Universidade Católica Dom Bosco

Curso de Bacharelado em Engenharia de Computação

Aspectos de Projeto e Implementação do Não-determinismo no AdapTools e seus Impactos no Aperfeiçoamento da Ferramenta

Denys G. Santos

Leandro de Jesus

Prof(a). Orientador(a): Hemerson Pistori

Relatório final, vinculado ao Grupo de Pesquisa em Engenharia e Computação da Universidade Católica Dom Bosco (GPEC), submetido como um dos requisitos para a obtenção do título de Bacharel em Engenharia de Computação.

UCDB - Campo Grande - MS - Novembro/2006

Às nossas famílias e amigos.

Agradecimentos

A Deus pai que em sua glória e sabedoria incontestável concedeu-nos graça e força suficiente para que pudéssemos concluir este trabalho.

À familia que com seu amor, apoio e compreensão deu-nos estímulos para continuar nossa caminhada.

Aos professores que contribuiram com nossa formação, profissional e pessoal, utilizando de seus conhecimentos e experiências de vida.

Ao professor Amaury A. Castro Jr., pela dedicação, amizade, pelos inúmeros erros de português corrigidos e pelo bom humor sempre presente. Ao professor Hemerson Pistori, grande pesquisador e docente que nos despertou o gosto pela pesquisa.

Aos colegas e amigos que durante estes cinco anos de universidade viveram conosco dia-a-dia passando por dificuldades e alegrias.

A todos esses e muitos outros que sempre nos apoiaram e estiveram presentes neste caminho até a graduação, deixamos nosso mais profundo agradecimento.

Resumo

Neste trabalho apresentamos uma ferramenta e técnicas baseadas em tecnologia adaptativa, mais especificamente autômatos adaptativos. Identificamos e mostramos os principais elementos envolvidos na vocalização humana e realizamos a fundamentação teórica necessária para implementação da execução não-determinística que utilizou o problema da síntese de voz como forma de validação. O AdapTools, uma ferramenta de apoio ao desenvolvimento de autômatos adaptativos, oferece um ambiente para execução, depuração e recursos de animação gráfica, além de incluir um protótipo para geração de voz através de ações semânticas. Identificamos e explanamos as principais características da ferramenta AdapTools, dando ênfase aos aspectos de implementação, ao novo modelo de codificação de autômatos utilizando um compilador integrado ao seu ambiente e a melhoria na inserção de rotinas semânticas que torna desnecessário a recompilação do código fonte, além, de uma série de melhorias e contribuições realizadas na ferramenta. Esse compilador, desenvolvido durante o trabalho, chamado AdapMap, utiliza uma linguagem de alto nível, de mesma denominação que oferece uma interface mais amigável para descrição e análise dos autômatos, além, de oferecer análises sintáticas e semânticas mais apuradas, diminuindo o número de erros ocorridos na codificação. Neste documento são exibidos os resultados obtidos da implementação do não-determinismo, sua utilização no problema da síntese de voz e as potencialidades na utilização deste recurso como solução para uma nova gama de problemas.

Abstract

In this work, we described a tool and a technique based in adaptive technology, more specifically adaptive automata,. We also identified and showed the main element involved at human vocalization, and realized elements of theoretical recital necessary for the problem solution no determinism that used the voice synthesis problem to validate it. The AdapTools, a support tool for adaptive automata development, provide an environment for automata execution and depuration, it also provides the graphics animation resources and include an archetype to speech generation through semantics actions. We talked about any AdapTools characteristics, emphasizing the implementation of new automata codification model, which use an integrated compiler in its environment and the improvement of insertion of semantic routines becoming unnecessary the recompilation of the source code and the series of improvements and contributions carried through in the tool. This compiler, developed during this work, called AdapMap, uses a high level language that receives the same name that offers a friendlier interface for description and analysis of the automata and provide a simple description form and analysis of automata, beyond provide a good syntactic and semantic analysis, minimizing the occurrence of error at codification. In this document the gotten results of implementation of no determinism are shown, its use in the problem of voice synthesis and the potentialities in the use of this resource as solution for a new gamma of problems.

Conteúdo

Capítulo 1

Introdução

A popularização dos computadores pessoais trouxe consigo uma busca constante pelo aumento do poder de processamento e desenvolvimento de novas aplicações. Uma das inúmeras atividades possíveis de se realizar hoje através dos computadores é o processamento e a síntese de voz [25].

Trabalhos sobre síntese de voz estão dando origem a uma vasta gama de aplicações. Transformar a forma de comunicação escrita através da conversão de texto em fala representaria uma solução ágil do uso de sistemas computacionais em situações em que a comunicação visual não é adequada. Isso se torna possível através da construção de um sintetizador de voz que tem como entrada um texto escrito. Atualmente, existem inúmeros sintetizadores de voz, mas a maioria para a língua inglesa, francesa ou espanhola (*The Festival Speech Synthesis System* [1], *Multilingual Text-to-Speech Systems* [2], *Java Speech API Programmer's: Speech Synthesis* [3], *The MBROLA Project* [4], entre outros). Para o português, estes sintetizadores são raros, como exemplo o *ORADOR* [5].

Os sistemas de conversão texto-voz são geralmente divididos em dois componentes específicos [2]:

1. Regras de pronúncia, que convertem um texto escrito para as suas transcrições fonéticas (Ex.: Padrão SAMPA [26, 24]), empregando para isso, regras sintáticas e semânticas, que podem ser encontradas em

[1] The University of Edinburgh - http://www.cstr.ed.ac.uk/projects/festival/

[2] Bell Laboratories - http://www.bell-labs.com/project/tts/

[3] Sun Microsystems - http://java.sun.com/products/java-media/speech/forDevelopers-/jsapi-guide/Synthesis.html

[4] TCTS Lab of the Faculté Polytechnique de Mons (Belgium) - http://-tcts.fpms.ac.be/synthesis/mbrola.html

[5] LINSE (Laboratório de Circuitos e Processamento se Sinais) - http://www.linse.ufsc.br/

gramáticas da língua.

2. Síntese da fala, responsável pela transformação do texto transcrito para uma mensagem audível.

A transformação de texto escrito para sua transcrição fonética pode ser feita de várias formas, algumas técnicas utilizadas são: transdutores de estados finitos [10], modelos ocultos de Markov [17, 9] e *autômatos adaptativos* [26, 27, 4, 19].

Estruturas que se destacam no processamento de linguagem natural são os modelos ocultos de Markov (HMM) e os transdutores de estados finitos (WFST), que trazem consigo novas técnicas e algoritmos para o aumento de sua capacidade de representação e eficiência. Tais algoritmos podem servir de base para novos métodos e melhorias na tecnologia dos *autômatos adaptativos*, que será apresentado neste trabalho. Mohri [12] apresenta um algoritmo para reescrita de regras em autômatos com propriedades muito similares às das *Funções Adaptativas*.

A construção de um conversor texto-voz através de *autômatos adaptativos* pode ser realizada tanto através do mapeamento de regras [4], que estão sujeitas as condições da formação de palavras segundo a gramática da língua, quanto através do mapeamento de palavras [26], que são representadas diretamente pelos estados e transições do autômato.

Uma característica importante presente na especificação dos *autômatos adaptativos*, é a capacidade de tratar transições de forma não-determinística. Em sua definição original, transições internas do autômato são avaliadas e se restando apenas uma transição aplicável, esta é executada deterministicamente, ou restando mais de uma transição aplicável, então todas as transições correspondentes são tratadas em paralelo, de forma não-determinística na operação do autômato. Esta propriedade é de extrema importância para o tratamento de linguagens dependentes de contexto [14].

A dependência de contexto é um fator de alta relevância para a conversão de texto para voz, pois, certas palavras da língua portuguesa, bem como símbolos, abreviações e datas podem ser traduzidas de maneiras foneticamente diferentes dependendo do local onde ela se encontra, por exemplo *"Gosto do gosto de maracujá"*, nesta sentença a palavra *gosto* assume dois significados diferentes: o primeiro diz respeito ao verbo gostar e o segundo ao substantivo [8, 26].

Autômatos adaptativos são uma solução atrativa para conversão de texto para sua transcrição fonética, respeitando as características intrinsecamente dependentes de contexto do problema. Segundo [14], pela sua generalidade e estrutura estratificada, *autômatos adaptativos* oferecem uma forma declarativa, unificada e hierárquica de representação para a definição de todos

os tipos de linguagens textuais, proporcionando, em adição, um substrato conceitual que permite implementações muito eficientes.

Formalismos baseados em *dispositivos adaptativos*, como por exemplo, os *autômatos adaptativos*, representam uma alternativa atraente no desenvolvimento de sintetizadores de voz, devido a seu grande poder de representação, contribuindo no desenvolvimento de projetos de aplicações complexas que necessitam tomar decisões ou ainda necessitem de um dispositivo de aprendizagem [21].

Como ferramenta para projetos, implementações e testes de *autômatos adaptativos*, foi desenvolvido um ambiente computacional chamado, AdapTools [16, 19, 20, 21]. Este ambiente computacional inclui recursos de depuração, execução passo-a-passo, visualização de variáveis de pilha, animação gráfica, oferece um mecanismo para controle e criação de projetos, uma grande variedade de exemplos, além de proporcionar uma interface para implementação de rotinas semânticas.

Visto que os tradicionais autômatos de estados finitos, não-determinísticos com transições vazias (ϵ-*automata*) e os autômatos a pilha estruturados são especializações dos autômatos adaptativos, o AdapTools pode ser usado também como um laboratório virtual, através do qual o aluno pode praticar experimentalmente conceitos da teoria dos autômatos.

Este trabalho contempla um estudo sobre autômatos adaptativos, formalismo utilizado para resolver uma ampla gama de problemas, principalmente problemas relacionados a linguagens [14]. A refatoração da estrutura de armazenamento de regras da ferramenta *AdapTools*, realizada neste trabalho, fornece novos métodos para manipulação eficiente da estrutura de regras do autômato. Essas alterações no AdapTools, proporcionaram ferramentas para construção de uma solução para a implementação do tratamento e execução de transições não-determinísticas. A validação do não-determinismo foi realizada quando submetido aos testes com o autômato de transcrição fonética, onde é aplicada uma rotina semântica de vocalização que produz o som.

O entendimento da formação da fala humana em todos seus estágios é de suma importância para solução de problemas e construção de aplicações em síntese de voz. No Capítulo 2 realizaremos um breve apanhado a respeito da estrutura do aparelho fonador e dos métodos para classificar, descrever e transcrever o som da fala (fonética) dando ênfase ao alfabeto fonético dos métodos de avaliação da fala (SAMPA).

No Capítulo 3 apresentaremos os principais formalismos e técnicas computacionais, que aliados aos conceitos apresentados no Capítulo 2, fornecerão um suporte teórico para construção de um tradutor texto-voz baseado em *tecnologia adaptativa*, mais especificamente *autômatos adaptativos*.

Apresentaremos, no Capítulo 4, uma proposta e o desenvolvimento da

integração do AdapTools com o compilador para autômatos adaptativos, criado durante este trabalho e denominado AdapMap. A linguagem utilizada neste compilador oferece uma interface de auto nível para codificação e resolve algumas dificuldades encontradas na descrição e análise estrutural dos autômatos. Para facilitar essa integração, faremos uma análise sobre os aspectos de implementação do AdapTools, que também será útil para integração e desenvolvimento de novas funcionalidades. No Capítulo 5 utilizaremos o conhecimento sobre a ferramenta para descrever as novas funcionalidades implementadas, abordando as principais dificuldades encontradas e como foram tratadas.

Capítulo 2

Fala Humana

O entendimento da formação da fala humana em todos seus estágios é de suma importância para solução de problemas e construção de aplicações em síntese de voz. A fala é uma característica presente na maioria dos seres humanos. O modo de utilização da fala, no entanto, pode variar de país para país, ou até mesmo de região para região dentro de um mesmo país, como exemplo pode ser citado o sotaque do Rio Grande do Sul e Rio Grande do Norte, que, apesar de muito diferentes dizem respeito à mesma língua (português do Brasil). Embora a fala aparente ser algo simples, ela envolve um complexo conjunto de órgãos, que juntos formam uma estrutura denominada *aparelho fonador*, que é responsável desde a captura do ar até a geração do som vocal [6].

2.1 Aparelho Fonador

A estrutura do aparelho fonador é bastante complexa envolvendo além de órgãos, como pulmões, laringe, boca, língua, mandíbula e dentes mais de 100 músculos envolvidos no controle direto da produção das ondas sonoras da fala. Os principais componentes da estrutura do aparelho fonador podem ser vistos na Figura 2.1.

Segundo [6] o aparelho fonador pode ser dividido em três partes: os foles, o vibrador e os resonadores. Os foles correspondem aos órgãos responsáveis pelo sopro, pulmões, brônquios, traquéia e diafragma. Este conjunto, na geração da fala, tem como objetivo a geração do sopro fonatório, que pode ser produzido de diferentes maneiras: baixamento da caixa torácica (sopro torácico superior), ação dos músculos abdominais (sopro abdominal) e as vezes por flexão torácica [6].

A emissão do sopro fonador inicia-se com uma inspiração seguida de um

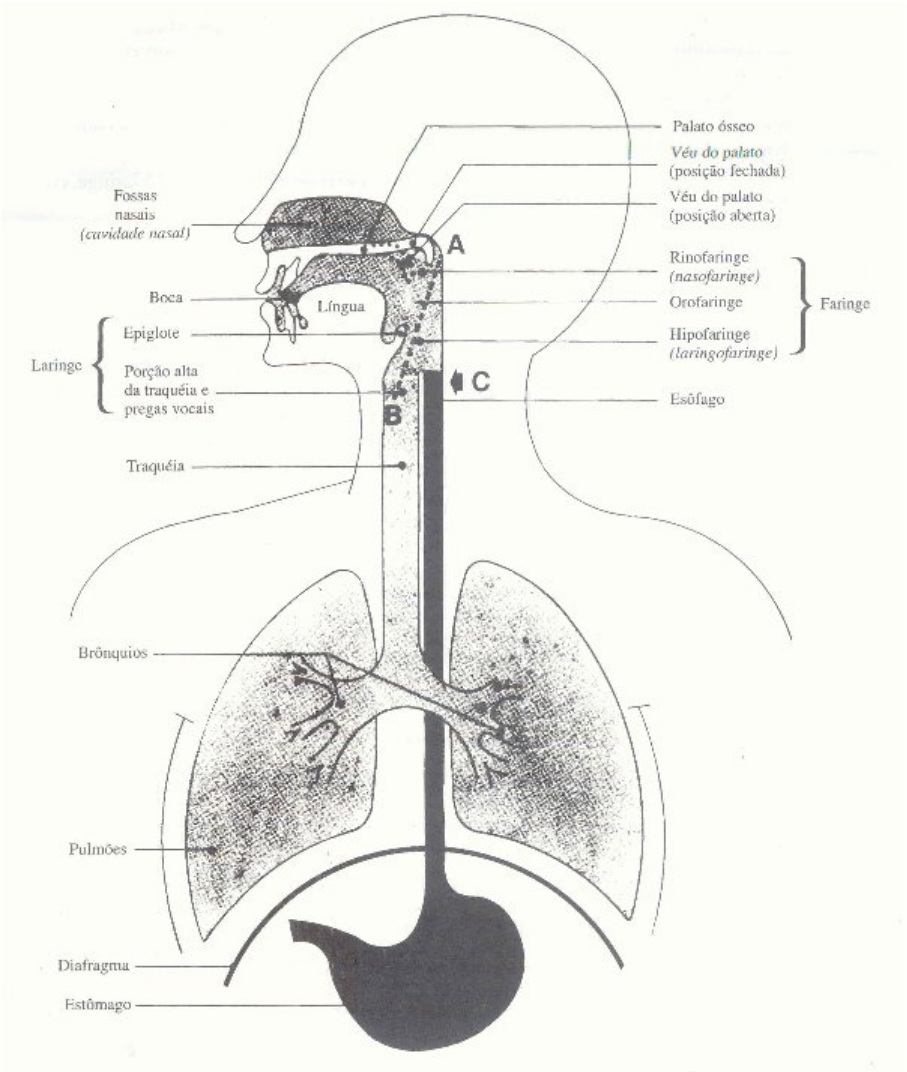

Figura 2.1: Figura que representa os aspectos fisiológicos do aparelho fonador humano [6]

impulso respiratório, ou seja, é necessário que o ar seja armazenado dentro dos pulmões para depois expeli-lo. Nesta etapa, o principal órgão é o diafragma, que é uma lâmina muscular em forma de calota que separa o tórax do abdome, tendo como função inspirar (impulso fonatório) e respirar (controlando o sopro fonatório no momento da produção vocal) [6].

O vibrador, ou laringe, é o nome dado a extremidade superior do tubo traqueal, esta é formada por cartilagens que estão ligadas entre si por ligamentos e lâminas de aponeurose e por músculos coberto por mucosas. As

pregas vocais (cordas vocais) são parte integrante do vibrador, formadas por dois músculos e pelas mucosas que os recobrem [6].

O posicionamento das pregas vocais apresenta-se como dois lábios formando saliência nas paredes interiores da laringe (um à direita e outro à esquerda). Essas pregas podem aproximar-se e utilizar do sopro fonador para gerar vibrações (como lábios ou cordas). Quando as cordas vocais se encontram afastadas existe uma espaço entre elas. Este espaço é denominado glote [6].

Os vibradores ou pavilhão faringobucal e cavidades anexas, correspondem a faringe, boca e fossas nasais [6].

A faringe está localizada acima da laringe. Essa é uma cavidade muscular capaz de se contrair lateralmente ou de traz para frente mediante a ação de músculos, podendo também, ter o seu volume alterado horizontalmente e verticalmente. A faringe, juntamente com a boca, formam um dos principais resonadores e o principal articulador [6].

As fossas nasais são duas cavidades dispostas paralelamente e seguem das narinas até a faringe. Essas duas cavidades são separadas por uma parede cartilaginosa denominada palato ósseo [6].

2.2 Fonética

A fonética é a ciência que estuda métodos para classificar, descrever e transcrever os sons da fala. Podemos dividir a fonética em três vertentes principais: articulatória, auditiva, acústica. A fonética articulatória estuda a produção da fala sob os aspectos fisiológicos articulatório, a auditiva e a percepção da fala, a acústica as propriedades físicas do som saído do falante para o ouvinte [26, 7]. Além dessas três vertentes, em [22] é apresentada uma quarta, a fonética instrumental, que diz respeito ao estudo das propriedades físicas da voz com o auxílio de instrumentos de laboratório.

A formação da voz humana pode ser observada através do modelo fonético, que utiliza da unidade básica de formação vocal, denominada fonema [26]. Uma outra característica da unidade básica do trato vocal é a não repetição, ou seja, sempre teremos a impressão de pronunciar ou de se ouvir o mesmo som, no entanto, o mesmo falante nunca o produzirá de maneira idêntica [23].

A unidade de trato vocal, fonema, pode ser dividido em classes que melhor representam o modelo de articulação utilizado para a produção da voz humana. As classes para classificação dos segmentos fonéticos são [26, 7]:

- **Vogais:** são segmentos vozeados produzidos sem obstáculos. Na língua

portuguesa são identificadas 13 vogais, que podem ser observadas na Tabela 2.1.

- **Glides:** Conhecidos também como semivogais, os glides são ditongos formados pela união da letra *a* em união com *i* e *u* e seus sons nasais. Os glides da língua portuguesa podem ser observados na Tabela 2.2.
- **Oclusivas:** Nestas, que podem ser observadas na Tabela 2.3, há uma contrição total do trato vocal seguido de uma rápida liberação da tensão acumulada.
- **Fricativas:** Observadas na Tabela 2.4, as fricativas são geradas pela turbulência provinda pela contrição do trato vocal.
- **Nasais:** Mostrada na Tabela 2.5, as nasais são formadas pela vibração das cordas vocais com a boca fechada, o que faz com que o ar circule por cima do palato ósseo e saia pelas narinas. Pelo fato de a geração desses fonemas se dar através de uma oclusão provocada pelo fechamento da boca, esses fonemas podem também ser chamados de oclusivos nasais.
- **Líquidas:** São geradas pela passagem do ar alterado pela língua. Esta pode ser dividida em líquidas vibrantes e laterais. As líquidas vibrantes são produzida pela vibração da língua. As laterais são aquelas formadas pelo ar passando pelas laterais da língua. A Tabela 2.6 apresenta os fonemas líquidos da língua portuguesa do Brasil.

A representação fonêmica da língua portuguesa, bem como de várias outras línguas, é uma tarefa complexa e exige um trabalho de transcrição, em que um texto escrito ou falado é mapeado em um alfabeto fonêmico. Um exemplo de um alfabeto fonêmico é o SAMPA (*Speech Assessment Methods Phonetic Alphabet* - Alfabeto Fonético dos Métodos de Avaliação da Fala) descrito na seção 2.2.1.

2.2.1 Alfabeto Fonético dos Métodos de Avaliação da Fala (SAMPA)

O IPA (*International Phonetic Alphabet* - Alfabeto Fonético Internacional) faz freqüente uso de símbolos não convencionais para a representação dos sons fonêmicos. Em meados de 1980 foi criado o padrão SAMPA, que conta com codificação em ASCII para a representação dos símbolos dessa linguagem [24, 26].

O SAM (*Phonetic Alphabet - Speech Assessment Methods Phonetic Alfabet*) foi inicialmente criado para a transcrição das línguas européias e mais

tarde adaptado para várias outras línguas, incluso português do Brasil. O SAMPA, no entanto, não compreende todo o Alfabeto Fonético Internacional (IPA), o que torna o SAMPA válido apenas aos idiomas os quais foi adaptado [24, 26].

Nas Tabelas 2.1, 2.2, 2.3, 2.4, 2.5 e 2.6 é mostrado o alfabeto fonético do padrão SAMPA adaptado à língua portuguesa do Brasil.

Símbolo IPA	**Símbolo Sampa**	**Elevação da Língua**	**Posição da Língua**	**Palavra**	**Transcrição SAMPA**
ɐ	6	média	média	cama	k6m6
a	a	baixa	média	cara	kar6
e	e	média	anterior	pêra	per6
ɛ	E	baixa	anterior	sete	sEt@
ɨ	@	alta	média	que	k@
i	i	alta	anterior	fita	fit6
o	o	média	posterior	dou	do
ɔ	O	baixa	posterior	corda	cOrd6
u	u	alta	posterior	mudo	mudu
ɐ̃	6~	média	média	manta	m6~t6
ẽ	e~	média	anterior	menta	me~t6
ĩ	i~	alta	anterior	pinta	pi~t6
õ	o~	média	posterior	ponta	po~ta
ũ	u~	alta	posterior	mundo	mu~du

Tabela 2.1: Tabela de Vogais do português do Brasil segundo o padrão SAMPA

Símbolo IPA	**Símbolo Sampa**	**Elevação da Língua**	**Posição da Língua**	**Palavra**	**Transcrição SAMPA**
w	w	alta	posterior	pau	paw
w	j	alta	anterior	pai	paj
w̃	w~	alta	posterior	cão	k6 w
j̃	j~	alta	anterior	mãe	m6 j

Tabela 2.2: Tabela de Glides do português do Brasil segundo o padrão SAMPA

Símbolo IPA	**Símbolo Sampa**	**Vozeamento**	**Ponto Articulação**	**Palavra**	**Transcrição SAMPA**
p	p0,p	não	bilabial	pai	p0paj
t	t0,t	não	apicodental	tia	t0ti6
k	k0,k	não	velar	casa	k0k6za
b	b0,b	sim	bilabial	bar	b0bar
d	d0,d	sim	apicodental	data	d0dat6
g	g0,g	sim	velar	gato	g0gatu

Tabela 2.3: Tabela de Oclusivas do português do Brasil segundo o padrão SAMPA

Símbolo IPA	**Símbolo Sampa**	**Vozeamento**	**Ponto Articulação**	**Palavra**	**Transcrição SAMPA**
f	f	não	labiodental	férias	fEri6S
s	s	não	apicodental	selo	selu
ʃ	S	não	palatal	chave	Sav@
v	v	sim	labiodental	vaca	vak6
z	z	sim	apicodental	azul	6zul
ʒ	Z	sim	palatal	agir	6Zir

Tabela 2.4: Tabela de Fricativas do português do Brasil segundo o padrão SAMPA

Símbolo IPA	**Símbolo Sampa**	**Vozeamento**	**Ponto Articulação**	**Palavra**	**Transcrição SAMPA**
m	m	sim	bilabial	meta	mEt6
n	n	sim	apicodental	neta	nEt6
ɳ	J	sim	palatal	senha	s6J6
	N	sim			

Tabela 2.5: Tabela de Nasais do português do Brasil segundo o padrão SAMPA

Símbolo IPA	**Símbolo Sampa**	**Vozeamento**	**Ponto Articulação**	**Palavra**	**Transcrição SAMPA**
l	l	sim	labiodental	lado	ladu
ɫ	l~	sim	apicodental	sal	sal
ʎ	L	sim	palatal	folha	foL6
R	R		velar	carro	kaRu
ɾ	r		apicodental	caro	karu

Tabela 2.6: Tabela de Líquidas do português do Brasil segundo o padrão SAMPA

Capítulo 3

Fundamentação Teórica

Neste capítulo apresentaremos os principais formalismos e técnicas computacionais, que aliados aos conceitos apresentados no capítulo anterior, fornecerão um suporte teórico para construção de um tradutor texto-voz baseado em *tecnologia adaptativa*, mais especificamente *autômatos adaptativos*.

Na Seção 3.1, faremos uma breve introdução de fatos históricos e métodos para tradução de texto-voz. Mostraremos também modelos computacionais utilizados atualmente para resolução desse problema, bem como fatores que implicam em uma maior complexidade no desenvolvimento de sintetizadores de voz, por exemplo, a dependência de contexto.

Na Seção 3.2, apresentaremos a *tecnologia adaptativa*, um formalismo que surgiu inicialmente como alternativa para solução de problemas relacionados a construção de compiladores e se tornou fonte de pesquisas em várias áreas da computação. Generalizando uma série de dispositivos o conceito de *dispositivos guiados por regras adaptativos* apresentados na seção 3.2.1, define os formalismos que são constituídos por conjuntos de regras, dentre esses formalismos os *autômatos adaptativos* são um dos mais utilizados para criação de novas soluções.

Discutiremos como os *autômatos adaptativos*, apresentados na seção 3.2.2, podem ser uma solução atrativa para solução do problema da *síntese de voz*, devido a possibilidade do tratamento de linguagens dependentes de contexto de forma naturalmente sintática. A existência de uma ferramenta para execução de *autômatos adaptativos*, *AdapTools*(seção 3.2.3), apresentará relevante importância para a criação do tradutor texto-voz.

O desenvolvimento do tradutor texto-voz, como em todo software, requer métodos para organização do projeto. A *engenharia de software*(seção 3.3) é ferramenta essencial para direcionar a construção do software de uma forma ágil e flexível mantendo certos padrões de qualidade, além de conservar a comunicação e o entendimento entre os desenvolvedores do projeto.

3.1 Síntese de voz

Em 1779 Christian Gottlieb Krazenstein desenvolveu o primeiro trabalho em *síntese de voz*, neste ele usava uma palheta vibrante e um fluxo contínuo de ar como o mecanismo de um órgão [18, 8]. Um pouco mais tarde, Wolfgang Von Kempelen criou uma máquina que podia criar algumas sentenças completas. Esta tinha como componentes principais o fole, que representa os pulmões, e a palheta para representação das cordas vocais. A produção da fala nesta máquina era feita pela alteração manual da caixa de ressonância, produzindo assim, sons distintos [18, 8].

Outros trabalhos bem sucedidos em síntese de voz só voltaram a ser mencionados novamente em 1939, em uma feira mundial realizada em Nova York, em que, Dudley desenvolveu nos Laboratórios Bell o VODER (*Voice Operation Demonstrator*). O VODER era formado por dois geradores de voz totalmente independentes, um para os sons periódicos, que simulava as cordas vocais durante o vozeado, e um segundo para o ruído que representa as turbulências causadas pelas constrições no trato vocal [18, 8].

Nos últimos anos, muitos métodos para a síntese de voz estão sendo desenvolvidos, motivados pela ampla gama de aplicações, como: a síntese de voz a partir de um e-mail ou a utilização de aplicações que ajudem deficientes visuais. Para uma boa síntese de voz é necessário que haja uma boa entonação e ritmo, que, quando aplicados corretamente, garantem uma maior naturalidade [26]. Os métodos de síntese de voz podem ser classificados em três tipos:

- **Síntese Articulatória:** Baseia-se no aparelho vocal humano, limitando a fala, resonâncias e articulação, produzidas por um conjunto de regras que simulam os lábios, línguas e cordas vocais.
- **Síntese Fonética:** É produzida através da soma de frequências (sons vocais) resultantes de resonadores dispostos estrategicamentes para processar a entrada.
- **Síntese por Concatenação:** Utiliza arquivos de som previamente gravados, em que, cada fonema é associado a um arquivo que representa seu som. A partir de uma cadeia de entrada, que representa o texto a ser processado, os fonemas são reconhecidos e seus sons concatenados a fim de produzir a pronúncia.

Uma das formas de se fazer síntese de voz é a conversão texto-voz (*text-to-speech*-TTS) [18]. Segundo [8, 26] a sintese de voz a partir do processamento

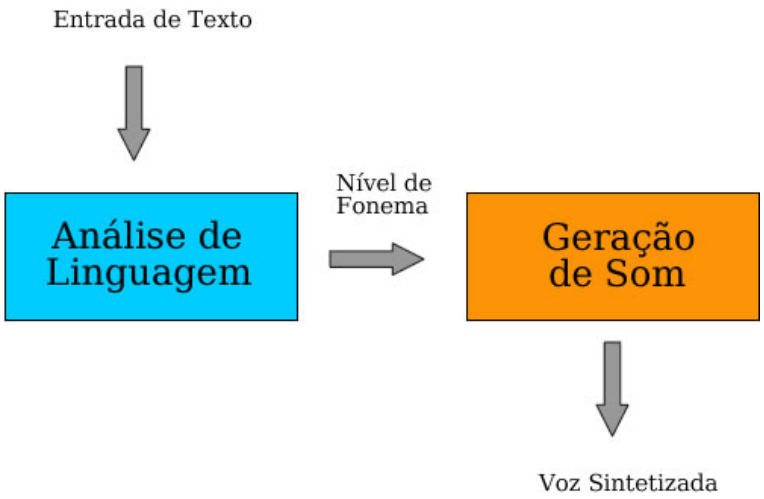

Figura 3.1: exemplo do procedimento de síntese de voz através do método de conversão Texto-voz

de um texto de entrada pode ser dividido em duas etapas: processamento de texto e geração de som, representados na Figura 3.1.

O processamento de texto é a primeira etapa para a construção de um tradutor texto voz. Nesta etapa o texto é analisado e traduzido para a forma fonética, seguindo as regras léxicas e sintáticas da linguagem a ser analisada. Esta fase é extremamente complexa, e envolve a dependência de contexto [8].

A dependência de contexto é um fator de extrema importância para a conversão de texto escrito para o texto fonético, pois, certas palavras da língua portuguesa, bem como símbolos, abreviações e datas podem ser traduzidas de maneiras foneticamente diferentes dependendo do local onde ela se encontra, por exemplo *"Fui à sede da fazenda porque estava com sede."*, nesta sentença a palavra *sede* assume dois significados diferentes: o primeiro diz respeito ao substantivo sede e o segundo ao advérbio de modo [8, 26].

Uma maneira simples para a produção de voz sintética é a utilização de um banco de amostras de sons pré-gravados, como por exemplo, fonemas [18]. Entretanto de nada adianta um sintetizador capaz de concatenar fonemas com excelente qualidade se o analisador de texto não faz a análise correta.

Uma outra técnica que vem se destacando no reconhecimento, síntese e em muitas áreas do processamento de linguagem natural são os Modelos Ocultos de Markov (*Hidden Markov Model*), mais conhecidos como HMM [17, 9]. Trata-se de um processo duplamente estocástico, a primeira camada es-

tocástica é um processo de Markov de primeira ordem, e não é diretamente observável (por isso o nome oculto); a segunda camada e constituída por estados que contém uma observação probabilística de um processo de Markov. Isso significa que, dada uma seqüência de saída, não se sabe a seqüência de estados percorrida pelo modelo, mas somente uma função probabilística deste caminho. Segundo Oliveiras [17], os processos ocultos consistem de um conjunto de estados conectados por transições (autômato finito) com probabilidades, enquanto que os processos observáveis (não ocultos) consistem de um conjunto de saídas ou observações, cada qual podendo ser emitido por cada estado de acordo com alguma saída da função de densidade de probabilidade.

Em Mohri [11], uma representação modificada dos autômatos de estados finitos é descrita como uma forma comum, natural e direta para uso em um HMM. Este formalismo matemático representa e define uma estrutura que alia, o poder estatístico do HMM, com a consolidada teoria dos autômatos de estados finitos e é chamado de WFST (*Weighted Finite-State Transducers*). Algoritmos utilizados em autômatos de estados finitos podem ser aplicados nesta representação combinando flexibilidade e eficiência. Mohri [11], demonstra que o algoritmo clássico de minimização de autômatos otimizaria o tempo e o espaço requerido pela execução de um HMM.

Outra forma de síntese de voz pode ser realizada através de *tecnologia adaptativa*, mais especificamente *autômatos adaptativos*, que aproveitando de suas características suficientemente abrangentes na representação de linguagens dependentes de contexto, podem processar o texto escrito de forma a produzir uma transcrição fonética mais apurada, na qual teremos o som de cada fonema representado por um caractere ASCII. Uma vez o texto transcrito, basta fazer a união dos sons correspondentes a cada fonema, formando assim uma palavra ou sentença.

Nas próximas seções será feita uma introdução à tecnologia adaptativa, seguida de dispositivos guiados por regras adaptativos e o seu principal formalismo, os autômatos adaptativos.

3.2 Tecnologia Adaptativa

A teoria dos *dispositivos adaptativos* surgiu da busca de um formalismo de fácil utilização como, por exemplo, os autômatos de estados finitos e autômatos a pilha estruturados, e que sejam capazes de representar problemas de maior complexidade, de fácil implementação, eficiente em sua operação, e que possam ser utilizados em aplicações diversas.

A tecnologia adaptativa provê grandes vantagens quanto a sua facilidade

de uso e sua relativa simplicidade. Seu comportamento dinâmico faz com que possa ocorrer uma evolução do dispositivo de maneira incremental, ou seja, seu comportamento pode ser alterado dinamicamente à medida que os estímulos de entrada são recebidos. Trata-se de um formalismo dotado de recursos de auto-modificação, que lhe oferece a possibilidade de adquirir e representar conhecimento, incorporar informações e realizar atividades de aprendizagem, podendo ser utilizado como alternativa em muitos problemas envolvendo inteligência artificial, heurísticas, algoritmos genéticos, redes neurais, entre outros.

A propriedade de auto-modificação originou dispositivos mais poderosos, em relação a sua capacidade de expressão, que são obtidos a partir de uma progressão suave dos recursos oferecidos pelos dispositivos mais simples, generalizando assim uma série de dispositivos formais.

3.2.1 Dispositivos Guiados por Regras Adaptativos

O conceito de *dispositivo guiado por regras* generaliza a formalização de uma série de dispositivos que têm sua operação definida por um conjunto fixo e finito de regras (autômatos finitos, autômatos de pilha, máquinas de Turing, etc.). Essas regras mapeiam cada possível configuração do dispositivo em uma nova configuração, levando em consideração um determinado estímulo de entrada e gerando algum símbolo de saída [21, 19]. Este, inicia seu funcionamento em uma determinada configuração, e segue aplicando sucessivamente uma regra do seu conjunto de regras, alternando entre as possíveis configurações, até que não existam mais estímulos de entrada ou até que se atinja uma configuração à qual nenhuma regra possa ser aplicada [13, 19].

Em um dispositivo *guiado por regras adaptativo*, ou simplesmente *dispositivo adaptativo*, o conjunto de regras passa a poder variar durante a leitura dos estímulos de entrada. Esta variação, no entanto, é completamente determinada por um outro nível de regras. Neste segundo nível, as funções adaptativas agem sobre o conjunto de regras original, modificando-o através da remoção e inserção de novas regras. Temos assim um dispositivo com duas camadas, a primeira, denominada camada subjacente, é representada por um dispositivo guiado por regras (não-adaptativo) e pelas ações adaptativas, que correspondem às chamadas de funções adaptativas do segundo nível. Ao segundo nível, dá-se o nome de camada adaptativa, constituída pelo conjunto de funções adaptativas. Dessa forma, define-se um mecanismo universal, capaz de transformar um dispositivo qualquer, não adaptativo, mas guiado por regras, em um dispositivo capaz de alterar sua estrutura interna (conjunto de regras) durante sua operação, a partir do acréscimo da camada adaptativa.

As primeiras aplicações das tecnologias adaptativas, derivadas da teo-

ria dos *dispositivos adaptativos* [13], apresentavam-se na área de construção de compiladores e podem ser utilizadas também em diversas áreas da computação como processamento de linguagens naturais, robótica, visão computacional, problemas de reconhecimento de padrões e síntese de voz [19].

Na próxima seção, falaremos sobre o autômato adaptativo, um dos principais e mais utilizado dispositivo guiado por regra adaptativo, que posteriormente será utilizado como ferramenta para transcrição fonética de linguagens dependentes de contexto.

3.2.2 Autômatos Adaptativos

As necessidades relacionadas à busca de técnicas eficientes na construção de compiladores e outros problemas relacionados a linguagens, como a tradução texto-voz, através de vários artifícios que, empregados em conjunto com autômatos finitos e de pilha, procuram resolver problemas com características de linguagens estritamente sensíveis ao contexto. Tais artifícios tentam simplificar as linguagens sensíveis ao contexto, através de sua substituição provisória por linguagens livres de contexto que as contenham de uma maneira que tenha sido retirado somente as características dependentes de contexto [14].

Para suprir a falta das características dependentes de contexto, causadas pela simplificação imposta na linguagem, é necessário inserir estas características através da utilização de rotinas semânticas auxiliares, as quais são ativadas à medida que a cadeia de entrada é consumida. Este método não trata diretamente a sintaxe dependente de contexto, mas, delega esta responsabilidade ao tratamento semântico, assunto que nada tem de semântico, mas que por isso, tornou-se conhecida no jargão da área como "semântica estática" [14].

O formalismo proposto pelos *autômatos adaptativos*, pela sua extensão, generalidade, estrutura em camadas e adaptabilidade, oferece um modelo capaz de realizar tratamentos livre de contexto com eficiência, através de ações sintáticas, não necessitando de uma simplificação de tais linguagens.

Sua propriedade mais importante, a auto-modificação, de forma controlada, através de regras estabelecidas em sua estrutura, faz dos *autômatos adaptativos* um modelo capaz de representar conhecimento, incorporar informações e realizar atividades básicas de aprendizagem, além de sua simplicidade estrutural herdada dos autômatos estruturados, o tornam uma ferramenta muito eficiente no ensino de aspectos de inteligência artificial e linguagens.

Um autômato adaptativo é um dispositivo guiado por regras adaptativo em que a camada subjacente consiste de um autômato de pilha estruturado

e as ações adaptativas são implementadas através de *funções adaptativas*. As funções adaptativas determinam exatamente quais modificações devem ser realizadas na camada subjacente do dispositivo. No caso dos *autômatos adaptativos*, tais ações determinam quais transições do autômato deverão ser removidas ou inseridas. O núcleo de uma função adaptativa consiste de uma lista de *ações adaptativas elementares*, que podem ser de três tipos: *ações elementares de consulta (?)*, que possibilitam a busca de padrões na estrutura definida pelas regras da camada subjacente, e as *ações elementares de inserção (+)* e de *remoção (-)*, que determinam, respectivamente, as regras que deverão ser inseridas ou removidas do conjunto de regras corrente da camada subjacente [19].

A estrutura de uma função adaptativa pode ser descrita como uma 9-upla $FA = (F, P, V, G, C, R, I, A, B)$ onde:

- F É o nome da função adaptativa.
- P É uma lista de parâmetros formais.
- V É uma lista de identificadores de variáveis.
- G É uma lista de identificadores de geradores.
- C É uma lista de ações de consultas.
- R É uma lista de ações de remoção.
- I É uma lista de ações de inserção.
- A É uma ação adaptativa inicial (opcional).
- B É uma ação adaptativa final (opcional).

As ações adaptativas são descritas na forma $AA = (F', P')$, onde F' é o nome da função adaptativa e P' é a lista de argumentos a serem passados para F'. A execução deste tipo de dispositivo se dá inicialmente com a execução opcional de uma ação adaptativa inicial, após isto passa-se as ações elementares *(inserção, remoção e consulta)* e por fim, se necessário, as ações adaptativas finais [19].

3.2.3 AdapTools

Como alternativa para o aprendizado, implementação, experimentos e depuração de *autômatos adaptativos*, foi desenvolvido um ambiente baseado em software livre, chamado *AdapTools* (Figura 3.2). O núcleo deste sistema

é composto por uma máquina virtual que executa uma versão levemente modificada de um autômato adaptativo. Essas pequenas modificações no formalismo original simplificam e uniformizam o formato de especificação de transições internas, transições externas e das ações adaptativas elementares. Com isso, todos os elementos do dispositivo passam a poder ser apresentados através de uma única tabela [19]. Dentro destas pequenas modificações no formalismo, podemos citar uma propriedade importante dentro da teoria dos autômatos, o não-determinismo. Embora o AdapTools seja uma excelente ferramenta de manipulação de autômatos, algumas modificações foram necessárias, dentre elas a implementação de um recurso que permitiu a execução de autômatos não determinísticos, já que a ferramenta somente executava autômatos determinísticos. No Capítulo 5, é mostrado como este recurso foi implementado na ferramenta e como o usuário pode utilizá-lo.

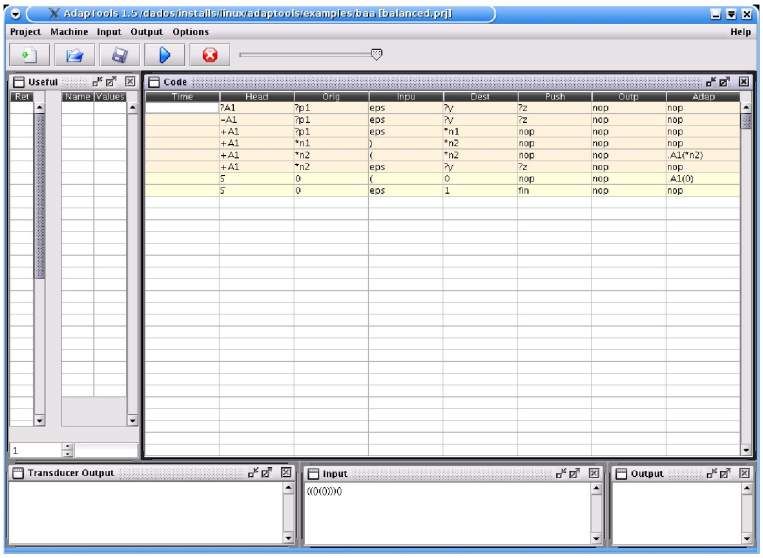

Figura 3.2: AdapTools: Ferramenta de apoio a implementação, depuração e execução de *autômatos adaptativos*

No AdapTools os autômatos são representados através da tabela de transição. Esta tabela é composta por sete colunas: a coluna (1) (*Head*) dá um nome à sub-máquina ou da função adaptativa contida na linha. Quando o conteúdo desta coluna for o nome de uma função adaptativa, estas devem ser identificadas pelas ações adaptativas elementares consulta (?), inserção (+) e remoção (−). A coluna (2) (*Orig*), bem como a coluna (3) (*Dest*), mantém identificadores que correspondem, respectivamente, aos estados de origem e destino da transição. A coluna (4) (*Inpu*) mantém o símbolo que pode ser lido a partir de um estado incluindo transições vazias (ϵ). A coluna

(5) (*Push*) contém um endereço de chamada de sub-máquina. A coluna (6) (*outp*) é utilizada para enviar uma saída para um transdutor. A coluna (7) (*Adap*) pode ser utilizada para a chamada de funções adaptativas [16, 20].

Visando facilitar a utilização da ferramenta AdapTools, foi implementada uma série de palavras reservadas, estas estão descritas abaixo:

eps: Representa cadeia vazia (ϵ).

nop: Nas colunas *Push*, *Outp* e *Adap*, indica que a função está inativa.

pop: Na coluna *Dest* indica uma transição de retorno de sub-máquinas.

fin: Na coluna *Push* indica que o estado de destino é um estado final.

spc: Na coluna *Inpu*, representa o caractere espaço da tabela ASCII.

digit: Representa o intervalo [0..9].

letter: Representa os intervalos [a..z] e [A..Z]

special: símbolos diferentes de letras e números.

other: Símbolo que não possa ser consumido estando no estado de origem da transição.

a..z: Faixa de valores ASCII de uma letra até uma outra.

A Figura 3.3 demonstra a estrutura de uma autômato adaptativo de estados finitos para produzir o balanceamento entre parênteses. Na Figura 3.3 é apresentado inicialmente dois estados *q0* e *q1*, o estado *q0* (inicial) apresenta um *"loop"* para a entrada *"("* que chama a função adaptativa *.A*, e uma transição para a *q1* quando a entrada for a cadeia vazia (ϵ). Durante a execução deste autômato, para toda entrada *"("* a estrutura inicial do autômato é replicada internamente na estrutura atual do mesmo, fazendo com que a cada parêntese aberto haja um novo estado, acessível através de uma transição *")"*, o que nos garante que se o autômato chegou ao estado final necessariamente este contém um fecha parênteses *")"* para cada abre parênteses *"("* lido [21, 26].

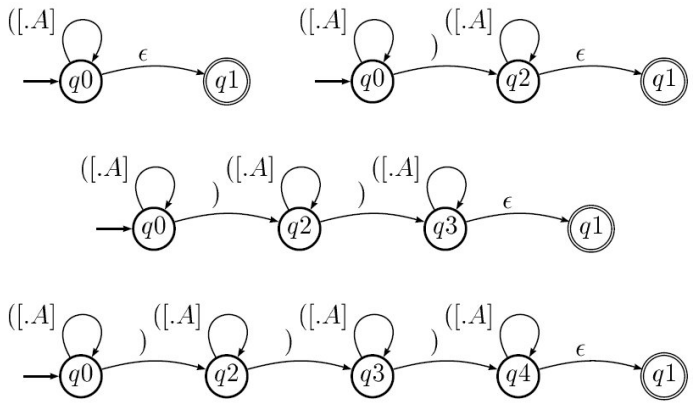

Figura 3.3: Execução de um autômato adaptativo de balanceamento de parênteses para a entrada: *((()))*.

3.3 Engenharia de Software

O desenvolvimento do tradutor texto-voz, como todo software, é uma tarefa complexa [3], que demanda tempo e organização. A necessidade de descrever de forma simples, porém, eficaz o sistema de tradução de texto-voz, impõe a necessidade de criação de *modelos*. O conceito chamado *modelo*, pode ser visto como uma representação dos detalhes, comportamentos, das relações entre as partes envolvidas, e principalmente deve ser visto como uma simulação onde diferentes soluções podem ser experimentadas e problemas podem ser detectados.

Os *modelos* são notações gráficas que seguem algum padrão lógico e possuem algum significado definido, também chamados de *diagramas*. Seu principal valor está na comunicação e entendimento. Um bom diagrama, freqüentemente pode ajudar a transmitir idéias sobre um projeto, particularmente quando você quer evitar muitos detalhes. Os diagramas também podem ajudá-lo a entender um sistema de software ou um processo de negócio. Como parte de uma equipe tentando descobrir algo, os diagramas ajudam toda a equipe a entender como comunicar esse entendimento [5].

Desses *modelos*, uma das mais importantes e utilizadas pela comunidade está a UML(*Unified Modeling Language*). A importância da UML é proveniente de seu uso amplo e da padronização dentro da comunidade de desenvolvimento orientado a objetos. A UML se tornou não somente a notação gráfica dominante dentro do mundo orientado a objetos, como também uma técnica popular nos projetos não-orientados a objetos.

A UML é um padrão OMG (*Object Management Group*) que atua como

linguagem de modelagem para o desenvolvimento de sistemas orientados a objetos. Ela define uma notação que consiste em elementos gráficos que você pode usar para modelar estruturas de classes, ações do sistemas além da colaboração entre os elementos da aplicação [5]. Em outras palavras, a notação é a sintaxe da linguagem de modelagem.

Embora o padrão UML defina como você representa itens, ele não estipula o processo usado para criar modelos. Isto é, a UML não impõe os passos que você deve dar para criar um modelo, entretanto, existem processos que devemos usar, como RUP(*Rational Unified Process*). Como alternativa, podemos utilizar nosso próprio processo, o qual implementa partes da UML que sejam convenientes para nossas necessidades em particular.

Nas próximas sessões veremos os diagramas mais usados da UML, suas características principais, sintaxe e semântica envolvidas na notação. Tais diagramas serão utilizados para documentação e descrição das partes do sistema de tradução texto-voz.

3.3.1 Diagramas de Caso de Uso

Um diagrama de caso de uso permite a modelagem das interações entre um usuário e um sistema. Um caso de uso consiste em um ou mais cenários, em que um cenário é uma seqüência de passos dados quando um usuário interage com o sistema. Os limites que definem os casos de uso e cenários freqüentemente se confundem, portanto, podemos dizer que certo grupo de interações formam um único caso de uso, mas outra pessoa poderia agrupar essas interações em mais de um caso de uso. Qualquer uma das estratégias está correta, pois fica por conta do modelador usar seu discernimento ao decidir que forma um caso de uso pode tomar.

A notação para caso de uso é simples, como a mostra a Figura 3.4.

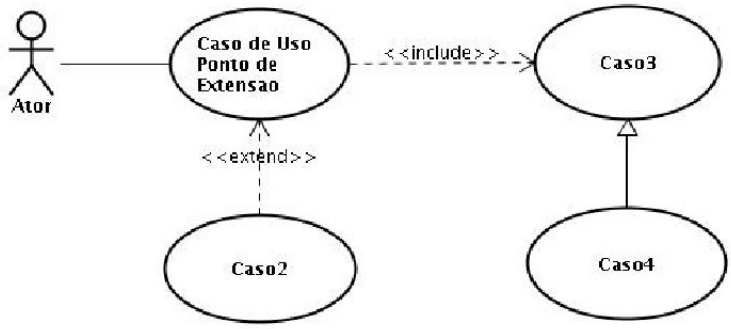

Figura 3.4: Notação de caso de uso

O ator é um indivíduo ou outro sistema que interage com o sistema, ou podemos dizer que ele põe em prática o caso de uso. Um único caso de uso pode ter vários atores ou um único ator. Do mesmo modo, um único ator pode executar vários casos de uso.

A notação <<*include*>> evita a repetição, pois ela permite a cópia do comportamento de um ou mais casos de uso. A Figura 3.4 mostra o caso de uso em que *Caso3* é incluído, evitando assim a repetição.

A notação generalização na Figura 3.4, permite a descrição de uma variação no curso normal do comportamento de uma caso de uso. Por exemplo, ainda na Figura 3.4, é mostrado o uso da notação em questão, informa que o caso de uso *Caso4*, sofrerá uma especialização de *Caso3*.

Uma notação mais rígida do que a generalização é a notação <<*extend*>>. Esta ilustra uma situação em que um caso de uso tem a mesma funcionalidade de outro, mas estende ou sobrepõe partes desta funcionalidade. Para mostrar isso, devemos listar os itens dentro do caso de uso de base que o caso de uso de extensão pode estender. Além disso, se o caso de uso de extensão estende alguns destes itens (não é obrigatório estender cada item oferecido), deve-se colocar os nomes desses itens entre parênteses, por meio da notação <<*extend*>>, como mostra a Figura 3.5.

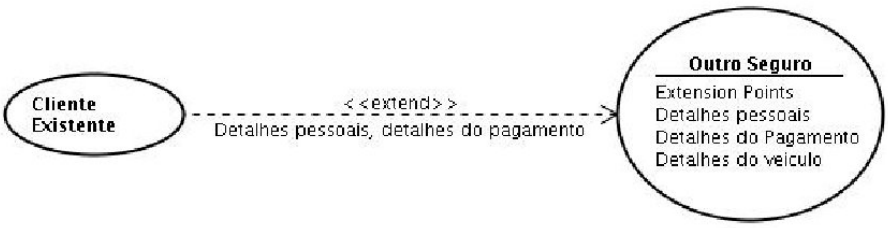

Figura 3.5: Notação <<extend>>

3.3.2 Diagramas de Classe

Estes diagramas mostram diferentes classes de um sistema e a maneira com que elas se relacionam. Em parte, o diagrama faz isso mostrando os atributos e operações de cada classe dentro do sistema e definindo as restrições impostas a essas classes. A notação básica para mostrar uma classe é uma caixa retangular contendo o nome da classe. Entretanto, você pode adicionar mais informações à caixa.

A notação UML para diagramas de classes é ampla, portanto, mostraremos apenas os elementos mais usados. Especificamente, veremos cinco tipos

principais de notação: associações, atributos, operações, generalização, restrições.

Associações

Utilizado para ilustrar quando uma instância de uma classe tem um relacionamento com uma outra instância de classe, podemos unir as duas classes com uma linha, conhecida como associação. Cada extremidade da linha é conhecida como papel. Podemos definir melhor os relacionamentos das classes por meio de multiplicidade e navegabilidade. A multiplicidade mostra quantos objetos podem participar de determinado relacionamento. Por exemplo, na Figura 3.6, um cliente pode ter entre zero e infinitos planos.

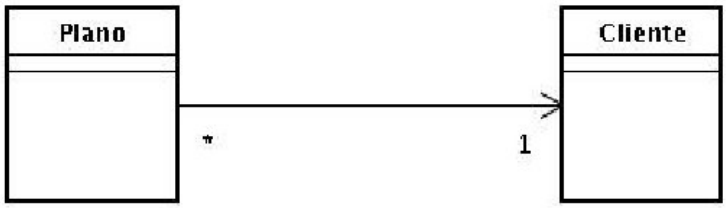

Figura 3.6: A associação

Existem quatro valores possíveis para ilustrar a multiplicidade:

1 - Uma instância.

* - Zero a infinitas instâncias.

0..1 - Zero ou uma instância.

m..n - Definido pelo usuário entre m e n.

A navegabilidade mostra as responsabilidades das classes. A seta aponta para classe que não é responsável pelo relacionamento. Por exemplo, o plano deve informar à qual cliente ele pertence, Figura 3.6, mas o cliente não precisa dizer quais instâncias do plano ele tem.

Atributos

Os atributos tem diferentes significados, dependendo do nível em que estivermos modelando. Em nível conceitual, um atributo define simplesmente que uma classe tem determinado recurso, como a classe Veículo tendo um número de licença. Em um nível de especificação, um atributo indica que

um objeto Veículo tem uma maneira de configurar o valor de sua licença e que também pode fornecer à você o valor dessa licença. Por fim, no nível de implementação, um atributo indica que um veículo tem um campo para a licença.

A Figura 3.7 mostra os nomes dos atributos, mas poderia mostrar mais detalhes. A UML a seguir define o que podemos usar para configurar um atributo.

visibilidade nome: tipo = valorPadrão

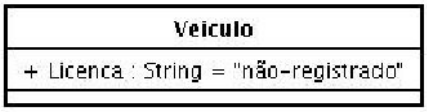

Figura 3.7: Notação de atributo

A visibilidade do atributo pode ser:

Pública - preceda o nome com + (sinal de adição)

Protegida - preceda o nome com # (cerquilha)

Privada - preceda o nome com - (subtração)

Em alguns casos talvez desejemos mostrar a multiplicidade de um atributo, possivelmente para mostrar se o sistema a exige. Para fazer isso, colocamos o valor da multiplicidade entre colchetes após o nome do atributo, como se vê a seguir:

 # licença[1]: String = "não-regitrado"

Operações

Operações são os algoritmos que uma classe executa e, novamente, elas podem ter diferentes significados em diferentes níveis. No nível conceitual, as operações ilustram as responsabilidades de uma classe. Em um nível de especificação, elas devem corresponder aos métodos com visibilidade pública. Por fim, no nível de implementação elas podem corresponder aos métodos com qualquer grau de visibilidade. A Figura 3.8 ilustra que você mostra operações na sessão inferior da caixa que representa a classe.

Você pode fornecer muitas informações sobre a operação dentro da caixa. Para fazer isso, use as partes da sintaxe UML necessárias:

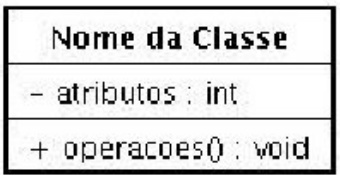

Figura 3.8: Notação de operações

visibilidade nome(lista de parâmetros): expressão_de_tipos_de_retorno { *Strings_de_propriedades* }

Como podemos ver, a sintaxe é semelhante aquela usada para mostrar um atributo, mas com algumas exceções notáveis. O termo expressão_de_tipos_de_retorno indica uma lista separada por vírgulas de tipos de retorno (UML permite múltiplos tipos de retorno). O item lista de parâmetros, mostrado entre parênteses, indica uma lista separada por vírgulas, de parâmetros, cuja sintaxe é a seguinte:

direção nome: tipo = valor padrão

As diferentes partes de sintaxe têm os mesmos valores vistos anteriormente, com exceção da direção. Isso mostra se um parâmetro é usado para entrada, saída ou entrada e saída. Os valores para indicar a direção são os seguintes:

in - parâmetro usado apenas para entrada, esse é o valor padrão.

out - Parâmetro usado apenas para saída.

inout - parâmetro usado para entrada e saída.

Por fim, para mostrar com toda esta sintaxe se encaixa, o exemplo a seguir apresenta uma operação protegida que aceita dois parâmetros de entrada e retorna um objeto String:

 # minhaoperacao(arg1: String, arg2: Inteiro=0): String

Generalização

A Figura 3.9 mostra que uma seta grande e vazia representa a generalização. Assim como uma associação, uma generalização mostra um relacionamento

entra duas classes. Entretanto, ao contrário de uma associação, ele mostra um tipo especial de relacionamento, em que uma classe é a filha de outra classe. Em outras palavras, uma classe ou um subtipo de outra classe.

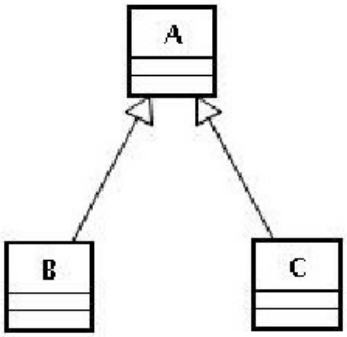

Figura 3.9: Notação de generalização

Restrições

Nós vimos várias maneiras de modelar os relacionamentos entre classes. Freqüentemente, esses relacionamentos atuam de modo que restrinjam as classes dentro desse relacionamento. Por exemplo, a multiplicidade restringe uma classe em termos de quantas instâncias dela podem existir. A UML também fornece sintaxe que permite mostrar restrições, as quais, de outro modo, são ocultadas. A sintaxe é a seguinte:

{ *descrição da restrição* }

Conectamos a descrição da restrição a uma classe usando uma linha tracejada. A descrição da restrição pode assumir a forma que quisermos, mas se quisermos uma sintaxe formal, poderemos usar a OCL (*Object Constraint Language*). O exemplo que a Figura 3.10 mostra, usa simplesmente texto puro.

3.3.3 Diagramas de Seqüência

Os diagramas de classes mostram como as classes interagem umas com as outras, mas não dão nenhuma idéia de como as instâncias destas classes irão interagir em uma situação real. Em vez disso, os diagramas de seqüência modelam esse tipo de comportamento. Esses diagramas fazem isso mostrando objetos e as mensagens que passam entre esses objetos. Vamos aprender a

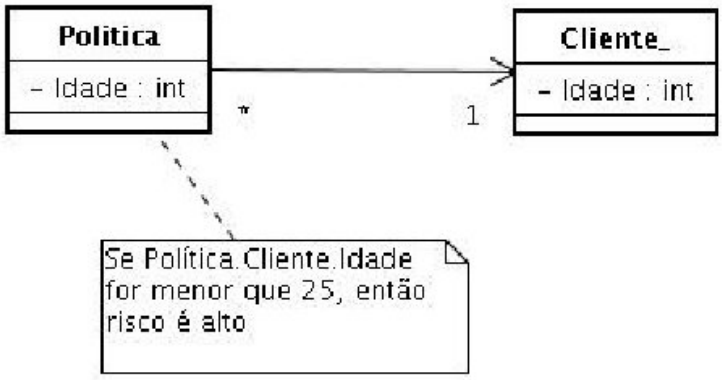

Figura 3.10: Notação de restrições

sintaxe UML para digramas de seqüência explorando um exemplo, apresentado na Figura 3.11.

Este exemplo mostra a maioria da sintaxe UML comumente usada para diagramas de seqüência. Colocamos na parte superior do diagrama os atores e caixas que representam objetos ou classes, quando seus métodos estáticos são acessados. Abaixo desses itens existem linhas verticais tracejadas, que representam a vida do item. Essas linhas são conhecidas como linha de vida. Os retângulos através dos quais essas linhas passam são ativações, e elas representam o tempo que os objetos requerem para fazer seu trabalho. As setas horizontais representam mensagens (a comunicação entre os itens que você modela).

A Figura 3.11 mostra um diagrama de seqüência no qual um usuário quer acessar algum serviço do sistema que exige autenticação. O usuário envia uma mensagem para o objeto controlador, informando que deseja acessar o sistema. Esse objeto cria uma instância da classe autenticador. Então, o usuário fornece a este objeto seu login e sua senha. O objeto autenticador recebe essas credenciais e chama o método estático éEmpregado() da classe Empregado. Observe como neste caso o rótulo da mensagem é o nome do método, junto com uma lista de parâmetros, separados por vírgula. O método éEmpregado(), realiza alguma forma de processamento dos parâmetros. Na realidade, isso poderia envolver a pesquisa de informações em um banco de dados. Entretanto, o exemplo está simplificado neste caso, portanto, suponha que este método não exija os serviços de nenhuma outra classe. Notaremos que até este ponto, toda a sintaxe UML é igual aquela mostrada no exemplo anterior

Após o método éEmpregado() terminar seu trabalho, ele retorna um objeto empregado para o objeto autenticador. Como você pode ver, a sintaxe

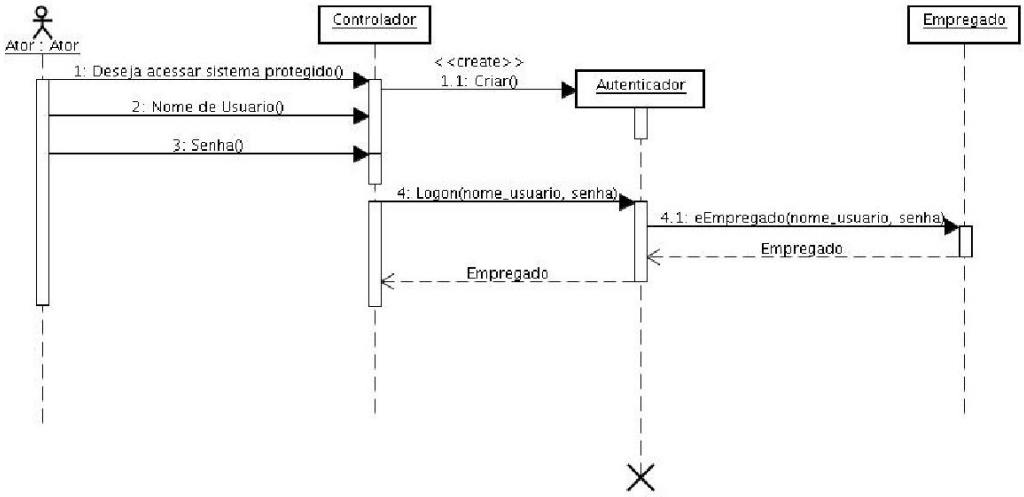

Figura 3.11: Diagrama de sequência 2

UML para isso é uma linha tracejada horizontal que aponta para a classe ou objeto chamador e um rótulo opcional que indica o que é retornado. Nesse ponto, o objeto autenticador retorna o objeto empregado para o objeto controlador. Então, ele envia uma mensagem <<*destroy*>> para si mesmo. Esse tipo de mensagem é conhecido como auto chamada e é mostrado com um X ao final de sua linha execução.

Todas as mensagens mostradas até aqui utilizam setas cheias, isso significa que as mensagens são síncronas (o chamador suspende a operação até que o objeto chamado retorne). Na realidade, talvez queiramos modelar situações em que sejam exigidas mensagens assíncronas. Por exemplo, podemos ter um objeto que crie linhas de execução, para que possa prosseguir com algum outro processamento. Para marcar esses tipos de mensagens, mostramos uma meia seta, em vez de uma seta cheia.

Capítulo 4

Ambiente Integrado para Desenvolvimento de Autômatos Adaptativos

Neste capítulo apresentaremos uma proposta e o desenvolvimento da integração do AdapTools com o compilador para autômatos adaptativos chamado, AdapMap. A linguagem utilizada neste compilador oferece uma interface de auto nível para codificação e resolve algumas dificuldades encontradas na descrição e análise estrutural dos autômatos. Para facilitar essa integração, faremos uma análise sobre os aspectos de implementação do AdapTools, que também será útil para integração e desenvolvimento de novas funcionalidades.

A análise da estrutura do AdapTools apresentará elementos do projeto, além de apresentar uma grande e importante melhoria realizada no tratamento de ações semânticas. Nas seções seguintes discutiremos os principais componentes da interface do AdapTools, e a integração com outros sistemas. Na seção final, exibiremos também, uma nova linguagem para a representação dos autômatos adaptativos e será mostrado o compilador que converte essa linguagem de alto nível para a notação utilizada pelo AdapTools.

4.1 Organização do Código do AdapTools

Uma análise mais profunda sobre os aspectos de implementação do AdapTools é útil para facilitar a integração e desenvolvimento de novas funcionalidades para futuros desenvolvedores. Uma característica fundamental deste projeto é a divisão conforme as camadas especificadas na tecnologia adaptativa, mecanismo subjacente e mecanismo adaptativo. Além das duas camadas

base do mecanismo adaptativo, o adaptools conta com uma terceira, que diz respeito à estrutura de dados utilizada pelas camadas adjacente e subjacente.

Segundo [19] o AdapTools encontra-se dividido em três camadas bem específicas: máquina virtual que implementa o mecanismo subjacente (*Kernel.java*), o mecanismo adaptativo (*Adapter.java*) e os tipos de dados de apoio a implementação são constituídos pela Virtual Machine Data Structures (*Vmds.java*), que é implementado por outras duas classes: a *Viewer.java*, que implementa a parte gráfica do AdapTools e a *VmdsImpl.java*, que dá suporte ao ambiente textual (*Runner*) [21].

4.1.1 Mecanismo Subjacente

A implementação do mecanismo subjacente (*Kernel.java*) atua como uma máquina virtual. Sua principal função é executar autômatos que podem ser autômatos de pilha estruturados ou autômatos adaptativos. Esta detém as estruturas que armazenam as informações do autômato, através dos tipos abstratos de apoio à execução da máquina, como por exemplo: a pilha de chamadas de submáquinas, a tabela que armazena o autômato, o estado corrente, entre outros, que são fornecidas pela interface *Vmds.java*.

Outra característica relevante da camada subjacente é a identificação da ação semântica a ser realizada pela execução do autômato. Esta identificação é realizada através das configurações do arquivo de projeto do autômato e listagem de ações semânticas contidas no diretório *semantics*, localizado junto ao arquivo executável do AdapTools.

4.1.2 Mecanismo Adaptativo

O mecanismo adaptativo é responsável pela lógica que torna o autômato um dispositivo dinâmico, através de algumas alterações na definição original das funções adaptativas proposta em [19]. A realização das funções de pesquisa, remoção e adição, na implementação do AdapTools, são executadas na respectiva ordem, esta estratégia evita que variáveis não inicializadas sejam usadas em remoções e adições. O formato tabular no qual o autômato é descrito não considera questões de desempenho, fazendo diminuir o andamento da execução de autômatos adaptativos com grande número de regras e funções adaptativas. Uma prova completa da complexidade do algoritmo de execução de função adaptativa [19], mostra que o custo, no pior caso, é da ordem de $O(n^k)$, sendo k o número de ações elementares de consulta e n o tamanho da lista de transições.

4.1.3 Rotinas Semânticas

Ao executar um autômato, a máquina virtual, antes de enviar o símbolo para a saída, faz chamada à um método, específico do objeto semântico definido no projeto do autômato, este método é pré-definido e denominado *execute*. Este método recebe o símbolo de saída, e pode utilizar os recursos de toda a API (*Aplication Programing Interface*) da linguagem JAVA para analisar e gerar os eventos necessários conforme a aplicação, ao final retorna uma cadeia de caracteres (pode ser vazia), que será impressa na saída auxiliar de texto do AdapTools.

Uma melhoria importante foi concebida no método de adição de rotinas semânticas. O pacote AdapTools dispõe de uma interface, *adaptools.vm.Semantics*, que através da simples implementação de seus métodos, gera novos módulos para tratamento de rotinas semânticas. Após a implementação e compilação do novo módulo, é apenas necessária a cópia da nova classe semântica para o local específico da ferramenta, no diretório *semantics* localizado junto ao arquivo executável do AdapTools.

As rotinas semânticas disponíveis na ferramenta AdapTools podem ser visualizadas no menu *Project*, simplificando a seleção da ação semântica a ser utilizada pelo autômato. A Figura 4.1 mostra a seleção da ação semântica.

4.1.4 Estrutura de Dados da Máquina Virtual

Representada pelo arquivo *Vmds.java*, a estrutura de dados da máquina virtual é uma interface (*Java Interface*), que no AdapTools é implementada duas vezes, uma para a parte texto e outra para a parte gráfica, *VmdsImpl.java* e *Viewer.java* respectivamente. Esta tem como função prover uma ligação entre a máquina virtual e a estrutura de armazenamento do autômato, através de métodos que manipulam e acessam os dados.

4.2 Interfaces do Ambiente Integrado

A ferramenta de apoio à implementação de software adaptativos AdapTools, apresenta ao usuário dois tipos de interface: Uma interface gráfica, descrita na seção 4.2.1, e uma modo texto, seção 4.2.2. O modo gráfico propicia ao usuário uma maior facilidade para compreensão, codificação e gerência de projetos. O modo texto é uma interface que visa a integração com outros sistemas.

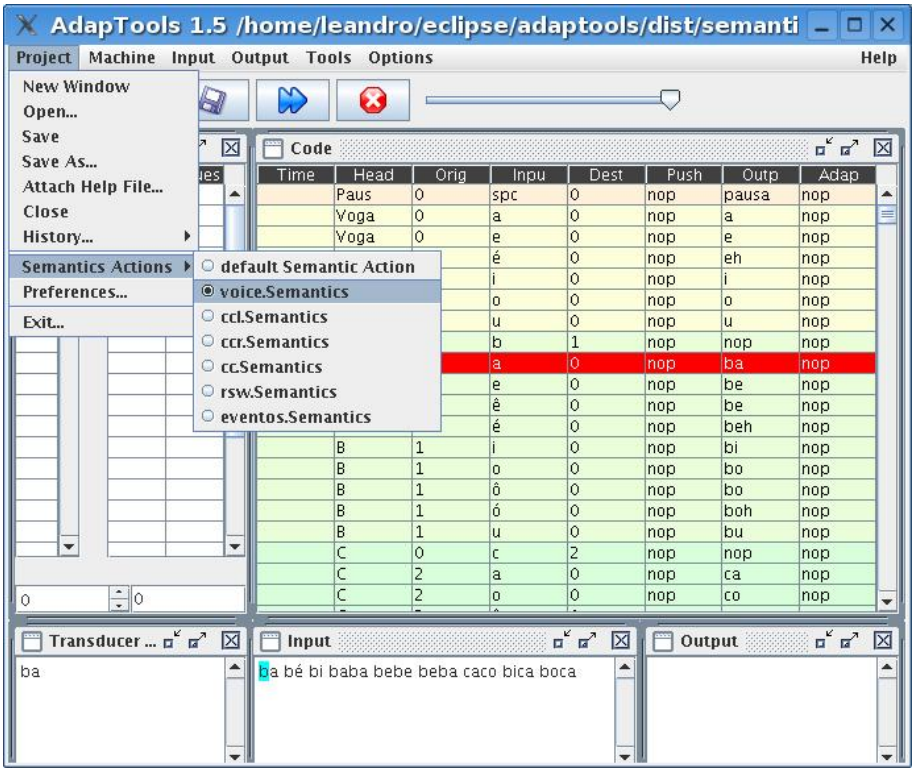

Figura 4.1: AdapTools: Seleção de ação semântica

4.2.1 Modo Gráfico

A Figura 4.2, mostra a janela principal do AdapTools, com os principais componentes destacados com marcadores (letras de A até J) que representam os principais componentes gráficos da ferramenta.

As três caixas de texto independentes, marcadas de A até C representam respectivamente a cadeia de entrada, cadeia de saída para os autômatos que funcionam como transdutores, e uma cadeia auxiliar de texto que é manipulado por uma rotina semântica associada as regras de transição do autômato. Para marcar a parte da cadeia de entrada já processada durante a execução do autômato é utilizada uma cor diferente como *background* da *substring* [20].

O marcador D corresponde a área de codificação para o código objeto AdapTools. Essa área é formada por uma tabela de transições similar a mostrada na Figura 4.5. Outro ambiente para a codificação de autômatos disponível na ferramenta adaptools é o AdapMap, que na Figura 4.2 é representado pelo marcador E.

Representada pelo marcador F a barra de utilidades contem campos para mostrar os estados finais e o estado atual, uma pilha para mostrar as chama-

das de sub-máquinas, e uma matriz de duas colunas para mostrar o conteúdo de variáveis instanciadas nas ações elementares de consulta.

A barra de ferramentas, representada pelo marcador *G*, contém botões que tem a função de abrir uma nova tela do AdapTools para uma nova codificação, abrir um novo projeto na janela corrente, salvar o projeto, iniciar a execução da máquina aberta, parar a execução de uma máquina e coloca-la no estado inicial. Nesta barra de ferramentas encontra-se também um *slider* para controle de velocidade da execução do autômato, que quando em seu valor mínimo, coloca a máquina no estado de execução passo-a-passo.

O marcador *H* representa a barra de menus, que agrega várias operações em arquivos, conexão de máquinas, configuração de opções e seleção de rotinas semânticas (marcador *J*) semânticas.

Além dos itens compreendidos nos parágrafos acima, a interface gráfica do Adaptools conta também com um visualizador de autômatos mostrado na seção 4.2.1.

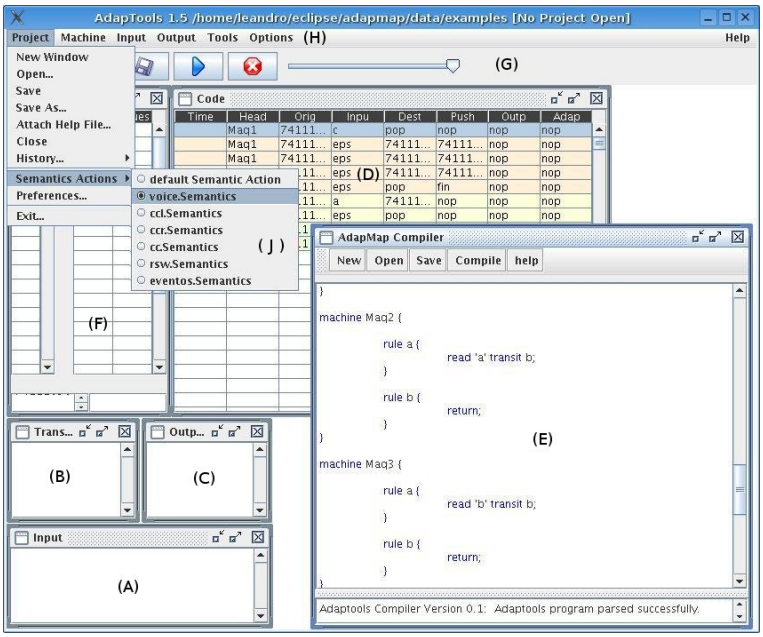

Figura 4.2: AdapTools:Ambiente Integrado com o compilador AdapMap

Visualização dos Autômatos

O acompanhamento da execução dos autômatos na parte gráfica pode ser melhor seguido com a ajuda de um gerador de visualizações de grafos levemente modificado para prover uma melhor visualização de autômatos. O

visualizador utilizado na ferramenta é o OpenJGraph. Este se encontra devidamente compactado em formato *jar* dentro da pasta lib, como mostra a Figura 4.3 da árvore de diretórios da ferramenta.

Figura 4.3: AdapTools:Árvore de diretórios do AdapTools

A Figura 4.4 mostra uma visualização do autômato para reconhecimento de parênteses, que foi mostrado na seção 3.2.3 pelas figuras 4.5 e 3.3, utilizando o OpenJGraph.

4.2.2 Modo Texto

Além do ambiente gráfico, o AdapTools conta com uma interface modo texto, que pode ser executado por um console em um ambiente texto ou mesmo em um ambiente gráfico. Esta interface pode ainda ser executada a partir de outro programa. Em programas desenvolvidos na linguagem Java, há a possibilidade de utilizar o recurso de herança para fornecer ao software em desenvolvimento o poder de executar as rotinas executadas pela classe *Runner*, classe responsável pela execução em modo texto.

Em uma execução modo texto os seguintes parâmetros podem ser utilizados:

- **-m** [Arquivo onde se encontra o código objeto AdapTools da máquina a ser executada] (Obrigatório).
- **-s** [Ação Semântica]

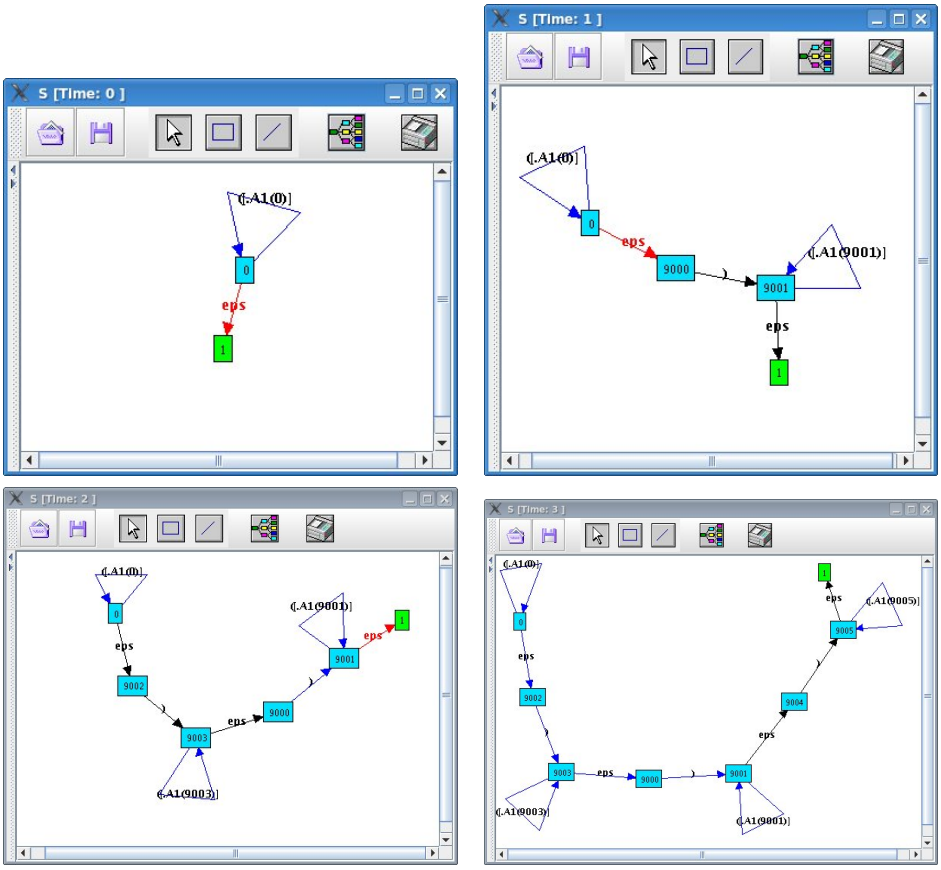

Figura 4.4: Visualização da execução de um autômato de balanceamento de parênteses através do OpenJGraph

- **-i** [Arquivo onde se encontra a cadeia de entrada a ser processada] (Obrigatório).
- **-o** [Arquivo de saída da máquina] (Obrigatório)
- **-p** [Arquivo onde se encontra a descrição do projeto] (Substitui -m, -i and -o).
- **-t** [Arquivo para a uma saída de Transdução] (opcional).
- **-im** [Arquivo onde se encontra a máquina de entrada para a execução] (opcional).
- **-d** [nível de debug, variando de 0 até 9.] (opcional), este tem como padrão o nível 1.

- **-n** [Tipo de tratamento de retorno das execuções não-determinísticas] (opcional).

Abaixo seguem dois exemplos de utilização do AdapTools modo texto em um console Linux, o primeiro mostrando a passagem dos principais parâmetros para execução, e o segundo fazendo apenas a passagem do parâmetro que diz respeito ao projeto.

```
$ java adaptools.vm.Runner -m sintatico.spa -i lista.in -o
  lista.out -im lexico.spa

$ java adaptools.vm.Runner -p sintatico.prj
```

4.3 Linguagem para Mapeamento de Autômatos

A linguagem usada para a criação de autômatos adaptativos na ferramenta AdapTools é a forma de interação entre o usuário e a ferramenta. Devido a alta complexidade da linguagem AdapTools na construção de autômatos adaptativos, difícil manutenção, abstração complexa, tempo elevado para seu aprendizado e a interface que representa o autômato em forma de tabela o que dificulta sua manipulação, foi desenvolvida uma nova linguagem para amenizar e resolver algumas dessas dificuldades além de contemplar características como: fácil utilização, instruções simples, fácil abstração e principalmente oferecer análises que diminuam erros ocorridos no desenvolvimento do autômato.

4.3.1 Linguagem AdapTools

No capítulo 3 apresentamos o AdapTools e a notação utilizada para codificação de autômatos em sua interface. Nesta seção apresentaremos um autômato exemplo mapeado na linguagem AdapTools, esse autômato será posteriormente mapeado na linguagem AdapMap, na Seção 4.3.2, para podermos confrontar e verificar as diferenças entre as linguagens de descrição da estrutura do autômato.

4.3.2 Linguagem AdapMap

Como todo compilador de linguagem de alto nível, o AdapMap procura minimizar os problemas decorrentes da codificação mostrando ao usuário

Head	*Inpu*	*Read*	*Dest*	*Push*	*Outp*	*Adap*
Machine1	a	x	b	nop	nop	nop
Machine1	a	z	c	nop	Estou_Aqui	nop
Machine1	a	w	a	b	Chamada_de_submachina_1	nop
Machine1	a	digit	n	d	Chamada_de_submachina_2	nop
Machine1	a	special	m	a	nop	nop
Machine1	a	eps	n	nop	nop	nop
Machine2	h	x	g	nop	nop	nop
Machine2	h	eps	i	nop	nop	nop
Machine2	i	digit	pop	nop	nop	nop
Machine2	g	eps	h	nop	nop	nop
Maquina3	n	x	o	nop	nop	Nop
Maquina3	n	z	m	nop	nop	nop
Maquina3	m	digit	pop	nop	Maquina3_legal	nop
Maquina3	m	w	p	nop	nop	nop
Maquina3	p	eps	pop	nop	Maquina3_Chata	nop

Figura 4.5: AdapTools: Tabela ilustrando formato do código AdapTools

possíveis erros e fatos indesejáveis na construção do código, através de análises sintáticas e semânticas gerando um código objeto (linguagem AdapTools) sem erros, o que não ocorre na linguagem AdapTools, que permite ao usuário a inserção de codificação inválida ou sem efeito semântico.

Uma característica adicional, de alta relevância, da linguagem AdapMap, corresponde a possibilidade de inserção de comentários sem efeitos semânticos no interior do código, isso possibilita a construção de um código com maior legibilidade, o que torna o processo didático e de desenvolvimento mais fáceis, contribuindo assim com um dos principais propósitos do AdapTools, ensino de tecnologia adaptativa.

O reconhecimento da linguagem que o autômato representa percorre uma trajetória no espaço de máquinas de estados [14]. A separação entre as máquinas na linguagem AdapMap, torna mais fácil a abstração entre as chamadas de sub-máquina do autômato estruturado. Os retornos de sub-máquinas estão mais fáceis de ser identificados devido a notação empregada, possibilitando uma melhor estruturação do código e conseqüentemente melhorando o entendimento.

Algumas considerações importantes da estrutura gramatical da linguagem AdapMap são: a identificação dos estados finais ocorre nas linhas iniciais de cada máquina; a organização e agrupamento de máquinas e regras são flexíveis; as várias possibilidades de descrição de regras extinguiram a identificação de parâmetros não utilizados, mais especificamente, a utilização da instrução **nop** na linguagem AdapTools.

Abaixo apresentaremos um modelo estrutural do código AdapMap, este, quando compilado gera o código objeto AdapTools da Tabela 4.5, as construções gramaticais estão comentadas visando maior entendimento.

```
/*Declaração da máquina, a palavra reservada "main" indica
```

```
 que esta máquina é a principal */
machine main Maquina1 {

    /*Declaração do estado inicial, se for omitido conside-
    ra-se o estado da primeira regra como inicial*/
    init a;
    /*Declaração dos estados finais*/
    final e , f;

    state a {
        /*Estando no estado "a" lendo um "x" vou para o es-
        tado "b"*/
        read 'x' transit b;

        /*Estando no estado "a" lendo um "z" vou para o es-
        tado"c" e imprime na saída padrão*/
        read 'z' transit c output "Estou aqui";

        /*Estando no estado "a" lendo um "w", faz uma chama-
        da a sub-máquina "Maquina1" dizendo que o estado ini-
        cial da sub-máquina deve ser o "a" e o retorno da sub-
        maquina deve levar ao estado "b" e imprime na saída
        padrão */
        read 'w' transit machine Maquina1.a,b output
            "Chamada de sub-maquina1";

        /*Estando no estado "a" lendo um digito, faz uma cha-
        mada a sub-máquina "Maquina2" sendo seu estado inicial
        o especificado em sua configuração e o retorno da sub-
        maquina deve levar ao estado "d" e imprime na saída
        padrão */
        read digit transit machine Maquina2,d
            output "Chamada de sub-maquina2";

        /*Estando no estado "a" lendo palavra vazia, faz
        uma chamada a sub-máquina "Maquina3" dizendo que o es-
        tado inicial da sub-máquina deve ser o "m" */
        read eps transit machine Maquina3.m ;

        /*Estando no estado "a" faz uma chamada a
sub-máquina "Maquina3" sendo seu estado
```

```
        inicial o especificado em sua configuração. Como o co-
        mando "read" foi omitido assume-se que é uma palavra
        vazia*/
        transit machine Maquina3;
    }
} /*fim da Maquina1*/

/*Declaração da máquina */
machine Maquina2 {
    /*Declaração do estado inicial */
    init g;

    state h {
        read 'x' transit g;
        transit i;
    }

    /*Retorna para a chamada de sub-máquina */
    state i {
        read digit return;
    }

    state g {
        read eps transit h;
    }
} /*fim da Maquina2*/

/*Declaração da máquina */
machine Maquina3 {
    state n {
        read 'x' transit o;
        read 'z' transit m;
    }

    state m {
        /*Retorna para a chamada de sub-máquina e imprime
        na saída padrão*/
        read digit return output "Maquina3 legal";
        read 'w' transit p;
    }
```

```
    /*Retorna para a chamada de sub-máquina e imprime na
    saída padrão Como o comando "read" foi omitido assu-
    me-se que é uma palavra vazia*/
    state p {
        return output "Maquina3 chata";
    }
} /*fim da Maquina3*/
```

4.4 Integração AdapTools e AdapMap

A integração dos softwares foi realizada sem problemas. O compilador AdapMap é um software independente que possui sua própria interface gráfica e eventos implementados. A partir de extensões de algumas classes foi possível integrar a interface gráfica do compilador ao ambiente do AdapTools, reutilizando de maneira eficiente todas as suas funcionalidades já implementadas. O botão *New* teve sua ação padrão reimplementada, e agora é utilizado para abrir um editor em branco dentro do próprio ambiente AdapTools. O botão *Compile*, teve sua ação reimplementada, para que ao final de uma compilação, sem erros, a tabela de codificação da linguagem AdapTools, fosse preenchida com código objeto resultante da compilação. Para a utilização do AdapMap utiliza-se o menu "Tools"selecionando a opção "AdapMap Compiler", como mostrado na Figura 4.6, dentro do *desktop* do AdapTools surgirá uma nova janela com o editor do AdapMap, exibido na Figura 4.2 e representado pelo marcador E.

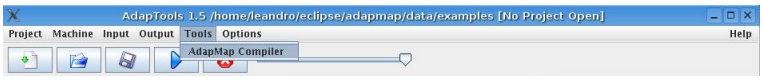

Figura 4.6: AdapTools:Seleção do compilador AdapMap para utilização

Capítulo 5

Desenvolvimento da Pesquisa

O AdapTools passou por um processo de refatoração, alteramos e acrescentamos elementos à arquitetura de armazenamento e processamento de regras, tornando-a mais robusta, de fácil entendimento e de simples manutenção.

A máquina Virtual do AdapTools foi alterada para tratar transições que caracterizam o não-determinismo em sua execução, transições internas do autômato são avaliadas e se restando apenas uma transição aplicável, este é executada deterministicamente, ou restando mais de uma transição aplicável, então todas as transições correspondentes são tratadas em paralelo, de forma não-determinística na operação do autômato. Esta propriedade é de extrema importância para o tratamento de linguagens dependentes de contexto e no problema da síntese de voz.

Visando uma maior facilidade e também minimizar os problemas decorrentes da codificação dos autômatos, foi desenvolvido e integrado ao AdapTools um compilador para conversão de uma linguagem de auto nível de mapeamento de autômatos para a linguagem AdapTools. Este compilador foi nomeado como AdapMap.

Neste capítulo faremos uma análise mais apurada das novas funcionalidades implementadas, abordando as principais dificuldades encontradas e como foram tratadas.

5.1 Arquitetura de Armazenamento de Dados no AdapTools

A estrutura de armazenamento e manipulação de regras foi dividida em camadas. A primeira camada fornece as regras de negócio necessárias para a manipulação da estrutura dos autômatos e faz parte do pacote *adaptools.bo*. A segunda camada fornece métodos necessários para persistência e

recuperação de dados e faz parte do pacote *adaptools.dao*.

No pacote *adaptools.bo* encontra-se uma interface para acesso aos dados, chamada de *RulesBO*, esta interface teve sua estrutura de métodos baseada na tabela de armazenamento utilizada anteriormente pelo AdapTools para que não causa-se maiores problemas em outras partes da ferramenta.

Ainda neste pacote, encontram-se duas implementações diferentes para esta camada. A primeira implementação, a classe *RulesBODefaultImpl*, provê todas as funcionalidades necessárias para a manipulação da estrutura dos autômatos. A segunda implementação, a classe *RulesBOViewImpl*, contém os mesmos métodos da classe *RulesBODefaultImpl* devido a uma exigência da camada de visualização que já existia, mas com comportamentos diferentes, pois os métodos da *RulesBOViewImpl* utilizam as funcionalidades da primeira implementação, diminuindo assim a quantidade de código replicado.

O pacote *adaptools.dao*, oferece um conjunto de métodos através da interface *RulesDAO* que manipulam, persistem e recuperam regras no banco de dados. A implementação desta interface se dá na classe *RulesDAOHSQLDBImpl* que se utiliza de classes auxiliares para criar e executar scripts na linguagem SQL, recuperando, gravando e excluindo regras dos autômatos.

O banco de dados utilizado para o armazenamento de dados é o HSQLDB, escrito na linguagem JAVA. Foi escolhido, pois, pode ser executado em qualquer plataforma desde que essa possua uma JVM, pela facilidade de integração com a ferramenta e pela maneira que os dados são recuperados, utilizando comandos SQL. Apesar de inserir uma camada adicional na recuperação das regras, a facilidade de uso e eficiência na recuperação de dados justifica seu uso.

5.2 Tratamento do Não-Determinismo no AdapTools

Durante os estudos teóricos para implementação do tratamento do não-determinismo na ferramenta AdapTools, verificamos dois grandes problemas na manipulação dos dispositivos adjascentes. O primeiro, ocorre quando uma transição não-determinística leva a uma execução infinita de outras transições não-determinísticas que podem ser causadas por funções adaptativas que geram muitas transições deste caráter no autômato ou em estruturas de autômatos que possuam um ciclo formado por transições do tipo ϵ, de modo que, a execução de uma transição não-determinística não é aceita ou rejeitada na operação do autômato, ou seja, o percurso percorrido nunca é dado como encerrado.

O segundo problema, diz respeito, a consistência dos transdutores, verificamos que não é trivial escolher qual regra de transdução deve ser aplicada, uma vez que podemos ter inumeras (dependendo do número de transições não-determinísticas realizadas paralelamente). Dado a 4-upla (A,B,C,D), sendo:

- A = estado inicial.
- B = símbolo lido da cadeia de entrada.
- C = estado destino.
- D = string gerada pelo transdutor.

Considere o seguinte estado do dispositivo:

(a,1,b,"Token1")
(a,1,c,"Token2")
(a,1,d,"Token3")

Lê-se, no estado "a", com a entrada "1", pode-se transitar (indiferentemente) para o estado "b"(gerando "Token1") ou para o estado "c"(gerando "Token2") ou para o estado "d"(gerando "Token3"), antes de prosseguir.

Estas transições serão executadas paralelamente, o problema consiste em escolher qual string do transdutor utilizar sabendo que o retorno da chamada não-determinística foi aceito. Para algumas aplicações como, transdutor fonético que implementamos, a transdução de um autômato é o produto do reconhecimento da linguagem, em outras palavras, é o resultado que esperamos após a execução do autômato. A dificuldade encontrada está relacionada intimamente com a característica não-determinística do modelo experimentado.

A definição, por outro lado, diz que nada se pode afirmar inicialmente sobre o(s) percurso(s) que será(ão) percorrido(s) pelo autômato depois de executada esta transição, pois a aceitação da cadeia de entrada só se fará conhecer depois de consumidos todos os símbolos de entrada [15].

Para resolver o problema, criamos para cada possível percurso, strings de transdução separadas, onde os resultados da execução daquele percurso particular são acumulados. No momento em que o percurso escolhido estiver rejeitando a cadeia de entrada, a transdução correspondente é descartada. No momento em que um percurso aceita a cadeia de entrada, a transdução correspondente é considerada como sendo a transdução do autômato para aquele percurso. Neste caso, é preciso prosseguir com a análise dos demais percursos porque pode haver mais de um percurso aceitando a cadeia de

entrada. Todas as transduções seriam possíveis soluções. Neste ultimo caso, entretanto, talvez seja mais coerente dizer que o autômato aceita de fato a cadeia de entrada, mas que a transdução só é considerada válida se as strings de transdução referentes a todos os caminhos contiverem a mesma cadeia de saída. Caso contrário é caracterizada uma inconsistência na estrutura do autômato [15, 1].

Para o tratamento e implementação do não-determinismo em autômatos na ferramenta AdapTools, propomos duas soluções, uma utilizando o mecanismo de *threads* e outro utilizando execução distribuída. A seguir faremos uma breve descrição da tecnologia e plataforma utilizada para implementação deste recurso.

5.2.1 AdapTools Distribuído

Para o AdapTools distribuído optamos por uma implementação utilizando RMI(Remote Method Invocation - Chamada de Métodos Remotos), que possibilita que um objeto operável em uma máquina virtual Java possa interagir com objetos de outras máquinas virtuais Java, independentemente da localização dessas máquinas virtuais. Configurando o arquivo *hosts.config* de todas as máquinas que serão envolvidas no processamento, a execução distribuída será ativada. Basta adicionar o nome de cada máquina no arquivo de modo que estejam separadas por uma quebra de linha, como exemplo:

```
Maquina01
Maquina02
Maquina99
```

A seqüência de passos realizada para o tratamento do não-determinismo de forma distribuída é mostrada no Anexo A Figura A.2 e descrita com mais detalhes abaixo:

- *Kernel* avalia quantas transições possíveis podem ser executadas em um dado estado consumindo um símbolo da cadeia de entrada.
- *Kernel* delega ao objeto *NoDeterministic* que trate o não-determinismo para executar as transições.
- Objeto *NoDeterministic* verifica quais são as máquinas disponíveis para execução distribuída.
- Objeto *NoDeterministic* pede para cada máquina que crie uma nova instância de uma *Vmds*.

- Objeto *NoDeterministic* cria os clientes para acesso aos objetos remotos do tipo *Vmds*.
- Objeto *NoDeterministic* inicializa os atributos de cada *Vmds* remota e coloca-as em execução.
- Objeto *NoDeterministic* aguarda uma resposta das chamadas remotas seguindo uma das estratégias que podem ser configuradas pelo usuário: 1) Seleciona a primeira transição aceita; 2) Espera todas as transições voltarem e trata a consistência dos transdutores, mesmo tendo o risco de algumas nunca voltarem; 3) Define o tamanho da árvore de chamadas não-determinísticas e testa a consistência das transições que foram aceitas.
- Objeto *NoDeterministic* escreve o retorno escolhido no transdutor do *Kernel* que iniciou o tratamento do não-determinismo e retorna o controle da execução para o *Kernel*.

Estes passos se repetem sempre que uma instância, local ou remota do *Kernel*, encontra uma transição não-determinística.

Tecnologias para Desenvolvimento Distribuído em Java

Java geralmente refere-se a combinação de três coisas: a linguagem de programação Java (uma linguagem de alto nível e orientado a objetos); a máquina virtual Java (uma máquina virtual de alto desempenho que executa bytecodes, especificado na plataforma, tipicamente abreviada como JVM); e a plataforma Java, uma plataforma que executa e compila bytecodes Java, usualmente chamando e setando bibliotecas fornecidas pelas plataformas Java Standart Edition (SE) ou Enterprise Edition (EE).

A Máquina Virtual Java está disponível em diferentes sistemas operacionais, oferecendo como grande diferencial a portabilidade de um programa Java que pode funcionar em várias máquinas independentemente da plataforma e sistema operacional.

RMI - Remote Method Invocation Java Remote Method Invocation (Chamada de Métodos Remotos) permite ao programador criar tecnologia distribuída, sem se preocupar com detalhes de comunicação, baseando-se nas bibliotecas fornecidas pela plataforma Java, em cada ponto remoto o objeto Java pode ser chamado por outras máquinas virtuais Java, disponibilizado em diferentes *hosts*. O RMI utiliza serializações de objetos para comunicação com suporte ao poliformismo da orientação a objetos.

Remote Method Invocation (RMI) facilita a chamadas de funções de objetos entre máquinas virtuais Java (JVMs). As JVMs podem estar em máquinas separadas, além disso, uma JVM pode ser chamada por um objeto armazenado em uma outra JVM. Os métodos podem passar objetos que a máquina virtual estrangeira nunca tenha encontrado antes, aceitando carga dinâmica de novas classes se necessário.

Considere o seguinte exemplo:

- O desenvolvedor A escreve um serviço que executa alguma função útil. Ele regularmente altera este serviço, adicionando novas características e fornecendo as existentes.
- O desenvolvedor B quer usar o serviço fornecido pelo desenvolvedor A. Contudo é inconveniente que o desenvolvedor A comunique a B sobre cada alteração feita.

O RMI fornece uma solução de fácil implementação, pode carregar novas classes dinamicamente, o desenvolvedor B pode deixar o RMI atualizar automaticamente para ele. O desenvolvedor A coloca a nova classe no diretório *web*, onde o RMI pode buscar pelas ultimas alterações feitas.

A Figura 5.1 mostra o modelo de conexão de um cliente usando RMI. primeiramente o cliente inicia um contato com o RMI *registry*, e pergunta pelo nome do serviço. O desenvolvedor B não saberá onde exatamente está localizado o serviço RMI, mas sabe o suficiente para acessar o serviço fornecido pelo desenvolvedor A que apontará a direção do serviço que será chamado.

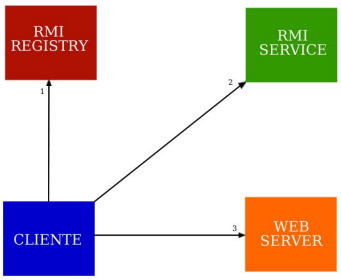

Figura 5.1: Conexões utilizadas pelo cliente RMI

O desenvolvedor A regularmente muda o arquivo, o desenvolvedor B não tem uma cópia desta classe, pois, esta será carregada automaticamente do serviço web onde as classes são armazenadas. A nova classe é armazenada

na memória, é lida e utilizada. Isto acontece transparentemente para o desenvolvedor B, que não precisa escrever nenhum código extra para achar a classe.

Na execução distribuída do AdapTools o modelo acima citado pode ser verificado facilmente. Antes de executar uma transição não-determinística, o cliente contata o serviço fornecido pelo *host* escolhido, este *host*, aloca um objeto remoto que implementa a interface *Vmds*, o cliente recebe a instância do objeto, tornando transparente o processo de localização e a implementação utilizada desse objeto fica a cargo do fornecedor do serviço.

5.2.2 AdapTools Local

A estratégia utilizada segue um fluxo de execução parecido com o distribuído, mas utiliza *threads* em sua execução, como descrito abaixo e mostrado no Anexo A, Figura A.1.

- *Kernel* avalia quantas transições possíveis podem ser executadas
- *Kernel* pede ao objeto *NoDeterministic* para executar as transições
- Objeto *NoDeterministic* verifica que não existem máquinas disponíveis para execução distribuída
- Objeto *NoDeterministic* cria uma thread para cada transição a ser tratada. Cada *thread* cria uma nova instância de uma *Vmds*.
- Objeto *NoDeterministic* fornece para cada *thread* os atributos necessários para inicialização de cada *Vmds* criada que as colocam em execução.
- Objeto *NoDeterministic* aguarda uma resposta das chamadas seguindo uma das estratégias que podem ser configuradas pelo usuário: 1) Seleciona a primeira transição aceita; 2) Espera todas as transições voltarem e trata a consistência dos transdutores, mesmo tendo o risco de algumas nunca voltarem; 3) Define o tamanho da árvore de chamadas não-determinísticas e testa a consistência das transições que foram aceitas
- Objeto *NoDeterministic* escreve o retorno escolhido no transdutor do *Kernel* que iniciou o tratamento do não-determinismo e retorna o controle da execução para o *Kernel*.

Este processo se repete sempre que uma instância, local ou remota do *Kernel*, encontra uma transição não-determinística. A execução local ocorrerá se e somente se o arquivo *hosts.config* se encontrar vazio.

5.2.3 Configuração do Não-Determinismo

O AdapTools oferece três maneiras diferentes de avaliação das transições não-determinísticas. Essas abordagens foram implementadas após estudos teóricos, nos quais verificamos que alguns autômatos podem gerar muitas transições não-determinísticas, principalmente quando o crescimento do autômato está em função da cadeia de entrada e das funções adaptativas.

O tratamento das diferentes avaliações das transições não-determinísticas foram definidas como:

- Utiliza-se a primeira transição que possua um retorno de um caminho aceito pelo autômato como a única verdadeira. Não espera pelo retorno das transições que foram executadas em paralelo.

- Espera o retorno de todas as transições, seleciona aquelas que tiveram o caminho aceito e verifica se a transdução destas são equivalentes, caso contrário caracteriza-se uma inconsistência no autômato. Esta abordagem pode causar uma espera infinita, pois, há o risco de alguma transição não possuir um retorno, geralmente caracterizam este tipo transições que consomem o símbolo ϵ.

- Utiliza a mesma abordagem do anterior, espera por todas as transições para avaliar o retorno, mas com uma diferença, é definido o tamanho máximo da árvore de chamadas não-determinísticas, caso atinja o limite máximo a transição é retornada como não aceita.

Para selecionar uma das opções descritas acima, clique sobre os menus *Options→ Execution Type* e selecione uma das respectivas opções: First Result, Wait All Result, Limited Search.

Para definir o tamanho máximo da árvore de chamadas não-determinísticas da opção *Limited Search*, clique sobre os menus *Project→ Preferences...* e preencha o campo *Limited Search* após clique em *OK*. O valor configurado será salvo no arquivo de projeto do autômato se esta ação for realizada pelo usuário. Se o usuário preferir, no arquivo de projeto poderá ser adicionado ou alterado diretamente esta opção utilizando a seguinte codificação como abaixo, definindo o tamanho da árvore de chamadas recursivas em 10 (dez).

[LimitedSearch] 10;

Se o valor não for definido no arquivo de configuração de projeto ou na interface gráfica do AdapTools, o valor padrão a ser utilizado pela ferramenta será 5 (cinco).

5.2.4 Alterações e Melhorias no Adaptools

Na versão anterior do AdapTools os estados somente poderiam ser dados através de números inteiros. Para melhorar a descrição e implementação dos autômatos na linguagem AdapTools o código foi refatorado para aceitar *strings* na criação dos estados dos autômatos.

Outra alteração que solucionou um *bug* na tabela de codificação de autômatos, estava relacionada ao número de cores disponíveis para pintar a tabela. O número de cores para visualização do autômato no *Viewer* se esgotava quando o número de regras era muito grande, causando uma exceção na ferramenta. Nesta alteração o número de cores disponíveis vai de 1 a 255, quando chegam em 255 as cores geradas voltam ao início, repetindo todas novamente.

A palavra reservada "other"utilizada na leitura de símbolos, foi retirada, pois, as transições aplicáveis devem ser executadas em paralelo e este, impõe um comportamento diferente do modelo. Quando nenhuma das transições era aceita o AdapTools executava a transição contendo o "other", consumindo qualquer símbolo da cadeia de entrada e transitando para o estado destino.

O processo de carga do autômato na estrutura de armazenamento de AdapTools foi melhorado, anteriormente o algoritmo era caracterizado por muitas comparações e invocação de métodos desnecessários. O processo apresentou melhoras significativas como podemos verificar nos testes utilizando JUnit Test Case(Testes unitários Java) e visualmente nas manipulações realizadas na ferramenta como usuário.

Thread Kernel não estava sincronizada com *Vmds*. *Kernel* estava executando antes da *Vmds* receber sua instância, por causa da dependência cíclica destas duas classes, quando o *Kernel* iniciava o objeto *Adapter* para execução, a *Vmds* ainda não tinha recebido a instância do *Kernel*, causando exceção do tipo *NullPointerExeception* em alguns métodos. Este *bug* apareceu devido as alterações na estrutura de armazenamento que deixaram o *Kernel* mais eficiente. Alteramos o *Kernel* que está estendendo da classe *Thread*. *Vmds* cria o *Kernel*, informa ao *Kernel* sua instância e após inicia a execução do *Kernel*.

5.3 AdapMap

O compilador AdapMap foi desenvolvido utilizando uma ferramenta denominada JavaCC, que é uma ferramenta para desenvolvimento de analisadores léxico-sintático que utiliza uma gramática escrita em um modelo que é uma extensão da especificação BNF, como exemplo: (A)*, (A)+, etc. contendo

as especificações léxicas e gramática.

Os módulos do compilador AdapMap foram projetados em 4 (quatro) camadas, *adapmap.core.sintatic*, *adapmap.core.semantico*, *adapmap.core.code* e *adapmap.gui*, que dizem respeito as análises léxico-sintática, semântica, geração de código e visualização respectivamente. Estes módulos têm em sua definição e modelagem ações bem definidas, as quais podem ser melhor observado na curta descrição abaixo.

O pacote denominado *adapmap.core.sintatic* corresponde ao módulo onde estão contidas as classes responsável pela análise léxica, bem como as de análise sintática. Este módulo é responsável por verificar se os símbolos de entrada estão contidos no alfabeto da linguagem e se as palavras (*Tokens*) já analisadas pela parte léxica estão dispostas de maneira que formem uma estrutura lógica aceitável pela gramática.

O pacote *adapmap.core.semantico* diz respeito a parte semântica e é formado por classes que representam os descritores de tipos e tabelas de símbolos.

O módulo de geração de código tem a função de gerar uma saída (código adaptools), a partir dos código (adapmap) analisado nas duas outras interfaces acima (léxico-sintática e semântica).

Os analisadores léxico-sintático, semântico e até mesmo a geração de código dependem de uma gramática, esta diz quem são e como deve ser estruturado e agrupado os símbolos. A gramática do AdapMap está descria no Anexo B.

5.4 Sintetizador de Voz

Utilizar o recurso do não-determinismo prove maior facilidade para o reconhecimento de palavras que tenham sílabas homógrafas, mesma grafia e sons diferentes, e dependam do contexto ou da estrutura de formação da palavra, como exemplo, as palavras *ovo* e *ovos* que não tem o fonema referente a primeira sílaba identificado antes da leitura completa da palavra. Com o recurso do não-determinismo as duas transcrições são tratadas em paralelo, no entanto, uma das duas será rejeitada, seja por não atingir o estado final devido a não leitura de toda cadeia de entrada ou pela falta de símbolo a ser lido.

A síntese de voz implementada na ferramenta AdapTools utiliza o método de síntese por concatenação (Sessão 3.1), os arquivos de som foram previamente gravados e cada arquivo representa um fonema. A reprodução dos fonemas pela ferramenta não faz nenhum tipo de otimização ou melhoria durante a concatenação dos sons para que ocorra uma maior naturalidade na

pronúncia.

A cadeia de entrada que representa o texto a ser pronunciado (*"tem ovo e ovos no cassino"*) é formada pelo alfabeto A={a,c,e,i,m,n,o,s,t,v, *"caractere ASCII espaço"*}. Este texto é reconhecido pelo autômato adaptativo não-determinístico apresentado na Figura 5.2. Durante o reconhecimento do texto este autômato se modifica de modo que ao final do reconhecimento da entrada passa a atuar como um transdutor

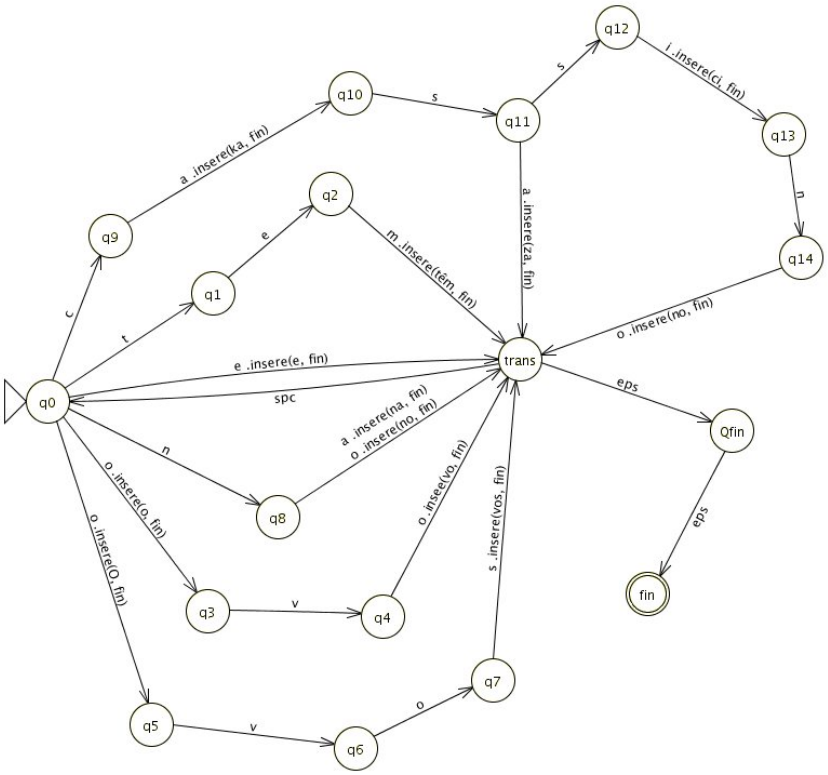

Figura 5.2: Estrutura inicial do autômato utilizado para síntese de voz

Capítulo 6

Considerações Finais

Durante a etapa de pesquisa, foram feitos levantamentos teóricos a respeito da teoria da tecnologia adaptativa, aspectos fonológicos e fonoaudiológicos, além da busca de alguns formalismos utilizados para resolução do problema da tradução texto-voz. Dentre estes formalismos pesquisados, os autômatos adaptativos, já ofereciam uma solução atraente para síntese de voz, como mostrado em [19, 26, 27].

A ferramenta AdapTools, que atualmente implementa um módulo de tradução texto-voz foi utilizada como base para uma implementação mais eficiente. Os estudos sobre engenharia de software e sobre as características estruturais da ferramenta, foram de grande valia para a realização das melhorias, documentação e criação de novas funcionalidades.

A execução de algumas tarefas do projeto, como a documentação do AdapTools através de diagramas de caso de uso, diagramas de classe e seqüência foram confeccionados devido as novas funcionalidades e melhorias adicionadas a ferramenta. Estudos de técnicas de programação distribuída utilizando a linguagem de programação Java e processamento local concorrente, foram realizados para seleção de técnicas que viabilizaram a implementação do não-determinismo no AdapTools.

6.1 Principais Resultados Obtidos

Algumas contribuições ao projeto AdapTools, como o desenvolvimento e a inclusão do compilador AdapMap, facilitou a criação e o aprendizado dos autômatos adaptativos, através de uma interface que simplifica a codificação em uma linguagem de alto nível, resolvendo algumas dificuldades encontradas na descrição e análise estrutural dos autômatos. Outra contribuição importante é a alteração da adição de rotinas semânticas na ferramenta,

anteriormente era necessário alteração no código fonte e recompilação do AdapTools, essa característica foi alterada, e no estado atual da ferramenta, a inclusão de novas rotinas semânticas passa a ser dinâmicas, como mostrado no capítulo 4.

Outra modificação realizada nas ações semânticas, foi a criação de uma nova chamada a este objeto ao final da execução de um autômato. À medida que o autômato é executado pelo AdapTools, o *Kernel* da ferramenta invoca o método *execute* passando como parâmetro a transdução gerada pela transição e ao final da execução do autômato ou após uma chamada não-determinística o Kernel invoca o método *finalise* da ação semântica com todo o transdutor gerado pelo autômato, essa nova característica foi essencial para criação do sintetizador de voz.

A implementação do recurso de execução de autômatos não-determinísticos no AdapTools oferece uma solução para uma nova gama de problemas como o processamento de linguagens dependentes de contexto, além de ser um recurso atraente para o ensino de linguagens. No AdapTools o aluno pode depurar o não-determinismo utilizando os recursos gráficos da ferramenta, executando passo-a-passo o autômato e verificando as variáveis, o consumo da cadeia de entrada, o conteúdo da pilha, os estados finais e o estado corrente de cada percurso.

Um problema relativo a execução do não-determinismo gerado principalmente por transições vazias ou chamadas recursivas de funções adaptativas é a não ocorrência de retorno, em outras palavras, o autômato é executado de maneira que nunca atinge um estado final. Este problema foi resolvido utilizando a interação do usuário que escolhe como devem ser manipuladas as chamadas de transições não-determinísticas, como mostrado na Seção 5.2.3.

Alguns diagramas UML foram produzidos para ilustrar a estrutura e as funcionalidades implementadas no AdapTools. A partir dos diagramas de seqüência realizados, pode-se entender com maior facilidade como foi implementada a principal funcionalidade adicionada ao AdapTools, a execução não-determinística.

Um problema na identificação dos estados finais foi identificado na implementação da interface gráfica. Considere o estado corrente chamado de “1”. Os estados finais estavam armazenados em uma string na forma “5 6 11”, lê-se, estados finais podem ser cinco, seis ou onze. O problema neste método estava no modo de verificação entre o estado corrente e os estados finais, o estado “1”era sempre procurado utilizando *matching* na string, podendo aceitar erroneamente o estado “1”pois o estado “11”possui este nome como *substring*.

A especificação da estrutura do autômato na ferramenta foi melhorada, a descrição dos estados apenas poderiam ser números inteiros, na modificação

realizada a identificação do estado pode ser feita utilizando cadeias de caracteres, melhorando a compreensão do código.

O número de cores disponíveis para pintar a tabela de visualização do autômato na interface gráfica, se esgotava quando o número de regras era muito grande. O número de cores disponíveis estava compreendido entre 1 e 255, então quando o número de regras carregadas era maior que 255 um erro ocorria na execução. Com a alteração o número de cores manteve-se o mesmo, com um diferencial, ao chegar ao limite as cores geradas voltam ao início, repetindo todas novamente.

A palavra reservada "other", símbolo que não possa ser consumido estando no estado de origem da transição, utilizada na leitura da cadeia de entrada, foi removida, pois, as transições devem ser executadas em paralelo e este, impõe um comportamento diferente do modelo proposto para os autômatos adaptativos.

Uma melhoria que causou um impacto perceptível na manipulação da ferramenta pelo usuário, foi a customização do método que realiza carga das regras persistidas em arquivo, na seleção de regras no *Kernel* e na busca por funções adaptativas, realizadas através da utilização eficiente da nova estrutura de armazenamento de regras que melhorou visivelmente o desempenho da ferramenta. Esta nova estrutura de armazenamento fornece métodos mais eficientes de manipulação de regras, Seção 5.1, principalmente métodos de pesquisa, remoção e adição.

O *Kernel* não estava sincronizado com a estrutura de dados da máquina virtual (Vmds). O primeiro, executava antes da *Vmds* receber sua instância, por causa da dependência cíclica destas duas classes, quando o *Kernel* iniciava o objeto *Adapter* para execução, a *Vmds* ainda não tinha recebido a instância do *Kernel*, causando uma exceção do tipo *NullPointerExeception* em alguns métodos. A melhoria se deu, com a extensão da classe *Thread* por parte do *Kernel*. Quando este é criado pela *Vmds*, essa cria o *Kernel*, recebe sua instância e depois inicia a execução do *Kernel*.

6.2 Sugestões para Trabalhos Futuros

As modificações na estrutura de dados do AdapTools forneceu métodos mais eficientes de busca de regras, com isso, podemos customizar os métodos de procura e manipulação de regras da camada adaptativa principalmente na execução de funções adaptativas de pesquisa, remoção e adição.

Novas melhorias podem ser realizadas no compilador AdapMap, entre elas uma verificação de número de parâmetros das chamadas de funções, uma melhoria na gramática para simplificar seu uso, ampliação das verificações

semânticas existentes e construção de um editor que facilite sua utilização. Na situação atual, o AdapMap mapeia os autômatos adaptativos em seu estado inicial, não possibilitando que o usuário acompanhe as modificações realizadas pelas funções adaptativas durante sua execução. Estender a ferramenta para incluir código AdapMap no mapeamento do autômato adaptativo e fornecer a possibilidade de acompanhar a execução nesta estrutura seria de grande contribuição para a pesquisa em tecnologia adaptativa.

O tratamento do não-determinismo desenvolvido neste trabalho oferece uma solução para uma série de problemas que podem ser pesquisados na área dos autômatos adaptativos. Na Seção 5.4 mostramos como o não-determinísmo pode ser explorado para resolver um problema relacionado a síntese de voz. A partir deste exemplo podemos perceber que esta nova funcionalidade pode ser de grande valia na manipulação de linguagens dependentes de contexto. A evolução na ferramenta ainda ressalta outro ponto interessante a ser explorado, a utilização do AdapTools como ferramenta com finalidades pedagógicas em várias áreas de interesse da Engenharia de Computação, principalmente no ensino de Linguagens Formais e Autômatos.

No desenvolvimento deste trabalho verificamos alguns problemas muito interessantes e de alta relevância para a pesquisa dos autômatos adaptativos. Primeiramente, em testes utilizando o autômato de síntese de voz, verificamos que as funções adaptativas, se não bem implementadas, podem gerar muitas transições com aspectos não-determinísticos e como conseqüência a execução do autômato pode ser comprometida pela grande quantidade de percursos gerados. Como solução para o problema criamos três tipos de avaliação das transições não-determinísticas, mostrado na Seção 5.2.3, entretanto, novas soluções podem ser empregadas, como por exemplo, aplicar quando possível a transformação de autômatos finitos não determinísticos (NFAS) em autômatos finitos determinísticos (DFAS) utilizando o teorema da Equivalência de NFAS e DFAS.

A segunda constatação sobre a execução de autômatos adaptativos não-determinísticos, ocorre na execução de ações elementares das funções adaptativas. As ações elementares no AdapTools sofreram alterações e refinamentos com relação a definição original das funções para simplificar a implementação da camada adaptativa e diminuir o custo da execução [19]. Nesta implementação não foi abordada a possibilidade de uma ação elementar de pesquisa retornar mais de um resultado para um padrão pesquisado, não considerando neste caso que a execução da ação caracteriza o não-determinismo. A abordagem utilizada para o tratamento de transições não-determinísticas na Seção 5.2, parece ser uma boa solução para o problema relatado.

Avanços ainda podem ser realizados na otimização da máquina virtual do AdapTools, bem como o projeto e implementação de uma interface

mais amigável e que separe os componentes utilizados para visualização dos métodos que manipulam e tratam a estrutura dos autômatos. Finalmente o AdapTools pode ser utilizado futuramente como um laboratório virtual para implementação, depuração, execução de dispositivos guiados por regras adaptativos, mas para isso é preciso definir um projeto de uma máquina virtual que compreenda apenas as funcionalidades relacionadas ao tratamento de regras e ofereça um conjunto mínimo de ações adaptativas que funcionem para qualquer dispositivo inserido.

Anexo A

Diagramas de sequência

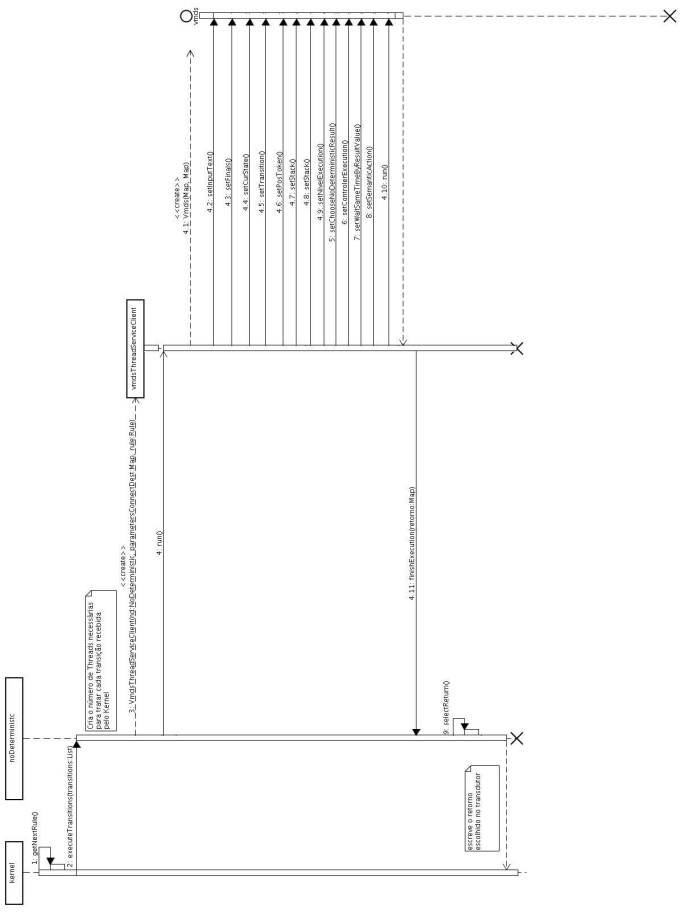

Figura A.1: Diagrama de sequência para não-determinismo local.

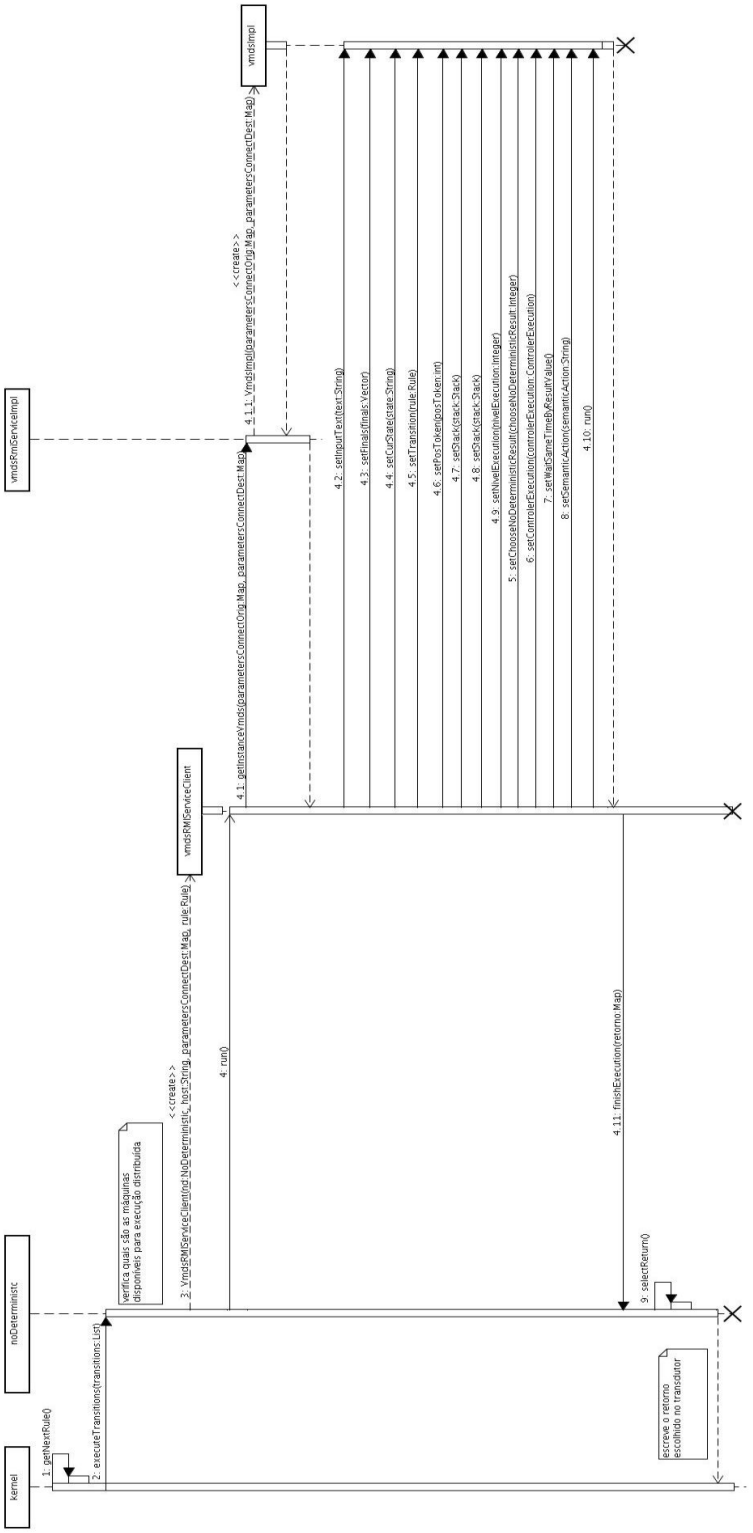

Figura A.2: Diagrama de sequência para não-determinismo distribuído.

Anexo B

Gramática AdapMap

```
DIGITO = [0..9]
LETRA = [a..z] | [A..Z]
CARACTER = DIGITO | LETRA
CARACTER_LITERAL = ‘ CARACTER ’
STRING_LITERAL = " ( CARACTER )* "

IDENTIFICADOR_GERADOR = * IDENTIFICADOR
IDENTIFICADOR_VAR_PESQUISA =  ? IDENTIFICADOR
IDENTIFICADOR = ( CARACTER )+

PROGRAMA = MAQUINA ( MAQUINA | FUNCTION_ADP )*

MAQUINA = DECLARA_MAQ { DECLARA_INIT DECLARA_FINAL DECLARA_REGRAS }

DECLARA_MAQ = machine TIPO <IDENTIFICADOR>

TIPO = ( main )?

DECLARA_INIT = ( VAR_INIT ; )?

VAR_INIT = ( init IDENTIFICADOR )?

DECLARA_FINAL = ( VARIAVEIS_FINAIS ; )*

VARIAVEIS_FINAIS = VAR_FINAL ( VAR_FINAL_EXT )*

VAR_FINAL = ( final IDENTIFICADOR )?
```

```
VAR_FINAL_EXT = , IDENTIFICADOR

DECLARA_REGRAS = ( REGRAS )*

REGRAS = state IDENTIFICADOR { REGRA_SUB }

REGRA_SUB = ( COMANDO_READ_TRANSIT ; )*

COMANDO_READ_TRANSIT = COMANDO_RETURN F_COMANDO_FUNCTION
| COMANDO_READ COMANDO_RETURN F_COMANDO_FUNCTION
| COMANDO_READ COMANDO_TRANSIT F_COMANDO_FUNCTION
| COMANDO_TRANSIT F_COMANDO_FUNCTION


COMANDO_READ = read READ_OP

COMANDO_RETURN = return COMANDO_OUTPUT

READ_OP = eps
| digit
| spc
| letter
| special
| CARACTER_LITERAL

COMANDO_TRANSIT = transit machine IDENTIFICADOR CH_MAQ_OP COMANDO_OUTPUT
| transit IDENTIFICADOR COMANDO_ADICIONAL
| transit pop COMANDO_ADICIONAL

CH_MAQ_OP = ( . IDENTIFICADOR , IDENTIFICADOR
| , IDENTIFICADOR
| . IDENTIFICADOR )?

COMANDO_ADICIONAL = COMANDO_PUSH COMANDO_OUTPUT
| COMANDO_OUTPUT COMANDO_PUSH

COMANDO_PUSH = ( push IDENTIFICADOR )?

COMANDO_OUTPUT = ( output STRING_LITERAL )?
```

```
FUNCTION_ADP = DECLARA_FUNCTION { CORPO_FUNCTION }

DECLARA_FUNCTION = function IDENTIFICADOR ( ESCOPO_FUNC )

ESCOPO_FUNC = ( IDENTIFICADOR )? ( , <IDENTIFICADOR> )*

CORPO_FUNCTION = SEARCH REMOVE INSERT

SEARCH = ( search { CORPO_SEARCH } )?

CORPO_SEARCH = ( IDENTIFICADOR = state ITENS_CORPO_SEARCH )+

ITENS_CORPO_SEARCH = MAQUINA_RULE F_COMANDO_READ_TRANSIT ;
|
VARIAVEIS_IDENTIFICADORES_RULE F_COMANDO_READ_TRANSIT ;

VARIAVEIS_IDENTIFICADORES_RULE = OBJECT_RULE
| IDENTIFICADOR
| ?sta

MAQUINA_RULE = IDENTIFICADOR . IDENTIFICADOR
| IDENTIFICADOR . IDENTIFICADOR_GERADOR

F_COMANDO_READ_TRANSIT = F_COMANDO_RETURN
| F_COMANDO_READ F_COMANDO_RETURN
| F_COMANDO_READ F_COMANDO_TRANSIT
| F_COMANDO_TRANSIT

F_COMANDO_READ = read F_READ_OP

F_READ_OP = eps
| digit
| spc
| letter
| special
| pop
| CARACTER_LITERAL
| VARIAVEIS_IDENTIFICADORES
| ?inp

F_COMANDO_RETURN = return F_COMANDO_OUTPUT
```

```
F_COMANDO_TRANSIT = transit VARIAVEIS_IDENTIFICADORES F_COMANDO_ADICIONAL
| transit ?sta F_COMANDO_ADICIONAL

F_COMANDO_ADICIONAL = F_COMANDO_PUSH F_COMANDO_OUTPUT F_COMANDO_FUNCTION
| F_COMANDO_OUTPUT F_COMANDO_PUSH F_COMANDO_FUNCTION

F_COMANDO_PUSH = ( push VARIAVEIS_IDENTIFICADORES
| push ?sta )?

F_COMANDO_OUTPUT = ( output STRING_LITERAL
| output VARIAVEIS_IDENTIFICADORES )?

F_COMANDO_FUNCTION = ( function F_COMANDO_FUNCTION_PRE_OU_POS )?

F_COMANDO_FUNCTION_PRE_OU_POS = . VARIAVEIS_IDENTIFICADORES ( F_ARGUMENTOS )
| VARIAVEIS_IDENTIFICADORES ( F_ARGUMENTOS ) .

F_ARGUMENTOS = ( VARIAVEIS_IDENTIFICADORES )? (, VARIAVEIS_IDENTIFICADORES )*

VARIAVEIS_IDENTIFICADORES = OBJECT_RULE
| IDENTIFICADOR

REMOVE = ( remove { CORPO_REMOVE } )?

CORPO_REMOVE = ( IDENTIFICADOR_RULE ;
| state MAQUINA_RULE F_COMANDO_READ_TRANSIT ; )+

IDENTIFICADOR_RULE = IDENTIFICADOR

INSERT = ( insert { CORPO_INSERT } )?

CORPO_INSERT = ( state MAQUINA_RULE { INSERT_COMANDO_READ_TRANSIT  }
| state VARIAVEIS_IDENTIFICADORES_RULE { INSERT_COMANDO_READ_TRANSIT } )+

INSERT_COMANDO_READ_TRANSIT = ( F_COMANDO_READ_TRANSIT ; )+

OBJECT_RULE = IDENTIFICADOR_VAR_PESQUISA
| IDENTIFICADOR_GERADOR
```

Referências Bibliográficas

[1] CASTRO JR. A. A. Comunicação pessoal (orientação) - GPEC/UCDB. Junho 2006.

[2] J. BACHENCO, E. FITZPATRICK, and C. E. WRIGTH. The contribuition of parsing to prosodic phrasing in an experimental text-to-speech system. *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, 1986.

[3] E. BEZERRA. *Princípios de análise e projeto de sistemas com UML*, volume Primeira Edição. , Campus, Rio de Janeiro, 2002.

[4] A. J. B. CASTRO, D. A ALFENAS, D. P SHIBATA, and H. T. SOEJIMA. Sintetizador texto-voz com autômatos adaptativos. Projeto apresentado à Disciplina de PCS, Escola politécnica de São Paulo, SP, Brasil, 2004.

[5] M. FOWLER. *UML Essencial - Um breve guia para linguagem-padrão de modelagem de objetos*, volume 3.ed. , Bookman, Porto Alegre, Brasil, 2002.

[6] F. HUCHE and Andé ALLALI. *A voz*. Porto Alegre, RS, Brasil, 1999.

[7] Solange ISSLER. *Articulação e linguagem*. São Paulo, SP, Brasil, 1996.

[8] Sami LEMMETTY. *Review of Speech Synthesis Technology*. PhD thesis, Review of Speech Synthesis Technology, Vuorimiehentie 1 A, 02150 Espoo, Finland, 1999.

[9] R. S. MAIA, H. ZEN, TOKUDA K., T. KITAMURA, and F. G. V. RESENDE JR. Towards the development of a brazilian portuguese text-to-speech system based on HMM. *EUROSPEECH*, 2003.

[10] M. MOHRY. Finite-state transducer in language and speech processing. In *Association for Computational Lingistics*, Murray Hill, NJ, USA, 1997.

[11] M. MOHRY, F. PEREIRA, and RILEY M. Weighted finite-state transducers in speech recognition. *Proc. Automated Speech Recognition: Challenges for the Next Millennium*, 2000.

[12] M. MOHRY and SPROAT R. An efficient compiler for weighted rewrite rules. *Proc. 34th Meeting of the Association for Computational Linguistics (ACL 96)*, 1996.

[13] J. J. NETO. Adaptative rule-driven devices - general formulation and case study. In *CIAA'2001 Sixth International Conference on Implementation and Application of Automata*, pages 234–250, Pretoria, South Africa, July 2001.

[14] J. J. NETO. Autômatos em engenharia de computação - uma visão unificada. *Primeira Semana de Ciencia y Tecnología - I SCT*, 2003.

[15] J. J. NETO. Comunicação pessoal - LTA/USP. Agosto 2006.

[16] J. J. NETO, E. R. COSTA, and PISTORI H. A free software for the development of adaptive automata. *IV International Forum on Free Software*, 2003.

[17] L. E. S. OLIVEIRA and M. E. MORITA. Introdução aos modelos escondidos de markov (hmm). Technical report, Pontifícia Universidade Católica do Paraná - PUCPR, Curitiba, PR, Brasil, 1992.

[18] P. E. OSTERMANN FILHO. Desenvolvimento de regras de pronúncia para a síntese de voz em língua portuguesa. *VII Semana Acadêmica do PPGC - UFRGS*, 2000.

[19] H. PISTORI. *Tecnologia Adaptativa em Engenharia de Computação: Estado da Arte e Aplicações*. PhD thesis, Universidade de São Paulo, São Paulo, Brasil, 2003.

[20] H. PISTORI and J. J. NETO. Adaptools: Aspectos de implementação e utilização. *Boletim técnico PCS - USP*, 2003.

[21] D. G SANTOS, PISTORI H., A. A. CASTRO JR, and J. J NETO. Adaptools: Um ambiente gráfico de apoio ao desenvolvimento de software adaptativo. *SIMS - Simposio de Informatica e Mostra de Software Acadêmico*, 2005.

[22] T. C. SILVA. *Fonética e Fonologia do Português*. São Paulo, SP, Brasil, 2003.

[23] R. C. P. SILVEIRA. *Estudo de Fonologia portuguesa*. São Paulo, SP, Brasil, 1986.

[24] J. C. WELLS. Computer-coding the ipa: a propoused extension of sampa. University College London, 1985.

[25] A. E. XOUSIF. Speech analysis and synthesis on a personal computer. In *ACM SIGSMALL/PC symposium on Small systems*, San Francisco, California, United States, 1986.

[26] F. A. ZUFFO. Utilização de autômatos adaptativos para tradução texto-voz. Monografia de Conclusão de Curso de Engenharia de Computação - UCDB, Campo Grande, MS, Brasil, Dezembro 2004.

[27] F. A. ZUFFO and H. PISTORI. Tecnologia adaptativa e síntese de voz: Primeiros experimentos. *V Workshop de Software Livre - WSL*, 2004.