múltiplas camadas, utilizando uma linguagem própria.
- Edições filológicas: manualmente.
- Acoplamento de outras ferramentas: permite instalar outras ferramentas (plugins) diretamente do Market Place do Eclipse ou desenvolvidas por terceiros.
- Garantia de suporte técnico: oferece suporte técnico por e-mail. Também dispõe de uma lista de e-mail para fornecer informações de atualização.

3.2 SACODEYL Annotator

O *System Aided Compilation and Open Distribution of European Youth Language* (SACODEYL) é um resultado do Projeto Socrates-Minerva da União Europeia. De acordo com Pérez-Paredes (2008) o objetivo original do projeto é:

> SACODEYL presents an innovative ICT-based solution for the compilation and pedagogical, language learning-oriented exploitation of linguistic teen talk oral

[10] Application Programming Interface (API) é uma espécie de biblioteca com padrões de programação para uso em softwares.

> corpora. SACODEYL is an integration initiative that takes into account areas as:- multilingual corpus compilation and innovation support,- pedagogical adaptation of existing ICT tools for the integration of corpora and corpus-based methods of Data Driven Learning (DDL) into mainstream language teaching, and- know-how building which will contribute to the future drafting of specifications that can take both DDL and the use of oral corpora in language education a step forward.[11]

Apesar desse objetivo original, é possível criar outros *corpora*, estender marcadores e utilizar para outros tipos de pesquisa.

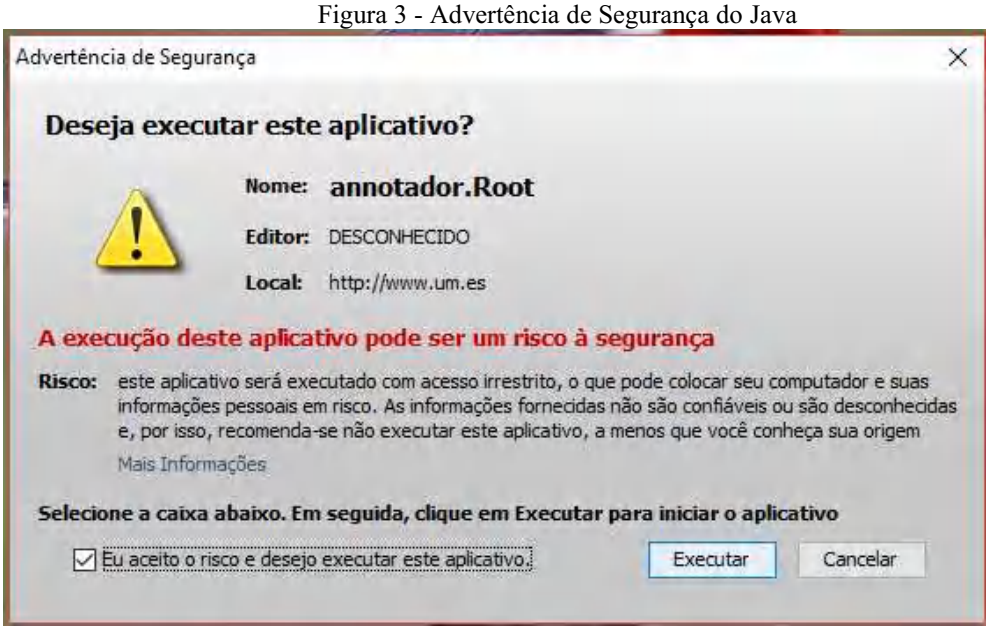
3.2.1 Características

- Desenvolvedor: Projeto Coordenado por PÉREZ-PAREDES, Pascual
- Ano de desenvolvimento: 2008.
- Licença de uso: Distribuído sob licença Creative Commons CC BY-NC-ND 3.0. Pode ser utilizado, pode ser modificado, pode ser utilizado para fins de pesquisa, mas não pode ser redistribuído se for modificado.
- Local onde fazer o *download* do programa: http://www.um.es/sacodeyl/en/pages/software.htm#annotator. O *download* inclui apenas o arquivo annotator,jnlp que é um script para a execução do programa no servidor remoto12. Como a página do provedor do serviço não apresenta segurança adequada, ao executar o *script*, é gerada uma advertência conforme mostra a Figura 3. Também é necessário adicionar uma exceção de segurança na máquina virtual do Java.

[11] O SACODEYL apresenta uma solução inovadora baseada em Tecnologias da Informação e da Comunicação (TICs) para a compilação e a exploração pedagógica, orientada para a aprendizagem de linguagem adolescente, usando *corpora* orais linguísticos. SACODEYL é uma iniciativa de integração que leva em consideração áreas como:
- compilação de *corpus* multilíngue e apoio à inovação,
- adaptação pedagógica de ferramentas TIC existentes para a integração de corpora e métodos de aprendizagem direcionada a dados (Data Driven Learning - DDL) usando corpus para o ensino convencional de línguas; e
- construção de *know-how* que contribuirá para a futura criação de especificações que podem levar a DDL e o uso de *corpora* orais na educação linguística a um passo à frente. (Livre tradução do autor).

[12] Servidor remoto: equipamento que fica fora da rede local. No caso, fica na Universidade onde está o projeto.

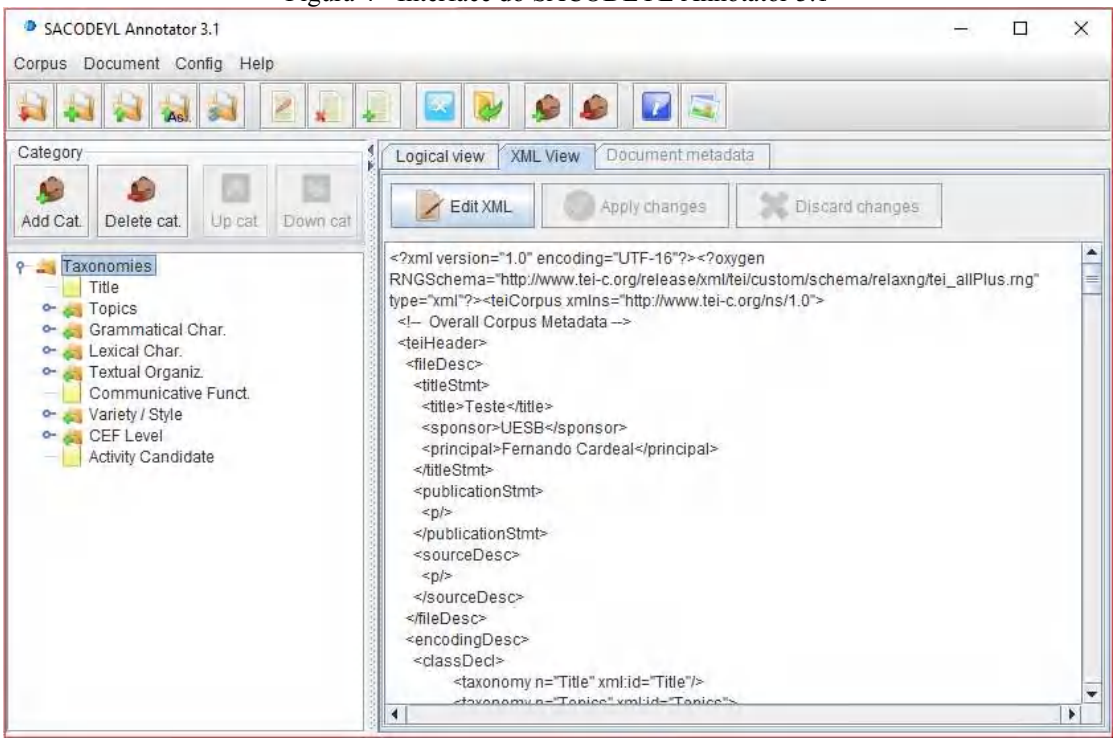
Figura 3 - Advertência de Segurança do Java



Fonte: *print screen* de mensagem gerada pelo Java no Sistema Operacional Windows 10.

- Facilidade de uso da interface: a interface de uso, mostrada na Figura 4, é simples e pode ser configurada em Inglês, Italiano, Espanhol, Alemão, Romeno e Lituano.

Figura 4 - Interface do SACODEYL Annotator 3.1

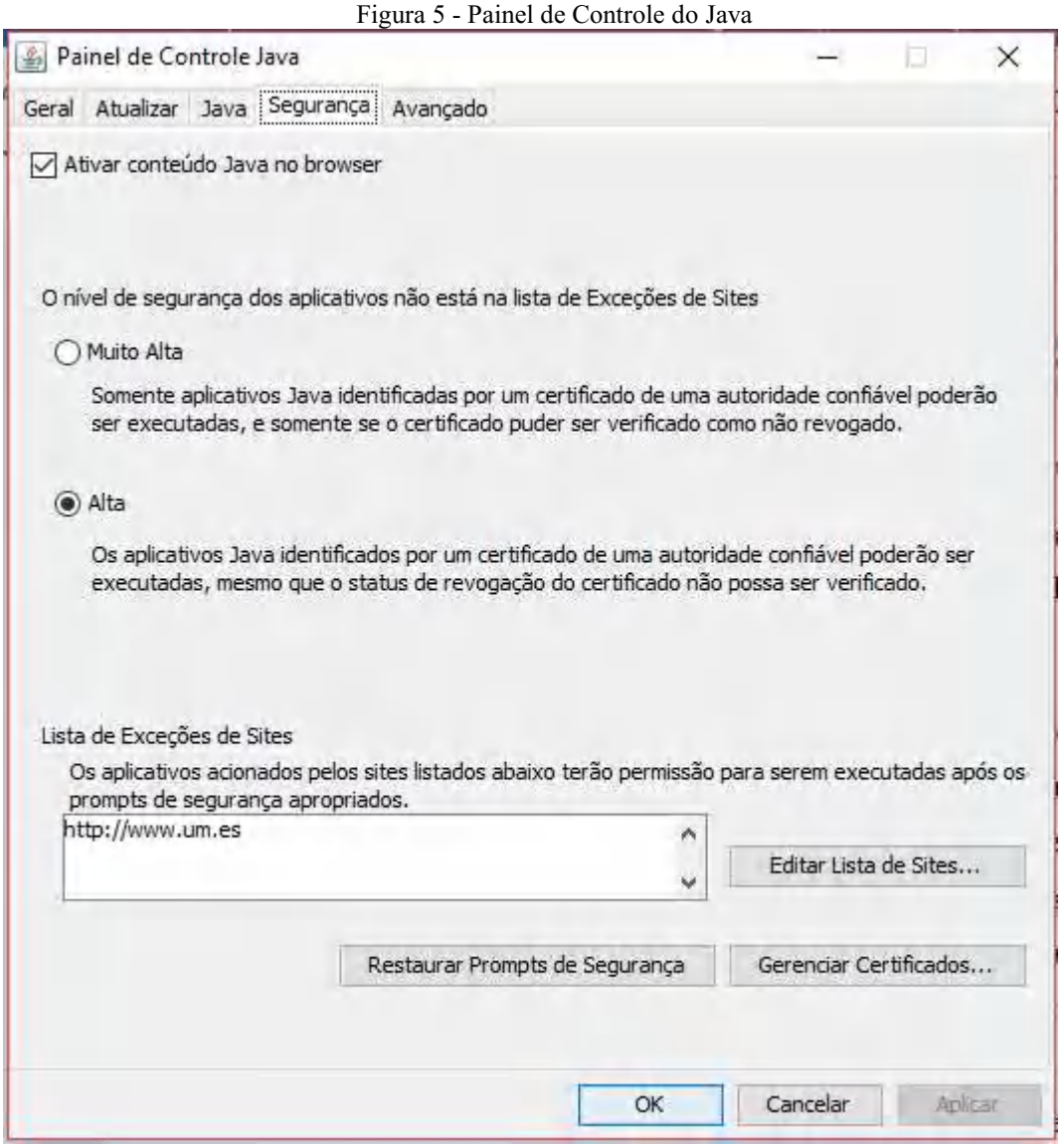


Fonte: print screen do aplicativo no Sistema Operacional Windows 10.

A interface apresentou eventualmente problemas para encerramento da aplicação. A figura 4 mostra um exemplo de *corpus* configurado para uso independente dos *corpora* oferecidos online.

- Disponibilidade de documentação: existe *help* incorporado ao programa e também são oferecidos vídeos de treinamento na página do desenvolvedor, além de roteiro para instalação e *corpora* de exemplo.
- Local de execução: o SACODEYL é baseado na WEB. Isso significa que é necessário ter acesso à Internet para utilizar os seus recursos. Percebeu-se que é necessário adicionar uma lista de exceção de segurança, fornecendo permissão à página http://www.um.es que, como não é uma página segura, pode trazer riscos ao computador. A permissão deve ser feita através do painel de controle do Java conforme ilustra a Figura 5.
-

Figura 5 - Painel de Controle do Java



Fonte: print screen da tela de configuração do Java no Sistema Operacional Windows 10.

- Plataforma(s) de uso: o SACODEYL está disponível para as plataformas Windows, Linux e Mac OS X.
- Estágio do desenvolvimento: Está disponível, no site do desenvolvedor, a versão 3.1 que foi utilizada para as comparações.
- *Software* adicional: para o funcionamento, necessita da máquina virtual JRE 1.6 ou posterior.
- Armazenamento dos metadados: os dados são armazenados em formato XML. Os metadados de configuração do *corpus* são armazenados em formato de texto simples (txt).
- Compatibilidade com XML: Permite utilizar *corpus* em formato XML e pode estender os marcadores já existentes no *software*.
- Anotações em múltiplas camadas: se permite fazer anotações em diversas camadas de forma consistente.
- Edições filológicas: permite fazer anotações morfossintáticas de forma manual, exigindo conhecimento de XML.
- Acoplamento de outras ferramentas: Não permite acoplar outras ferramentas, entretanto, são fornecidas duas ferramentas auxiliares: SACODEYL Transcriptor que é utilizada para transcrever dados orais ou vídeos para o formato do SACODEYL Annotator; e SACODEYL Search para pesquisar os *corpora* do SACODEYL.
- Garantia de suporte técnico: o desenvolvedor afirma que os códigos fontes serão disponibilizados sob licença GNU. Na guia de ajuda do SACODEYL Annotator é mostrado o e-mail do desenvolvedor, mas sem informações sobre suporte técnico ou garantia de continuidade do projeto.
-

3.3 MULTEXT

De acordo com Paixão de Sousa, Kepler, Faria (2009), o Multext é:

> [...] uma série de projetos cujas metas são o desenvolvimento de padrões e especificações para a codificação e processamento de *corpora* linguísticos, e o desenvolvimento de ferramentas, *corpora* e recursos linguísticos que encarnem estes padrões. O Multext tem desenvolvido ferramentas, *corpora* e recursos linguísticos para uma grande variedade de línguas, incluindo Bambara, Búlgaro, Catalão, Tcheco, Holandês, Inglês, Estoniano, Francês, Alemão, Húngaro, Italiano, Quicongo, Occitano, Romeno, Esloveno, Espanhol, Sueco e Suaíli. Todos os

> resultados do Multext são liberados e disponíveis publicamente para propósitos não-comerciais e não-militares (PAIXÃO DE SOUZA; KEPLER; FARIA, 2009, p. 5).

Não foi possível fazer o *download* da ferramenta para testes, pois o *link* oferecido na página do projeto não funcionou. Foram feitas tentativas através do site da Universidade Aix-en-Provence e do Laboratoire Parole et Langage, todas sem sucesso.

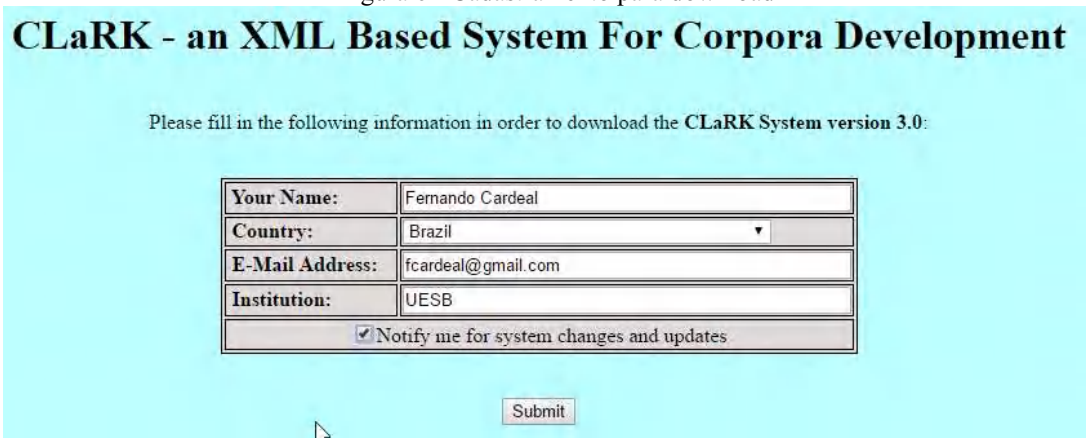
Conforme os autores citados anteriormente, "este editor parece mediar o contato do usuário com a linguagem XML, gerando a estrutura subjacente a partir da formatação do texto através da interface do editor" (PAIXÃO DE SOUZA; KEPLER; FARIA, 2009, p. 5).

3.4 CLaRK

O *Computational Linguistics and Represented Knowledge* (CLaRK) é um sistema desenvolvido em Java no Tübingen-Sofia International Graduate Programme com o objetivo de minimizar o trabalho humano durante o processo de criação de *corpora* conforme explica Simov (2001, p. 553).

Para fazer o *download* do programa, é solicitado um cadastramento do pesquisador, conforme é ilustrado na Figura 6.

Figura 6 - Cadastramento para download



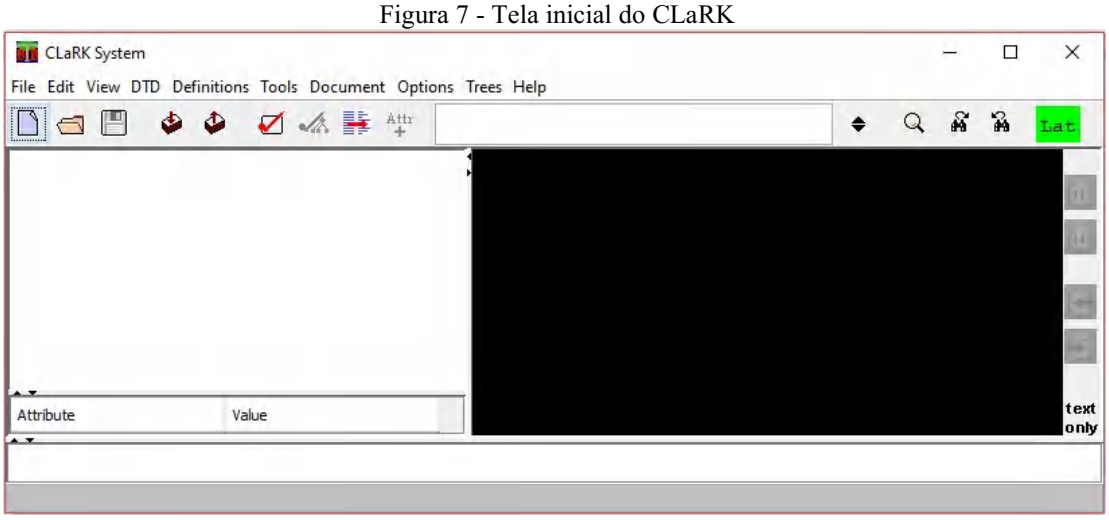
Fonte: Clark – Página de *download*[13]

[13] http://www.bultreebank.org/clark/download_form.html

3.4.1 Características

- Desenvolvedores: Kiril Simov, Zdravko Peev, Milen Kouylekov, Alexander Simov, Marin Dimitrov, Atanas Kiryakov. The CLaRK Programme - Linguistic Modelling Laboratory - CLPPI, Bulgarian Academy of Sciences.
- Ano de desenvolvimento: 2000/2001.
- Licença de uso: copyright pelos autores. Pode ser utilizado livremente, mas os autores pedem uma pequena taxa (voluntária) para cobrir custos.
- Local onde fazer o download: http://www.bultreebank.org/clark/download_form.html. É fornecido em formato ZIP e sua instalação requer apenas a descompactação do arquivo fornecido.
- Facilidade de uso da interface: Disponível apenas em Inglês, a interface de uso é aparentemente simples, mas requer uma série de configurações para ser utilizada adequadamente. A tela inicial é ilustrada na Figura 7.
-

Figura 7 - Tela inicial do CLaRK



Fonte: Print Screen do aplicativo no Sistema Operacional Windows 10.

- Disponibilidade de documentação: o CLaRK dispõe de ajuda incorporada ao programa (help) em formato de referência. Além disso, existe um manual disponível no site do programa. Também dispõe de demonstrações (demo) no site *download*.
- Local de execução: trata-se de uma aplicação *desktop*, executando localmente no computador onde está instalado.

- Plataforma(s) de uso: no momento, está disponível para as plataformas Windows, Linux e Mac OS X.
- Estágio do desenvolvimento: versão estável mais recente é a CLaRK 3.0 de 25/11/2004.
- *Software* adicional: Para execução, necessita apenas do Java Runtime Environment 1.5 ou mais nova.
- Armazenamento dos metadados: de acordo com Simov et al (2001), todo mecanismo do CLaRK é baseado em XML.
- Compatibilidade com XML: permite o uso do XML diretamente.
- Anotações em múltiplas camadas: permite fazer anotações em múltiplas camadas, utilizando XML, exigindo esse conhecimento do linguista.
- Edições filológicas: manualmente.
- Acoplamento de outras ferramentas: não.
- Garantia de suporte técnico: oferece suporte técnico por e-mail. Também dispõe de uma lista de e-mail para fornecer informações de atualização.

3.5 eDICTOR

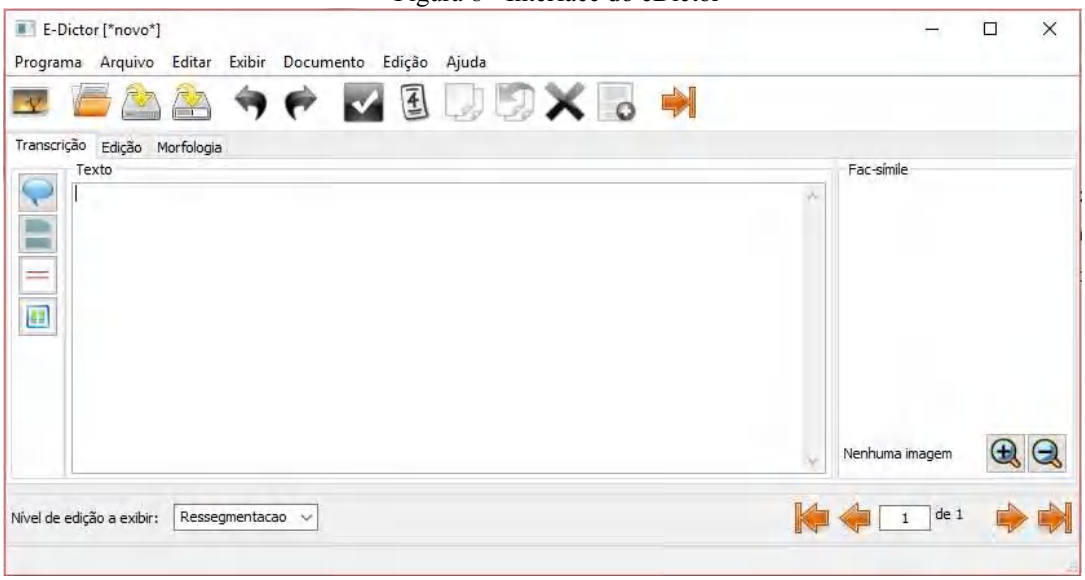
O eDictor (anteriormente E-Dictor) é uma ferramenta concebida para auxiliar a edição eletrônica em XML de textos antigos para fins de análise linguística automática, de acordo com Paixão de Sousa, Kepler e Faria (2009, p. 1).

3.5.1 Características

- Desenvolvedores: Pablo Faria, Fábio Kepler, Maria Clara Paixão de Sousa.
- Ano de desenvolvimento: Foi desenvolvida uma versão preliminar em 2007, depois foi modernizado em 2010.
- Licença de uso: MIT. Isso significa que pode ser livremente utilizado, desde que seja para fins não comerciais.
- Local onde fazer o download: https://edictor.net/download. É fornecido um programa instalador: edictor-1.0beta_b010-win32.exe. A instalação é intuitiva.

- Facilidade de uso da interface: Disponível em Inglês e em Português do Brasil, a interface de uso é simples, porém intuitiva, conforme é ilustrado na Figura 8. Requer algumas configurações iniciais que são esclarecidas no manual do programa.

Figura 8 - Interface do eDictor



Fonte: Print screen do programa no Sistema Operacional Windows 10

- Disponibilidade de documentação: o eDictor dispõe de ajuda incorporada ao programa (help) e inclui todos os seus comandos. Além disso, existe um manual disponível no site do programa e um manual em formato PDF.
- Local de execução: trata-se de uma aplicação *desktop*, executando localmente no computador onde está instalado.
- Plataforma(s) de uso: no momento, está disponível para as plataformas Windows e Linux. Está sendo considerado o desenvolvimento de versão para Mac OS X.
- Estágio do desenvolvimento: versão mais recente é a eDictor Versão 1.0 build 010 (fev. 2010).
- *Software* adicional: Não requer *software* adicional para a execução.
- Armazenamento dos metadados: armazena os dados em XML e permite exportar para HTML e outros formatos.
- Compatibilidade com XML: utiliza XML de forma transparente para o usuário.

- Anotações em múltiplas camadas: O eDictor permite operar com vários níveis de edição: Junção, Segmentação, Grafia, Modernização, Expansão, Correção, Pontuação (e é possível criar novos níveis, de acordo com a necessidade do pesquisador).
- Edições filológicas: Sim, por se tratar de um editor especializado, desenvolvido para atender a essa necessidade.
- Acoplamento de outras ferramentas: já possui outras ferramentas instaladas como um etiquetador.
- Garantia de suporte técnico: não.

3.6 RESUMO COMPARATIVO DAS CARACTERÍSTICAS DOS *SOFTWARES*

O Quadro 1 mostra um resumo de todas as características levantadas, facilitando a comparação entre os *softwares* pesquisados. Esse tipo de estudo tem grande importância. pois complementa o levantamento de requisitos funcionais para um novo *software*, enfatizando funcionalidades e limitações dos programas já existentes no mercado

Quadro 1 – Resumo dos Softwares Anotadores

CARACTERÍSTICA	ATOMIC	SACODEYL Annotator	ClarK	eDictor
Desenvolvedor	Stephan Druskat e outros – ALEMANHA	Pascual Péres Paredes (coord) – ESPANHA. Projeto em conjunto cm várias Universidades europeias	Kiril Simov e outros - Bulgarian Academy of Sciences – BULGÁRIA	Pablo Faria, Fábio Kepler, Maria Clara Paixão de Sousa - BRASIL
Ano do desenvolvimento	2014	2008	2000	Preliminar em 2007; modernizada em 2010
Licença de uso	Creative Commons	Creative Commons	Copyright	Licença MIT
Download.	http://linktype.iaa.uni-jena.de/atomic/#download	http://www.um.es/sacodeyl/en/pages/software.htm#annotator	http://www.bultreebank.org/clark/download_form.html	https://edictor.net/download/
Interface / Facilidade de uso	Inglês. Simples e intuitiva	Espanhol, Italiano, Inglês, Romeno e Lituano. Interface complexa tanto na configuração quanto na utilização.	Inglês – requer configurações complexas	Português brasileiro e Inglês. Fácil de usar. Requer configurações das marcações.
Documentação:	Help e manual	Help, manual e vídeos	Help, manual e demonstrações	Help e manual
Local de execução	Desktop	WEB	Desktop	Desktop
Plataforma	Windows, Linux, MAC	Windows, Linux, MAC	Windows, Linux	Windows, Linux
Estágio do desenvolvimento / versão	Versão não estável 0.2.1 (Dez. 2015)	Versão estável 3.1 (data não identificada)	Versão estável 3.0 (25/11/2004)	Versão 1.0 build 014 (fev. 2014)
Software adicional:	JRE 1.6 ou posterior	JRE 1.6 ou posterior	JRE 1.5 ou posterior	Nenhum
Armazenamento dos metadados:	Formato da API Salt	Texto simples XML	Texto simples XML	Texto simples XML
Compatibilidade com XML:	Não nativa. Necessita de plugin para importar	Sim, mas exige conhecimento do usuário	Sim, nativo	Sim, nativo de forma transparente para o usuário
Anotações em múltiplas camadas:	Sim, utilizando linguagem própria, obrigando o linguista a aprender uma nova linguagem	Não	Sim, mas exige conhecimento de XML, permitindo que o texto fique com o XML corrompido	Sim diretamente no texto, sem necessidade de conhecer XML. Ideal
Edições filológicas:	Manualmente, usando linguagem própria	Sim em XML manualmente, exigindo conhecimento	Sim em XML manualmente, exigindo conhecimento	SIM, diretamente, dispensando conhecimento

CARACTERÍSTICA	ATOMIC	SACODEYL Annotator	ClarK	eDictor
		complementar de XML e permitindo a geração de XML malformado	complementar de XML e permitindo a geração de XML malformado	adicionais.
Acoplamento de outras ferramentas:	SIM	SIM	NÃO	SIM. Já possui ferramentas acopladas
Suporte técnico:	Sim, por e-mail	Não	Sim, por e-mail	Não

Fonte: Elaborado pelo autor

3.7 CONSIDERAÇÕES SOBRE OS *SOFTWARES* PESQUISADOS

Após instalar, testar os *softwares* pesquisados e comparar as suas características, é possível concluir que a maior dificuldade na sua utilização é a necessidade de domínio de alguma linguagem computacional, seja ela uma linguagem de uso comum como a XML ou uma linguagem desenvolvida especificamente para o *software*, como faz o *software* Atomic. Dessa forma, torna-se necessário acrescentar esses conhecimentos à formação dos linguistas, o que gera uma curva de aprendizagem e aumenta o risco de causar danos aos *corpora*, tornando-os inacessíveis total ou parcialmente.

Por outro lado, dentre os softwares pesquisados. o eDictor é o que possui mais características adequadas ao uso por linguistas para anotações filológicas. É possível destacar que o seu uso independe do conhecimento da linguagem XML, pois, apesar de o software utilizar esse formato, o seu uso é transparente, dispensando essa aprendizagem e evitando a criação de documentos malformados dentro dos *corpora*.

Uma característica importante do eDictor é a possibilidade de visualizar o texto original e o texto anotado (editado). Essa característica também foi notada no SACODEYL Annotator, embora esse apenas permita esconder ou exibir as etiquetas de marcadores XML.

De acordo com Paixão de Sousa, Kepler e Faria, o eDictor apresenta as seguintes possibilidades de uso:

> Flexibilidade dos formatos gerados, permitindo tanto a leitura humana como a leitura automática; Garantia da qualidade filológica da edição por se tratar de um editor especializado; Software livre: o que dá a possibilidade de trabalho no código-fonte (alterar o programa original para atender a necessidades específicas); Previsão de continuidade do programa; Transferibilidade garantida. Ferramenta completa: O resultado combina correção do reconhecimento e edição de variação de grafia (PAIXÃO DE SOUSA; KEPLER; FARIA, 2009, p. 22).

De acordo com Paixão de Sousa (2011, p. 21) o eDictor tem uma limitação importante: não é "treinável", o que tem duas implicações:

- Os resultados das anotações não são transferidos para o restante do acervo;
- Os resultados das anotações não são transferidos para outros projetos e acervos.

Conforme já foi mencionado em outra seção, isso pode implicar em retrabalho, aumentando a possibilidade de erros de anotação.

Nenhum dos *softwares* pesquisados tem a capacidade de treinamento mencionada por Paixão de Sousa (2011). Para atender a essa necessidade, é preciso incluir algoritmos de

“aprendizado de máquina”, uma das técnicas apresentadas no Apêndice A - INTELIGÊNCIA ARTIFICIAL.

4 MODELAGEM DO *SOFTWARE* eASSIGNER

Esta seção descreve o projeto do *software* eAssigner, iniciando com a sua arquitetura, sua modelagem e finalizando com seu protótipo.

O eAssigner é um *software* que utiliza os algoritmos de Aprendizado de Máquina para analisar as anotações XML feitas em *Corpora* digitais para aprender e replicar essas anotações no mesmo *corpus* ou propagar em outros *corpora*. A principal vantagem é superar a limitação apontada no eDictor.

O eAssigner está sendo desenvolvido para trabalhar em conjunto com o eDictor, compondo um conjunto de ferramentas de auxílio aos pesquisadores, reduzindo o esforço humano e aumentando a eficiência e padronização das edições filológicas, contribuindo, assim, para atingir melhores resultados na análise de *Corpora*. Isso não impede o *software* de ser utilizado de forma autônoma.

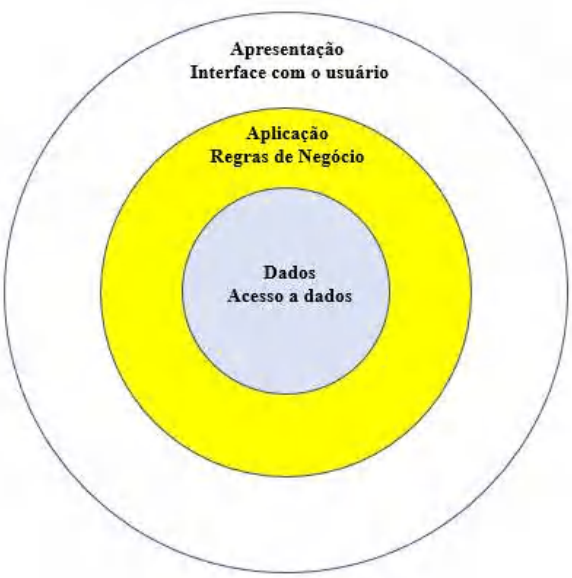
4.1 ARQUITETURA DO eASSIGNER

Silveira *et al* (2012) explicam que uma das importantes decisões em arquitetura de *software* é a sua divisão em camadas (*layers*) de forma a reduzir seu acoplamento e facilitar futuras mudanças, sejam elas evolutivas ou corretivas.

O eAssigner foi dividido em três camadas conforme ilustrado na Figura 9. A camada APRESENTAÇÃO é a interface de comunicação com o usuário; a camada APLICAÇÃO é a responsável pelas ações, decisões, resolução de conflitos, ou seja, as regras de negócio; a camada Dados é a responsável pela leitura e armazenamento de dados no computador.

Para a representação das camadas foi utilizado o formato de Arquitetura Cebola (*Onion Architecture*).

Figura 9- Camadas



Fonte: Elaborado pelo autor

A separação da camada de dados tem pelo menos duas grandes vantagens:

- Torna as regras de negócio independentes da forma como os dados estiverem armazenados. Nessa implementação inicial, os dados estão armazenados em formato XML, que é o mesmo formato utilizado pelo eDictor.
- Como a integração do software eAssigner com o *software* eDictor é feita utilizando arquivos, futuras alterações do eDictor que modifiquem a sua forma de armazenamento de dados terão impacto apenas na camada de dados, mantendo a essência do funcionamento do eAssigner.

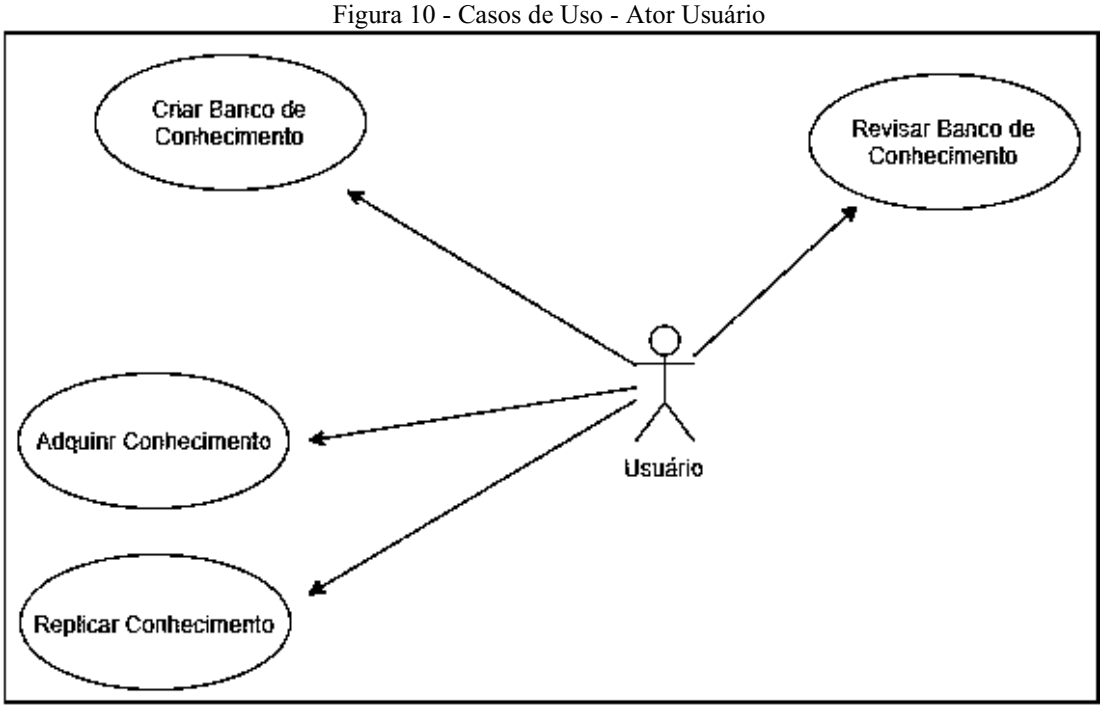
4.2 ANÁLISE E MODELAGEM DO eASSIGNER

Os seguintes requisitos funcionais foram identificados durante a análise do eAssigner:

1. Permitir ao usuário a criação de um Banco de Conhecimento para registrar o aprendizado das anotações em *corpora*;
2. Permitir ao usuário a revisão do Banco de Conhecimento para corrigir aprendizados incorretos e eliminar ambiguidades;
3. Permitir ao usuário treinar o *software* para a aquisição de conhecimento;
4. Permitir ao usuário a replicação automatizada do conhecimento.

A partir desse levantamento inicial, foi construído o Diagrama de Casos de Uso ilustrado na Figura 10.

Figura 10 - Casos de Uso - Ator Usuário



Fonte: Elaborado pelo Autor

Além dos requisitos funcionais relacionados, também foram identificados os seguintes requisitos não funcionais – que são requisitos internos ao *software*:

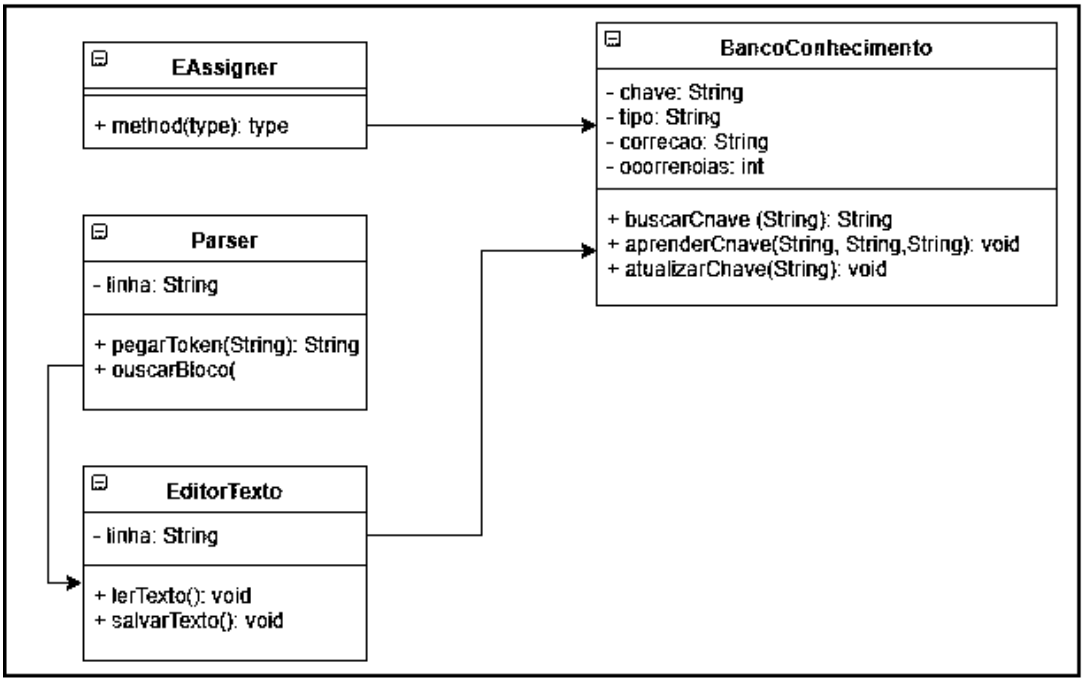
1. Reconhecer e tratar arquivos em formato XML;
2. O *software* deve ser compatível com as edições do eDictor.

O eAssigner é estruturado em classes que executam métodos para atender aos requisitos do *software*. Essas classes estão representadas em um Diagrama de Classes simplificado, ilustrado na Figura 11.

A sua estrutura essencial é composta das seguintes classes:

- EAssigner: corresponde à camada de apresentação. Gerencia as telas do sistema e faz a interface com as regras de negócio.
- BaseConhecimento: é onde o eAssigner registrará as anotações que aprender. Inicialmente é um arquivo vazio e, à medida que o *software* aprende, registrará informações como: o padrão a reconhecer; a operação a executar; frequência de ocorrência (para utilização de métodos estatísticos de aprendizado e para gerar relatórios posteriormente);
- EditorTexto: carrega e salva arquivos;
    - Módulo de Leitura: permite reconhecer um *corpus*, abrir e ler o seu conteúdo, passando os dados para o módulo *Parser*;
- *Parser*: separa em blocos as tags XML dos dados lidos para permitir a análise de padrões. Cada *tag* identificada é chamada de *token*;
    - Módulo Analisador de Padrões (embutido no Parser e implementado através da biblioteca StAX): recebe os *tokens* e verifica similaridade na Base de Conhecimento.

Figura 11 - Diagrama de Classes (Simplificado)



Fonte: Elaborado pelo Autor

Vale ressaltar que esse é um Diagrama Simplificado, incluindo apenas os elementos principais.

4.3 PRINCIPAIS ALGORITMOS DO eASSIGNER

Esta subseção apresenta os principais algoritmos do *software* eAssigner. Estão desenvolvidos em forma de pseudocódigo, utilizando a técnica do Português Estruturado, permitindo melhor compreensão. Também foram acrescentados comentários que são precedidos pelo indicador dupla barra, exemplo: // isto é um comentário.

4.3.1 Algoritmo de Atualização do Banco de Conhecimentos

O algoritmo de atualização do banco de conhecimentos ilustrado pela Figura 12 apresenta os procedimentos para o treinamento do eAssigner. Pode ser executado quantas vezes forem necessárias com novos arquivos de treinamento.

A busca de chave no Banco de Conhecimentos utiliza a técnica de Busca em Largura, explicada no Apêndice A – Inteligência Artificial. A busca em largura foi escolhida porque as suas características atendem adequadamente ao tipo de estrutura do banco de conhecimento.

Figura 12 – Algoritmo de Treinamento

```
// Atualização do Banco de Conhecimentos

Conectar Banco de Conhecimentos
Abrir Arquivo de Treinamento <Treinar>

Enquanto (Houver dados em <Treinar>) faça
    Obter Anotação ("junção", "ressegmentação")

    Buscar chave no Banco de Conhecimentos (<o>)    // localiza a palavra original
    Se não achar   então
        Adicionar Anotação ao Banco de Conhecimentos // salva a chave, o tipo e a anotação
    senão   // achou a palavra no Banco, mas precisa conferir a anotação
        Se Anotação idêntica  então //  encontrou a palavra e não houve conflito
            Somar 1 à ocorrência no banco de Conhecimentos
        senão  // achou a mesma chave, mas a anotação é diferente: possível conflito
            Mensagem com opção: Substituir (S), Acrescentar (A), Ignorar (I)
            Avaliar resposta:
                Caso 'S':  Substituir anotação no Banco de Conhecimentos
                Caso 'A':  Adicionar nova anotação com a mesma palavra chave
                Caso 'I':  Ignorar e abandonar a ocorrência
            fim-da-avaliação
        fim-se // que testa anotação idêntica
    fim-se  // fim do tratamento dessa palavra chave

   obter novo registro em <Treinar>
fim-enquanto  // quando acabarem os dados, segue para a próxima linha

Salvar e fechar Banco de Conhecimentos
Fechar arquivo <Treinar>

Mensagem "fim de treinamento"
retornar
```

Fonte: Elaborado pelo autor

Ao processar um *corpus* de treinamento, é possível se deparar com a seguinte situação de conflito: a mesma palavra ter anotações diferentes. Nesse caso, o eAssigner oferece ao linguista três opções: substituir a anotação, adicionar a nova anotação ou ignorar esse fato, abandonando tal ocorrência.

Caso o algoritmo seja executado mais de uma vez com o mesmo *corpus* de treinamento, terá como única consequência o incremento de ocorrências de cada palavra chave registrada.

4.3.2 Algoritmo de Atualização de um Corpus

O algoritmo ilustrado pela Figura 13 mostra o processo de atualização de um *corpus*, propagando as anotações aprendidas durante o treinamento do eAssigner. Esse algoritmo também implementa uma Busca em Largura (veja no Apêndice A – Inteligência Artificial).

Figura 13 - Algoritmo de atualização de corpus

```
// Atualização de um Corpus a partir do Banco de Conhecimentos
Conectar Banco de Conhecimentos
Carregar Banco de Conhecimentos na memória
Abrir Arquivo com o Corpus <Corpus>

Enquanto (Houver dados em <Corpus>) faça
    Obter bloco <o>

    Buscar chave no Banco de Conhecimentos (<o>)    // localiza a palavra original
    Se achar então
        Para cada tipo de anotação faça
            Se só tiver uma anotação  então  //  encontrou a palavra e não houve conflito
                Inserir anotação no <corpus>
            senão  // Mais de uma possibilidade para o mesmo tipo de anotação
                Mensagem com sugestões
                Avaliar resposta:
                   Caso 'S':  Aplicar a anotação selecionada
                   Caso 'I':  Ignorar e abandonar a ocorrência
                fim-da-avaliação
            fim-se // que testa anotação múltipla
        fim-para
    fim-se  // fim do tratamento dessa palavra chave

   obter novo registro em <Corpus>
fim-enquanto  // quando acabarem os dados, segue para a próxima linha

Fechar Banco de Conhecimentos
Salvar e Fechar arquivo <Corpus>

Mensagem "fim de atualização de <Corpus>"
Retornar
```

Fonte: Elaborado pelo autor

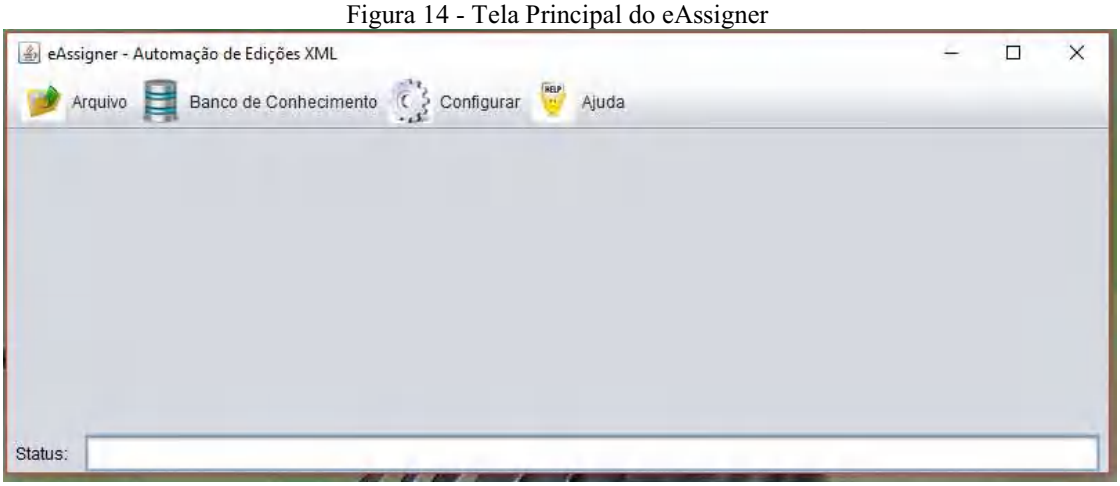
Para que o algoritmo funcione é preciso atender a dois requisitos: (a) a existencia de um banco de conhecimentos; e (b) um o *corpus* previamente editado por linguístas para ser analisar já tenha sofrido anotações por linguistas utilizando o *software* eDictor.

O algoritmo tem uma limitação conhecida: se for aplicado mais de uma vez no mesmo *corpus*, duplicará as anotações. Essa limitação será resolvida na próxima versão do eAssigner.

4.4 APRESENTAÇÃO DA INTERFACE

A interface com usuário foi desenvolvida utilizando os métodos da classe Swing, nativa da linguagem Java. Sua operação é intuitiva e guiada por mensagens. O módulo de Ajuda (Help) ainda não está disponível nessa versão. A tela principal é exibida na Figura 14.

Figura 14 - Tela Principal do eAssigner

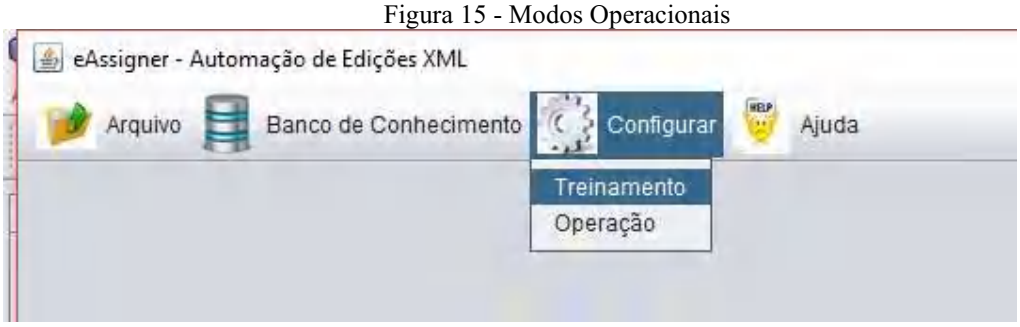


Fonte: Print screen no Sistema Operacional Windows 10

O *Software* tem dois modos operacionais conforme ilustrado na Figura 15:

- Modo Treinamento: para ler as marcações, aprender e registrar na Base de Conhecimento;
- Modo Operação: para ler e atualizar um *corpus* de acordo com as marcações e a base de conhecimento.

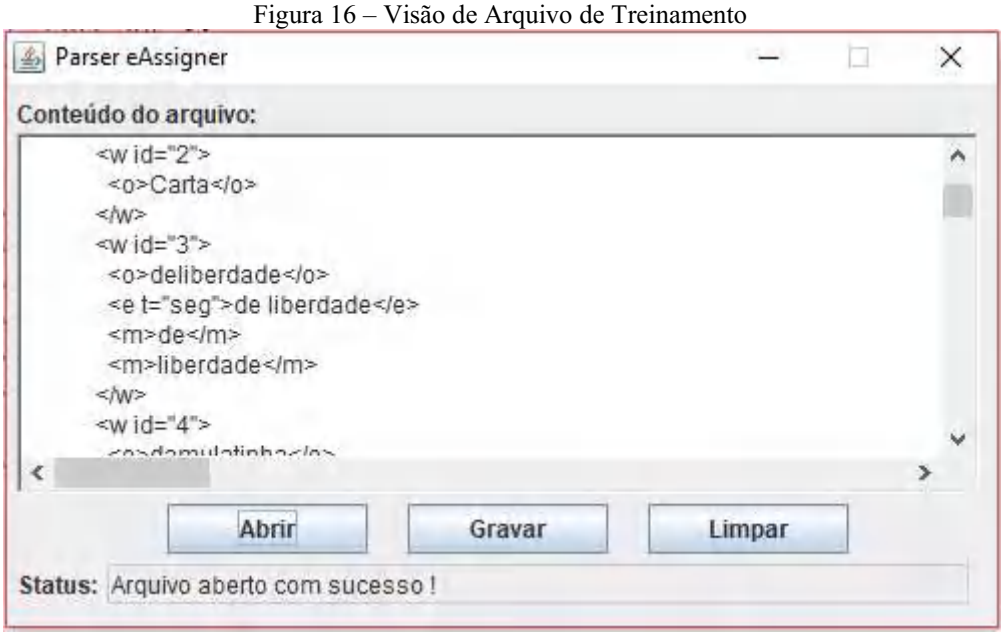
Figura 15 - Modos Operacionais



Fonte: Print Screen no Sistema Operacional Windows 10

No modo de Treinamento, ao abrir um arquivo para visualização, é exibida a tela da Figura 16. Durante o processamento do arquivo de treinamento, o banco de conhecimento é atualizado para registrar o aprendizado das marcações.

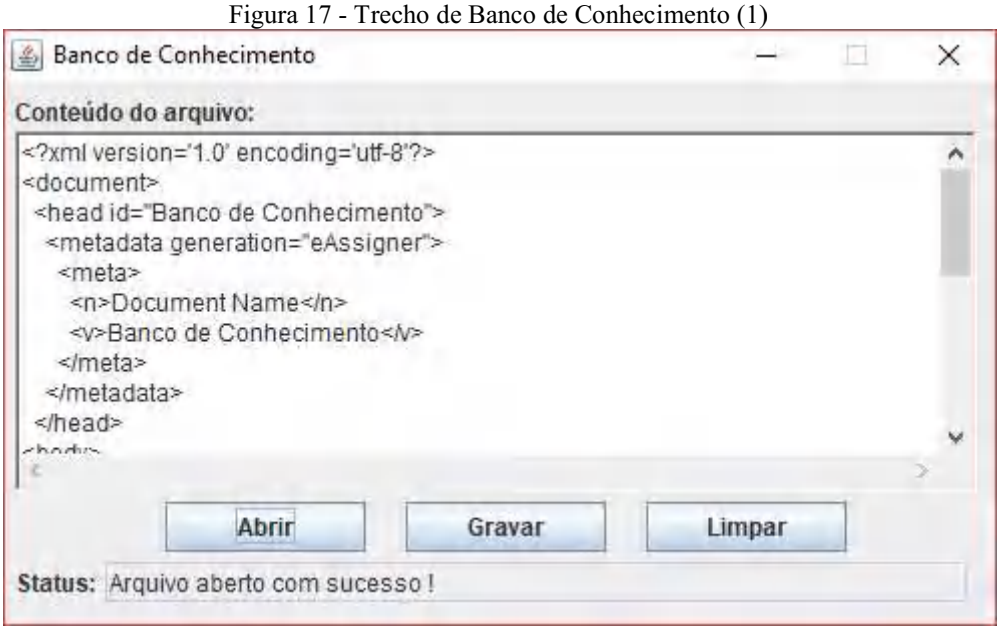
Figura 16 – Visão de Arquivo de Treinamento



Fonte: Print Screen do Aplicativo no Sistema Operacional Windows 10.

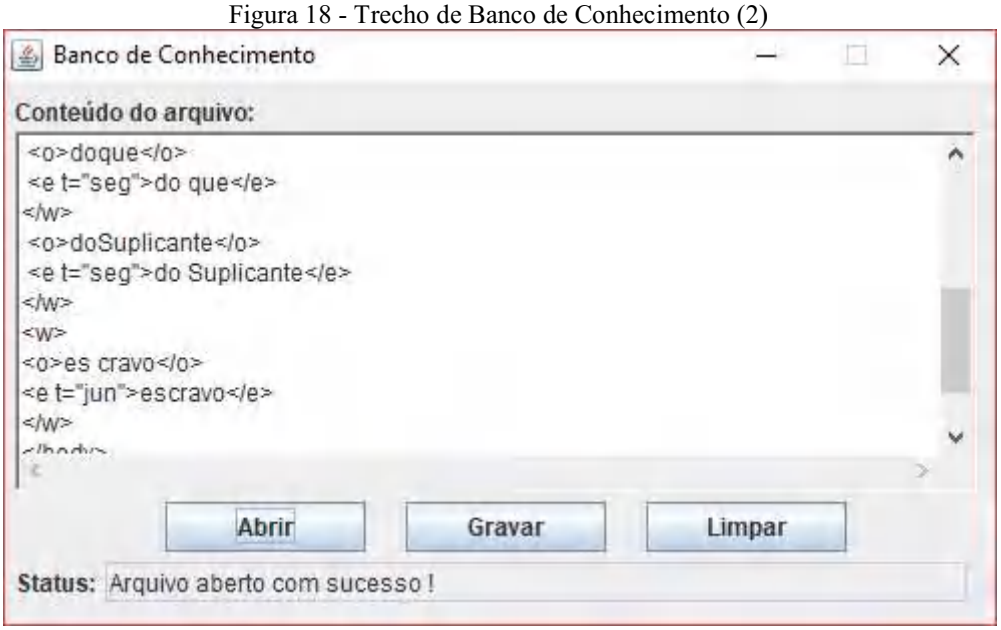
A Figura 17 e a Figura 18 mostram exemplos do banco de conhecimento gerado a partir do arquivo de treinamento utilizado.

Figura 17 - Trecho de Banco de Conhecimento (1)



Fonte: Print Screen do Aplicativo no Sistema Operacional Windows 10.

Figura 18 - Trecho de Banco de Conhecimento (2)



Fonte: Print Screen do Aplicativo no Sistema Operacional Windows 10.

Um aspecto importante a considerar é que o linguista não precisa conhecer a linguagem XML para a utilização nem do eDictor nem do eAssigner, pois esses *softwares* foram projetados e desenvolvidos para tornar o seu uso transparente.

Como o banco de conhecimentos atualmente é mantido em XML, nada impede que o linguista o atualize manualmente, desde que domine essa linguagem e a utilize de forma a não gerar uma má-formação de XML, o que pode causar mal funcionamento posterior.

5 TESTES E DISCUSSÃO DOS RESULTADOS PRLIMINARES

A presente seção apresenta testes realizados com o protótipo do eAssigner e discute os resultados encontrados. Os testes do protótipo foram realizados de forma a verificar as suas duas funcionalidades: Aprendizado e Anotação automática, conforme descrito nas próximas subseções. É importante registrar que a base de testes utilizada foi muito limitada. Para

5.1 TESTE DE APRENDIZADO

Para a realização dos testes, foram selecionados três documentos do *corpus* DOViC que haviam sido previamente anotados por linguistas. O Quadro 2 mostra a quantidade de palavras por documento. Cada palavra pode ter mais de uma anotação, sendo que apenas as anotações de "junção" e de "ressegmentação" são suportadas pela versão atual do eAssigner.

Quadro 2 - Corpus de Treinamento

DOCUMENTO	PALAVRAS
Carta 05 - parte 3 com edição	13
Carta 15 com edição	277
Carta 16 com edição	368
Total de palavras	658

Fonte: Elaborado pelo autor

Após gerar o Banco de Conhecimento, foi feita uma inspeção para verificar a sua acurácia e foi possível apontar algumas situações que requerem ajustes:

- Quando a mesma palavra tem anotação de segmentação e depois uma anotação de correção de grafia, o eAssigner registra a segmentação com a grafia original, o que certamente causará um problema posterior. O Quadro 3 ilustra a situação encontrada e o
-

- Quadro 4 ilustra como o eAssigner registrou no Banco de Conhecimentos.

Quadro 3 - Anotação de segmentação seguida por grafia

<o>deJezus</o>
<e t="seg">de Jezus</e>
<m v="P">de</m>
<m v="NPR">Jezus</m>
<e t="gra">de Jesus</e>

Quadro 4 - Registro no Banco

<o>deJezus</o>
<e t="seg">de Jezus</e>

Como é possível perceber, isso provocará um erro, pois a correção de grafia foi ignorada. Essa situação poderá ser corrigida ao implementar o recurso de reconhecimento de outros tipos de anotação no eAssigner.

- Outra situação percebida foi a duplicação de uma palavra chave conforme mostra o Quadro 5. Após análise da situação, foi possível perceber que uma das palavras (ocorrência 1) tinha um espaço em branco antes da tag de fechamento. Já a ocorrência 2 não tinha. Isso pode ter sido causado por alguma interferência manual no *corpus* de treinamento, pois o eDictor e o eAssigner não inserem espaço antes da tag. A situação pode ser facilmente resolvida modificando o Parser do eAssigner para eliminar os espaços em branco do final das palavras.
-

Quadro 5 - Duplicação de Palavra Chave

Ocorrência 1
<o>deliberdade </o>
<e t="seg">de liberdade </e>
Ocorrência 2
<o>deliberdade</o>
<e t="seg">de liberdade</e>

A Figura 19 mostra um segmento do banco de conhecimento produzido durante o treinamento.

Figura 19 - Segmento do Banco de Conhecimento

```
<w>
<o>es cravo</o>
<e t="jun">esoravo</e>
</w>
<w>
<o>Crio- lo</o>
<e t="jun">Criolo</e>
</w>
<w>
<o>deSouza</o>
<e t="seg">de Souza</e>
</w>
<w>
<o>esSuamulher</o>
<e t="seg">e Sua mulher</e>
</w>
<w>
<o>deliberdade</o>
<e t="seg">de liherdade</e>
</w>
<w>
<o>sede clara</o>
<e t="seg">se declara</e>
<e t="jun">sedeclara</e>
</w>
```

O total de 658 palavras do corpus de treinamento produziu um banco de conhecimento com 497 palavras. Foram identificados 74 registros incorretos, portanto o índice de acerto foi de 85%. Esse índice de acertos será melhorado após fazer os seguintes ajustes:

1. Incluir outros tipos de marcação no eAssigner;
2. Melhorar o Parser para remover espaços em branco desnecessários.

Mesmo com esses problemas, pode-se considerar o resultado obtido como bom.

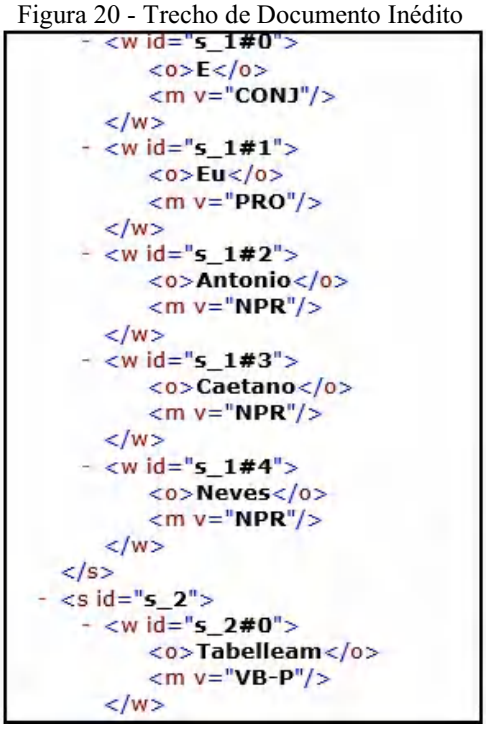
5.2 TESTE DE ANOTAÇÃO AUTOMÁTICA

Esse grupo de testes foi desenvolvido para verificar se o eAssigner consegue replicar em outros *corpora* as anotações aprendidas. Esses testes foram feitos em três etapas simulando situações diferentes:

5.2.1 Teste 1 – Arquivo 100% Inédito

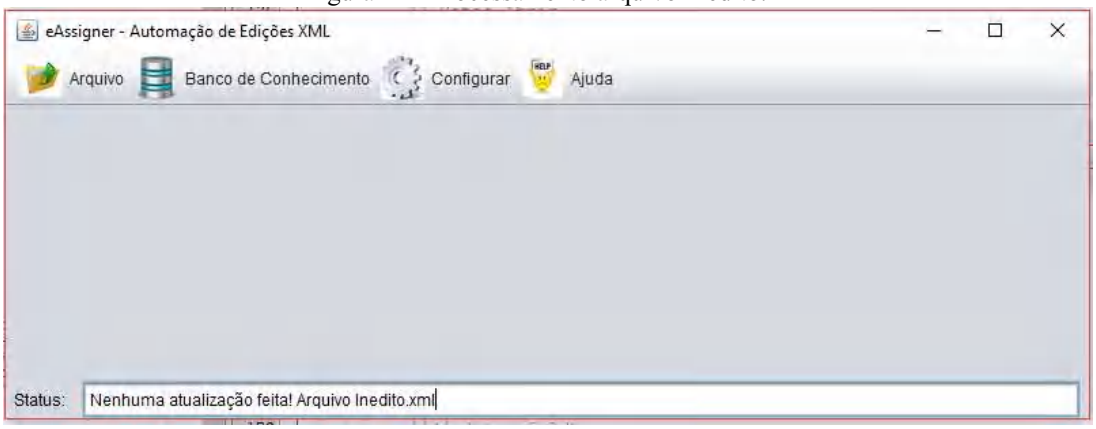
Para esse teste, foi selecionado um documento inédito (Inedito.xml) e foi feito um recorte de forma que nenhuma das anotações existisse no Banco de Conhecimento. O resultado esperado era que o eAssigner percorresse esse documento e não fizesse alterações. A Figura 20 mostra um trecho do documento preparado para o teste.

Figura 20 - Trecho de Documento Inédito



A Figura 21 mostra que o eAssigner não encontrou o que alterar no arquivo. Após inspeção e comparação com uma cópia feita antes do processamento, foi constatada a aprovação nesse teste preliminar.

Figura 21 - Processamento arquivo Inedito.xml



Fonte: Print Screen do software sob o Sistema Operacional Windows 10

Dentro do limite observável, esse teste foi executado sem problema.

5.2.2 Teste 2 – Arquivo 100% Conhecido

Para executar esse teste, foi utilizada uma cópia não anotada de um dos arquivos utilizados no treinamento. A ideia era testar se o eAssigner seria capaz de reproduzir todas as anotações registradas.

A Figura 22 é um recorte da Carta 15 com edição, que foi usada no treinamento do eAssigner. Esse arquivo foi escolhido por ter 277 palavras e tem todas as situações que foram apontadas durante o teste de aprendizado.

A Figura 23 é uma versão sem anotação do mesmo documento e foi feito um recorte no mesmo local para permitir a comparação. Esse arquivo foi **submetido ao eAssigne**r para fazer as anotações automáticas.

A Figura 24 é um recorte do documento gerado **após as anotações automáticas** feitas pelo eAssigner.

Figura 22 - Trecho do documento usado no treinamento

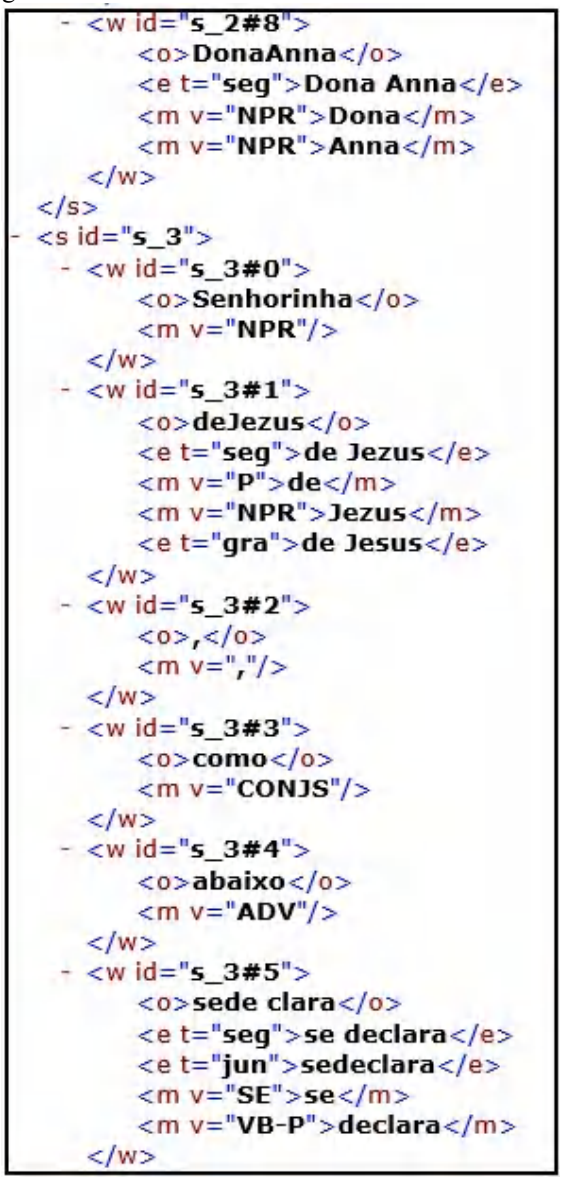
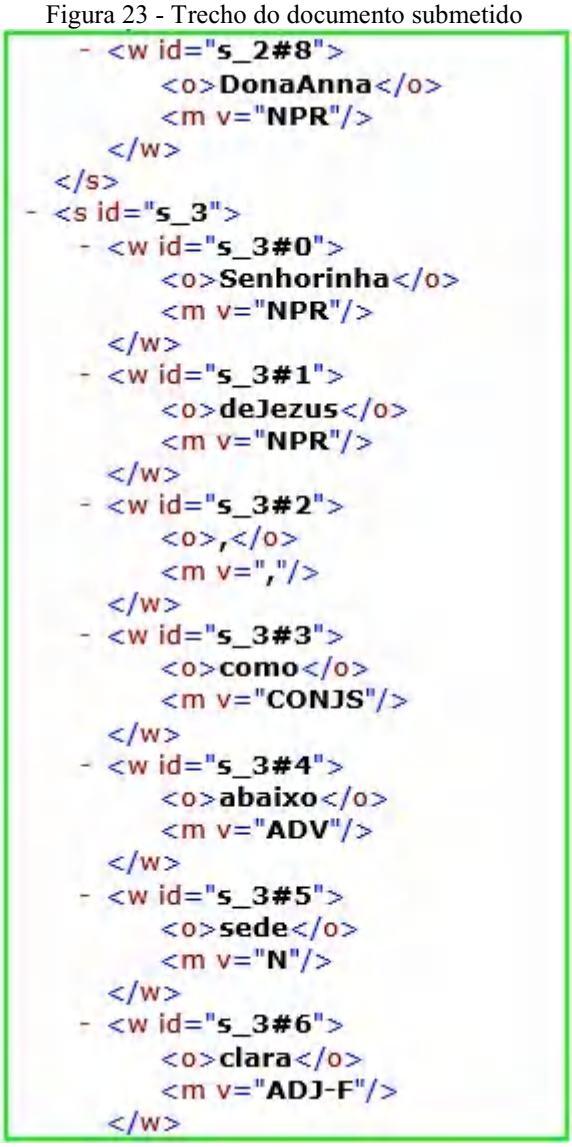
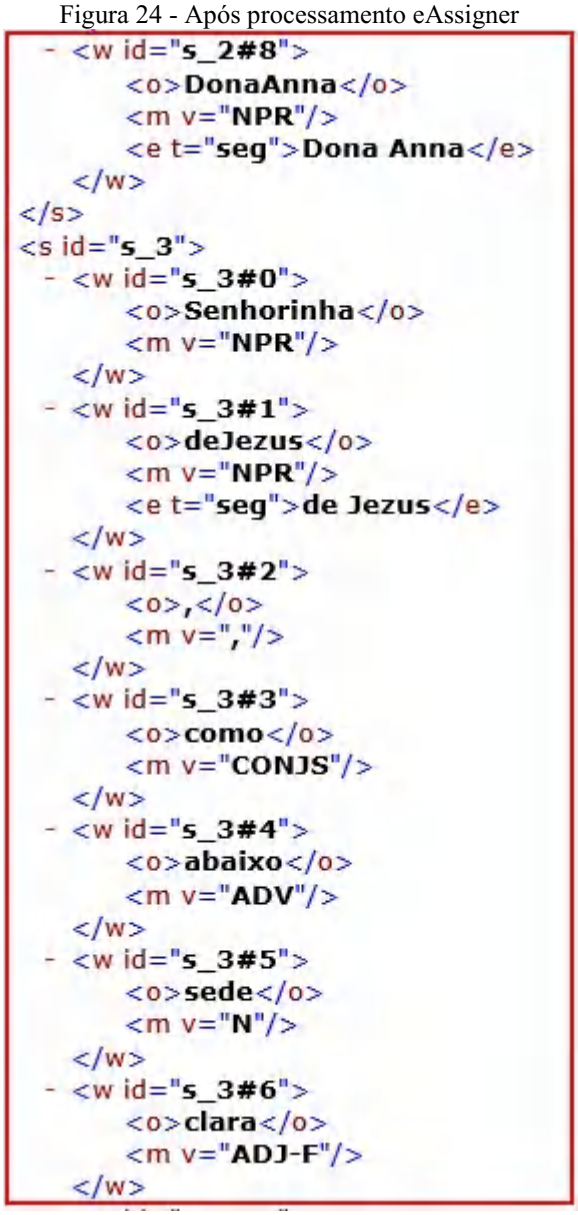


Figura 23 - Trecho do documento submetido



É importante observar que o recorte foi feito no mesmo trecho do documento para permitir comparações.

Figura 24 - Após processamento eAssigner



Como o eAssigner - na versão atual - lida apenas com as anotações do tipo “junção” e “ressegmentação”, era esperado que o arquivo refletisse apenas esses tipos de anotação.

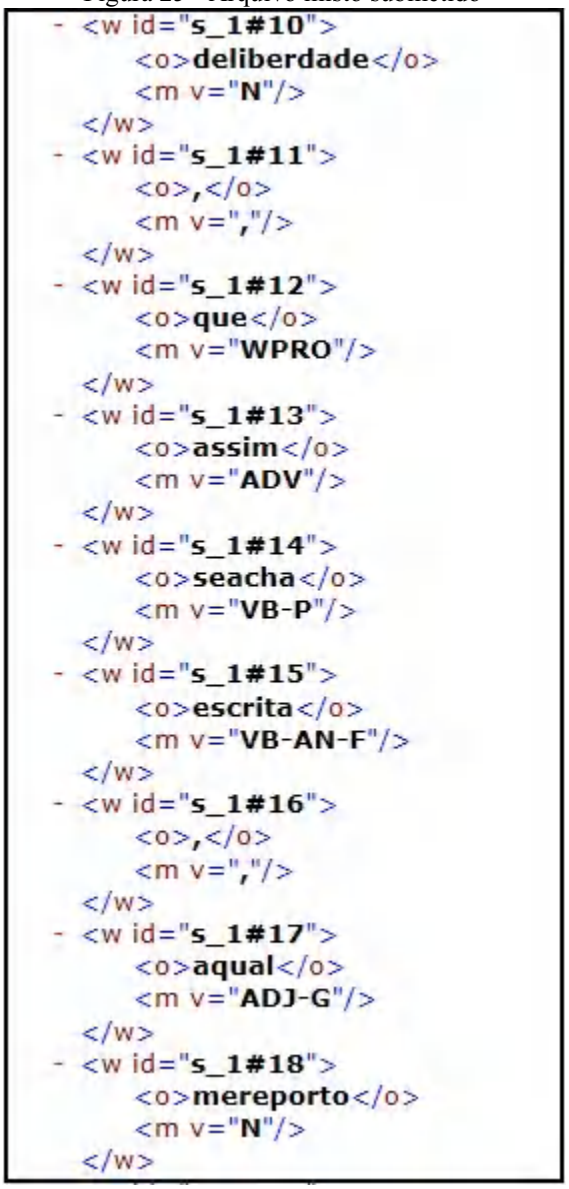
Observações feitas nos recortes dos documentos após esse teste:

1. A anotação “s2_8” (DonaAnna) foi feita corretamente;
2. Conforme esperado, a anotação “s_3#1” (deJezus) foi feita corretamente considerando apenas a segmentação.
3. A anotação “s_3#5 / s_3#6” (sedeclara) não foi feita porque não consta dessa forma no Banco de Conhecimento. O banco de conhecimento tem registrada a junção de <o>sede clara</o>, ou seja, as duas palavras a serem juntadas estão na mesma chave. Observe que no arquivo submetido são duas anotações.

5.2.3 Teste 3 – Arquivo Misto

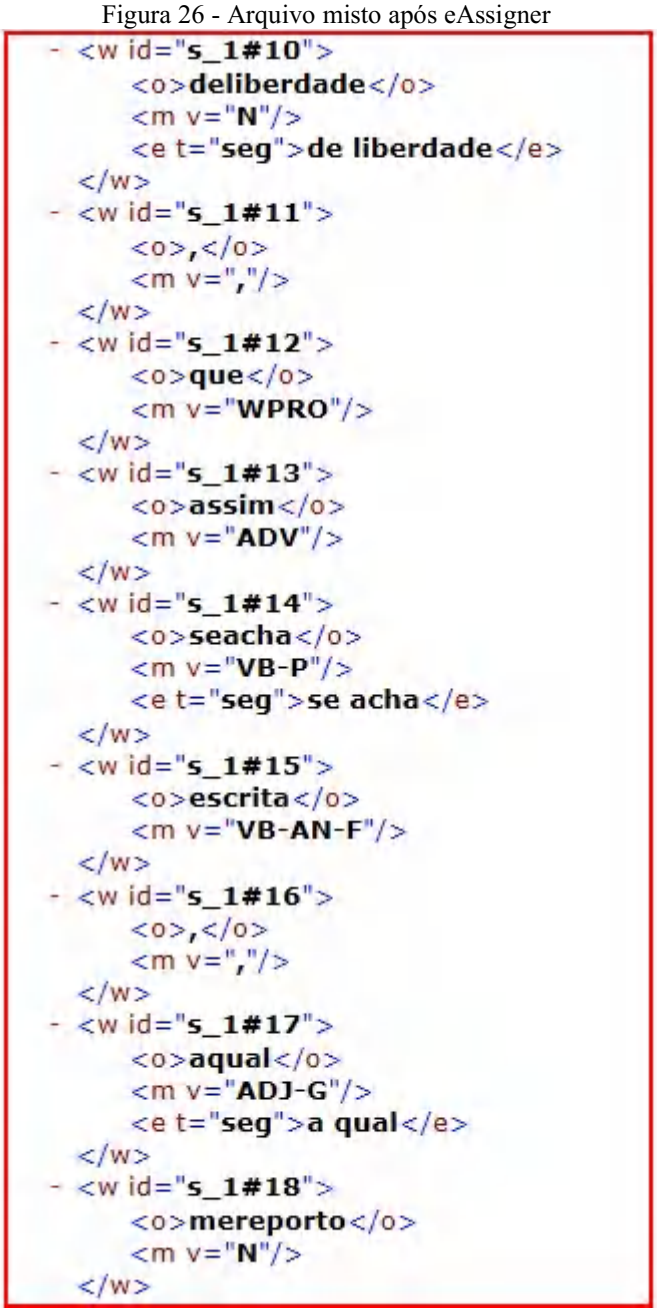
Para esse teste, foi selecionado um documento que não foi usado durante o treinamento, mas que possui palavras que foram registradas durante o treinamento. O resultado esperado desse teste é que as anotações registradas previamente sejam reproduzidas e as demais sejam ignoradas. A Figura 25 mostra um recorte de um arquivo que contém algumas anotações já aprendidas e outras não.

Figura 25 - Arquivo misto submetido



A Figura 26 mostra o resultado do processamento do eAssigner. Foi feito um recorte na mesma região do arquivo para permitir comparações.

Figura 26 - Arquivo misto após eAssigner



Observe que a anotação “s_1#18” (mereporto) não foi corrigida pelo eAssigner, pois não consta no Banco de Conhecimento. Já as anotações 10, 14 e 17 foram replicadas de forma correta em mais de um trecho do arquivo conforme esperado.

5.3 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Após os testes realizados, é possível tecer algumas considerações sobre o funcionamento do ptotótipo do eAssigner.

Quanto a operação de ressegmentação, o eAssigner realizou com sucesso tanto o aprendizado quanto a anotação automática. As anotações feitas pelo *software* são sempre gravadas no final do bloco da tag <w>, ou seja, antes do </w>.

Quanto a operação de junção, os testes apontam uma questão importante: se as palavras que devem ser juntadas estiverem no mesmo bloco <o>, o eAssigner consegue realizar a junção com sucesso. Entretanto, se estiverem em dois ou mais blocos, o *software* não consegue fazer a junção porque não identifica a palavra chave. Para esclarecer melhor, veja os exemplos a seguir:

```
<w ...>
<o>sepa  rado</o>
</w>
```

Nesse caso, é possível fazer a junção, pois os segmentos estão na mesma chave. Já o exemplo a seguir:

```
<w ...>
<o>sepa</o>
</w>
<w ...>
<o>rado</o>
</w>
```

O eAssigner não consegue fazer a junção, pois ele interpreta como duas palavras-chave. Deparamos com essa situação em vários arquivos, dificultando a automatização das edições de junção. Se as palavras forem colocadas na mesma chave, a junção funcionará normalmente.

Os resultados alcançados indicam que o eAssigner, apesar de algumas limitações da versão inicial, atende os seus principais objetivos:

1. Aprender anotações a partir de *corpus* de treinamento;
2. Anotar automaticamente em outros *corpora*, reproduzindo as anotações aprendidas.

5.4 LIMITAÇÕES CONHECIDAS

Conforme mencionado anteriormente, o protótipo do eAssigner tem atualmente as seguintes limitações:

1. Reconhece apenas anotações de junção e de ressegmentação.
2. Junções de palavras que estão registradas em chaves <o> diferentes não são reconhecidas.

3. Anotações que contêm uma chave no meio da chave <o> não são identificadas corretamente: <o>eanno<bk id="bk_1" t="p"/> </o>. Aparentemente, essas anotações ocorrem quando há quebra de linha no documento original. Para resolver essa limitação é preciso alterar o parser do eAssigner.

6 CONSIDERAÇÕES FINAIS

A partir dos estudos e resultados obtidos nesse trabalho, é possível demonstrar a importância atual das ferramentas de Tecnologia da Informação para facilitar e até viabilizar os trabalhos dos linguistas. Os *corpora* digitais são certamente de vital importância para os estudos diacrônicos da língua, para o estudo de hábitos e costumes dos povos, para o estudo da sua cultura etc. Para que esses estudos sejam realizados com sucesso, é preciso estudar e fazer anotações em bases de dados cada vez maiores.

O trabalho apresenta e discute a Linguística Computacional e a sua importância para o tratamento de bases de dados cada vez maiores. Para isso, foi feita a descrição do processo de edição filológica, tecendo comentários sobre as dificuldades e avanços da área. Também foi feita a descrição e comparação de várias ferramentas de edição e anotação de *corpora* e apresentadas suas características e limitações. No trabalho de pesquisa também foram feitas considerações sobre a importância de tecnologias de mineração de texto e de métodos de busca utilizados na Inteligência Artificial – IA, importante área de apoio para a Linguística Computacional. O estado da arte da IA foi apresentado no Apêndice A, incluindo o aprendizado de máquina;

Sobre o anotador automático eAssigner, um dos principais objetivos desse trabalho, foi feita a sua concepção e modelagem, descrevendo a sua arquitetura, seus módulos, principal diagrama de classes e principais algoritmos. Também foi desenvolvido um protótipo para testar os algoritmos e os seus resultados foram apresentados na seção 5.

Sobre a hipótese, embora os resultados do eAssigner ainda sejam incipientes, é possível perceber que a automação do processo de edição filológica em *corpora* digitais, tornará a tarefa de edição mais rápida e eficiente, diminuindo as inconsistências e consequentemente a incidência de erros na anotação linguística automática aplicada ao texto editado, gerada pelas inconsistências e erros na edição filológica.

Mesmo com as limitações conhecidas do eAssigner, os testes demonstraram que a anotação linguística automática é viável e acelera o processo de edição filológica.

PROJETOS FUTUROS

- IMEDIATO:
    - Desenvolver a fermenta eAssigner a partir do protótipo e das sugestões apresentadas, resolvendo limitações do *parser*, integrando os módulos do *software.*
- MÉDIO PRAZO:

- Incluir novos recursos de anotação como:
    - Grafia;
    - Modernização;
    - Expansão;
    - Correção;
    - Pontuação.
- Como o eDictor será modificado para funcionamento através da WEB, viabilizar essa mesma característica para o eAssigner.

REFERÊNCIAS

ALMIRO FILHO, Américo; XIMENES, Expedito Eloísio. Estudo de Documento Oitocentista. **Revista Philologus** / Círculo Fluminense de Estudos Filológicos e Linguísticos, ano 20, N. 59, (maio/ago. 2014) – Rio de Janeiro, 2014.

ALUÍSIO, Sandra Maria; ALMEIDA, Gladis Maria de Barcellos. O que é e como se constrói um corpus? **Calidoscópio** Vol. 4, n. 3, p. 156-178, set/dez 2006.

APPOLINÁRIO, Fábio. **Metodologia da Ciência**: filosofia e prática da pesquisa. 2ª ed. São Paulo: Cengage Learning, 2012.

AUROUX, Sylvain. **A filosofia da linguagem.** Tradução de José Horta Nunes. Campinas, SP: Editora da Unicamp, 1998.

_____. **A revolução tecnológica da gramatização**. Campinas, SP: Editora da Unicamp, 2001.

BENVENISTE, Émile. Da subjetividade na linguagem. In: **Problemas de Linguística Geral**. São Paulo: Ed. Nacional, Editora da Universidade de São Paulo, 1976.

BRASIL. **Lei n. 9609 de 19 de fevereiro de 1998**. Dispõe sobre a proteção da propriedade intelectual de programa de computador, sua comercialização no País, e dá outras providências. Disponível em http://www.planalto.gov.br/ccivil_03/leis/L9609.htm. Acesso em 10 jan. 2017.

CHATBOTS. **Virtual Agents / Chatbots Directory**. List of all chatbots (virtual assistants, chat bot, conversational agents, virtual agents) in the World. Disponível em https://www.chatbots.org/chatbot/eliza. Acesso em Dez. 2015.

COPPIN, Ben. **Inteligência Artificial**. Rio de Janeiro: LTC, 2010.

DACOS, Marin. **Manifesto das Humanidades Digitais**. Produzido em 26 mar. 2011. Poster publicado no THAT Camp 2012. Tradução de Hervé Théry. Disponível em https://humanidadesdigitais.org/manifesto-das-humanidades-digitais. Acesso em 10 jan. 2017.

DIAS-DA-SILVA, Bento Carlos. O estudo Linguístico-Computacional da Linguagem. **Letras de Hoje**. Porto Alegre. v. 41, nº 2, p. 103-138, junho, 2006. Disponível em http://revistaseletronicas.pucrs.br/ojs/index.php/fale/article/viewFile/597/428. Acesso em 13 out. 2015.

FARIA, Pablo; GALVES, Charlotte. Criando "Bancos de Árvores": O Sistema de Anotação e o Processo Automático. **Cadernos de Estudos Linguísticos**. Campinas: v. 58, n. 2 p. 299-315, maio/ago./2016. Disponível em http://revistas.iel.unicamp.br/index.php/cel/article/view/5133. Acesso em 30 dez. 2016.

GIRÃO, Márcio. Humanidades Digitais: quando o software conta a história através do tempo. In: **Timaior**, Rio de Janeiro. 11ª edição, dezembro, 2016. Disponível em

http://www.timaior.com.br/m/capa-portal-do-software-publico/intervalo-humanidades-digitais. Acesso em 12 fev. 2017.

GONÇALVES, Maria Filomena; BANZA, Ana Paula (coord.). **Património Textual e Humanidades Digitais**: da antiga à nova Filologia. Évora: CIDEHUS, 2013. ISBN: 978-989-95669-7-2.

HEXSEL, R. A. **SOFTWARE LIVRE**. Paraná: Universidade Federal do Paraná, 2002, disponível em < http://www.inf.ufpr.br/pos/techreport/RT_DINF004_2002.pdf >. Acesso em 02 fev. 2017.

HOUAISS, Antônio (Ed.). **Dicionário Houaiss da Língua Portuguesa**, Edição digital, Versão 3.0. Rio de Janeiro: Editora Objetiva, 2009. 1 CD ROM.

HOVY, Eduard; LAVID, Julia. Towards a 'Science' of Corpus Annotation: A New Methodological Challenge for Corpus Linguistics. **International Journal of Translation**. Califórnia: v. 22, n. 1, Jan-Jun 2010. Disponível em . Acesso em 20 jun. 2017.

LAKATOS, Eva Maria; MARCONI, Marina de Andrade. **Fundamentos de Metodologia Científica**. 6ª ed. São Paulo: Atlas, 2006.

LEECH, Geoffrey. *Adding Linguistic Annotation*. Lancaster University: 2004. In: WYNNE, Martin (editor). ***Developing Linguistic Corpora***: *a Guide to Good Practice*. ISSN 1463 5194. Virgínia (EUA): Oxbow Books, 2005.

LOMBARDO, Elena. **Representação do Conhecimento e Humanidades.** HD br: publicado em 08/08/2014. Disponível em <http://hdbr.hypotheses.org/5125#more-5125>. Acesso em 10 fev. 2017.

NAMIUTI, Cristiane (Coord.). **Novos meios para antigas fontes**: Sintaxe diacrônica em corpus eletrônico: do português pré-clássico às variantes modernas. Projeto de Pesquisa. UESB, Vitória da Conquista, 2010.

NAMIUTI, Cristiane (Coord.); SANTOS, Jorge Viana (Co-coordenador). **Memória Conquistense**: implementação de um corpus digital. CNPq 485098/2013-0. UESB, Vitória da Conquista, 2013. (Projeto de Pesquisa).

NAMIUTI-TEMPONI, Cristiane; COSTA, Aline Silva. Reflexões sobre anotação sintática e ferramentas de busca - Uso da linguagem XML para anotação sintática no corpus digital DOViC. **Letras & Letras**. v. 30, n. 2 (jul/dez. 2014) - ISSN 1981-5239. Disponível em http://www.seer.ufu.br/index.php/letraseletras. Acesso em 08 Jun. 2016.

NAMIUTI-TEMPONI, Cristiane; SANTOS, Jorge Viana. Novos desafios para antigas fontes: a experiência DOViC na nova linguística histórica. "**E-Book do Congresso de Humanidades Digitais em Portugal**: Construir pontes e quebrar barreiras na era digital – 2015". Lisboa: Universidade Nova de Lisboa, 2017 (no prelo).

NAMIUTI-TEMPONI, Cristiane; VIANA SANTOS, Jorge; LEITE, Cândida Mara Brito. **Propostas e Desafios dos Novos Meios das Antigas Fontes**: a preservação da memória pela Linguística de Corpus. Trabalho apresentado no IX Colóquio do Museu Pedagógico. UESB,

Vitória da Conquista: 2011. Disponível em <http://periodicos.uesb.br/index.php/cmp/article/viewFile/2717/2382>. Acesso em 2 ago. 2016. ISSN: 2175-5493.

OTHERO, Gabriel de Ávila. Linguística Computacional: uma breve introdução. **LETRAS DE HOJE** – ediPUCRS v. 41, n. 2 (2006) disponível em <http://revistaseletronicas.pucrs.br/ojs/index.php/fale/article/view/605>, acesso em 29 set. 2015.

OTHERO, Gabriel de Ávila; MENUZZI, Sérgio de Moura. **Linguística Computacional - princípios e aplicações**. São Paulo: Parábola Editorial, 2005.

PAIXÃO DE SOUSA, Maria Clara. **A anotação semiautomática de divergências de grafia como fundamento para o processamento automático de textos antigos**: Uma experiência na Brasiliana Digital. 18º Intercâmbio de Pesquisas em Linguística Aplicada, PUC, São Paulo, 2011.

_____. Memórias do Texto. In: **Revista Texto Digital**, ISSN 1807-9288, ano 2 n.1 2006. Disponível em http://www.textodigital.ufsc.br/num02/paixao.htm. Acesso em 02 jan. 2017.

_____. A Filologia Digital em Língua Portuguesa: alguns caminhos. In: GONÇALVES, Maria Filomena; BANZA, Ana Paula (coord.). **Património Textual e Humanidades Digitais**: da antiga à nova Filologia. Évora: CIDEHUS, 2013. p. 113-138. ISBN: 978-989-95669-7-2.

_____. **O Corpus Tycho Brahe**: contribuições para as humanidades digitais no Brasil. Filologia e Linguística Portuguesa, Brasil, v. 16, p. 53-93, dec. 2014. ISSN 2176-9419. Disponível em: <http://www.revistas.usp.br/flp/article/view/88404/91296>. Acesso em: 12 maio 2017. doi:http://dx.doi.org/10.11606/issn.2176-9419.v16ispep53-93.

PAIXÃO DE SOUSA, Maria Clara; KEPLER, Fabio Natanael; FARIA, Pablo Picasso Feliciano de. **E-dictor**: Novas perspectivas na codificação e edição de corpora de textos históricos. In: VIII Encontro de Linguística de Corpus, 2009, Rio de Janeiro. Resumos, 2009.

_____. **e-Dictor**. Versão 1.0 beta 10, 2013. Programa de Computador. Disponível em: <https://edictor.net/download>. Acesso em 01 jun.2016.

PÉREZ-PAREDES, Pascual (Coord.). **SACODEYL**. Múrcia, Espanha: Universidad de Murcia, 2008. Disponível em http://www.um.es/sacodeyl/en/pages/software.htm#annotator. Acesso em 17 jun.2017.

RIBEIRO, Rafael Dias. **Métodos Revogáveis de Busca**. Rio de Janeiro: 2011. Disponível em http://www.rafaeldiasribeiro.com.br/downloads/IC1_7.pdf. Acesso em 25 jun. 2017.

ROCHA, Anderson; DORINI, Leyza Baldo. **Algoritmos gulosos**: definições e aplicações. Campinas: 2004. Disponível em http://www.ic.unicamp.br/~rocha/msc/complex/algoritmosGulososFinal.pdf. Acesso em 25 jun. 2017.

RUSSELL, Stuart; NORVIG, Peter. **Inteligência Artificial**. 3ª ed. Rio de Janeiro: Elsevier, 2013 [e-book].

SANTOS, Jorge Viana. **Técnicas de transporte do texto manuscrito para o meio digital**. Trabalho apresentado na I Oficina de Linguística de Corpus da Bahia (UEFS, UESB, UFBA). Feira de Santana, Brasil, Dezembro 15-17, 2010.

SARDINHA, Tony Berber. Linguística de Corpus: Histórico e Problemática. **DELTA**, vol.16 nº 2, páginas 323-369. Scielo: São Paulo, 2000. Disponível em <http://www.scielo.br/scielo.php?pid=S0102-44502000000200005&script=sci_abstract&tlng=pt>. Acesso em 03 jul. 2016.

SILVEIRA, Paulo et al. **Introdução à Arquitetura e Design de Software**. Rio de Janeiro: Elsevier, 2012 [e-book].

SILVEIRA, S. A. da. **SOFTWARE LIVRE**: a luta pela liberdade de conhecimento. São Paulo: ed. Fundação Perseu Abramo, 2004.

SIMOV, Kiril et al. CLaRK - an XML-based System for Corpora Development. In: **Proc. of the Corpus Linguistics,** 2001. Conference, pages: 558-560.

SINCLAIR, John. Corpus and Text - Basic Principles. In: WYNNE, Martin (editor). **Developing Linguistic Corpora**: a Guide to Good Practice. ISSN 1463 5194. Virgínia (EUA): Oxbow Books, 2005.

STEPHAN, Druskat et al. **Atomic**: an open-source software platform for multi-layer corpus annotation. In Josef Ruppert and Gertrud Faaß (eds.): Proceedings of the 12th Konferenz zur Verarbeitung natürlicher Sprache (KONVENS 2014), Hildesheim, October 2014. 228–234. ISBN 978-3-934105-46-1. Disponível em <http://corpus-tools.org/atomic/>. Acesso em 16 jun. 2017.

TAURION, C. ***SOFTWARE* LIVRE**: Potencialidades e Modelos de Negócio. Rio de Janeiro: ed. Brasport, 2004.

TURING, Alan. Computing Machinery and Intelligence**. Mind**, vol. 59, nº 236, outubro, 1950, páginas 433-460, disponível em <http://loebner.net/Prizef/TuringArticle.html>, acesso em 10 out. 2015.

VELLOSO, Fernando de Castro. **Informática**: conceitos básicos. 7ª ed. revista e atualizada. Rio de Janeiro: Elsevier, 2004.

VIEIRA, Renata; LIMA, V.L.S. **Linguística Computacional**: princípios e aplicações. In: IX Escola de Informática da SBC-Sul. Luciana Nedel (Ed.) Passo Fundo, Maringá, São José. SBC-Sul, 2001.

WAZLAWICK, Raul Sidnei. **Metodologia de Pesquisa para Ciência da Computação**. Rio de Janeiro: Elsevier, 2008.

APÊNDICE A – INTELIGÊNCIA ARTIFICIAL

Essa seção apresenta um panorama da Inteligência Artificial (IA), incluindo uma breve introdução, sua evolução e aplicações e, finalmente, a relaciona com a Linguística. De forma alguma pretende ser um estudo exaustivo da IA, pois fugiria ao escopo desse trabalho.

INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL (IA)

De acordo com Coppin (2010), existem duas escolas de Inteligência Artificial: IA Forte e IA Fraca[14], assim definidas:

> Os seguidores da **IA forte** acreditam que, dispondo de um computador com suficiente capacidade de processamento e fornecendo a ele suficiente inteligência, pode-se criar um computador que possa literalmente pensar e ser consciente do mesmo modo que um ser humano é consciente.
> [...] por outro lado, a **IA fraca** é simplesmente a visão de que o comportamento inteligente pode ser modelado e utilizado por computadores para solucionar problemas complexos (COPPIN, 2010, p. 5).

O ponto de vista da IA fraca deixa claro que o fato de um computador ser programado para agir inteligentemente não prova que ele seja verdadeiramente inteligente no sentido humano. Coppin (2010) e Russell e Norvig (2013) esclarecem que a IA forte é, por enquanto, matéria explorada apenas pela ficção científica. Assim, ao longo do restante do texto, será considerada unicamente a IA fraca.

Outra questão que logo se apresenta é: O que é Inteligência Artificial? Para definir IA, Coppin (2010) propõe outra questão: o que é inteligência? E considera que a inteligência pode ser definida pelas propriedades por ela exibidas: "[...] uma capacidade de lidar com novas situações; a capacidade de solucionar problemas, de responder a questões, de engendrar planos e assim por diante" (COPPIN, 2010, p. 4). Em seguida, o autor apresenta a seguinte definição para IA: "Inteligência Artificial é o estudo dos sistemas que agem de um modo que a um observador qualquer pareceria ser inteligente" (COPPIN, 2010, p. 4). É limite do computador parecer inteligente, pois, para ser inteligente, o computador precisaria, mais do que processar dados, fazer juízos, conforme afirma Velloso (2004). Em 1950, ainda nos primórdios da computação, o matemático Alan Turing no seu artigo *Computing Machinery and Intelligence* já questionava se as máquinas poderiam pensar:

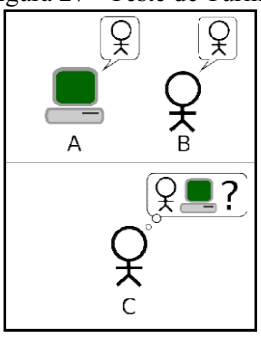
> I propose to consider the question, "Can machines think?" This should begin with definitions of the meaning of the terms "machine" and "think." The definitions might be framed so as to reflect so far as possible the normal use of the words, but this attitude is dangerous. If the meaning of the words "machine" and "think" are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning and the answer to the question, "Can machines think?" is to be sought in a statistical survey such as a Gallup poll. But this is absurd. Instead

[14] Nota do autor: não confundir com "métodos fortes" e "métodos fracos", que serão tratados posteriormente.

> of attempting such a definition I shall replace the question by another, which is closely related to it and is expressed in relatively unambiguous words[15] (TURING, 1950, p. 433).

Posteriormente, esse artigo foi reconhecido como um dos primeiros a questionar a possibilidade de criação da Inteligência Artificial[16], embora não com esse nome. Esse artigo propõe o chamado Teste de Turing ou "Jogo da Imitação", ilustrado na Figura 27. O jogo tem três elementos identificados como A, B e C (sendo que ou A ou B é um computador). O elemento C é um humano que fica isolado dos demais elementos e deve fazer perguntas que devem ser respondidas por A e por B. Então, com base nas respostas recebidas, deve identificar quem é o computador. Por outro lado, o computador (A no exemplo) deve ser programado de forma a responder às questões formuladas por C, imitando um ser humano para não ser identificado como um computador. Então, se isso for possível, o computador estaria se comunicando de forma inteligente[17].

Figura 27 - Teste de Turing



Fonte: Elaborado pelo autor. Adaptado de Turing (1950).

Para isso, o computador deveria compreender a linguagem humana, interpretar e emular o desempenho linguístico humano, conforme Dias-da-Silva (2006). O mesmo autor afirmou:

> A essa preocupação com a comunicação natural, que já se instalava nos círculos universitários norte-americanos e europeus, concomitantemente com a criação dos primeiros computadores, somou-se outra não menos complexa: a iniciativa voltada para o desenvolvimento de sistemas de tradução realizada automática ou semi-automaticamente por computador (DIAS-DA-SILVA, 2006, p. 104).

---

[15] Eu proponho considerar a questão, "As máquinas podem pensar?" Isso deve começar com definições do significado dos termos "máquina" e "pensar". As definições podem ser enquadradas de modo a refletir, na medida do possível, o uso normal das palavras, mas essa atitude é perigosa. Se o significado das palavras "máquina" e "pensar" for encontrado ao examinar como eles são comumente usados, é difícil escapar da conclusão de que o significado e a resposta à pergunta, "As máquinas podem pensar?", deve ser procurado em uma pesquisa estatística, como uma pesquisa Gallup. Mas isso é absurdo. Em vez de tentar essa definição, substituo a pergunta por outra, que está intimamente relacionada com ela e é expressa em palavras relativamente inequívocas (livre tradução do autor).

[16] O termo Inteligência Artificial foi cunhado somente em 1956 por John McCarthy em uma conferência de especialistas no Darmouth College.

[17] Nota do autor: até a presente data, nenhum computador conseguiu ser aprovado nesse teste.

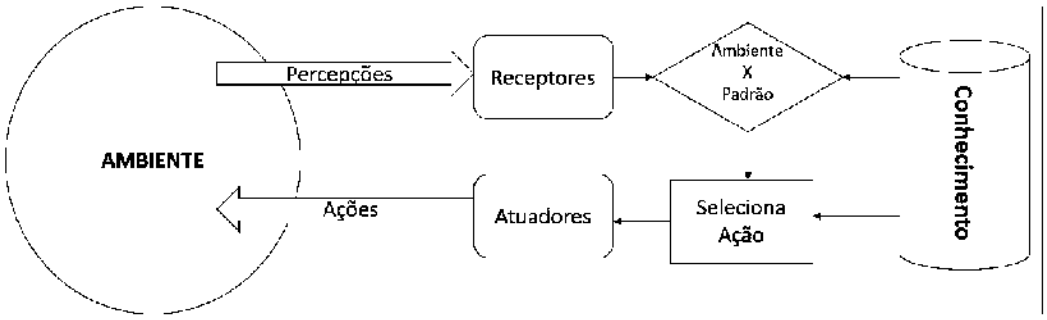
Ainda hoje, a tradução automática de textos é um importante desafio para os desenvolvedores de sistemas e linguistas. Apesar de já ter evoluído muito, há ainda muito a fazer nessa área.

A Inteligência Artificial (IA) tem sido estudada através de quatro estratégias, de acordo com Russell e Norvig (2013). Embora, em alguns aspectos, essas estratégias se ajudem, costumam ser correntes de pensamento que se opõem.

- Agindo de forma humana: é a estratégia clássica proposta pelo teste de Turing, já descrito. Prevê a programação dos computadores para que possam agir imitando os seres humanos a ponto serem confundidos com seres humanos, conforme prevê o referido teste.
- Pensando de forma humana: essa estratégia propõe que a IA deve fazer com que os computadores sejam programados com algoritmos de modelagem cognitiva, unindo modelos computacionais da IA com técnicas experimentais da psicologia para tentar construir algoritmos capazes de imitar o pensamento humano. De acordo com Russell e Norvig (2013), os pesquisadores Allen Newell e Herbert Simon desenvolveram em 1961 o "*General Problem Solver*" (Resolvedor Geral de Problemas), capaz de demonstrar teoremas usando a mesma estratégia do pensamento humano.
- Pensando racionalmente: também chamada de abordagem das "leis do pensamento", essa estratégia propõe que a IA resolva qualquer problema, desde que esteja representado de forma lógica. Essa estratégia usa como base o silogismo aristotélico que designou a argumentação lógica perfeita, constituída de três proposições declarativas que se conectam de tal modo que a partir das duas primeiras, chamadas premissas, é possível deduzir uma conclusão. Um clássico exemplo do silogismo é: "Todo homem é mortal; Sócrates é homem; Logo, Sócrates é mortal". De acordo com Russel e Norvig (2013), essa estratégia do pensamento racional enfrenta dois grandes problemas: nem sempre é possível representar o conhecimento informal usando a notação formal que é exigida pela lógica; e problemas que tenham algumas centenas de fatos podem esgotar os recursos computacionais de praticamente qualquer computador.
- Agindo racionalmente: também é conhecida como a "abordagem do agente racional". Essa estratégia considera a criação de programas chamados "agentes

racionais" que, dada uma sequência de percepções, segundo os seus conhecimentos, escolhem as melhores ações para atingir os seus objetivos, conforme ilustrado pela Figura 28.

Figura 28 - Agente Racional



Fonte: elaborado pelo autor, adaptado de Coppin (2010).

Conforme explicam Russel e Norvig (2013), a abordagem do agente racional tem duas vantagens sobre as demais: ela é mais geral do que a abordagem das "leis do pensamento" porque a inferência correta é apenas um dentre os vários mecanismos possíveis para se alcançar a racionalidade; em segundo lugar, é mais acessível ao desenvolvimento científico do que as estratégias baseadas no comportamento ou no pensamento humano.

Os agentes racionais serão descritos com mais detalhes na subseção AGENTES INTELIGENTES.

Para viabilizar a utilização da Inteligência Artificial é preciso levar em conta a necessidade de representar o conhecimento, conforme é descrito na próxima subseção.

REPRESENTAÇÃO E BUSCA

De acordo com Russel e Norvig (2013, p. 33), "[...] a representação do conhecimento é o estudo de como colocar o conhecimento em uma forma que o computador possa utilizar". Esse estudo tem grande impacto nos algoritmos desenvolvidos para buscar o conhecimento armazenado, podendo tornar os algoritmos mais ágeis ou simplesmente fazer com que certo algoritmo não funcione, conforme explica Coppin (2010).

REPRESENTAÇÃO DE CONHECIMENTO

Russel e Norvig (2013, p. 440) definem representação de conhecimento como "[...] o estudo de como colocar o conhecimento em uma forma que o computador possa utilizar".

Segundo Coppin (2010) e Russel e Norvig (2013), o computador necessita de uma boa representação do conhecimento para que possa resolver adequadamente um problema. Segundo os mesmos autores, essa representação precisa ser:

- Eficiente: de forma a não desperdiçar tempo e recursos computacionais;
- Útil: de forma a permitir que um computador possa resolver um problema;

- Significativa: de forma a descrever adequadamente o problema a ser resolvido.

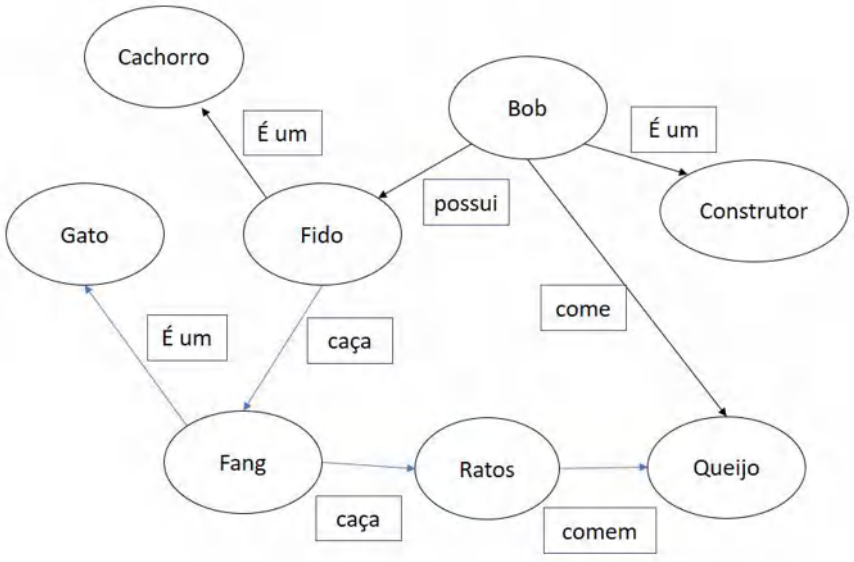
Russel e Norvig (2013) explicam que o principal objetivo da representação de conhecimento é representar aspectos do mundo real como ações, espaço, tempo etc. Essa é uma observação importante, pois, ao modelar um problema para solução computacional, é praticamente impossível representar tudo. Vem daí a necessidade de busca de uma representação eficiente, útil e significativa. A representação desses conceitos abstratos é chamada de Engenharia Ontológica.

A seguir são apresentadas algumas das técnicas utilizadas na IA para a representação e para a busca de conhecimentos.

REDES SEMÂNTICAS

De acordo com Coppin (2010), uma rede semântica é um **grafo** consistindo de **vértices** conectados por **arestas**. Os vértices representam os objetos e as arestas representam os relacionamentos entre os objetos. Um exemplo simples é mostrado na Figura 29.

Figura 29 - Rede Semântica simples



Fonte: Coppin (2010, p. 26)

É importante observar que as arestas são setas que indicam a direção do relacionamento. Por exemplo, Bob come queijo, Fido caça Fang etc. A rede semântica representa indivíduos específicos como Bob, Fido e Fang, mas também representa classes gerais de coisas como gatos e cachorros. Fang é uma instância da classe Gato e Bob é uma instância da classe Construtor.

Coppin (2010) afirma que as redes semânticas representam muito intuitivamente os conhecimentos sobre objetos, entretanto, têm dificuldade para representar negações como "Fido não é um gato".

QUADROS

Coppin (2010, p. 28) explica que “a representação baseada em quadros é um desenvolvimento de redes semânticas e nos permite explicar a ideia de herança”. Consiste de um conjunto de quadros interligados por relações; cada quadro tem um ou mais compartimentos aos quais são atribuídos valores de compartimento. O Quadro 6 representa a mesma rede semântica da Figura 29.

Quadro 6 - Conhecimento em forma de quadro

Nome do quadro	**Compartimento**	**Valor do Compartimento**
Bob	É um	Construtor
	Possui	Fido
	Come	Queijo
Fido	É um	Cachorro
	Caça	Fang
Ratos	Comem	Queijo
Fang	É um	Gato
	Caça	Ratos
Ratos		
Queijo		
Construtor		
Cachorro		
Gato		

Fonte: Coppin (2010, p. 29)

Uma importante utilização dos quadros é que esses podem ser úteis como estruturas de dados para sistemas especialistas.

ESPAÇOS DE BUSCA

Russel e Norvig (2013) e Coppin (2010) definem espaço de busca com uma representação de do conjunto de possíveis escolha de um dado problema, uma ou mais das quais é solução do problema. Conforme Coppin (2010), ao tentar localizar uma palavra específica em um livro com 100 páginas, cada uma das páginas será um espaço de busca. A página que contém a palavra desejada é chamada de alvo.

Pode-se então afirmar que o principal objetivo de algoritmos que utilizam espaços de busca é identificar o alvo com a maior rapidez e/ou com o menor custo.

ÁRVORES SEMÂNTICAS

Coppin (2010) define árvore semântica como um tipo de rede semântica que tem as seguintes propriedades que podem ser verificadas na Figura 30:

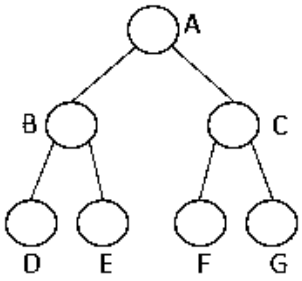
- Cada nó (exceto o nó raiz) tem exatamente um predecessor (pai) e um ou mais sucessores (filhos);
- Somente um nó não tem predecessor e é chamado de nó raiz (A no exemplo);

- Os nós que não têm sucessores são chamados de folhas (D, E, F, G no exemplo). Uma ou mais folhas são chamadas de alvos e representação de um estado em que a busca foi bem-sucedida.

Define-se também os seguintes elementos em uma árvore semântica:

- Ramo: é uma aresta que conecta dois nós da árvore;
- Fator de ramificação: é o número de sucessores de um nó. Se um nó tem 2 sucessores, diz-se que ele tem fator de ramificação 2 e assim por diante;
- Nível: é a altura do nó na árvore. O nó raiz fica no nível zero. Os sucessores do nó raiz ficam no nível 1.

Figura 30 - Árvore Semântica

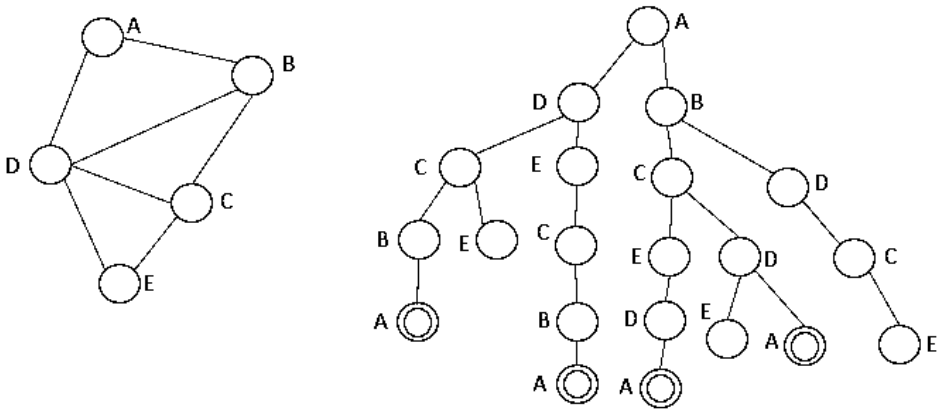


Fonte: Coppin (2010, p. 39)

ÁRVORE DE BUSCA

A busca em redes semânticas é uma pesquisa sistemática em seus nós para localizar um alvo. O caminho percorrido durante esse processo forma a árvore de busca. A Figura 31 mostra à esquerda uma rede semântica simples e, à direita, a árvore de busca correspondente, considerando que a busca sempre será iniciada a partir do nó **A**. Quando a busca retorna ao nó de partida, significa que houve uma busca cíclica, o que é inútil.

Figura 31 - Rede Semântica e Árvore de Busca



Fonte: Adaptado de Coppin (2010, p. 39-41)

Dessa forma, buscar um nó na árvore de busca corresponde a buscar um caminho completo na rede semântica.

De acordo com Coppin (2010), se um problema real tiver muitos elementos a considerar, a sua árvore de busca pode crescer exponencialmente de forma a inviabilizar o tratamento computacional. Esse tipo de problema é chamado de **explosão combinatória**. Ainda segundo o autor, em alguns casos, é possível resolver o problema subdividindo-o em pequenos problemas. Uma vez resolvidos os pequenos problemas, estará resolvido o problema maior. Esse tipo de problema é geralmente representado por um tipo de árvore chamado **Árvore e-ou**.

Uma vez definida a forma de representar/armazenar o conhecimento, são necessários métodos de busca para localizar a informação desejada. Alguns desses métodos são apresentados na próxima subseção.

MÉTODOS DE BUSCA

A busca é uma das técnicas da IA para a solução de problemas. Coppin (2010, p. 63) afirma que "[...] como os computadores tendem a funcionar de forma sequencial, a busca é necessária para determinar a solução para uma enorme gama de problemas". Pode-se definir **problema** como: um **objetivo** a ser alcançado e um **conjunto de ações** que podem ser praticadas para alcançar esse objetivo. Essas ações devem ocorrer dentro de um **espaço de busca**, já definido anteriormente.

Coppin (2010) e Russel e Norvig (2013) explicam que as seguintes propriedades devem ser consideradas ao estudar e definir o método de busca a ser utilizado:

- Complexidade: eficiência de um algoritmo em relação ao tempo ao espaço. Complexidade de tempo está relacionada ao tempo que o algoritmo leva para alcançar o objetivo, enquanto complexidade de espaço está ligada à quantidade de memória que o método necessita utilizar. É comum utilizar a notação **O** para descrever a complexidade de um método. Por exemplo, a busca em largura (que será apresentada mais tarde) tem complexidade de tempo $\mathbf{O(b^d)}$, onde **b** é o fator de ramificação da árvore de busca e **d** é a profundidade do nó objetivo na árvore. Interpretando: a profundidade e o fator de ramificação da árvore fazem o tempo de busca aumentar exponencialmente.
- Completude (também chamada de completeza): um método de busca é dito **completo** se ele garantir encontrar um estado objetivo, se existir algum. Evidentemente essa é uma característica altamente desejável, pois um método não completo pode não ser confiável se relatar que um problema não tem solução.

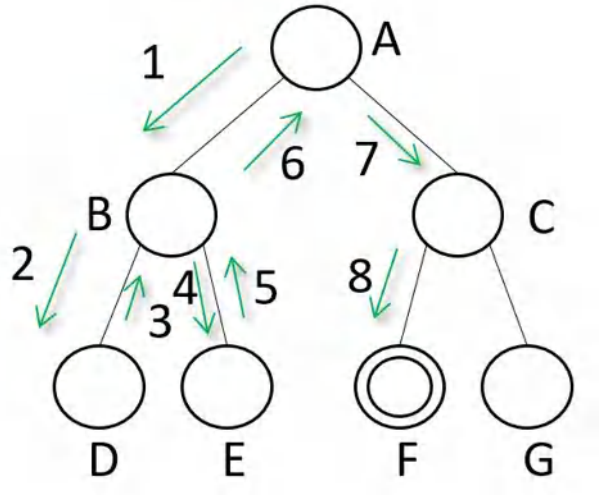
- Quanto a ser ótimo: um método de busca é **ótimo** se ele garantir encontrar a melhor solução que exista. Isso geralmente envolve a utilização de menor número de passos para atingir o objetivo.
- Irrevogabilidade: é a característica de métodos que buscam a solução sem retroceder. Métodos irrevogáveis correm o risco de encontrar uma solução "sub-ótima" para um problema, pois podem ser enganados ao encontrar uma solução localmente boa, mas que é menos favorável do que uma solução que esteja em outro local do espaço de busca (COPPIN, 2010).

A seguir são apresentados alguns métodos de busca. A abordagem escolhida foi a de mostrar os métodos, comentar suas características (vantagens e desvantagens), sem entrar em detalhes sobre os seus algoritmos, pois isso acrescentaria pouco ao objetivo dessa pesquisa.

BUSCA EM PROFUNDIDADE

De acordo com Coppin (2010), este é um algoritmo comumente utilizado e é assim chamado por seguir cada caminho até a sua maior profundidade antes de seguir para o próximo caminho. A Figura 32 ilustra um exemplo de busca em profundidade onde o nó objetivo é o "F". A busca em profundidade utiliza o método de **retrocesso cronológico** para voltar na árvore de busca sempre que um caminho sem saída é encontrado.

Figura 32 - Busca em Profundidade



Fonte: Elaborada pelo autor, adaptada de Coppin (2010)

A busca em profundidade é um exemplo de **busca de força bruta** ou **busca exaustiva**, conforme explica Coppin (2010, p. 67). Geralmente, computadores utilizam esse método para localizar arquivos em disco. O principal problema com esse método é que, se o ramo da árvore for muito grande, o computador poderá levar muito tempo para explorá-lo.

Coppin (2010) propõe um algoritmo de implementação de busca em profundidade mostrado no Quadro 7:

Quadro 7 - Algoritmo de Busca em Profundidade

```
function profundidade() {
        fila = [];    // inicializa uma fila vazia
        estado = no_raiz;  // inicializa o estado inicial
        while (true) {
            if eh_objetivo(estado)   // se atingiu o objetivo
                then return (SUCESSO)
            else inserir_na_frente_da_fila (sucessores (estado));
            if fila == []
                then report FALHA;
            estado = fila [0];  // estado = primeiro item na fila
            remover_primeiro_item (fila);
         }
 }
```

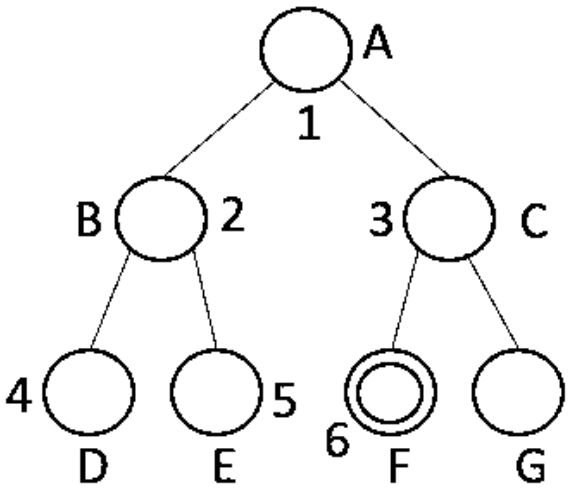
Fonte: Coppin (2010, p. 75)

O algoritmo inicialmente cria uma fila vazia e estabelece que o início da pesquisa se dará a partir do nó raiz. Cria uma repetição e, a cada iteração do laço, verifica se o estado (nó atual) é o objetivo. Se for verdade, encerra o algoritmo com sucesso. Se não for verdade, expande os sucessores no início da fila de busca. Caso a fila esteja vazia, significa que expandiu todos os sucessores e não encontrou o objetivo, o que implica em falha, ou seja, objetivo não foi encontrado.

BUSCA EM LARGURA

O método de busca em largura é uma alternativa ao de busca em profundidade. A Figura 33 ilustra a mesma árvore da Figura 32, porém executando a busca em largura. Pode-se perceber pela ilustração que a busca é feita por nível da árvore. A busca inicia a partir do nó raiz (nível 0), depois percorre todos os nós do nível 1, todos os nós do nível 2 e assim por diante até encontrar o objetivo ou falhar na busca.

Figura 33 - Busca em Largura



Fonte: Elaborada pelo autor, adaptada de Coppin (2010)

Coppin (2010) propõe o algoritmo de Busca em Largura apresentado no Quadro 8.

Quadro 8 - Algoritmo de Busca em Largura

```
function largura() {
        fila = [];    // inicializa uma fila vazia
        estado = no_raiz;  // inicializa o estado inicial
        while (true) {
            if eh_objetivo(estado)   // se atingiu o objetivo
                then return (SUCESSO)
            else inserir_no_final_da_fila (sucessores (estado));
            if fila == []
                then report FALHA;
            estado = fila [0];  // estado = primeiro item na fila
            remover_primeiro_item (fila);
         }
   }
```

Fonte: Coppin (2010, p. 76)

O algoritmo inicialmente cria uma fila vazia e estabelece que o início da pesquisa se dará a partir do nó raiz. Cria uma repetição e, a cada iteração do laço, verifica se o estado (nó atual) é o objetivo. Se for verdade, encerra o algoritmo com sucesso. Se não for verdade, expande os sucessores no final da fila de busca. Caso a fila esteja vazia, significa que expandiu todos os sucessores e não encontrou o objetivo, o que implica em falha, ou seja, objetivo não foi encontrado.

Comparando os resultados das buscas nos exemplos apresentados, percebe-se que, nesse caso, a busca em largura foi mais eficaz, pois encontrou o nó objetivo no sexto acesso, enquanto a busca em profundidade encontrou apenas no oitavo acesso. De forma alguma isso significa que a busca em largura é mais eficaz sempre. Conforme explica Coppin (2010), a busca em largura é mais eficaz quando a árvore tem poucas ramificações e quando o nó alvo (o objeto de busca) estiver em um nível não muito profundo. O Quadro 9 compara a busca em profundidade e a busca em largura em diversas situações, conforme Coppin (2010).

Quadro 9 - Comparação das buscas em profundidade e em largura

CENÁRIO	BUSCA EM PROFUNDIDADE	BUSCA EM LARGURA
Alguns caminhos são muito longos ou mesmo infinitos	Funciona mal	Funciona bem
Todos os caminhos têm comprimentos parecidos	Funciona bem	Funciona bem
Todos os caminhos têm comprimentos parecidos e todos os caminhos levam a um estado objetivo	Funciona bem	Desperdício de tempo e memória
Alto fator de ramificação	Desempenho depende de outros fatores	Funciona precariamente

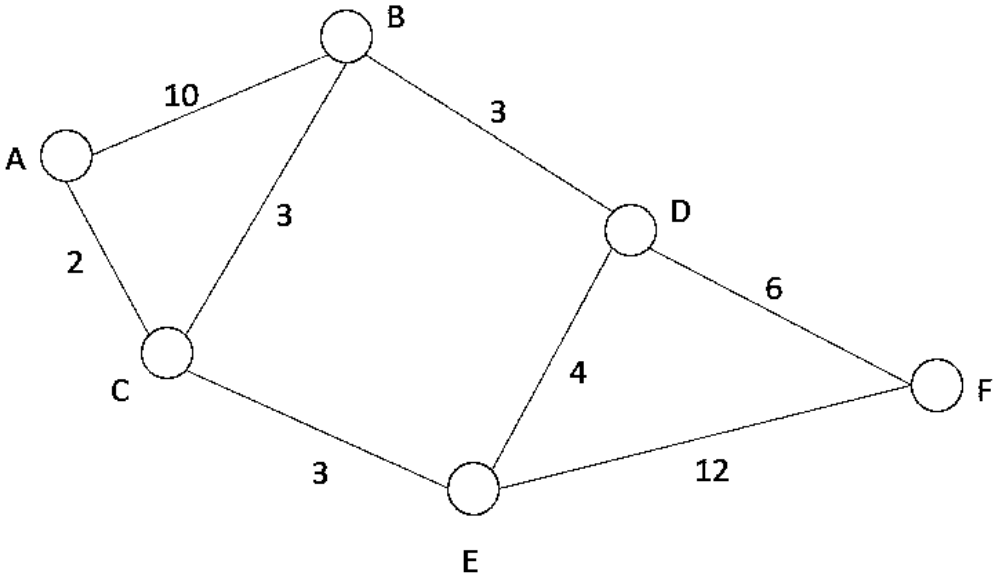
Fonte: Coppin (2010, p. 69)

Para estudar outros métodos de busca, inicialmente será apresentado um problema clássico da IA: o problema do caixeiro viajante (modificado).

O PROBLEMA DO CAIXEIRO-VIAJANTE

De acordo com Coppin (2010), para o estudo de métodos de busca heurística, é usual que sejam apresentadas situações do mundo real. Então será usado o problema do caixeiro-viajante, ilustrado na Figura 34 (a figura não está em escala). O objetivo do caixeiro é encontrar o menor caminho entre as cidades A (ponto de partida) e F (ponto de chegada). Esse problema é chamado de modificado porque, no problema original, o caixeiro-viajante deve percorrer várias cidades e depois retornar ao ponto de partida. No diagrama apresentado, cada nó representa uma cidade (de A até F) e cada aresta representa a estrada que liga duas cidades, indicando a distância entre elas.

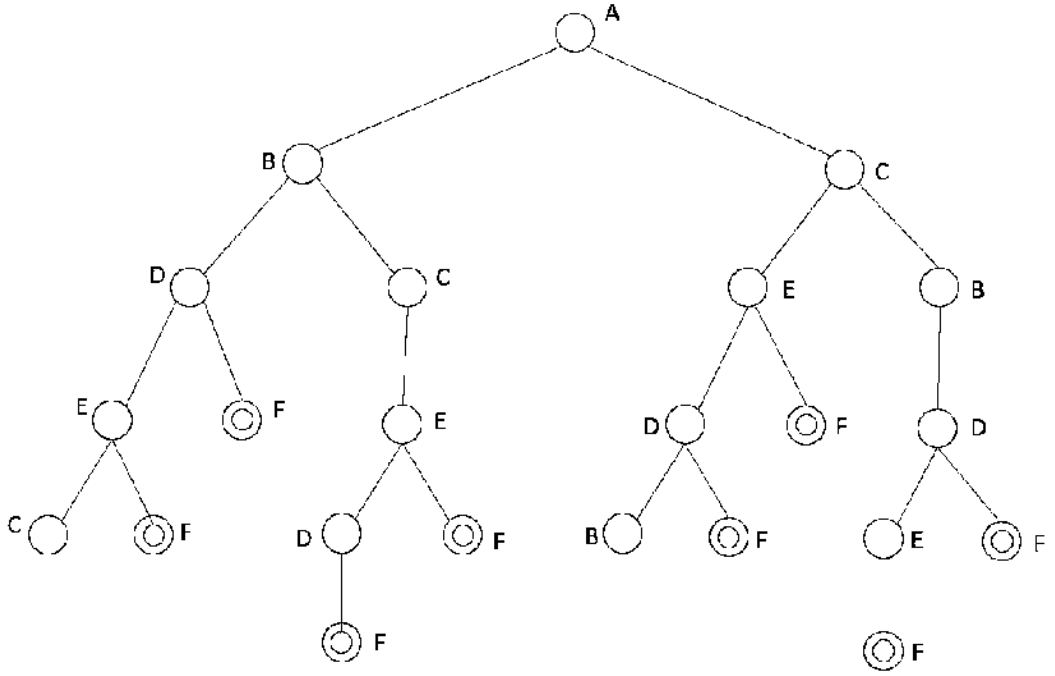
Figura 34 - Problema do Caixeiro-Viajante



Fonte: Coppin (2010, p. 85)

Esse diagrama é uma rede semântica. Assim, pode-se construir uma árvore para representar o espaço de busca, conforme ilustra a Figura 35.

Figura 35 - Árvore de Busca do problema do Caixeiro-Viajante



Fonte: Coppin (2010, p. 86)

A árvore tem dez nós folhas, oito dos quais são nós objetivos (F). Os outros dois são caminhos cíclicos, portanto não devem ser considerados. Os caminhos que vão de A até F com sucesso são: ABDEF, ABDF, ABCEDF, ABCEF, ACEF, ACBDEF, ACBDF e ACEDF. Os dois caminhos cíclicos são: ABDEC (que levaria de volta para A ou B) e ACEDB (que levaria de volta para A ou C).

De acordo com Coppin (2010), uma busca em profundidade apresentaria o caminho ABDEF como solução e que tem um comprimento total de 29. A busca em largura forneceria o caminho que tem o menor número de passos, mas não necessariamente o caminho mais curto. Nesse caso, produziria ABDF que tem o comprimento de 19, o que é bem menos do que o resultado da busca em profundidade, entretanto, o caminho mais curto é o ACBDF que tem o comprimento de 14.

Explicado o problema, serão estudados métodos de busca que utilizam heurística em busca de otimização da solução.

SUBIDA DA COLINA

De acordo com Russel e Norvig (2013, p. 122), o algoritmo é um "laço que percorre os nós de forma a se mover continuamente à procura de um nó com valor maior, ou seja, subindo a colina. O algoritmo para quando nenhum nó vizinho tem um valor maior do que o nó atual, o que seria o 'pico' da colina".

Coppin (2010) afirma que esse método tem uma variação chamada SUBIDA DA COLINA pela Encosta de Maior Aclive. A única diferença entre esse e o método original é

que, enquanto o original procura o PRIMEIRO nó com valor maior do que o atual, o método de maior aclive procura em todas as direções até achar o nó com maior valor nas redondezas. Esse algoritmo é representado no Quadro 10.

Quadro 10 – Algoritmo Subida da Colina

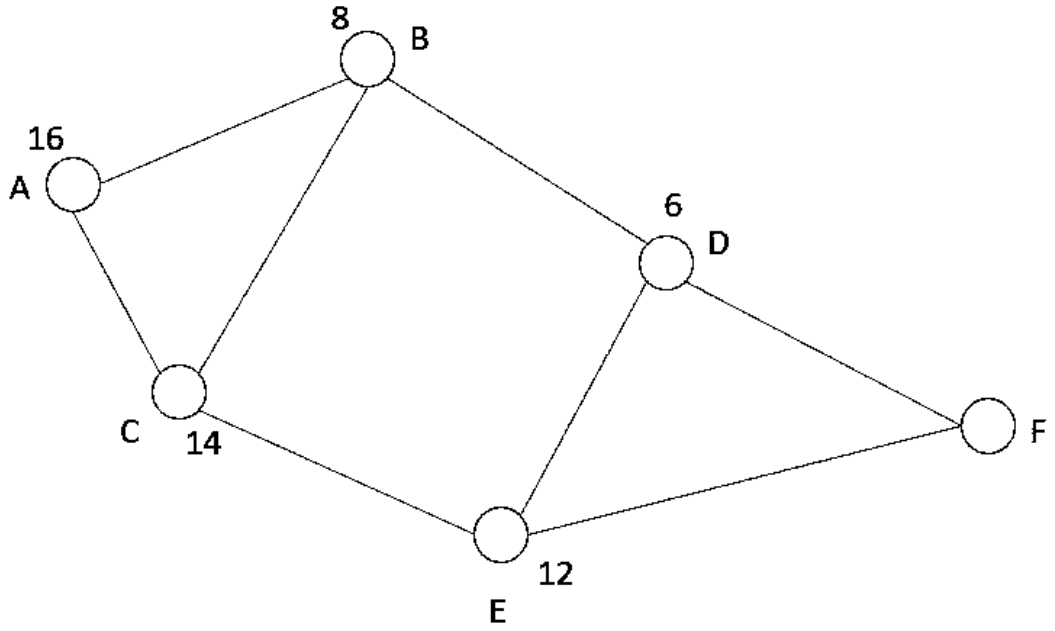
```
function colina() {
        fila = [];    // inicializa uma fila vazia
        estado = no_raiz;  // inicializa o estado inicial
        while (true) {
            if eh_objetivo(estado)   // se atingiu o objetivo
                then return (SUCESSO)
            else {
               ordenar (sucessores(estado));
               inserir_na_frente_da_fila (sucessores (estado));
             }
            if fila == []
                then report FALHA;
            estado = fila [0];  // estado = primeiro item na fila
            remover_primeiro_item (fila);
         }
   }
```

Fonte: Coppin (2010, p. 88)

O algoritmo percorre o espaço de forma similar à busca em profundidade, porém, a cada passo, escolhe caminhos que parecem ser mais prováveis de levar ao objetivo. O algoritmo inicialmente cria uma fila vazia e estabelece que o início da pesquisa se dará a partir do nó raiz. Cria uma repetição e, a cada iteração do laço, verifica se o estado (nó atual) é o objetivo. Se for verdade, encerra o algoritmo com sucesso. Se não for verdade, **ordena os sucessores do estado atual (para obter o de maior ou de menor valor** de acordo com o tipo de problema**)** e expande os sucessores no início da fila de busca. Caso a fila esteja vazia, significa que expandiu todos os sucessores e não encontrou o objetivo, o que implica em falha, ou seja, objetivo não foi encontrado.

Para aplicar ao problema do caixeiro-viajante, a rede semântica deve ser modificada para indicar a distância (em linha reta) entre cada cidade e a cidade destino, conforme ilustra a Figura 36.

Figura 36 - Rede Semântica com distâncias de cada cidade até o destino



Fonte: Coppin (2010, p. 88)

Aplicado o algoritmo de subida da colina ao problema, seria obtido como resposta o caminho ABDF (comprimento 19), o mesmo resultado da busca em largura, embora não seja a melhor solução.

BUSCA PELA MELHOR ESCOLHA

Também chamado de Busca pelo Primeiro Melhor, "[...] esse método emprega uma heurística similar ao método de Subida da Colina. A diferença é que no Primeiro Melhor, a fila inteira é ordenada após receber a inserção de novos caminhos, em vez de receber um conjunto de caminhos ordenados" (COPPIN, 2010, p. 90). Em seguida é apresentado no Quadro 11**Erro! Fonte de referência não encontrada.** um algoritmo para a implementação do método de busca pela melhor escolha.

Quadro 11 - Algoritmo Busca Melhor

```
function melhor() {
        fila = [];    // inicializa uma fila vazia
        estado = no_raiz;  // inicializa o estado inicial
        while (true) {
            if eh_objetivo(estado)   // se atingiu o objetivo
                then return (SUCESSO)
            else {
               inserir_na_frente_da_fila (sucessores (estado));
               ordenar (fila); // ordena a fila toda
             }
            if fila == []
                then report FALHA;
            estado = fila [0];  // estado = primeiro item na fila
            remover_primeiro_item (fila);
         }
   }
```

Fonte: Coppin (2010, p. 92).

O algoritmo inicialmente cria uma fila vazia e estabelece que o início da pesquisa se dará a partir do nó raiz. Cria uma repetição e, a cada iteração do laço, verifica se o estado (nó atual) é o objetivo. Se for verdade, encerra o algoritmo com sucesso. Se não for verdade, expande os sucessores no início da fila de busca e **depois** ordena toda a fila. Caso a fila esteja vazia, significa que expandiu todos os sucessores e não encontrou o objetivo, o que implica em falha, ou seja, objetivo não foi encontrado.

Aplicado ao problema do caixeiro-viajante, produz como resultado o caminho ABDF que chega ao resultado, embora não seja o caminho mais econômico.

Existem algoritmos que são focados na identificação de caminhos ótimos. Os principais são: A*, Busca de Custo Uniforme e Busca gulosa, abordados nas próximas subseções.

ALGORITMOS A*

A busca A* (lê-se A estrela) é ainda uma das técnicas de busca mais usadas por ser uma estratégia completa e ótima. Consiste em expandir o nó que possui a melhor função de avaliação, ou seja, a soma do custo e a avaliação heurística. Tenta minimizar o custo total da solução combinando a Busca Gulosa, que é econômica, porém não é completa nem ótima, e a Busca de Custo Uniforme, que é ineficiente, porém é completa e ótima.

Avalia os nós combinando o custo para alcançar cada nó e o custo estimado para ir do nó atual até o objetivo, assim representado: f(nó) = g(nó) + h(nó), onde g(nó) é o custo do caminho que leva ao nó atual e h(nó) é uma subestimativa da distância deste nó até um estado objetivo, conforme explicam Russel e Novig (2013).

Um exemplo de pseudocódigo para implementar o método de busca A* é mostrado no Quadro 12.

Quadro 12 - Algoritmo A* - pseudocódigo

```
Q = conjunto de nós a serem pesquisados;
S = o estado inicial da busca
Faça:

Inicialize Q com o nó de busca (S) como única entrada;
Se Q está vazio, interrompa. Se não, escolha o melhor elemento de Q;
Se o estado (n) é um objetivo, retorne n;
(De outro modo) Remova n de Q;
Encontre os descendentes do estado (n) que não estão em visitados e crie todas as
extensões de n para cada descendente;
Adicione os caminhos estendidos a Q e vá ao passo 2;
```

Fonte: adaptado de Russel e Novig (2013)

Dificilmente superestima o custo verdadeiro de uma solução. Encontra uma solução ótima, mas gera menos nós que os outros métodos de busca ótimos, e tem complexidades de tempo e espaço exponenciais.

BUSCA DE CUSTO UNIFORME

De acordo com Coppin (2010), a **busca de custo uniforme** (também conhecida como **busca** ordenada) é uma variação da **busca pelo primeiro melhor** que usa a função de avaliação g(nó) que mede o custo do caminho que leva ao nó atual. Ou seja, é um algoritmo A* onde o h(nó) é zerado, não levando em conta a distância do nó corrente até um nó objetivo.

Esta busca é completa e é ótima. Será ótima desde que, dado um nó **m** que tenha um sucessor **n**, o custo do antecessor é menor do que o custo do sucessor, ou seja, $g(m) < g(n)$. Um algoritmo de implementação da busca é apresentado no Quadro 13**Erro! Fonte de referência não encontrada.**.

Quadro 13 - Algoritmo de Busca de Custo Uniforme

```
resposta = nulo
lista-de-abertos = estado inicial
lista-de-fechados = nulo
sucesso = falso
enquanto (sucesso = falso) e (lista-de-abertos = vazio) faça
        nó-candidato = elemento de menor custo da lista-de-abertos
        remova o elemento de menor custo da lista-de-abertos
        coloque-o em lista-de-fechados
        se nó-candidato é solução
         então
                sucesso = verdadeiro
                resposta = nó-candidato
         senão
                expanda nó-candidato
                coloque o(s) nó(s) filho(s) na lista-de-abertos
        fim-se
fim-enquanto
retorna sucesso e reposta
```

Fonte: Ribeiro (2011)

O método inicia expandindo o nó raiz. Se nenhum dos nós filhos for alguma solução, então aquele que tiver o menor custo de obtenção é expandido. Se alguma solução ainda não tiver sido encontrada, então de todos os nós abertos, expande-se aquele que tiver o menor custo e assim por diante. Ressalta-se que os nós abertos podem estar em qualquer profundidade. A busca d custo uniforme não leva em consideração a posição de um nó, mas sim o seu custo.

BUSCA GULOSA

Conforme Coppin (2010, p. 97), “a busca gulosa é uma variação do algoritmo A*, onde a g(nó) é zerada de tal modo que apenas h(nó) é utilizada para avaliar caminhos apropriados”. Assim, o algoritmo tende a selecionar o caminho com menor custo heurístico ou menor distância estimada até o objetivo.

Conforme explicam Rocha e Dorini (2004), de forma geral, os algoritmos gulosos e os problemas por eles resolvidos são caracterizados pelos itens abordados a seguir:

- Há um problema a ser resolvido de maneira ótima, e para construir a solução existe um conjunto de candidatos;
- Durante a “execução” do algoritmo são criados dois conjuntos: um contêm os elementos que foram analisados e escolhidos;
- Há uma função que verifica se um conjunto de candidatos produz uma solução para o problema;
- Há uma segunda função que verifica a viabilidade do conjunto de candidatos.

O Quadro 14 mostra um exemplo de algoritmo guloso desenvolvido pelos autores.

Quadro 14 - Algoritmo Guloso Genérico

```
1: função ALGORITMOGULOSO(C: conjunto)          ▷ C é o conjunto de candidatos
2:     S ← ∅                                    ▷ S é o conjunto que irá conter a solução
3:     enquanto C ≠ ∅ e não solução(S) faça
4:         x ← seleciona C
5:         C ← C {x}
6:         se é viável S ∪ {x} então
7:             S ← S ∪ {x}
8:         fim se
9:     fim enquanto
0:     se solução(S) então
1:         retorne S
2:     senão
3:         retorne "Não existe solução!"
4:     fim se
5: fim função
```

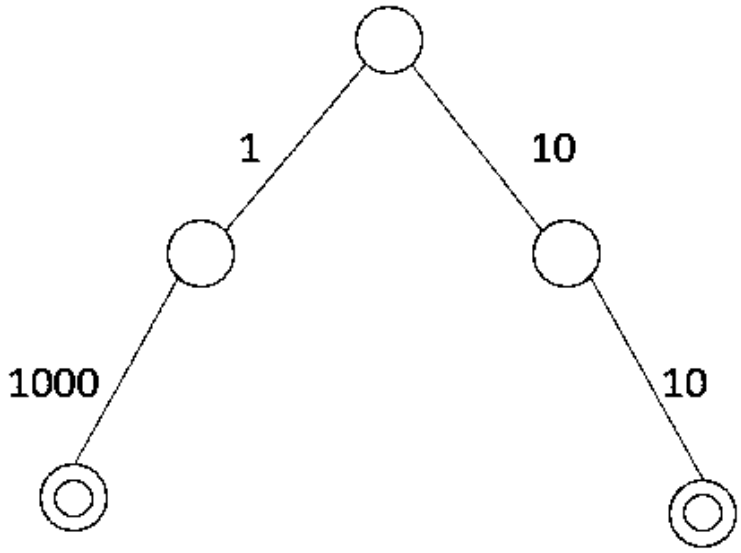
Fonte: Rocha e Dorini (2004, p. 5).

De acordo com Rocha e Dorini (2004), o algoritmo funciona da seguinte forma:

> A princípio o conjunto S está vazio, ou seja, não há candidatos escolhidos. Então, a cada passo, utiliza-se a função de seleção para determinar qual é o melhor candidato (lembrando que a função de seleção considera apenas os elementos que ainda não foram avaliados). Caso o conjunto ampliado de candidatos não seja viável, ignora-se o termo que está sendo avaliado no momento. Por outro lado, se tal conjunto é viável, adiciona-se o elemento em questão ao conjunto S. O elemento considerado, sendo ele aceito ou rejeitado, não é mais considerado pela função de seleção nos passos posteriores.
>
> Cada vez que o conjunto de candidatos escolhidos (S) é ampliado, é verificado se a solução do problema foi obtida. Quando um algoritmo guloso trabalha corretamente, a primeira soluçãoo encontrada da maneira aqui descrita é ótima (ROCHA; DORINI, 2004, p. 5).

Coppin (2010) afirma que a busca gulosa pode ser induzida a erro se o primeiro passo do caminho mais curto até o objetivo for maior que o primeiro passo por outro caminho conforme ilustra a Figura 37.

Figura 37 - Árvore de Busca onde a Busca Gulosa não achará a melhor solução



Fonte: Adaptado de Coppin (2010, p. 98)

Analisando a árvore, vê-se que ambos os caminhos levam ao objetivo, entretanto a Busca Gulosa seguirá o caminho com custo 1+1000 = 1001 em vez do caminho com custo 10+10 = 20.

AGENTES INTELIGENTES

Anteriormente, foram apresentados os **Agentes Racionais** como importantes ferramentas da IA, sem defini-los e sem acrescentar detalhes sobre o seu funcionamento. De acordo com Coppin (2010, p. 549), agente é "[...] uma entidade (geralmente uma entidade de *software*) cuja função é auxiliar humanos a realizar alguma tarefa ou resolver problemas". Já Russel e Norvig (2013, p. 34), definem agente como "[...] tudo o que pode ser considerado capaz de perceber seu **ambiente** por meio de **sensores** e de agir sobre esse ambiente por meio de **atuadores**". Para efeito dessa pesquisa, somente serão considerados os agentes de *software*.

O comportamento de um agente é baseado na percepção que ele tem do ambiente e no mapeamento dessas percepções associando-as a ações. Russel e Norvig (2013) definem que o comportamento de um agente é descrito pela **função do agente** que mapeia a sequência de percepções específica para cada ação. Assim, é feita uma tabulação das possibilidades de percepções e das ações a elas associadas. Deve-se observar que isso pode levar a uma tabela de tamanho infinito, o que pode ser evitado limitando as percepções e ações possíveis. Uma vez definida a tabela, a **função** do agente deve ser implementada pelo **programa do agente**. Russel e Norvig alertam para essa importante diferença: "[...] a **função do agente** é uma

descrição matemática abstrata, enquanto o **programa do agente** é uma implementação concreta, executada em um sistema físico" (RUSSEL; NORVIG, 2013, p. 35).

Os autores definem **Agente Racional** como aquele agente que faz tudo certo, ou seja, todas as entradas na tabela correspondente à função do agente são preenchidas de forma correta. Isso significa que o estado do ambiente, após a ação do agente, é o desejado.

Os agentes devem ter as seguintes propriedades, de acordo com Coppin (2010):

Inteligência: Agentes Inteligentes têm conhecimento adicional de domínio, habilitando-os a cumprir as suas tarefas mesmo quando parâmetros de tarefas são alterados ou surgem situações inesperadas. Coppin (2010) afirma que agentes inteligentes devem ser capazes de aprender a partir do próprio desempenho, de outros agentes, do usuário ou do ambiente onde estão.

Autonomia: é a capacidade de tomar decisões e de agir sem uma intervenção do usuário ou do programador. Essa propriedade pode dar a falsa impressão de total independência ao agente. A autonomia existe dentro de parâmetros previamente definidos e programados.

Capacidade de Aprender: diante de novas situações, o agente inteligente pode armazenar novas informações e passar a agir de acordo com elas. Um agente pode aprender a partir da forma que um usuário opera o sistema ou a partir da forma como outros agentes agem (vide próxima característica).

Cooperação: em sistemas multiagentes, é possível que um agente possa interagir com outros e operar de forma a um ajudar o outro. Mais uma vez, é preciso que os agentes sejam programados para isso.

Algumas das características mencionadas apontam para um importante recurso que pode ser implementado nos agentes: o aprendizado de máquina, descrito na próxima subseção.

APRENDIZADO DE MÁQUINA

Conforme já foi mencionado anteriormente, agentes inteligentes podem ser capazes de aprender comportamentos. O objetivo desta subseção é explicar como um computador (ou um agente) é capaz de aprender comportamentos.

INTRODUÇÃO

Desde os primórdios da IA, uma das estratégias de estudo era averiguar como o computador seria capaz de **pensar como um ser humano**. Essa estratégia já foi mencionada na subseção INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL. Dentro dessa estratégia,

busca-se usar o computador para executar tarefas associadas ao pensamento humano e, uma delas, certamente é o aprendizado (ou aprendizagem).

Uma importante área na IA é o desenvolvimento de agentes que sejam capazes de aprender a classificar dados a partir de dados previamente classificados, conforme diz Coppin (2010).

Russel e Norvig (2013, p. 693-694) explicam que "[...] qualquer componente de um agente pode ser melhorado através da aprendizagem a partir de dados". Os autores acrescentam que existem quatro fatores principais que influenciam o processo de construção de agentes capazes de aprender: a) que componente deve ser melhorado; b) o conhecimento prévio que o agente tem; c) que representação é usada para os dados e o componente; e d) que *feedback* está disponível para o aprendizado.

Em relação ao *feedback* recebido, o aprendizado pode ser classificado como: supervisionado, aprendizado não supervisionado e aprendizado por reforço, descritos nas próximas subseções,

APRENDIZADO SUPERVISIONADO

De acordo com Coppin, "Na maioria dos problemas de aprendizado, a tarefa é aprender a classificar entradas de acordo com um conjunto finito (ou, às vezes, infinito) de classificações" (COPPIN, 2010, p. 234). A partir dos dados previamente classificados, o agente faz um mapeamento relacionando os dados de entrada (os dados classificados) e a forma de classificá-los. Esse conjunto de dados de entrada é chamado de **conjunto de treinamento**.

Esse processo de treinamento utilizado no aprendizado supervisionado é bastante comum na anotação em *corpora* digitais, conforme demonstram Namiuti-Temponi e Costa (2014, p. 87), Paixão de Sousa (2014, p. 65) e Faria e Galves (2016, p. 302). Essa metodologia é utilizada para treinar *parsers*, anotadores e outros *softwares* que utilizam padrões de dados para executar as suas funções.

Quando o conjunto de treinamento tem poucos exemplos ou quando não é possível estabelecer uma relação exata entre o conjunto de treinamento e as ações a executar, chama-se esse processo de **aprendizagem parcialmente supervisionada** ou **aprendizagem semissupervisionada** conforme definem Russel e Norvig (2013).

APRENDIZADO NÃO SUPERVISIONADO

Segundo Russel e Norvig |(2013), no processo de aprendizado não supervisionado, o agente também aprende padrões a partir dos dados de entrada, embora não receba *feedback*

específico. Nesse caso, utiliza a estratégia de **agrupamento** que é a detecção de grupos de exemplos de entrada úteis.

Coppin (2010) afirma que o aprendizado não supervisionado ocorre sem intervenção humana, portanto sem receber dados de treinamento.

APRENDIZADO COM REFORÇO

De acordo com Coppin, "[...] um sistema que use aprendizado com reforço receberá um reforço positivo ao operar corretamente e um reforço negativo ao operar incorretamente" (COPPIN, 2010, p. 249).

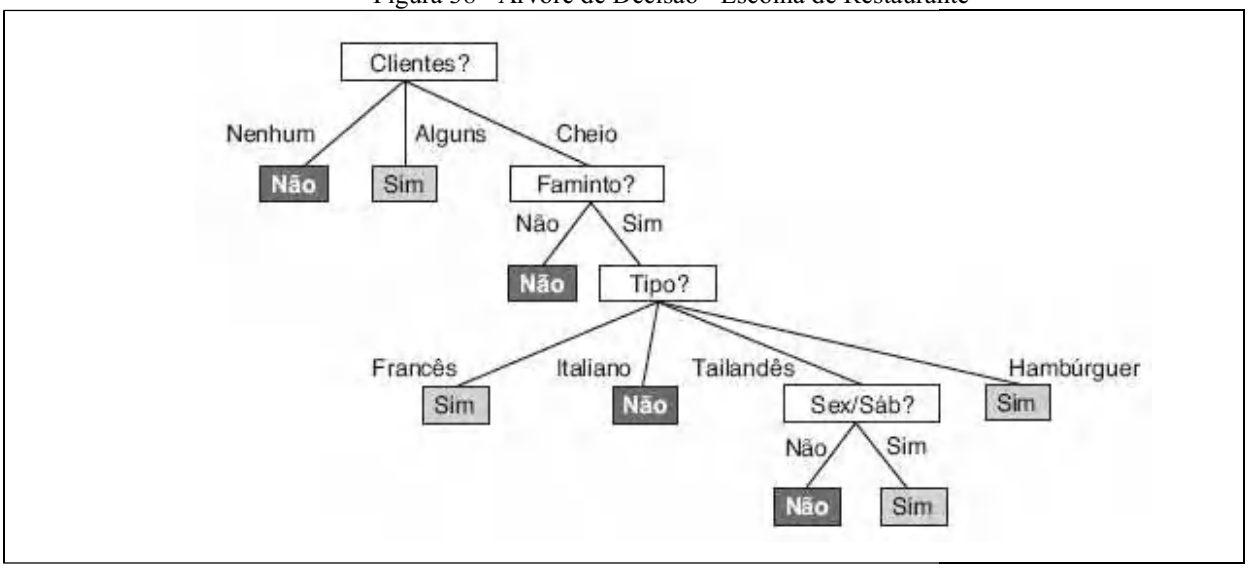
Um exemplo desse tipo de agente é um programa de jogo de xadrez que, após efetuar uma jogada, perca uma peça importante. Isso seria um reforço negativo que ensinaria o agente a evitar essa jogada em circunstância similar.

APRENDIZADO EM ÁRVORE DE DECISÃO

Também conhecido como aprendizado em Árvore de Decisão Indutiva, de acordo com Coppin (2010). Essa técnica também utiliza um conjunto de dados de treinamento que, ao serem lidos, farão com que o agente construa uma árvore onde cada nó é uma questão sobre alguma característica dos dados. A partir da resposta de cada questão (nó), podem ser seguidos caminhos diferentes que permitirão classificar os dados desconhecidos, caso os dados de treinamento sejam representativos.

A Figura 38 mostra uma árvore de decisão para a escolha de um restaurante. A árvore foi montada a partir de dados de treinamento.

Figura 38 - Árvore de Decisão - Escolha de Restaurante



Fonte: Russel e Norvig (2013, p. 699).

Se os dados de treinamento forem insuficientes, a árvore pode induzir a um comportamento incorreto. No exemplo dado, pode-se observar que, se o restaurante estiver

cheio e o cliente não estiver faminto, a árvore recomenda não aguardar. Isso pode ser um engano dos dados de entrada.

Segundo Coppin (2010), um dos mais famosos algoritmos de árvore é o ID3 que foi desenvolvido por Quinlan nos anos 1980. Para determinar que características usar em cada nó da árvore, esse algoritmo analisa a que oferece maior ganho de informação.

O PROBLEMA DA SUPERADAPTAÇÃO

Quando um método de aprendizado pode classificar corretamente **todos os dados de treinamento**, mas não tem bom desempenho com os dados reais, diz-se que ocorreu um problema de **superadaptação**.

Coppin afirma que "esse problema geralmente ocorre quando há ruído nos dados de treinamento ou quando os dados de treinamento não representam adequadamente o espaço completo de dados possíveis" (COPPIN, 2010, p. 245).

APLICAÇÕES DA INTELIGÊNCIA ARTIFICIAL NA LINGUÍSTICA

Conforme Coppin (2010) explica, a Linguística foi uma das primeiras áreas de estudo da Inteligência Artificial. O autor comenta que "[...] parte do otimismo em torno do Processamento de Linguagem Natural veio dos escritos de Noam Chomsky, que nos anos 50 propôs a teoria sobre estruturas sintáticas, que era uma teoria formal da estrutura da linguagem humana" (COPPIN, 2010, p. 11). Russel e Norvig (2013) acrescentam que a teoria de Chomsky (datada de 1957) era baseada em modelos sintáticos criados pelo linguista indiano Panini (350 a. C.) e era formal o bastante para poder ser programada. Assim, prosseguem os autores, "[...] a linguística moderna e a IA 'nasceram' praticamente na mesma época e cresceram juntas, cruzando-se em um campo híbrido chamado **linguística computacional** ou **processamento de linguagem natural**" (RUSSEL; NORVIG, 2013, p. 16).

A maior parte das técnicas de IA depende de representação de conhecimento, de forma que um computador possa encontrar soluções para os problemas propostos, o que torna a relação da IA com a Linguística ainda mais estreita.

Nas próximas subseções, são descritas as mais importantes aplicações da IA na Linguística.

PROCESSAMENTO DE LINGUAGEM NATURAL

Uma das áreas da IA de maior utilização na Linguística é o Processamento de Linguagem Natural (PLN).

Sobre a pesquisa na área de PLN, Dias da Silva (2006) registra:

> Uma peculiaridade desse amplo e controverso domínio de pesquisa é o fato de agregar uma heterogeneidade de objetivos: desde a meta de investigar meios de empregar o computador como uma simples ferramenta auxiliar para investigar material linguístico (por exemplo, a criação de programas de computador para calcular estatísticas de ocorrências de palavras em textos ou para identificar e indexar palavras e segmentos de texto) até a meta de criar uma inteligência artificial, nos moldes do supercomputador *HAL-9000* do clássico de Stanley Kubrick – 2001: Uma Odisseia no Espaço (DIAS-DA-DILVA, 2006, p. 104-105).

Coppin (2010) e Russel e Norvig (2013) comentam que um sistema computacional necessita, para lidar com linguagem natural, ser capaz de compreender a linguagem em diversos níveis como fonológico, morfológico, sintático, semântico e pragmático. Além disso, deveria ser capaz de ter algum tipo de conhecimento do mundo.

Segundo explicam Othero e Menuzzi (2005) os desenvolvimentos de PLN juntamente com a IA têm produzido estudos e avanços dentro de áreas como o reconhecimento de fala, síntese de fala e interação entre ser humano e máquinas através de diálogos em linguagem natural.

TRADUÇÃO POR MÁQUINA

Russel e Norvig (2013) mencionam que essa aplicação foi uma das maiores motivações nos primórdios da IA. Na época, pensava-se que seria possível, uma vez registrada a tradução de palavras nas diversas línguas, fazer traduções de textos instantaneamente. Logo se percebeu que traduzir textos envolve questões culturais, questões regionais e de contexto de utilização das palavras. Mesmo que o entusiasmo inicial tenha esmorecido, essa ainda é uma das áreas onde grandes companhias como a Microsoft e Google investem e aperfeiçoam seus programas tradutores, tornando-os cada vez melhores.

No final de 2016, a Microsoft anunciou o lançamento de um "tradutor universal"[18] que permitiria a conversação presencial entre até cem pessoas que, tendo instalado o *software* em seus aparelhos celulares, poderiam conversar entre si, utilizando as 60 línguas diferentes que o sistema suporta.

A Google oferece o seu tradutor, *Google Translator* que permite traduzir palavras, frases e textos completos entre diversas línguas. Uma importante evolução desse *software* é que ele atualmente permite que o usuário escolha entre opções de tradução e até discorde da tradução feita automaticamente. Nesse caso, o usuário pode sugerir uma tradução alternativa que será analisada pelos linguistas da Google e, eventualmente, incorporada ao sistema tradutor.

---

[18] Esse nome é uma alusão à série Jornada nas Estrelas, um dos ícones da ficção científica. https://www.tecmundo.com.br/microsoft/112716-microsoft-anuncia-primeiro-tradutor-universal-conversas-presenciais.htm. Acesso em 10 jan. 2017.

RECUPERAÇÃO DE INFORMAÇÃO

Coppin (2010) explica que, geralmente, a recuperação de informação envolve a busca de documentos em *corpus* relevantes à consulta de um usuário. Para permitir maior precisão no processo de recuperação é preciso reduzir ambiguidades através de mecanismos como: eliminação de palavras irrelevantes à busca e cálculo da frequência de termos, ambos detalhados a seguir:

Eliminação de palavras irrelevantes: os programas de busca geralmente possuem uma "lista de supressão" (*stop list*) que contém palavras que devem ser eliminadas de todas as consultas, por exemplo: qual, é, de etc. Dessa forma, antes de processar a consulta, o *software* descarta essas palavras e pesquisa as demais.

A **frequência de termos**, segundo explica Coppin (2010) é uma técnica bastante utilizada por mecanismos de busca e é conhecida como TF-IDF (do inglês *Term Frequency*[19] – *Inverse Document Frequency*). Assim, o TF-IDF é calculado para cada um dos termos de uma consulta e os valores resultantes são armazenados em um vetor que representaria um documento. A frequência inversa de documento (IDF) de uma palavra W é calculada da seguinte forma, segundo Coppin (2010, p. 518):

$$IDF\ (W) = \log\ \frac{|D|}{DF\ (W)}$$

Onde |D| é o número de documentos no *corpus*; DF(W) é a **frequência de documento** de W, que é o número de documentos no *corpus* que contêm a palavra W.

A frequência de termo da palavra W no documento D é escrita como TF (W, D) e representa o número de vezes que a palavra W ocorre no documento D.

O TF-IDF (D, W_i) = TF (W_i, D) x IDF (W_i), onde W_i é cada uma das palavras da consulta.

Ao utilizar essa técnica, sempre que uma consulta for feita, seria calculado o TF-IDF e os documentos seriam ranqueados de acordo com a maior frequência de ocorrência das palavras buscadas, otimizando a recuperação de informação.

[19] *Term Frequency* – Frequência de Termo; *Inverse Document Frequency* – Frequência Inversa do Documento.

APÊNDICE B – Noções sobre Software Livre

Segundo Brasil (1998), programa de computador é a expressão de um conjunto organizado de instruções em linguagem natural ou codificado, contido em suporte físico de qualquer natureza, de emprego necessário em máquinas automáticas de tratamento da informação, dispositivos, instrumentos ou equipamentos periféricos, baseados em técnica digital ou análoga, para fazê-los funcionar de modo e para fins determinados.

Quando um usuário deseja utilizar um *software* no seu computador, precisa adquirir a licença de uso, conforme Brasil (1998) em seu artigo 9. A violação dos direitos autorais, ou seja, a cópia não autorizada ou a utilização sem licença implica em detenção de quatro meses a dois anos e a pagamento de multa que pode atingir o valor de até 3000 vezes o valor de uma licença original, tudo isso **por cópia não autorizada**.

Segundo Taurion (2004), os *softwares* se dividem em duas categorias em relação à licença de uso: *software* proprietário e *software* livre. Os primeiros são programas de computador com código-fonte fechado, patenteado por uma única empresa, que cobra direito de propriedade intelectual. Em oposição, segundo Silveira (2004), *software* livre é aquele que obedece às quatro liberdades:

> Liberdade n.0: A liberdade de executar o programa para qualquer propósito;
> Liberdade n.1: A liberdade de estudar como o programa funciona, e adaptá-lo para as suas necessidades. Acesso ao código-fonte é um pré-requisito para esta liberdade;
> Liberdade n.2: A liberdade de redistribuir cópias de modo que você possa beneficiar o próximo;
> Liberdade n.3: A liberdade de aperfeiçoar o programa, e liberar os seus aperfeiçoamentos, de modo que toda a comunidade se beneficie. Acesso ao código-fonte é um pré-requisito para esta liberdade

Portanto, "a licença do Software livre é uma licença não-proprietária de uso" (SILVEIRA, 2004, p. 11).

Existe uma confusão em relação ao termo software livre. O termo original em Inglês (free software) leva a confundir liberdade com gratuidade. Isso porque a palavra *free* pode significar "livre", mas também pode significar "grátis", o que não é o caso. Software livre pode ser pago, embora raramente os autores cobrem por eles, o que aumenta a confusão a respeito do nome. Por esse motivo, modernamente, tende-se a utilizar o termo *Open Software* (Programa aberto), que tende a gerar menos confusão.

VANTAGENS NA UTILIZAÇÃO DE SOFTWARE LIVRE

Segundo Hexsel (2002), a utilização de Software livre apresenta as seguintes vantagens:

- O custo social é baixo: os benefícios do desenvolvimento são voltados para os usuários e não para os fabricantes, como acontece com os softwares proprietários.
- Não se fica refém de tecnologia proprietária: evita o risco de fazer um upgrade obrigatório quando o fornecedor resolver descontinuar uma versão antiga de um produto.
- Independência de fornecedor único: mesmo que uma das empresas envolvidas no desenvolvimento de um software livre venha a sair do negócio, outras podem assumir. O SL é comunitário.
- Custo inicial próximo de zero: a maioria dos SL é grátis ou custa pouco.
- Não-obsolescência do hardware: geralmente, quando um software proprietário é atualizado, aumenta uma série de funcionalidades, muitas vezes inúteis para a maioria dos usuários. Esse aumento de funcionalidades, pode implicar na necessidade de atualização do hardware para suportar a nova versão. Geralmente isso não ocorre com o SL.
- Robustez e segurança: os SL são reconhecidamente robustos, principalmente pela grande quantidade de pessoas envolvidas no processo de identificar e corrigir eventuais defeitos.
- Possibilidade de adaptar os aplicativos: como são fornecidos com os códigos-fontes, os SL têm essa como uma das suas fortes características.
- Suporte: de forma contrária à que se imagina, o suporte do SL é rápido por envolver milhares de pessoas no mundo inteiro. Algumas empresas oferecem suporte por 24h, como a Conectiva e a IBM.

A descrição do *software* livre tem grande aderência aos objetivos explicitados no Manifesto das Humanidades Digitais, como se pode observar principalmente pelo aspecto do desenvolvedor e do cliente não ficarem reféns de tecnologias proprietárias o que poderia inviabilizar a continuidade dos trabalhos de pesquisa.

DESVANTAGENS NA UTILIZAÇÃO DE SOFTWARE LIVRE

Segundo Hexsel (2002), a utilização de *Software* Livre pode trazer algumas desvantagens:

- Interface de usuário não é uniforme: de fato não existe um ambiente integrado com interface padronizada o que pode dificultar a aprendizagem por parte dos usuários novatos.

- Instalação e configuração podem ser difíceis: a característica do Linux de permitir configurações complexas pode dificultar a sua instalação. Atualmente a comunidade tem trabalhado e tem conseguido criar instalações práticas e automáticas para usuários leigos. A distribuição UBUNTU, por exemplo, atualmente é instalada de forma quase que automática.
- Mão-de-obra escassa: essa dificuldade tem diminuído, à medida que as Universidades e Faculdades têm formado profissionais habilitados em SL. A linguagem de programação Java, por exemplo, é utilizada em muitos cursos

De fato, pode-se observar que essas desvantagens têm diminuído ao longo do tempo, o que torna o uso do *software livre* mais atrativa e cada vez mais confiável.