Guilherme Carvalho Januário
Leonardo Alexandre Ferreira Leite
Marcelo Li Koga

POLI-LIBRAS
UM TRADUTOR DE PORTUGUÊS PARA LIBRAS

São Paulo
2010

Guilherme Carvalho Januário
Leonardo Alexandre Ferreira Leite
Marcelo Li Koga

POLI-LIBRAS
UM TRADUTOR DE PORTUGUÊS PARA LIBRAS

Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia.

Área de Concentração:
Engenharia de Computação

Orientador: João José Neto

São Paulo
2010

AGRADECIMENTOS

Agradecemos ao nosso orientador, João José Neto, por seu apoio neste trabalho e por seu exemplo de dedicação ao ensino enquanto professor.

Agradecemos a estudante de design Heloisa Yoshioka pela cooperação técnica no Virtual Jonah.

Agradecemos também aos que se envolveram neste projeto, prestando-nos esclarecimentos sobre a língua de sinais e motivando-nos nesse empreendimento, a saber: Larissa, Dora, Carol, Ricardo e professora Maria Cristina Pereira.

Agradecemos ao projeto Poli Cidadã pela disponibilização do servidor para a hospedagem de nossas aplicações.

"Se vi mais longe foi por estar de pé sobre ombros de gigantes."

Isaac Newton

Resumo

Neste trabalho pretende-se criar uma ferramenta de tradução automatizada de frases em português para uma sequência de sinais em LIBRAS, a Língua Brasileira de Sinais, utilizada pelos surdos no Brasil. A saída gerada deve ser representada graficamente por um avatar 3D e o processo de tradução deve levar em conta aspectos sintáticos das línguas envolvidas, evitando a geração do chamado *português sinalizado* (correspondência direta um-pra-um entre palavras em português e sinais), o que seria de pouco interesse aos surdos. Como diferencial de outros projetos relacionados à tradução de LIBRAS, pretendemos produzir um sistema aberto (*open source*) que permita à comunidade acompanhar seu desenvolvimento e também a outros interessados em desenvolver novas aplicações úteis à comunidade surda utilizarem módulos de nossos sistema, sendo que para este último aspecto pretende-se gerar um sistema modular, conferindo maior reusabilidade ao software.

Palavras-chave: Tradução automática; Língua Brasileira de Sinais; Engenharia de Computação; Linguagens naturais; Linguagens formais;

Abstract

In this work we intend to build a Portuguese to LIBRAS automatic translation tool, where LIBRAS is the Brazilian Sign Language, used by Brazilian deaf people. The generated output must be signalized by a 3D graphical avatar and the translation process must consider syntactical aspects of the languages, avoiding the *signalized portuguese* (one sign per Portuguese word), which wouldn't be interesting to deaf people. An important difference between our work and other related works is that our intention is to produce an *open source* system, that is going to be available for the community to use and to collaborate with the system progress, including building new pro-deaf systems based in our modules, as we have projected a modular system aiming to produce reusable code.

Keywords: Machine translation; Sign language; Computation; Natural language; Formal languages

Lista de Figuras

Sumário

Capítulo 1

INTRODUÇÃO

A Língua Brasileira de Sinais – LIBRAS – é a língua utilizada pelos surdos brasileiros e é uma língua de sinais que, como as usadas em diferentes países, apresenta regras que respondem pela formação dos sinais e pela organização destes nas estruturas frasais e no discurso.

Diferentemente das línguas orais, os articuladores primários das línguas de sinais são as mãos, que se movimentam no espaço em frente ao corpo e articulam sinais em determinadas localizações nesse espaço (SECRETARIA MUNICIPAL DE EDUCAÇÃO, 2008).

Pesquisas sobre as línguas de sinais vêm mostrando que essas línguas são comparáveis em complexidade e expressividade a quaisquer línguas orais. Essas línguas expressam ideias sutis, complexas e abstratas. Os seus usuários podem discutir filosofia, literatura ou política, além de esportes, trabalho, moda e utilizá-las com função estética para fazer poesias, estórias, teatro e humor (FELIPE, 1997).

Com isso temos que, para os surdos, a LIBRAS é sua língua materna e o português (escrito) a língua secundária, sendo que se percebe uma dificuldade de grande parte da comunidade surda no aprendizado do português. Essa dificuldade deve-se, dentre outros, à falta de sentido do raciocínio com base em fonemas como nós ouvintes fazemos (associando o fonema ao grafema; associação esta totalmente arbitrária ao surdo sinalizador).

Esse tipo de dificuldade é a motivação para a criação de ferramentas que auxiliem os surdos no aprendizado e na compreensão de textos escritos em português.

Além disso, há também a necessidade de que ouvintes aprendam a língua de sinais, principalmente educadores que devem garantir o cumprimento do Decreto Federal 5626, de 22 de dezembro de 2005, que estabelece que os alunos surdos devem ter uma educação bilíngue, na qual a Língua Brasileira de Sinais é a primeira e a Língua Portuguesa, na modalidade escrita, a segunda.

1.1 Objetivo

O objetivo primordial do projeto é a produção de um tradutor capaz de analisar um texto em português e gerar uma saída gráfica em que um avatar realiza os sinais correspondentes em LIBRAS. A esse tradutor demos o nome de "Tradutor Poli-Libras".

O tradutor deve não apenas fazer a tradução palavra por palavra, mas fazer as considerações de sintaxe e de contexto para que a saída se aproxime o máximo possível da expressão natural em LIBRAS.

Além da função básica e mais direta, que seria dar suporte ao surdo para compreender um texto em português, este tradutor pode ser usado em alguns contextos ou outros produtos, tais como:

- Ferramenta de aprendizado pra ouvinte aprender LIBRAS;
- Ferramenta de aprendizado pra surdo aprender português;
- Gerador de vídeo com LIBRAS pra poder aumentar a acessibilidade de *websites* ou programas de televisão.

O resultado da tradução, que seria correspondente ao "texto em LIBRAS" deve também poder ser armazenado, de forma a se criar uma forma de codificação digital da LIBRAS.

1.2 Motivação

Conforme explicado, os surdos no Brasil normalmente se comunicam por LIBRAS, sendo esta a língua materna dessas pessoas, enquanto o Português é como uma segunda língua.

Dada a parcela significativa dessa população que possui dificuldade em ler e compreender os textos em português, seria-lhes desejável obter acesso a esses conteúdos codificados em LIBRAS. Dessa forma, o presente projeto visa apresentar uma alternativa para este problema.

No entanto, não só a tradução instantânea é desejada, mas também se espera que o resultado da tradução, codificado em LIBRAS, seja um produto que possa ser manipulado

por outras ferramentas voltadas ao público surdo, de forma a possibilitar a criação de um legado digital codificado em LIBRAS.

Na academia brasileira existem outros projetos com objetivos semelhantes, no entanto, em geral, esses não se encontram disponíveis para o público geral, exceto pelas publicações de artigos acadêmicos. Portanto, outro objetivo do projeto é a publicização dos resultados, inclusive das ferramentas produzidas, de modo que sejam públicos para o uso por todos.

A disponibilização para o público em geral inclui não só os usuários finais, mas eventuais desenvolvedores que trabalhem em projetos relacionados, sendo que para se atender a esse público outra meta do projeto é desenvolver o sistema de uma forma modularizada, de forma que determinados módulos possam ser reaproveitados de forma independente.

Para que terceiros possam aproveitar esses módulos na construção de novos sistemas é também fundamental a liberação de nosso sistema sobre uma licença livre que assim o permita. Isso ocorre porque nas situação em que algo está disponível e nada define o que pode ser feito, vale a regra do nada é permitido até que se receba permissão específica.

Nesse contexto surgem as licenças utilizadas em *softwares livres*, também denominados sistemas *open source* (código aberto), que garantem ao usuário o acesso ao código-fonte do sistema, o que lhes permite fazer alterações para adaptar os *softwares* a suas necessidades, colaborar com seu desenvolvimento ou ainda gerar um novo produto com base no original.

Dessa forma, para incentivar e facilitar a produção de novas ferramentas que atendam a necessidades da comunidade surda, nossos sistemas serão publicados sobre licenças livres.

1.3 Trabalhos Relacionados

Nessa seção discorre-se sobre outros trabalhos que apresentam semelhanças com o projeto proposto, evidenciando-se as diferenças, vantagens e desvantagens de cada.

Dicionário Acessa Brasil

Um trabalho relacionado bastante utilizado na comunidade surda é o dicionário de LIBRAS do Acessa Brasil (LIRA; SOUZA, 2008), que consiste numa ferramenta *online* de

consulta de sinais a partir de palavras portuguesas. O dicionário é de fato bem completo, porém a nossa proposta apresenta diversas diferenças. Primeiro, o fato de traduzirmos frases completas e não apenas palavras. Além disso, nosso vocabulário é espansível e escalável por apresentar saída gerada computacionalmente, em contraposição ao formato em vídeo do dicionário, e nosso módulo de dicionário apresenta também os sinais formalizados num modelo baseado em XML, de modo que possa ser reutilizado por outras aplicações.

Falibras

Falibras é um projeto feito pelo Instituto de Computação da Universidade Federal de Alagoas (CORADINE et al., 2007) cujo propósito é bem semelhante ao nosso: fazer um tradutor de português para LIBRAS, com saída animada. O projeto Falibras passou por várias evoluções, começando por uma tradução puramente léxica e de pequenas frases, depois implementando análise sintática e redução de ambiguidades, em seguida modularizando seus componentes. Existe também uma abordagem alternativa com a tradução baseada em memória. O problema com o Falibras é que, apesar de ser muito fácil encontrar diversos artigos relacionados, não se encontra nada disponível ao usuário final, nem tampouco a desenvolvedores. Não existem ferramentas que de fato auxiliem a comunidade surda em geral, e esse é o diferencial do nosso projeto, apresentar ferramentas *open-source* para a comunidade.

TLibras

Um outro tradutor Português–LIBRAS que encontramos é o TLibras (LIRA, 2003), projeto coordenado pela OSCIP Acessibilidade Brasil e que conta com 3 equipes distintas: uma de LIBRAS, do FENEIS (Federação Nacional de Educação e Integração dos Surdos), uma do NILC-USP (Núcleo Interinstitucional de Lingüística Computacional) para Linguagens Naturais e uma de computação gráfica, da própria Acessibilidade Brasil. Este projeto conta com mais de 20 pessoas, o que demonstra a complexidade proposta do presente trabalho. Eles planejaram 3 etapas, começando com textos em português, adicionando reconhecimento de voz na segunda, e na terceira implementando na TV digital.

O processo de tradução do TLibras utiliza-se da UNL (Universal Networking Language) (UCHIDA; ZHU, 2001) para fazer a ponte entre as línguas. A UNL é uma proposta de língua intermediária para traduções, uma interlíngua (topo da pirâmide de Vauquois citada

na seção de Aspectos Conceituais) para qual todas as línguas devem traduzir e da qual todas podem ser traduzidas. Ou seja, para utilizarem a UNL, a equipe do projeto TLibras teve que fazer basicamente um mapeamento de diversos textos em LIBRAS para a UNL, incluindo informações morfológicas, sintáticas e semânticas, e então a ponte estaria feita pois a tradução português-UNL já existe.

Um módulo bastante interessante do TLibras é o que propõe principalmente a criação de um avatar 3D que fala LIBRAS, que poderia ser usado em diversas aplicações através de uma entrada na *Notação-Libras* (que não foi especificada no artigo lido). No entanto, esse projeto usa tecnologias não facilmente compatíveis com a Web, e como nosso intuito é ter um tradutor acessível via internet, a utilização desse avatar ficaria inviável (sempre seriam gerados vídeos, o que consumiria muitos recursos).

Entretanto, uma dificuldade que tivemos foi o fato de não encontrar nada disponível ao público desse projeto, ainda que seu lançamento estivesse previsto para o ano de 2004. Tudo que encontramos foi uma história da turma da Mônica traduzida, que em (LIRA, 2003) era referenciada que como restrição da etapa inicial: o único texto que iria ser transcrito para UNL era uma pequena história em quadrinhos com 5 frases.

Rybená

O player Rybená (FERNEDA; COSTA; ALMEIDA, 2003) é uma ferramenta web que promete traduzir textos de páginas na internet para LIBRAS, através do seu avatar animado. Um destaque desse trabalho é a possível integração com celulares para o envio de torpedos, ainda que somente 4 modelos de aparelhos são compatíveis atualmente [1]. Esse tradutor foi um dos únicos que conseguimos de fato testar, pois existe uma versão funcional em seu site. Entretanto, identificamos que o seu uso não é muito intuitivo e a tradução foi realizada palavra por palavra, gerando o chamado *português sinalizado* o que sintaticamente difere muito da LIBRAS e que, portanto, os surdos costumam não aceitar. Sem contar que o Rybená é um produto comercial, divergindo da nossa filosofia livre.

[1] Segundo o próprio site do Rybená. Disponível em: `http://www.rybena.com.br/produtos/devices_list.jsp?ckRybena=marcado`. Acesso em 05/12/2010

1.4 Organização

O presente documento está dividido nas seguintes seções:

- Capítulo 2 - Aspectos Conceituais: são definidos os conceitos básicos a serem discutidos ao longo de todo o documento, como as características da LIBRAS, conceitos de linguagens formais e métodos de tradução;
- Capítulo 3 - Tecnologias Avaliadas: estudos sobre APIs e ferramentas já existentes que foram avaliadas e utilizadas no projeto;
- Capítulo 4 - Especificação do Projeto: nesta seção são definidos os requisitos do projeto, casos de uso e a explicação sobre algumas decisões. Em especial define-se também a arquitetura do sistema com sua divisões em módulos, especificando a funções de cada um deles, incluindo aqui a definição de modelo de dados que codifica os sinais;
- Capítulo 5 - Implementação: este capítulo descreve a metodologia de trabalho e detalhes técnicos sobre a implementação dos módulos especificados na arquitetura;
- Capítulo 6 - Testes e Avaliação: este capítulo descreve os principais testes executados para a validação das funcionalidades do tradutor;
- Capítulo 7 - Conclusões: este capítulo avalia os resultados do projeto, além de ressaltar os projetos futuros que poderão se basear no presente projeto;
- Apêndice A - Gramática livre-de-contexto para modelar a língua portuguesa;
- Apêndice B - Organização do CD em anexo.

Capítulo 2

ASPECTOS CONCEITUAIS

2.1 LIBRAS

A Língua Brasileira de Sinais é a língua utilizada pelos surdos para se comunicar no Brasil. As línguas de sinais não diferem muito das línguas faladas, sendo tão complexas e expressivas quanto elas. A LIBRAS está presente em todos os níveis de análise das outras línguas, possuindo também sua própria gramática, sua própria sintaxe e seu próprio vocabulário. A principal diferença em relação às línguas faladas é o fato da LIBRAS ser uma língua visual-espacial, que é feita através de gestos e expressões com mãos, cabeça e corpo, percebidos pela visão. É portanto diferente do português por exemplo, que é uma língua oral-auditiva pois usa sons percebidos pela audição.

2.1.1 A morfologia do sinal

A formação de um sinal é definida basicamente por cinco parâmetros, quatro que se referem às mãos: configuração, localização (ou ponto de articulação), movimento e a orientação das palmas, e um quinto que se refere ao uso dos recursos não-manuais, que incluem expressões faciais, movimentos da boca, direção do olhar, o que permite a expressão de um número significativamente maior de informações linguísticas.

A configuração das mãos refere-se às formas das mãos, que podem ser da datilologia (alfabeto digital) ou outras formas feitas pela mão dominante (mão direita para os destros), ou pelas duas mãos. Já a localização é o lugar, no corpo ou no espaço, em que o sinal é articulado, podendo a mão tocar alguma parte do corpo ou estar em um espaço neutro. O movimento envolve tanto os movimentos internos da mão, quanto do pulso, movimentos direcionais no espaço, e o conjunto de movimentos no mesmo sinal. A orientação das palmas das mãos é a direção para a qual a palma da mão aponta na produção do sinal. Pode ser para cima, para baixo, para o corpo, para frente, para a esquerda ou para a direita. Por fim, os traços não-manuais envolvem expressão facial, movimento corporal e olhar. Sinais como BONITO, BONITINHO e BONITÃO, por exemplo, são representados

pela mesma configuração, posição, movimento e orientação da mão, no entanto possuem expressões faciais diferentes.

2.1.2 Convenções da Forma escrita

Nesta seção são descritas as convenções que adotamos ao longo do trabalho quanto à representação de LIBRAS na forma escrita.

1. Sinais em LIBRAS são representados por letras maiúsculas em português. Exs: CASA, BOLA.
2. Duas ou mais palavras separadas por hífen representam um sinal único. Exs.: CORTAR-COM-FACA, COMER-MAÇÃ
3. Quando uma palavra é representada por Datilogia (alfabeto manual), ela aparece separada letra por letra. Exs.: M-A-R-C-E-L-O
4. Concordância de número/pessoa ou de lugar é feita com elementos subscritos aos sinais. Ex: JOÃOa MARIAb aDARb PRESENTE (o verbo dar concorda com os pontos a e b, que se referem a João e Maria, respectivamente).
5. Expressões faciais/corporais e advérbios de intensidade são indicados sobrescritos ao sinal.
   Exs: $ANDAR^{rapidamente}$, $SIM^{balanca-cabeca}$
6. Por geralmente não haver uma desinência para diferenciar entre masculino e feminino e singular ou plural nos sinais, eles serão representados com um @ no fim. Portanto, AMIG@ significa "amigo", "amiga", "amigos" ou "amigas".

2.1.3 SignWriting

A notação para escrita em LIBRAS descrita na seção 2.1.2 é a alternativa normalmente utilizada pelos diversos artigos e estudos nessa área, uma vez que não existe uma forma oficial de representação gráfica. No entanto, como é evidente, esta notação traz pouca ou nenhuma informação sobre os aspectos morfológicos dos sinais utilizados, isto é, a partir do lexema CASA é impossível imaginar como seria o correspondente sinal; em contraposição, as palavras em português são formadas por uma sequência de sílabas que

representam unidades sonoras bem conhecidas, que permitem ao leitor "sintetizar" o som da palavra, mesmo sem nunca tê-la visto antes.

Mas existe também uma notação que possibilita a escrita de línguas de sinais de forma a explicitar os aspectos morfológicos do sinal, sendo denominada SignWriting , e que pode ser utilizada para escrever sinais de qualquer língua de sinais, a exemplo do nosso alfabeto romano que também permite a escrita de várias línguas diferentes.

Este sistema foi inventado em 1974 pela americana Valerie Sutton, uma dançarina que havia dois anos antes desenvolvido a DanceWriting.

Com um símbolo de SignWriting é possível especificar características do sinal como posição, forma da mão, articulação dos dedos, expressões facial e corporal etc. Um exemplo de texto em SignWriting pode ser observado na figura 2.1.

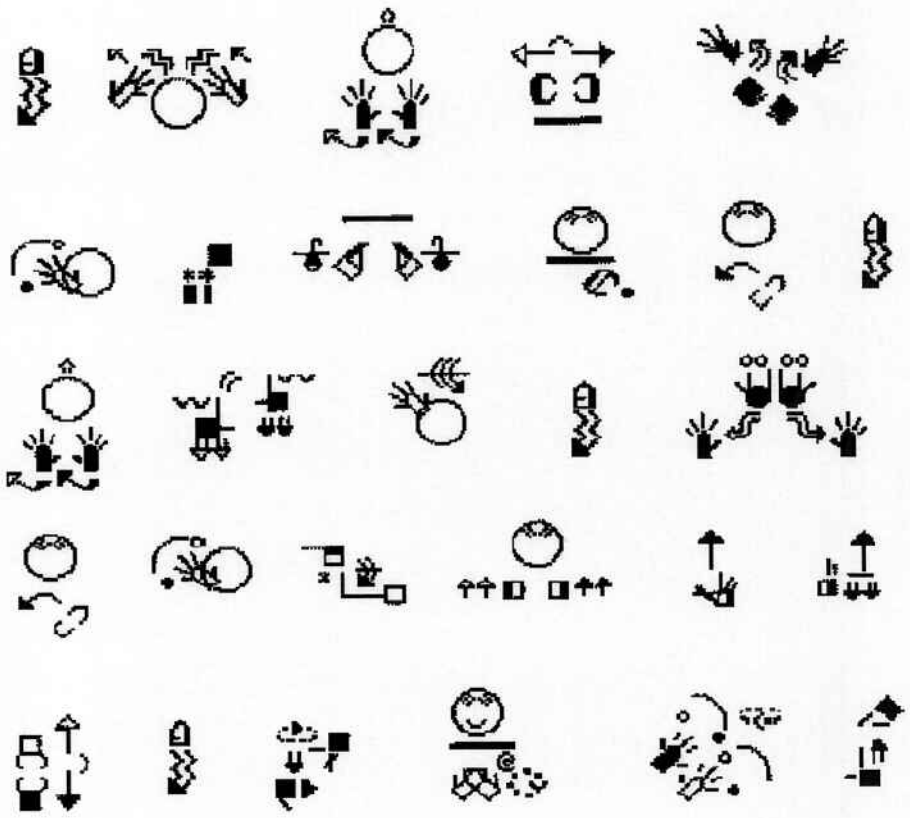

Figura 2.1: Exemplo de texto em SignWriting

O sistema começou a receber atenção no Brasil desde 1996, tendo como uma das obras mais relevantes sobre o assunto Dicionario Enciclopedico Ilustrado Trilingue (CAPOVILLA; RAPHAEL, 2001), cujo autor é o psicólogo responsável pela adaptação do SignWriting para a LIBRAS; outra importante obra é o Manual de SignWriting (SUTTON, 2003),

que é uma adaptação da obra originalmente americana, escrita pela própria Valerie Sutton, sendo focada em mostrar o SignWriting no contexto da LIBRAS.

Uma das principais vantagens do SignWriting é permitir aos surdos que escrevam em sua própria língua, sem que seja necessário recorrer a escrita fonética da língua oral para simbolizar os sinais. No entanto atualmente a aceitação da notação pela comunidade surda ainda é baixa, sendo um dos principais motivos considerados sua complexidade que a torna de difícil aprendizado.

Outro fato observado pelo grupo é que na prática vários sinais da LIBRAS são difíceis de se escrever em SignWriting pela dificuldade de se determinar os valores dos atributos que formam o símbolo do SignWriting , e isto, cremos, pode também ser um motivo para a não receptividade ao SignWriting .

Se junta ainda o fato de que, ao contrário da LIBRAS, o SignWriting é uma criação artifical, feita por ouvintes como uma solução a ser assimilada pelos surdos (muito embora se admita que haja um certo envolvimento para um aprimoramento do SignWriting neste sentido).

2.1.4 Classes gramaticais

As classes ou categorias gramaticais correspondem a paradigmas sobre um corpo de palavras. Através desses paradigmas se obtêm novos lexemas a partir de outros. Há também paradigmas relacionando elementos de classes gramaticais distintas. A exemplo, cita-se a concordância de número entre substantivos e verbos.

O conjunto das classes gramaticais da LIBRAS é bem mapeado no do português, havendo em cada poucas categorias não presentes no outro. Seguem observações sobre as categorias da LIBRAS (FELIPE, 1997):

- Verbos

  Basicamente, há dois tipos de verbos: os que não fazem concordância e os que fazem (chamados também de *plain* e *non-plain*, respectivamente). Os que não concordam estão sempre no infinitivo, sendo mais simples.

  Ex.: EU TRABALHAR ESCRITÓRIO.

  Já quanto aos que concordam, podem fazê-lo de três maneiras:

  1. Concordância número-pessoal:

A orientação marca as pessoas do discurso. O ponto inicial concorda com o sujeito e o final, com o objeto.

Ex.: 1sPERGUNTAR2s = "eu pergunto a você" é diferente de 2sPERGUNTAR1s = "você me pergunta"

2. Concordância de gênero (pessoa, animal, coisa): configuração de mão se altera;

   Ex.: quanto aos sinais do verbo andar, "pessoa anda" ≠ "carro anda" ≠ "animal anda"

3. Concordância com a localização: ocorre com verbos designando ações que começam ou terminam em determinado lugar. A explicação fica mais simples através de exemplo: para se indicar que algo foi colocado sobre a mesa deve-se, antes de se sinalizar o verbo 'colocar', representar a mesa em certo ponto de articulação; o sinal do verbo deve, então, ser realizado tendo como destino o ponto de articulação do sinal de mesa.

- Classificadores

  Os classificadores são configurações de mão que, substituindo o nome que as precedem, podem vir junto ao verbo para classificar o sujeito ou o objeto que está ligado à ação do verbo. Portanto os classificadores na LIBRAS são marcadores de concordância de gênero: PESSOA, ANIMAL, COISA. Os classificadores para PESSOA e ANIMAL podem ter plural, que é marcado ao se representar duas pessoas ou animais simultaneamente com as duas mãos ou fazendo um movimento repetido em relação ao número.

- Artigo

  Se em português não há correspondente direto dos classificadores da LIBRAS, nesta língua não há o correspondente ao artigo.

- Pronomes

  1. Pronomes pessoais

     Essa subcategoria de pronomes em LIBRAS não possui apenas as classificações singular e plural, mas indicam até certa quantidade mais precisamente o número de pessoas do discurso. Por exemplo, existem sinais para: EU, NÓS 2, NÓS 3, NÓS 4, NÓS-GRUPO e NÓS-TODOS. O mesmo vale para as segunda (VOCÊ) e terceira (ELE) pessoas.

Lista completa:

(a) primeira pessoa (singular, dual, trial, quatrial e plural): EU; NÓS-2, NóS-3, NÓS-4, NÓS-GRUPO, NÓS-TOD@;

(b) segunda pessoa (singular, dual, trial, quatrial e plural): VOCÊ, VOCÊ-2, VOCÊ-3, VOCÊ-4, VOCÊ-GRUPO, VOCÊ-TOD@;

(c) terceira pessoa (singular, dual, trial, quatrial e plural): EL@, EL@-2, EL@-3, EL@-4, EL@-GRUPO, EL@-TOD@

2. Pronomes demonstrativos / advérbios de lugar

   Os pronomes demonstrativos e os advérbios de lugar possuem basicamente o mesmo sinal em LIBRAS. EST@ / AQUI, ESS@ / AÍ e AQUELE / LÁ são feitos apontando-se para os locais apropriados acompanhados de um olhar para o mesmo.

3. Pronomes interrogativos

   Os pronomes QUE, QUEM, POR-QUE são utilizados geralmente no início da frase, enquanto QUAL, COMO, PARA-QUE, ONDE e QUEM (no sentido de "quem é?") são usados no final.

   Sempre simultaneamente aos pronomes ou expressões interrogativas há uma expressão facial indicando que a frase está na forma interrogativa.

- Adjetivos

  Os adjetivos geralmente vêm após o substantivo que qualificam e representam a característica de forma icônica.

- Advérbios

  Os advérbios de intensidade ou de modo não possuem sinais próprios e são indicados modificando a velocidade ou repetindo várias vezes o sinal a que se referem. Como não há marca de tempo nos verbos, os advérbios HOJE, PASSADO (ou ONTEM, ANTEONTEM) e FUTURO (AMANHÃ) são geralmente usados no início da frase para darem essa idéia.

2.1.5 Estrutura sintática na LIBRAS

Segundo Quadros (1999) a ordem base presente na LIBRAS é a SVO (Sujeito–Verbo–Objeto). Ordem base pois justifica que qualquer estrutura poderia ser entendida nessa

ordem e que as demais ordens podem ser derivadas desta e não o contrário. Isso não significa que a ordem mais comum seja SVO. Existem muitas situações para as quais a melhor ordem é OSV (Objeto–Sujeito–Verbo), na qual ocorre o fenômeno de topicalização, que consiste em se evidenciar o assunto primeiro, para contextualização, ou ainda SOV (Sujeito–Objeto–Verbo).

Resumidamente, em LIBRAS permite-se SVO, OSV e SOV sendo estes dois últimos com restrições. Essas restrições são, de modo geral, devidas à exigência de concordância entre verbo e sujeito/objeto e à presença de marcas não-manuais.

SOV é preferível em frases onde figura certa "iconicidade", como em MULHER TORTA COLOCAR-NO-FORNO. Ou seja, primeiro mostra-se a torta e depois coloca-se a no forno. Essa ordem é útil para que primeiro se posicionem os objetos no espaço e depois se sinalize a ação, ligando-os.

Ex.: JOAOa MARIAb aDARb LIVRO.

OSV é muito utilizado também (muitos apontam como a ordem mais comum, como BRITO (2006)), pois representa a ordem tópico-comentário, permitindo primeiro que se explique o contexto ao seu interlocutor para depois se explicitar a ação ocorrida. É bastante comum, desde que não haja restrições ao seu uso.

2.1.6 Graduação de Complexidade da LIBRAS

Considerando todas essas características e dificuldades apresentadas, montamos o que seria uma definição da LIBRAS em diversos níveis de complexidade, onde cada nível acrescenta uma camada extra de complexidade à língua. Essa escala foi dividida em 2: uma analisa somente o sinal e seus aspectos internos (nível léxico) e outra analisa a estrutura da frase como um todo (nível sintático). O objetivo dessa escala é também servir de guia para uma implementação iterativa do trabalho, aumentando a complexidade a cada nível.

A figura 2.2 mostra a graduação no nível da palavra e a figura 2.3, ao nível da frase.

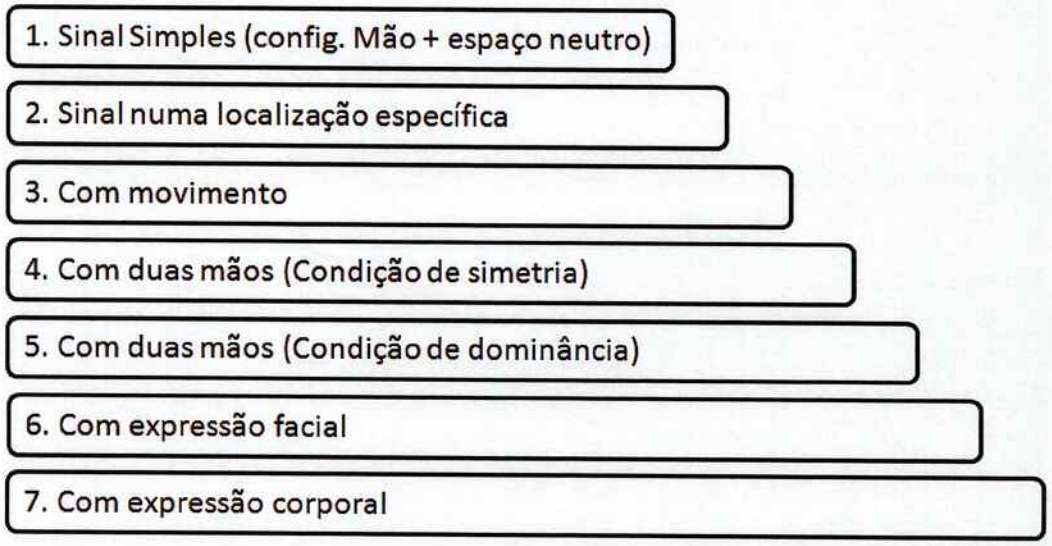

Figura 2.2: Graduação da LIBRAS - nível de palavra

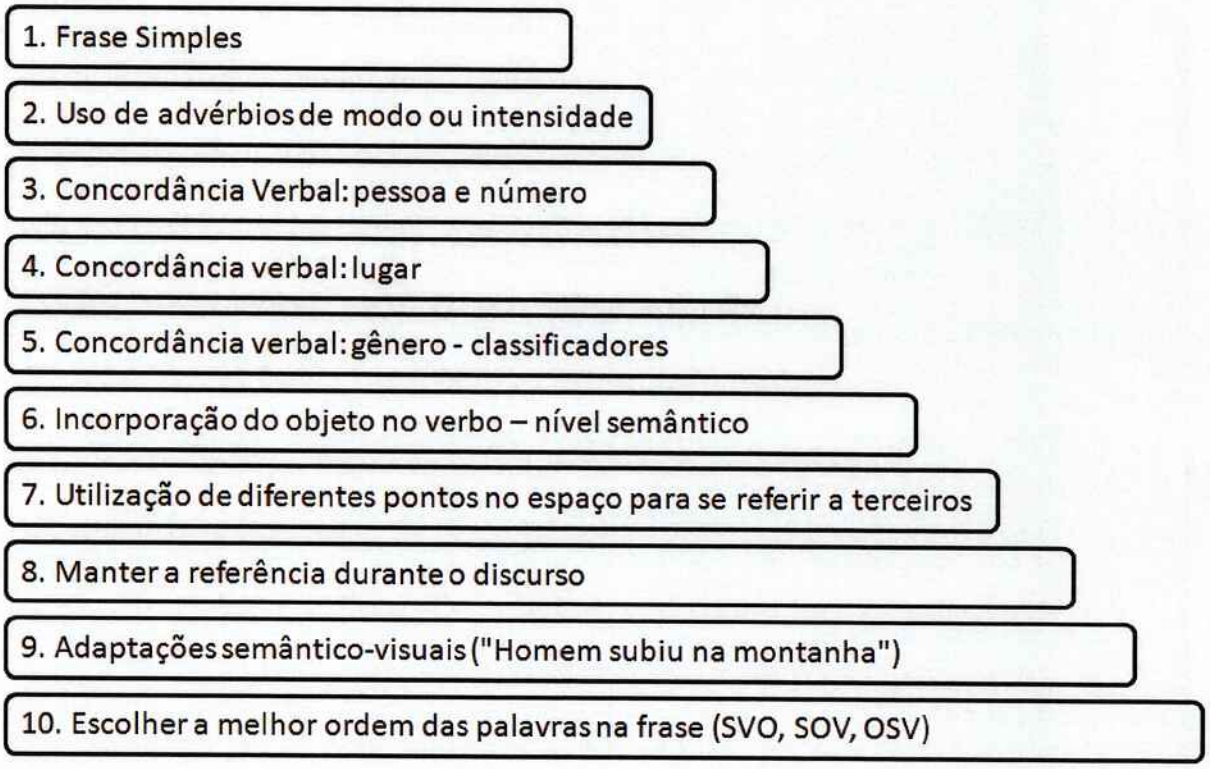

Figura 2.3: Graduação da LIBRAS - nível da frase

2.2 Linguística

2.2.1 Gramáticas para Linguagens Naturais

Uma linguagem formal é um conjunto de cadeias, onde cada cadeia pode ser formada combinando-se os símbolos de um alfabeto Σ, formando então um subconjunto de Σ^*(todas as combinações possíveis de símbolos). Uma gramática é um conjunto de regras que definem a formação dessas cadeias, portanto uma gramática define uma linguagem. Assim sendo, existe uma relação de equivalência entre gramáticas e linguagens, dado um deles existe sempre pelo menos um correspondente do outro.

O linguista Noam Chomsky definiu uma hierarquia com 4 tipos de gramáticas (linguagens), sendo a de tipo 0 (linguagens recursivamente enumeráveis) a que não apresenta nenhuma restrição e cada tipo subsequente: 1 (linguagens sensíveis ao contexto), 2 (linguagens livres de contexto) e 3 (linguagens regulares) mais restritivo que o anterior. Ou seja, tanto para gramáticas como para linguagens: $tipo3 \subseteq tipo2 \subseteq tipo1 \subseteq tipo0$. (CHOMSKY, 1959)

Além da correspondência entre linguagens e gramáticas, há ainda a correspondência desses com os reconhecedores, que aceitam sentenças pertencentes à linguagem correspondente e rejeitam as sentenças não correspondente à linguagem; as correspondências são: linguagens regulares $\leftrightarrow$ autômato finito; linguagens livre de contexto $\leftrightarrow$ autômato de pilha; linguagem recursivamente enumerável $\leftrightarrow$ máquina de Turing.

Uma gramática deve ser capaz de produzir todas as sentenças sintaticamente possíveis da linguagem correspondente e deve ser incapaz de gerar sentenças sintaticamente inválidas para a mesma linguagem. No entanto, devido a enorme complexidade das linguagens naturais, não se consegue encontrar uma gramática perfeita que representa uma língua, sendo qualquer gramática definida no contexto de linguagens naturais uma aproximação da realidade, ou seja, gerará e reconhecerá algumas sentenças inválidas.

Trabalhos de linguistas têm procurado analisar a estrutura sintática de sentenças através de árvores sintáticas (FIORIN, 2005), o que nos leva ao uso das gramáticas livre de contexto, pois essas correspondem a autômatos de pilha, que são capazes de reconhecer estruturas em árvores. O próprio Chomsky também desenvolveu a teoria de *gramáticas transformativas* para efetuar análises e transformações sintáticas de linguagens naturais baseando-se em gramáticas livre de contexto (FRIEDMAN et al., 1971).

Além disso, o uso de gramática livre de contexto para descrever uma linguagem natural, que a princípio parece corresponder a uma gramática irrestrita, se dá pelo fato de que o uso dos tipos superiores implicaria numa complexidade computacional muito elevada para o reconhecimento de sentenças, tornando impraticável seu uso em aplicações em que o tempo do processamento é relevante, caso da maioria delas.

Em (LUFT, 2002) temos um trabalho de um linguista que procura gerar uma descrição do Português com o uso de regras de produção de gramáticas livres de contexto, também apresentando as estruturas sintáticas analisadas na forma de árvores.

2.2.2 Gramáticas transformacionais

A teoria de *gramáticas transformacionais* foi criada e descrita por Noam Chomsky em seu livro Aspects of the Theory of Syntax (CHOMSKY, 1965) em que procura criar uma teoria geral da linguística que evidencie o substrato comum inerente a todas as línguas humanas (esse conjunto de normas bases é a própria "linguagem", a capacidade inata de qualquer ser humano aprender sua língua materna) (LYONS, 1970).

Chomsky faz um avanço significativo no sentido da formalização teórica na área da linguística, no entanto essa formalização ainda fica aquém do necessário para utilização em computadores. Assim sendo, pesquisadores da computação procuraram interpretar a proposta de Chomsky, preenchendo lacunas e fazendo alterações quando necessário para torná-la algo computacionalmente tratável. A primeira proposta de formalização das gramáticas transformacionais foi apresentada pelo livro (FRIEDMAN et al., 1971), escrito em 1971.

Originalmente Chomsky descreve três tipos de representação da linguagem:

- Gramática gerativa (geração linear de sentenças)
- Gramática de estrutura de frase (geração de sentenças com aninhamento)
- Gramática transformativa

Uma gramática transformativa é composta de:

- Estrutura de frase
- Dicionário (léxico)

- Transformações

A *estrutura de frase* é uma gramática livre de contexto ordenada, da mesma forma que gramáticas livres de contexto, mas com algumas restrições adicionais.

Os nós das árvores, que descrevem as estruturas sintáticas, podem ser qualificados por símbolos complexos.

Um *símbolo complexo* é formado por uma lista de *"value + feature"*, em que value = *, +, -, sendo o * um sinal de indefinição, o + de obrigatoriedade de presença e o - de obrigatoriedade de ausência (presença e ausença do featuure).

Features podem ser dos seguintes tipos:

- Categoria: verbo, substantivo, artigo etc.
- Inerência: qualificação subjetiva, como HUMANO, ABSTRATO, COISA, ANIMAL, ANIMADO etc.
- Contexto: descreve uma árvore que se for uma sub-árvore na árvore analisada, pode determinar a obrigação de presença ou ausência de determinado elemento (qualificado por feature de categoria ou inerência).

O *dicionário* da gramática transformativa é composto por um conjunto de definições de features, um conjunto de regras de redundância, e um conjunto de entradas léxicas, sendo cada uma dessas entradas formadas por um vocabulário e um símbolo complexo (vocabulário é formado por vários vocábulos).

A *inserção léxica* é o processo pelo qual se insere vocábulos aos nós da árvore que assim permite. A inserção deve ser feita mediante a análise dos símbolos complexos (um vocábulo possui um símbolo complexo, que deve ser compatível com símbolo complexo do nó no qual será inserido).

A *análise* para que uma transformação ocorra, uma determinada descrição estrutural (structural description) deve estar presente na árvore; além disse se efetua um teste de inclusão entre símbolos complexos (entre a descrição de estrutura e a árvore).

Para que uma inserção léxica ocorra, uma determinda feature contextual deve estar presente na árvore; além disso se efetua um teste de compatibilidade entre símbolos complexos da entrada léxica e do elemento (elemento é nó onde ocorre inserção léxica).

Exemplo de transformação:

```
TRANS PLADEL // identificação
SD % 2 INDEF N |-SG| % . // descrição da estrutura
                         //na qual se aplica a transformação
SC ERASE 2. // mudança estrutural aplicada
```

As mudanças também podem ser mudanças condicionais. Fica assim: *IF restrição THEN mudança ELSE mudança.*

Além das transformações, a terceira componente da gramática transformativa pode contar também com um *programa de controle*, que na verdade é um programa, cujas instruções são as transformações definidas, ou seja, o programa de controle é uma forma de ordenar as transformações e determinar sobre quais pontos elas são aplicadas.

Exemplos de programas de controle:

```
1) CP PASSIVE. // aplica a transformação PASSIVE
               // mudança é efetuada caso haja alguma sub-árvore que
               // condiza com a estrutura definida na transformação.

2) CP PASSIVE, FLIP, REGDEL. // aplica várias transformações

3) CP I. // aplica todas as transformações do conjunto I
         // para cada S existente na árvore
         // isso é efetuado ciclicamente até que
         // não se possa mais operar nenhuma mudança
```

Como uma linguagem de programação, o programa de controle conta também com elementos de controles, que são: IN, RPT (repeat), IF, FLAG, GOTO, TRACE, STOP.

E é com base nesses estudos realizados é que percebe-se que a gramática transformativa é uma ferramenta poderosa na análise de estruturas sintáticas e que já considera vários aspectos previsivelmente relevantes para processos de tradução; um exemplo é o fato do verbo na LIBRAS ter uma concordância quanto ao sujeito ser COISA, ANIMAL OU HOMEM. Trata-se aqui de um caso a ser tratado com as features de inerência.

Torna-se claro aqui que um tradutor em si na verdade pode-se constituir em um programa de controle de gramática transformativa que invocaria as transformações adequadas, nos momentos adequados, nos pontos adequados.

2.3 Tradução Automática

O problema da tradução de um texto em outro feito por máquinas é um problema com diversas soluções adotadas por muitos pesquidores (VAUQUOIS, 1976). Uma abordagem muito utilizada era a tradução por regras, ou seja, definiam-se regras morfogógicas, sintáticas ou semânticas para se fazer a ponte entre as duas línguas. Esse método exige alto conhecimento das línguas, e grande participação de especialistas nelas para a criação das regras. Atualmente, diz-se que estamos na "era da tradução estatística", que segue uma abordagem diferente da por regras. Essa "nova era" basea-se em modelos estatísticos cujos parâmetros são derivados da análise de um *corpus linguístico*, composto por diversos textos nas duas línguas (BROWN et al., 1990) e é amplamente usada por soluções de tradução hoje em dia (LOPEZ, 2008). No entanto, conforme melhor explicado na seção 4.3 o método estatístico não é o mais indicado para o caso de tradução para LIBRAS.

Dentro do campo da tradução baseada em regras, uma escala do nível de complexidade e sofisticação pode ser descrita pela pirâmide de Vauquois (VAUQUOIS, 1976) (também chamada de triângulo de Vauquois) que possui uma versão ligeiramente modificada representada na figura 2.4. Ela apresenta níveis para a tradução, sendo que quanto mais alto na pirâmide, mais "profunda" é a análise feita na linguagem e portanto melhor seria a tradução realizada. No caso desta figura, a pirâmide original foi discretizada em quatro níveis. O primeiro nível, morfológico, representa uma tradução apenas baseada em palavras, ou seja, baseada apenas num dicionário. Esse tipo de tradução é pobre, pois desconsidera qualquer estrutura de frase ou relação entre as palavras. O segundo nível é o sintático, que realiza análise da sintaxe da língua, considerando então esses aspectos mencionados. O terceiro nível corresponde a um estudo das características semânticas do texto, levando em conta o significado das palavras, possibilitando a diminuição de ambiguidades presentes nos níveis abaixo e a identificação expressões linguísticas. O topo da pirâmide seria a tradução perfeita, através de uma única língua intermediária na qual as outras línguas pudessem ser descritas completamente.

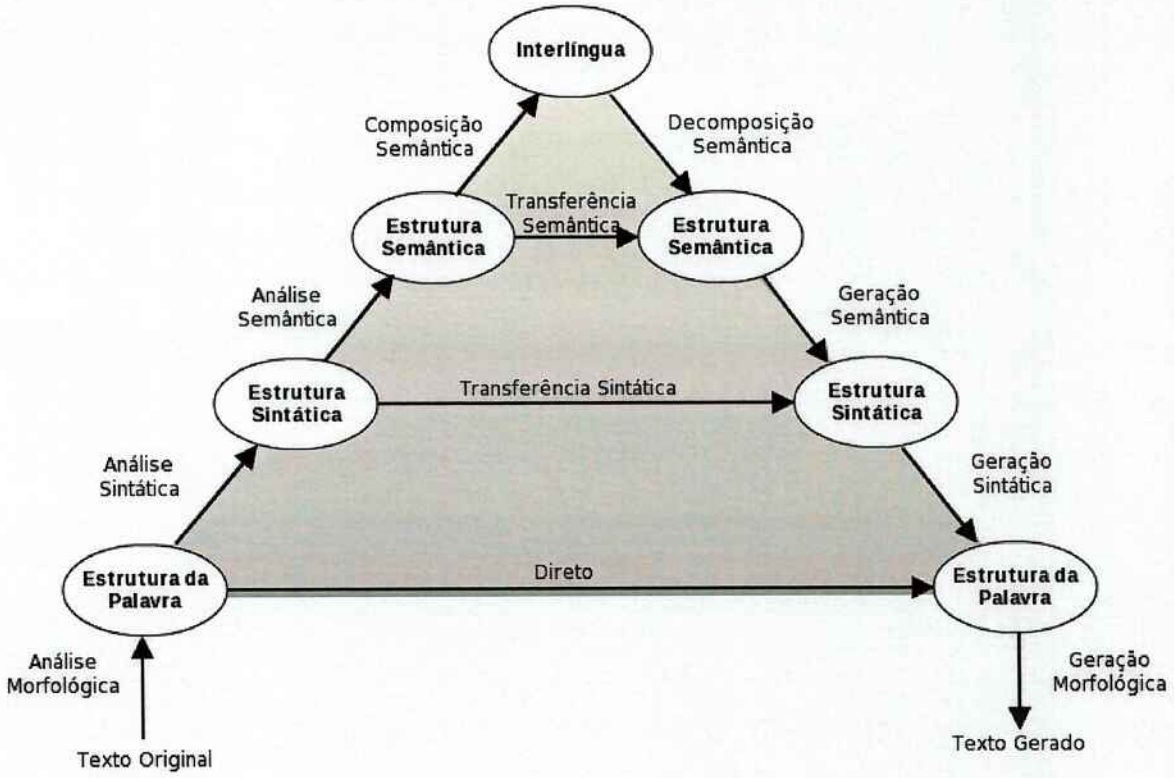

Figura 2.4: Pirâmide de Vauquois - uma versão modificada

Capítulo 3

TECNOLOGIAS AVALIADAS

Em várias partes do projeto nos apoiamos em tecnologias já existentes com o intuito de obter mais agilidade no desenvolvimento e maior qualidade no trabalho final obtido, procurando aproveitar as soluções já existentes para resolver principalmente os problemas mais periféricos do nosso sistema, isto é, os que não são relacionados diretamente com os algoritmos de tradução.

Nesta seção descrevemos as tecnologias que foram analisadas nesse contexto, sendo algumas delas selecionadas e por nós utilizadas.

3.1 Analisadores morfológicos

Foram levantados possíveis analisadores morfológicos para a língua portuguesa, para que servissem de base a alguma implementação de analisador morfológico que satisfizesse à interface levantada, a qual lida das relações entre os analisadores sintático e morfológico.

Os dois analisadores avaliados foram o MXPOST e o JSpell. Estudou-se a complitude das informações fornecidas por cada um e o grau de facilidade prevista para integração com os demais módulos do projeto. As conclusões foram:

MXPOST

A bem de verdade, menos que um analisador morfológico, MXPOST é um programa rotulador (*tagger*), ou seja, ele aplica rótulos às palavras que lhe são apresentadas; isso, com base num conjunto de dados de treinamento que lhe proveram uma aprendizagem supervisionada. Utilizando o trabalho "NILC's Taggers"[1], no qual pesquisadores aplicaram ao MXPOST um treinamento de rótulos morfológicos anexos a textos em português, pode-se utilizar o programa para se descobrirem as classes morfológicas de palavras na língua portuguesa.

[1] NILC's Taggers. Disponível em: <http://www.nilc.icmc.usp.br/nilc/tools/nilctaggers.html>. Acesso em 05/12/2010

O MXPOST apresentou resultados com alto índice de acerto nos testes realizados. Um exemplo de par entrada/saída do MXPOST treinado:

Entrada: Os carros vermelhos bateu no poste.

Saída: Os_ART carros_N vermelhos_ADJ bateu_VERB no_PREP+ART poste_N

No entanto, o MXPOST apresenta algumas limitações que nos levaram a preteri-lo. Em primeiro lugar, não seria fácil usá-lo como biblioteca: as classes existentes tinham nomes pouco intuitivos e não encontramos documentação, além dele também não ser *open-source* e, portanto, não se poder ver seu código-fonte em Java. Além disso, as informações apresentadas pelo programa são limitadas e podem ser insuficientes, pois ele só retorna a classe morfológica: não retorna gênero, número, nem tempo verbal e transistividade para os verbos

JSpell

Outra alternativa encontrada foi o JSpell[2]. Este é um analisador morfológico de código aberto, que apresenta maiores possibilidades e informações em sua saída.

Por exemplo: dado um lexema de entrada, caso a ele correspondam mais de uma interpretação, todas são apresentadas. Ainda, caso não haja derivação própria para o lexema (ou seja, ele é uma palavra desconhecida do JSpell), é retornado um conjunto de possíveis soluções. No idioma inglês, essas soluções para sequências de letras aproximadas são chamadas de *near misses*, e é assim como o JSpell as designa; durante o projeto descrito neste documento, contudo, pela natureza da informação representada, trataram-se as como "soluções aproximadas".

O exemplo acima mostra quanto o JSpell é completo para obtenções sobre o universo ao redor dum lexema, mas não diz nada sobre como o lexema em si é descrito pela ferramenta. Segue uma explicação: para cada possível interpretação, seja para o exato lexema, seja referente a soluções aproximadas, é informada a forma de dicionário da palavra, sua categoria e informações pertinentes à categoria (*e.g.*: para verbos, uma informação pertinente seria, o número, a pessoa, *etc*; já para substantivos, o gênero, *etc*).

O único porém dele é que foi feito para português de Portugal. Uma grande vantagem do JSpell é uma saída com muito mais informações:

[2] JSpell. Disponível em: <http://natura.di.uminho.pt/wiki/doku.php?id=ferramentas:jspell>. Acesso em 05/12/2010.

- Se existe dúvida ou várias possibilidades para uma mesma palavra, ele mostra todas as opções;
- Gênero e número;
- Tempo verbal;
- Identificação dos sufixos e prefixos das palavras compostas

3.2 Ferramentas gráficas

Para implementar o sintetizador de sinais é fundamental o uso de tecnologias de computação gráfica, sendo essas ferramentas divididas principalmente em duas categorias: *ferramentas de modelagem* que geram os modelos tridimensionais e *APIs* que manipulam esses modelos em tempo de execução.

A seguir uma listagem das ferramentas pesquisadas e consideradas neste processo, incluindo ferramentas que se enquadram em uma das duas categorias ou até mesmo em ambas.

Flash

Num primeiro momento considerou-se a possibilidade de seu uso por ser uma tecnologia amplamente difundida, tanto entre usuários quanto entre desenvolvedores e até por observarmos soluções de tradução de LIBRAS (CORADINE et al., 2007) usando Flash.

No entanto, sendo o uso do Flash mais adequado para o 2D, observou-se que embora pudesse ser relativamente fácil criar uma animação de um avatar falando LIBRAS , seria extremamente complexo criar um modelo com as unidades morfológicas segmentadas de forma que pudessem ser sintetizadas e integradas em tempo de execução; para este tipo de tarefa modelos 3D se mostram bem mais adequados.

Blender

Blender é uma suíte de criação de conteúdo 3D *open source*, disponível sob a *GNU General Public License* (GPL) para os principais sistemas operacionais.

Além de permitir a modelagem de objetos 3D, contém também uma *game engine* embutida que permite a manipulação desses objetos em tempo real, inclusive com o uso de scripts na linguagem Python (o que equivale ao que definimos ferramentas gráfiicas do tipo API).

Neste aspecto, um dos mais recentes feitos na ferramenta foi a produção de um jogo, o *Yo Frankie!*, disponível para download, incluindo os arquivos de projeto do Blender.

O site da ferramenta conta com uma farta documentação, de modo que iniciamos o aprendizado através de seus tutorial no wiki do site [3]. Além disso o grupo realizou a aquisição do livro *Mastering Blender* (MULLEN, 2009), que discutia com mais profundidade questões ligadas à game engine, uma vez que a comunidade de usuário em geral encontra-se mais voltada para o uso do Blender como modelador 3D.

Seguindo essas fontes chegamos a construir uma animação de uma mão, sendo construída com base em um modelo de mão baixado da Internet; o modelo continha apenas a "pele"e nós criamos a estrutura de armadura (*bones*) para o modelo e com a manipulação da armadura gerou-se o vídeo com a animação.

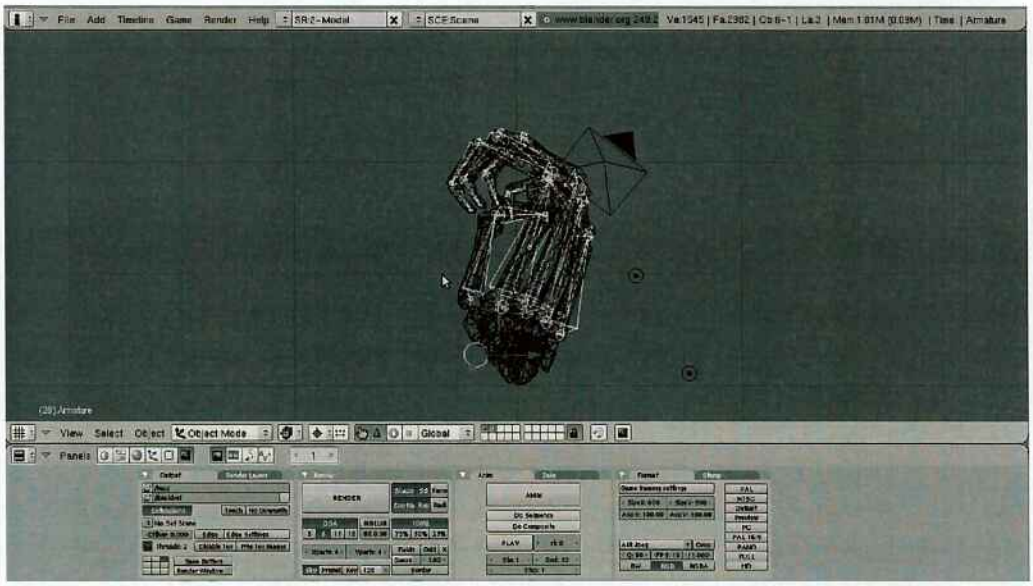

Figura 3.1: Exemplo de tela do Blender

Uma desvantagem do uso do Blender é que parece não haver nenhuma forma de integrar o resultado em um ambiente como a *web* ou mesmo gerar uma aplicação *stand alone* sem que o usuário possua o próprio Blender instalado em sua máquina.

[3] Blender Wiki. Disponível em: <http://wiki.blender.org/>. Acesso em 05/12/2010

Outra suposta desvantagem seria a alta curva de aprendizagem do Blender, característica pela qual é famoso. Mostra disso é que a própria comunidade reconhece que o Blender não foi projetado para ser fácil ou intuitivo, mas sim para ser uma ferramenta de alta produtividade tão logo o usuário a domine. No entanto em nossa experiência com o ambiente, essa dificuldade acabou sendo bem menor do que a esperada devida a essa impressão geral.

Processing

Consiste em uma API *open source* em Java para geração de aplicações interativas em 3D mais simples, de uso geralmente didático, que permite a manipulação direta de modelos no formato texto OBJ, suportado por praticamente todas as ferramentas modelagem 3D.

Sua principal vantagem é a facilidade de aprendizado, mas apresenta graves restrições, como a impossibilidade de lidar com modelos que apresentem *bones*, que são usados para criar vínculos e restrições de movimentos entre partes do modelo a ser animado; de forma geral pode-se dizer que não é indicado para manipulação de personagens animados, embora suficiente e prático para outras aplicações gráficas, como *puzzles* por exemplo.

Outra vantagem é que a aplicação resultante é na verdade um Applet, ou seja, uma aplicação Java integrável com navegadores web.

Panda3D

Assim como o Processing, o Panda3D também é uma API *open source* para manipulação em tempo de execução de modelos gráficos.

Seu diferencial é que esta é uma ferramenta profissional, tendo sido desenvolvida e utilizada pela Disney e pela Carnegie Mellon University's Entertainment Technology Center.

Uma das principais vantagens do Panda3D sobre o Processing é a possibilidade de lidar com o conceito de *bones* em tempo de execução.

Embora tenha uma base bem reduzida de usuário, sua comunidade de desenvolvedores se mostrou bem aberta a novatos, prestando um auxílio bem amigável aos que desejam dominar a tecnologia, ao contrário do que tipicamente acontece na comunidade do Blender, que prefere prezar pela vasta documentação.

Também apresenta vantagem sobre à *game enngine* do Blender, que não possui uma documentação tão vasta em comparação com o total de documentação da aplicação como um todo e também pelo fato de *game engine* do Blender ser a parte mais imatura da ferramenta.

Mas como o Processing é apenas a API ainda seria preciso de uma ferramenta de modelagem, de forma que uma boa combinação seria a do Panda3D como API e do Blender como ferramenta de modelagem.

3DStudio

Assim como o Blender, é também um ambiente de modelagem 3D. A principal diferença é o fato de o 3DStudio ser uma ferramenta proprietária e cara, mas, ao mesmo tempo, possuir uma penetração bem maior no mercado.

Tanto o Blender quando o 3DStudio podem gerar modelos compatíveis com o Processing e o Panda3D.

Comparação

Na tabela abaixo resumimos as principais características das ferramentas de computação gráfica consideradas.

Tabela 3.1: Comparação entre tecnologias de computação gráfica

	Flash	*Blender*	*Processing*	*Panda3D*	*3DStudio*
Modelagem	Sim (2D)	Sim	Não	Não	Sim
API	Sim	Sim	Sim	Sim (a melhor)	Não
Base de usuários	Grande	Média	Pequena	Pequena	Grande
Documentação	Ampla	Ampla	Pequena	Média	?
Linguagem	Action Script	Python	Java	C++ / Python	-
Integração web	Sim	Não	Sim	Sim	-
Curva de aprendizado	Média	Grande	Pequena	Média	?
Open source	Não	Sim	Sim	Sim	Não

Obs1: considerando as linguagens da tabela, a que seria mais vantajosa para nós seria Java, por já possuirmos um domínio maior e pelo resto do projeto ser em Java;

Obs2: não listada na tabela, mas lembrando que o Panda3D ainda apresenta a vantagem adicional da cordialidade de sua comunidade.

Com essa análise, concluímos que a princípio a melhor combinação seria o uso do Blender para modelagem e a do Panda3D para a API; no entanto, por motivos que serão explicados na seção que trata da implementação do sistema, já adiantamos que a combinação escolhida foi a do 3DStudio para a modelagem e a do Processing para a API.

3.3 Frameworks web

Para o desenvolvimento das módulos web estudamos tecnologias que poderiam ser usadas e nos proporcionar um ganho de produtividade, evitando a perda de tempo com a escrita de código repetitivo, o chamdo *boilerplate code*, seja na construção de interfaces e no gerenciamento de navegação entre páginas, quanto na automatização do mapeamento objeto-relacional para armazenar as informações em bancos de dados.

Java EE - Servlets

Java EE (ou J2EE, ou Java 2 Enterprise Edition, ou em português Java Edição Empresarial) é uma plataforma de programação para servidores na linguagem de programação Java.

A tecnologia básica para processamento das requisições na plataforma Java EE são os *servlets*, enquanto que a interface é gerada principalmente pelas *JSP* (Java Server Pages).

Os servlets são basicamente classes definidas pelo programador que realiza o processamento da execução web e gera a página HTML de resposta, ou mais adequadamente, seta parâmetros e encaminha a requisição para uma página JSP que é mais apropriada para o *design* da interface. A parte do sistema que gerencia o ciclo de vida dos servlets e lhes encaminham as requisições chegadas ao servidor é denominada *container*, sendo as implementações open source mais conhecidas o *Tomcat* e o *Glassfish*.

Em geral, os frameworks mais avançadas para Java / web são baseados na tecnologia de servlets.

Em nosso trabalho primeiramente estudamos extensivamente essas tecnologias através de um livro (BATES; SIERRA; BATES, 2008) e iniciamos a implementação com base nele.

No entanto, percebemos que seu uso demandava códigos extensivos e repetidos para tratar a navegação além de fazer o mapeamento entre os parâmetros da requisição e os objetos.

Portanto decidimos verificar a possibilidade de usar frameworks mais avançados.

Struts

Struts foi um dos primeiros frameworks para Java EE, implementando uma camada controladora, de acordo com o modelo MVC de aplicações web, simplificando o uso de servlets, necessitando apenas a criação de classes Action

Este framework foi originalmente desenvolvido por Craig McClanahan e doado para a Apache Software Foundation em 2002, onde continua sendo desenvolvido segundo o padrão desta fundação.

JBoss Seam

É um framework mais avançado desenvolvido pela Red Hat, fazendo parte do bastante utilizado servidor de aplicações JBoss, que consiste em uma plataforma que engloba o container e mais outras facilidades para os desenvolvedores criarem aplicações Java de servidor.

A ideia do framework consiste basicamente em criar uma forma fácil de unir (daí o nome) as tecnologias de componentes EJB (lógica do negócio) e as interfaces JSF (uma "evolução"do JSP). Também oferece componentes que facilitam em muito a criação de CRUD's (formulários para criação, edição, atualização e remoção de entidades do banco de dados).

No entanto, sendo bem poderoso, possui também uma curva considerável de aprendizado, em especial é preciso lidar com vários arquivos de configurações complexos.

VRaptor

É um Framework MVC Java para Web focado em desenvolvimento rápido, prometendo aos seus usuários uma alta produtividade com uma baixa curva de aprendizagem, economizando tempo no desenvolvimento de soluções.

O VRaptor é desenvolvido pela Caelum, escola que oferece cursos de Java em São Paulo.

Assim como nos outros frameworks, a ideia é liberar o desenvolvedor de código repetitivo e tedioso ligado à tecnologia para que se possa focar no desenvolvimento da lógica do negócio.

Alguns aspectos aproveitados é a automatização da conversão de parâmetros da requisição HTTP em objetos esperados pelo controlador e a facilidade de controle da navegação entre as páginas.

Uma característica interessante é que a necessidade da escrita de arquivos de configuração em geral e suplantada pelo uso de convenções na programação das partes da aplicação que interagem com o framework.

Por ter uma estrutura mais simples, com uma documentação direta e de fácil entendimento, e por ser suficientes para nossas necessidades (dado o porte relativamente pequeno de nossos módulos web) foi essa a nossa escolha para framework web.

Hibernate

Quando lidamos com aplicações web é muito comum a necessidade de se armazenar informações no banco de dados, mas dado que na aplicação lidamos com objetos em geral é preciso realizar um complexo processo de conversão dos objetos em dados que possam ser registrados em tabelas do banco de dados e vice-versa.

Felizmente esse trabalho pode ser automatizado com o uso de frameworks que implementam a JPA (Java Persitence API) que define como aplicações Java EE devem realizar esta conversão.

A implementação open source mais conhecida e utilizada da JPA é o Hibernate, que possui suporte a vários bancos de dados disponíveis no mercado, entre eles ao MySQL, banco de dados também open source por nós utilizado.

E mais ainda, vários frameworks, entre eles o Seam e o VRaptor, já possuem mecanismos de integração que facilitam a configuração e uso do Hibernate.

Capítulo 4

ESPECIFICAÇÃO DO PROJETO

4.1 Requisitos

Requisitos funcionais

- O Sistema deve realizar a tradução de um texto em português para um texto em LIBRAS, considerando os aspectos sintáticos das línguas envolvidas.
- O corpo do dicionário devem ser expansível, permitindo que palavras sejam cadastradas gradativamente.
- A base de regras sintáticas da língua portuguesa deve ser facilmente modificável, permitindo adequações iterativas durante o desenvolvimento do projeto e facilitando interações de linguistas com o sistema.
- O resultado da tradução deve ser exibido através de uma animação, uma vez que poucos surdos conhecem um sistema de notação escrita de língua de sinais, como ocorre com o SignWriting .
- O repositório de dados do dicionário deve estar disponível para que outros sistemas possam utilizá-la (via uma arquitetura de serviço)

Requisitos não-funcionais

- Embora a saída ocorra com computação gráfica, espera-se que a animação seja executável em computadores típicos, sem placas especiais para aceleração gráfica.
- Um aspecto desejado para o sistema é o de modularidade, de forma que componentes do sistema possam ser utilizados em outros projetos envolvendo línguas de sinais.
- Para o tradutor espera-se também um sistema o mais transparente possível, sem que haja dificuldades do usuário operá-lo.

- Deseja-se que o sistema seja implementado utilizando plataformas e tecnologias abertas.

4.2 Casos de Uso

Esta seção descreverá os casos de uso esperados do sistema. A figura 4.1 contém o diagrama de casos de uso, que estão detalhados abaixo.

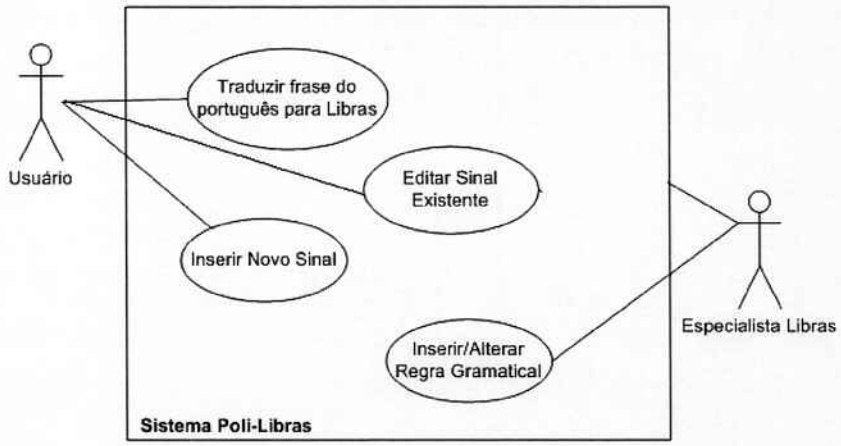

Figura 4.1: Casos de uso

UC1 - Título: Traduzir frase

Atores: usuário

Pré-condições: nenhuma

Fluxo de eventos primário:

1. Usuário acessa o sistema, selecionando opção de "traduzir"
2. Usuário informa ao sistema o texto desejado em português
3. Usuário solicita a tradução
4. Sistema responde com o mesmo texto, mas em Libras.

Fluxos alternativos:

Em 2, caso o usuário entre com um texto em outra língua ou com palavras desconhecidas, o sistema responderá com as palavras através de soletração manual das mesmas, ou seja, a palavra será soletrada através do empréstimo de sinais da Libras.

UC2- Inserir novo sinal

Atores: especialista Libras

Pré-condições: nenhuma

Fluxo de ventos primário:

1. Usuário acessa o sistema, selecionando opção de "Inserir novo sinal"
2. Usuário fornece os parâmetros do sinal
3. Sistema responde como ficaria o sinal descrito
4. Usuário valida a saída
5. Usuário fornece significado do sinal em português
6. Sistema adiciona sinal a sua base

Fluxo de eventos alternativos:

Em 4, usuário pode rejeitar a resposta e voltar ao passo 2.

Pós-Condições: novo sinal adicionado ao sistema.

UC3 - Editar sinal existente

Atores: especialista Libras

Pré-condições: nenhuma

Fluxo de ventos primário:

1. Usuário acessa o sistema, selecionando opção de "Editar sinal existente"
2. Sistema apresenta lista de todos os sinais disponíveis
3. Usuário escolhe um sinal a ser editado

4. Sistema apresenta todos os parâmetros atuais do sinal
5. Usuário modifica os parâmetros que deseja
6. Sistema responde como ficaria o sinal descrito
7. Usuário valida a saída
8. Usuário fornece significado do sinal em português
9. Sistema salva modificações do sinal a sua base

Fluxo de eventos alternativos:

Em 7, usuário pode rejeitar a resposta e voltar ao passo 5.

Pós-Condições: sinal modificado no sistema.

UC4 - Inserir/Alterar regra gramatical

Atores: especialista Libras

Pré-condições: nenhuma

Fluxo de ventos primário:

1. Usuário acessa o sistema, selecionando opção de "Alterar regra gramatical"
2. Sistema apresenta lista de todas as regras gramaticais
3. Usuário edita as regras ou insere novas
4. Sistema salva modificações na sua base

Pós-Condições: regras modificadas no sistema.

4.3 Decisões de Projeto sobre a Tradução Automática

Apesar de na Tradução por Máquina os modelos estatísticos serem os mais comumente usados, o presente trabalho possui uma grande restrição quanto ao uso deste método: a língua objeto é a LIBRAS , que é constituída por sinais e é espaço-visual, não

possuindo forma escrita. Portanto, praticamente não existem textos escritos em LIBRAS para servirem de *corpus* e os que existem, estão em notações difíceis de serem processadas ou pouco utilizadas, como o SignWriting . Dados esses fatos, optamos por usar o método da tradução por regras neste trabalho.

Dos vários níveis de tradução apresentados na seção 2.3, este tradutor atuará no segundo, o nível sintático, i.e., a ponte entre as línguas será na transformação da estrutura sintática do português para a estrutura sintática de LIBRAS . Esse nível foi escolhido por representar um boa relação custo-benefício, não apresentando a simplicidade indesejada de uma tradução morfológica mas não entrando ainda no mérito da análise semântica, por essa apresentar complexidade muito elevada que fugiria do escopo desse trabalho.

A tese defendida por Quadros (1999) apresenta uma proposta do que seria a estrutura frasal de LIBRAS , mais especificamente a estrutura de uma oração. Ela dividiu-a em duas categorias, para verbos *plain* e *non-plain*, ou seja, definiu duas árvores sintáticas para a LIBRAS . Portanto essas estruturas serão consideradas neste trabalho para a definição das regras.

4.4 Arquitetura

A arquitetura do sistema de tradução Poli-Libras, isto é, a representação das unidades de software que constituem o sistema (também chamados módulos) e o relacionamento entre elas, pode ser descrita resumidamente pela figura 4.2.

Destaca-se já que vários desses blocos são unidades independentes que podem ser diretamente reaproveitadas em outras aplicações

A seguir descreve-se cada um desses módulos do sistema, especificando função, entrada e saídas esperadas.

4.4.1 Modelo de dados

Uma questão fundamental do presente do trabalho é como se pode codificar digitalmente uma sequência de sinais (de LIBRAS ou de qualquer outra língua de sinal) para que eles possam ser processados por aplicações diversas.

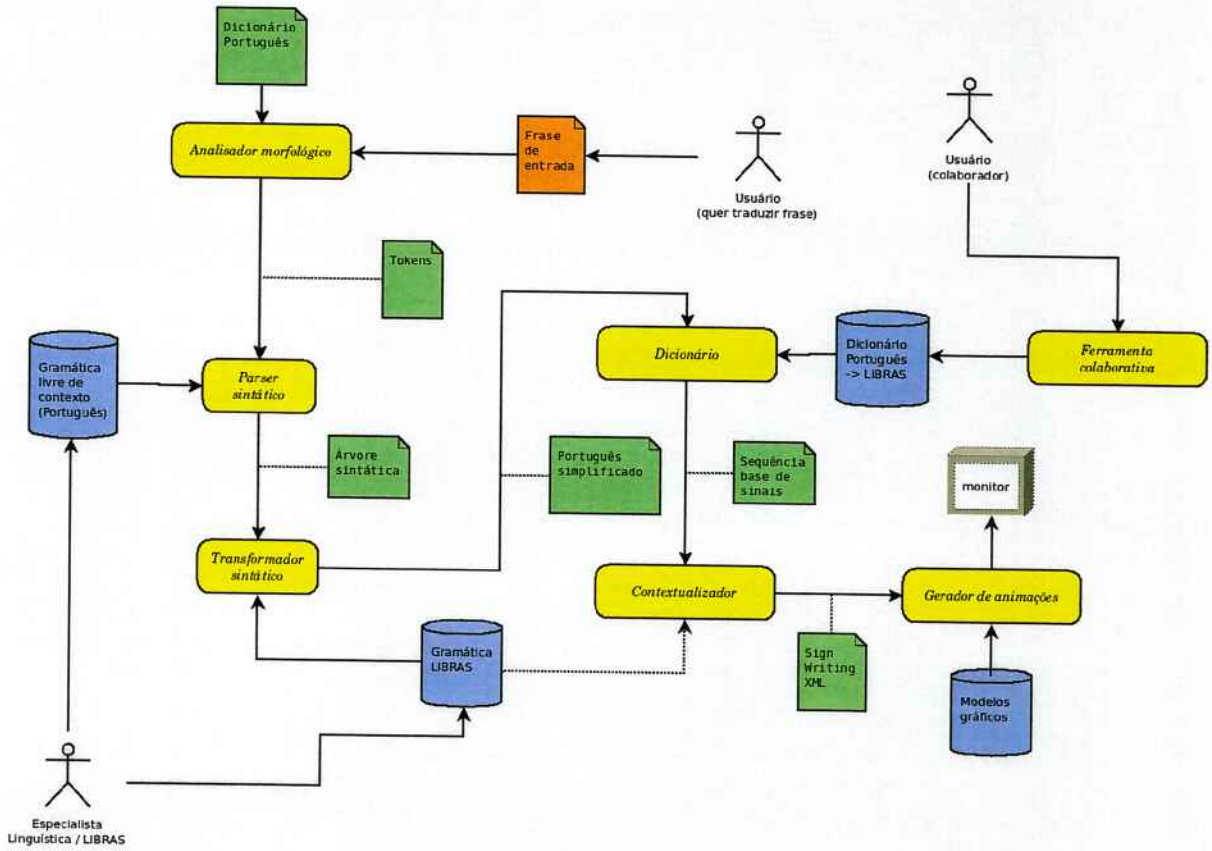

Figura 4.2: Arquitetura do sistema

Essa questão aproximadamente corresponde ao problema de como se registrar por escrito textos em língua de sinais, o que como já descrevemos na seção de aspectos conceituais, ainda não se encontra bem resolvido dentro da própria comunidade surda.

De início, uma possibilidade para nosso sistema de tradução seria apresentar a saída já nessa notação de SignWriting , o que também contribuiria mais facilmente pra construção de um legado escrito em LIBRAS . No entanto, como também já foi exposto, atualmente no Brasil a aceitação do SignWriting pela comunidade surda é baixa, de forma que optamos por utilizar este modelo apenas como sistema de codificação "interno" dos sistemas que o utilizam, ou seja, de forma que o usuário que saiba LIBRAS não precisa saber SignWriting para operar nossos sistema.

Dessa forma, nosso sistema de codificação possui atributos diretamente correspondentes aos atributos expressos graficamente em um símbolo de SignWriting .

Fizemos essa correspondência baseados principalmente em (SUTTON, 2003), obra que descreve de forma extensiva os atributos e seus possíveis valores no SignWriting ;

uma complementação de alguns aspectos da notação (possibilidades de movimentação das mãos) foram retirados da obra de Quadros (1999).

Dessa forma, pudemos definir uma estrutura de dados capaz de codificar os sinais, representada pela figura 4.3

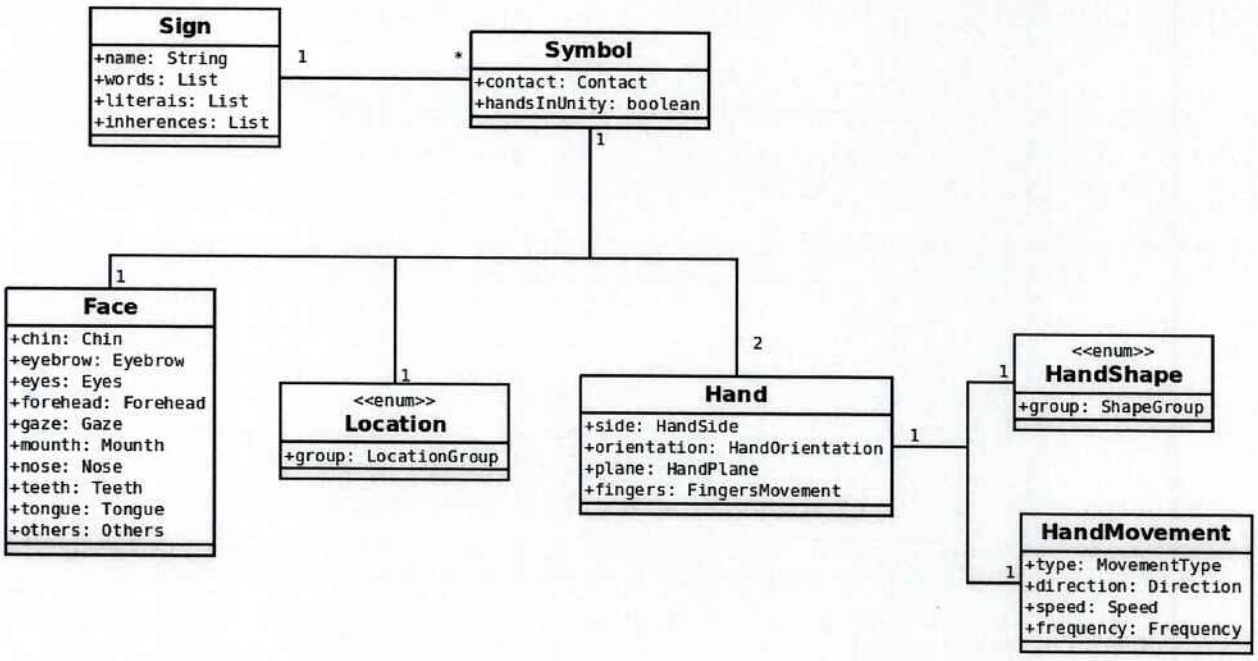

Figura 4.3: Modelo de dados

A figura mostra que um *sinal* é modelado como uma sequência de *símbolos*, sendo um símbolo aquilo que pode ser representado por uma símbolo do SignWriting ; criamos essa distinção entre sinal e símbolo devido a certos "sinais compostos" como AA-BB, em que "AA" possui uma forma de escrita em SignWriting e "BB" outra, sendo que dessa forma "AA-BB" fica sendo o sinal, enquanto "AA" e "BB" são símbolos.

Além da sequência de símbolos, agregamos ao sinal também informações de cunho sintático que podem auxiliar o tradutor em sua tarefa.

Dessa forma, temos a seguinte hierarquia de atributos para descrever o sinal:

```
Sinal: expressa uma ideia, que normalmente corresponde a uma
   palavra em português;
    nome: designação do sinal, como se fosse uma palavra;
    palavras: palavras do português que podem corresponder ao
              sinal;
    literais: palavras que devem aparecer no contexto do sinal;
```

```
inerências: classes de palavras que devem aparecer no
            contexto do sinal (ex: palavras que representam
            animais);
Símbolo: representa uma combinação de aspectos morfológicos
        que podem ser expressos com um símbolo, ou seja,
        uma figura do SignWrite;
    Locação: em que parte do corpo as mãos se posicionam;
    Contato: forma de interação da mão com a locação;
    Mãos em unidade: indica que se as duas mão vão para o
                ponto de locação ou se a mão não-dominante
                permanece em espaço neutro;
    Expressão facial: define características de bochecha,
                olhos, sobrancelhas, testa, olhar, boca,
                nariz, dentes, língua e outros;
    Mão (dominante e não-dominante):
        Configuração: como os dedos estão articulados;
        Orientação: se a palma está voltada para o emissor
                ou contra ele, voltada para o chão ou
                para o céu;
        Plano: se o braço com a mão encontram-se na vertical
               ou na horizontal;
        Dedos: como os dedos se movimentam.
```

Com base nessa estrutura de dados definida primeiramente sintetiza-se em um conjunto de *classes*, de acordo com o paradigma de Orientação a Objetos.

Essas classes são usadas posteriormente para servir de entrada a processos que definem estruturas em tabelas para um banco de dados capaz de armazenar esses dados e a estrutura (ou *schema*) de documentos XML que também possam representar esses dados.

4.4.2 Analisador Morfológico

A figura 4.4 ilustra a especificação deste módulo.

Para que analisadores sintáticos tenham foco na análise sintática das frases, sendo esta, como o nome indica, seu objetivo supremo, faz-se necesserária certa independência

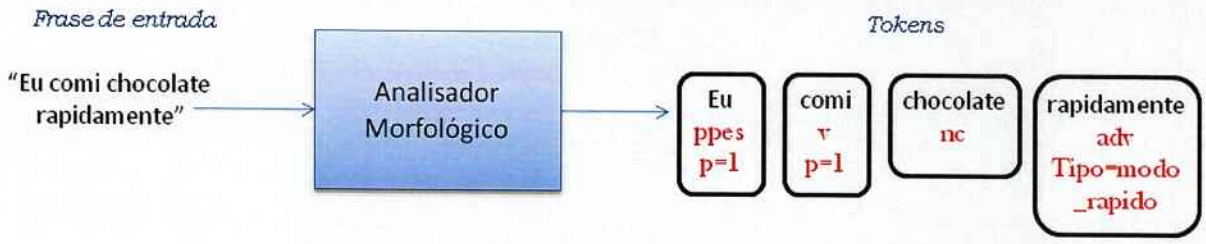

Figura 4.4: Especificação do analisador morfológico

quanto à obtenção dos lexemas. Visando a prover esse desacoplamento ao analisador sintático, surge a ideia de se usar um analisador morfológico. Diz-se que esta estrutura subjaz àquela, pois um analisador sintático se utiliza de analisadores morfológicos obtendo *tokens* sintáticos à medida que sua análise, sintática, prossegue.

Deve-se observar que a verificação de pontuações e, logicamente, de sinais diacríticos, nessa estrutura, cabe ao analisador morfológico. Mas por não ser função deste tratar esses sinais, ele deve notificá-los à camada sobrejacente através de *tokens* sintáticos acordados.

Tipicamente, um analisador morfológico consome *tokens* morfológicos até que a maior quantidade possível destes seja consunta, ao que retorna ao autômato sobrejacente algum *token* correspondente.

No caso de linguagens naturais é comum que à mesma sequência de *tokens* morfológicos se associem mais de uma acepção sintática. Exemplifica-se: Na frase "como uma onda no mar", o analisador morfológico ficaria dúbio sobre decidir se a sequência de letras (e, portanto, de *tokens* morfológicos) "como" é um advérbio, uma conjunção ou, ainda, se corresponde à conjugação da primeira pessoa do singular do presente do indicativo da voz ativa do verbo "comer".

De fato, essa decisão não lhe cabe. Ele deve apenas informar ao analisador sintático as possíveis acepções morfológicas do *token* em questão. Ou seja, para o exemplo anterior, as três acepções possíveis para o lexema "como" devem ser retornadas.

4.4.2.1 Tokens morfológicos

Foram previstas para os *tokens* morfológicos as classes de características listadas a seguir, sendo que não necessariamente um *token* possui todas essas classes definidas (pelo contrário, há classes de características que gramaticalmente não podem coexistir). Essas classes são muitíssimo inspiradas no dicionário padrão disposto pelo projeto JSpell.

Vale notar ainda que apenas a característica "categoria"é de uso obrigatório, pois é essencial para análises sobrejacentes.

CAT, "categoria".

Valores possíveis e significados:
adj: adjetivo
adv: advérbio
art: artigo
a_nc: adejetivo/substantivo comum
card: numeral cardinal
con: conjunção
cp: contração
in: interjeição
nc: substantivo comum
nord: numeral ordinal
np: nome próprio
pass: partícula apassivante
punct: pontuação
ppos: pronome possessivo
pind: pronome indefinido
pdem: pronome demonstrativo
pint: pronome interrogativo
ppes: pronome pessoal
prel: pronome relativo
prep: preposição
v: verbo

Características para adjetivos e substantivos (*adj, a_nc, nc , np*)
nome: <LA>

Descrição: "leva artigo", comum para topônimos.

Valor possível e significado:
1: sim

nome: <G>, "gênero"

Valores possíveis e significados: _: indefinido
f: feminino
m: masculino
n: neutro
2: ambos masculino e femíneo

nome: <N>, "número"

Valores possíveis e significados:
n: neutro
p: plural
_: indefinido
s: singular

nome: <GR>, "grau"

Valores possíveis e significados:
sup: superlativo
dim: diminutivo

Verbos, *v*

Os verbos também devem ser rotulados, além de com o número, com as pessoas e transitividade e com o tempo. Assim, são específicos da categoria:

nome: <P>, "pessoa"

Valores possíveis e significados:
1: primeira
3: terceira
2: segunda
1_3: primeira/terceira

nome: <T>, "tempo"

Valores possíveis e significados:
ip: infinitivo pessoal
inf: infinitivo

pp: pretérito perfeito
ppa: particípio passado
pc: presente do conjuntivo
pic: pretérito imperfeito do conjuntivo
c: condicional
p: presente
fc: futuro do conjuntivo
g: gerúndio
pmp: pretérito mais que perfeito
pi: pretérito imperfeito
f: futuro
i: imperativo

nome: <TR>, "transitividade"

Valores possíveis e significados:
_: transitivo/intransitivo
i: intransitivo
t: transitivo

Dos Advérbios, *adv*

Para essa classe gramatical, será retornada marcação também de sua subcategoria, no seguinte molde:

nome: <SUBCAT>, "subcategoria adverbial"

Valores possíveis e significados:
lugar: advérbio de lugar
modo: advérbio de modo
neg: advérbio de negação
quant: advérbio de quantidade
tempo: advérbio de tempo

Contrações, *cp*

No caso de contrações envolvendo advérbios, artigos, preposições ou pronomes, as seguintes *tags* serão utilizadas quando conveniente:

Notação:

tag: possibilidade, possibilidade, (...).

Lista:

Adv: a, onde, ali, algures, aqui, outrora.
Art: as, a, um, o.
Prep:em, por, de, a, com.
Prep2: entre.
Pdem: isto, isso, esse, aquele, aquilo, este.
Pdem2: outro.
Pind: algo, outro, outrem, algum, alguém, o.
Ppes: sigo, ele, te, tigo, nosco, me, vosco, lhe, migo.

As marcações *Prep2* e *Pdem2* devem ser utilizadas quando a contração ocorrer entre dois elementos da mesma classe, sendo essas marcações destinadas à identificação do segundo elemento da contração.

Da Semântica ou das Características Inerentes

As características de inerência necessárias para uma conversão de português escrito para LIBRAS mais correta devem ser marcadas com o rótulo *SEM*, que já é o modo utilizado pelo analisador morfológico Jspell para informar ao pesquisador sobre características semânticas da palavra. Assim,

Característica: <SEM>

Valores propostos e significados:

rio: rio
mes: mês
livro: obra literária
po: povo
ter: localidade
country: país
n: número romano

mar: referente a marca
p: nome português
p1: nome estrangeiro
pl: planeta
sigla: sigla
mitol: seres mitológicos
cont: continente
cid: cidade
lingua: língua
instituicao: instituição
servico: serviço
proj: Projeto

4.4.3 Analisador sintático

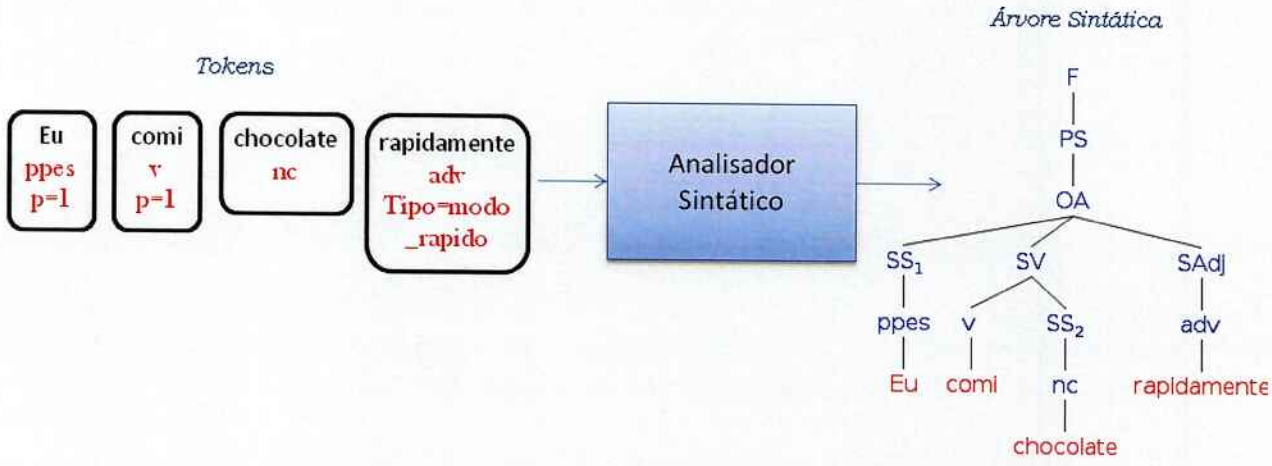

Figura 4.5: Especificação do analisador sintático

O analisador sintático recebe uma sequência de tokens e primeiramente diz se esta sequência é gramaticalmente correta, ou seja, se pertence a língua portuguesa. Esta operação modela a língua Portuguesa como uma linguagem livre de contexto e baseia-se em uma gramática livre de contexto formulado com base no trabalho Moderna Gramática Brasileira (LUFT, 2002). Sendo a frase gramaticalmente aceita, o analisador sintático deve retornar a árvore sintática da frase para que possa ser operada na próxima fase. Esse é o comportamento descrito pela figura 4.5.

Aqui está um dos ponto de maior limitação do trabalho, ao querer tratar a língua portuguesa como uma linguagem livre de contexto. Para isso define-se um escopo de frases formais e "bem comportadas".

Nesta etapa já se elimina algumas ambiguidades do analisador morfológico (ex: "casa" pode ser substantivo ou verbo (casar)), uma vez que nem todas as combinações lineares de classes gramaticais podem gerar árvores válidas; mas ainda assim poderemos ter mais de uma árvore válida, situação que poderia ser resolvida com análise semântica.

Para que a atuação deste componente possa ser melhorada, as regras da gramática livre de contexto deverão ser passíveis de edição por especialistas linguistas.

4.4.4 Transformador Sintático

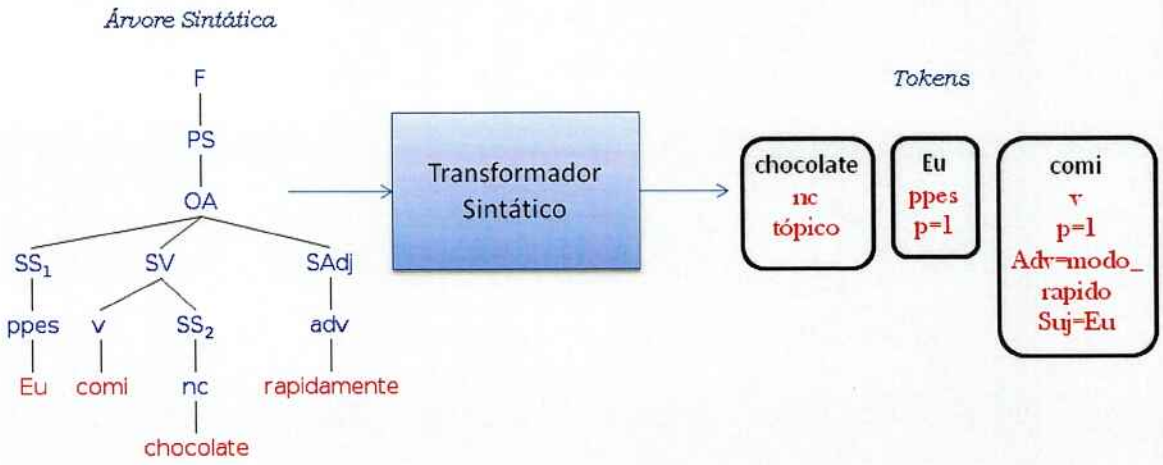

Figura 4.6: Especificação do transformador sintático

O transformador sintático é o primeiro elemento que realiza a ponte entre as duas línguas no nosso tradutor. Como a tradução que propomos é no nível da sintaxe, cabe ao transformador sintático, através de regras que mapeiam a sintaxe do português para a sintaxe de LIBRAS , modificar a estrutura da frase em português para uma estrutura de LIBRAS , de modo que ela seja sintaticamente correta na língua destino e a sua semântica seja mantida.

Como se vê na figura 4.6, o objetivo do módulo do transformador sintático é receber uma árvore sintática numa língua (português) e devolver a sequência de *tokens* (palavras) resultante. Opera, portanto, sobre a árvore sintática obtida, executando transformações definidas numa gramática transformativa a fim de se obter uma frase em "português sim-

plificado", que seria a forma de escrita com palavras em português que mais se aproxima da sintaxe da LIBRAS . Essa sequência ainda contém palavras em português, mas organizadas de forma a representar uma frase em LIBRAS , exatamente do mesmo modo que usamos o português para representar as frases em LIBRAS na forma escrita ao longo deste trabalho, com a adição de informações sintáticas e morfológicas de cada palavra.

O transformador sintático é baseado em regras gramaticais da LIBRAS , porém desconhece como as palavras podem ser representadas por sinais (morfologia da LIBRAS). Esta etapa envolve os seguintes problemas:

- remoção de artigos e preposições desnecessários
- posicionamento dos advérbios de tempo no início das frases
- colocação do adjetivo após o substantivo qualificado
- detectar comparação (igualdade ou desigualdade), e fazer as trocas adequadas
- posicionamento dos pronomes QUEM, ONDE, QUE
- posicionamento de QUAL, COMO e PARA-QUE (início) e de POR-QUE (fim)
- alteração da ordem da sentença (SVO, SOV, OSV) conforme verbo presente

4.4.5 Contextualizador

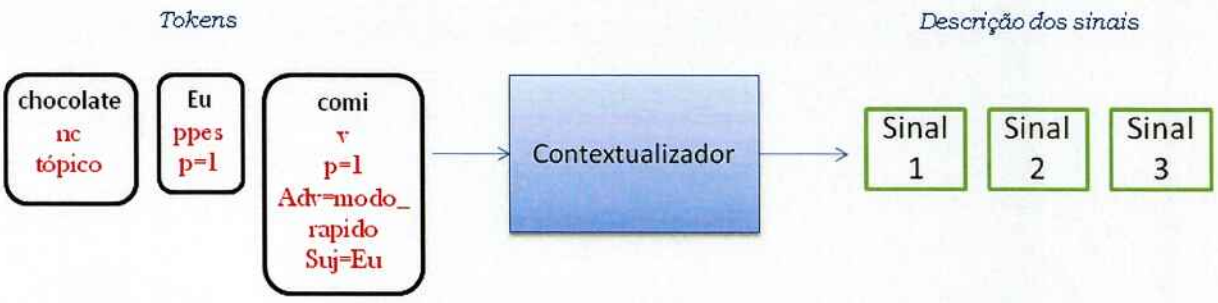

Figura 4.7: Especificação do contextualizador

Esse módulo atua após a oração ter sido simplificada pelo transformador sintático e os sinais correspondentes, obtidos pelo dicionário português–LIBRAS , conforme a figura 4.7. Será visto que pode haver alterações sobre a realização prevista para os sinais, oriundas

puramente de análise de contexto ou da combinação desta com desamgibuação. Alterações de origem contextual ocorrem tipicamente na presença de advérbios, do contexto circunjacente ao verbo principal. Já a desambiguação, essa ocorre quando há mais de um sinal para a palavra portuguesa a ser traduzida.

Sua atuação é particularmente notável em casos como o do advérbio "rapidamente". Esse advérbio não é transmitido pelo falante de LIBRAS através de um novo sinal, mas sim alterando características do sinal principal, do verbo. Nesse caso, a informação de que o verbo principal está acompanhado pelo advérbio "rapidamente" é adicionada pelo transformador sintático às informações contidas no *token* verbal. Futuramente, após o módulo contextualisador realizar as conversões básicas de *tokens* para sinais, os quais contêm a descrição sobre como o sinal deve ser realizado, ele deve modificar os atributos descritivos adequadamente, agregando informações fornecidas pelo transformador sintático.

Continuando com o exemplo, o contextualisador aumentaria o atributo *speed* do objeto *HandMovement* das mãos utilizadas pelo sinal correspondente ao verbo modificado.

Para a frase "comer chocolate rapidamente", o fluxo de informação ocorre como descrito:

1. o transformador sintático inclui no *token* de "comer" a informação de que o advérbio de intensidade "rapidamente" o modifica.
2. o contextualisador relaciona o advérbio "rapidamente" à velocidade de realização do sinal "comer"
3. retorna a lista de objetos *sinais*.

Conforme as características descritas na seção 2.1, outras alterações podem ser:

- procura do sinal mais adequado para uma posição ambígua com base na análise de contexto;
- aspecto tempo pode determinar frequência de movimento;
- o ponto de articulação pode ser uma marca de concordância verbal com o advérbio de lugar;
- configuração de mão pode se alterar conforme sujeito; variando se pessoa, animal ou coisa;

- a orientação pode ser uma concordância número-pessoal;
- algumas palavras mudam de sinal conforme o contexto (análise de literais ou características de inerência circundantes);
- negação formada por movimento contrário ou por movimento da cabeça;
- detectar variações (de número) dos pronomes pessoais;
- variação do pronome QUEM conforme o contexto;
- adicionar expressão interrogativa / exclamativa conforme pontuação encontrada;
- desambiguação de QUANDO (três sinais possíveis);
- desambiguação de DIA (dois sinais possíveis);
- desambiguação de numerais.

Por ora não há definição formal de uma estrutura de dados que seja capaz de dizer ao contextualizador o que fazer. Portanto, embora se possa dizer que este componente é construído com base nos conhecimentos da gramática de LIBRAS , por enquanto não se pode separar este conhecimento, típico do especialista linguista, da implementação da ferramenta (feita pelo desenvolvedor).

Nesse módulo há tratamento especial para algumas palavras, como verbos de ligação e advérbios que alterem a morfologia dos sinais (ex: rapidamente); esse tratamento "hard-coded" não é de todo absurdo, uma vez essas categorias são constituídas por listas fechadas de palavras.

4.4.6 Sintetizador de sinais 3D

A saída gerada pelo tradutor, uma codificação dos sinais da sentença em XML, pode ser utilizada por vários tipos de ferramentas, como um sintetizador de SignWriting ou mesmo por um texto plano que descreva os sinais, mas neste trabalho a solução adotada será uma saída com animação por computação gráfica, em que um avatar virtual realiza os sinais descritos no XML.

Assim, o sintetizador consiste em um módulo acoplável a outras aplicações que recebe a descrição em XML de uma sentença em LIBRAS , descrição essa baseada no nosso mo-

delo de dados, e sintetiza os movimentos necessários de acordo com os atributos setados nessa descrição.

4.4.7 Ferramenta colaborativa

A ferramenta colaborativa trata-se de uma aplicação web voltada para a criação e consulta de verbetes no dicionário Português - LIBRAS de forma colaborativa, conceitualmente similar à Wikipedia, e que permite o acesso via software desses verbetes, na forma de WebServices. Essa ferramenta foi batizada de "WikiLibras".

Esta ferramenta é muito importante para o projeto na medida em que permite a colaboração de outras pessoas para o preenchimento de um vocabulário de tradução, que dificilmente atingiria um tamanho significativo se dependesse apenas da equipe de desenvolvimento.

O WikiLibras também deve subsistir como um importante projeto independente, uma vez que o banco de dados por ela gerado pode fornecer dados pra outras aplicações de LIBRAS. Este último aspecto é no que consiste o módulo de Dicionário do sistema.

Telas (ou fluxo de navegação)

As funcionalidades do sistema WikiLibras (para o usuário web) são a inserção, edição e consulta de sinais.

A criação/edição de um sinal é feita através da sequência de formulários das seguintes páginas:

- nome do sinal: no caso de criação o sinal não deve existir e no caso de edição já deve existir;
- informações sintáticas do sinal: informa-se palavras (em português) que no texto corresponderão ao sinal assim como informações de contexto que poderão ser utilizadas na tradução;
- informações gerais do símbolo: informa-se a localização do sinal e ponto de contato;
- informações da mão dominante do símbolo: informa-se a configuração, orientação e plano da mão (neste formulário a escolha da configuração de mão é auxiliada pela exibição de fotos das configurações correspondentes);

- informações do movimento da mão dominante: informa-se tipo, direção, velocidade e frequência do movimento desta mão;
- informações da mão não dominante;
- informações do movimento da mão não dominante;
- expressão facial do símbolo: informa-se características das bochechas, sobrancelhas, olhos, testa, olhar, boca, nariz, dentes, língua e outros.

A nomenclatura das informações de contexto são inspiradas na *gramática transformativa* e são os:

- *literais*: palavras que devem aparecer no contexto daquele sinal (exemplo: sinal SENTAR-EM-RODA é substituído no lugar da palavra *sentar* que possua a palavra *roda* em seu contexto);
- *inerência*: similar aos literais, mas denotam um conjunto de palavras contextuais representadas por um aspecto mais abstrato que essas palavras possuem em comum (ex: *armas* é inerência de espada, faca, granada etc).

Já a nomenclatura dos atributos dos símbolos foi baseado no modelo de classes anteriormente já modelado, que por sua vez baseia-se em obras como *Lições sobre o SignWriting* (SUTTON, 2003) e *Língua de Sinais Brasileira - Estudos Linguísticos* (KARNOPP; QUADROS, 2004).

É importante que no processo de preenchimento do formulário permita-se que um sinal tenha vários símbolos, assim como na edição se permita a edição de qualquer um dos símbolos existentes para o sinal.

Além da edição / criação do sinal, o usuário pode também realizar uma busca por sinais já existentes. Para tal o usuário preenche um valor em um campo de texto, e a busca retornará os sinais com nomes similares e os sinais cujas palavras correspondentes em português sejam similares ao argumento da busca.

4.4.8 Dicionário

O dicionário consiste no bloco responsável por converter palavras em português escrito para a representação codificada em LIBRAS , ou seja, implementa uma função que

recebe uma string e retorna uma conjunto de sinais (de acordo com o modelo de dados já especificado).

Nosso dicionário irá consumir os dados disponibilizados pelo *WebService* do sistema WikiLibras, sendo esses dados povoados de forma colaborativa pelos usuários do sistema.

As operações definidas no *WebService* são as seguintes:

```
Nome: signByName
Descrição: Obtêm um sinal pelo seu nome
Parâmetro: nome exato do sinal (ex: SENTAR-EM-RODA)
Retorno: o sinal correspondente ou valor nulo
         em caso de não haver sinal

Nome: nearSigns
Descrição: Obtêm os sinais cujos nomes
           se assemelham ao argumento
Parâmetro: o nome aproximado do sinal (ex: SENTAR)
Retorno: os sinais correspondentes
 (ex: SENTAR, SENTAR-EM-RODA, SENTAR-NA-CADEIRA)

Nome: translate
Descrição: Traduz uma palavra em português para LIBRAS
Parâmetro: palavra em português
Retorno: lista dos sinais que possivelmente
         correspondem à palavra dada

Nome: simpleTranslate
Descrição: Traduz uma palavra em português para LIBRAS;
 caso haja vários possíveis sinais correspondentes,
 apenas um sinal será retornado
Parâmetro: palavra em português
Retorno: um sinal em LIBRAS que
         corresponde à palavra dada
```

Para garantir a demonstração do sistema em caso de ausência de conexão com a Internet, o dicionário deve ser capaz de especificar a localização do *WebService* a ser

consumido, para que assim possamos definir o *WebService* em *localhost* e utilizar um WikiLibras local.

Capítulo 5

IMPLEMENTAÇÃO

Neste capítulo descreve-se a metodologia de desenvolvimento adotada, bem como detalhes técnicos da implementação realizada.

Para a implementação deste projeto, os integrantes do grupo optaram por escrever todos os programas utilizando a linguagem Java. Essa foi a linguagem escolhida pelo fato dos membros do grupo já terem domínio dela, dela ser portátil e independente de sistema operacional, de ser largamente utilizada mundialmente, possuir muitas ferramentas e bibliotecas livres e possuir facilidade para desenvolver aplicações web.

5.1 Metodologia de desenvolvimento

Como proposto pela coordenação da disciplina, a estratégia geral de desenvolvimento foi de ter o primeiro semestre para estudos e especificação e o segundo semestre para a implementação.

Durante o primeiro semestre foi necessário empreender estudos relacionados ao domínio do nosso problema e avaliações de opções de tecnologias que poderiam ser empregadas.

Os estudos relacionados ao domínio do nosso problema eram basicamente os ligados à LIBRAS, em especial a notação SignWriting, e questões linguísticas, como o estudo feito sobre gramáticas transformativas.

Quanto às opções tecnológicas, foram estudos sobre bibliotecas e frameworks a serem utilizados, indo desde a parte gráfica, até o reconhecimento de categorias gramaticais.

Em bases desses estudos e avaliações pudemos descrever a especificação do sistema, que consistiu basicamente na arquitetura do sistema e função de cada um dos blocos, acompanhados já de uma análise de domínio, bem como considerações de requisitos funcionais e não-funcionais.

Durante o primeiro semestre, também antecipamos a construção do módulo do sintetizador de sinais em 3D, pois este foi desenvolvido como um projeto para a disciplina PCS2520 - Tecnologias de Computação Gráfica.

A metodologia de implementação para o segundo semestre foi a de construir os blocos especificados na arquitetura um de cada vez, ou seja, fazendo primeiro os blocos independentes, e deixando para o final os que dependiam dos blocos feitos anteriormente.

O fluxo de desenvolvimento, com as dependências (setas) evidenciadas, pode ser observado na figura 5.1, na qual também se destaca o principal responsável pela implementação cada módulo (embora, deixemos claro, procurássemos fazer as atividades em conjunto, em especial às relacionadas a especificação e tomadas de decisão).

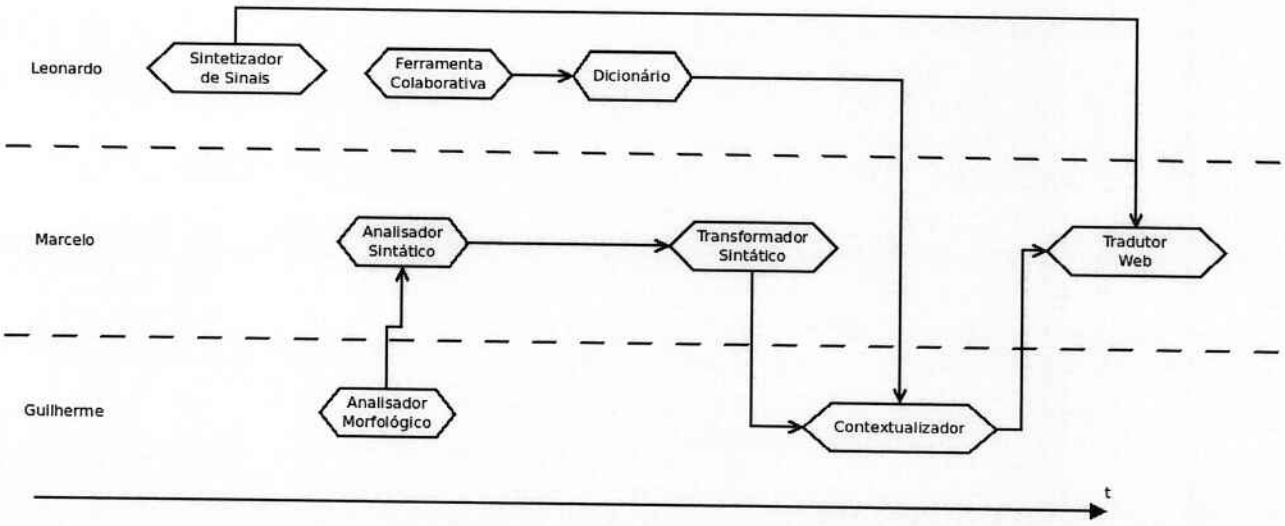

Figura 5.1: Fluxo de desenvolvimento

Obs: a posição horizontal de cada bloco na figura dá ideia de quando aquele bloco começou a ser implementado, porém na figura não está retratado quando o bloco foi finalizado, ou seja, o tempo de desenvolvimento gasto em cada um deles.

No entanto percebemos que uma metodologia mais iterativa poderia ter-nos valido mais, pois os blocos básicos apresentavam requisitos exigidos pelos blocos superiores que os usariam e sem começar a fazer antes os blocos superiores por vezes era difícil enxergar esses requisitos; assim um processo em que se fizessem aos poucos versões simplificadas de cada bloco poderia trazer um benefício maior para os blocos superiores e evitar retrabalho nos blocos inferiores.

Esse fluxo também trouxe o inconveniente de se super trabalhar alguns dos componentes inferiores, enquanto que os últimos acabaram sofrendo penalização por possuírem

menos tempo de desenvolvimento, sendo esses últimos justamente os módulos responsáveis pelo processo de tradução, onde se aplica o estudo do domínio da LIBRAS.

Deixar esses módulos para o final não trouxe apenas o problema da escassez de tempo, mas também o fato de que caso os tivéssemos iniciados antecipadamente, poderíamos aproveitar melhor a ajuda de especialistas em LIBRAS que tínhamos a disposição (dúvidas respondidas sobre o domínio do problema mais no final do projeto são mais difícil de serem aproveitadas e incorporadas ao adequadamente ao projeto).

Sobre a metodologia de desenvolvimento cabe ainda comentar sobre o uso de tecnologias que auxiliaram o trabalho em grupo, notadamente o *wiki* para construção colaborativa de documentos, o *svn* para a construção colaborativa de código e a lista de e-mails para organização e arquivação de discussões; pode-se afirmar que o uso dessas tecnologias fazem uma grande diferença em termos de praticidade, evitando protocolos morosos e propensos a erros na relação entre os membros da equipe.

5.2 Analisador Morfológico

A solução elaborada para o analisador morfológico subsiste no uso da biblioteca JSpell. Como o presente projeto é desenvolvido em tecnologia Java e o módulo JSpell, disponível apenas via código em linguagem C, o módulo implementador da interface analisador morfológico–sintático deve também lidar com módulos compilados em linguagem nativa (de máquina).

Para tanto, lança-se mão da Java Nat Interface (JNI)[1]. Essa biblioteca permite que classes Java acessem códigos compilados para código de máquina. Porque as informações dos tipos primários de dados podem ser armazenadas distintamente na Java Virtual Machine e na estrutura nativa, faz-se necessário também que o código nativo utilize funções de conversão java–código nativo.

Essas funções estão disponíveis ao código C através do cabeçalho *jni.h*, disponível no diretório *include* da JDK (*Java Development Kit*). A fim de não se modificar o código original do JSpell, a integração com as funções de conversão citadas apenas podem ser utilizadas via outro módulo C. A este cabe a conversão de representação de tipos e o uso da biblioteca JSpell. Deve-se notar ainda que há, propriamente, certas idiossincrasias quanto

[1] JNI. Disponível em `http://download.oracle.com/javase/1.4.2/docs/guide/jni/spec/functions.html`. Acesso em 06/12/2010

à nomenclatura das funções e quando aos tipos a serem exportadas pelo módulo nativo ou passados a este como parâmetro. Para automatização dessa geração bem definida de nomes e parâmetros há um programa também disponível pela JDK, o *javah*.

Esse programa recebe como parâmetro o caminho duma classe Java já compilada e identifica as funções dessa classe que devem ser providas por um módulo nativo, gerando o cabeçalho adequado para o desenvolvedor C. O procedimento é exemplificado abaixo:

1. Declaração, na classe Java, das funções naturais previstas:

```
private native String[]
      consultWordPossibilities(String consultanda);
```

2. Compilação da classe que usará a biblioteca nativa:

```
javac br/usp/language/morph/AnalisadorMorfologico.java
```

3. Geração automática do cabeçalho nativo necessário:

```
javah -d jamori/ br.usp.language.morph.AnalisadorMorfologico
```

Nota: acima, `jamori/` é o diretório escolhido como destino do cabeçalho a se gerar.

Nome do cabeçalho gerado: `br_usp_language_morph_AnalisadorMorfologico.`

4. Inclusão, no arquivo *jamori.c*, do cabeçalho para uso da biblioteca JSpell e do cabeçalho recém gerado:

```
#include <jslib.h>
#include "br_usp_language_morph_AnalisadorMorfologico.h"
```

5.3 Analisador sintático

Autômatos

Foi criado primeiramente um modelo para autômatos para autômatos finitos. Nesse modelo, os símbolos de entrada devem ser sempre representados "Strings".

Esse modelo foi incrementado para conter também ações (trechos de código) relacionadas às transições, i.e., o autômato vira um trandutor. A ação foi implementada através

de uma interface (Action) com um único método (doAction()), e portanto, para adicionar uma ação ao autômato basta criar uma classe que implementa a interface.

Para utilizarmos autômatos para linguagens livres de contexto, o modelo foi estendido para um autômato de Pilha Estruturado (APE), cuja descrição pode ser encontrada na obra de Ramos, Neto e Vega (2009).

Assim sendo, foi adicionado o conceito de chamadas de submáquinas e de empilhamento de máquinas. Na definição original do APE determinístico, quando um estado possui uma chamada de submaquina, não deve possuir mais nenhuma outra transição, sendo portanto a chamada a transição obrigatória quando naquele estado. Para facilitar a construção da gramática e permitir maior flexibilidade, fizemos as seguintes modificações:

1. É permitido que um estado possua transições "normais"e também chamadas de submáquina;
2. É permitido que um estado possua mais de uma chamada de submáquina
3. O estado pode ainda possuir transições epsilon

Essas modificações trazem ao autômato algum não-determinismo nas suas decisões. Para resolver os itens números 1 e 3, foi definida uma ordem de precedência implícita para as transições. Primeiro, a prioridade é para as chamadas de submáquina. Em seguida, as transições normais e se nenhuma for válida, a transição epsilon é escolhida.

Para o item 2 foi implementado uma espécie de *look-ahead* de estados, pesquisa-se em cada sub-maquina se ela possui uma transição para o símbolo de entrada atual. Se possuir, este é o escolhido. Esse *look-ahead* é propagado se o estado inicial da máquina chamada possuir outras chamadas de sub-maquina ou transições epsilon.

Vejamos como exemplo um excerto da gramática que utilizamos para o português, baseada em Luft (2002). Esse pedaço mostra as derivações de um sintagma substantivo (SS).

```
* SS -> PrAdj + NOME
* SS -> NOME
* SS -> *ppes*
* PrAdj -> *art*
* PrAdj -> *pdem*
```

```
* PrAdj -> *pind*
* PrAdj -> *ppos*
* NOME -> *nc*
* NOME -> *np*
```

A partir dessa gramática, foi gerado o autômato de pilha estruturado da figura 5.2. Vemos que ele tem uma relação direta com a gramática, é composto por 3 submáquinas: SS (a inicial), PrAdj e NOME, uma para cada não-terminal e cada transição representa um elemento das derivações.

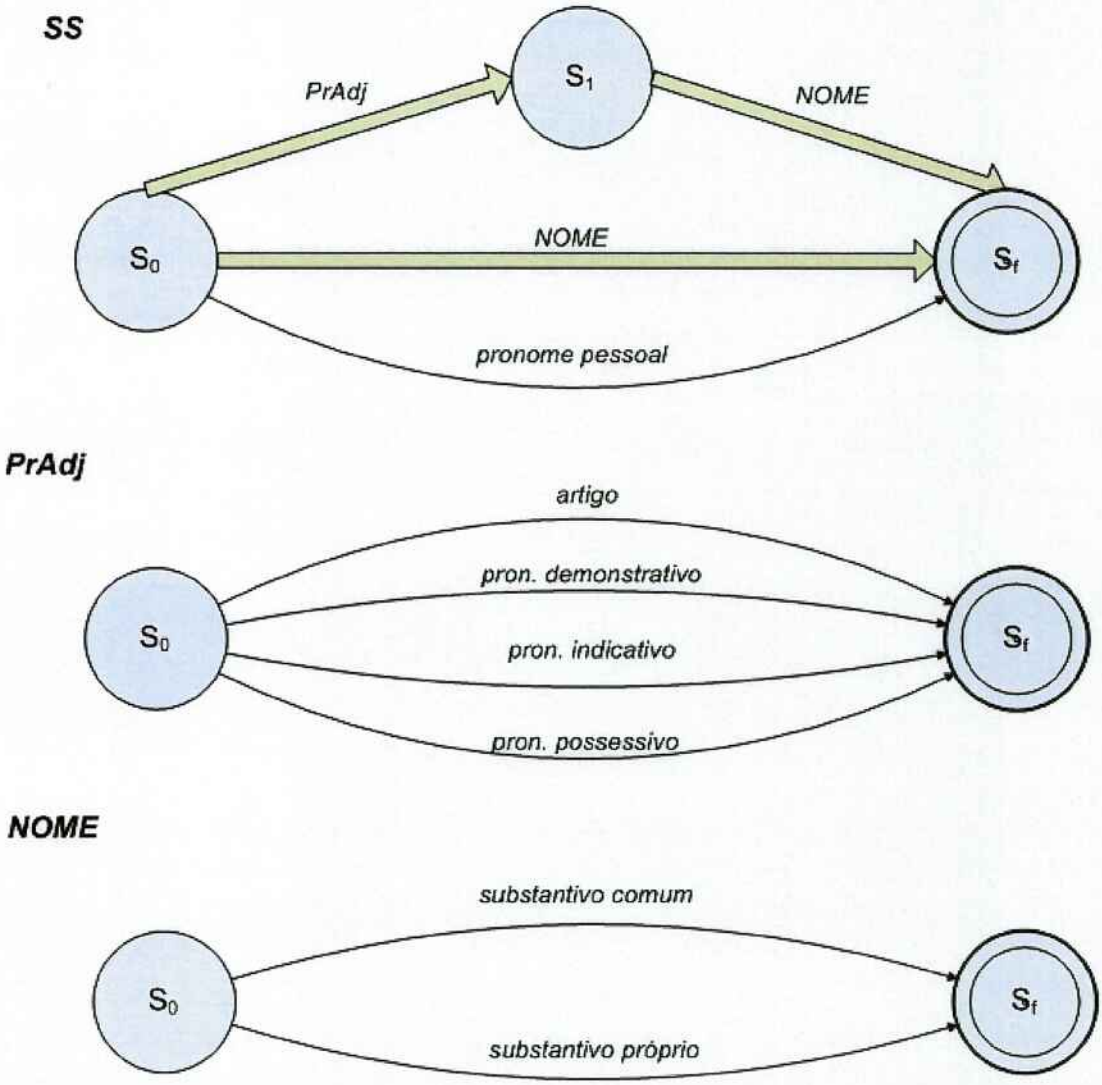

Figura 5.2: Autômatos exemplo

Consideremos então a seguinte cadeia de entrada: "art nc".

O estado inicial é S0 do SS. Vemos que ele possui 3 transições, sendo 2 chamadas de submáquina e 1 transição "normal". Conforme definido, a precedência é para as chamadas de submáquina. Mas existem 2 chamadas, então o autômato deve fazer um "look-ahead" em cada submaquina para ver se elas aceitariam o primeiro token "art". Vendo o estado inicial de NOME, vamos que ele não possui transição válida para "art", por outro lado PrAdj possui, portanto essa é a chamada escolhida. O estado de retorno de SS (S1) é empilhado e o autômato transita para a maquina PrAdj e consome o símbolo "art", transitando para o estado final.

O próximo símbolo da cadeia de entrada é "nc (substantivo comum)". Como a máquina já se encontra num estado de aceitação, o estado presente na pilha é desempilhado e a máquina passa para o estado S1 de SS. S1 só possui uma transição, então NOME é chamada, "nc" é consumido, retorna-se para SS e a máquina para no estado final de aceitação.

Funcionamento

O analisador sintático então lê uma gramática definida pelo usuário e contrói um APE para ela. Ou seja, o usuário define um conjunto de regras gramaticais e o programa gera um reconhecedor da linguagem por elas descrita. Nas ações atribuídas para cada transição do autômato foram colocadas as operações de criação da árvore sintática. Basicamente, para cada novo não-terminal definido na gramática, um novo autômato finito é criado. E cada elemento gerado pela regra forma um novo estado neste mesmo autômato. Quando uma regra gramatical possui um não-terminal gerado, é colocada uma chamada de submáquina no autômato. Desse modo, o autômato de pilha estruturado é construído.

Quando o analisador opera, ele recebe um *token* proveniente do analisador morfológico e usa a categoria (classe gramatical) dele para alimentar o autômato. Se ao final de todos os tokens de entrada o autômato se encontrar num estado de aceitação, a análise sintática ocorreu com sucesso.

Quando há o caso do analisador morfológico devolver mais de uma possibilidade, o analisador sintático primeiro tenta com a primeira. Em caso de insucesso de consumo, a opção seguinte é a considerada e caso nenhuma opção seja válida, então é verificado se houve alguma decisão anterior que tinha mais opções e nesse caso, essa decisão é refeita. Ou seja, é feita uma busca cega em profundidade com retrocesso (*backtracking*) até que se encontre uma árvore sintática válida.

A biblioteca para descrição e manipulação de árvores foi desenvolvida pelo grupo também, por não encontrarmos solução satisfatória no "mercado".

5.4 Transformador sintático

Nessa seção detalhes da implementação do módulo do transformador sintático, especificado na seção 4.4.4, serão apresentados. Como esse é o módulo que altera a estrutura sintática, ele principalmente realiza operações de árvores, utilizando extensivamente a biblioteca de árvores desenvolvida pelo grupo.

Um conjunto de regras foi definido para transformar a árvore do português na árvore da LIBRAS com maior grau de fidelidade possível. Nesse presente trabalho as regras estão fixas por não termos chegado a uma conclusão satisfatória sobre qual seria o melhor formalismo para definir tais regras. Mas a ideia para trabalhos futuros, conforme descrito nos requisitos, é que essas regras possam ser definidas e modificadas dinamicamente, sem necessidade de recompilação do programa, com o intuito de ser possível sempre refinar a tradução, papel que deve ser exercido principalmente por linguistas das duas línguas em questão.

Dentre as funções listadas na especificação, neste trabalho não foram todas implementadas, deixando as demais para trabalhos futuros. Estão implementadas as funções de:

- Cortar elementos irrelevantes
- Inserir advérbios
- Concordância verbal de pessoa e número
- Escolha de ordem da sentença em alguns casos

Essas funções correspondem a uma implementação até o nível 3 da graduação proposta por esse trabalho na figura 2.3.

Para implementar essas funções, os métodos usados nas árvores foram:

- Cortar elementos
- Busca por folha mais próxima

- Junção de elementos

A primeira função do transformador é simplesmente cortar elementos que não são relevantes para a língua-alvo. Tais elementos não carregam informações significativas para o entendimento da oração em LIBRAS ou não fazem parte da estrutura dela. É o caso de preposições ou artigos, que não existem em LIBRAS . Quando o programa encontra um nó da árvore que carrega um token com essas características, esse nó simplesmente é desligado do restante da árvore.

Outra função muito importante é a busca na árvore pela folha "mais próxima"de uma outra folha. Por indefinições a respeito de qual seria a correta definição de distância nesse caso, por ora estamos utilizando uma heurística para resolver isso. A heurística consiste em simplesmente listar todas as folhas em ordem e retornar a mais próxima dentro dessa lista ordenada. Esse processo não é uma funcionalidade final do transformador sintático, mas sim um método de suporte a diversas funções.

Uma das funções que utilizam a busca de folha mais próxima como suporte é a junção de nós. Em LIBRAS, temos algumas classes de palavras que não possuem seu próprio sinal, mas elas modificam o sinal relacionado. Por exemplo, advérbios de modo como "rapidamente"não são falados em LIBRAS , mas modificam o jeito de expressar o sinal do verbo, aumentado a velocidade com que são feitos. Assim sendo, para que o verbo possua essa característica, as propriedades do nó "advérbio"são inclusas nas do nó "verbo"e o nó do advérbio é cortado, assim o nó do verbo conterá todas as informações necessárias para que o módulo contextualizador faça a modificação no sinal do verbo.

Sempre que o transformador encontra um nó que carrega um verbo, ele busca na árvore quem são o sujeito e o objeto dele para incorporar essas informações ao token do verbo, também para o contextualizador poder fazer a flexão verbal.

Então basicamente o transformador sintático varre a árvore nó por nó e dependendo do tipo de cada nó ele realiza ações que modificam a árvore. Esse processo está resumido na figura 5.3.

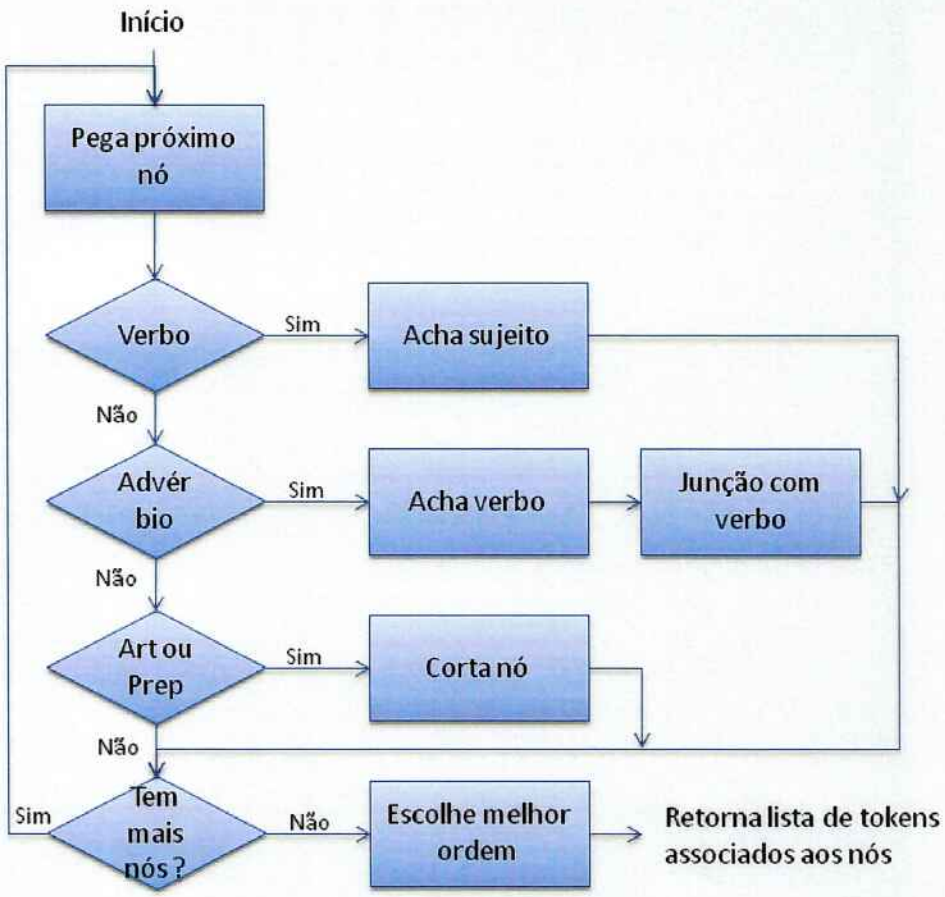

Figura 5.3: Funcionamento do transformador sintático

5.5 Sintetizador de sinais 3D

Este módulo, denominado *Virtual Jonah* foi implementado em parceria com a aluna de design da FAU Heloisa Yoshioka em trabalho desenvolvido em contexto da disciplina PCS-2520 (Tecnologia de Computação Gráfica), ministrada pelo professor Ricardo Nakamura.

As tecnologias utilizadas acabaram sendo a API Processing, por já ser utilizada na disciplina e o 3DStudio para a modelagem, por ser uma ferramenta já dominada pela estudante de design, que criou os modelos.

O nome do trabalho (Virtual Jonah) foi inspirado no filme And Your Name Is Jonah, de 1979, que narra as dificuldades da vida de um menino surdo e sua família, incapazes de se comunicar entre sí, que apenas conseguem superar esse obstáculo quando o pai aprende Língua de Sinais.

A modelagem foi feita no 3D Studio Max, apesar de os modelos terem sido salvos no formato OBJ, que é um formato de modelos tridimensionais em formato texto, o que permite que ele seja facilmente interpretado por praticamente qualquer programa de modelagem.

No programa foram feitos dois modelos: o da mão, que serviu para gerar os .obj tanto da mão direita quanto da esquerda (eles são apenas espelhados) e o da cabeça. A principal diferença deles foi o fato de o modelo da mão ter bones e não ter textura e o da cabeça ter textura e não ter bones.

Mesmo que o formato .obj não suporte bones, o esqueleto da mão foi utilizado para definir mais facilmente as diferentes conformações dela, sendo que cada uma dessas posições é exportada com um .obj diferente.

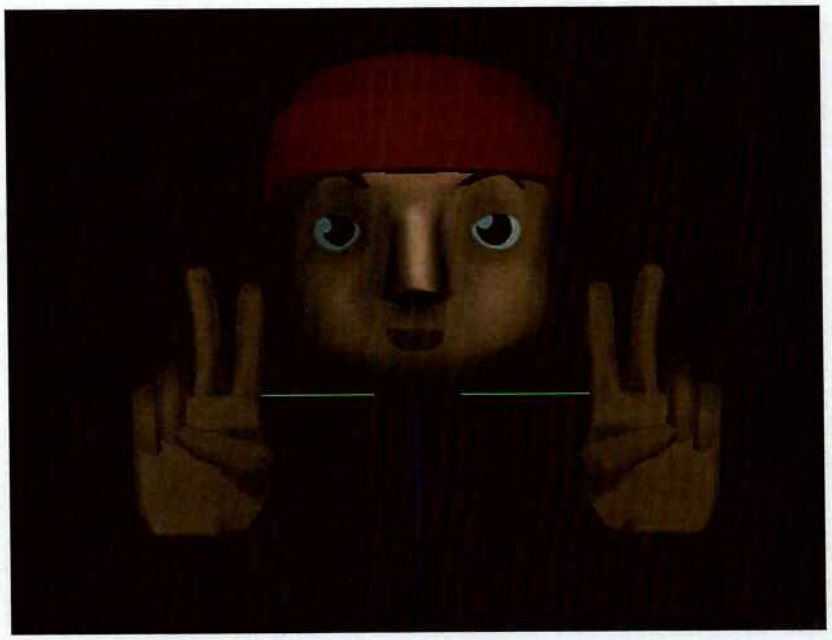

Figura 5.4: Saída gráfica com o Virtual Jonah

A animação foi feita utilizando a biblioteca OBJ Loader [2], responsável por carregar e renderizar os modelos no ambiente do Processing.

Com a classe OBJModel da OBJ Loader pudemos carregar os modelos salvos em obj e manipular suas posições e orientações através dos comandos de deslocamento e de rotacionamento do Processing para que as mãos ficassem de acordo com o especificado no arquivo XML de entrada, que corresponde à discrição dos sinais discutida na seção Modelo de Dados.

[2] OBJ Loader. Disponível em <http://code.google.com/p/saitoobjloader/>. Acesso em 06/12/2010

Para que a mão pudesse mudar de configuração, a tática adotada foi herdar a classe OBJModel pela nossa classe AnimObj para criar novos métodos que pudessem determinar um outro modelo obj e executar o deslocamento de cada um dos vértices do obj original, através de interpolação linear, até que o modelo acabasse ficando idêntico ao novo obj especificado.

Para que essa técnica dê certo é importante que os vértices correspondentes listados no obj estejam descritos na mesma ordem em ambos os modelos.

Por considerarmos a classe AnimObj como algo útil, que inclusive poderia ser usado pelos próximos alunos da disciplina, acabamos por publicar essa classe de forma livre [3]independentemente do resto do projeto.

A criação da classe AnimObj e a relevância de sua disponibilização está numa interessante peculiaridade do Virtual Jonah: acontece que normalmente em jogos quando um personagem precisa realizar um movimento localizado, como abrir e fechar sua mão, este é um movimento pré-definido e usado com frequência, logo a solução nestes casos é, em tempo de modelagem, gerar animações dos movimentos da mão se abrindo ou fechando e carregar essa animações em tempo de execução; no nosso caso, as possibilidades de transição, de uma configuração de mão para outra, são tantas que tal técnica se torna inviável, e por isso é preciso a modificação do modelo em tempo de execução, seja pela manipulação dos *bones*, seja pela manipulação da própria malha (*mesh*), que é como é feito atualmente.

5.6 Ferramenta colaborativa

Inicialmente o WikiLibras começou a ser feito com base nas APIs de servlets Java EE e fazendo o acesso aos dados com a API JDBC, que é o que há de mais básico que há no mundo Java para aplicações web (básico no sentido de ser a tecnologia que serve de base para as outras mais avançadas).

Embora a inserção de sinal trate apenas de uma única entidade, essa mesma entidade é formada por uma composição de outras entidades (símbolos, mãos, movimentos e faces). Essa composição é na verdade um tanto quanto complexa, e fazer um único formulário para preencher esses dados, com estas tecnologias mais básicas, mostrou vá-

[3] AnimObj. Disponível em `http://code.google.com/p/objanim/`. Acesso em 06/12/2010

rios desafios (quebrar o formulário em várias telas também apresentava outros problemas nestas condições).

Além de inúmeros problemas que surgiam devido a detalhes da tecnologia, também havia o fato de que o uso de apenas essas ferramentas levava a uma ineficiência devido a grande repetição de código similares e burocráticos que era preciso escrever (*boillerplate code*).

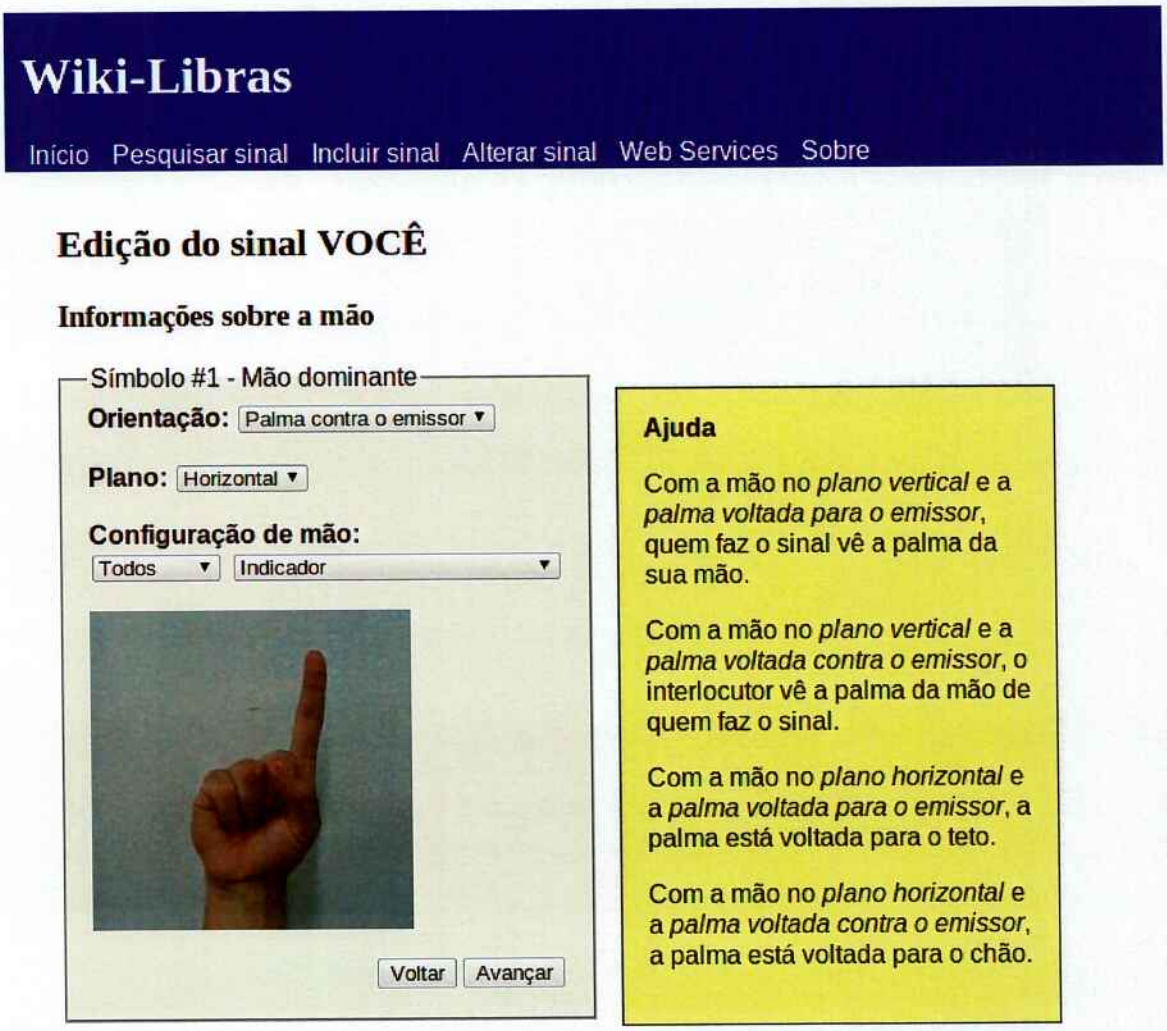

Figura 5.5: Tela de edição de sinal no WikiLibras

Ferramentas

Devido às questões apresentadas procuramos utilizar frameworks que facilitassem o desenvolvimento dessa etapa do projeto, sendo o escolhido o VRaptor, que além de auxiliar

no fluxo de navegação entre páginas e *binds* entre parâmetros de formulários e objetos, também já trazia consigo o Hibernate, que realiza o mapeamento objeto-relacional para fazer conversões entre objetos e tabelas de banco de dados.

Assim, utilizando o Hibernate criou-se uma pequena classe que gerava automaticamente o banco de dados com base nas classes que modelavam os dados a serem salvos.

```
public class GeraBanco {

    public static void main(String[] args) {
        Configuration conf = new AnnotationConfiguration();
        conf.configure();
        SchemaExport se = new SchemaExport(conf);
        se.create(true, true);
    }
}
```

Dado que o formulário é dividido em várias telas, criou-se também uma classe para armazenar os dados que estão sendo editados na sessão, a SignEditor, que possui os seguintes atributos:

```
// sinal que está sendo criado/editado
private Sign sign;
// índice do símbolo sendo editado como é mostrado para o usuári
// na lista é acessado como signIndex-1
private int signIndex = 1;
private HandSide side;
private boolean twoHands = true;
//indica se estamos editando um novo sinal
private boolean newSign;
```

Para a interface usou-se páginas JSP, mas sem a escrita de *scriptlets* (código Java embutido diretamente no HTML), em vez disso usou-se a *expression language*, que consiste na escrita de variáveis definidas pelo servlet nas páginas HTML e no uso de bibliotecas de tags, a JSTL, para criar condições e laços na escrita de código HTML.

O uso dessas tags de bibliotecas poderia ser substituída pelo uso de tecnologias mais avançadas, como a JSF (Java Server Faces), que permite ao desenvolvedor escrever tags

como se fossem elementos HTML mais sofisticados, no entanto nesse caso permanecemos com a tecnologia mais básica.

Como parte das tecnologias mais básica do mundo web também utilizamos o CSS para a formatação do estilo de apresentação do site, isto é, para definir seu layout e o Java Script para criar ações dinâmicas nos formulários quando necessário, a exemplificar:

- adição de mais caixas de texto para entrada de palavras correspondentes;
- alterar a foto de exibição ao se selecionar uma configuração de mão diferente;
- restringir a lista de configurações de mão pela opção selecionada na lista de grupos de configurações de mão;

sendo que no último caso houve até mesmo uma interessante interação entre o código Java Script e as bibliotecas de tag JSTL.

Fotos

Um aspecto bem importante para a facilidade de operação da interface foi a utilização de imagens representando cada uma das configurações de mão selecionadas.

Primeiro pesquisamos para saber se poderíamos usar algum conjunto de imagens prontas, mas não encontramos nadas disponível e prático de usar a mão, então acabamos optando por nós mesmo tirarmos fotos para representar as configurações.

Ao total foram 69 fotos, uma para cada configuração de mão, agrupadas em 10 grupos.

Codificação de caracteres

Por vezes, em projetos como esse, acabamos por perder muito tempo em pequenos problemas que não eram previstos, o que em geral contribui para não precisão de estimativas na construção de sistemas de software.

Um desses casos que enfrentamos foi a dificuldade de lidar com a correta codificação de caracteres em determinados pontos do sistema. No caso do WikiLibras isso ocorreu quando devíamos gerar links que possuíssem como parâmetro o nome do sinal, que poderiam ter acentos.

Dessa forma, para permitir acentos nos nomes dos sinais, descobrimos que era necessário gerar as URLs com o comando `URLEncoder.encode(link, "ISO-8859-1")`.

Ambiente de execução

Para deixar o WikiLibras disponível na Internet, a fim de contar com a colaboração de outras pessoas para criar o vocabulário a ser usado em nosso tradutor, conseguimos o apoio do projeto Poli Cidadã, que permitiu-nos hospedar o WikiLibras sem seu servidor.

No servidor disponibilizado havia um servidor Apache instalado, mas não o container Tomcat, que é responsável por executar o código Java e gerar as páginas HTML a partir desse processamento.

Uma vez cedido o servidor, a responsabilidade de efetuar toda a configuração necessária era nossa.

Após instalar o Tomcat (que é basicamente descompactar o Tomcat em qualquer lugar) o primeiro foi preciso encontrar uma porta de conexão disponível; mas mesmo após encontrarmos essa porta, vimos que só podíamos acessar a aplicação instalada no Tomcat pelo localhost (aliás, como o acesso ao servidor era feito via *ssh*, esses testes eram realizados com o navegador modo-texto *lynx*).

Dada as restrições de seguranças de configurações de porta, decidimos que o jeito certo de tornar a aplicação disponível seria fazer a ponte entre a requisição e o Tomcat através do Apache, ou seja, o Apache recebe a requisição e encaminha para o Tomcat executar o processamento; essa técnica é na verdade o modo correto de se fazer isso mesmo sem considerar as restrições das portas, pois desejamos que o usuário acesse a página *http://www.policidada.poli.usp.br/wikilibras*, e não algo como *http://www.policidada.poli.usp.br:8081/wikilibras*.

A primeira forma encontrada para fazer isso foi baseada em páginas de documentação do próprio Tomcat [4].

O Tomcat quando iniciado, já deixa uma porta aberta para comunicações pelo protocolo *AJP*, que é uma versão otimizada do protocolo HTTP para a comunicação do Tomcat com outros web servers. A ideia então era fazer com que o Apache se conectasse a esse serviço AJP do Tomcat.

[4] The Apache Tomcat Connector - Generic HowTo: <http://tomcat.apache.org/connectors-doc/generic_howto/workers.html>. Acesso em 06/12/2010

Seguindo o site, baixamos o módulo do Apache para que isso funcionasse: o *mod_jk*; também criamos um arquivo de configuração (workers.properties) e adicionamos algumas linhas ao arquivo de configuração do Apache (apache2.conf).

Esse procedimento funcionou com sucesso em uma de nossas máquinas pessoais, mas no servidor não funcionou. Nem o acesso em localhost à aplicação pelo Tomcat funcionava.

Depois de gastar algum tempo investigando a situação, não obtivemos sucesso e recorremos a um fórum para pedir ajuda (utilizamos o GUJ, fórum brasileiro para desenvolvedores Java).

O post se encontra em: http://www.guj.com.br/posts/list/221913.java

Obtivemos uma resposta que nos indicava para utilizar outro módulo do apache para fazer isso: o *mod_proxy_ajp* (que por sua vez depende do *mod_proxy*).

Seguimos então o procedimento que foi muito mais simples: uma vez que os módulos necessários já se encontravam instalados no Apache, bastou acrescentar as seguintes linhas ao apache2.conf:

```
# Enable Tomcat to work with Apache
ProxyPass  /WikiLibras ajp://localhost:8009/WikiLibras
LoadModule proxy_module
 /usr/lib/apache2/modules/mod_proxy.so
LoadModule proxy_ajp_module
 /usr/lib/apache2/modules/mod_proxy_ajp.so
```

Obs: 8009 é a porta utilizada pelo Tomcat para a comunicação AJP.

5.7 Dicionário

A camada de WebService feita no WikiLibras, responsável por disponibilizar a base de conhecimento construída, consiste no lado servidor do dicionário.

Para realizar a implementação deste componente o primeiro passo foi estudar os conceitos de WebServices, que podemos resumir rapidamente aqui:

- XML (Extensible Markup Language): linguagem de descrição de dados flexível em formato texto padronizada pela W3C (World Wide Web Consortium);
- SOAP (simple object access protocol): *envelope* com a estrutura da mensagem enviada pelo WebService;
- WSDL (web services description language): arquivo XML que descreve a interface pública do WebService para que outros softwares possam utilizar seus serviços; a descrição contem as funções disponíveis e os tipos de dados envolvidos;
- UDDI (universal description, discovery and integration): repositório de WebServices para que serviços possam ser encontrados de forma automática (não utilizado neste trabalho).

Após entendido os conceitos, era preciso escolher qual framework usar. Esse passo é particularmente complexo devido a variedade de informações e a rápida evolução das ferramentas, de forma que o que usamos há um ano atrás para fazer web services já não é a melhor ferramenta no momento.

O framework escolhido foi o *JAX-WS* (Java API for XML Web Services), que é a solução aberta mais moderna no mundo Java para WebServices, sendo que os mesmos podem ser facilmente criados através do uso de *anotações* nos métodos que devem ser publicados.

Dessa forma os passos necessários para a criação do WebService foram os seguintes:

1 - Criação da interface

Neste passo o importante é o uso correto das anotações (*tokens* precedidos pelo "@").

```
import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
public interface ISignDictionaryWS {

    @WebMethod
    public Sign signByName(String name);
```

```
    @WebMethod
    public List<Sign> nearSigns(String name);

    @WebMethod
    public List<Sign> translate(String word);

    @WebMethod
    public Sign simpleTranslate(String word);
}
```

2 - Implementação da classe

Feito com a classe SignDictionaryWS (que, obviamente, implementa ISignDictionary).

A princípio implementar cada um desses métodos é fácil dado que temos um objeto *signDao*, que é responsável pelo acesso ao banco de dados, já fazendo o mapeamento objeto-relacional utilizando o Hibernate.

A única parte mais complicada deste código é o procedimento para obter o DAO através de uma sessão do Hibernate, que é feito pelo seguinte trecho código.

```
// obtendo o dao que acessa o banco de dados pelo Hibernate
SessionFactory sessions =
   new AnnotationConfiguration().configure().buildSessionFactory();
Session session = sessions.openSession();
SignDaoFactory<Sign> factory =
   new SignDaoFactory<Sign>(session);
this.dao = factory.getSignDao();
```

Obs: este procedimento não é necessário na maior parte do WikiLibras, pois a sessão do Hibernate é fornecida "pronta"pelo framework VRaptor.

3 - Arquivos de configuração

Foi necessário criar o arquivo WEB-INF/sun-jaxws.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<endpoints xmlns='
    http://java.sun.com/xml/ns/jax-ws/ri/runtime' version='2.0'>
    <endpoint
     name='signws'
     implementation='br.usp.wikilibras.ws.SignDictionaryWS'
     url-pattern='/ws/dictionary'
    />
</endpoints>
```

Também foi necessário adicionar as seguintes linhas ao WEB-INF/web.xml:

```
<listener>
  <listener-class>
    com.sun.xml.ws.transport.http.servlet.WSServletContextListener
  </listener-class>
</listener>
  <servlet>
    <description>Dicionário JAXWS</description>
<display-name>signws</display-name>
    <servlet-name>signws</servlet-name>
<servlet-class>
  com.sun.xml.ws.transport.http.servlet.WSServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
     <servlet-name>signws</servlet-name>
     <url-pattern>/ws/dictionary</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
```

4 - Gerando o WSDL e demais arquivos

Uma vez que os passos anteriores foram seguidos, basta iniciar o Tomcat que o JAX-WS se encarregará de criar o arquivo WSDL.xml e demais classes que forem necessárias para o WebService funcionar (essas classes criadas automaticamente nem aparecem em nossa estrutura de projeto).

Para ver se está tudo funcionando, basta acessar pelo navegador 'http://localhost:8081/wikilibras/ws/dictionary?wsdl', onde 8081 é a porta do Tomcat. Se o arquivo xml aparecer no navegador, o WebService está pronto para uso!

Cliente WebService

Enquanto o WebService foi implementado no projeto WikiLibras, o lado cliente foi implementado no projeto PoliLibras, que é onde está o tradutor (do ponto de vista da organização no Eclipse, são projetos totalmente distintos, sem dependências).

Para criar o cliente, o primeiro passo é executar o seguinte comando (o prompt deve se localizar no diretório raiz do projeto):

```
$> wsimport -d src -s src
     http://localhost:8081/wikilibras/ws/dictionary?wsdl
```

Esse comando irá ler a descrição do WSDL e criar as classes necessárias para que os métodos do WebService possam ser acessados do projeto cliente. Dentre as classes geradas, está a interface ISignDictionaryWS (note que a implementação não aparece para o cliente, pois a ideia de WebService é a invocação da implementação no servidor).

Todas as classes e interfaces geradas apareceram num novo pacote dentro do projeto (wikilibras.ws).

A forma de se utilizar os métodos do dicionário então é:

```
SignDictionaryWSService service = new SignDictionaryWSService();
ISignDictionaryWS dictionary = service.getSignDictionaryWSPort();
```

Dessa forma utilizar o objeto dictionary criado será a mesma coisa que usar um objeto SignDictionaryWS.

Mas criou-se ainda uma classe WikiLibrasDictionary (implementando uma interface SignDictionary, idêntica à interface ISignDictionary do servidor) no projeto cliente, que é responsável por executar as linhas acima e redirecionar as chamadas para o WebService. O propósito principal é fazer com que as outras partes do projeto (cliente) que queiram utilizar o dicionário, utilizem apenas uma classe simples, sem precisar saber que estão usando WebServices ou qualquer outra coisa.

Uma facilidade acrescentada foi fazer com que o cliente pudesse facilmente trocar a localização do WebService; isso foi feito para garantir que em caso de falta de conectividade, poderíamos utilizar o WebService em localhost. Isso foi feito criando um arquivo de configuração no projeto (conf/polilibras.conf) que seta o caminho do WSDL a ser utilizado e alterando-se uma das classes geradas automaticamente (a SignDictionaryWSService) para utilizar esse valor em vez do *default hard-coded*.

Um outro bom motivo para criar essa classe de isolamento (WikiLibrasDictionary) foi isolar as classes de domínio originais do projeto das classes do domínio que são repetidas e geradas automaticamente pelo WebService, que acabam parando no pacote wikilibras.ws no projeto do cliente, sendo que essas classes geradas automaticamente perdem a hierarquia de pacotes original das classes do modelo, ou seja, embora "idênticas"as classes retornadas por ISignDictionaryWS no cliente não são as mesmas classes de domínio utilizadas no resto do projeto (que estão no pacote libras.sign).

Dessa forma, WikiLibrasDictionary utiliza a classe ClassConverter para retornar ao usuário (no projeto do tradutor) as classes de domínio do pacote libras.sign, facilitando seu uso.

5.8 Contextualizador

Consoante desenvolvido na especificação, vários aspectos do contextualizador se encontram no estilo *hard-coded*. Por carência de cadastro de sinais ou mesmo da própria compreensão da língua alvo, LIBRAS , não se obteve muita generalização para este módulo. De fato, é o mais dependente da língua alvo, envolvendo entendimento desde a semântica até a morfologia, passando pela sintaxe.

Outras adaptações e ideias de como obtê-las podem ser vistas na sessão sobre trabalhos futuros.

5.9 Organização dos repositórios

Como mencionado, desejamos uma divisão modular do nosso sistema, de forma que os módulos possam ser reaproveitados independentemente em outros sistemas. Mas conforme discutiremos, realizar essa divisão não é tão trivial.

A não trivialidade depende basicamente de conceber dois cenários: o primeiro em que nosso sistema está em constante desenvolvimento, sendo todos os módulo atualizados constantemente em paralelo e um segundo cenário posterior, em que os módulos estão mais estáveis e recebem menos atualizações.

Considerando o cenário estável concebemos a seguinte divisão:

- classes que modelam os sinais
- animobj
- virtual jonah
- wiki-libras
- bosque (biblioteca de árvores)
- jjspell
- analisador sintático
- tradutor enquanto api
- tradutor com interface web

Essa divisão corresponde a diferentes repositórios que permitam um desenvolvimento independente dos módulo.

No entanto há as relações de dependência, a exemplo do analisador sintático, que depende do analisador morfológico (o JJSpel). Em um ambiente estável uma atualização do jjspel não implicaria a automática atualização de seu uso no analisador sintático, seria necessário que no repositório do analisador sintático se decidisse por atualizar a versão utilizada do jjspel.

Esse tipo de procedimento é bem inconveniente no cenário inicial de desenvolvimento, pois quando estamos desenvolvendo a versão inicial que acaba por integrar todos os módulos, é desejável que todos os módulos utilizem a versão mais recente dos módulos dos quais dependem.

Em especial, há uma complexidade especial no módulo do conjunto de classes do domínio, pois além de ser usada em vários módulos, acabou por sofrer uma diferenciação por receber um atributo *id* e *anotações* para permitir que as classes fossem utilizadas pelo Hibernate. Na verdade, nesse caso o ideal seria substituir as anotações por uma descrição do mapeamento objeto-relacional por arquivo XML denominado ORM, uma vez que tais anotações não fazem sentido nos outros módulos.

Considerando então esses aspectos, decidimos por adotar uma estrutura de repositórios durante o desenvolvimento que fosse mais conveniente e pretendemos após a conclusão do projeto enquanto trabalho de conclusão de curso alterar a estrutura dos repositórios para a estrutura já apresentada, condizente com o cenário posterior.

A atual estrutura de repositórios consiste em três divisões:

- Projeto Java que engloba as classes de modelagem dos sinais, o jjspel, o bosque, o analisador sintático, o virtual jonah e o tradutor em si enquanto API, que consiste nos módulos transformador sintático, dicionário e contextualizador;
- Projeto Java EE para o WikiLibras, que repete as classes de domínio, mas com as anotações; neste projeto está basicamente a aplicação web do WikiLibras e seu WebService;
- Projeto Java EE do tradutor, que consiste na interface web do tradutor, que utiliza o Virtual Jonah e o tradutor enquanto API.
- Mais um repositório extra já criado para a classe AnimObj, responsável por criar animações com modelos OBJ.

Capítulo 6

TESTES E AVALIAÇÃO

O tradutor Poli-Libras implementado neste trabalho possui as seguintes restrições quanto ao texto de entrada:

- Períodos simples;
- Orações na voz ativa;
- Orações na ordem direta;
- Sem apostos ou vocativos

6.1 Exemplo

Consideradas as restrições, será mostrado um exemplo detalhado do fluxo de informações na tradução de uma frase.

Texto de entrada: "Eu quero um chocolate"

O analisador morfológico recebe o texto de entrada, consulta seu dicionário de português e atribui as possíveis categorias para cada palavra. A saída gerada é apresentada na tabela 6.1.

Tabela 6.1: Saída do analisador morfológico

	Lexema	**Forma Dicionário**	**Aproximada?**	**Propriedades**
1	Eu	eu	false	cat=ppes, c=n, p=1, n=s
2	quero	querer	false	cat=v, t=p, tr=_, p=1, n=s
3	um	um	false	g=m, cat=art, n=s
3	um	um	false	g=m, cat=card, n=s
3	um	um	false	g=m, cat=pind, n=s
4	chocolate	chocolate	false	g=m, cat=nc, n=s

Observa-se que no verbo "quero"o analisador identificou corretamente que o verbo está conjugado na 1a pessoa (p=1) do singular (n=s) e no presente do indicativo (t=p).

Repara-se também que a palavra "um"retorna três possibilidades diferentes, pois pode ser um artigo, um numeral cardinal ou um pronome indefinido. Caberá ao analisador sintático decidir qual é o correto.

Em seguida o analisador sintático, utilizando o reconhecedor construído a partir da gramática definida (presente no Apêndice I), atuou sobre as classes gramaticais dos tokens e gerou a árvore presente na figura 6.1. As derivações ocorridas foram as seguintes:

1. F $\Rightarrow$ PS
2. PS $\Rightarrow$ OA
3. OA $\Rightarrow$ SS_1 + SV + SAdv
4. SS_1 $\Rightarrow$ *ppes*
5. SV $\Rightarrow$ Aux + *v* + SS + SP
6. Aux $\Rightarrow$ epsilon
7. SS $\Rightarrow$ PrAdj + NOME
8. PrAdj $\Rightarrow$ *art*
9. NOME $\Rightarrow$ *nc*
10. SP $\Rightarrow$ *epsilon*
11. SAdv $\Rightarrow$ *epsilon*

Nesse caso, a primeira tentativa já formou uma árvore válida. Caso não tivesse formado, o analisador substituíria o "um" artigo pela próxima opção.

O transformador sintático então aplica suas regras na árvore. Nesse exemplo, basicamente ele corta o artigo. O pronome "eu" e a conjugação do verbo indicam o sujeito, que é indicado também (suj=1), caso o contextualisador precise. A lista de tokens gerada está representada na tabela 6.2.

Finalmente, o contextualizador faz uma busca pelo sinal de cada palavra no dicionário do WikiLibras. Ele identifica também que "querer"não flexiona com pessoa e número portanto não precisa modificar sua orientação. O contextualizador retorna então a descrição de 3 sinais, aqui representados em XML.

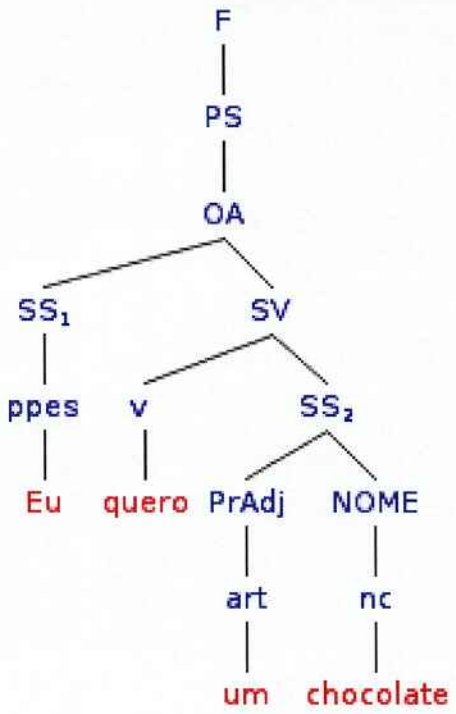

Figura 6.1: Árvore sintática gerada

Tabela 6.2: Lista de tokens saídos do transformador sintático

	Lexema	Forma Dicionário	Propriedades
1	Eu	eu	cat=ppes, c=n, p=1, n=s
2	quero	querer	cat=v, t=p, tr=_, p=1, n=s, suj=1
3	chocolate	chocolate	g=m, cat=nc, n=s

```
<sml>
  <Sign name="eu">
    <words>
      <word id="16" word="eu" />
    </words>
    <symbols>
      <Symbol location="BUSTO" handsInUnity="false"
          contact="BATER" id="13" sequence="1">
        <rightHand side="right" orientation="half"
            shape="INDICADOR" plane="horizontal">
          <movement direction="PARA_FRENTE" frequency="SIMPLES"
              speed="NORMAL" type="RETILINIO" />
```

```
      </rightHand>
    </Symbol>
  </symbols>
</Sign>
<Sign name="querer">
  <words>
    <word id="17" word="querer" />
  </words>
  <symbols>
    <Symbol location="ESTOMAGO" handsInUnity="false"
          id="14" sequence="1">
      <rightHand side="right" orientation="white"
           shape="CURVADOS" plane="horizontal">
        <movement direction="PARA_TRAS" frequency="SIMPLES"
            speed="RAPIDO" type="RETILINIO" />
      </rightHand>
    </Symbol>
  </symbols>
</Sign>
<Sign name="chocolate">
  <words>
    <word id="19" word="chocolate" />
  </words>
  <symbols>
    <Symbol location="LADO_INDICADOR" handsInUnity="false"
         contact="ESFREGAR" id="16" sequence="1">
      <rightHand side="right" orientation="white"
           shape="mao_U" plane="horizontal">
        <movement direction="PARA_TRAS" frequency="REPETIDO"
           speed="RAPIDO" type="SEMICIRCULAR" />
      </rightHand>
      <leftHand side="left" orientation="half" shape="MAO_U"
         plane="horizontal"/>
    </Symbol>
  </symbols>
```

```
  </Sign>
</sml>
```

Essa descrição vai ser lida pelo sintetizador de sinais que os fará de forma animada para o usuário final.

6.2 Testes Morfológicos

Nesta seção serão apresentados alguns testes realizados com o analisador morfológico. Esses testes estão focados na definição da classe gramatical das palavras e verificam, através de posterior julgamento humano, a taxa de acerto do módulo. Estes testes estão listados na tabela 6.3.

Tabela 6.3: Testes realizados com analisador morfológico

Palavra	*Lexema Retornado*	*Categoria(s) retornada(s)*	*Comentários*
carro	carro	nc	OK
cantar	cantar	v	OK
esperto	esperto	adj	OK
você	você	ppes	OK
empreendimento	empreendimento	nc	OK
falou	falar	v	Identificou corretamente a conjugação
doce	doce	adj,nc	Retornou 2 opções - OK
casa	casa	nc,v	Retornou 2 opções - OK
a	a	prep,art,ppes	Retornou todas as opções
projeto	projecto	nc	Diferença com Portugal
raptok	rapto	nc	Palavra inexistente - retornou aproximada

Os testes mostram que o analisador morfológico é bem completo e possui alta taxa de acerto, apenas "errando" quando nos deparamos com diferenças entre português brasileiro e de Portugal, mas ainda assim o analisador sugere uma palavra correta.

Capítulo 7

CONCLUSÕES

Durante a elaboração do presente trabalho, o estudo das características morfológicas da LIBRAS nos auxiliou na formulação de um modelo de codificação digital para sinais de LIBRAS baseado na notação SignWriting , tendo esse sistema de codificação sido utilizado ao longo do projeto; este modelo é de extrema importância, pois além de facilitar a escrita e armazenamento digital de LIBRAS (seja por XML ou bancos de dados), pode criar uma forma comum de comunicação entre diferentes sistemas de processamento computacional que utilizem essa língua.

Tendo em vista a preocupação em obter um sistema de tradução que levasse em conta a coerência sintática da frase gerada em LIBRAS , adotamos uma arquitetura de tradução baseada em sintaxe e por regras, em que se definem as alterações sintáticas da sentença original para a sentença traduzida.

Neste trabalho não nos limitamos ao processo de tradução em si, mas definimos uma arquitetura que incluísse todos os módulos adicionais para uma tradução completa, a saber:

- analisador morfológico;
- analisador sintático;
- dicionário de português para LIBRAS ;
- sintetizador de sinais;
- ferramenta colaborativa para criação e edição de sinais;

Para a criação e integração desses diferentes módulos, mais o tradutor em si, foram necessários estudos e prática em diferentes tecnologias, tais como de computação gráfica, banco de dados, tecnologias web, compiladores, integração Java-C e WebServices, dentre outros.

Tendo os últimos módulos, os de tradução, sido implementados mais ao fim do período do trabalho, não foi possível realizar uma avaliação adequada do processo completo de

tradução, embora pudemos submeter a alguns testes a ferramenta colaborativa, que considerando as limitações do modelo, apresentou bom desempenho, embora fosse desejado a integração com o sintetizador de sinais.

Por fim, conseguiu-se estabelecer uma arquitetura modular para o processo de tradução, estabelecendo módulos bem definidos que, mais alguns do que outros, tem potencial para serem re-aproveitados em novos projetos.

Esperamos assim que de alguma forma, seja direta ou indireta ao incentivar ou facilitar o surgimento de novas aplicações, nosso projeto cumpra um papel positivo que traga algum benefício à comunidade surda.

7.1 Trabalhos Futuros

O projeto Poli-Libras deu início a um legado de sistemas abertos para o desenvolvimento de aplicações voltadas ao público surdo, mas cada um de seus componentes ainda precisam de melhorias, que são listadas a seguir:

Modelo de dados

Inclusão de atributos especificados em (SUTTON, 2003), como expressão corporal (ex: movimento de ombros).

Realizar uma avaliação da capacidade de cobertura do modelo sobre os sinais da LIBRAS .

Investigar mais o vocabulário da LIBRAS e estudar como o modelo pode ser alterado para poder representar uma quantidade maior de sinais.

Expandir o modelo de dados para ter um grupo de configurações de mão que abarque as configurações correspondentes às letras.

Virtual Jonah

Deve evoluir para implementar todos os atributos previstos no modelo de dados, entre eles o movimento da mão, o ponto de contato, movimento dos dedos e expressão facial.

Para implementar os atributos acima também será importante o design do avatar de corpo inteiro, uma vez que alguns sinais utilizam o braço como ponto de locação, sendo que para isso, provavelmente, dever-se-á portar o programa para alguma plataforma que possibilite o controle de *bones* em tempo de execução.

Uma possível melhoria também seria um "controle de prosódia" na execução dos sinais, retirando a linearidade dos movimentos, deixando-os mais naturais.

WikiLibras

Integrar o Virtual Jonah ao WikiLibras assim que o primeiro apresentar uma implementação mais ampla dos atributos previstos para o sinal.

Adequar a interface às diretrizes de construção de web sites para o público surdo.

Analisador Morfológico

Dada a base de dados (dicionário), o analisador morfológico obtido age como o esperado: devolve uma estrutura de dados contendo informações sobre cada interpretação morfológica contida na base de dados.

Infelizmente, a base de dados padrão que ora acompanha a publicação do JSpell não é completa, faltando várias palavras contidas no VOLP [1]. É verdade que muitas dessas palavras faltantes são de uso muito especializado ou raras, mas há aquelas mais comuns, como "ímã", que não são suportadas. Ainda, o dicionário padrão não contém variações corretas brasileiras do português. Tome-se como exemplo o substantivo "projeto"; essa palavra não consta na base padrão, mas a vertente do português europeu, "projecto", lá figura. Mais fatídico ainda é o fato de, pelo novo acordo ortográfico, segundo interpretação da Academia Brasileira de Letras, "projecto"não mais ser grafia válida. O caso do 'c' mudo é o mais frequente.

O problema de inserção de novas palavras pode ser resolvido através do uso da função *insert_word* própria do JSpell; contudo, as operações de remoção ou atualização de palavra diretas não são suportadas diretamente, donde vem que, para efetuá-las, é necessário se criar um novo dicionário e povoá-lo sem a palavra a ser removida e/ou com as atualizações de palavras previamente existentes.

[1] Vocabulário Ortográfico da Língua Portuguesa. Disponível em <http://www.academia.org.br/abl/cgi/cgilua.exe/sys/start.htm?sid=23>. Acesso em 06/12/2010

Analisador sintático

Existe muito espaço para evolução no analisador sintático. Em primeiro lugar, a atual implementação só reconhece orações na ordem direta. Apesar do nosso analisador poder aceitar qualquer gramática livre de contexto, esta deve ser determinística e portanto dificilmente essa questão seria resolvida na gramática (assim como a gramática por nós utilizada, que gera as orações na ordem direta apenas), necessitando de recursos externos para tal. Uma opção seria o uso de técnicas adaptativas para realizar a inversão de certos termos nas regras, como vem sendo desenvolvido em (CERQUEIRA et al., 2010).

Outra aspecto que deve ser trabalhado futuramente é o reconhecimento de períodos compostos, i.e., frases com mais de uma oração, principalmente no caso de orações subordinadas. Para isso são necessários algumas modificações na gramática para que aceitem que um determinado sintagma (substantivo, por exemplo) possa se transformar numa oração completa. Além disso, a análise terá que ser mais profunda para que seja reconhecido qual é o nome que a conjunção que une as orações está substituindo, ou qual é o sujeito da 2ª oração, em geral oculto pois foi explicitado na 1ª oração. Por exemplo, na frase: "Aquele é o menino que venceu", a oração "que venceu" é subordinada adjetiva restritiva pois confere uma qualidade ao substantivo "menino", sendo que a conjunção "que" refere-se (substitui) justamente ele.

É possível também que a análise sintática gere mais de uma árvore válida, formando assim um caso de ambiguidade sintática. Outra evolução possível seria realizar o tratamento dessas ambiguidades, mas para isso seria necessário subir mais um nível na tradução, chegando ao semântico, para que essas alternativas possam ser esclarecidas quando comparados os seus significados.

O tradutor

As regras de tradução sintática devem ser alteráveis após o deploy do projeto, para que especialistas (linguistas, por exemplo) possam melhorar o conjunto de regras (definição formal em alguma notação similar à gramática transformativa).

No caso do tradutor não conhecer um sinal possível para uma palavra, esta deve ser representada com a execução soletrada, usando os sinais correspondentes às letras. Isso é especialmente útil para nomes próprios.

Para uma melhor tradução para Libras, faltam ser considerados alguns aspectos sintáticos como a flexão de gênero e de número e a escolha da melhor ordem da frase (SVO, SOV ou OSV). Além disso, a incorporação de uma análise semântica também representaria uma grande evolução, possibilitando certas junções de sinais como a incorporação do objeto no próprio verbo ou determinadas adaptações semântico-visuais que tornariam a fala mais fluente.

Quanto a saída gerada, após a tradução ser exibida ao usuário é desejável que um vídeo possa ser baixado; alternativamente pode-se baixar o XML com a descrição dos sinais da sentença, que poderia ser usado em uma versão desktop do Virtual Jonah que funcionaria como um *LIBRAS player*.

Referências Bibliográficas

BATES, B.; SIERRA, K.; BATES, B. Head First Servlets and JSP. O'RELLY, 2008.

BRITO, L. Estrutura ling
"uística da Libras. *INES. Disponível em:< http://www. ines. org. br/i-nes_livros/35/35_PRINCIPAL. HTM>. Acesso em 05/12/2010*, v. 11, 2006.

BROWN, P. et al. A statistical approach to machine translation. *Computational linguistics*, MIT Press, v. 16, n. 2, p. 79–85, 1990. ISSN 0891-2017.

CAPOVILLA, F.; RAPHAEL, W. *Dicionário enciclopédico ilustrado trilíng
"ue da língua de sinais brasileira.* São Paulo: EdUSP, 2001. ISBN 8531406692.

CERQUEIRA, A. J. D. O. S. et al. Implementação de Buscas Utilizando Linguagem Natural Através de Algoritmos Adaptativos. *Trabalho de Conclusão de Curso apresentado à Escola Politécnica da USP*, 2010.

CHOMSKY, N. On certain formal properties of grammars. *Information and control*, Elsevier, v. 2, n. 2, p. 137–167, 1959. ISSN 0019-9958.

CHOMSKY, N. *Aspects of the Theory of Syntax.* [S.l.]: The MIT press, 1965. ISBN 0262530074.

CORADINE, L. et al. Sistema Falibras: Um Intérprete Virtual como Ferramenta de Apoio pedagógico à Educação de Surdos. In: *VII Congresso Iberoamericano de Informática Educativa Especial.* Argentina,Mar del Plata: [s.n.], 2007.

FELIPE, T. Introdução à gramática da LIBRAS. *Série Atualidades Pedagógicas*, v. 4, n. 3, p. 81–107, 1997.

FERNEDA, E.; COSTA, E. de B.; ALMEIDA, H. de. Inclusão digital de portadores de necessidades especiais através da comunicação móvel. 2003.

FIORIN, J. *Introdução à linguística: princípios de análise.* [S.l.]: Contexto, 2005. ISBN 8572442219.

FRIEDMAN, J. et al. *A computer model of transformational grammar.* New York: American Elsevier, 1971. ISBN 0444000844.

KARNOPP, L.; QUADROS, R. Língua de sinais brasileira: estudos ling
"uísticos. *Porto Alegre: Artmed*, 2004.

LIRA, G.; SOUZA, T. A. F. Dicionário da Língua Brasileira de Sinais. *Acessibilidade Brasil. Disponível em: <http://www.acessobrasil.org.br/libras/>. Acesso em: 05/12/2010*, 2008.

LIRA, G. A. O Impacto da Tecnologia na Educação e Inclusão Social da Pessoa Portadora de Deficiência Auditiva: Tlibras Tradutor Digital Português x Língua Brasileira de Sinais-Libras. *Boletim Técnico Senac*, 2003. Disponível em: <http://www.senac.br/BTS/293/boltec293d.htm>. Acesso em: 05/12/2010.

LOPEZ, A. Statistical machine translation. *ACM Computing Surveys (CSUR)*, ACM, v. 40, n. 3, p. 1–49, 2008. ISSN 0360-0300.

LUFT, C. *Moderna gramática brasileira: edição revista e atualizada*. São Paulo: Globo Livros, 2002. ISBN 8525036218.

LYONS, J. *As idéias de Chomsky. Tradução de Octanny Silveira da Mota e Leônidas Hegenberg*. São Paulo: Cultrix, 1970.

MULLEN, T. *Mastering Blender*. [S.l.]: Sybex, 2009. ISBN 0470407417.

QUADROS, R. *Phrase structure of Brazilian Sign Language*. Tese (Doutorado) — Pontifícia Universidade Católica do Rio Grande de Sul, Porto Alegre, 1999.

RAMOS, M.; NETO, J.; VEGA, I. *Linguagens Formais - teoria, modelagem e implementação*. Porto Alegre: Editora Bookman, 2009.

SECRETARIA MUNICIPAL DE EDUCAÇÃO. *Orientações curriculares e proposição de expectativas de aprendizagem para Educação Infantil e Ensino Fundamental : Libras*. São Paulo, 2008.

SUTTON, V. Lições sobre o SignWriting; tradução de STUMPF, M.R. 2003.

UCHIDA, H.; ZHU, M. The Universal Networking Language beyond Machine Translation. In: *International Symposium on Language in Cyberspace, Seoul*. [S.l.: s.n.], 2001. p. 26–27.

VAUQUOIS, B. Automatic Translation - A Survey of Different Approaches. *Statistical Methods in Linguistics*, p. 127–135, 1976.

Glossário

- API: conjunto de código-fonte utilizado por programadores para desenvolver novas aplicações.
- Applet: tecnologia Java para executar aplicações em navegadores web (browsers).
- Árvore: estrutura de dados de uso comum na computação.
- Árvore sintática: árvore utilizada para descrever a estrutura sintática de uma sentença.
- Banco de dados: sistema de informática para o armazenamento de informações.
- Lexema: símbolo gráfico processado como uma unidade, tipicamente uma palavra.
- Ferramenta de modelagem: software utilizado para a criação de modelos tridimensionais.
- Framework: ferramentas de desenvolvimento de softwares focadas em um tipo de problema específico (ex: existem frameworks para sistemas web, para computação gráfica, para manipulação de bancos de dados etc).
- Java: linguagem de programação multi-plataforma bastante utilizada.
- LIBRAS: Língua Brasileira de Sinais (grafada com letras maiúsculas).
- Morfema: unidade mínima com significado; morfemas formam palavras; um morfema geralmente não faz sentido isoladamente (ex: radicais de verbos). Um morfema pode apresentar variações fonéticas (morfes); ex: em "irreal"e "infeliz", as partes em negrito são o mesmo morfema.
- SignWriting: forma de escrita da LIBRAS através de símbolos gráficos especiais que representam aspectos morfológicos dos sinais.
- Sintagma: constitui uma unidade sintática, que pode ser formada por várias palavras.
- Modelo 3D: é o "desenho"3D de coisas e personagens.
- MVC: prática de programação para separar o código da apresentação do código da lógica de negócio.

- Open source: software que permite ao usuário ter acesso ao código-fonte e realizar alterações na versão original; também chamado de software livre.
- OSCIP - Organização da Sociedade Civil de Interesse Público.
- VOLP - Vocabulário Ortográfico da Língua Portuguesa
- WebService: forma de comunicação entre software através da troca de mensagens encapsuladas em documentos XML.
- XML: linguagem de descrição de dados flexível em formato texto padronizada pela W3C.
- W3C: consórcio internacional que define os padrões das tecnologias utilizadas no desenvolvimento de sites e aplicações web.

Apêndice A

Gramática livre de contexto para Português

Segue listagem das regras de produção da gramática livre de contexto descritiva da língua portuguesa, baseada na obra Moderna Gramática Brasileira (LUFT, 2002), a qual apresenta vários exemplos de formação de árvores sintáticas. A partir de tais exemplos, construiu-se a uma proposta de gramática restrita apenas às orações com período simples, ou seja, a frases com apenas uma oração (verbo) e que estejam na ordem direta (as regras sugeridas por Luft não geram sentenças em ordem inversa).

Elementos em itálico representam elementos terminais, que nessa caso são as classes morfológicas das palavras. Já os demais elementos são os não-terminais.

Legenda:

- F - Frase
- PS - Período Simples
- OA - Oração Absoluta
- SS - Sintagma Substantivo
- SV - Sintagma Verbal
- SN - Sintagma Nominal
- SAdv - Sintagma Adverbial
- SP - Sintagma Preposicional
- PrAdj - Pronome Adjetivo
- *subst* - Substantivo
- *ppes* - Pronome Pessoal
- *v* - Verbo intransitivo ou transitivo
- *vlig* - Verbo de ligação

•*prep* - Preposição

Gramática:

•F ⇒ PS

•PS ⇒ OA

•OA ⇒ SS + Aux + SV + SAdv

•OA ⇒ SS + Aux + SN + SAdv

•SS ⇒ PrAdj + *subst*

•SS ⇒ *subst*

•SS ⇒ *ppes*

•SS ⇒ vazio

•NOME ⇒ *subst. comum*

•NOME ⇒ *subst. próprio*

•SV ⇒ *v* + SS + SP

•SN ⇒ *vlig* + SAdj

•Aux ⇒ *negação*

•Aux ⇒ vazio

•SP ⇒ *prep* + SS

•SP ⇒ *ppes*

•SP ⇒ vazio

•SAdv ⇒ *Advérbio*

•SAdv ⇒ vazio

•SAdj ⇒ *Advérbio* + *Adjetivo*

•SAdj ⇒ *Adjetivo*

- PrAdj ⇒ *Artigo (definido / indefinido)*
- PrAdj ⇒ *Pronome Demonstrativo*
- PrAdj ⇒ *Pronome indefinido*
- PrAdj ⇒ *Pronome possessivo*

Apêndice B

Organização do CD

O CD que acompanha essa documentação tem seu conteúdo organizado com a seguinte estrutura de diretórios:

- Código-Fonte - Contém todos os códigos-fonte utilizados nesse projeto
- Documentação - Contém cópia digital deste próprio documento
- Executáveis - Contém arquivos necessários para a execução do sistema