Memórias do WTA 2015

IX Workshop de Tecnologia Adaptativa

Laboratório de Linguagens e Técnicas Adaptativas
Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo

São Paulo
2015

Ficha catalográfica

Workshop de Tecnologia Adaptativa (9: 2015: São Paulo)
Memórias do WTA 2015. – São Paulo; EPUSP, 2015. 131p.

ISBN 978-85-86686-82-5

Agência Brasileira do ISBN
ISBN 978-85-86686-82-5

1. Engenharia de computação (Congressos) 2. Teoria da computação (Congressos) 3. Teoria dos autômatos (Congressos) 4. Semântica de programação (Congressos) 5. Linguagens formais (Congressos) I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II. t.

CDD 621.39

Apresentação

A nona edição do Workshop de Tecnologia Adaptativa realizou-se em São Paulo, Brasil, nos dias 29 e 30 de Janeiro de 2015, nas dependências da Escola Politécnica da Universidade de São Paulo. As contribuições encaminhadas na forma de artigos relacionados à Tecnologia Adaptativa, nas seguintes áreas, abrangeram, de forma não exclusiva, os tópicos abaixo:

Fundamentos da Adaptatividade

- Modelos de computação, autômatos, gramáticas, grafos e outros dispositivos automodificáveis, suas notações, sua formalização, complexidade, propriedades e comparações com formalismos clássicos.

Tecnologia Adaptativa – Técnicas, Métodos e Ferramentas

- Aplicação dos conhecimentos científicos relativos à adaptatividade e dos dispositivos adaptativos como fundamento para a formulação e para a resolução de problemas práticos.
- Ferramentas, técnicas e métodos para a automatização da resolução de problemas práticos usando técnicas adaptativas.
- Programação adaptativa: linguagens, compiladores e metodologia para o desenvolvimento, implementação e validação de programas com código adaptativo.
- Meta-modelagem de software adaptativo.
- Engenharia de Software voltada para a especificação, projeto, implementação e desenvolvimento de programas automodificáveis de qualidade.
- Adaptatividade multinível e outros conceitos introduzidos recentemente: avanços teóricos, novas idéias para aplicações, sugestões de uso prático.
- Linguagens de alto nível para a codificação de programas automodificáveis: aplicações experimentais e profissionais, práticas e extensas, das novas idéias de uso de linguagens adequadas para a codificação de programas adaptativos e suas metodologias de desenvolvimento.

Aplicações da Adaptatividade e da Tecnologia Adaptativa

- Inteligência computacional: aprendizagem de máquina, representação e manipulação do conhecimento;
- Computação natural, evolutiva e bio-inspirada;
- Sistemas de computação autonômica e reconfigurável;

– Processamento de linguagem natural, sinais e imagens: aquisição, análise, síntese, reconhecimento, conversões e tradução;

– Inferência, reconhecimento e classificação de padrões;

– Modelagem, simulação e otimização de sistemas inteligentes de: tempo real, segurança, controle de processos, tomada de decisão, diagnóstico, robótica;

– Simulação, arte por computador e jogos eletrônicos inteligentes;

– Outras aplicações da Adaptatividade, nas diversas áreas do conhecimento: ciências exatas, biológicas e humanas.

Comissão de Programa

Comissão Organizadora

- André Riyuiti Hirakawa (São Paulo, SP, Brasil)
- João José Neto, Chair (São Paulo, SP, Brasil)
- Paulo Roberto Massa Cereda (São Paulo, SP, Brasil)
- Ricardo Luis de Azevedo da Rocha (São Paulo, SP, Brasil)

Apoio

- Escola Politécnica da Universidade de São Paulo
- IEEE – Institute of Electrical and Electronics Engineers
- Olos Tecnologia e Sistemas S.A.
- Pagga Tecnologia de Pagamentos Ltda.
- PCS – Departamento de Engenharia de Computação e Sistemas Digitais
- São Paulo Convention & Visitors Bureau
- SBC – Sociedade Brasileira de Computação
- SPC – Sociedad Peruana de Computación
- Universidade de São Paulo

Memórias do WTA 2015

Esta publicação do Laboratório de Linguagens e Técnicas Adaptativas do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo é uma coleção de textos produzidos para o WTA 2015, o Nono Workshop de Tecnologia Adaptativa, realizado em São Paulo nos dias 29 e 30 de Janeiro de 2015. A exemplo da edição de 2014, este evento contou com uma forte presença da comunidade de pesquisadores que se dedicam ao estudo e ao desenvolvimento de trabalhos ligados a esse tema em diversas instituições brasileiras e estrangeiras, sendo o material aqui compilado representativo dos avanços alcançados nas mais recentes pesquisas e desenvolvimentos realizados.

Introdução

Com muita satisfação compilamos neste documento estas memórias com os artigos apresentados no WTA 2015 – Nono Workshop de Tecnologia Adaptativa, realizado na Escola Politécnica da Universidade de São Paulo nos dias 29 e 30 de Janeiro de 2015.

Esta edição contou com 17 trabalhos relacionados à área de Tecnologia Adaptativa e aplicações nos mais diversos segmentos. O evento foi registrou uma participação efetiva de uma centena de pesquisadores durante os dois dias, constatando-se o sucesso do mesmo.

Esta publicação

Estas memórias espelham o conteúdo apresentado no WTA 2015. A exemplo do que foi feito no ano anterior, todo o material referente aos trabalhos apresentados no evento estará acessível no portal do WTA 2015, incluindo softwares e os slides das apresentações das palestras. Adicionalmente, o evento foi gravado em vídeo, em sua íntegra, e os filmes serão também disponibilizados aos interessados. Esperamos que, pela qualidade e diversidade de seu conteúdo, esta publicação se mostre útil a todos aqueles que desejam adquirir ou aprofundar ainda mais os seus conhecimentos nos fascinantes domínios da Tecnologia Adaptativa.

Conclusão

A repetição do sucesso das edições anteriores do evento, e o nível de qualidade dos trabalhos apresentados atestam a seriedade do trabalho que vem sendo realizado, e seu impacto junto à comunidade. Somos gratos aos que contribuíram de alguma forma para o brilho do evento, e aproveitamos para estender a todos o convite para participarem da próxima edição, em 2016.

Agradecimentos

Às instituições que apoiaram o WTA 2015, à comissão organizadora, ao pessoal de apoio e a tantos colaboradores voluntários, cujo auxílio propiciou o êxito que tivemos a satisfação de observar. Gostaríamos também de agradecer às seguintes pessoas pelo valioso auxílio para a viabilização das necessidades locais do evento:

Apoio administrativo

- Alan Borim
- Cleusa Cruz
- Leia Sicília
- Lúcio Coiti Yajima
- Luís Emilio Cavechiolli Dalla Valle
- Rosany Perez
- Wagner Mendes
- Ronaldo de Lima Tavares

Apoio operacional

- Beatriz Conceição Silva
- Daniel Costa Ferreira
- Edson de Souza
- Laércio Lindoso Ferreira
- Newton Kiyotaka Miura
- Nilton Araújo do Carmo
- Rodrigo Kavakama
- Rosalia Edith Caya Carhuanina

Divulgação

- Amanda Rabelo
- Otávio Nadaleto

De modo especial, agradecemos ao Departamento de Engenharia de Computação e Sistemas Digitais e a Escola Politécnica da Universidade de São Paulo pelo inestimável apoio para a realização da nona edição do Workshop de Tecnologia Adaptativa.

São Paulo, 30 de Janeiro de 2015

João José Neto
Coordenador geral

Mensagens

As seguintes mensagens foram enviadas pelos profs. Marcus Vinícius Midena Ramos e Amaury Antônio de Castro Júnior para registrar suas participações na nona edição do Workshop de Tecnologia Adaptativa.

Marcus Vinícius Midena Ramos

Dando continuidade ao meu trabalho com formalização matemática e o uso de assistentes de prova interativos, eu estou buscando a formalização de uma parte importante da teoria de linguagens livres de contexto. Essa teoria, apesar de ser clássica e de apresentar grande interesse prático e teórico para diversas áreas da computação, em especial a área de processamento de linguagens, ainda não foi objeto de formalização no sentido mais amplo, com apenas algumas iniciativas isoladas tendo sido produzidas até o momento. O mesmo não acontece com a classe das linguagens regulares, que já se encontra num estágio bastante avançado de formalização. Por isso, o meu interesse em formalizar os principais resultados relacionados com a classe das linguagens livres de contexto.

Formalização matemática significa formular e demonstrar os teoremas de uma certa teoria usando o suporte de computadores através dos chamados assistentes de prova interativos. Tais assistentes implementam uma linguagem, geralmente baseada numa lógica de primeira ordem e numa teoria de tipos, com eventuais extensões, através da qual podem ser construídas as provas dos teoremas de interesse. No caso do assistente de provas *Coq*, que eu estou utilizando, essa linguagem é o Cálculo de Construções com Definições Indutivas. Por se tratar de um cálculo rico em detalhes, normalmente os termos de prova são construídos de forma indireta, através do uso de táticas que auxiliam a obtenção dos mesmos. Como vantagem, o uso dos assistentes de prova permite a verificação dos termos de prova construídos, garantindo a sua correção. Além disso, através da correspondência entre provas e programas, é possível obter programas certificados que implementam os tipos correspondentes aos enunciados dos teoremas. Tudo isso significa verificação automática de provas, maior rapidez na mesma, menos erros nas provas obtidas e geração automática de programas certificados, vantagens importantes para matemáticos e programadores de sistemas e teorias cada vez maiores e mais complexos.

No caso específico do meu trabalho, eu pretendo desenvolver uma biblioteca que permita a outros pesquisadores raciocinar sobre gramáticas livres de contexto e provar novos resultados a partir de uma coleção de definições relacionadas e de teoremas fundamentais já provados. Assim, ele foi dividido em quatro etapas, a saber a formalização *(i)* das propriedades de fechamento, incluindo união, concatenação e estrela de Kleene, *(ii)* das simplificação gramaticais, incluindo a eliminação de símbolos inúteis e inacessíveis, e também a eliminação de regras unitárias e vazias, *(iii)* das formas normais (Chomsky e Greibach) e, finalmente, *(iv)* do Lema do Bombeamento.

No presente momento, a etapa *(i)* está finalizada. A etapa *(ii)* está praticamente terminada, faltando apenas a conclusão da formalização da eliminação de regras vazias. As etapas

(iii) e *(iv)* ainda não foram iniciadas. No total, foram desenvolvidos mais de 7000 linhas de scripts em Coq, com mais de 150 lemas e teoremas provados.

Além do objetivo pretendido, o resultado deste trabalho, quando concluído, será útil no ensino de linguagens formais e autômatos, de teoria de tipos, de lógica e de engenharia de software, áreas que estão relacionadas através do mesmo. Fora isso, eu considero que o trabalho de formalização matemática é uma tendência crescente, tanto na área teórica da computação e da matemática, quanto na área de desenvolvimento e certificação de software. Por isso, eu vejo grande potencial de aplicação da mesma na tecnologia adaptativa, seja certificando sistemas seja ajudando a entender a semântica dos dispositivos e a provar resultados relacionados com as suas propriedades. Eu espero estar apto para enfrentar esse desafio no futuro próximo, além de poder compartilhar o pouco que eu aprendi até o momento com outros interessados nessa área tão promissora.

Marcus Vinícius

Amaury Antônio de Castro Júnior

Prezado Prof. João e demais amigos e amigas, palpitantes e colaboradores do WTA,

Tudo bem?

Quero, primeiramente, parabenizar o Prof. João, o Paulo Cereda e toda a equipe organizadora do WTA 2015.

Infelizmente, não pude estar presente no evento deste ano em razão de não termos horários de vôo adaptativos. :-)

Mas, enfim, envio esta mensagem a todos para que, mesmo que virtualmente, possa me fazer presente entre tantos amigos e colegas que diz desde que passei a fazer parte do LTA.

Lá se vão 9 anos de história do WTA. Um evento que se apresenta como um dos grandes exemplos de crescimento e fortalecimento dessa área de pesquisa. Ano que vem, completaremos nossos 10 anos e o Prof. João já está preparando alguma novidades para marcar, definitivamente, a história do WTA. Não vou comentar sobre isso no texto, pois creio que ele mesmo contará a vocês sobre tudo que está pensando e criando.

Enfim, acredito que todos nós teremos um ano de bastante trabalho e de muita produção, com o objetivo de juntarmos forças e consolidarmos esse grande grupo que vem se formando há tempos. Aqui em Mato Grosso DO SUL, a Tecnologia Adaptativa se espalhou pelo estado, desde a capital até o interior. Quem sabe, também possamos atingir o Paraguai! :-)

Continuamos tentando motivar e incentivar novos pesquisadores e acadêmicos a conhecerem um pouco dessa área. Esse ano, eu e o Prof. Reginaldo vamos ministrar uma disciplina de Fundamentos e Aplicações da Tecnologia Adaptativa dentro do Programa de Mestrado Profissional da UFMS. Estamos atuando em uma linha de pesquisa para o desenvolvimento de aplicações no agronegócio. Em pararelo, temos projetos em parcerias com a UCDB, através do Prof. Hemerson Pistori, um dos grandes nomes da área, visando a aplicação dessas tecnologias no tratamento e classificação de imagens coletadas por VANTs na agricultura e pecuária de precisão.

Como podem ver, as perspectivas de pesquisa são muitas e esperamos, no ano que vem, poder apresentar alguns desses projetos e resultados nos 10 anos de WTA.

Desejo a todos um bom evento e deixo aqui um forte abraço a todos os amigos. Espero revê-los em breve e continuo à disposição para contatos colaborativos.

Um abraço da fronteira!

Aweté (Obrigado em Guarani).

Amaury Jr. – UFMS/Câmpus de Ponta Porã

Sumário

Lista de autores

Parte I

Submissões

Utilizando linguagens de programação orientadas a objetos para codificar programas adaptativos

P. R. M. Cereda e J. José Neto

***Abstract*—This paper presents concepts and techniques for writing adaptive code using a normal object-oriented programming language as reference. We aim at making adaptivity accessible to users via well-known object-oriented constructs, as a feasible alternative to the traditional development approaches.**

***Palavras-chave:*—Adaptatividade, linguagens de programação, orientação a objetos**

I. Introdução

Adaptatividade é o termo utilizado para denotar a capacidade de um dispositivo em modificar seu próprio comportamento, sem a interferência de agentes externos [1], [2]. A modificação espontânea ocorre em resposta ao histórico de operação e aos dados de entrada, resultando em comportamentos distintos ao longo do tempo [3]. A adaptatividade é adicionada como uma extensão aos formalismos já consolidados, aumentando seu poder de expressão [4].

De modo particular, a adaptatividade tem sido associada com sucesso às linguagens de programação, como pode ser observado em diversas publicações [5], [6], [7], [8], [9], [10]. Em [6], Freitas e José Neto, inspirados em um trabalho anterior de Rocha e José Neto [5], iniciaram o desenvolvimento de linguagens de programação orientadas a adaptatividade. Os autores enfatizam a necessidade de uma mudança de postura em relação às iniciativas existentes para o tratamento do fenômeno da adaptatividade juntamente com um estilo de programação. Linguagens de programação estendidas com construtos adaptativos permitem um fluxo de trabalho consistente para aplicações que requerem código auto-modificável em tempo de execução [6].

Um programa escrito em uma linguagem de programação com características adaptativas tem, no início de sua execução, um comportamento semelhante ao de um código não-adaptativo tradicional [7]. Tal programa inicia a partir de uma configuração de código inicial C_1 e evolui para configurações sucessivas $C_2 \ldots C_n$ no tempo através de ações adaptativas [1], conforme ilustra a Figura 1. A sequência de configurações é fortemente dependente dos dados de entrada; execuções do mesmo programa com dados diferentes consequentemente resultarão em sequências diferentes.

Inspirado nas recentes contribuições em linguagens de programação com características adaptativas e na expressão do próprio fenômeno da adaptatividade, este artigo apresenta conceitos e técnicas para a escrita de código adaptativo utilizando linguagens de programação orientadas a objetos convencionais. O atual estado da arte apresenta iniciativas baseadas em um subconjunto de código de montagem adaptativo [9] e especificações para linguagens de alto nível [8], [10]; este artigo contempla técnicas para a incorporação dos conceitos adaptativos em linguagens de programação orientadas a objetos já conhecidas.

Os autores podem ser contatados através dos seguintes endereços de correio eletrônico: paulo.cereda@usp.br e jjneto@usp.br.

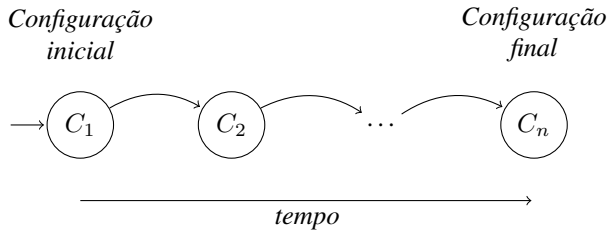


Figura 1. Evoluções sucessivas de código de um programa escrito em linguagem de programação com características adaptativas no tempo, através de ações adaptativas.

É importante observar que as técnicas apresentadas estão vinculadas ao estilo adaptativo e à própria adaptatividade. Em outras palavras, em algum nível, o desenvolvedor necessita entender como um código adaptativo funciona e como este código evoluirá entre configurações. Existe um trabalho em andamento para oferecer uma ferramenta gráfica com o objetivo de auxiliar o desenvolvedor na escrita de código adaptativo. A ideia é encapsular a adaptatividade em uma biblioteca de código, oferecendo uma interface fluente e transparente ao desenvolvedor, minimizando a necessidade de familiaridade com o estilo adaptativo como pré-requisito para a escrita de código adaptativo.

Este artigo está organizado da seguinte forma: a Seção II apresenta uma revisão da formulação geral para dispositivos adaptativos guiados por regras. A Seção III introduz duas técnicas para a escrita de código adaptativo, incluindo algumas discussões sobre implementação e trechos de código em Java e C#. A Seção IV apresenta um estudo de caso de duas implementações de um dispositivo adaptativo para o reconhecimento de cadeias na forma $a^n b^n c^n$, $n \in \mathbb{N}$, utilizando as técnicas apresentadas na seção anterior; vários testes são realizados nas implementações e os resultados são discutidos. As considerações finais são apresentadas na Seção V.

II. Dispositivos adaptativos

Esta seção apresenta uma revisão da formulação geral para dispositivos adaptativos guiados por regras, que são máquinas formais que permitem a alteração de seu comportamento, de modo dinâmico e espontâneo, como resposta a estímulos de entrada, sem a interferência de agentes externos.

A. Dispositivos guiados por regras

Um dispositivo guiado por regras é todo e qualquer dispositivo formal cujo comportamento depende, de forma exclusiva, de um conjunto finito de regras que definem o mapeamento entre configurações [11]. Em outras palavras, um dispositivo guiado por regras pode ser definido pela sêxtupla $ND = (C, NR, S, c_0, A, NA)$, onde ND é um dispositivo guiado por regras, C é o conjunto de todas as possíveis configurações, $c_0 \in C$ é a configuração inicial, S é o conjunto de todos os possíveis eventos que são estímulos de entrada do dispositivo, $\epsilon \in S$, $A \subseteq C$ é o subconjunto de todas as configurações de aceitação (da mesma forma, $F = C - A$ é o subconjunto das configurações de rejeição), NA é o conjunto de todos os possíveis símbolos de saída de ND como efeito da aplicação das regras do dispositivo, $\epsilon \in NA$, e NR é o conjunto de regras que definem ND através de uma relação $NR \subseteq C \times S \times C \times NA$. Uma regra $r \in NR$ tem a forma $r = (c_i, s, c_j, z)$, $c_i, c_j \in C$, $s \in S$ e $z \in NA$, indicando que, em resposta a um estímulo s, r muda a configuração corrente c_i para c_j, consome s e gera z como saída [11]. Uma regra $r = (c_i, s, c_j, z)$ é dita compatível com a configuração corrente c se, e somente se, $c_i = c$ e s é vazio ou igual ao estímulo de entrada corrente; neste caso, a aplicação da regra r move o dispositivo para a configuração c_j, denotada por $c_i \Rightarrow^s c_j$, e adiciona z ao fluxo de saída.

Um fluxo de estímulos de entrada $w = w_1 w_2 \dots w_n$, $w_k \in S - \{\epsilon\}$, $k = 1, \dots, n$, $n \geq 0$, é aceito por um dispositivo ND quando $c_0 \Rightarrow^{w_1} c_1 \Rightarrow^{w_2} \dots \Rightarrow^{w_n} c$ (de modo resumido, $c_0 \Rightarrow^w c$), e $c \in A$. Da mesma forma, w é rejeitado por ND quando $c \in F$. A linguagem descrita pelo dispositivo guiado por regras ND é representada pelo conjunto $L(ND) = \{w \in S^* \mid c_0 \Rightarrow^w c, c \in A\}$.

B. Dispositivos adaptativos guiados por regras

Um dispositivo guiado por regras $AD = (ND_0, AM)$ é considerado adaptativo sempre que, para todos os passos de operação $k \geq 0$ (k é o valor de um contador embutido T iniciado em zero e que é incrementado de uma unidade toda vez que uma ação adaptativa não nula é executada), AD segue o comportamento do dispositivo subjacente ND_k até que a execução de uma ação adaptativa não nula inicie o passo de operação $k + 1$ através de mudanças no conjunto de regras; em outras palavras, a execução uma ação adaptativa não nula em um passo de operação $k \geq 0$ faz o dispositivo adaptativo AD evoluir do dispositivo subjacente ND_k para ND_{k+1}.

O dispositivo adaptativo AD inicia sua operação na configuração c_0, com o formato inicial definido por $AD_0 = (C_0, AR_0, S, c_0, A, NA, BA, AA)$. No passo k, um estímulo de entrada move AD para uma configuração seguinte e inicia seu passo de operação $k + 1$ se, e somente se, uma ação não-adaptativa for executada; dessa forma, estando o dispositivo AD no passo k, com o formato $AD_k = (C_k, AR_k, S, c_k, A, NA, BA, AA)$, a execução de uma ação adaptativa não nula leva a $AD_{k+1} = (C_{k+1}, AR_{k+1}, S, c_{k+1}, A, NA, BA, AA)$, onde $AD = (ND_0, AM)$ é um dispositivo adaptativo com um dispositivo subjacente inicial ND_0 e um mecanismo adaptativo AM, ND_k é o dispositivo subjacente de AD no passo de operação k, NR_k é o conjunto de regras não adaptativas de ND_k, C_k é o conjunto de todas as configurações possíveis para ND no passo de operação k, $c_k \in C_k$ é a configuração inicial no passo k, S é o conjunto de todos os eventos possíveis que são estímulos de entrada para AD, $A \subseteq C$ é o subconjunto as configurações de aceitação (da mesma forma, $F = C - A$ é o subconjunto de configurações de rejeição), BA e AA são conjuntos de ações adaptativas (ambos contendo a ação nula, $\epsilon \in BA \cap AA$), NA, com $\epsilon \in NA$, é o conjunto de todos os possíveis símbolos de saída de AD como efeito da aplicação de regras do dispositivo, AR_k é o conjunto de regras adaptativas definido pela relação $AR_k \subseteq BA \times C \times S \times C \times NA \times AA$, com AR_0 definindo o comportamento inicial de AD, AR é o conjunto de todas as possíveis regras adaptativas para AD, NR é o conjunto de todas as possíveis regras não-adaptativas subjacentes de AD, e AM é o mecanismo adaptativo, $AM \subseteq BA \times NR \times AA$, a ser aplicado em um passo de operação k para cada regra em $NR_k \subseteq NR$. Regras adaptativas $ar \in AR_k$ são da forma $ar = (ba, c_i, s, c_j, z, aa)$ indicando que, em resposta a um estímulo de entrada $s \in S$, ar inicialmente executa a ação adaptativa anterior $ba \in BA$; a execução de ba é cancelada se esta elimina ar do conjunto AR_k; caso contrário, a regra não-adaptativa subjacente $nr = (c_i, s, c_j, z)$, $nr \in NR_k$ é aplicada e, finalmente, a ação adaptativa posterior $aa \in AA$ é executada [11].

Ações adaptativas podem ser definidas em termos de abstrações chamadas funções adaptativas, de modo similar às chamadas de funções em linguagens de programação usuais [11]. A especificação de uma função adaptativa deve incluir os seguintes elementos: *(a)* um nome simbólico, *(b)* parâmetros formais que referenciarão valores passados como argumentos, *(c)* variáveis que conterão valores de uma aplicação de uma ação elementar de inspeção, *(d)* geradores que referenciam valores novos a cada utilização, e *(e)* o corpo da função propriamente dita.

São definidos três tipos de ações adaptativas elementares que realizam testes nas regras ou modificam regras existentes, a saber:

1) *ação adaptativa elementar de inspeção:* a ação não modifica o conjunto de regras, mas permite a inspeção deste para a verificação de regras que obedeçam um determinado padrão.
2) *ação adaptativa elementar de remoção:* a ação remove regras que correspondem a um determinado padrão do conjunto corrente de regras.
3) *ação adaptativa elementar de inclusão:* a ação insere uma regra que corresponde a um determinado padrão no conjunto corrente de regras.

Tais ações adaptativas elementares podem ser utilizadas no corpo de uma função adaptativa, incluindo padrões de regras que utilizem parâmetros formais, variáveis e geradores disponíveis.

Como exemplo, considere dispositivo adaptativo, mais precisamente um autômato adaptativo M, que reconhece cadeias pertencentes à linguagem $L(M) = \{w \in \{a, b\}^* \mid w =$

$a^n b^n, n \in \mathbb{N}, n \geq 1\}$, apresentado na Figura 2. A função adaptativa $\mathcal{F}$ é apresentada no Algoritmo 1.

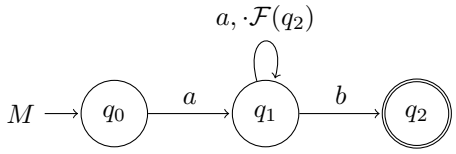


Figura 2. Autômato adaptativo M que reconhece cadeias na forma $a^n b^n$, com $n \in \mathbb{N}$, $n \geq 1$.

Algoritmo 1 Função adaptativa $\mathcal{F}(p)$

função adaptativa $\mathcal{F}(p)$
variáveis: $?x$
geradores: $g^\star$
// procura pelo estado que possui uma transição
// consumindo o símbolo b até p e remove essa transição
$?(?x, b) \rightarrow p$
$-(?x, b) \rightarrow p$
// remove o laço existente em q_1
$-(q_1, a) \rightarrow q_1, \cdot\mathcal{F}(p)$
// adiciona as novas transições
$+(?x, b) \rightarrow g^\star$
$+(g^\star, b) \rightarrow p$
// adiciona um novo laço em q_1
$+(q_1, a) \rightarrow q_1, \cdot\mathcal{F}(g^\star)$
fim da função adaptativa

Durante o processo de reconhecimento de uma cadeia w, M sofrerá alterações em sua topologia para acomodar mais símbolos. No caso da linguagem $L(M)$, a cada ocorrência de um símbolo a adicional (indicada por um laço em q_1), a função adaptativa posterior $\mathcal{F}$ será executada para que M possa receber um símbolo b adicional. A Figura 3 ilustra o autômato M resultante após o reconhecimento da cadeia $w = aabb$.

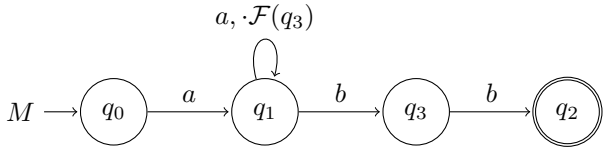


Figura 3. Autômato adaptativo M resultante após o reconhecimento da cadeia $w = aabb$.

É importante destacar que o mecanismo adaptativo pode ser visto como uma simples extensão do dispositivo não-adaptativo subjacente, o que preserva a integridade e as propriedades deste [11].

III. Técnicas para a escrita de código adaptativo

Esta seção apresenta duas técnicas para a escrita de código adaptativo utilizando uma linguagem de programação como referência. As linguagens Java e C# foram escolhidas para fins de ilustração, mas é importante mencionar que qualquer linguagem de programação pode ser utilizada, desde que alguns requisitos mínimos estejam presentes nas especificações da linguagem escolhida.

A. Reflexão estrutural

A primeira técnica para a escrita de código adaptativo explora reflexão, uma característica que permite a análise e modificação de classes em tempo de execução. Todas as informações sobre uma classe, incluindo seus métodos e atributos, estão disponíveis como metadados que são preservados durante a compilação. Considere um exemplo de uma classe simples, apresentada na Figura 4.

Classe em Java

```
public class Bar {

  private int a;
  private String b;
  private double c;

  public int sum(int n, int m) {
    return n + m;
  }
}
```

Classe em C#

```
public class Bar {

  private int a;
  private string b;
  private double c;

  public int sum(int n, int m) {
    return n + m;
  }
}
```

Figura 4. Exemplo de uma classe.

É possível escrever um trecho de código que, utilizando reflexão, analisa a classe `Bar` da Figura 4 e retorna todas as informações sobre seus atributos e métodos, como visto na Figura 5. É importante observar que tal análise da classe ocorre em tempo de execução.

A execução do trecho de código apresentado na Figura 5 retorna uma lista de atributos e métodos definidos na classe `Bar` (Figura 4) através de classes utilitárias que oferecem reflexão para as duas linguagens.

A possibilidade de uma linguagem de programação usar reflexão permite a construção de programas auto-modificáveis, isto é, programas capazes de alterar seu próprio código. Entretanto, a maioria das linguagens de programação, incluindo Java e C#, apresentam restrições em suas classes utilitárias de reflexão: apenas ações de introspecção são permitidas, isto é, a capacidade de analisar estruturas [12], [13], [14]. Assim, diversas extensões para linguagens foram propostas para oferecer suporte à reflexão comportamental – interceptar uma operação tal como uma invocação de método e mudar o

Reflexão em Java

```
for (Field field :
      Bar.class.
      getDeclaredFields()) {
  System.out.println(field.
      toString());
}
for (Method method :
      Bar.class.
      getDeclaredMethods()) {
  System.out.println(method.
      toString());
}
```

Reflexão em C#

```
Type type = typeof(Bar);
foreach (FieldInfo field in
      type.GetFields(
      BindingFlags.NonPublic |
      BindingFlags.Instance)) {
  Console.WriteLine(field);
}
foreach (MethodInfo method in
      type.GetMethods()) {
  System.Console.
      WriteLine(method);
}
```

Figura 5. Trecho de código que analisa a classe `Bar` e retorna todas as informações sobre seus atributos e métodos.

comportamento dessa operação [15] – e à reflexão estrutural – oferecer a possibilidade de modificar estruturas de dados [16].

A seguir, é apresentada uma iniciativa interessante do uso de reflexão estrutural em Java através da biblioteca Javassist [17], [16], [18]. Tal característica é oferecida através de uma transformação de bytecode que pode acontecer tanto em tempo de compilação quanto em tempo de carga. As modificações não são permitidas em tempo de execução por causa de uma limitação da linguagem: uma vez que uma classe é instanciada na memória, todas as tentativas de modificar sua estrutura lançarão uma exceção [12], [13]. Entretanto, é possível utilizar o conceito de herança do paradigma de programação orientada a objetos para estender a classe-alvo (já instanciada), alterar a estrutura da nova classe em tempo de execução, instanciá-la e definir uma nova referência para a variável contendo o objeto ascendente original, como ilustrado na Figura 6. É importante mencionar que o mesmo conceito utilizado na linguagem Java aplica-se para a linguagem C#.

De acordo com a Figura 6, todas as modificações são realizadas na nova classe, tendo a classe original como sua antecedente. Agora é possível retornar um novo objeto da nova classe que é compatível com as especificações originais, mas seus atributos e métodos podem estar modificados.

A Figura 7 ilustra a criação de uma nova classe, `Baz`, que herda da classe original `Bar` (Figura 4), e sua modificação posterior, adicionando um novo atributo. A biblioteca Javassist [17], [16], [18] foi utilizada para facilitar a modificação

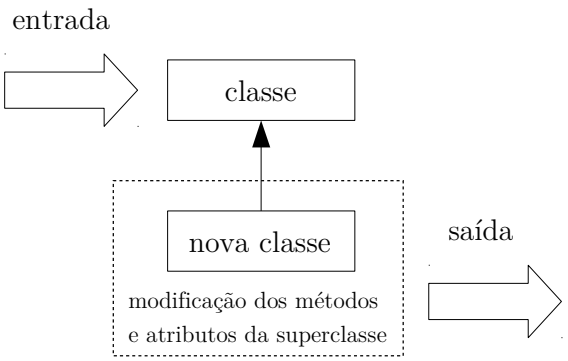


Figura 6. Estendendo a classe-alvo e alterando a nova classe.

de código.

```
ClassPool pool = ClassPool.getDefault();
CtClass clazz = pool.get(Bar.class.
    getCanonicalName());
CtClass newClazz = pool.makeClass("Baz");
newClazz.setSuperclass(clazz);
newClazz.addField(new CtField(pool.get(
    Float.class.getCanonicalName()),
     "d", newClazz));
newClazz.toClass();
Class createdClazz = Class.
    forName("Baz");
```

Figura 7. Criação de uma nova classe `Baz` através de herança e reflexão estrutural.

Como visto na Figura 7, através de herança, `Baz` tem todos os atributos e métodos da classe original `Baz` com a adição de um novo atributo de ponto flutuante `d`. É possível repetir a extensão de classes exaustivamente através da modificação de atributos e métodos, quando apropriado.

A reflexão estrutural é uma característica interessante para o desenvolvimento de código adaptativo, pois permite modificações em tempo de execução. Entretanto, uma deficiência do uso de reflexão é o tempo de execução e consumo de memória consideráveis devido à ausência de otimizações do compilador – código injetado via reflexão não existe em tempo de compilação – eao fato de que cada modificação simples requer descoberta, isto é, toda a estrutura de classe requer uma inspeção completa de modo a encontrar quais são os pontos modificáveis. Além disso, quando existe a utilização de bibliotecas de manipulação de bytecode, como Javassist, deve-se considerar também o processo de reconstrução de bytecode ao criar novas classes.

B. Blocos de código

Como uma alternativa à reflexão estrutural e suas deficiências, a segunda técnica aplica um conceito utilizado por simuladores e máquinas virtuais: um conjunto de blocos de código, conectados por ligações simbólicas, e um executor

que roda cada bloco, um por vez. Linguagens de programação modernas possuem suporte a threading, um componente utilitário que permite a representação de blocos de código. A motivação para as interfaces de threading surge da necessidade de oferecer um protocolo comum para objetos executarem um determinado trecho de código durante seus ciclos de vida. A Figura 8 ilustra um esboço da estrutura de execução de um programa implementado com blocos de código.

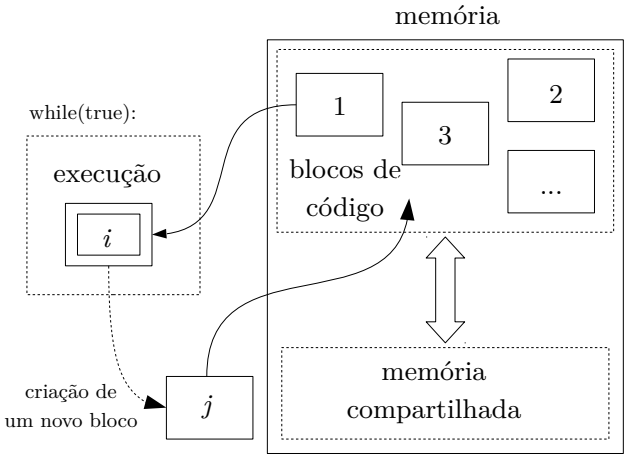


Figura 8. Esboço da estrutura de execução de um programa implementado com blocos de código.

A Figura 8 esboça uma execução típica de um programa implementado com blocos de código, consistindo de um laço principal e a execução sequencial de cada bloco até que uma instrução de parada seja encontrada. Uma possível implementação de um bloco de código é apresentada na Figura 9. Para o escopo do artigo, optou-se pela implementação da interface `Runnable` em Java e a utilização da classe `Thread` com `delegate` em C#.

Um bloco de código em Java

```
Runnable r = new Runnable() {
  public void run() {
    // implementation
  }
};
```

Um bloco de código em C#

```
Thread t = new Thread(
  delegate() {
    // implementation
  }
);
```

Figura 9. Uma possível implementação de um bloco de código.

Uma implementação mais sofisticada de um bloco de código em Java e sua utilização é ilustrada na Figura 10. A classe `CodeBlock` implementa parcialmente a interface `Runnable` e define um mapa de memória e um identificador único para distinguir o bloco corrente dos outros. A implementação completa de um bloco de código ocorre no programa principal, através da especificação do corpo do método `run()`.

```
public abstract class CodeBlock
     implements Runnable {
  private HashMap<String,
     Object> memory;
  private String identifier;
  public abstract void run();
  public CodeBlock(
       HashMap<String,
       Object> memory,
       String identifier) {
    this.memory = memory;
    this.identifier = identifier;
  }
...
}
...
CodeBlock q0 = new CodeBlock(
     memory, "q0") {
  @Override
  public void run() {
    System.out.
        println("Hello world!");
    }
};
```

Figura 10. Uma implementação mais sofisticada de um bloco de código em Java e sua utilização.

O bloco de código `q0`, como visto na Figura 10, simplesmente imprime uma mensagem de saudação no terminal. É importante notar que blocos de código podem conter definições aninhadas de outros blocos de código em suas especificações. Assim, novos blocos podem ser criados e novas ligações dinâmicas podem ser estabelecidas.

Usar blocos de código na representação de componentes de um programa adaptativo é uma solução viável para implementar características adaptativas em linguagens de programação que suportam threading. Entretanto, é importante observar que tal solução requer um alto nível de especificação do que seu equivalente reflexivo, isto é, blocos de código são definidos com operações básicas ao invés de métodos completos, podendo gerar uma quantidade considerável de blocos para representar um algoritmo.

IV. Experimentos

Esta seção apresenta um estudo de caso consistindo em duas implementações de um dispositivo adaptativo, mais precisamente um autômato adaptativo, que reconhece cadeias na forma $a^n b^n c^n$, com $n \in \mathbb{N}$, $n \geq 1$ [1], ilustrado na Figura 11. A função adaptativa $\mathcal{A}$ presente em M é apresentada no Algoritmo 2. A primeira implementação utiliza reflexão estrutural para realizar as ações adaptativas, disparadas por uma função adaptativa definida no conjunto de regras do dispositivo. A segunda implementação usa blocos de código para representar cada estado do dispositivo, e ações adaptativas são disparadas através da criação de novos blocos e da modificação das ligações dinâmicas existentes entre blocos em tempo de execução.

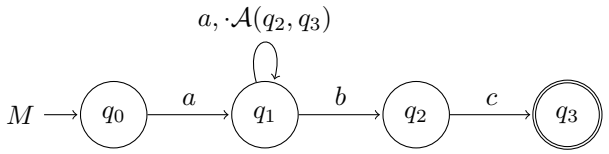
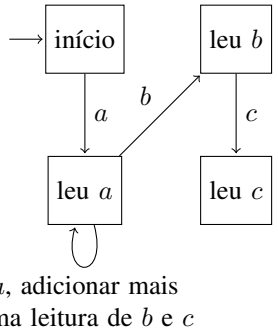


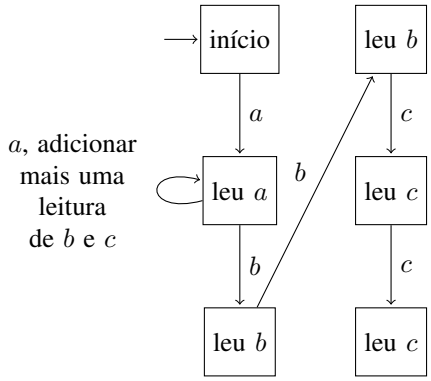
Figura 11. Autômato adaptativo M que reconhece cadeias na forma $a^n b^n c^n$, com $n \in \mathbb{N}$, $n \geq 1$.

Algoritmo 2 Função adaptativa $\mathcal{A}(p_1, p_2)$

função adaptativa $\mathcal{A}(p_1, p_2)$
variáveis: $?x, ?y$
geradores: $g_1^\star, g_2^\star$
// procura pelo estado que possui uma transição
// consumindo o símbolo b até p_1 e remove essa transição
$?(?x, b) \rightarrow p_1$
$-(?x, b) \rightarrow p_1$
// procura pelo estado que possui uma transição
// consumindo o símbolo c até p_2 e remove essa transição
$?(?y, c) \rightarrow p_2$
$-(?y, c) \rightarrow p_2$
// remove o laço existente em q_1
$-(q_1, a) \rightarrow q_1, \cdot\mathcal{A}(p_1, p_2)$
// adiciona as novas transições
$+(?x, b) \rightarrow g_1^\star$
$+(g_1^\star, b) \rightarrow p_1$
$+(?y, c) \rightarrow g_2^\star$
$+(g_2^\star, c) \rightarrow p_2$
// adiciona um novo laço em q_1
$+(q_1, a) \rightarrow q_1, \cdot\mathcal{A}(g_1^\star, g_2^\star)$
fim da função adaptativa



Considere o reconhecimento da cadeia $aabbcc$. Ao ler o segundo a da cadeia, o programa realizará modificações para acomodar novas situações de leitura de símbolos.



Após a leitura do segundo símbolo a da cadeia, o programa adicionou duas leituras em seu fluxo de execução. Caso mais um símbolo a seja lido, o programa repetirá a adição de mais leituras.

Figura 12. Lógica do programa de reconhecimento de cadeias na forma $a^n b^n c^n$, com $n \in \mathbb{N}$, $n \geq 1$.

As duas implementações propostas seguem a lógica apresentada na Figura 12: quando há a ocorrência de um símbolo a adicional, o programa sofrerá modificações em seu fluxo de execução para acomodar as novas situações de leitura de símbolos.

Dois experimentos foram executados para cada implementação: o primeiro consistiu na medição, em milissegundos, do tempo de reconhecimento de cada cadeia w, $w = a^n b^n c^n$, com $1 \leq n \leq 310$, durante 1000 iterações. O segundo experimento consistiu na medição, em bytes, do tamanho de objeto de cada componente implementando o motor adaptativo, e o consumo de memória para cada cadeia w, $w = a^n b^n c^n$, com $1 \leq n \leq 310$, também durante 1000 iterações. Os experimentos foram realizados em um ambiente Linux de 64 bits, utilizando uma máquina virtual Oracle JRockit. Cada valor de medição obtido a partir de cada iteração foi coletado para um processamento posterior, com a geração de estatísticas.

Os resultados do primeiro experimento são sintetizados na Figura 13. O programa reflexivo apresentou uma linha mais íngreme, mesmo para valores pequenos de n, do que seu equivalente não-reflexivo. Para $n = 310$, obteve-se um tempo de $8395{,}13\,\mathrm{ms}$ do programa reflexivo em comparação a $56{,}01\,\mathrm{ms}$ do programa utilizando blocos de código.

De acordo com a Figura 13, o programa reflexivo requer muito mais tempo do que seu equivalente não-reflexivo. No estudo de caso proposto, existem $n-1$ chamadas de uma função adaptativa para cada valor n, o que significa que, ao longo do tempo, mudanças no código tornam-se extremamente onerosas (com inclinação $= 2{,}4239 \times 10^1$). O programa utilizando blocos de código obteve um desempenho significativamente melhor (com inclinação $= 1{,}744 \times 10^{-1}$).

Os resultados do segundo experimento são exibidos nas Figuras 14 e 15. A primeira parte do experimento consistiu na medição do tamanho de objeto, em bytes, de cada componente implementando o motor adaptativo, enquanto que a segunda parte analisou o consumo de memória.

De acordo com a Figura 14, é possível notar que o programa reflexivo produz um tamanho de objeto menor do que seu equivalente não-reflexivo no início do experimento, para valores muito pequenos de n, mas termina com uma taxa de crescimento mais acentuada ao longo do tempo (com inclinação $= 9{,}95 \times 10^2$). O programa utilizando blocos de código teve um crescimento aparentemente linear durante as iterações, gerando um modelo de objeto mais conciso

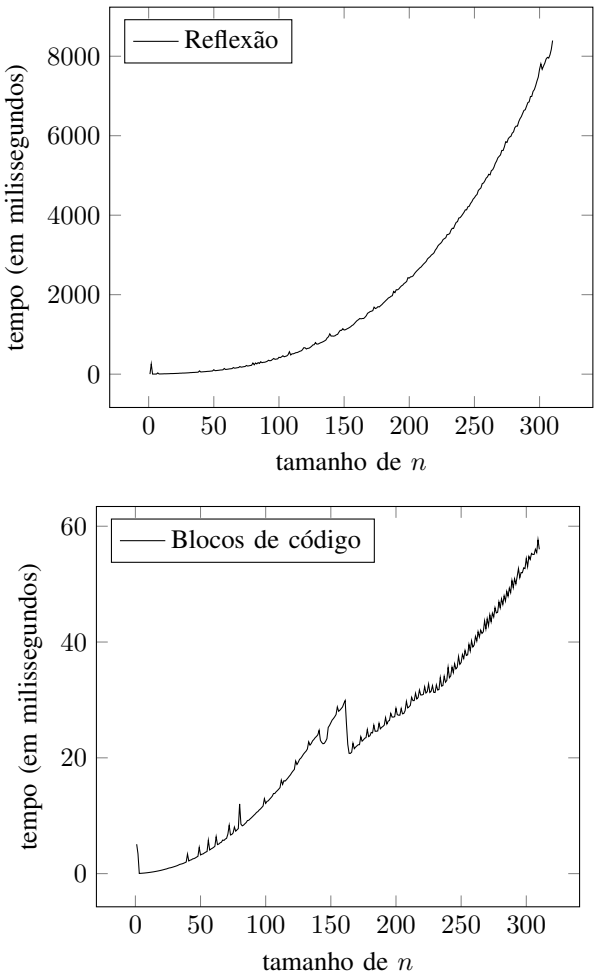


Figura 13. Primeiro experimento, medindo o tempo de reconhecimento de cada cadeia w, $w = a^n b^n c^n$, com $1 \leq n \leq 310$, durante 1000 iterações.

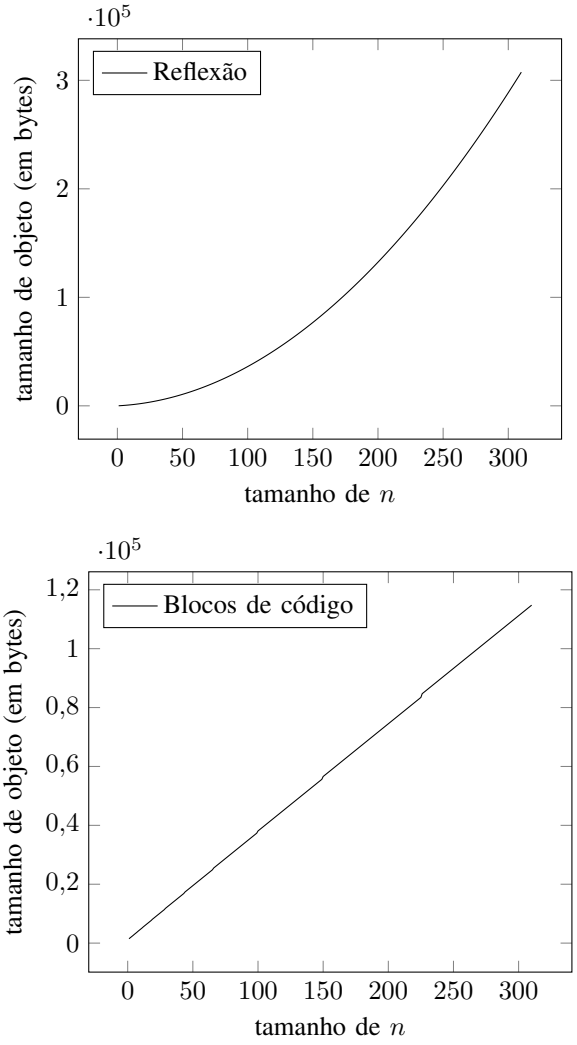


Figura 14. Primeira parte do segundo experimento, medindo o tamanho de objeto, em bytes, de cada componente implementando o motor adaptativo para cada cadeia w, $w = a^n b^n c^n$, com $1 \leq n \leq 310$, durante 1000 iterações.

(com inclinação $= 3{,}6774 \times 10^2$). A análise do consumo de memória é apresentada na Figura 15.

É possível observar, de acordo com os resultados da Figura 15, que o programa reflexivo teve um consumo de memória íngreme (com inclinação $= 3{,}1503 \times 10^9$) devido à enorme quantidade de buscas na estrutura da classe para definir os pontos de entrada e realizar as modificações. O programa utilizando blocos de código teve um consumo de memória significativamente menor (com inclinação $= 1{,}4171 \times 10^8$), mesmo para grandes valores de n. É importante notar que o processo de coleta de lixo (*garbage collection*) ocorre frequentemente durante a execução do programa utilizando blocos de código e, portanto, reduzindo o consumo de memória de tempos em tempos; no programa reflexivo, devido à utilização do conceito de herança, toda a hierarquia de classes deve ser preservada, portanto a coleta de lixo não pode ser realizada.

V. Considerações finais

O estudo de caso apresentado na Seção IV indica a viabilidade da implementação de programas adaptativos através da inserção de trechos de código, escritos em linguagens de programação convencionais com características de automodificação. As implementações podem ter a inspiração de técnicas tais como reflexão estrutural ou particionamento por blocos de código, apresentadas neste artigo. Apesar da reflexão ter problemas de desempenho, evidenciados nos testes realizados, ainda assim é uma solução viável para a inspeção de código já existente e injeção de blocos de código com características de adaptatividade quando necessário.

Entretanto, é importante observar que tais técnicas dependem fortemente de conceitos oriundos do estilo adaptativo [6] e da própria adaptatividade. Para minimizar tal dependência e permitir um fluxo de trabalho mais simplificado para a escrita de código adaptativo, uma ferramenta gráfica integrada ao ambiente de desenvolvimento de software está em estudo e análise. A proposta é tornar as características de adaptatividade encapsuladas na forma de bibliotecas adaptativas, escritas em linguagens de programação existentes, permitindo que usuários escrevam códigos adaptativos sem a necessidade imediata de conhecer os aspectos de implementação do estilo adaptativo.

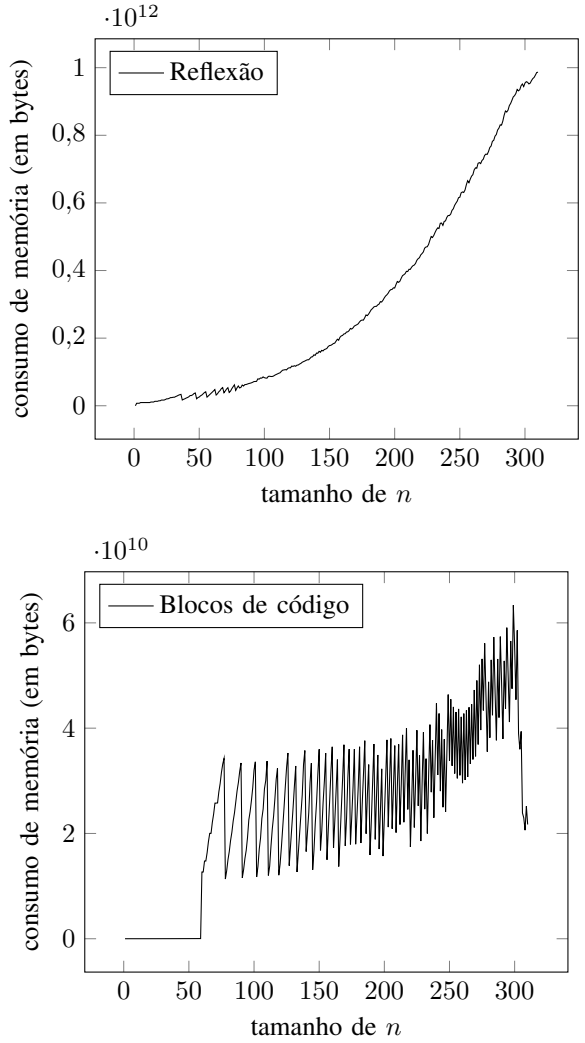


Figura 15. Segunda parte do segundo experimento, medindo o consumo de memória de cada cadeia w, $w = a^n b^n c^n$, com $1 \leq n \leq 310$, durante 1000 iterações.

REFERÊNCIAS

[1] J. José Neto, "Contribuições à metodologia de construção de compiladores," Thesis (Livre-docência), Escola Politécnica, Universidade de São Paulo, 1993.

[2] ——, "Adaptive automata for context-dependent languages," *SIGPLAN Notices*, vol. 29, no. 9, pp. 115–124, 1994.

[3] ——, "Um levantamento da evolução da adaptatividade e da tecnologia adaptativa," *IEEE Latin America Transactions*, vol. 5, pp. 496–505, 2007.

[4] H. Pistori, "Tecnologia em engenharia de computação: estado da arte e aplicações," PhD thesis, Escola Politécnica, Universidade de São Paulo, 2003.

[5] R. L. A. Rocha and J. José Neto, "Uma proposta de linguagem de programação funcional com características adaptativas," in *IX Congreso Argentino de Ciencias de la Computación*, 2003.

[6] A. V. Freitas and J. José Neto, "Adaptive languages and a new programming style," in *WSEAS International Conference on Applied Computer Science*, Tenerife, Canary Islands, Spain, 2006.

[7] A. V. Freitas, "Considerações sobre o desenvolvimento de linguagens adaptativas de programação," PhD thesis, Escola Politécnica, Universidade de São Paulo, 2008.

[8] A. A. Castro Junior, "Aspectos de projeto e implementação de linguagens para codificação de programas adaptativos," PhD thesis, Escola Politécnica, Universidade de São Paulo, 2009.

[9] E. J. Pelegrini, "Códigos adaptativos e linguagem de programação adaptativa: conceitos e tecnologias," Master's thesis, Escola Politécnica, Universidade de São Paulo, 2009.

[10] S. R. B. Silva, "Software adaptativo: método de projeto, representação gráfica e implementação de linguagem de programação," Master's thesis, Escola Politécnica, Universidade de São Paulo, 2011.

[11] J. José Neto, "Adaptive rule-driven devices: general formulation and case study," in *International Conference on Implementation and Application of Automata*, 2001.

[12] J. Gosling, B. Joy, G. Steele, G. Bracha, and A. Buckley, "The Java language specification, Java SE 7 edition," Oracle America, Inc., Tech. Rep., 2011.

[13] T. Lindholm, F. Yellin, G. Bracha, and A. Buckley, "The Java virtual machine specification, Java SE 7 edition," Oracle America, Inc., Tech. Rep., 2011.

[14] A. Hejlsberg, S. Wiltamuth, and P. Golde, *C# Language Specification*. Addison-Wesley Longman Publishing, 2003.

[15] Y. Hassoun, R. Johnson, and S. Counsell, "Reusability, open implementation and Java's dynamic proxies," in *Proceedings of the 2nd international conference on Principles and practice of programming in Java*, ser. PPPJ'03. New York, NY, USA: Computer Science Press, Inc., 2003, pp. 3–6.

[16] S. Chiba, "Load-time structural reflection in Java," in *ECOOP 2000, Object-Oriented Programming*, 2000.

[17] ——, "Javassist, a reflection-based programming wizard for Java," in *Proceedings of the ACM OOPSLA, Workshop on Reflective Programming in C++ and Java*, 1998.

[18] S. Chiba and M. Nishizawa, "An easy-to-use toolkit for efficient Java bytecode translators," in *International Conference on Generative Programming and Component Engineering*, 2003.

Paulo Roberto Massa Cereda é graduado em Ciência da Computação pelo Centro Universitário Central Paulista (2005) e mestre em Ciência da Computação pela Universidade Federal de São Carlos (2008). Atualmente, é doutorando do Programa de Pós-Graduação em Engenharia de Computação do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo, atuando como aluno pesquisador no Laboratório de Linguagens e Técnicas Adaptativas do PCS. Tem experiência na área de Ciência da Computação, com ênfase em Teoria da Computação, atuando principalmente nos seguintes temas: tecnologia adaptativa, autômatos adaptativos, dispositivos adaptativos, linguagens de programação e construção de compiladores.

João José Neto é graduado em Engenharia de Eletricidade (1971), mestre em Engenharia Elétrica (1975), doutor em Engenharia Elétrica (1980) e livre-docente (1993) pela Escola Politécnica da Universidade de S ão Paulo. Atualmente, é professor associado da Escola Politécnica da Universidade de São Paulo e coordena o LTA – Laboratório de Linguagens e Técnicas Adaptativas do PCS – Departamento de Engenharia de Computação e Sistemas Digitais da EPUSP. Tem experiência na área de Ciência da Computação, com ênfase nos Fundamentos da Engenharia da Computação, atuando principalmente nos seguintes temas: dispositivos adaptativos, tecnologia adaptativa, autômatos adaptativos, e em suas aplicações à Engenharia de Computação, particularmente em sistemas de tomada de decisão adaptativa, análise e processamento de linguagens naturais, construção de compiladores, robótica, ensino assistido por computador, modelagem de sistemas inteligentes, processos de aprendizagem automática e inferências baseadas em tecnologia adaptativa.

Contribuições à Modelagem Adaptativa da Norma Culta do Português Brasileiro

D. Padovani e J. J. Neto

***Resumo*— Este trabalho apresenta conceitos de processamento de língua natural, abordando diferentes métodos e discorrendo sobre os principais problemas encontrados para a realização desta tarefa. Em seguida, é feita uma revisão dos trabalhos desenvolvidos no tema, com enfoque no processamento do Português do Brasil. Por fim, é apresentada uma proposta de modelo unificado que usa o formalismo adaptativo como modelo teórico subjacente, sem a necessidade de recorrer a técnicas auxiliares.**

Palavras Chave*— *Autômatos Adaptativos, Processamento de Linguagem Natural, Reconhecedores Gramaticais, Gramáticas Livres de Contexto

Processamento da Linguagem Natural

A língua natural é o meio pelo qual os seres humanos se comunicam. Ela é caracterizada pela riqueza semântica, léxica e sintática, que permite a elaboração de textos complexos e com alto grau de abstração, tais como os vistos nas grandes obras da literatura, ou precisos e direcionados, como aqueles encontrados em tratados, trabalhos acadêmicos e científicos. A língua natural não permanece estanque, ao contrário, está sempre em permanente evolução, agregando novos termos e estruturas e eliminando outros, em um processo constante de adaptação às necessidades impostas pela realidade.

Há um grande apelo do ponto de vista dos seres humanos em se comunicar com uma máquina da mesma forma que o fazem entre si. Muitas pessoas encontram dificuldades para utilizar os dispositivos convencionais de interação com os computadores, que, em maior ou menor grau, restringem as possibilidades linguísticas para que as instruções possam ser interpretadas e convertidas em comandos que a máquina possa executar. Estas limitações podem gerar desconforto e, em alguns casos, rejeição; daí a grande procura de computadores e sistemas que interpretem a língua natural. Datam da década de 1940 as primeiras pesquisas que visavam possibilitar o uso da língua natural como forma de interação entre o homem e a máquina [1]. Existem trabalhos que estudam a geração de programas a partir de textos [2], elaboração de tradutores, revisores e corretores gramaticais [3],[4], identificação de padrões e extração de resumos [5], geração automática de padrões de extração [6] e geração automática de taxonomias hierárquicas [7].

O processamento da língua natural requer o desenvolvimento de programas que sejam capazes de determinar e interpretar a estrutura léxico-sintática e semântica das sentenças em muitos níveis de detalhe. As línguas naturais exibem um intricado comportamento estrutural visto que são profusos os casos particulares a serem considerados, devido à extrema generalização ou especialização adotadas nas modelagens usuais das regras gramaticais vigentes na literatura. Uma vez que as línguas naturais nunca são formalmente projetadas, suas regras sintáticas não são simples nem tampouco óbvias e tornam, portanto, complexo o seu processamento computacional. Diversas abordagens são empregadas em sistemas de processamento de língua natural, tais como os métodos exatos, aproximados, pré-definidos ou interativos, inteligentes ou algorítmicos [8]. Independentemente do método utilizado, o processamento da língua natural envolve as operações de análise léxico-morfológica, análise sintática, análise semântica e análise pragmática [9].

Os principais problemas enfrentados no processamento da língua natural estão relacionados ao tratamento das ambiguidades e aos não-determinismos. Rocha [10] aponta que os primeiros resultados animadores no processamento de língua atual foram resultados de técnicas estatísticas. No entanto, ressalta, nenhum modelo estatístico conseguiu resolver os problemas mais complexos de processamento da língua natural e que o uso indiscriminado de modelos estatísticos apenas comprovou a necessidade de evitar uma resposta única. Entre as propostas atuais, o autor relaciona o uso de modelos estatísticos baseados em métodos racionais, o uso de regras lógicas dentro de modelos estatísticos, o uso de modelos racionais com base em algum método estatístico ou o uso de modelos estatístico como base para escolha de modelos não determinísticos.

Nota-se que não há um consenso sobre qual método usar e também não se encontram estudos a respeito dos contextos em que os métodos são mais adequados, nem tampouco, sobre quando são mais eficientes e como transitar de um método para outro. Visto que nenhuma técnica isoladamente resolve completamente o problema do processamento da língua natural, aparentemente existe uma lacuna na literatura relacionada ao tema.

Revisão da Literatura

Ladeira [11] faz um levantamento produção científica nacional realizada no campo de processamento da língua natural nos últimos 40 anos, categorizando-a e analisando a evolução dos grandes temas neste período. Dentre as 621 publicações consideradas da área, a autora definiu um material empírico, constituído por uma amostra de 68 trabalhos, que foi submetido à análise de conteúdo, que tinha por objetivo elucidar as temáticas discutidas pela comunidade científica nacional no campo do processamento da língua natural. A autora conclui que a maior parte da produção científica nacional foi publicada depois do ano 2000. Poucos grupos foram responsáveis por grande parte da produção nacional, sendo a maioria proveniente da ciência da computação, linguística e engenharia elétrica. Com relação à evolução das problemáticas mais discutidas, o trabalho de tradução foi intensamente abordado na década de 70; os estudos com indexação diminuíram a partir da década de 80; e as pesquisas sobre classificação passaram por um período de dormência na

década de 90, notando-se uma clara tendência no desenvolvimento de pesquisas em sumarização automática. A análise de conteúdo realizada nas 68 publicações selecionadas revelou que a recuperação de informação foi a problemática que teve maior destaque na produção científica nacional com 18 trabalhos, seguida da temática de sumarização (10 trabalhos), tratamento de ambiguidade e parsers (10 trabalhos), tradução (9 trabalhos), aplicações para a própria área e exemplos de aplicações de processamento de língua natural (4 trabalhos) e correção automática (3 trabalhos). Dos trabalhos analisados sobre sumarização, observou-se que a maioria das pesquisas tem privilegiado a abordagem empírica para gerar extratos. As pesquisas em tradução automática têm utilizado métodos estatísticos e regras de transferências, com resultados muito próximos.

Especificamente, com relação à problemática de *parsers*, a autora identificou três níveis de análise da língua natural: análise léxico-morfológica, análise sintática e análise semântica. Além disso, ela observou que alguns trabalhos abordaram dois ou até mesmo três níveis de análise. Dentre os trabalhos que propõem análise léxico-morfológica, Neto e Menezes [12] apresentam um método para a construção de um etiquetador morfológico, que pode ser usado em várias línguas. Segundo os autores, existem, basicamente, quatro métodos de etiquetagem morfológica de textos em língua natural: o estatístico, o que se utiliza de regras escritas manualmente, o baseado em regras inferidas automaticamente; e o com base em exemplos memorizados. Os autores acrescentam que todos os métodos utilizam três fontes de informação linguística, extraídas de um corpus de treinamento: os sufixos de palavras, como parte do processo de inferência da etiqueta morfológica de palavras desconhecidas; uma lista de palavras associadas a categorias morfológicas (léxico), para fornecer informações sobre palavras conhecidas; e o contexto próximo ao item lexical que se quer etiquetar (2 ou 3 etiquetas ao redor), para refinar a escolha de sua etiqueta. Assim, o método proposto etiqueta primeiro as palavras conhecidas, depois as desconhecidas usando heurística de acordo com o sufixo, e finalmente faz um refinamento, de acordo com o contexto.

Padilha e Viccari [13] propõem o desenvolvimento de processadores para a morfologia do português utilizando máquinas de estados finitos (transdutores). Os autores alegam que os transdutores são adequados para o processamento morfológico da língua portuguesa, mas ressaltam, como limitações, a sua construção generativa (não há algoritmos de aprendizado de novas transformações, nestes casos, a gramática deve ser alterada e o transdutor reconstruído); e a ausência de pesos para diferenciar mapeamentos ambíguos.

Dentre os trabalhos que abordaram a análise sintática está o trabalho de Bonfante e Nunes [14] que propõem um *parser* probabilístico baseado na noção de núcleos lexicais, na qual, para cada regra observada no conjunto de treinamento, as palavras que não são núcleo são chamadas de modificadores, exercendo influência sobre ele. Segundo as autoras a grande dificuldade de se especificar uma gramática com poder de descrição abriu caminho para a pesquisa empírica. Assim, um conjunto de sentenças anotadas sintaticamente é usado, como dados de treinamento, num processo de aprendizado para realizar a anotação de uma sentença desconhecida. Dentre as abordagens empíricas, as autoras citam o aprendizado de máquina simbólico, conexionista e estatístico. A formação da estrutura sintática de uma sentença se dá através de um processo *bottom-up* comandado pela probabilidade de um núcleo e um modificador se juntarem para formar um sintagma. Utilizou-se um conjunto de sentenças obtidas do corpus NILC e anotadas sintaticamente pelo *parser* PALAVRAS [15].

Julia, Seabra e Semeghini-Siqueira [16] propõem um *parser* que realiza a análise sintática e semântica de afirmações sobre especificação de software expressas de maneira irrestrita em língua natural. O analisador proposto corresponde a uma estrutura, como definido por Piaget [17], que automaticamente gera regras semânticas durante a análise, através de um método heurístico. Segundo os autores, uma estrutura é um sistema de transformações caracterizadas por um grupo de regras. A parte sintática da gramática é expressa por meio de regras, tais como as regras de gramática proposta por Chomsky [18]. O *parser* é baseado em algoritmos de busca que têm como objetivo encontrar um caminho da árvore sintática até um nó folha que contenha uma categoria de significado. A categoria de cada palavra na sentença irá depender da ordem em que ela aparece na sentença.

Sardinha [19] apresenta pesquisas realizadas na área de processamento de língua natural para a Língua Portuguesa, procurando criar um panorama da produção científica não apenas proveniente do Brasil, mas também de Portugal, França e Dinamarca. Nunes et al. [20] apresentam, entre outras ferramentas, o *parser* Curupira, desenvolvido pelo NILC. O Curupira é um analisador de propósito geral para Português do Brasil. Ele analisa sentenças de cima para baixo e da esquerda para a direita através de uma gramática funcional livre de contexto para o padrão escrito Português do Brasil e um léxico de ampla cobertura para Português do Brasil. Este último é um conjunto de 1,5 milhões de formas livres (incluindo formas flexionadas e derivadas), com informações morfossintáticas, tais como *part-of-speech*, número, pessoa, gênero, tempo, aspecto, e transitividade. O Curupira não faz uso de estratégias de poda, redução ou simplificação, procurando etiquetar cada um dos itens lexicais da sentença, por menores ou menos expressivos que sejam. O Curupira também não desambigua a estrutura sintática das sentenças da língua portuguesa, fornecendo todas as classificações sintáticas segundo as prioridades das regras gramaticais que lhe servem de base. O Curupira parte do pressuposto de que as sentenças informadas estão corretas, informando todas as possibilidades sintáticas de acordo a gramática subjacente. O Curupira também não realiza análise semântica e, como trabalha combinando palavras e classes de palavras para fornecer as possíveis estruturas sintáticas, pode gerar classificações que não fazem sentido no contexto.

Bick [15] apresenta pesquisa desenvolvida pelo projeto VISL – *Visual Interactive Syntax Learning*, sediado na Universidade do Sul da Dinamarca, na qual também usa a abordagem determinística no desenvolvimento do *parser* PALAVRAS, um analisador reducional que seleciona as etiquetas com base em regras de constrição da Gramática Constritiva proposta por Karlson [21]. Sardinha [19] explica que esta não é uma gramática tradicional, que explica as categorias gramaticais e sintáticas, e, sim, um conjunto de

regras formalizadas de tal modo que possam ser usadas por computadores. O PALAVRAS procura desambiguar possíveis interpretações morfológicas através da aplicação de regras que utilizam condições contextuais para restringir as possíveis classificações, selecionando, ao final, a etiqueta mais adequada. Bick explica que, no nível sintático, o *parser* trabalha com regras produtivas e restritivas, sendo que as primeiras mapeiam etiquetas ambíguas e as últimas rejeitam as etiquetas com base no contexto.

Uma abordagem estatística é apresentada por Marques e Lopes [22]. Os autores partem do pressuposto de que para realizar a tarefa de etiquetagem morfossintática é necessário um grande volume de texto anotado manualmente, com centenas de milhares de palavras desambiguadas morfossintaticamente e dizem não existir corpora com as dimensões necessárias na língua portuguesa, nem tampouco disponibilidade para a construção de um corpus com estas características. Os autores também refutam a construção de um sistema baseado em regras, pois estes também requerem a interferência de uma pessoa com conhecimento para fornecer regras tão específicas e dependentes do texto que o sistema vai processar. A alternativa que propõem é o uso de uma rede neuronal combinada com um sistema de análise lexical e um texto manualmente etiquetado com 5400 palavras. Segundo os autores a precisão de etiquetagem obtida por este modelo ultrapassa ligeiramente 98%. Outra abordagem estatística é empregada por Dias e Lopes [23], na extração automática de unidades poli lexicais para o português. Os autores defendem que o uso de regras sintáticas para extrair unidades poli lexicais contíguas requer um conhecimento muito profundo da língua, tonando a abordagem pouco flexível para aplicação a novas línguas, assim como pouco adequada ao caráter evolutivo e dinâmico da língua.

Neto [24] apresenta outro tipo de abordagem com a introdução do conceito de adaptatividade. Segundo o autor, dispositivos adaptativos são abstrações matemáticas capazes de se modificarem dinamicamente, criando formalismos capazes de se auto modificarem autonomamente. São compostos de um conjunto de regras, denominado dispositivo subjacente, tipicamente não adaptativo, e de um mecanismo adaptativo cuja conexão ao dispositivo subjacente proporciona-lhe os recursos complementares necessários para a realização de tarefas responsáveis pela auto modificação autônoma que os caracteriza. O mecanismo adaptativo é composto de dois elementos: as declarações das funções adaptativas e as associações das funções adaptativas ao dispositivo subjacente, denominadas ações adaptativas. Cada regra pode estar associada a duas ações adaptativas: uma antes e outra após a execução da regra. Se não houver nenhuma ação adaptativa a uma determinada regra, ele se comporta como uma regra não adaptativa. Entre os principais campos de aplicação da Tecnologia Adaptativa encontram-se as Linguagens de Programação, o Processamento de Línguas Naturais, a Computação Natural, a Inteligência Artificial e a Engenharia de Software, Reconhecimento de Padrões, Tomada de Decisão Robótica e Aprendizagem de Máquina. Dispositivos adaptativos apresentam forte potencial de aplicação para a construção de um modelo computacional unificado, pois permitem representar fenômenos linguísticos complexos, tais como dependências de contexto, além de serem usados como um formalismo de reconhecimento, o que permite seu uso no pré-processamento de textos para análise morfossintática, verificação de sintaxe, processamento para traduções automáticas, interpretação de texto e correção gramatical.

Observa-se que na literatura analisada não há referência a um modelo computacional unificado que possa representar as diferentes abordagens usadas no processamento da linguagem natural: determinística, estatística e heurística. Os trabalhos analisados não se beneficiam de técnicas diferentes das que seu modelo utiliza, perdendo com isso, a possibilidade de melhorar os resultados obtidos, ou mesmo, de buscar o aprimoramento do processo como um todo. O processamento puramente estatístico pode levar a necessidade de análise de cadeias que não são permitidas pelas regras gramaticais vigentes e que, no entanto, acabam fazendo parte do contexto da análise em virtude de representarem uma possível combinação estatística. Por outro lado, um modelo puramente determinístico requer a identificação exaustiva de todas as regras utilizadas na formalização da língua, o que além de requerer conhecimento muito profundo da língua, nem sempre disponível, torna a abordagem pouco adequada ao caráter evolutivo e dinâmico da língua. Heurísticas, por outro, lado, não oferecem embasamento teórico para serem utilizadas como abordagem principal no processamento da língua natural, apresentando-se muito mais como instrumento de apoio para resolução de ambiguidades.

Nota-se que os trabalhos refletem as limitações de cada modelo, por exemplo, o *parser* Curupira não desambigua a estrutura sintática das sentenças da língua portuguesa, o que talvez pudesse fazer, caso também utilizasse modelos estatísticos. O *parser* PALAVRAS utiliza milhares de regras da gramática constritiva, o que certamente lhe confere precisão, porém também o limita para acompanhar a evolução da língua, que requer sempre novas regras de representação. Por outro lado, no enfoque estatístico usado no etiquetador morfológico de Marques e Lopes [22], a etiqueta escolhida é a que apresenta maior probabilidade em função da sequencia de palavras e sequencia de etiquetas observadas no corpus. Este método desconsidera etiquetas válidas, porém com baixa probabilidade de ocorrência. A abordagem proposta por Neto [24] aparentemente é mais robusta, pois permite incorporar elementos determinísticos e probabilísticos no mesmo modelo, que se adapta em função do contexto observado, adicionando ou eliminando funções adaptativas. Além disso, em um mesmo modelo é possível representar características de linguagens dependentes de contexto, livres de contexto e irregulares, o que o torna, além de consistente, flexível para a construção de parsers e reconhecedores gramaticais.

PROPOSTA DE MODELO UNIFICADO

Em [25], os autores apresentam o Linguístico, um reconhecedor gramatical que usa técnicas adaptativas para reconhecimento da estrutura morfossintática de textos em Português Brasileiro, incorporando técnicas probabilísticas para desambiguação morfológica. A estrutura do Linguístico é apresentada na Fig. 1. A máquina M0 – Sentenciador é responsável pela primeira etapa do processamento, dividindo o texto inicial nas sentenças que o compõem. Em seguida, as sentenças são divididas em *tokens* por meio da máquina M1 –

Tokenizador, que identifica palavras e caracteres de pontuação. No passo seguinte, a máquina M2 – Identificador Morfológico classifica os *tokens* morfologicamente, com o auxilio de textos de apoio, tais como o Bosque [26] e o Tep 2.0 [27] e também com o uso de autômatos que desenvolvem conjugações e montam palavras a partir de regras de formação da Língua Portuguesa.

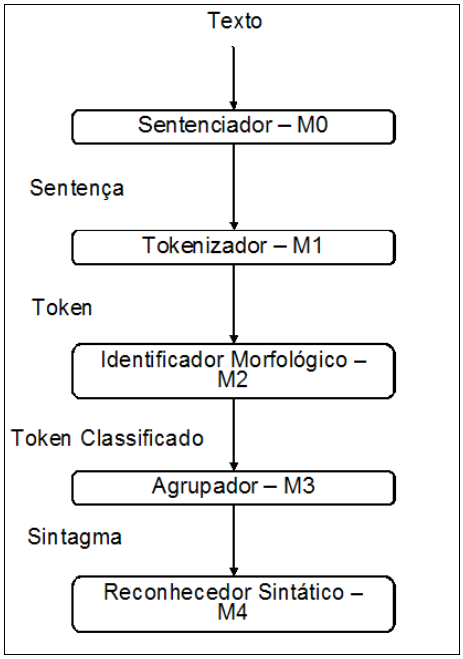


Figura 1. Estrutura do Linguístico [25]

Com os *tokens* classificados, a máquina M3 - Agrupador realiza a montagem de sintagmas, utilizando, para isso, um autômato, uma tabela de agrupamento, e as regras definidas na gramática de Celso Luft [28]. Por fim, os sintagmas são enviados para a máquina M4 – Reconhecedor Sintático, que os analisa e verifica se eles compõem sentenças válidas, segundo as regras sintáticas da gramática de Luft. Embora o Linguístico utilize técnicas determinísticas e probabilísticas, elas são apresentadas de forma pontual, para resolver partes específicas do processamento da língua, não sendo mencionado em nenhum momento, um modelo unificado subjacente que possa ser generalizado para qualquer situação.

A proposta apresentada neste artigo é uma evolução do Linguístico para as tarefas de reconhecimento e desambiguação morfológica e sintática. Como premissa, pressupõe-se que os *tokens* já estejam identificados antes do início da análise morfológica, o que corresponde às duas primeiras etapas do processamento realizado pelo Linguístico. A mudança inicia-se com a máquina M2 – Identificador Morfológico, que seria alterada para ser um gerador de classificações morfológicas e respectivas probabilidades de ocorrências. As máquinas M3 e M4 seriam incorporadas no mesmo autômato adaptativo, consolidando toda a tarefa de reconhecimento e desambiguação em um único modelo computacional, sem necessidade de tabelas de agrupamento ou outro recurso de apoio. A Fig.2 apresenta a nova estrutura do Linguístico modificado de acordo com a proposta.

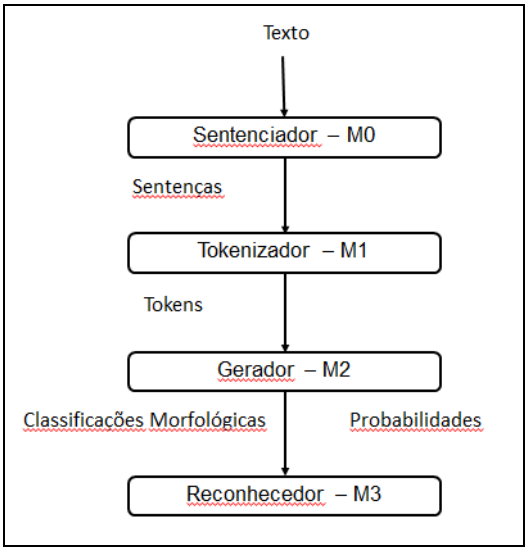


Figura 2. Estrutura Modificada do Linguístico

A máquina M3 – Reconhecedor é composta por um autômato que possui inicialmente um estado, um vetor de pilhas e uma função de transição adaptativa, responsável por criar os passos seguintes. Como o modelo incorpora técnicas probabilísticas, a cada passo a função adaptativa cria dois estados, um para representar a probabilidade de ocorrência do trigrama formado pelas três últimas classificações morfológicas e outro para representar a probabilidade do *token* analisado ser da classificação morfológica indicada pelo último símbolo do trigrama. Sempre que o autômato chega a um estado em que uma classificação morfológica é encontrada, ele atualiza o vetor de pilhas para controlar a montagem de sintagmas e fazer o reconhecimento sintático.

A Fig.3 apresenta a configuração inicial do autômato e vetor de pilhas. A função adaptativa (α(i)) é usada para criar dinamicamente os estados e transições conforme os *tokens* são processados. No estado inicial, não há indicação de movimentação na fila e ela encontra-se vazia. Usando como exemplo a frase “A casa estava aberta”, pode-se acompanhar o mecanismo de reconhecimento e desambiguação.

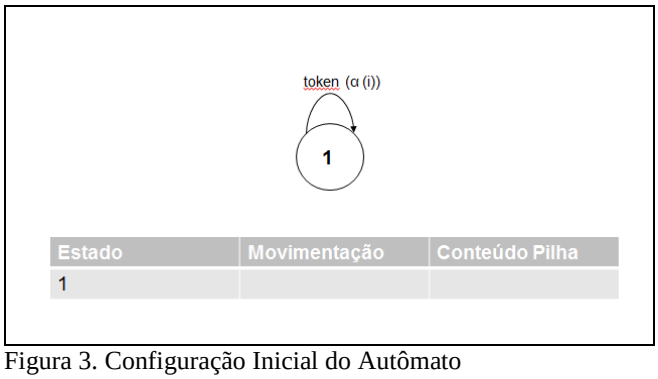
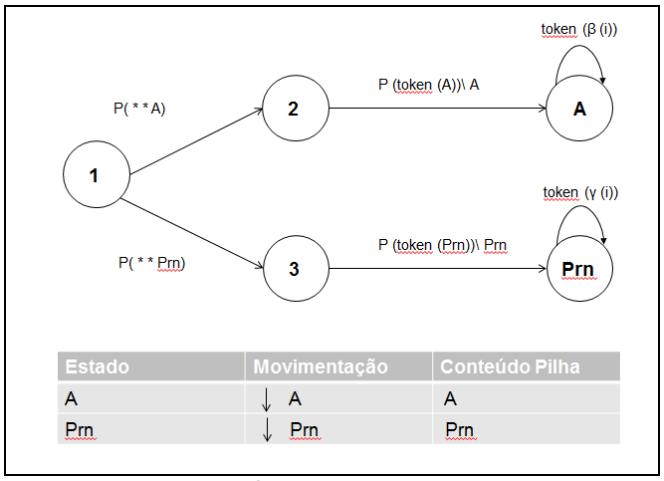


Figura 3. Configuração Inicial do Autômato

A Fig.4 apresenta a configuração do autômato antes processar o *token* “A”. Como existem duas classificações possíveis (artigo e pronome), a função adaptativa (α(i)) cria duas sequências de estados com suas respectivas transições, sendo uma para indicar a probabilidade de “A” ser artigo e outra para indicar a probabilidade de “A” ser pronome. O

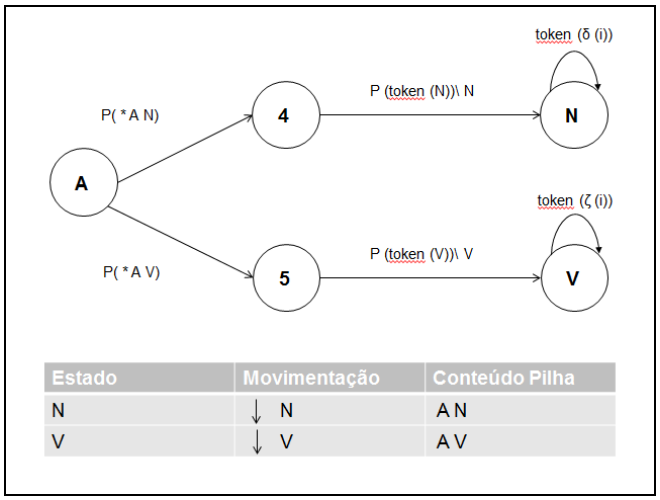
estado 2 representa o trigrama formado pela sequência * * A, na qual o * * representam as classificações morfológicas antes do inicio da frase e A indica artigo. O estado A indica o estado no qual foi identificado um artigo. Entre o estado 1 e o estado 2 indica-se a probabilidade de ocorrência do trigrama * * A e, entre o estado 2 e o estado A, a probabilidade do *token* "A" ser um artigo dado que os dois últimos *tokens* são * *. Ao chegar ao estado A, insere-se um símbolo "A" na pilha (representado por ⧫ A) para indicar que um *token* "A" foi encontrado. O processo ocorre de maneira análoga entre os estados 1, 3 e Prn. Ao final, (α(i)) cria outras duas funções adaptativas (β (i)) e (γ (i)) para prosseguir com a análise.



Estado	Movimentação	Conteúdo Pilha
A	↓ A	A
Prn	↓ Prn	Prn

Figura 4. Primeira transformação adaptativa

A Fig. 5 representa o passo seguinte. Neste caso, o *token* analisado é "casa", que pode ser classificado como substantivo ou verbo. Como o passo anterior criou dois estados, o reconhecimento continua a partir deles. Em A, a função adaptativa (β (i)) cria os estados 4, 5, N e V, sendo que N e V representam substantivos e verbos, respectivamente.

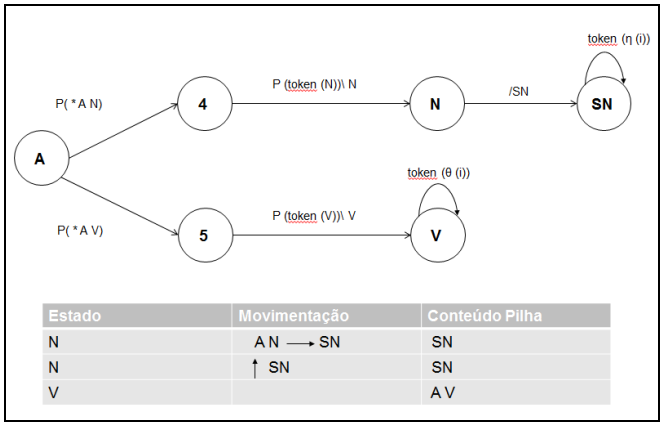


Estado	Movimentação	Conteúdo Pilha
N	↓ N	A N
V	↓ V	A V

Figura 5. Segunda transformação adaptativa

A transição entre o estado A e o estado 4 indica a probabilidade de ocorrência do trigrama * A N e a transição do estado 4 para o estado N indica a probabilidade do *token* "casa" ser um substantivo. Ao final, um símbolo N é armazenado na pilha e a função adaptativa (δ (i)) é criada para prosseguir com o reconhecimento a partir do estado N. O processo é análogo entre os estados A, 5 e V, sendo que, ao final, cria-se uma nova pilha com o símbolo inicial A e armazena-se o símbolo V. A função adaptativa (ζ (i)) é criada no estado V para prosseguir com o reconhecimento a partir deste estado.

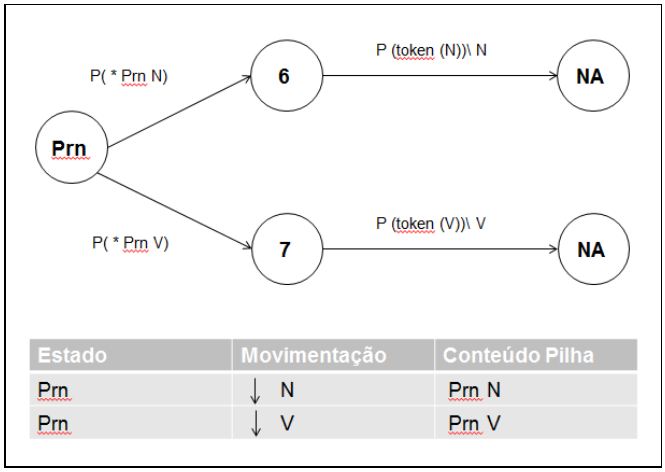
A Fig. 6 mostra como o autômato identifica e monta o primeiro sintagma. A coluna Movimentação indica que uma regra de redução foi encontrada na primeira pilha do vetor e o sintagma SN foi formado a partir dos dois elementos armazenados anteriormente (A N). O conteúdo da pilha é atualizado com a remoção dos símbolos A e N, e o símbolo SN é armazenado em seu lugar. Em seguida, SN é devolvido para a função adaptativa (δ (i)), que por sua vez cria o estado SN e uma nova função (η (i)) para continuar o reconhecimento a partir dele. Nota-se que a pilha A V não sofreu alteração e que continua ativa representando uma ramificação válida até este momento.



Estado	Movimentação	Conteúdo Pilha
N	A N ⟶ SN	SN
N	↑ SN	SN
V		A V

Figura 6. Formação do Sintagma SN

A Fig. 7 representa caminho seguido pelo autômato a partir do estado Prn. A movimentação é análoga àquela apresentada na Fig. 4. A função adaptativa (γ (i)) cria os estados 6, 7 e NA, este último usado para indicar que a sequencia analisada não é válida, o que ocorre quando a probabilidade do trigrama e da classificação do *token* combinados serem zero.



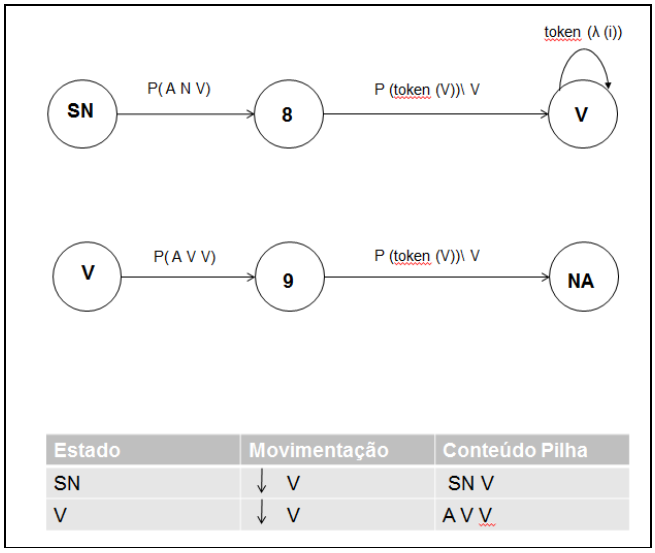
Estado	Movimentação	Conteúdo Pilha
Prn	↓ N	Prn N
Prn	↓ V	Prn V

Figura 7. Reconhecimento a partir de Prn

É o que ocorre com o trigrama * Prn N e o *token* "casa" classificado como substantivo. Embora "casa" possa ser um substantivo, de acordo com as regras gramaticais vigentes, nunca um substantivo vem antecedido de um pronome iniciando uma frase. No caso da segunda ramificação, chega-se a mesma conclusão com relação ao trigrama * Prn V e o *token* "casa" classificado como verbo.

A Fig.8 apresenta o reconhecimento a partir dos estados SN e V e do *token* "estava", classificado como verbo. A função adaptativa (η (i)) cria os estados 8 e V, indicando nas transições a probabilidade de ocorrência do trigrama A N V e do *token* "estava" ser classificado como verbo. O símbolo V é armazenado na pilha na qual se encontra SN e uma nova função (λ (i)) é criada para continuar o reconhecimento a partir de V.

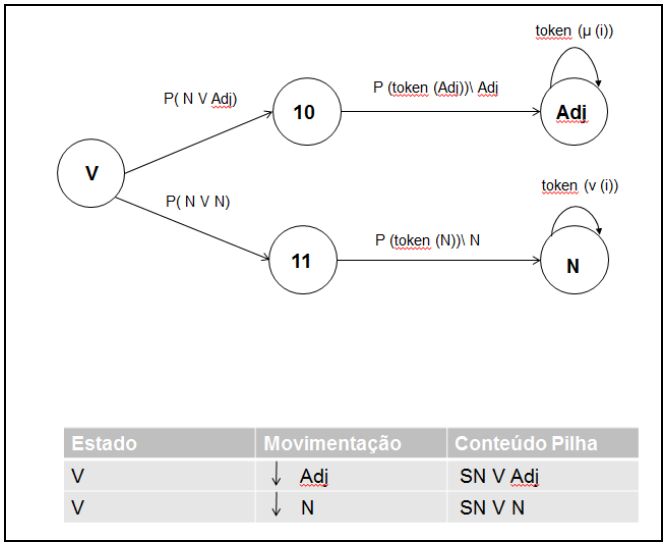
Por outro lado, partindo de V, a função (ζ (i)) cria os estados 9 e NA, indicando que o trigrama A V V não é valido. Ao final, o símbolo Adj é armazenado na fila correspondente e a função (μ (i)) é criada para continuar o reconhecimento a partir de Adj. Na outra ramificação, representa-se o reconhecimento do trigrama N V N e da probabilidade do *token* "estava" ser classificado como substantivo. Ao final, o símbolo N é armazenado na fila correspondente e a função (v (i)) é criada para continuar o reconhecimento a partir de N.



Estado	Movimentação	Conteúdo Pilha
SN	↓ V	SN V
V	↓ V	A V V

Figura 8. Reconhecimento a partir dos estados de SN e V

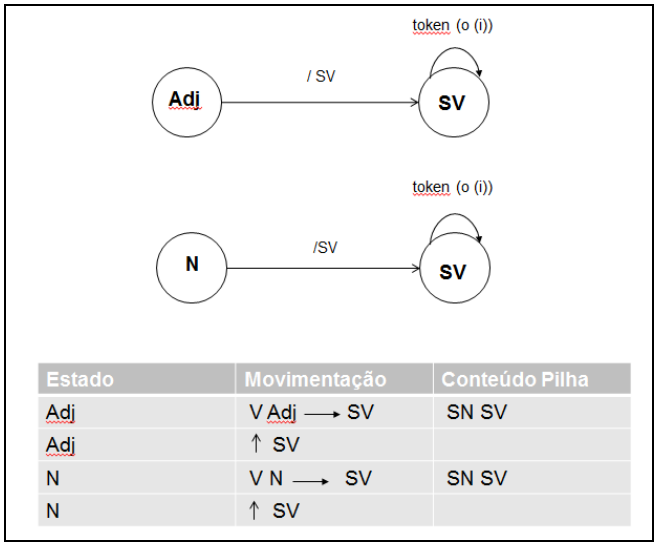
A Fig.9 apresenta o reconhecimento a partir do estado V e do *token* "aberta". Neste caso, o *token* pode ser classificado como adjetivo ou, menos usualmente, como substantivo. A função adaptativa (λ (i)) cria os estados 10, 11, Adj e N. Na ramificação V 10 Adj é representado o reconhecimento do trigrama N V Adj, assim como a probabilidade do *token* "aberta" ser classificado como adjetivo. Ao final, o símbolo Adj é armazenado na fila correspondente e a função (μ (i)) é criada para continuar o reconhecimento a partir de Adj. Na outra ramificação, representa-se o reconhecimento do trigrama N V N e da probabilidade do *token* "estava" ser classificado como substantivo. Ao final, o símbolo N é armazenado na fila correspondente e a função (v (i)) é criada para continuar o reconhecimento a partir de N.



Estado	Movimentação	Conteúdo Pilha
V	↓ Adj	SN V Adj
V	↓ N	SN V N

Figura 9. Reconhecimento a partir de V

A Fig. 10 apresenta o reconhecimento a partir dos estados Adj e N. A coluna Movimentação indica que uma regra de redução foi encontrada, gerando o sintagma SV a partir dos símbolos V e Adj, que são removidos da pilha. Em seguida, o símbolo SV é armazenado na pilha e a função (μ (i)) é notificada, criando o estão SV e a função adaptativa (o (i)). Na outra ramificação, o procedimento é análogo. O estado final SV é o mesmo nos dois casos e a função adaptativa responsável pelo reconhecimento a partir de SV também.

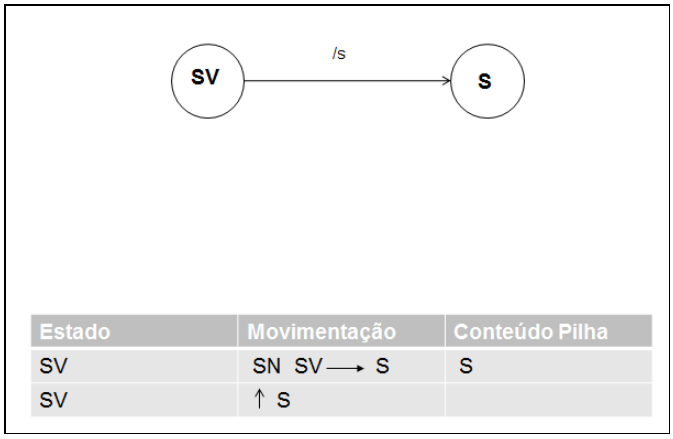


Estado	Movimentação	Conteúdo Pilha
Adj	V Adj ⟶ SV	SN SV
Adj	↑ SV	
N	V N ⟶ SV	SN SV
N	↑ SV	

Figura 10. Reconhecimento a partir de Adj e N

A Fig.11 apresenta a última etapa do reconhecimento. A coluna Movimentação indica que uma regra de redução foi encontrada, gerando o sintagma S a partir dos símbolos SN e SV, que são removidos da pilha. Em seguida, o símbolo S é armazenado na pilha e a função (o (i)) é notificada, criando o estado final S. As movimentações, assim como as probabilidades usadas para cálculo de cada transição podem ser recuperadas através de um transdutor vinculado ao autômato.

Ao final da movimentação, caso o autômato não chegue a um estado de aceitação, a frase é considerada incorreta pela gramática utilizada.



Estado	Movimentação	Conteúdo Pilha
SV	SN SV ⟶ S	S
SV	↑ S	

Figura 11. Etapa final

Considerações Finais

Este artigo apresentou conceitos de processamento da língua natural e uma revisão bibliográfica dos trabalhos realizados sobre o tema. Em seguida, foi proposto um modelo que unifica métodos determinísticos, estatísticos e heurísticos, utilizando autômatos adaptativos como tecnologia subjacente.

Referências

[1] JURAFSKY, D.; MARTIN, J. H.. Speech and Language Processing. 1024 p. Prentice Hall, 2000.

[2] PRICE, D. et al. Natural Java: A Natural Language Interface for Programming in Java. Proceedings of the 5th international conference on intelligent user interfaces. New Orleans, Louisiana, United States. Pages: 207 – 211, 2000. ISBN:1-58113-134-8.

[3] NUNES, M.G.V.; OLIVEIRA, O.N. O processo de desenvolvimento do Revisor Gramatical ReGra. Anais do XXVII SEMISH (XX Congresso Nacional da Sociedade Brasileira de Computação), 2000, Volume 1, p.6

[4] NAVIGLI, R.;VELARDI, P.; GANGEMI, A. Ontology Learning and its Application to Automated Terminology Translation. IEEE Intelligent Systems, Volume 18 Issue 1. Publisher: IEEE Educational Activities Department , 2003.

[5] GRISHMAN, RALPH. Information Extraction: Techniques and Challenges. Lecture Notes In Computer Science; Vol. 1299. International Summer School on Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology, 1997.

[6] RILOFF, ELLEN; LORENZEN JEFFREY. Extraction-based text categorization: Generating Domain Specific role relationships automatically. In Natural Language Information Retrieval, Dordrecht, The Netherlands: Kluwer Academic Publishers, 1999. p. 167-196.

[7] LLORÉNS, JUAN; ASTUDILLO, HERNÁN. Automatic generation of hierarchical taxonomies from free text using linguistic algorithms. Lecture Notes in Computer Science Vol. 2426. Proceedings of the Workshops on Advances in Object-Oriented Information Systems, 2002.

[8] MORAES, M. Alguns aspectos de tratamento sintático de dependência de contexto em linguagem natural empregando tecnologia adaptativa. Tese de Doutorado, Escola Politécnica da Universidade de São Paulo, 2006.

[9] RICH,E.; KNIGHT, K. Inteligência Artificial, 2. Ed. São Paulo: Makron Books, 1993.

[10] ROCHA, R.L.A. Tecnologia Adaptativa Aplicada ao Processamento Computacional de Língua Natural. Workshop de Tecnologias Adaptativas – WTA 2007, 2007.

[11] LADEIRA, M.P. Processamento da Linguagem Natural: Caracterização da Produção Científica dos Pesquisadores Brasileiros. Tese de Doutorado, UFMG, Minas Gerais, 2010.

[12] NETO, J. J.; MENEZES, C. E. D. Um Método para a Construção de Etiquetadores Morfológicos Aplicado a Língua Portuguesa, baseado em Autômatos Adaptativos. In: PROPOR 2000 – V Encontro para o Processamento Computacional da Língua Portuguesa, 2000, Atibaia. Anais do V Encontro para o Processamento Computacional da Língua Portuguesa. São Carlos : ICMS-USP, 2000. p. 53-64.

[13] PADILHA, E. G. ; VICCARI, R. M. Morfologia da Língua Portuguesa com Máquinas de Estados Finitos. In: 5o. Workshop de Processamento da Língua Portuguesa Falada e Escrita (PROPOR-2000), 2000, Atibaia. Anais do 5o. PROPOR - Workshop de Processamento da Língua Portuguesa Falada e Escrita, 2000.

[14] BONFANTE, A. G.; NUNES, M. G. V. Parsing Probabilístico para o Português do Brasil. In: I Workshop de Teses e Dissertações em Inteligência Artificial (I WTDIA), 2002, Porto de Galinhas - Recife. I Workshop de Teses e Dissertações em Inteligência Artificial (I WTDIA), 2002.

[15] BICK, E. The Parsing System PALAVRAS: Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework, 2000. Ph. D. Thesis, Arhus University.

[16] JULIA, R. M. S.; SEABRA, J. R.; SEMEGHINI-SIQUEIRA, I. An Intelligent Parser that Automatically Generates Semantic Rules during Syntactic and Semantic Analysis. In: IEEE International Conference on Systems, Man and Cybernetics, 1995, Vancouver. v. i. p. 806-811.

[17] PIAGET, J. A construção do real na criança. Trad. Álvaro Cabral. Rio de Janeiro: Zahar, 1970. 360p.

[18] CHOMSKY, NOAM. Three models for the description of language. IRE Transactions on Information Theory 2: 113-124, 1956.

[19] SARDINHA, T. B. A Língua Portuguesa no Computador. 295p. Mercado de Letras, 2005.

[20] NUNES et al. Introdução ao Processamento das Línguas Naturais. Notas didáticas do ICMC Nº 38, São Carlos, 88p, 1999. Paris, Hachette, 1992. 158p.

[21] KARLSON, F. Constraint Grammar as a Framework for Parsing Running Text. 13o. International Conference on Computational Linguistics, 1990, Helsinki (Vol.3, p.168-173).

[22] MARQUES, N.M.C.; LOPES, J.G.P. Redes Neuronais e um Léxico na Etiquetagem Morfossintática para o Estudo da Subcategorização Verbal. In: SARDINHA, T. B. A Língua Portuguesa no Computador. 295p. Mercado de Letras, 2005. P. 71-90.

[23] DIAS, G.H.; LOPES, J.G.P. Extração Automática de Unidades Poli lexicais para o Português. In: SARDINHA, T. B. A Língua Portuguesa no Computador. 295p. Mercado de Letras, 2005. P. 155-184.

[24] NETO, J.J. Laboratório de Linguagens e Tecnologias Adaptativas, 2004. Disponível em: < http://lta.poli.usp.br/lta/roteiro-de-estudos >. Acesso: 05/06/2014

[25] CONTIER, A.; PADOVANI, D.; NETO,J.J.. Linguístico: Usando Tecnologia Adaptativa para a Construção de Um Reconhecedor Gramatical. Em: Memórias do VIII Workshop de Tecnologia Adaptativa – WTA 2014. EPUSP, São Paulo, ISBN: 978-85-86686-76-4, pp. 8-20. 06 e 07 de Fevereiro de 2014.

[26] http://www.linguateca.pt/

[27] http://www.nilc.icmc.usp.br/tep2/

[28] LUFT, C.. Moderna Gramática Brasileira. 2ª. Edição Revista e Atualizada. 265p. Editora Globo, 2002.

Djalma Padovani nasceu em São Paulo em 1964. Cursou bacharelado em Física pelo Instituto de Física da Universidade de São Paulo, formou-se em administração de empresas pela Faculdade de Economia e Administração da Universidade de São Paulo, em 1987 e obteve o mestrado em engenharia de software pelo Instituto de Pesquisas Tecnológicas de São Paulo - IPT, em 2008. Trabalhou em diversas empresas nas áreas de desenvolvimento de software e tecnologia de informação e atualmente é responsável pela arquitetura tecnológica da Serasa S/A, empresa do grupo Experían.

João José Neto graduado em Engenharia de Eletricidade (1971), mestrado em Engenharia Elétrica (1975) e doutorado em Engenharia Elétrica (1980), e livre-docência (1993) pela Escola Politécnica da Universidade de São Paulo. Atualmente é professor associado da Escola Politécnica da Universidade de São Paulo, e coordena o LTA - Laboratório de Linguagens e Tecnologia Adaptativa do PCS - Departamento de Engenharia de Computação e Sistemas Digitais da EPUSP. Tem experiência na área de Ciência da Computação, com ênfase nos Fundamentos da Engenharia da Computação, atuando principalmente nos seguintes temas: dispositivos adaptativos, tecnologia adaptativa, autômatos adaptativos, e em suas aplicações à Engenharia de Computação, particularmente em sistemas de tomada de decisão adaptativa, análise e processamento de linguagens naturais, construção de compiladores, robótica, ensino assistido por computador, modelagem de sistemas inteligentes, processos de aprendizagem automática e inferências baseadas em tecnologia adaptativa.

Um arcabouço para extensibilidade em linguagens de programação

P. R. M. Cereda e J. José Neto

***Abstract*—This paper aims at providing a support framework for programming languages through syntactic modifications at compilation time. Such modifications are available from an extension mechanism and allow an increase in expressive power. It is also desirable to use extensibility as a resource to provide special constructs targeting the project and implementation of programs with adaptative features.**

***Palavras-chave:*—Linguagens de programação, extensibilidade, adaptatividade**

I. Introdução

Dá-se o nome de *adaptatividade* à manifestação do fenômeno no qual um dispositivo modifica seu próprio comportamento espontaneamente, em resposta ao seu histórico de operação e aos dados de entrada [1], [2], [3]. Dados de entrada diferentes submetidos a um mesmo dispositivo adaptativo podem resultar, no final do processamento de tais dados e de acordo com diferentes séries de eventos do dispositivo, em configurações de comportamento e de topologia totalmente distintos. A adaptatividade tem como característica maior promover a extensão de formalismos já consolidados, aumentando seu poder de expressão [4].

A adaptatividade possui inúmeras aplicações computacionais nas mais diversas áreas de pesquisa. De acordo com José Neto [3], a generalidade resultante do modelo, a capacidade de aprendizado devida à característica de auto-modificação, bem como o poder de expressão faz dos dispositivos adaptativos uma alternativa muito atraente para expressar fatos complexos e manipular situações difíceis que surgem durante uma busca de soluções computacionais para problemas complexos.

Em particular, a área de pesquisa de linguagens de programação apresenta diversas contribuições extremamente significativas na utilização da tecnologia adaptativa em diversos níveis de abstração [5], [6], [7], [8], [9], [10]. Este artigo apresenta um arcabouço que ofereça extensibilidade em linguagens de programação através da especificação de construtos sintáticos definidos pelo usuário em uma metalinguagem de extensão. Como saída, o arcabouço permite a obtenção de um programa-objeto escrito em linguagem extensível a ser executado pelo sistema operacional ou de um programa-fonte que transforma as extensões sintáticas em comando válidos da linguagem base. A possibilidade de extensão de linguagens de programação pode proporcionar o projeto e implementação de construtos especiais para a escrita de programas com características adaptativas.

Este artigo está organizado da seguinte forma: a Seção II apresenta uma breve revisão bibliográfica dos assuntos relevantes ao arcabouço para extensibilidade em linguagens de programação, proposto neste artigo. A Seção III apresenta os subsídios para o projeto do arcabouço, tais como um conjunto de ações adaptativas para a geração de um analisador sintático, o mecanismo de tratamento de extensões para análise e injeção de novas construções sintáticas, e a organização estrutural proposta. A Seção IV apresenta algumas discussões em relação à proposta deste artigo, incluindo a necessidade de um protocolo de comunicação com o compilador da linguagem base. As considerações finais são apresentadas na Seção V.

Os autores podem ser contatados através dos seguintes endereços de correio eletrônico: paulo.cereda@usp.br e jjneto@usp.br.

II. Conceitos iniciais

Esta seção apresenta uma breve revisão bibliográfica dos assuntos relevantes ao arcabouço para extensibilidade em linguagens de programação, proposto neste artigo. São apresentados os conceitos referentes às linguagens extensíveis, autômatos de pilha estruturados, autômatos adaptativos, introdução à notação de Wirth como metalinguagem para descrição de linguagens de programação e sua interpretação, bem como uma breve discussão sobre a análise sintática e geração a partir da notação de Wirth.

A. Linguagens extensíveis

Linguagens extensíveis são linguagens de programação que permitem a adição ou alteração de construtos sintáticos ou a associação de novas formas sintáticas com semântica [11]. Os novos construtos podem incluir novas notações ou operações, estruturas de controle novas ou modificadas, ou até mesmo elementos provenientes de diferentes paradigmas de programação. Castro Junior [8] menciona os dois componentes de uma linguagem extensível e suas propriedades:

- *Componente léxico:* é descrito por expressões regulares da linguagem base, acrescidas das expressões regulares que compõem a linguagem que define o mecanismo de extensão.
- *Componente sintático:* para estender a sintaxe de uma linguagem base, deve-se fazer com que o compilador possa aceitar uma nova coleção de regras gramaticais, definidas em tempo de programação através de mecanismos declarativos incorporados à linguagem base.

A gramática da linguagem extensível é definida como a junção entre as gramáticas que definem o mecanismo de extensão e a linguagem base [8]. A partir de tais especificações, é possível obter um compilador para a linguagem extensível, de acordo com a Figura 1.

De acordo com a Figura 1, o compilador obtido incorpora as regras gramaticais da linguagem base e do mecanismo de extensão, construindo um reconhecedor apropriado para cada

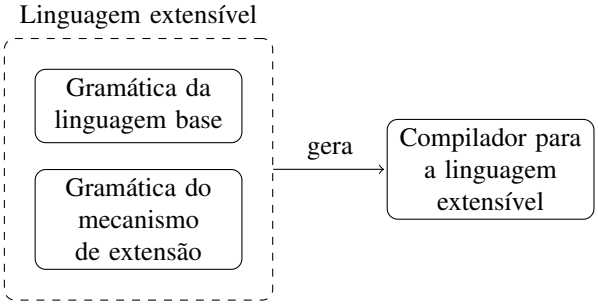


Figura 1. Obtenção do compilador para a linguagem extensível a partir das gramáticas da linguagem base e do mecanismo de extensão [8]

gramática e estabelecendo relacionamentos entre elas, quando aplicáveis.

A Figura 2 apresenta, em linhas gerais, o processo de obtenção do código-objeto para um programa P_0 escrito em uma linguagem extensível L_{E_0}, que consiste em um código-fonte escrito em uma linguagem base L_{base} acrescido de extensões sintáticas E_0 definidas pelo usuário, além da obtenção de um compilador para a nova linguagem L_{E_0}.

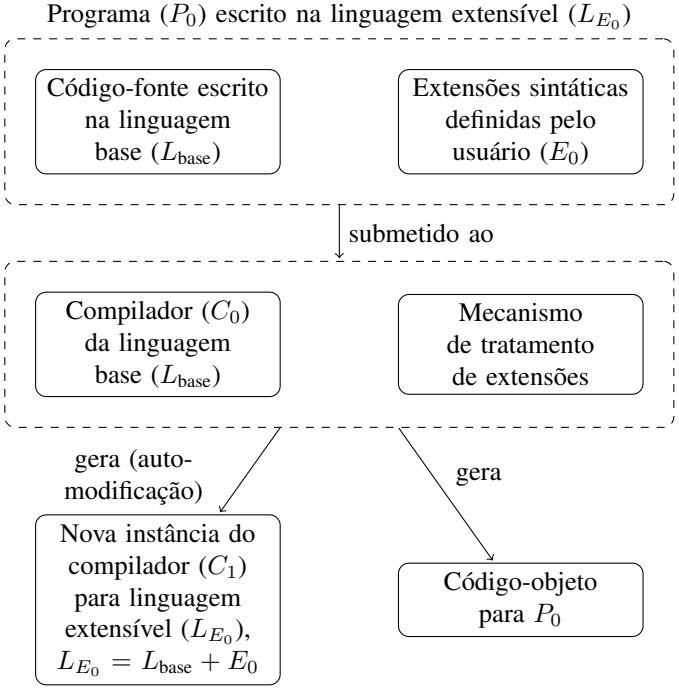


Figura 2. Obtenção do código-objeto para um programa escrito em L_{E_0} e do compilador C_1 que incorpora as novas construções sintáticas definidas pelo usuário (E_0) [8].

De acordo com a Figura 2, é possível notar que a nova linguagem L_{E_0} permite, por sua vez, o acréscimo de novas extensões sintáticas, tornando-se, portanto, a linguagem base de um novo processo de extensibilidade. Além disso, é possível utilizar-se deste conceito para projetar extensões que permitam a auto-modificação de código em tempo de execução, tornando a linguagem obtida propícia para a escrita de programas com características adaptativas.

Para o tratamento da linguagem que suporte adaptatividade por meio de extensão, de acordo com Castro Junior [8], é necessário projetar uma biblioteca de ações adaptativas para gerar as novas extensões em um formato que seja compatível com o ambiente de execução proposto para a linguagem base, e uma biblioteca de ações adaptativas para incorporar as extensões obtidas à nova instância da linguagem base.

Em [8], Castro Junior apresenta e discute os aspectos relacionados com o projeto de linguagens de programação adaptativa e introduz uma linguagem imperativa simplificada de programação denominada *MINLA* e sua linguagem de extensão, denominada *EXTLA*, definindo um conjunto de primitivas para o mapeamento de suas instruções na forma de construtos adaptativos. Adicionalmente, um ambiente de execução fundamentado no autômato adaptativo [1] é introduzido, definindo a estrutura básica da máquina de execução de tais programas, e implementando as seguintes rotinas [8]:

1) Carga do programa para a memória da máquina, cujo segmento de código pode ser representado por um estado do autômato que aponta para o estado correspondente à primeira instrução do respectivo código.
2) Busca da próxima instrução a ser executada no segmento de código, seguido de sua decodificação e execução.

Em [8], o autor ainda destaca que a implementação de um ambiente de execução adequado é muito vantajosa do ponto de vista experimental, uma vez que os comandos para a auto-modificação de código serão tratados sem a necessidade de grandes esforços para a adequação do ambiente de execução.

B. Autômato de pilha estruturado

O autômato de pilha estruturado [1] é um tipo de autômato de pilha formado por um conjunto de autômatos finitos mutuamente recursivos, também chamados de *sub-máquinas*. Diferentemente do autômato de pilha tradicional, a pilha tem a finalidade exclusiva de armazenar estados de retorno a cada chamada de uma sub-máquina. As chamadas e retornos consistem em transferir o controle entre uma sub-máquina e outra; essa transição consiste em utilizar o símbolo de entrada apenas para a tomada de decisão do autômato em relação a qual transição executar, sendo o tal símbolo consumido na próxima transição [1], [2].

Formalmente, um autômato de pilha estruturado M pode ser definido como $M = (Q, A, \Sigma, \Gamma, P, Z_0, q_0, F)$, onde Q é o conjunto finito não-vazio de estados, A é um conjunto de sub-máquinas, definidas a seguir, Σ é o alfabeto do autômato, correspondendo ao conjunto finito não-vazio dos símbolos de entrada, Γ é o conjunto finito não-vazio dos símbolos de pilha, armazenados na memória auxiliar do autômato, P é a relação de transição de estados, $q_0 \in Q$ é o estado inicial (da primeira sub-máquina), Z_0 é o símbolo marcador de pilha vazia, e $F \subseteq Q$ é o conjunto dos estados de aceitação do autômato (da primeira sub-máquina) [1].

Uma sub-máquina $a_i \in A$ é definida como um autômato finito tradicional, da forma $a_i = (Q_i, \Sigma_i, P_i, q_{i,0}, F_i)$, onde $Q_i \subseteq Q$ é o conjunto de estados de a_i, $\Sigma_i \subseteq \Sigma$ é o conjunto de símbolos de entrada de a_i, $q_{i,0}$ é o estado de entrada da sub-máquina a_i, $P_i \subseteq P$ é a relação de transição de estados de a_i, e $F_i \subseteq F$ é o conjunto de estados de retorno da sub-máquina.

A relação de transição P é definida como $P \subseteq \Gamma \times Q \times \Sigma \times \Gamma \times Q$, na forma $(\gamma g, e, s\alpha) \rightarrow (\gamma g', e', \alpha)$, onde e, e' são os estados corrente e de destino, respectivamente, s é o símbolo consumido, α é o restante da cadeia de entrada, g é o topo da pilha, g' é o novo topo da pilha, e γ é o restante da pilha. Uma configuração é um elemento de $Q \times \Sigma^* \times \Gamma^*$, e uma relação entre configurações sucessivas $\vdash$ é definida como:

- *Consumo de símbolo:* $(q, \sigma w, uv) \vdash (p, w, xv)$, com $p, q \in Q$, $u, x \in \Gamma$, $v \in \Gamma^*$, $\sigma \in \Sigma \cup \{\epsilon\}$, $w \in \Sigma^*$, se σ foi consumido pelo autômato, $x = u$, e $(\gamma, q, \sigma\alpha) \rightarrow (\gamma, p, \alpha) \in P$.
- *Chamada de sub-máquina:* $(q, w, uv) \vdash (r, w, xv)$, com $q, r \in Q$, $u \in \Gamma$, $v, x \in \Gamma^*$, $w \in \Sigma^*$, $x = pu$, com chamada da sub-máquina R, estado inicial r, retorno em p, e $(\gamma, q, \alpha) \rightarrow (\gamma p, r, \alpha) \in P$.
- *Retorno de sub-máquina:* $(q, w, uv) \vdash (p, w, v)$, com $p, q \in Q$, $u, x \in \Gamma$, $v \in \Gamma^*$, $w \in \Sigma^*$, $u = p$, com retorno de sub-máquina para p, e $(\gamma g, q, \alpha) \rightarrow (\gamma, g, \alpha) \in P$.

A linguagem reconhecida por um autômato de pilha estruturado M é dada por $L(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (f, \epsilon, Z_0), f \in F\}$.

C. Autômato adaptativo

O autômato adaptativo [1] é um extensão do formalismo do autômato de pilha estruturado (Subseção II-B) que permite o reconhecimento de linguagens recursivamente enumeráveis. O termo *adaptativo*, neste contexto, pode ser definido como a capacidade de um dispositivo em alterar seu comportamento de forma espontânea. Um autômato adaptativo, portanto, tem como característica a possibilidade de alterar sua própria topologia durante o processo de reconhecimento de uma dada cadeia, em resposta a algum estímulo [2]. A possibilidade de alteração do autômato ocorre através da utilização de ações adaptativas, que lidam com situações esperadas, mas ainda não consideradas, detectadas na cadeia submetida para reconhecimento pelo autômato [12].

Ao executar uma transição que contém uma ação adaptativa associada, o autômato sofre mudanças, obtendo-se uma nova configuração. Para a aceitação de uma determinada cadeia, o autômato percorrerá um caminho em um espaço de autômatos; haverá, portanto, um autômato E_0 que iniciará o reconhecimento de uma cadeia w, autômatos intermediários E_i que serão criados ao longo do reconhecimento, e um autômato final E_n que corresponde ao final do reconhecimento de w. Em outras palavras, seja a cadeia $w = \alpha_0\alpha_1 \ldots \alpha_n$; o autômato M descreverá um caminho de autômatos $\langle E_0, \alpha_0\rangle \rightarrow \langle E_1, \alpha_1\rangle \rightarrow \ldots \rightarrow \langle E_n, \alpha_n\rangle$, onde E_i representa um autômato correspondente à aceitação da sub-cadeia a_i.

Formalmente, um autômato adaptativo M pode ser definido como $M = (Q, A, \Sigma, \Gamma, P, Z_0, q_0, F, E, \Phi)$, onde Q é o conjunto finito não-vazio de estados, A é um conjunto de sub-máquinas, definidas a seguir, Σ é o alfabeto do autômato, correspondendo ao conjunto finito não vazio dos símbolos de entrada, Γ é o conjunto finito não-vazio de símbolos de pilha, armazenados na memória auxiliar do autômato, P é a relação de transição de estados, $q_0 \in Q$ é o estado inicial (da primeira sub-máquina), Z_0 é o símbolo marcador de pilha vazia, $F \subseteq Q$ é o conjunto dos estados de aceitação do autômato (da primeira sub-máquina), E representa o modelo de autômato utilizado (inicialmente, o reconhecimento inicia-se no autômato E_0 do caminho de autômatos), e Φ é o conjunto de funções adaptativas, aplicáveis às transições [1], [2].

Uma sub-máquina $a_i \in A$ do autômato adaptativo M é definida como $a_i = (Q_i, \Sigma_i, P_i, q_{i,0}, F_i, \Phi_i)$, onde $Q_i \subseteq Q$ é o conjunto de estados de a_i, $\Sigma_i \subseteq \Sigma$ é o conjunto de símbolos de entrada de a_i, $q_{i,0}$ é o estado de entrada da sub-máquina a_i, $P_i \subseteq P$ é a relação de transição de estados de a_i, $F_i \subseteq F$ é o conjunto de estados de retorno da sub-máquina a_i, e $\Phi_i \subseteq \Phi$ é o conjunto de funções adaptativas aplicáveis.

A relação de transição P é definida como $P \subseteq (\Gamma \times Q \times \Sigma \times (\Phi \cup \{\epsilon\})) \times (\Gamma \times Q \times \Sigma \times (\Phi \times \{\epsilon\}))$, na forma $(\gamma g, e, s\alpha), \mathcal{B} \rightarrow (\gamma g', e', s'\alpha), \mathcal{A}$, onde e, e' são os estados corrente e de destino, repectivamente, s é o símbolo consumido, α é o restante da cadeia de entrada, g é o topo da pilha, g' é o novo topo da pilha, γ é o restante da pilha, $\mathcal{B}$ é uma função adaptativa anterior, e $\mathcal{A}$ é uma função adaptativa posterior. Uma configuração para um autômato adaptativo é um elemento de $E \times Q \times \Sigma^* \times \Gamma^*$, e uma relação entre configurações sucessivas $\vdash$ é definida como:

- *Consumo de símbolo:* $(E_i, q, \sigma w, uv) \vdash (E_i, p, \sigma' w, xv)$, com $p, q \in Q$, $u, x \in \Gamma$, $v \in \Gamma^*$, $\sigma, \sigma' \in \Sigma \cup \{\epsilon\}$, $w \in \Sigma^*$, se σ foi consumido pelo autômato, $x = u$, e $(\gamma, q, \sigma\alpha) \rightarrow (\gamma, p, \sigma'\alpha) \in P$.
- *Chamada de sub-máquina:* $(E_i, q, w, uv) \vdash (E_i, r, w, xv)$, com $q, r \in Q$, $u \in \Gamma$, $v, x \in \Gamma^*$, $w \in \Sigma^*$, $x = pu$, com chamada da sub-máquina R, estado inicial r, retorno em p, e $(\gamma, q, \alpha) \rightarrow (\gamma p, r, \alpha) \in P$.
- *Retorno de sub-máquina:* $(E_i, q, w, uv) \vdash (E_i, p, w, v)$, com $p, q \in Q$, $u, x \in \Gamma$, $v \in \Gamma^*$, $w \in \Sigma^*$, $u = p$, com retorno de sub-máquina para p, e $(\gamma g, q, \alpha) \rightarrow (\gamma, g, \alpha) \in P$.
- *Funções adaptativas:* $(E_i, q, \sigma w, uv) \vdash^{\mathcal{B},\mathcal{A}} (E_{i+2}, p, \sigma' w, u'v)$, indicando $(E_i, q, \sigma w, uv) \rightarrow^{\mathcal{B}} (E_{i+1}, q, \sigma w, uv) \vdash (E_{i+1}, p, \sigma' w, u'v) \rightarrow^{\mathcal{A}} (E_{i+2}, p, \sigma' w, u'v)$, com $p, q \in Q$, $u, u' \in \Gamma$, $v \in \Gamma^*$, $w \in \Sigma^*$, se $(\gamma u, q, \sigma\alpha), \mathcal{B} \rightarrow (\gamma u, p, \sigma'\alpha), \mathcal{A} \in P$.

Uma chamada de função adaptativa $\mathcal{F}_i$ assume a forma $\mathcal{F}_i(\tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,m})$, onde cada $\tau_{i,j}$ representa um argumento passado à função. A declaração de uma função adaptativa $\mathcal{F}_i$ com m parâmetros consiste de um cabeçalho da forma $\mathcal{F}_i(\Theta_{i,1}, \Theta_{i,2}, \ldots, \Theta_{i,m})$ e um corpo contendo nomes (uma lista de identificadores que representam objetos no escopo da função, incluindo variáveis e geradores) e ações (lista de ações adaptativas elementares precedida por uma ação adaptativa inicial e seguida por outra final). A inicialização de uma função adaptativa preenche os geradores e os parâmetros passados e, a seguir, as ações são efetivamente aplicadas.

São definidos três tipos de ações adaptativas elementares que realizam testes no conjunto de regras ou modificam regras existentes (relação de transição de estados P), a saber:

1) *ação adaptativa elementar de inspeção:* as ação não modifica o conjunto de regras, mas permite a inspeção deste para a verificação de regras que obedeçam um certo padrão.

2) *ação adaptativa elementar de remoção:* a ação remove regras que correspondem a um determinado padrão do conjunto corrente de regras.
3) *ação adaptativa elementar de inclusão:* a ação insere uma regra que corresponde a um determinado padrão no conjunto corrente de regras.

A linguagem reconhecida por um autômato adaptativo M é dada por $L(M) = \{w \in \Sigma^* \mid (E_0, q_0, w, Z_0) \vdash^* (E_n, f, \epsilon, Z_0), f \in F\}$. A simplicidade e facilidade de entendimento do autômato adaptativo em relação à máquina de Turing (modelo clássico de reconhecimento de linguagens recursivamente enumeráveis) fazem com que sua utilização seja muito ampla [3].

D. Notação de Wirth

Em [13], Niklaus Wirth introduz uma metalinguagem para descrição de linguagens de programação, na tentativa de oferecer uma notação simplificada em relação às iniciativas existentes; tal metalinguagem ficou conhecida como *notação de Wirth* e apresenta as seguintes propriedades:

1) A notação apresenta uma distinção clara entre metassímbolos, símbolos terminais e símbolos não-terminais. Os metassímbolos existentes são: `=`, `.`, `(`, `)`, `[`, `]`, `{`, `}`, `|` e `"`. Um símbolo não-terminal é denotado por um identificador, isto é, uma letra seguida zero ou mais letras e dígitos (como uma definição de variável em uma linguagem de programação), enquanto o símbolo terminal é expresso por uma cadeia de caracteres entre aspas duplas.
2) Não há restrição quanto à utilização de metassímbolos como símbolos da linguagem sendo descrita. Em outras palavras, o metassímbolo `|`, por exemplo, difere do símbolo terminal `"|"`.
3) A notação evita o uso intenso de recursão para expressar repetições simples; dessa forma, existe um construto para iteração explícita.
4) Não há a necessidade da utilização do uso de um símbolo explícito que represente a cadeia vazia (como $\langle empty \rangle$ na notação BNF ou ϵ), pois já existem construtos que tratam dessa situação.
5) A notação é baseada no conjunto de caracteres ASCII.

A metalinguagem proposta por Wirth é apresentada na Listagem 1, com as palavras `identifier` denotando um símbolo não-terminal, `literal` denotando um símbolo terminal e `character` denotando um símbolo válido.

```
syntax = { production } .
production = identifier "=" expression "." .
expression = term { "|" term } .
term = factor { factor } .
factor = identifier | literal | "(" expression ")" |
         "[" expression "]" | "{" expression "}" .
literal = """" character { character } """" .
```

Listagem 1. Notação de Wirth, em sua forma original [13], expressa utilizando a própria notação que a define.

De acordo com [13], a repetição é denotada por chaves, isto é, `{ a }` representa ϵ `| a | aa | aaa |` ... (fecho de Kleene). Elementos opcionais são expressos através de colchetes, isto é, `[ a ]` representa `a` $\mid \epsilon$. Parênteses são utilizados para representar agrupamentos, isto é, `( a | b) c` representa `ac | bc`. Os símbolos terminais são expressos entre aspas; caso as aspas apareçam como símbolos literais, estas são duplicadas[1].

E. Analisador sintático

A análise sintática (também chamada de *parsing*) é o segundo estágio de um processo de três partes (Figura 3) que o compilador emprega para a análise do programa-fonte e geração do programa-objeto [14]. O analisador sintático trabalha com o programa-fonte transformado pelo analisador léxico; ao invés de lidar com um fluxo de caracteres, ele trabalha com um fluxo de tokens que contêm palavras associadas a uma categoria sintática (semelhante à sua classe gramatical). A partir destas informações, o analisador sintático deriva uma estrutura sintática para o programa, adequando as palavras em um modelo gramatical da linguagem especificada; se o analisador sintático reconhece o fluxo de tokens como válido, o programa-fonte é válido sintaticamente, isto é, foi escrito de acordo com as especificações da gramática da linguagem; caso contrário, o analisador acusa o erro e fornece outras informações de diagnóstico ao usuário. Quando o programa-fonte é válido, o analisador constrói um modelo concreto do programa para ser utilizado no terceiro estágio, isto é, na geração de código propriamente dita [14].

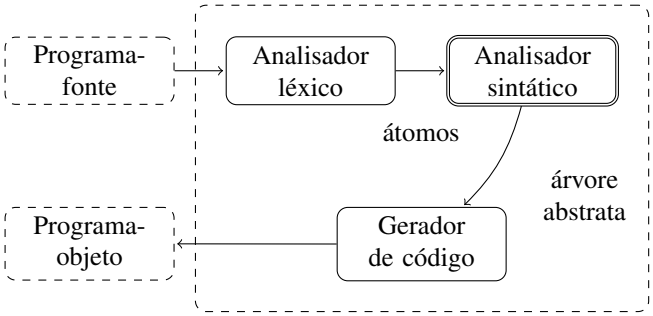


Figura 3. Organização de um compilador, tendo o analisador sintático como programa principal.

Uma linguagem de programação é, em geral, dependente de contexto. Entretanto, é comum tratá-la como livre de contexto, utilizando dispositivos reconhecedores de tais linguagens (como autômatos de pilha) e tratando a parte dependente de contexto como uma extensão semântica (o nome comum de tal verificação é *semântica estática*). A semântica estática é uma atividade realizada pelo analisador semântico para verificar a consistência do programa em termos das dependências existentes (por exemplo, se a expressão em um comando condicional realmente resolve para um valor lógico, se uma variável inteira recebe um valor inválido, como ponto flutuante, entre outros). O nome "semântica estática" é

[1] Algumas representações alternativas expressam aspas duplas literais como `"\""` ao invés de `""""`. O artigo original de Wirth mantém a opção pela duplicação das aspas.

inadequado porque a dependência de contexto é um fenômeno puramente sintático e não semântico; o analisador semântico acaba assumindo esse tratamento, além de suas funções usuais que compreendem a interpretação da linguagem [15], [2].

É possível gerar um analisador sintático a partir da notação de Wirth apresentada na Subseção II-D, obtendo-se um autômato de pilha estruturado que reconheça sentenças válidas da gramática especificada [15]. É imperativo que a gramática seja preparada previamente, tal que:

1) a gramática esteja escrita na notação de Wirth; caso esta esteja em outra notação, convertê-la, e
2) seja resolvido o sistema de equações representado pela gramática, interpretando os não-terminais como variáveis e os terminais como constantes.

Em relação ao item 2, a resolução do sistema de equações representado pela gramática consiste, em linhas gerais, em:

a) agrupar as várias opções do mesmo não-terminal,
b) fatorar as auto-recursões à esquerda e à direita,
c) transformar as auto-recursões à esquerda e à direita em iterações,
d) determinar os não-terminais essenciais, pesquisando-os a partir da raiz da gramática, de acordo com as dependências impostas pela gramática, e
e) eliminar, por substituição, os demais não-terminais.

Após o tratamento da gramática, é possível obter o analisador sintático utilizando o autômato de pilha estruturado apresentado na Figura 4; as ações semânticas associadas às transições estão definidas na Tabela I.

Tabela I
AÇÕES SEMÂNTICAS ASSOCIADAS ÀS TRANSIÇÕES DO AUTÔMATO DE PILHA ESTRUTURADO DA FIGURA 4.

Transição	Nome da ação semântica	Algoritmo
$(1, \texttt{'não-terminal'}) \rightarrow 2$	criar nova sub-máquina	1
$(2, \texttt{'='}) \rightarrow 3$	novo escopo	2
$(4, \texttt{'.'}) \rightarrow 5$	fechar escopo	3
$(5, \texttt{'não-terminal'}) \rightarrow 2$	criar nova sub-máquina	1
$(6, \texttt{'não-terminal'}) \rightarrow 7$	nova transição	4
$(6, \texttt{'terminal'}) \rightarrow 7$	nova transição	4
$(6, \texttt{'}\epsilon\texttt{'}) \rightarrow 7$	nova transição	4
$(7, \texttt{'não-terminal'}) \rightarrow 7$	nova transição	4
$(7, \texttt{'terminal'}) \rightarrow 7$	nova transição	4
$(7, \texttt{'}\epsilon\texttt{'}) \rightarrow 7$	nova transição	4
$(6, \texttt{'('}) \rightarrow 8$	novo escopo	2
$(6, \texttt{'['}) \rightarrow 10$	abre colchetes	5
$(6, \texttt{'\{'}) \rightarrow 12$	abre chaves	6
$(7, \texttt{'('}) \rightarrow 8$	novo escopo	2
$(7, \texttt{'['}) \rightarrow 10$	abre colchetes	5
$(7, \texttt{'\{'}) \rightarrow 12$	abre chaves	6
$(9, \texttt{')'}) \rightarrow 7$	fecha escopo	3
$(11, \texttt{']'}) \rightarrow 7$	fecha escopo	3
$(13, \texttt{'\}'}) \rightarrow 7$	fecha escopo	3
$(7, \texttt{'\|'}) \rightarrow 6$	adiciona opção	7

Algoritmo 1 Ação semântica de criação de nova sub-máquina

ação semântica *criar nova sub-máquina*

```
pilha.esvaziar()
estado corrente := 0
contador := 1
```

fim da ação semântica

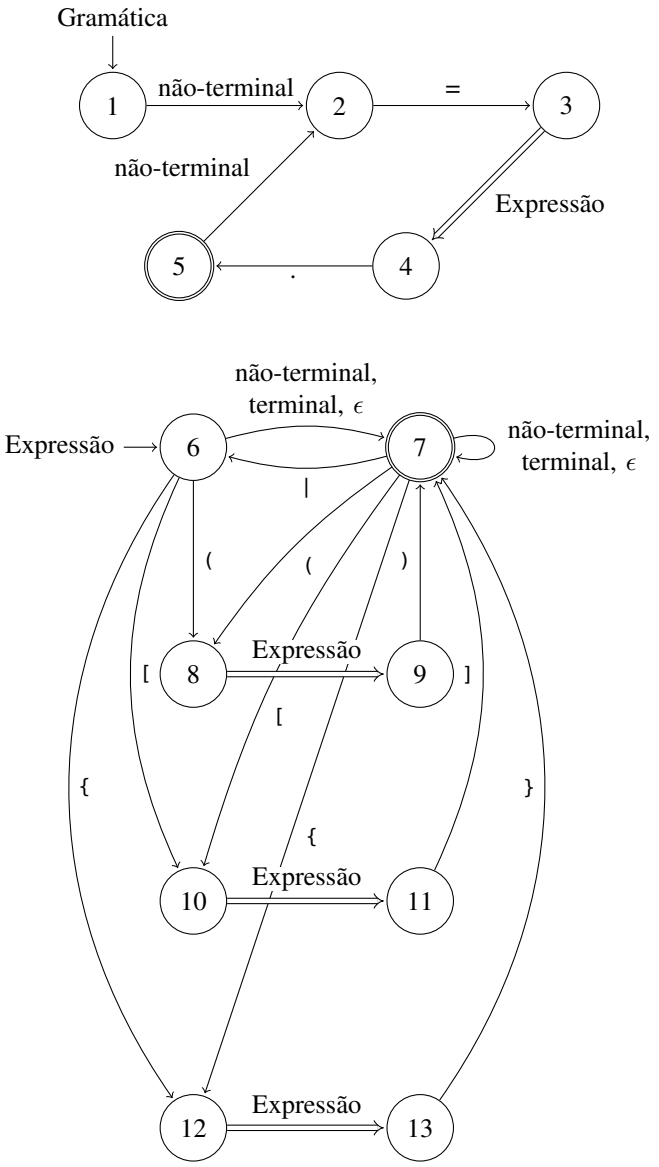


Figura 4. Autômato de pilha estruturado para geração do analisador sintático a partir de uma gramática escrita em notação de Wirth.

Algoritmo 2 Ação semântica de novo escopo

ação semântica *novo escopo*

```
pilha.empilhar(par(estado corrente, contador))
contador := contador + 1
```

fim da ação semântica

Algoritmo 3 Ação semântica de fechamento de escopo

ação semântica *fechar escopo*

```
transições += transição(estado corrente, ϵ,
pilha.topo().segundoElemento())
estado corrente := pilha.topo().segundoElemento()
pilha.desempilhar()
```

fim da ação semântica

Algoritmo 4 Ação semântica de nova transição

ação semântica *nova transição*

```
transições += transição(estado corrente,
token.valor(), contador)
estado corrente := contador
contador := contador + 1
```

fim da ação semântica

Algoritmo 5 Ação semântica de abertura de colchetes

ação semântica *abre colchetes*

```
transições += transição(estado corrente, ϵ,
contador)
pilha.empilhar(par(estado corrente, contador))
contador := contador + 1
```

fim da ação semântica

Algoritmo 6 Ação semântica de abertura de chaves

ação semântica *abre chaves*

```
transições += transição(estado corrente, ϵ,
contador)
estado corrente := contador
pilha.empilhar(par(contador, contador))
contador := contador + 1
```

fim da ação semântica

Algoritmo 7 Ação semântica de adição de opção

ação semântica *adiciona opção*

```
transições += transição(estado corrente, ϵ,
pilha.topo().segundoElemento())
estado corrente := pilha.topo().primeiroElemento()
```

fim da ação semântica

Como exemplo, considere a gramática de uma brincadeira de roda muito comum em países de língua inglesa chamada *duck, duck, goose*[2], apresentada na Listagem 2; a gramática do texto usado na brincadeira de roda é muito simples: a palavra `duck` deve ocorrer pelo menos uma vez, e pode repetir-se n vezes até que a palavra `goose` ocorra (com $n \geq 0$).

```
program = "duck" { "duck" } "goose" .
```

Listagem 2. Gramática do texto usado na brincadeira de roda *duck, duck, goose* escrita em notação de Wirth.

A partir da gramática da Listagem 2 e utilizando-se o autômato de pilha estruturado da Figura 4, obtém-se o autômato de pilha estruturado apresentado na Figura 5, que faz a análise sintática e reconhece sentenças pertencentes à brincadeira de roda *duck, duck, goose*.

É importante notar que o autômato gerado através do autômato de pilha estruturado da Figura 4 é não-determinístico, devendo este ser convertido para sua forma determinística. A minimização também pode ocorrer, mas deve ser utilizada com cautela para que estados semelhantes mas com semânticas diferentes não sejam aglutinados em um só, comprometendo o estágio de geração de código.

[2]O Brasil possui uma versão similar conhecida como *jogo do lenço*.

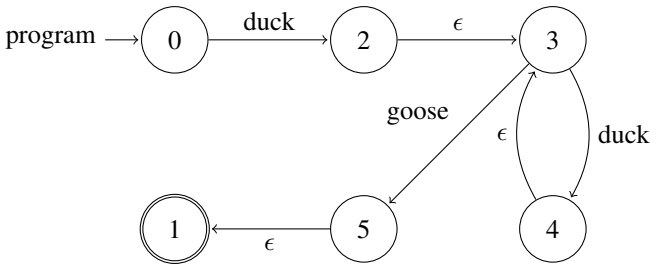


Figura 5. Autômato de pilha estruturado que reconhece sentenças pertencentes à brincadeira de roda *duck, duck, goose*.

III. Arcabouço para extensibilidade

Esta seção apresenta subsídios para o projeto de um arcabouço que ofereça extensibilidade em linguagens de programação, permitindo a adição ou a alteração de construtos sintáticos destas (Subseção II-A), tais como um conjunto de ações adaptativas para a geração automática de um analisador sintático a partir de uma gramática especificada utilizando-se a notação de Wirth (substituindo as ações semânticas discutidas na Subseção II-E), o mecanismo de tratamento de extensões para analisar e injetar novas construções sintáticas, e a organização do arcabouço proposto.

A. Ações adaptativas para geração do analisador sintático

O autômato adaptativo da Figura 6 foi inspirado no analisador sintático apresentado na Subseção II-E. Tal qual o autômato de pilha estruturado da Figura 4, o autômato proposto também constrói um autômato que reconheça sentenças válidas de uma linguagem especificada na notação de Wirth; entretanto, o autômato adaptativo altera sua topologia de forma que, ao terminar a leitura da cadeia de entrada (indicada pelo símbolo de terminação de arquivo $\langle \text{EOF} \rangle$), este poderá atuar como analisador sintático da linguagem recém-analisada, reconhecendo cadeias válidas de forma imediata. As ações adaptativas associadas às transições estão definidas na Tabela II.

Algoritmo 8 Função adaptativa $\mathcal{A}(p)$

função adaptativa $\mathcal{A}(p)$

geradores: $g_1^\star, g_2^\star$

$-(1, \texttt{'não-terminal'}) \rightarrow 2$

$+(\gamma, 1, \alpha) \rightarrow (\langle i, 6 \rangle \gamma, q_{p,g_1^\star}, \alpha)$

$+(\langle \text{Gramática}, 6 \rangle \gamma, g_2^\star, \alpha) \rightarrow (\gamma, q_{\text{Gramática},6}, \alpha)$

$+(g_1^\star, \tau_{\text{corrente}}) \rightarrow g_1^\star$

$+(g_2^\star, \tau_{\text{contador}}) \rightarrow g_2^\star$

fim da função adaptativa

É importante observar que, na especificação das ações adaptativas, a pilha sintática auxiliar originalmente utilizada nas ações semânticas do autômato de pilha estruturado da Figura 4 foi simulada através do encadeamento de estados auxiliares e

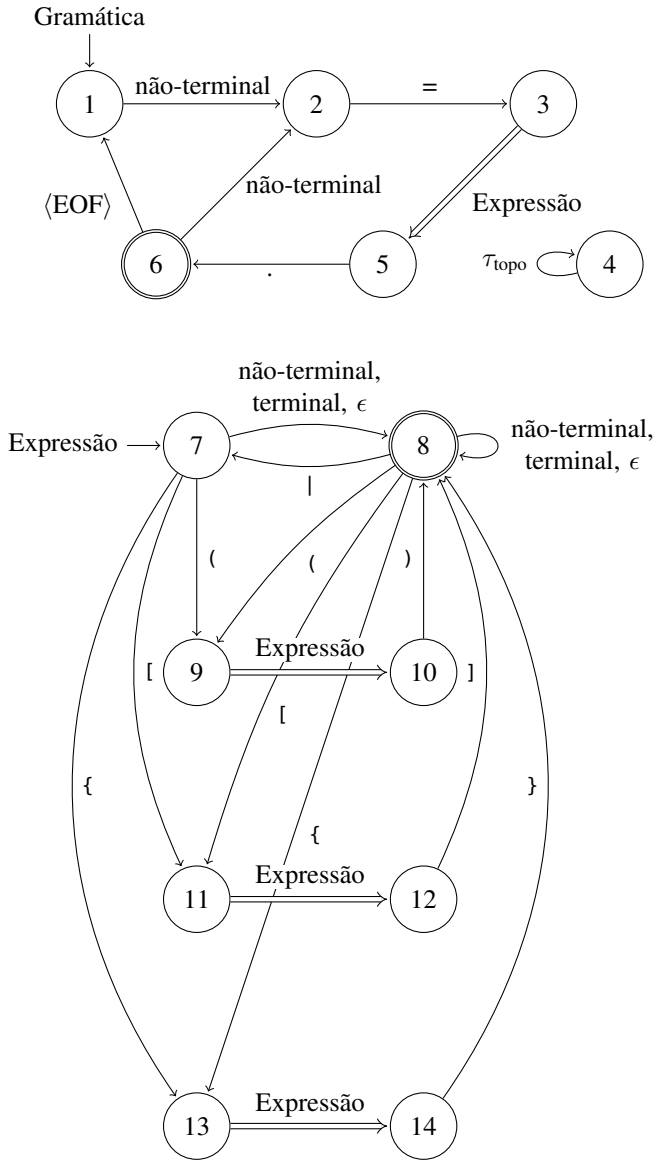


Figura 6. Autômato adaptativo generativo para realizar a análise léxica de sentenças escritas em um gramática definida através da notação de Wirth.

Algoritmo 9 Função adaptativa $\mathcal{B}()$

função adaptativa $\mathcal{B}()$
geradores: $g_1^\star, g_2^\star$
variáveis: $?a, ?b, ?c, ?v_1, ?v_2, ?v_3$

$?(?a, \tau_{\text{corrente}}) \rightarrow ?v_1$
$?(?b, \tau_{\text{contador}}) \rightarrow ?v_2$
$?(?c, \tau_{\text{topo}}) \rightarrow ?v_3$
$-(?c, \tau_{\text{topo}}) \rightarrow ?v_3$
$+(?c, \tau_{\text{elemento}}) \rightarrow g_1^\star$
$+(g_1^\star, \tau_{\text{topo}}) \rightarrow g_1^\star$
$+(g_1^\star, \tau_{\text{primeiro}}) \rightarrow ?a$
$+(g_1^\star, \tau_{\text{segundo}}) \rightarrow ?b$
$-(?b, \tau_{\text{contador}}) \rightarrow ?v_2$
$+(g_2^\star, \tau_{\text{contador}}) \rightarrow g_2^\star$

fim da função adaptativa

Tabela II
AÇÕES ADAPTATIVAS ASSOCIADAS ÀS TRANSIÇÕES DO AUTÔMATO ADAPTATIVO GENERATIVO DA FIGURA 6.

Transição	Ação adaptativa	Algoritmo
$(1, \texttt{'não-terminal'}) \rightarrow 2$	$\cdot\mathcal{A}(\texttt{'não-terminal'})$	8
$(2, \texttt{'='}) \rightarrow 3$	$\cdot\mathcal{B}()$	9
$(5, \texttt{'.'}) \rightarrow 6$	$\cdot\mathcal{C}()$	10
$(6, \texttt{'não-terminal'}) \rightarrow 2$	$\cdot\mathcal{D}(\texttt{'não-terminal'})$	11
$(6, \langle\text{EOF}\rangle) \rightarrow 1$	$\cdot\mathcal{E}()$	12
$(7, \texttt{'não-terminal'}) \rightarrow 8$	$\cdot\mathcal{F}(\texttt{'não-terminal'})$	13
$(7, \texttt{'terminal'}) \rightarrow 8$	$\cdot\mathcal{G}(\texttt{'terminal'})$	14
$(7, \texttt{'}\epsilon\texttt{'}) \rightarrow 8$	$\cdot\mathcal{G}(\texttt{'}\epsilon\texttt{'})$	14
$(8, \texttt{'não-terminal'}) \rightarrow 8$	$\cdot\mathcal{F}(\texttt{'não-terminal'})$	13
$(8, \texttt{'terminal'}) \rightarrow 8$	$\cdot\mathcal{G}(\texttt{'terminal'})$	14
$(8, \texttt{'}\epsilon\texttt{'}) \rightarrow 8$	$\cdot\mathcal{G}(\texttt{'}\epsilon\texttt{'})$	14
$(7, \texttt{'('}) \rightarrow 9$	$\cdot\mathcal{B}()$	9
$(7, \texttt{'['}) \rightarrow 11$	$\cdot\mathcal{H}()$	15
$(7, \texttt{'\{'}) \rightarrow 13$	$\cdot\mathcal{I}()$	16
$(8, \texttt{'('}) \rightarrow 9$	$\cdot\mathcal{B}()$	9
$(8, \texttt{'['}) \rightarrow 11$	$\cdot\mathcal{H}()$	15
$(8, \texttt{'\{'}) \rightarrow 13$	$\cdot\mathcal{I}()$	16
$(10, \texttt{')'}) \rightarrow 8$	$\cdot\mathcal{C}()$	10
$(12, \texttt{']'}) \rightarrow 8$	$\cdot\mathcal{C}()$	10
$(14, \texttt{'\}'}) \rightarrow 8$	$\cdot\mathcal{C}()$	10
$(8, \texttt{'\|'}) \rightarrow 7$	$\cdot\mathcal{J}()$	17

Algoritmo 10 Função adaptativa $\mathcal{C}()$

função adaptativa $\mathcal{C}()$
variáveis: $?a, ?b, ?c, ?d, ?e, ?v_1, ?v_2$

$?(?a, \tau_{\text{corrente}}) \rightarrow ?v_1$
$?(?b, \tau_{\text{topo}}) \rightarrow ?v_2$
$?(?b, \tau_{\text{primeiro}}) \rightarrow ?c$
$?(?b, \tau_{\text{segundo}}) \rightarrow ?d$
$+(?a, \epsilon) \rightarrow ?d$
$?(?e, \tau_{\text{elemento}}) \rightarrow ?b$
$-(?b, \tau_{\text{topo}}) \rightarrow ?v_2$
$-(?b, \tau_{\text{primeiro}}) \rightarrow ?c$
$-(?b, \tau_{\text{segundo}}) \rightarrow ?d$
$-(?e, \tau_{\text{elemento}}) \rightarrow ?b$
$+(?e, \tau_{\text{topo}}) \rightarrow ?e$
$-(?a, \tau_{\text{corrente}}) \rightarrow ?v_1$
$+(?d, \tau_{\text{corrente}}) \rightarrow ?d$

fim da função adaptativa

Algoritmo 11 Função adaptativa $\mathcal{D}(p)$

função adaptativa $\mathcal{D}(p)$
geradores: $g_1^\star, g_2^\star$
variáveis: $?a, ?b, ?c, ?d, ?v_1, ?v_2, ?v_3, ?v_4$

$q_{p,0} \leftarrow g_1^\star$
$?(?a, \tau_{\text{topo}}) \rightarrow ?v_1$
$-(?a, \tau_{\text{topo}}) \rightarrow ?v_1$
$?(?b, \tau_{\text{elemento}}) \rightarrow ?v_2$
$-(?b, \tau_{\text{elemento}}) \rightarrow ?v_2$
$?(?c, \tau_{\text{primeiro}}) \rightarrow ?v_3$
$-(?c, \tau_{\text{primeiro}}) \rightarrow ?v_3$
$?(?d, \tau_{\text{segundo}}) \rightarrow ?v_4$
$-(?d, \tau_{\text{segundo}}) \rightarrow ?v_4$
$+(4, \tau_{\text{topo}}) \rightarrow 4$
$+(g_1^\star, \tau_{\text{corrente}}) \rightarrow g_1^\star$
$+(g_2^\star, \tau_{\text{contador}}) \rightarrow g_2^\star$

$+(\langle n, m\rangle\gamma, g_2^\star, \alpha) \rightarrow (\gamma, q_{n,m}, \alpha)$

fim da função adaptativa

Algoritmo 12 Função adaptativa $\mathcal{E}()$

função adaptativa $\mathcal{E}()$
variáveis: $?a, ?b, ?c, ?d, ?v_1, ?v_2, ?v_3, ?v_4$
$-(6, \langle\text{EOF}\rangle) \rightarrow 1$
$-(6, \texttt{'não-terminal'}) \rightarrow 2$
fim da função adaptativa

Algoritmo 13 Função adaptativa $\mathcal{F}(p)$

função adaptativa $\mathcal{F}(p)$
geradores: $g^\star$
variáveis: $?a, ?b, ?v_1, ?v_2$

$?(?a, \tau_{\text{corrente}}) \rightarrow ?v_1$ $\quad$ $-(?b, \tau_{\text{contador}}) \rightarrow ?v_2$
$?(?b, \tau_{\text{contador}}) \rightarrow ?v_2$ $\quad$ $+(?b, \tau_{\text{corrente}}) \rightarrow ?b$
$-(?a, \tau_{\text{corrente}}) \rightarrow ?v_1$ $\quad$ $+(g^\star, \tau_{\text{contador}}) \rightarrow g^\star$

$+(\gamma, ?a, \alpha) \rightarrow (\langle i, ?b\rangle\gamma, q_{p,a}, \alpha)$
fim da função adaptativa

Algoritmo 14 Função adaptativa $\mathcal{G}(p)$

função adaptativa $\mathcal{F}(p)$
geradores: $g^\star$
variáveis: $?a, ?b, ?v_1, ?v_2$

$?(?a, \tau_{\text{corrente}}) \rightarrow ?v_1$ $\quad$ $-(?b, \tau_{\text{contador}}) \rightarrow ?v_2$
$?(?b, \tau_{\text{contador}}) \rightarrow ?v_2$ $\quad$ $+(?b, \tau_{\text{corrente}}) \rightarrow ?b$
$+(?a, p) \rightarrow ?b$ $\quad$ $+(g^\star, \tau_{\text{contador}}) \rightarrow g^\star$
$-(?a, \tau_{\text{corrente}}) \rightarrow ?v_1$

fim da função adaptativa

Algoritmo 15 Função adaptativa $\mathcal{H}()$

função adaptativa $\mathcal{G}()$
geradores: $g_1^\star, g_2^\star$
variáveis: $?a, ?b, ?c, ?v_1, ?v_2, ?v_3$

$?(?a, \tau_{\text{corrente}}) \rightarrow ?v_1$ $\quad$ $+(g_1^\star, \tau_{\text{topo}}) \rightarrow g_1^\star$
$?(?b, \tau_{\text{contador}}) \rightarrow ?v_2$ $\quad$ $+(g_1^\star, \tau_{\text{primeiro}}) \rightarrow ?a$
$+(?a, \epsilon) \rightarrow ?b$ $\quad$ $+(g_1^\star, \tau_{\text{segundo}}) \rightarrow ?b$
$?(?c, \tau_{\text{topo}}) \rightarrow ?v_3$ $\quad$ $-(?b, \tau_{\text{contador}}) \rightarrow ?v_2$
$-(?c, \tau_{\text{topo}}) \rightarrow ?v_3$ $\quad$ $+(g_2^\star, \tau_{\text{contador}}) \rightarrow g_2^\star$
$+(?c, \tau_{\text{elemento}}) \rightarrow g_1^\star$

fim da função adaptativa

Algoritmo 16 Função adaptativa $\mathcal{I}()$

função adaptativa $\mathcal{H}()$
geradores: $g_1^\star, g_2^\star$
variáveis: $?a, ?b, ?c, ?v_1, ?v_2, ?v_3$

$?(?a, \tau_{\text{corrente}}) \rightarrow ?v_1$ $\quad$ $+(g_1^\star, \tau_{\text{primeiro}}) \rightarrow ?b$
$?(?b, \tau_{\text{contador}}) \rightarrow ?v_2$ $\quad$ $+(g_1^\star, \tau_{\text{segundo}}) \rightarrow ?b$
$+(?a, \epsilon) \rightarrow ?b$ $\quad$ $-(?b, \tau_{\text{contador}}) \rightarrow ?v_2$
$?(?c, \tau_{\text{topo}}) \rightarrow ?v_3$ $\quad$ $+(g_2^\star, \tau_{\text{contador}}) \rightarrow g_2^\star$
$-(?c, \tau_{\text{topo}}) \rightarrow ?v_3$ $\quad$ $-(?a, \tau_{\text{corrente}}) \rightarrow ?v_1$
$+(?c, \tau_{\text{elemento}}) \rightarrow g_1^\star$ $\quad$ $+(?b, \tau_{\text{corrente}}) \rightarrow ?b$
$+(g_1^\star, \tau_{\text{topo}}) \rightarrow g_1^\star$

fim da função adaptativa

Algoritmo 17 Função adaptativa $\mathcal{J}()$

função adaptativa $\mathcal{I}()$
variáveis: $?a, ?b, ?c, ?d, ?v_1, ?v_2$

$?(?a, \tau_{\text{corrente}}) \rightarrow ?v_1$ $\quad$ $+(?a, \epsilon) \rightarrow ?d$
$?(?b, \tau_{\text{topo}}) \rightarrow ?v_2$ $\quad$ $-(?a, \tau_{\text{corrente}}) \rightarrow ?v_1$
$?(?b, \tau_{\text{primeiro}}) \rightarrow ?c$ $\quad$ $+(?c, \tau_{\text{corrente}}) \rightarrow ?c$
$?(?b, \tau_{\text{segundo}}) \rightarrow ?d$

fim da função adaptativa

da utilização de sinalizadores especiais em cada execução das funções adaptativas, para fins didáticos. A utilização de uma pilha sintática auxiliar torna a implementação mais eficiente, reduzindo o número de estados criados e o número de ações elementares de inspeção no autômato.

B. Mecanismo de tratamento de extensões

O mecanismo de tratamento de extensões é um componente do arcabouço de extensibilidade que atua na manipulação de mudanças estruturais definidas pelo usuário, exigindo que o compilador modifique-se em tempo de compilação do programa-fonte, em resposta a tais mudanças requeridas sobre a linguagem original [8]. Uma metalinguagem é definida para o mecanismo de extensão, que realizará o mapeamento das novas construções sintáticas ao seu significado.

O mecanismo de tratamento de extensões analisa o processo de reconhecimento sintático da linguagem base, mapeando as novas construções sintáticas definidas na metalinguagem de extensão e disponíveis em seu componente sintático para o analisador original; assim, em tempo de compilação do programa-fonte, é possível alterar a topologia do analisador sintático original para acomodar as extensões definidas. A Figura 7 apresenta uma possível organização do mecanismo de tratamento de extensões, realizando a junção dos analisadores sintáticos das duas linguagens.

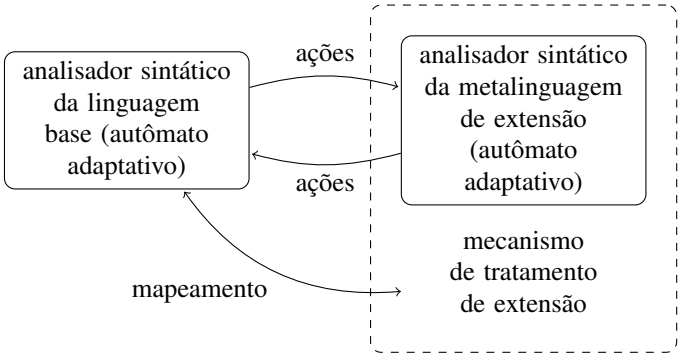


Figura 7. Uma possível organização do mecanismo de tratamento de extensões, realizando a junção dos analisadores sintáticos das duas linguagens.

De acordo com a Figura 7, o mecanismo de tratamento de extensões exerce um papel determinante para prover integração entre a linguagem básica e as extensões definidas pelo usuário; a adequação da sintaxe estendida em tempo de execução é

realizada através de ações adaptativas, que acomodam as novas construções no analisador sintático original.

Como exemplo, considere a gramática apresentada na Listagem 2 acrescida de ações semânticas, tal que `duck` $\rightarrow a_1$ e `goose` $\rightarrow a_2$; a entrada `duck duck goose`, portanto, gera a sequência de ações semânticas $\langle a_1, a_1, a_2 \rangle$. Suponha que a linguagem base seja estendida, através de uma definição utilizando a metalinguagem implementada pelo mecanismo de tratamento de extensões, para suportar a palavra `drake` ocorrendo zero ou mais vezes no início da sentença. Para fins ilustrativos, um exemplo de metalinguagem hipotética (fortemente inspirada em [8]) para suportar tal extensão é apresentada na Listagem 3.

```
DEFINE drake_command AS
   [ "drake":1 { "drake":1 } ] .
INJECT drake_command AT
   program.
MEANING
   1:
      a1
END
```

Listagem 3. Exemplo de metalinguagem hipotética, fortemente inspirada em [8], para suportar a nova palavra `drake` ocorrendo zero ou mais vezes no início da sentença.

A Figura 8 ilustra os dois analisadores sintáticos resultantes da geração através de ações adaptativas (Subseção III-A). É importante observar que algumas transições foram associadas a ações semânticas (note ainda que, neste exemplo particular, a extensão proposta faz uso tão somente dos construtos semânticos da linguagem base – admitindo que esta possui apenas as ações a_1 e a_2 disponibilizadas).

O mecanismo de tratamento de extensões, através de ações adaptativas, pode detectar a ocorrência da nova extensão em um programa-fonte (escrito na linguagem extensível) e realizar alterações no reconhecimento sintático para acomodar a nova situação, conforme ilustra a Figura 9.

De acordo com a Figura 9, o reconhecedor sintático já permite o tratamento de extensões definidas pelo usuário. Como exemplo, a sentença `drake drake duck duck goose` é corretamente aceita (linguagem extensível), resultando na sequência de ações semânticas $\langle a_1, a_1, a_1, a_1, a_2 \rangle$.

C. Organização do arcabouço

O arcabouço para extensibilidade em linguagens de programação atua como uma camada sobre a linguagem de programação a ser estendida, permitindo que mudanças estruturais definidas pelo usuário sejam incorporadas à linguagem base, de forma modular e transparente. Uma possível organização do arcabouço proposto é apresentada na Figura 10.

De acordo com a Figura 10, uma possível organização do arcabouço proposto inclui os seguintes componentes principais:

– *Mecanismo de tratamento de extensões:* introduzido na Subseção III-B, este componente atua na manipulação

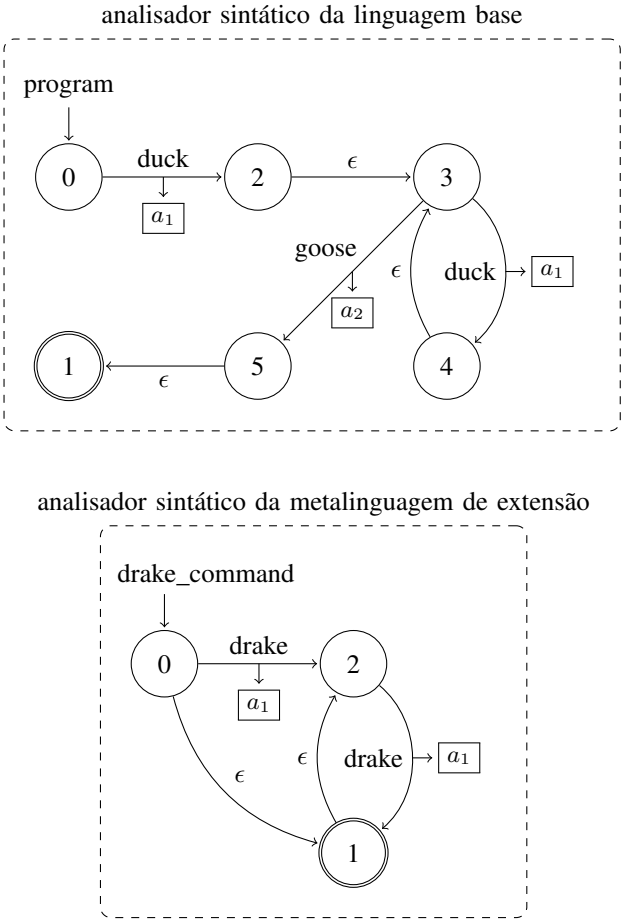


Figura 8. Analisadores sintáticos da linguagem base definida através da gramática da Listagem 2 e da especificação da extensão `drake` a partir da metalinguagem apresentada na Listagem 3.

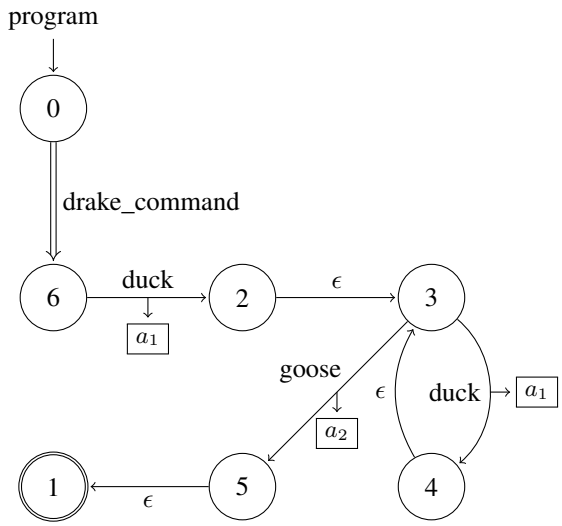


Figura 9. Analisador sintático alterado pelo mecanismo de tratamento de extensões, através de ações adaptativas, para acomodar a nova situação de ocorrência da palavra `drake` na sentença.

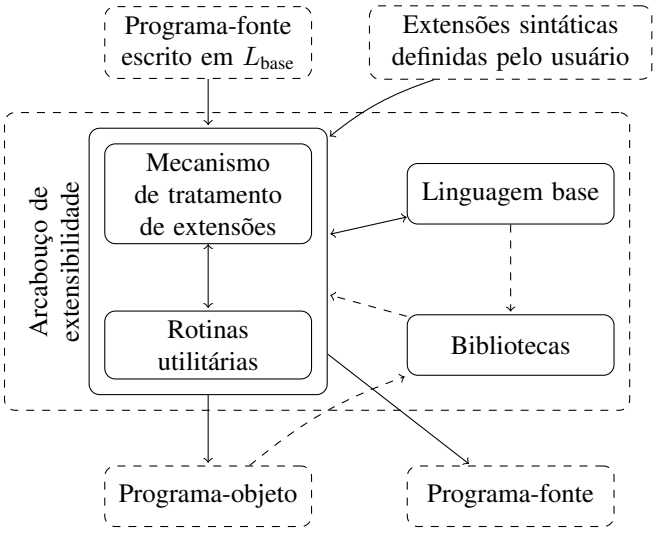


Figura 10. Uma possível organização do arcabouço proposto.

de mudanças estruturais definidas pelo usuário sobre a linguagem base. O mecanismo de tratamento de extensões analisa a especificação das extensões sintáticas escritas em uma metalinguagem de extensão e monitora a análise sintática do programa-fonte escrito na linguagem base, procurando por padrões que coincidam com as regras definidas pelo usuário.

– *Rotinas utilitárias:* o arcabouço apresenta um conjunto de rotinas utilitárias que auxiliam o mecanismo de extensão durante o processo de reconhecimento e análise da linguagem base e das extensões definidas. Em geral, tais rotinas incluem a manipulação de estruturas de dados e eventuais chamadas a bibliotecas do sistema operacional.
– *Acoplamento da linguagem base:* a característica modular do arcabouço proposto permite o acoplamento da linguagem base no contexto do processo de extensão. Deseja-se que a linguagem base possua mecanismos de comunicação para realizar a troca de informações entre o mecanismo de tratamento de extensão, as rotinas utilitárias e o próprio compilador da linguagem base.
– *Bibliotecas:* as extensões sintáticas definidas podem utilizar chamadas de bibliotecas do sistema operacional ou em nível do usuário. Além disso, é possível utilizar um conjunto de bibliotecas pré-definidas para estender a linguagem base, como por exemplo, a adição de construtos adaptativos definidos em biblioteca de propósito geral.

O arcabouço proposto pode resultar, como saída de seu processamento, em um programa-objeto a ser executado pelo sistema operacional (opcionalmente, utilizando chamadas de bibliotecas) ou em um programa-fonte que transforma as extensões sintáticas em comandos válidos da linguagem base, permitindo sua posterior geração de código por um compilador tradicional da linguagem base.

IV. Discussões

O arcabouço para extensibilidade proposto neste artigo almeja proporcionar uma estrutura de suporte para linguagens de programação, permitindo a extensão das mesmas através de modificações sintáticas em tempo de compilação. Tais modificações podem contribuir para o aumento do poder expressivo da linguagem em questão (por exemplo, em [8], o autor introduz o comando `while` em uma linguagem contendo apenas `if` e `goto`, o que aumenta consideravelmente a expressividade da linguagem).

É importante notar que o acoplamento da linguagem base depende de um protocolo de comunicação com o arcabouço; em outras palavras, é imperativo que a linguagem base ofereça mecanismos para troca de informações durante a análise sintática e geração de código. Caso o compilador ou interpretador da linguagem base não ofereça tais subsídios, é necessário intervir no código-fonte da ferramenta e adicionar manualmente os pontos de comunicação; como último recurso, caso o código-fonte não esteja disponível, a extensibilidade pode ocorrer na forma de chamadas a bibliotecas específicas que implementem as funcionalidades definidas através das extensões.

A possibilidade de extensão de linguagens de programação, através do arcabouço proposto neste artigo, pode proporcionar o projeto e implementação de construtos especiais para a escrita de programas com características adaptativas. A adição de uma camada adaptativa em linguagens de programação tradicionais por meio de extensibilidade pode conferir um novo modelo para desenvolvimento de software auto-modificável sem a necessidade imediata de mudanças significativas na forma de escrita de código de um conhecimento profundo, por parte do usuário, dos conceitos de adaptatividade.

V. Considerações finais

Este artigo apresentou um arcabouço que ofereça extensibilidade em linguagens de programação, permitindo a adição ou a alteração de construtos sintáticos definidos pelo usuário através de uma metalinguagem de extensão; o mecanismo de tratamento de extensão monitora a análise da linguagem base, realizando alterações de acordo com as extensões sintáticas definidas. Como saída, o arcabouço permite a obtenção de um programa-objeto a ser executado pelo sistema operacional ou de um programa-fonte que transforma as extensões sintáticas em comandos válidos da linguagem base.

A extensibilidade é um poderoso recurso para a adição e alteração dos construtos sintáticos em linguagens de programação, aumentando seu poder expressivo; espera-se que, através do arcabouço proposto neste artigo, o processo de extensão de uma linguagem de programação seja simplificado de tal forma que o usuário possa beneficiar-se das novas notações, operações ou estruturas de controle para a escrita de programas mais consistentes. Além disso, construtos especiais podem viabilizar o acréscimo de uma camada adaptativa em linguagens de programação tradicionais, permitindo a escrita de programas com características adaptativas.

Referências

[1] J. José Neto, "Contribuições à metodologia de construção de compiladores," Tese de livre docência, Escola Politécnica da Universidade de São Paulo, São Paulo, 1993.
[2] ——, "Adaptive automata for context-sensitive languages," *SIGPLAN Notices*, vol. 29, no. 9, pp. 115–124, set 1994.

[3] J. José Neto, “Um levantamento da evolução da adaptatividade e da tecnologia adaptativa,” *IEEE Latin America Transactions*, vol. 5, pp. 496–505, 2007.
[4] H. Pistori, “Tecnologia em engenharia de computação: estado da arte e aplicações,” Tese de Doutorado, Escola Politécnica, Universidade de São Paulo, 2003.
[5] R. L. A. Rocha and J. José Neto, “Uma proposta de linguagem de programação funcional com características adaptativas,” in *IX Congreso Argentino de Ciencias de la Computación*, 2003.
[6] A. V. Freitas and J. José Neto, “Adaptive languages and a new programming style,” in *WSEAS International Conference on Applied Computer Science*, Tenerife, Canary Islands, Spain, 2006.
[7] A. V. Freitas, “Considerações sobre o desenvolvimento de linguagens adaptativas de programação,” Tese de Doutorado, Escola Politécnica, Universidade de São Paulo, 2008.
[8] A. A. Castro Junior, “Aspectos de projeto e implementação de linguagens para codificação de programas adaptativos,” Tese de Doutorado, Escola Politécnica, Universidade de São Paulo, 2009.
[9] E. J. Pelegrini, “Códigos adaptativos e linguagem de programação adaptativa: conceitos e tecnologias,” Dissertação de Mestrado, Escola Politécnica, Universidade de São Paulo, 2009.
[10] S. R. B. Silva, “Software adaptativo: método de projeto, representação gráfica e implementação de linguagem de programação,” Dissertação de Mestrado, Escola Politécnica, Universidade de São Paulo, 2011.
[11] D. Zingaro, “Modern extensible languages,” Dissertação de Mestrado, McMaster University, Hamilton, Ontario, Canadá, 2007.
[12] J. José Neto, “Adaptive rule-driven devices: general formulation and case study,” in *International Conference on Implementation and Application of Automata*, 2001.
[13] N. Wirth, “What can we do about the unnecessary diversity of notation for syntactic definitions?” *Commun. ACM*, vol. 10, no. 11, pp. 822–823, nov 1977.
[14] K. Cooper and L. Torczon, *Construindo Compiladores*, 2nd ed. Elsevier, 2014.
[15] J. José Neto, *Introdução à compilação*, ser. Engenharia de Computação. LTC – Livros Técnicos e Científicos, 1987.

Paulo Roberto Massa Cereda é graduado em Ciência da Computação pelo Centro Universitário Central Paulista (2005) e mestre em Ciência da Computação pela Universidade Federal de São Carlos (2008). Atualmente, é doutorando do Programa de Pós-Graduação em Engenharia de Computação do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo, atuando como aluno pesquisador no Laboratório de Linguagens e Técnicas Adaptativas do PCS. Tem experiência na área de Ciência da Computação, com ênfase em Teoria da Computação, atuando principalmente nos seguintes temas: tecnologia adaptativa, autômatos adaptativos, dispositivos adaptativos, linguagens de programação e construção de compiladores.

João José Neto é graduado em Engenharia de Eletricidade (1971), mestre em Engenharia Elétrica (1975), doutor em Engenharia Elétrica (1980) e livre-docente (1993) pela Escola Politécnica da Universidade de S ão Paulo. Atualmente, é professor associado da Escola Politécnica da Universidade de São Paulo e coordena o LTA – Laboratório de Linguagens e Técnicas Adaptativas do PCS – Departamento de Engenharia de Computação e Sistemas Digitais da EPUSP. Tem experiência na área de Ciência da Computação, com ênfase nos Fundamentos da Engenharia da Computação, atuando principalmente nos seguintes temas: dispositivos adaptativos, tecnologia adaptativa, autômatos adaptativos, e em suas aplicações à Engenharia de Computação, particularmente em sistemas de tomada de decisão adaptativa, análise e processamento de linguagens naturais, construção de compiladores, robótica, ensino assistido por computador, modelagem de sistemas inteligentes, processos de aprendizagem automática e inferências baseadas em tecnologia adaptativa.

Em Direção ao Desenvolvimento de Programas Adaptativos Utilizando MDE

[1]S. R. M. Canovas, [2]C. E. Cugnasca

***Abstract* — *Model Driven Engineering* (MDE) é uma abordagem para desenvolvimento de software baseada na transformação de modelos de alto nível, tais como diagramas UML, até a geração automática do código-fonte em uma linguagem alvo. *Set Based Meta Modeling* (SBMM) é um formalismo que permite descrever metamodelos, os quais consistem em um tipo específico de modelos para descrever linguagens de modelagem de software tais como a UML. Este trabalho apresenta um metamodelo para programas adaptativos, definido em SBMM, que estende um metamodelo anterior encontrado na literatura pela adição de restrições. Esta especificação formal de metamodelo é introduzida no software SBMMTool juntamente com uma função de mapeamento para geração de código, que passa então a funcionar como uma ferramenta CASE para edição de modelos de programas adaptativos e abre caminho em direção à aplicação da MDE nesta área.**

***Keywords*— Tecnologias adaptativas (*adaptive technologies*), programas adaptativos (*adaptive programs*), *Model Driven Engineering*, *Set Based Meta Modeling*.**

I. INTRODUÇÃO

Um dispositivo adaptativo é composto por um dispositivo não-adaptativo subjacente e um mecanismo formado por funções adaptativas capaz de alterar o conjunto de regras que define seu comportamento [4]. Esta camada adaptativa confere ao dispositivo capacidade de automodificação, onde as alterações nas regras de comportamento são disparadas em função da configuração corrente do dispositivo e dos estímulos recebidos. Essas alterações se caracterizam pela substituição, inserção ou remoção de regras.

Um autômato adaptativo, por exemplo, é um dispositivo adaptativo que estende o conceito de autômato finito incorporando a característica de desenvolver uma autoreconfiguração em resposta a um estímulo externo [5]. Criação de novos estados e transições em tempo de execução são exemplos de autoreconfiguração.

A tecnologia adaptativa traz uma nova abordagem para resolver problemas, podendo ser aplicada em diversas áreas tais como automação e robótica [5], reconhecedores gramaticais [18] e até mesmo na área agrícola [19].

Um programa adaptativo pode ser entendido como uma especificação de uma sequência automodificável de instruções, que representa um código dinamicamente alterável [3]. Podem ser considerados dispositivos adaptativos onde o dispositivo não-adaptativo subjacente seria um programa estático.

Em um programa adaptativo, as ações adaptativas podem inserir ou remover linhas de código, antes ou depois de processar um estímulo.

Basic Adaptive Language (BADAL) é uma linguagem de programação adaptativa de alto nível proposta por [3]. Uma linguagem adaptativa deve prover instruções explícitas para alteração do código-fonte em tempo de execução, e assim o faz a BADAL. O compilador BADAL apresentado por [3] gera código para o ambiente de execução desenvolvido em [6], que é uma máquina virtual com características específicas que possibilita que um programa realize automodificações em seu código em tempo de execução.

Juntamente com a linguagem BADAL, uma representação gráfica para programas adaptativos é apresentada em [3], descrita em linguagem natural. Cada instância dessa representação gráfica é um modelo, não necessariamente completo, para um programa adaptativo, assim como um diagrama de classes UML é um modelo, também não necessariamente completo, para um sistema orientado a objetos.

Model Driven Engineering (MDE) é uma abordagem para desenvolvimento de software através da qual um sistema é construído pela transformação de modelos em código-fonte em alguma linguagem alvo. Essa transformação pode ocorrer em um número arbitrário de passos. Um modelo de um sistema é uma descrição ou especificação do mesmo considerando um determinado propósito, e por isso pode não representar todos os aspectos e propriedades do sistema por si só [8]. Usualmente um modelo consiste em uma combinação de textos e desenhos, podendo estar descrito em linguagem de modelagem (ex: UML) ou em linguagem natural [1].

Entretanto, para que a MDE seja possível na prática, é necessário que os modelos de software estejam descritos em linguagens que permitam a leitura por programas sendo, portanto, a linguagem natural inadequada para este fim. Os modelos seriam então combinados e transformados em código-fonte em uma linguagem alvo em um processo muito semelhante à compilação de código-fonte em código executável. Com isso, os modelos de software servem não apenas como documentação, mas também como artefatos fundamentais para a construção do programa. Aumenta-se o nível de abstração no desenvolvimento e ganha-se na possibilidade de geração do mesmo sistema para diversas plataformas, uma vez que o mesmo modelo poderia dar origem a código-fonte em mais de uma linguagem alvo, desde que se tenham as transformações necessárias disponíveis.

Embora atualmente a UML seja a linguagem padrão de fato para modelagem de software [9], ela apresenta alguns

[1] S. R. M. Canovas, Escola Politécnica da Universidade de São Paulo, Brasil (e-mail: sergio.canovas@usp.br).
[2] C. E. Cugnasca, Escola Politécnica da Universidade de São Paulo, Brasil (e-mail: carlos.cugnasca@poli.usp.br).

problemas para aplicação da MDE, tais como a dependência de linguagem natural para descrever certos modelos [10] e a falta de semântica clara e precisa em alguns conceitos, que causa ambiguidades em sua interpretação [11]. Além disso, a UML é uma linguagem de modelagem de propósito geral, ou *general purpose language* (GPL), e nem sempre adequada para tipos de aplicação específicos.

Por outro lado, linguagens específicas de domínio, ou *domain specific languages* (DSLs), são linguagens desenhadas para serem úteis em um conjunto específico de tarefas [12]. Elas podem ser combinadas com GPLs para construir conjuntos de modelos de forma mais adequada, ou usadas por si só caso atendam às necessidades.

Seja GPL ou DSL, é fundamental para a MDE a existência de uma forma padronizada de descrever linguagens de modelagem, afinal as transformações de modelos devem ser descritas sobre os conceitos permitidos, e não sobre elementos de modelos específicos. Linguagens de modelagem são descritas por um tipo específico de modelo denominado metamodelo. Existem, por exemplo, metamodelos que descrevem os diagramas da UML. Um metamodelo que descreve uma DSL para modelagem de programas adaptativos é apresentado em [15]. Modelos de *software* devem estar descritos conforme as regras de seus metamodelos pretendidos.

O *Object Management Group* (OMG) provê o *Meta Object Facility* (MOF) como uma linguagem para definir metamodelos, juntamente com a linguagem *Object Constraint Language* (OCL) para estabelecer restrições. Atualmente, existe um consenso geral de que o MOF está subformalizado [13], apresentando alguns problemas similares aos da UML tais como ambiguidade e complexidade. Outros formalismos da literatura [12][13][14] buscam suprir carências, mas normalmente estão focados em aspectos específicos (ex: inferência lógica ou melhorias na definição semântica) e não nos requisitos para aplicação prática da MDE como um todo. Um formalismo alternativo é o *Set Based Meta Modeling* (SBMM) [15], que permite definir metamodelos e modelos precisamente através de equações baseadas em conjuntos, relações e lógica de primeira ordem.

O objetivo deste trabalho é apresentar a utilização de uma ferramenta CASE para o desenvolvimento de programas adaptativos, utilizando o formalismo SBMM para a especificação do metamodelo necessário, dando continuidade ao trabalho apresentado em [15]. As definições do metamodelo são introduzidas na SBMMTool, uma ferramenta meta-CASE. Ferramentas meta-CASE são capazes de gerar ou emular ferramentas CASE customizadas [7], tomando metamodelos como entradas. Com isso, é possível editar modelos de programas adaptativos e ir em direção à aplicação da MDE nesse contexto, na medida em que o modelo pode posteriormente ser transformado em código-fonte BADAL.

A seção II apresenta a linguagem gráfica para representação de programas adaptativos definida por [3]. A seção III descreve um resumo do formalismo SBMM no que diz respeito à definição de metamodelos, enquanto a seção IV apresenta o metamodelo para programas adaptativos introduzido por [15], acrescentando restrições para o mesmo. A seção V apresenta a aplicação deste metamodelo na ferramenta SBMMTool, que passa a se comportar uma ferramenta CASE especializada em programas adaptativos. A seção VI discute os resultados, enquanto a seção VII apresenta as conclusões. Por fim, a seção VIII lista as referências bibliográficas.

II. Programas Adaptativos e Representação Gráfica

Primeiramente será apresentada uma notação para o dispositivo subjacente não-adaptativo do programa adaptativo, isto é, o programa estático escrito em uma linguagem de alto nível hospedeira. A Figura 1 [3] apresenta a arquitetura de um programa projetado como um dispositivo guiado por regras, onde é possível verificar uma camada de código formado por blocos básicos escritos em linguagem hospedeira (camada 1).

A camada 3 provê conexões que ligam a saída de um bloco básico à entrada de algum dos blocos básicos do programa. O valor de saída do bloco recém executado é utilizado por um decisor (camada 2), que encaminha a execução do programa para um dos blocos conectados à saída em função deste valor.

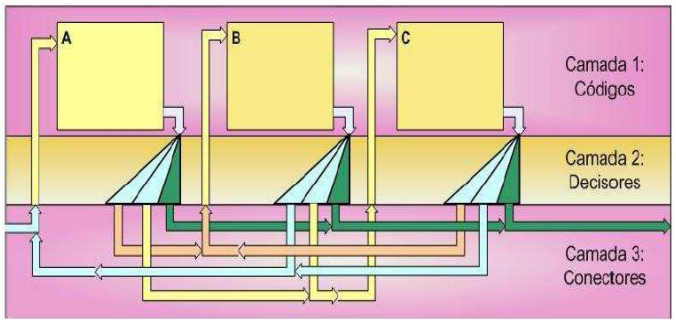


Fig. 1. Arquitetura de um programa não-adaptativo na forma de um dispositivo guiado por regras [3].

Define-se um bloco básico como sendo uma parcela do programa expressa na forma de uma sequência de comandos da linguagem hospedeira, e que deve ser descrito de tal modo que apresente uma só entrada e uma só saída. O valor de saída deve exprimir de alguma forma convencionada uma condição referente ao resultado de sua execução.

Desta forma, os programas adaptativos a serem elaborados pelo programador contêm, como elementos construtivos iniciais, blocos básicos escritos puramente na linguagem hospedeira [3]. Para assegurar a coerência estrutural dos programas assim construídos, é preciso que o programador projete adequadamente as conexões e decisores, e que o compilador faça as validações necessárias.

Cabe ao programador definir os possíveis valores de saída de cada bloco básico. Caso se obtenha um valor de saída não especificado nos decisores, BADAL determina que a cláusula OTHERWISE, obrigatória em todas as conexões, garanta que sempre haverá algum destino para o fluxo do programa após o término da execução de um bloco básico.

Uma entrada de bloco básico pode receber mais de uma conexão, conforme exemplo da Figura 1. Por outro lado, cada valor de saída de um bloco básico deve estar associado a uma única conexão, garantindo que o próximo bloco a executar seja obtido deterministicamente.

Até aqui foi definido o dispositivo não-adaptativo subjacente. A camada adaptativa é introduzida entre a camada de decisores e de conectores. Ela se responsabiliza pela capacidade de alteração do programa em tempo de execução, correspondendo a funções adaptativas. Suas chamadas ficam atreladas às conexões condicionais estabelecidas entre os blocos básicos. A Figura 2 apresenta a arquitetura atualizada com a camada adaptativa.

Na nova camada 3 aparecem todas as funções adaptativas cuja utilização esteja prevista na lógica do programa, e essas são associadas aos conectores da camada 4.

As funções adaptativas devem ser especificadas em algum lugar no corpo do programa adaptativo. A declaração de uma função adaptativa resume-se a indicar as ações de modificação do programa adaptativo, a serem efetuadas em tempo de execução nas ocasiões em que a função for ativada.

BADAL determina que as funções adaptativas se restrinjam a executar ações de inserção ou de remoção, tanto de blocos básicos como de conexões entre eles [3]. As partes do metamodelo que formalizam as funções adaptativas devem refletir esse aspecto.

A referência a um bloco básico deve ser feita por nome, enquanto a referência a uma conexão deve especificar o respectivo bloco básico de origem, bem como o valor de saída desse bloco básico que o seleciona [3].

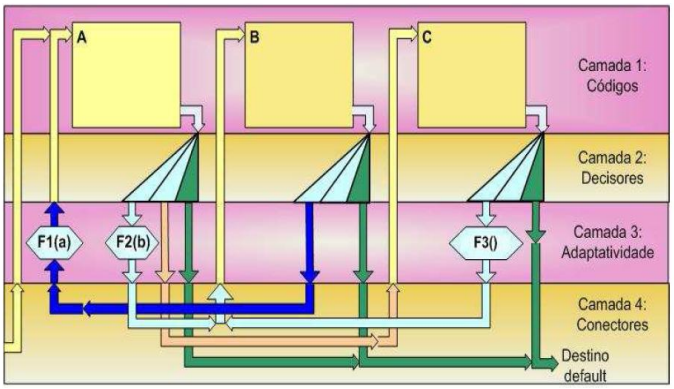


Fig. 2. Arquitetura de um programa adaptativo [3].

A seguinte seção introduzirá o SBMM, e a próxima apresentará um metamodelo formal para a representação gráfica aqui descrita.

III. METAMODELOS EM SBMM

O formalismo SBMM provê meios para descrever metamodelos, modelos e modelos de marcação, operadores de mesclagem de metamodelos, entre outros recursos. Nesta seção será apresentado um resumo do que se refere a metamodelos.

Um metamodelo no SBMM consiste em um nome (cadeia de símbolos) $name_{MM}$, um conjunto C de metaclasses, um conjunto Γ de generalizações, um conjunto E de enumerações e um conjunto R de restrições. Ou seja, um metamodelo é uma quíntupla MM:

$$MM = (name_{MM}, C, \Gamma, E, R) \quad (1)$$

Em que:

- $name_{MM} \in \Sigma^+$, onde Σ é um conjunto de símbolos pré-definido (fora da definição de MM) para a formação de cadeias que representam nomes. Por exemplo, Σ pode ser o conjunto das letras alfabeto romano com maiúsculas e minúsculas, dígitos de 0 a 9 e símbolos adicionais como *underscore*. O elemento $name_{MM}$ representa o nome do metamodelo, servindo para implementar capacidade equivalente a dos identificadores do MOF.
- $C = \{c_1, ..., c_n\}$ é o conjunto finito, eventualmente vazio, de metaclasses. Uma metaclasse representa uma abstração de um conceito de modelagem. Por exemplo, na UML, os conceitos de classe, associação, caso de uso e ator são metaclasses. As classes, associações, casos de uso e atores presentes em um modelo UML de um software são instâncias dessas metaclasses.
- $\Gamma \subset C \times C$ é uma relação não reflexiva livre de ciclos que mapeia submetaclasses em suas supermetaclasses, representando o conceito de herança da orientação a objetos. Não pode ser reflexiva, pois uma metaclasse não é submetaclasse dela mesma. Por não permitir ciclos nem reflexão, então Γ é subconjunto próprio de $C \times C$ desde que $C \neq \varnothing$. A relação Γ indica as relações diretas de herança, enquanto seu fecho transitivo Γ^+ contém também as relações indiretas.
- $E = \{e_1, ..., e_m\}$é o conjunto finito, eventualmente vazio, de enumerações.
- $R = \{r_1,..., r_l\}$ é o conjunto finito, eventualmente vazio, de restrições sobre modelos baseados em MM. Cada restrição é uma sentença em lógica de primeira ordem em que metaclasses e enumerações são vistas como predicados unários e propriedades são enxergadas como enumerações. A expressão lógica $r_1 \wedge r_2 \wedge ... \wedge r_l$ não pode consistir uma falácia, caso contrário qualquer modelo baseado em MM seria não conforme. Um melhor entendimento sobre as restrições será possibilitado através de exemplos mais adiante na seção IV.

A metaclasse a partir da qual um elemento de modelo foi instanciado é denominada de seu tipo base (*base type*). As supermetaclasses dessa metaclasse, e também ela mesma, são denominadas de tipos do elemento (*type*). Logo, o tipo base também um tipo do elemento.

Cada metaclasse c_i possui um nome w_i e um conjunto de propriedades P_i. Sendo assim, podemos defini-las como pares ordenados:

$$c_i = (w_i, P_i) \quad (2)$$

Em que:

- $w_i \in \Sigma^+$ é a cadeia de símbolos que nomeia c_i.
- $P_i = \{p_{i1}, ..., p_{ik}\}$ é o conjunto finito de propriedades, eventualmente vazio.

Os nomes devem ser únicos dentro do metamodelo, tais que se $c_i \neq c_j$ então $w_i \neq w_j$.

Do ponto de vista semântico, as propriedades $p_{ij} \in P_i$ definem *slots* de informação para elementos de modelos (instâncias) da metaclasse C_i e de suas submetaclasses.

Cada propriedade p_{ij} consiste em um nome, multiplicidade, tipo alvo (que pode ser uma metaclasse ou enumeração) e uma multiplicidade que restringe quantos elementos o *slot* de informação representado consegue armazenar. Isto é:

$$p_{ij} = (v_{ij}, t_{ij}, m_{ij}) \quad (3)$$

Em que:

- $v_{ij} \in \Sigma^+$ é a cadeia de símbolos que nomeia p_{ij}.
- $t_{ij} \in C \cup E$ é o tipo alvo da propriedade, ou seja, pode ser uma metaclasse ou uma enumeração do metamodelo.
- $m_{ij} \in \mathbf{N} \times (\mathbf{N}^+ \cup \{*\})$ é a multiplicidade da propriedade, sendo um par ordenado cujo primeiro elemento é um número natural que denota o limite inferior de *slots* que a propriedade é capaz de armazenar. O segundo elemento pode ser um número natural positivo ou um símbolo *, que representa infinito, denotando o limite superior de *slots*. Se $m_{ij} = (1,1)$, por exemplo, isso significa que a propriedade possui um e apenas um *slot* de informação. Se $m_{ij} = (0,*)$, então a propriedade representa uma lista com um número arbitrário de *slots*. Na UML e no MOF as multiplicidades não necessariamente estão dentro de uma faixa de valores inicial e final, podendo ser valores arbitrários tais como 1, 3 e 5 (sem incluir 2 e 4). Mas para os propósitos do SBMM essa definição é suficiente. Para tornar a notação mais semelhante a da UML e MOF, um par ordenado de multiplicidade (x,y) também pode ser denotado por $x..y$.

Os nomes das propriedades devem ser únicos dentro da metaclasse, tais que se $p_{ij} \neq p_{ik}$ então $v_{ij} \neq v_{ik}$.

As enumerações $e_i \in E$ servem para definir tipos de dados básicos cujas instâncias podem assumir um valor de um conjunto finito, definido pela própria enumeração. Ou seja, cada enumeração pode ser definida por:

$$e_i = (u_i, L_i) \quad (4)$$

Em que:

- $u_i \in \Sigma^+$ é a cadeia de símbolos que nomeia e_i.
- L_i é o conjunto finito de valores que as instâncias de e_i podem assumir.

Os nomes devem ser únicos dentro do metamodelo, tais que se $e_i \neq e_j$ então $u_i \neq u_j$.

Por exemplo, pode-se definir uma enumeração de um tipo básico booleano como:

$$e_1 = (\text{“Boolean”}, \{true, false\}) \quad (5)$$

As cadeias de símbolos que representam nomes são denotadas entre aspas para evitar confusão com referências a outros elementos do metamodelo ou valores permitidos em enumerações, ou seja, não confundir a propriedade *p* com o eventual nome (cadeia de símbolos) de um elemento do metamodelo “p”.

Estender a definição de enumeração para aceitar conjuntos L_i infinitos é simples, mas não faz sentido na prática pois uma variável de computador é sempre armazenada em um número finito de bits, e portanto uma instância de enumeração na prática não pode assumir um dentre infinitos valores. Sendo assim, um tipo básico de número inteiro de 64 bits com sinal pode ser definido pela seguinte enumeração:

$$e_2 = (\text{“Integer64”}, \{x \mid (x \in \mathbf{Z}) \wedge (-2^{63} \leq x \leq 2^{63} - 1)\}) \quad (6)$$

Uma enumeração para o tipo String sobre um alfabeto Σ pode ser definida por:

$$e_3 = (\text{“String”}, \{x \mid (x \in \Sigma^*) \wedge (|x| \leq h)\}) \quad (7)$$

A restrição $|x| \leq h$ para algum limitante superior *h* garante que a lista seja finita. Na prática, as linguagens de programação permitem variáveis do tipo String de tamanhos bastante elevados, de forma que *h* é muito grande. Normalmente é limitado pela memória disponível ou alguma característica da linguagem de programação utilizada.

Entretanto, como o SBMM é independente de implementação e usa apenas conceitos teóricos, para facilitar definições poderão ser utilizadas enumerações com infinitos valores permitidos, tais como:

$$e_4 = (\text{“Integer”}, \mathbf{Z}) \quad (8)$$

Isso porque, do ponto de vista teórico, não há impedimentos para que uma enumeração permita infinitos valores possíveis. Até mesmo conjuntos infinitos não enumeráveis poderiam ser usados, como por exemplo:

$$e_5 = (\text{“Real”}, \mathbf{R}) \quad (9)$$

O leitor deve subentender que, ao ser transportado para uma implementação computacional, os eventuais conjuntos infinitos de valores de enumerações deverão ser substituídos por equivalentes práticos. Por exemplo, a enumeração e_2 apresentada acima é uma implementação usual de e_4, já que inteiros de 64 bits resolvem a maioria dos problemas que precisam manipular inteiros em geral.

As enumerações não contêm propriedades e nem possuem uma relação de generalização.

Cada elemento do conjunto *R* de restrições é uma sentença em lógica de primeira ordem que impõe alguma restrição pretendida para os modelos. A seção IV clarificará como funcionam as restrições através de exemplos.

IV. METAMODELO PARA PROGRAMAS ADAPTATIVOS

Para facilitar o entendimento do metamodelo proposto, as sentenças que o compõem serão apresentadas ao longo das explicações. Esse metamodelo foi originalmente introduzido em [15], mas não apresentava o conjunto *R*. Este trabalho acrescenta este conjunto e seus elementos, explicando a interpretação de cada sentença restritiva.

Seja MM_{PA} um metamodelo que descreve a representação de programas adaptativos apresentada na seção II.

$$MM_{PA} = (name_{MM\_PA}, C, \Gamma, E, R) \quad (10)$$

Inicialmente define-se $name_{MM\_PA}$ = "AdaptiveProgram Metamodel" como o nome do metamodelo. Introduz-se então no conjunto E duas enumerações de tipos básicos e_1 e e_2, já discutidas anteriormente. Os índices foram trocados para manter a sequência dentro deste metamodelo.

- $e_1 = (\text{"String"}, \{x \mid (x \in \Sigma^*) \wedge (|x| \leq h)\})$
- $e_2 = (\text{"Integer"}, \boldsymbol{Z})$

Essas enumerações serão referenciadas como tipos de propriedades nas metaclasses.

Conforme mostrado na Figura 2, identifica-se que um programa adaptativo é composto por quatro componentes fundamentais: bloco básico, decisor, função adaptativa e conexão. Serão criadas metaclasses para abstrair esses conceitos, bem como metaclasses auxiliares.

Seja $c_1 \in C$ a metaclasse que representa um bloco de código básico. Por convenção, todos os nomes de metaclasse serão iniciados com o prefixo *MC*.

- $c_1 = (\text{"MCBasicBlock"}, P_1)$
  - $P_1 = \{p_{11}, p_{12}\}$
    - $p_{11} = (\text{"Name"}, e_1, 1..1)$
    - $p_{12} = (\text{"Code"}, e_1, 1..1)$

Um bloco básico é composto por um nome e um código-fonte na linguagem hospedeira, neste caso BADAL, ambos representados como as propriedades do tipo String p_{11} e p_{12}. Neste metamodelo apresentado por [15] não se entra no mérito das regras sintáticas do código na linguagem hospedeira, supondo que este seja um problema à parte. O foco será dado aos aspectos arquiteturais do modelo do programa adaptativo apresentados na Figura 2.

O nome do bloco básico não pode ser vazio, e para isso introduz-se a restrição $r_1 \in R$:

- r_1: $\forall o \mid \text{MCBasicBlock}(o) \rightarrow \neg (\text{MCBasicBlock.Name}(o) = \varepsilon)$

A interpretação de r_1 é que para todo elemento do modelo *o*, se *o* for do tipo *MCBasicBlock*, então o valor atribuído à propriedade *Name* de *o* não é vazio. O símbolo ε denota a cadeia (*string*) vazia. Como pôde ser visto, as metaclasses (neste caso *MCBasicBlock*) são enxergadas como predicados unários nas expressões das sentenças de *R*. O mesmo vale para enumerações. As propriedades, referenciadas pelo nome da metaclasse seguido por um ponto e seu próprio nome (neste caso *MCBasicBlock.Name*), são funções que retornam o valor contido no *slot* correspondente no elemento do modelo passado como argumento. Caso a propriedade tenha multiplicidade maior que a unitária, então é retornado um conjunto de valores.

Também não pode haver blocos básicos com o mesmo nome dentro do mesmo modelo. Nesse caso a restrição $r_2 \in R$ exerce esta imposição:

- r_2: $\forall o_1, o_2 \mid \text{MCBasicBlock}(o_1) \wedge \text{MCBasicBlock}(o_2) \wedge (\text{MCBasicBlock.Name}(o_1) = \text{MCBasicBlock.Name}(o_2)) \rightarrow o_1 = o_2$

Interpreta-se r_2 de forma que para todos os elementos do modelo o_1 e o_2, se ambos forem do tipo *MCBasicBlock* e seus nomes forem iguais, então obrigatoriamente $o_1 = o_2$.

Seja $c_2 \in C$ a metaclasse que abstrai a conexão adaptativa, que conecta a saída de um bloco básico a um ou mais blocos básicos conforme valores de saída.

- $c_2 = (\text{"MCAdaptiveConnection"}, P_2)$
  - $P_2 = \{p_{21}, p_{22}, p_{23}\}$
    - $p_{21} = (\text{"From"}, c_1, 1..1)$
    - $p_{22} = (\text{"ConditionalConnections"}, c_3, 0..*)$
    - $p_{23} = (\text{"OtherwiseConnection"}, c_4, 1..1)$

A propriedade p_{21} representa o bloco básico de origem da conexão, enquanto p_{22} referencia as possíveis múltiplas conexões condicionais que partem do bloco. Todo bloco deve ter uma e apenas uma conexão *default* caso a saída em tempo de execução fornecida pelo bloco não corresponda a nenhuma conexão condicional. Essa única conexão é representada por p_{23}.

As propriedades p_{22} e p_{23} definem *slots* de informação para instâncias de c_3 e c_4 respectivamente. Essas, por sua vez, são metaclasses que representam uma conexão condicional e uma conexão *default*, na ordem. Uma conexão condicional nada mais é que uma conexão *default* acrescida de um valor que define que ela será acionada quando a saída do bloco for igual a este valor. Como possuem aspectos comuns, pode-se definir c_4 inicialmente e depois criar c_3 como submetaclasse de c_4.

- $c_4 = (\text{"MCGeneralConnnection"}, P_4)$
  - $P_4 = \{p_{41}, p_{42}, p_{43}\}$
    - $p_{41} = (\text{"To"}, c_1, 1..1)$
    - $p_{42} = (\text{"AdaptiveFuncCallBefore"}, c_5, 0..1)$
    - $p_{43} = (\text{"AdaptiveFuncCallAfter"}, c_5, 0..1)$
- $c_3 = (\text{"MCConditionalConnection"}, P_3)$
  - $P_3 = \{p_{31}\}$
    - $p_{31} = (\text{"OutputValue"}, e_2, 1..1)$

Para que a metaclasse da conexão condicional c_3 seja de fato submetaclasse de c_4, é necessário introduzir o par ordenado (c_3, c_4) em Γ. Até aqui, portanto, $\Gamma = \{(c_3, c_4)\}$. A semântica do SBMM estabelece que c_3 herda as propriedades de c_4.

Observar que a propriedade p_{31}, que representa o valor de saída que decide a utilização da conexão é definido como tipo inteiro (e_2), de acordo com as definições de [3].

Para não haver ambiguidades nos modelos, é necessário estabelecer uma restrição que indique que não pode haver mais de uma conexão condicional baseada no mesmo valor de saída partindo do mesmo bloco básico. Para isso, introduz-se r_3:

- r_3: $\forall a, o_1, o_2 \mid$ MCAdaptiveConnection(a) $\wedge$ MCConditionalConnection(o_1) $\wedge$ MCConditionalConnection(o_2) $\wedge$ ($o_1 \in$ MCAdaptiveConnection.ConditionalConnections(a)) $\wedge$ ($o_2 \in$ MCAdaptiveConnection. ConditionalConnections(a)) $\wedge$ (MCConditionalConnection.OutputValue(o_1) = MCConditionalConnection.OutputValue(o_2)) $\rightarrow o_1 = o_2$

Interpreta-se r_3 de modo que para todos os elementos do modelo o_1 e o_2, se ambos forem do tipo *MCConditionalConnection* e pertencerem ao mesmo objeto a do tipo *MCAdaptiveConnection*, caso seus valores condicionantes sejam iguais então obrigatoriamente $o_1 = o_2$.

Aparece nas definições de p_{42} e p_{43} a metaclasse c_5, ainda não mencionada. Essa metaclasse deve ser definida de modo a representar uma chamada de função adaptativa. Este tipo de chamada se caracteriza por uma função adaptativa alvo e uma lista ordenada, eventualmente vazia, de blocos básicos passados como parâmetros, conforme define [3]. Para refletir esses aspectos, cria-se então a metaclasse $c_5 \in C$.

- c_5 = (*"MCAdaptiveFunctionCall"*, P_5)
  - P_5 = {p_{51}, p_{52}}
    - p_{51} = (*"AdaptiveFunction"*, c_7, 1..1)
    - p_{52} = (*"ParametersValues"*, c_6, 0..*)

Uma vez que a multiplicidade da propriedade p_{52} é 0..*, o que significa que a chamada pode passar um número arbitrário de parâmetros (inclusive nenhum), é necessário introduzir a metaclasse c_6 que representa um par nome/valor, evitando não determinismos caso haja mais de um parâmetro sendo passado. Os nomes dos parâmetros da chamada devem corresponder aos nomes dos parâmetros de entrada definidos na especificação da função adaptativa. Mais adiante, a restrição r_9 garantirá este requisito.

- c_6 = (*"MCAdaptiveFuncParamValue"*, P_6)
  - P_6 = {p_{61}, p_{62}}
    - p_{61} = ("ParameterName", e_1, 1..1)
    - p_{62} = ("ParameterValue", c_1, 1..1)

Na mesma chamada de função adaptativa, não podem ser passados dois parâmetros com o mesmo nome.

- r_4: $\forall a, o_1, o_2 \mid$ MCAdaptiveFunctionCall(a) $\wedge$ MCAdaptiveFuncParamValue(o_1) $\wedge$ MCAdaptiveFuncParamValue(o_2) $\wedge$ ($o_1 \in$ MCAdaptiveFunctionCall.ParametersValues(a)) $\wedge$ ($o_2 \in$ MCAdaptiveFunctionCall. ParametersValues(a)) $\wedge$ (MCAdaptiveFuncParamValue.ParameterName(o_1) = MCAdaptiveFuncParamValue. ParameterName (o_2)) $\rightarrow o_1 = o_2$

A construção da expressão r_4 é exatamente igual à construção de r_3, trocando-se apenas os nomes das metaclasses e propriedades. Interpreta-se da mesma maneira.

Continuando a criação do metamodelo, estabelece-se a metaclasse c_7 para representar as funções adaptativas em si, lembrando que c_5 representa apenas a chamada de uma função adaptativa associada a um conector, e não a especificação da função em si. A metaclasse c_7 fará esse papel.

- c_7 = (*"MCAdaptiveFunction"*, P_7)
  - P_7 = {p_{71}, p_{72}, p_{73}}
    - p_{71} = (*"Name"*, e_1, 1..1)
    - p_{72} = (*"ParametersNames"*, e_1, 0..*)
    - p_{73} = (*"Code"*, e_1, 1..1)

Assim como este metamodelo não entrou no mérito do código dos blocos básicos em linguagem hospedeira, representado por p_{12}, tendo sido o mesmo modelado apenas como uma *string*, o mesmo se aplica a p_{73}, servindo como *slot* para armazenar o código que implementa a função adaptativa em linguagem de programação adaptativa. A restrição r_5 impede funções adaptativas com nome vazio, enquanto r_6 não permite nomes repetidos no mesmo modelo. Já r_7, por sua vez, impede que uma função adaptativa defina um parâmetro com nome vazio, ao mesmo tempo em que r_8 garante que não há dois parâmetros com o mesmo nome.

- r_5: $\forall o \mid$ MCAdaptiveFunction(o) $\rightarrow \neg$ (MCAdaptiveFunction.Name (o) = ε)
- r_6: $\forall o_1, o_2 \mid$ MCAdaptiveFunction(o_1) $\wedge$ MCAdaptiveFunction(o_2) $\wedge$ (MCAdaptiveFunction.Name(o_1) = MCAdaptiveFunction. Name(o_2)) $\rightarrow o_1 = o_2$
- r_7: $\forall a, o \mid$ MCAdaptiveFunction(a) $\wedge$ String(o) $\wedge$ ($o \in$ MCAdaptiveFunction.ParametersNames(a)) $\rightarrow \neg$(o = ε)
- r_8: $\forall a, o_1, o_2 \mid$ MCAdaptiveFunction(a) $\wedge$ String(o_1) $\wedge$ String(o_2) $\wedge$ ($o_1 \in$ MCAdaptiveFunction.ParametersNames(a)) $\wedge$ ($o_2 \in$ MCAdaptiveFunction.ParametersNames(a)) $\wedge$ (String.Value(o_1) = String.Value(o_2))) $\rightarrow o_1 = o_2$

Por definição, os parâmetros das funções adaptativas correspondem a blocos básicos, não podendo ser de outro tipo. Por isso não é necessário prever propriedades para tipos de parâmetros.

A restrição r_9, citada anteriormente, deve garantir que para toda chamada de função adaptativa existente em um modelo, ou seja, instâncias de c_5 (*MCAdaptiveFunctionCall*), sejam informados apenas parâmetros esperados pela função chamada.

- r_9: $\forall a, o, f \mid$ MCAdaptiveFunctionCall(a) $\wedge$ MCAdaptiveFuncParamValue(o) $\wedge$ ($o \in$ MCAdaptiveFunctionCall.ParametersValues(a)) $\wedge$ MCAdaptiveFunction(f) $\wedge$ (MCAdaptiveFunctionCall.AdaptiveFunction(a) = f) $\rightarrow$ MCAdaptiveFuncParamValue.ParameterName(o) $\in$ MCAdaptiveFunction.ParametersNames(f)

Por fim, cria-se a metaclasse c_0, que estabelece os blocos de entrada e saída do programa adaptativo, conforme especificado em [3]. Serve como agregador principal dos blocos, conexões e funções adaptativas.

- c_0 = (*"MCAdaptiveProgram"*, P_0)
  - P_0 = {p_{01}, p_{02}, p_{03}, p_{04}, p_{05}, p_{06}}
    - p_{01} = (*"Name"*, e_1, 1..1)
    - p_{02} = (*"EntryBlock"*, c_1, 1..1)
    - p_{03} = (*"ExitBlock"*, c_1, 1..1)
    - p_{04} = (*"OtherBlocks"*, c_1, 0..*)
    - p_{05} = (*"AdaptiveConnections,* c_2, 1..*)
    - p_{06} = (*"AdaptiveFunctions"*, c_7, 0..*)

Como restrição, um modelo de programa adaptativo construído sobre o metamodelo aqui proposto deve possuir uma e apenas uma instância de c_0. Ou seja, não é permitido que um modelo contenha mais de um programa adaptativo, e nem que haja ausência do mesmo. As restrições r_{10} e r_{11} estabelecem, respectivamente, que existe um programa adaptativo no modelo e que ele é único. Enquanto isso, r_{12} impede que o programa adaptativo tenha nome vazio.

- r_{10}: $\exists o \mid$ MCAdaptiveProgram(o)
- r_{11}: $\forall o_1, o_2 \mid$ MCAdaptiveProgram(o_1) $\wedge$ MCAdaptiveProgram(o_2) $\rightarrow o_1 = o_2$
- r_{12}: $\forall o \mid$ MCAdaptiveProgram (o) $\rightarrow \neg$ (MCAdaptiveProgram.Name (o) = ε)

Um aspecto interessante é que neste metamodelo as conexões que saem dos blocos foram definidas não como propriedades dos próprios blocos (metaclassse c_1), mas sim como elementos externos (metaclasse c_2). Esta forma é coerente com o fato de enxergar a camada de conectores e de adaptatividade da Figura 2 como desacopladas das definições dos blocos, podendo ser remanejadas em cada modelo de programa adaptativo enquanto se reusa blocos já existentes de outros programas prévios. Como restrição, em um modelo de programa adaptativo não pode haver mais de uma instância da metaclasse c_2 que referencia o mesmo bloco na propriedade p_{21}, do contrário pode ocorrer não determinismos em tempo de execução. Para isso introduz-se a restrição r_{13}.

- r_{13}: $\forall o_1, o_2 \mid$ MCAdaptiveConnection(o_1) $\wedge$ MCAdaptiveConnection(o_2) $\wedge$ (MCAdaptiveConnection.From(o_1) = MCAdaptiveConnection.From(o_2)) $\rightarrow o_1 = o_2$

Em resumo, os conjuntos da quíntupla do metamodelo MM_{PA} são:

- C = {c_0, c_1, c_2, c_3, c_4, c_5, c_7}
- Γ = {(c_3,c_4)}
- E = {e_1, e_2}
- R = {r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}}

O SBMM propõe uma notação gráfica similar a do MOF e UML, na qual se representa as metaclasses e enumerações por retângulos nomeados. As propriedades são representadas por linhas que ligam a metaclasse ao tipo alvo, que pode ser também uma metaclasse ou enumeração. As linhas são rotuladas pelo nome da propriedade e direcionadas com uma seta ao tipo alvo. Nesta notação, o metamodelo MM_{PA} está representado pela Figura 3.

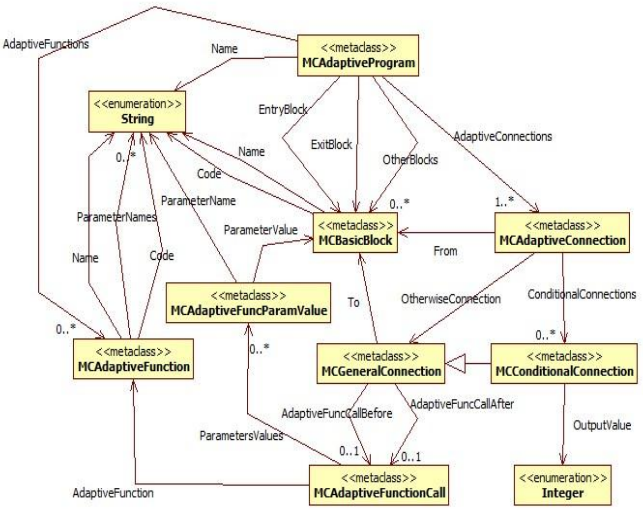


Fig. 3. Notação gráfica do metamodelo.

V. Ferramenta Case para Programas Adaptativos

A ferramenta SBMMTool é um software desenvolvido como parte da pesquisa que originou o SBMM, que permite a edição de metamodelos, modelos e funções de mapeamento, bem como a geração de código-fonte a partir dos modelos baseada nas funções de mapeamento. Trata-se de uma ferramenta meta-CASE. Esse tipo de ferramenta é capaz de gerar ou emular ferramentas CASE customizadas a partir de metamodelos de entrada [7]. Nesse caso, foi inserido na ferramenta o metamodelo para programas adaptativos apresentado na seção IV. Essa inserção é feita através de funcionalidades providas pela ferramenta, tais como inserção e edição de metaclasses, propriedades, enumerações, etc. A Figura 4 ilustra uma tela da ferramenta, na qual são montados os elementos da quíntupla que define o metamodelo definido na seção IV.

As sentenças em lógica de primeira ordem são inseridas na ferramenta com algumas adaptações sintáticas para trabalhar apenas com caracteres ASCII. Por exemplo, o símbolo $\forall$ é denotado por `forall` e $\neg$ é representado por `not`.

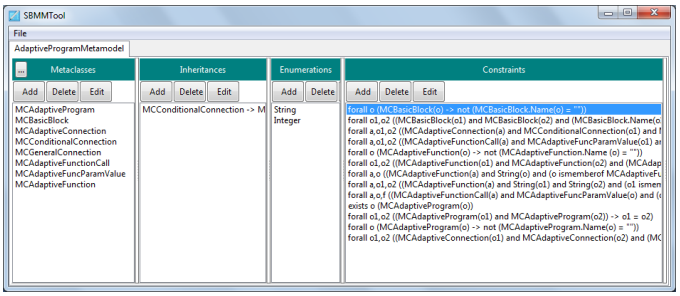


Fig. 4. Tela da ferramenta SBMMTool com o metamodelo AdaptiveProgramMetamodel.

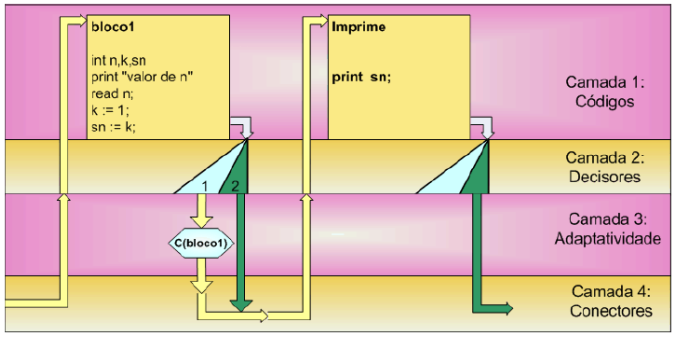


Fig. 5. Modelo de programa adaptativo exemplo [3].

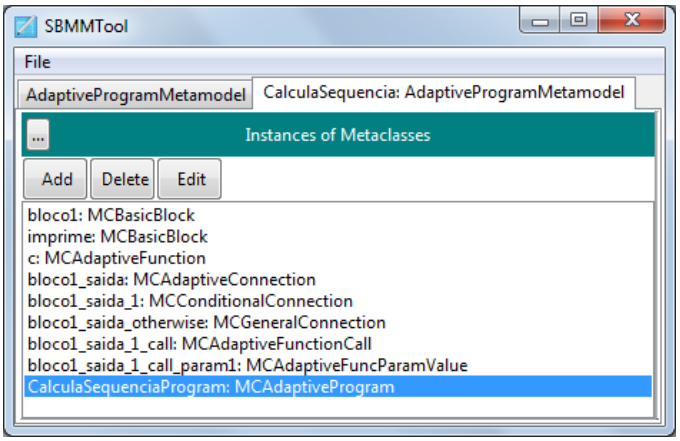


Fig. 6. Modelo de programa adaptativo exemplo editado na ferramenta SBMMTool.

Uma vez que o metamodelo *AdaptiveProgramMetamodel* foi carregado na SBMMTool, é possível criar modelos de programas adaptativos. A ferramenta é responsável por aplicar todas as restrições do conjunto *R* sobre o modelo em edição, garantindo a geração de modelos conformes. A Figura 5 ilustra a tela de edição do modelo de um programa adaptativo exemplo apresentado em [3]. Esse programa calcula o n-ésimo termo da sequência matemática $S(n) = S(n-1) + 2n$ para $n \geq 2$, sendo $S(1) = 1$. A estrutura do programa consiste em dois blocos: *bloco1* e *imprime*. O *bloco1* é responsável por solicitar ao usuário o valor de *n*. Uma função adaptativa *c* é colocada em uma conexão condicional na saída de *bloco1* e insere código no programa para processar mais uma iteração da sequencia quando ainda necessário. O bloco *imprime* é responsável por apresentar o resultado final ao usuário. A Figura 5 mostra o modelo do programa conforme representação gráfica apresentada na seção II [3]. A Figura 6, por sua vez, mostra esse modelo editado na ferramenta SBMMTool.

Como cada bloco básico deve gerar um valor de saída (exceto o último bloco a ser processado), é necessário então atribuir a *bloco1* um valor de saída e um decisor. O decisor direcionará a execução de acordo com o valor de saída. Quando o valor de entrada digitado pelo usuário (variável *n*) é *1*, já se sabe que $S(1) = 1$ e, portanto, o resultado é apenas impresso sem necessidade de cálculo. Para os termos a partir do segundo, a variável *k* de *bloco1* guarda o número de ordem do termo cujo valor foi computado, o qual é armazenado na variável *sn*. A comparação de *k* com *n* permite saber se o termo requerido pelo usuário já foi encontrado, condição que está implementada no decisor. Por convenção, o valor de saída será *1* se o termo *n* ainda não foi atingido e *2* caso contrário. Esse valor é testado pela conexão para determinar o próximo bloco básico a ser processado. O código fonte de *bloco1* e *imprime* pode ser conferido a seguir [3].

```
code bloco1: <
   int n,k,sn;
   print "valor de n";
   read n;
   k := 1;
   sn := k;
   if n>k then bloco1 := 1 else bloco1 := 2;
>;

code imprime: <
   print sn;
>;
```

O bloco *imprime* não necessita de um decisor pois corresponde ao bloco de saída do programa, ou seja, à propriedade *ExitBlock* da única instância de programa adaptativo permitida no modelo , isto é, da metaclasse *MCAdaptiveProgram*.

De acordo com a Figura 5, o valor de saída *1* do bloco *bloco1* indica que o valor informado pelo usuário é maior que *1* e, portanto, é necessário computar os termos seguintes da sequência até o *n*-ésimo. Sendo assim, a conexão que o interliga ao bloco *imprime* chama a função adaptativa *c*, que calculará o *n*-ésimo termo antes de processar o bloco *imprime*.

A montagem da conexão, juntamente com os respectivos valores de decisão e a chamada da função adaptativa para a saída *1*, estão presentes no modelo do programa conforme a Figura 6, mais precisamente de *bloco1_saida* até *bloco1_saida_1_call_param1*. A Figura 7 apresenta uma outra tela da ferramenta com detalhes do preenchimento dos valores das propriedades para a instância do modelo *bloco1_saida*.

A função adaptativa *c*, por sua vez, é responsável por instanciar um novo bloco básico, não presente no modelo original. Também é instanciada uma conexão deste novo bloco básico para *imprime*, passando por um decisor igual ao de *bloco1*, que novamente aciona a função adaptativa *c* para a saída igual a *1*. A condição para a saída deste novo bloco instanciado resultar em *2* é que o valor de *k* atinja o valor de *n*, provocando então a parada do laço de repetição montado através de sucessivas instanciações de blocos básicos pela função adaptativa *c*. O código desta função adaptativa em BADAL pode ser conferido a seguir [3].

```
adaptive function c(i) [generators g] {
   insert [ code g: <
      k := k + 1;
      sn := sn + 2 * k;
      if n>k then g := 1 else g := 2;
      >;
   ];
   insert [ connection from g (case 1: to
imprime perform c(g) before otherwise to
imprime)
   ];
}
```

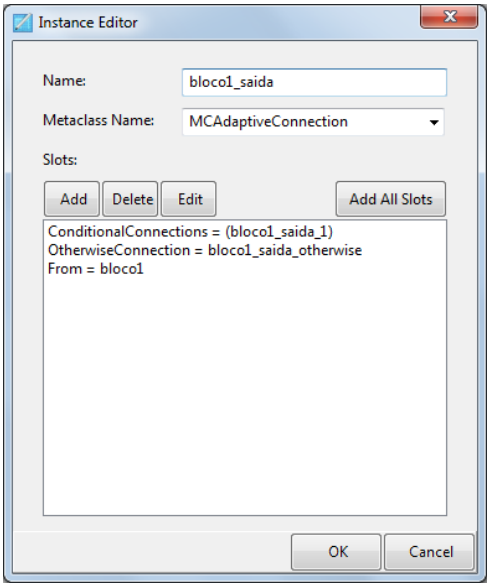


Fig. 7. Editor de instância da ferramenta SBMMTool.

A ferramenta SBMMTool permite a definição e execução de funções de mapeamento do tipo modelo para código, ou seja, funções que tomam modelos como entrada e geram como saída uma cadeia de caracteres representando código-fonte. A definição dessas funções de mapeamento é feita através da linguagem *MOF Model To Text Transformation Language* (MOFM2T) [17]. Esta linguagem é baseada em *templates* de código-fonte dentro dos quais é possível realizar iterações e extração de valores de propriedades dos modelos de entrada.

Embora tenha sido originalmente projetada para trabalhar com metamodelos descritos em MOF, a ferramenta SBMMTool implementa um interpretador baseado em um subconjunto da MOFM2T para operar sobre metamodelos descritos em SBMM. Essa possibilidade existe pois os conceitos principais de metamodelagem e modelagem do SBMM (metaclasse, propriedade, instância e *slot*) são semanticamente os mesmos do MOF, mesmo que com diferenças técnicas nas definições e notações.

Como exemplo, considere-se o diagrama de classes UML da Figura 8, contendo uma única classe.

O *template classToJava* [17], escrito em MOFM2T, é capaz de gerar código Java para a estrutura básica das classes de um modelo, contendo seus atributos e um construtor vazio.

```
[template public classToJava(c : Class)]
class [c.name/] {
   // Attribute declarations
   [for(a : Attribute | c.attribute)]
   [a.type.name/] [a.name/];
   [/for]

   // Constructor
   [c.name/]() {}
}
[/template]
```

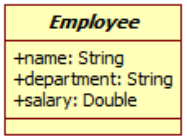


Fig. 8. Exemplo de diagrama de classes.

As expressões entre colchetes fazem com que o interpretador realize algum processamento, tais como iterações ou extração de valores de propriedades do modelo, para então gerar a saída de caracteres naquela posição. O exemplo acima contém um laço do tipo *for*, que itera sobre os atributos da classe, e também extrações de valores de propriedades de tipos primitivos. A saída gerada por esta função de mapeamento, quando aplicada sobre o modelo da Figura 8, está reproduzida a seguir.

```
class Employee {
  // Attribute declarations
  String name;
  String department;
  Double salary;

  // Constructor
  Employee() {}
}
```

Uma função de mapeamento em MOFM2T capaz de gerar código-fonte BADAL para modelos baseados no metamodelo apresentado na seção IV foi criada como parte deste trabalho. Por questões de espaço, apenas o trecho do código desta função que gera o cabeçalho do programa e o fechamento de sua estrutura principal está reproduzido a seguir. O código completo está disponível para *download* no link http://www.p2s.com.br/team/scanovas/wta2015/amtobadal.txt.

```
[template public Main()]
  [if(MCAdaptiveProgram[0])]
  [file('AdaptiveProgram.txt',0)]ADAPTIVE MAIN
\[NAME = [echo(MCAdaptiveProgram[0].Name) /],
ENTRY = [value(MCAdaptiveProgram[0].
EntryBlock.Name) /], EXIT = [echo(
MCAdaptiveProgram[0].ExitBlock.Name) /] ] IS

  ...

  END MAIN.
  [/file]
  [/if]
[/template]
```

Com isso, a SBMMTool passa a se comportar como uma ferramenta CASE para elaboração de modelos de programas adaptativos, com possibilidade de geração automática de código BADAL. O modelo do programa de cálculo de sequência apresentado nesta seção pode ser convertido em código-fonte através da execução da transformação baseada

nesta função de mapeamento, podendo o resultado ser obtido pelo link http://www.p2s.com.br/team/scanovas/wta2015 /calcseq.txt.

VI. Resultados e Discussão

O presente trabalho apresenta a utilização da ferramenta SBMMTool para edição de modelos de programas adaptativos. Esse resultado é obtido graças às equações e sentenças apresentadas na seção IV, que especificam formalmente um metamodelo para programas adaptativos, adicionando restrições sobre modelos com relação ao trabalho apresentado em [15].

Existem diversos tipos de ferramentas CASE, desde aquelas que servem apenas para desenhar modelos baseados em notações gráficas, tais como UML, até ferramentas capazes de transformar modelos em outros modelos ou em código-fonte, ponto chave para aplicação da MDE. A ferramenta SBMMTool provê a capacidade deste tipo de transformação através de um interpretador de um subconjunto da linguagem MOFM2T [17], permitindo a geração automática de código a partir de modelos.

No entanto, um modelo descrito no metamodelo proposto será capaz de gerar código adaptativo apenas no que diz respeito à estrutura dos blocos e conexões. A implementação dos blocos e funções adaptativas em si foram modeladas como propriedades *String*, ou seja, a SBMMTool ou outra ferramenta que permita o programador editar um modelo conforme ao metamodelo apresentado na seção IV considera que essas implementações são texto livre, devendo o programador preencher código na linguagem hospedeira e linguagem adaptativa de interesse. Futuras extensões do metamodelo proposto podem considerar detalhes dos aspectos de implementação ao menos das funções adaptativas, permitindo que o modelo contenha informações completas sobre sua implementação nativamente, sem depender de nenhuma linguagem específica baseada em texto tal como a própria BADAL.

Outro aspecto a ser considerado é a forma com a qual os modelos são editados na ferramenta. Por ser genérica e trabalhar com qualquer metamodelo válido, de programa adaptativo ou não, a ferramenta SBMMTool permite a edição de modelos através da inserção de instâncias de metaclasses e preenchimento de propriedades em listas. Essa forma não é muito conveniente, sendo as notações gráficas mais adequadas para facilitar o trabalho da construção e entendimento de modelos. A própria UML não teria obtido sucesso se não fosse primariamente baseada em uma sintaxe concreta composta por desenhos. Trabalha-se também em um editor de sintaxe concreta para a SBMMTool, que permite editar os símbolos gráficos utilizados para cada metaclasse. Desse modo, as instâncias inseridas atualmente em forma de listas poderiam ser arrastadas para um editor gráfico que permitiram a modelagem do programa adaptativo na representação gráfica apresentada na seção II.

VII. Conclusão

O presente trabalho mostra um caminho em direção à utilização da abordagem MDE para o desenvolvimento de programas adaptativos. A MDE é considerada por alguns a grande tendência futura da engenharia de software [16], embora atualmente pouco explorada na prática. Programas adaptativos representam uma nova abordagem para resolver problemas. O lado prático do caminho apresentado consiste na utilização de ferramentas CASE para programas adaptativos com possibilidade de gerar código, nas quais os modelos servem não só como documentação, mas também como artefatos básicos de construção dos programas. Tal objetivo foi colocado inicialmente por [15]. Como um passo futuro, funções de mapeamento adicionais podem ser criadas para gerar o programa para distintas plataformas ou linguagens alvo a partir do mesmo modelo, não somente em BADAL

VIII. Referências

[1] OMG, MDA Guide Version 1.0.1, 2003. Disponível em: http://www.omg.org/cgi-bin/doc?omg/03-06-01.

[2] Ambler, S., The Object Primer: Agile Modeling-Driven Development with UML 2.0. New York: Cambridge University Press, 2004.

[3] Silva, S. R. B., Software Adaptativo: Método de Projeto, Representação Gráfica e Implementação de Linguagem de Programação. Dissertação (Mestrado). Escola Politécnica da Universidade de São Paulo, 2010.

[4] Neto, J. J., *Adaptive Rule-Driven Devices - General Formulation and Case Study*. Lecture Notes in Computer Science. Watson, B.W. and Wood, D. (Eds.): Implementation and Application of Automata 6th International Conference, CIAA 2001, Vol. 2494, Pretoria, South Africa, July 23-25, Springer-Verlag, 2001, pp. 234-250.

[5] Hirakawa, A. R., Saraiva, A. M., Cugnasca, C. E., *Adaptive Automata Applied on Automation and Robotics (A4R)*. Latin America Transactions, IEEE, 2007. 5(7), 539-543.

[6] Pelegrini, E. J., Códigos Adaptativos e Linguagem para Programação Adaptativa: Conceitos e Tecnologia. Dissertação (Mestrado). Escola Politécnica da Universidade de São Paulo, 2009.

[7] De Lara, J., Vangheluwe, H., Using AToM as a Meta-CASE Tool. In: Proceedings of the 4th International Conference on Enterprise Information Systems (ICEIS), 2002.

[8] Rutle et al., A formal approach to the specification and transformation of constraints in MDE. The Journal of Logic and Algebraic Programming 81, no. 4, 2012.

[9] Gessenharter, D.; Rauscher, M., Code generation for UML Activity Diagrams. In Modeling Foundations and Applications (pp 205-220), Springer, 2011.

[10] Marchetti, G., Um método de transformação de modelos UML para a inclusão de componentes de frameworks com o uso de planejador lógico. Dissertação (Mestrado), Escola Politécnica da USP, 2012.

[11] Pastor, O.; Molina, J., Model-driven architecture in practice. New York: Springer-Verlag, 2010.

[12] Jouault, F.; Bézivin, J., KM3: A DSL for metamodel specification. In Formal Methods for Open Object-Based Distributed Systems (pp. 205-220), Springer, 2006.

[13] Jackson et al., Automatically reasoning about metamodeling. Software & Systems Modeling, 2013.

[14] Alanen, M.; Porres, I., A metamodeling language supporting subset and union properties. Software & Systems Modeling 7, 2008.

[15] Canovas, S.; Cugnasca, C., Um Metamodelo para Programas Adaptativos Utilizando SBMM. In: Oitavo Workshop de Tecnologias Adaptativas, Escola Politécnica da Universidade de São Paulo, São Paulo, 2014.

[16] Yang, D.; Liu, M.; Wang, S., Object Oriented Methodology Meets MDA Software Paradigm. In: Software Engineering and Service Science (ICSESS), IEEE 3rd International Conference on (pp. 208-211), 2012.

[17] OMG, MOF Model To Text Transformation Language v1.0, 2008. Disponível em: http://www.omg.org/spec/MOFM2T/1.0.
[18] Contier, A.; Padovani, D.; Neto, J. J., Linguístico: Usando Tecnologia Adaptativa para a Construção de Um Reconhecedor Gramatical. In: Oitavo Workshop de Tecnologias Adaptativas, Escola Politécnica da Universidade de São Paulo, São Paulo, 2014.
[19] Silva, A. P.; Silva Filho, R. I.; Rodrigues, F. A.. Swarm Robotics: comportamento Adaptativo Aplicado ao problema de dispersão de pássaros em áreas agrícolas. In: Oitavo Workshop de Tecnologias Adaptativas, Escola Politécnica da Universidade de São Paulo, São Paulo, 2014.

Sergio R. M. Canovas nasceu em São Paulo, Brasil, em 1981. Graduou-se e recebeu o título de mestre em Engenharia Elétrica pela Universidade de São Paulo (USP), São Paulo, Brasil em 2003 e 2006, respectivamente. Em 2011 ingressou no programa de doutoramento em Engenharia Elétrica pela mesma universidade, sendo pesquisador no Laboratório de Automação Agrícola (LAA). Seus interesses de pesquisa incluem redes de controle, sistemas ERP, MDE/MDA, formalismos para metamodelagem e transformação de modelos de software.

Carlos E. Cugnasca graduou-se em Engenharia Elétrica na Escola Politécnica da Universidade de São Paulo (EPUSP), Brasil, em 1980. Na mesma instituição, obteve o seu mestrado (1988), doutorado (1993) e Livre-Docência (2002). Suas principais atividades de pesquisas incluem instrumentação inteligente, automação agrícola e agricultura de precisão. É professor do Departamento de Engenharia de Computação e Sistemas Digitais da EPUSP desde 1988, ocupando atualmente a função de Professor Associado 3. Vem desenvolvendo pesquisas sobre redes de controle, redes de sensores sem fio, eletrônica embarcada e aplicações de computação pervasiva.

Construcción de un Compilador de Asertos de Programación Metódica Utilizando El Método De Autómatas Adaptativos (28 de agosto de 2014)

D. Berolatti y C. Zapata, *Member IEEE*

***Abstract*— The project presented in this article carries on a study of the development about an asset's compiler using the adaptive automata technique. These automatons are designed trying to solve a pair of assets. These assets represent the initial and final states of executing a program. Being the program the solution of the assets, the compiler uses the methodic programing technique to solve the pair of assets. It is possible to execute formal tests to evaluate the program generated by the compiler and compare the result with others using the same method programming technique by hand.**

Keywords* — *compiler, assets, methodic programming technique, adaptive automata.

I. Nomenclatura

- AA: Autómata adaptativo.
- TPM: Técnica de programación metódica.

II. Introducción

Siempre ha existido la necesidad de validar la codificación de un programa. Este proyecto tiene como objetivo la implementación de un compilador que, mediante notaciones matemáticas que especifican un programa, genere las instrucciones de manera automática. El resultado de la compilación tiene como principal característica que sea formalmente correcto. Esto se da debido a la utilización de una metodología llamada derivación de programas la cual garantiza esa característica.

Este trabajo ha sido apoyado por la Pontificia Universidad Católica del Perú a través del curso de Tesis que se imparte en la especialidad de Ingeniería Informática de la Facultad de Ciencias e Ingeniería.

Claudia Zapata imparte docencia en el Departamento de Ingeniería, Sección de Informática de la Pontificia Universidad Católica del Perú, Av. Universitaria 1801, San Miguel, Lima 32, Perú (correo e.: zapata.cmp@pucp.edu.pe).

Diego Berolatti es alumno de pregrado en la Especialidad de Ingeniería Informática de la Facultad de Ciencias e Ingeniería de la Pontificia Universidad Católica del Perú, Av. Universitaria 1801, San Miguel, Lima 32, Perú (correo e.: berolatti.diego@pucp.edu.pe).

La implementación de esta metodología se da mediante la estructura formal de un compilador y la inclusión de un autómata adaptativo capaz de aplicar las reglas de programación metódica.

El proyecto tiene como limitación, no aplicar ninguna regla que implique resolver un problema de complejidad *np*. Debido a esto la expresividad del lenguaje y su capacidad de generación automática se encuentra limitada. El resultado es un compilador capaz de generar código de manera automática en base a las especificaciones que es capaz de compilar. Este proyecto es la base de los compiladores de programación automática.

III. Marco conceptual

Existen metodologías, técnicas y herramientas cuya función es la de corregir errores cometidos durante la codificación de un programa de computadora como lo hecho por Dijkstra [7] y Rosenblum[10]. Sin embargo, éstos no garantizan que el software producido contenga la menor cantidad de imperfecciones posible, excepto aquellos que se basan en las leyes de Charles Richard Hoare [1]. La lógica de Hoare es un sistema formal que proporciona una serie de reglas de inferencia para razonar sobre la corrección de programas imperativos con el rigor de la lógica matemática y su aplicación garantiza que el programa que las utilice cumpla con las especificaciones que este tenga a priori [2].

Para entender estas reglas es importante explicar antes lo que es el triple de Hoare. El triple de Hoare tiene la forma {P} C {Q} y es una estructura formada por dos expresiones matemáticas con un resultado de tipo booleano llamadas asertos (P y Q) y un comando o instrucción (C). El aserto P, llamado precondición, es una expresión que representa el estado del programa antes de ejecutar la instrucción C. Mientras que el aserto Q, llamado postcondición, representa el estado del programa después de ejecutar la instrucción C.

El conjunto de reglas busca en cada momento que la precondición y la postcondición sean verdaderos, en consecuencia el conjunto de instrucciones formadas en C van a cumplir las especificaciones. Esto debido a que las

condiciones iniciales de las especificaciones de C son representadas en la precondición y el resultado de estas en la postcondición. Es por eso que todo programa, que sea capaz de seguir estas reglas o cualquier conjunto de reglas derivadas, es considerado "formalmente correcto". Según [2] estas reglas son tan importantes en el desarrollo de programas como lo son las leyes físicas al momento de construir un edificio o hacer un circuito electrónico.

Una adaptación de estas reglas, llamada programación metódica, permite mediante la inclusión de métodos formales deducir el conjunto de instrucciones C. Esto se logra a partir de definir la precondición y la postcondición en función de las especificaciones deseadas para las instrucciones. Esta forma de construir código a partir de asertos, llamada derivación, conserva las mismas cualidades que la lógica de Hoare debido a que siguen sus reglas. La programación metódica permite entonces construir instrucciones formalmente correctas, siendo su aplicación una forma de garantizar que el software producido tenga una mínima cantidad de errores.

Sin embargo, de acuerdo a [3], debido al corto tiempo del que ahora disponen los programadores para realizar esta tarea y a que el dominio de las nuevas tecnologías se ha convertido en lo más importante durante el desarrollo de un producto de software es que ha existido siempre la necesidad de automatizar técnicas que permiten eliminar errores. Además afirma que la enseñanza en el pregrado de los métodos formales aplicados en la programación metódica ha disminuido su relevancia frente a otros temas.

Por otro lado los compiladores de los lenguajes más usados actualmente no incluyen alguna técnica automatizada que permita corregir errores en los programas [3].

Es entonces que ante esta problemática, se desarrollará un compilador de asertos de programación metódica cuyo fin sea el uso de estos métodos formales de manera automática y cuyo resultado se encuentre en un lenguaje de alto nivel. De esta forma se obtendrá un programa formalmente correcto con menor esfuerzo de parte del programador.

El proyecto utilizará la estructura general de un compilador convencional adaptando cada una de sus partes en función a lo requerido. La primera parte de este proyecto se hizó el análisis de viabilidad y documentación del conjunto de reglas a aplicar. Luego y en función a ellas, se desarrolló un compilador, siguiendo las etapas de análisis léxico, sintáctico y semántico. Y finalmente, se implementó el traductor automático.

IV. Estado Del Arte

A continuación se mencionaran proyectos ya existentes relacionados:

a) Forma computarizada aproximada de resolver el problema
   - The KeY System: Software de verificación de programas, soporta un subconjunto de estructuras del lenguaje Java llamado JavaCard y cuya verificación está basada en lógica dinámica, una generalización de la lógica de Hoare. Utiliza además para la especificación de objetos Uml y Ocl, No es lo suficientemente expresiva como para especificar el comportamiento de un programa. Además la verificación se realiza después de construir el programa.

b) Procedimientos aproximados para resolver el problema:
   - Guarded Command Language : es un lenguaje definido por Esdger Dijkstra que incorpora la lógica de Hoare mediante una estructura llamado "Guarded Command" o instrucciones protegidas. Tienen el mismo funcionamiento que las instrucciones protegidas de las instrucciones alternativas de la programación metódica pero se extienden a cualquier condicional del lenguaje y ofrecen una variante que ayuda a garantizar algoritmos determinísticos. Solo integra (instrucciones protegidas) de manera parcial las reglas de la lógica de Hoare.

   - Diseño por contrato: es una forma de diseñar software, se utiliza la lógica de Hoare en forma de metáfora afirmando que para uno existen obligaciones y beneficios a los cuales uno está ligado mediante un contrato. Mediante esto contrato uno asume roles de proveedor o de cliente buscando en cualquiera de los casos encontrar todas las formas de las cuales las obligaciones y los beneficios de un contrato se garanticen. Se encarga de garantizar el diseño del software pero no garantiza la correctitud formal del programa generado.

   - Asertos embebidos: herramienta desarrollada con el fin de detectar automáticamente errores de ejecución en distintas versiones de un sistema de software. Utiliza asertos los cuales especifican lo que el sistema debería hacer más que él como lo hace.

   - Análisis de programas estáticos: es el análisis de software sin ejecutarlo, se analiza mayormente el código fuente buscando probar que cumpla con los objetivos que propone el software

c) Productos comerciales para resolver aproximadamente el problema:
   - Perfect Developer: Es un software de verificación

de programas que utiliza un lenguaje llamado Perfect, cuyas características incluyen una herramienta para demostrar automáticamente un teorema y un traductor de Perfect a Java, Ada y C++. Sin embargo al ser este un proyecto muy avanzado se requiere de muchos conocimientos previos relacionados a la verificación de programas para poder utilizarlo.

- Prototype Verification System: Es un verificador automático de teoremas que no genera código de programa verificado, sino que prueba automáticamente las propiedades de los algoritmos. Este es muy versátil y necesita de un profundo conocimiento en lógica formal para poder ser utilizado.

d) Productos no comerciales (de investigación) para resolver aproximadamente el problema
   - Fredge Program Prover: [Feinerer, 2005] Software de verificación de programas, también conocido como FPP incluye estructuras de programas imperativos típicos (como el while, if y case). Solo permite enteros como variables y el lenguaje es muy restrictivo a la hora de enunciar las precondiciones y postcondiciones.

Las formas y productos que buscan resolver el problema establecen la teoría de la lógica de Hoare de manera implícita en su solución como metodología o lenguaje. Es decir, solo utilizan los conceptos pero no aplican de manera automática estos. Además ninguno incluye la capacidad de derivación automática de programas a partir de asertos. Sino la verificación de estos a través de las reglas.

Es entonces que solo el especialista de estas metodologías o lenguajes puede aplicar los beneficios de las soluciones fuera de los límites de aplicabilidad de las mismas. En cambio la solución propuesta ofrece un mecanismo que, mediante la ejecución automática de las reglas teóricas de la programación metódica, provee una alternativa capaz de aplicar de manera directa estos conceptos sin tener conocimientos profundos sobre la programación metódica. Desde el momento en que uno aprende a construir asertos matemáticos es suficiente como para desarrollar aplicaciones que tengan todos los beneficios que tienen aplicar la metodología.

La importancia de poder aplicar estas reglas en cuestión es tan importante como tener presente la ley de la gravedad para un ingeniero civil; es muy claro hoy en día que la mayoría de desarrolladores tienen la noción que no existen programas sin errores. Siendo una analogía la de afirmar que siempre una casa se caerá, siempre una pista terminara rajada, siempre se caerá el servidor; esta forma de pensar no es más que una consecuencia de trabajo mal hecho; y que, sin embargo, existen formas de garantizar la calidad de lo que se produce y no solo mediante pruebas. La principal razón por la cual la construcción de software es tan costosa es que la mayoría de involucrados desconocen o ignoran la lógica de Hoare o no ven facilidad al aplicarlas. Es entonces que se propone esta herramienta como un esfuerzo para la difusión de las leyes de la lógica de Hoare en forma de compilador de aplicación de reglas de programación metódica de manera automática.

Este proyecto propone implementación de un compilador convencional especificado en [8] de asertos de programación metódica utilizando el método de autómatas adaptativo

V. Descripción de la solución

Este proyecto es la implementación de un compilador convencional especificado en [8] de asertos de programación metódica utilizando el método de autómatas adaptativos en la representación de las propuestas de derivación aplicadas en la derivación de asertos. Cada una de ellas en función a las condiciones que existen para poder utilizarlas. La evaluación se hizo en función a resultados de los mismos par de asertos aplicando las reglas de la programación metódica a mano.

El proyecto se encuentra dividido en tres fases: la investigación de las reglas de la programación metódica a aplicar en el compilador, el diseño de cada autómata que represente a cada propuesta de derivación a aplicar y la implementación del compilador. Los objetivos establecidos son: establecer las reglas que el compilador va a ejecutar de manera automática para garantizar la aplicación correcta de la programación metódica, implementar el analizador léxico, sintáctico y semántico de la notación matemática utilizada en la programación metódica para satisfacer las especificaciones que tenga cada programa generado por el compilador e implementar un traductor de código intermedio que sea capaz de generar programas en el lenguaje de alto nivel para garantizar la correctitud formal de los programas y su aplicación en el desarrollo de software. Actualmente el proyecto se encuentra finalizado.

A. Sustento de la solución

La finalidad de este proyecto es la de obtener un producto que sirva como herramienta de desarrollo de software el cual sea formalmente correcto. Para que pueda ser usado por cualquier usuario con conocimientos básicos de programación y matemática, sin necesidad de tener conocimientos avanzados de programación metódica. Los programas generados son considerados formalmente correctos y se reduce con esto en gran medida la cantidad de errores que este pueda contener.

B. Resultados obtenidos

A continuación los resultados obtenidos:

- Mapeo de reglas de la programación metódica existentes.
- Conjunto de casos que verifiquen la implementación las reglas definidas en el mapeo.
- Análisis, clasificación y descripción de la aplicación de cada regla de la programación metódica.
- Análisis y diseño de los autómatas adaptativos para las propuestas de derivación seleccionadas.
- Implementación del analizador léxico.

- Implementación del analizador sintáctico.
- Implementación del analizador semántico.
- Implementación del traductor de código intermedio.
- Pruebas de un conjunto de casos que verifiquen la implantación correcta de las reglas en el compilador.

VI. Apoyo teórico

La identificación de distintos pares de asertos hace posible que, de acuerdo al contexto en el que se encuentran, la aplicación de distintas propuestas de derivación tengan un resultado distinto.

Para ello, es necesaria la evaluación del contexto en el que se encuentran los pares de asertos para poder determinar qué propuesta de derivación se usará y, a partir de esto, seleccionar la propuesta adecuada. Debido a que este modelo se usa normalmente para la lectura de lenguaje natural se tuvo que adaptar el algoritmo para cumplir con las necesidades planteadas. Para este proyecto a cada submaquina se le fue asignada una propuesta de derivación. El trabajo de cada submaquina es que, en función a los asertos generen propuestas de derivación en forma de estados. Luego a partir de cada uno de estos estados se generan nuevos asertos (debilitando la postcondición) llegando a una solución parcial. Esto genera por cada estado una lista de parejas de asertos (o duplas) los cuales son registrados en una tabla. Cada una de estas duplas son comparadas, si estas son equivalentes se devuelve el conjunto de estados (instrucciones), en otro caso se vuelve a aplicar el análisis de las submaquinas siempre y cuando esta dupla no haya sido analizada antes. En resumen:

1. Se crea una dupla en función de dos asertos.

2. Se valida la equivalencia de los dos asertos.

3. Si son equivalentes se devuelve la lista de estados involucrados en la transformación de los asertos.

4. Si no son equivalentes y no han sido analizados antes se aplica el análisis de las submaquinas.

5. Se registra la dupla en la lista de duplas ya analizadas.

6. Se recibe cada resultado parcial en forma de lista de estados de cada submaquina.

7. Por cada uno de estos estados y en función de la dupla actual se genera una nueva dupla.

8. Por cada nueva dupla formada que no haya sido analizada se aplica el paso 2.

9. El algoritmo termina si no se cuenta con duplas que no hayan sido analizadas con anterioridad con resultado de error de capacidad de derivación.

VII. Construcción del Compilador

La construcción de un compilador conforma dos partes, la primera parte dedicada al análisis del leguaje base y la segunda parte dedicada a la síntesis de este y la posterior transformación al lenguaje final. La adaptación del autómata adaptativo se encuentra en la segunda parte. De la primera parte:

- Análisis Léxico: La función específica del analizador léxico es la de convertir las líneas de texto de entrada en una lista de tokens. Para lograr esto es necesario profundizar en la estructura general del lenguaje fuente. Esta estructura se encuentra formada por 3 elementos:
  1. (definición de variables no ligadas)
  2. {estructura del aserto correspondiente a la precondición}
  3. {estructura de aserto correspondiente a la postcondición}
  4. …

  La primera parte (1) de la estructura se encuentra formada por dos paréntesis y entre ellos la definición de variables no ligadas separadas por comas. Ejemplo:

  (a,b,c)

  En este ejemplo las letras “a”, “b” y “c” representan las variables no ligadas.
  La segunda (2) parte de la estructura esta formada por una dupla de notaciones matemáticas cada una entre dos llaves. Ejemplo:

  { a= 2*b*c+r AND r< 2*b}
  { a= b*c+r AND r< b}

  La estructura de las notaciones matemáticas es la estructura clásica de formación de notaciones matemáticas a excepción de los símbolos relacionales “∧” y “∨”. Estos símbolos han sido reemplazados debido a la dificultad de escribirlos en un editor de textos convencional.
  La tercera (3) parte se encuentra formada por un número finito de duplas de asertos de la forma definida ya en la segunda (2) parte.

  A continuación el analizador se va a encargar de convertir las líneas de texto de entrada en una secuencia de tokens. Los cuales cada uno corresponde a un elemento de la construcción de asertos formales. Entre los tokens que existen se encuentran: VARIABLE, CONSTANTE, SUMA, RESTA, MULT, DIVI, AND, OR, MAYORIGUAL, MENORIGUAL, MENOR, MAYOR, IGUAL, CIERTO, ENTERO, COMA, LLAVEIZQ, LLAVEDER, PARIZQ, PARDER, CUANT_SUMA,CUANT_MULT,CUAN_MAX,CUANT_MIN,CUANT_TODO,CUANT_ALME.

- Analizador Sintáctico: El siguiente paso en el proceso de compilación es determinar si la secuencia de tokens formada es sintácticamente correcta. Para esto es necesaria la definición de una gramática libre de contexto.
  Esta gramática sirve para establecer las especificaciones necesarias para cumplir con las reglas de formación de asertos de la programación metódica. Para la

realización del analizador sintáctico se va a definir la siguiente gramática libre de contexto.

Terminales:

{AND,OR,"{","}","(",")",MAYORIGUAL,MENORIGUAL, MAYOR,MENOR,IGUAL,SUMA,RESTA,MULT,DIVI,CIERTO , PARIZQ, PARDER,CONSTANTE, ENTERO,VARIABLE,CUANT_SUMA,CUANT_MULT,CUAN_ MAX,CUANT_MIN,CUANT_TODO,CUANT_ALME }

No-Terminales:

{PROGRAM, DEFVAR, DUPLAS, ASERTO, LISTAVAR, ECUACION, RELACION, EXPRESION, COMPA, MODIFIC, VALOR}

Reglas:

(1) PROGRAM → DUPLAS
(2) PROGRAM → DEFVAR DUPLAS
(3) DUPLAS → LLAVEIZQ ASERTO LLAVEDER LLAVEIZQ ASERTO LLAVEDER DUPLAS
(4) DUPLAS → LLAVEIZQ ASERTO LLAVEDER LLAVEIZQ ASERTO LLAVEDER
(5) DEFVAR → PARIZQ LISTAVAR PARDER
(6) LISTAVAR → VARIABLE COMA LISTAVAR
(7) LISTAVAR → VARIABLE
(8) ASERTO → ECUACION RELACION ASERTO
(9) ASERTO → ECUACION
(10) RELACION → AND
(11) RELACION → OR
(12) ECUACION → EXPRESION COMPA ECUACION
(13) ECUACION → EXPRESION
(14) ECUACION → PARIZQ ECUACION PARDER
(15) ECUACION → CIERTO
(16) ECUACION → CUANTIFICADOR
(17) CUANTIFICADOR → VALOR IGUAL CUANT_TRIP
(18) CUANTIFICADOR → VALOR IGUAL CUANT_CUAT
(19) CUANT_TRIP → CUANT_SUMA (VALOR,VALOR,VALOR)
(20) CUANT_TRIP → CUANT_MULT (VALOR,VALOR,VALOR)
(21) CUANT_TRIP → CUANT_CONT (VALOR,VALOR,VALOR)
(22) CUANT_TRIP → CUANT_MAX (VALOR,VALOR,VALOR)
(23) CUANT_TRIP → CUANT_MIN (VALOR,VALOR,VALOR)
(24) CUANT_CUAT → CUANT_TODO (VALOR,VALOR,COMPA,VALOR)
(25) CUANT_CUAT → CUANT_ALME (VALOR,VALOR,COMPA,VALOR)
(26) COMPA → MAYORIGUAL
(27) COMPA → MENORIGUAL
(28) COMPA → MAYOR
(29) COMPA → MENOR
(30) COMPA → IGUAL
(31) EXPRESION → VALOR MODIFIC EXPRESION
(32) EXPRESION → VALOR
(33) EXPRESION → PARIZQ EXPRESION PARDER
(34) MODIFIC → SUMA
(35) MODIFIC → RESTA
(36) MODIFIC → MULT
(37) MODIFIC → DIVI
(38) VALOR → CONSTANTE
(39) VALOR → ENTERO
(40) VALOR → VARIABLE

Símbolo inicial PROGRAM

Para la formación de este analizador en Java se utilizó BYACC y se especificó cada terminal y no terminal para cumplir con las características de este proyecto.

- Analizador semántico: El analizador semántico se encarga de formar el árbol de traducción y la tabla de traducción a partir de la gramática libre de contexto y los tokens definidos anteriormente. Son el paso previo a la traducción en si del código fuente.

Árbol de Traducción: Para formar este árbol se crearon las clases necesarias que soporten esta estructura. Por ejemplo:

- Para los valores se diseñó una clase llamada Valor la cual tiene como atributos el tipo de valor y su valor en forma de variable o constante según sea el caso.
- Para las ecuaciones se diseñó una clase llamada Ecuación la cual está formada por dos clases llamadas Expresion (que forman una expresión) y un enumerador de tipo Comparador que representa el comparador de la ecuación.

VIII. Metodología aplicada

El proyecto de investigación se encuentra guiado por la metodología propuesta en **Error! Reference source not found.** para proyectos de investigación. La adaptación de las líneas generales descritas en **Error! Reference source not found.** para este proyecto dieron como resultado las siguientes etapas:

1. Identificación de las reglas de la programación metódica existentes a utilizar en el proyecto.
2. Análisis y clasificación de la participación de cada regla en cada parte del compilador.
3. Desarrollo de un conjunto de casos de prueba que verifiquen la implementación de las reglas definidas anteriormente.
4. Construcción del autómata adaptativo que implementa las propuestas de derivación.
5. Implementación de los analizadores léxico, sintáctico y semántico del compilador.
6. Implementación del traductor de código intermedio el cual incluye el autómata adaptativo.
7. Validación mediante el conjunto de pruebas de la correcta aplicación de las reglas de la programación metódica.
8. Análisis de resultados.
9. Elaboración de conclusiones.

IX. Diseño e implementación del Autómata Adaptativo

Para identificar las propuestas de derivación del autómata adaptativo se tuvo que investigar cada regla y clasificar en función a su aplicación dentro del compilador siendo la clasificación:

• Reglas para el Lenguaje Fuente y Final: Relacionados al análisis del conjunto de palabras en un lenguaje, a las características de cada palabra en particular y a la funcionalidad de cada palabra en el lenguaje. Estas reglas se aplicaran como características del lenguaje y están basadas en las descritas en [9].

• Reglas para el Compilador:

- Reglas de Aplicación Matemática:

- Despeje Ecuacional: Se utilizaran las reglas relacionadas al despeje ecuacional de una variable en función a las otras, así como el remplazar una variable por otra. Ejemplo:

a+4d-2f/c=K +4a (despejar a en función de d,f,c y k)
a+4d-2f/c -a=K +4a –a (igualdad en los lados)
4d-2f/c=K+3a (simplificación)
4d-2f/c-K=3a (igualdad y simplificación)
(4d-2f/c-K)/3=a (cambio de factor)

- Lógica Matemática: Se utilizaran las reglas relacionadas al análisis lógico de una sentencia lógico-matemática. Ejemplo:

p AND q = p
p AND (p AND q) (reemplazo)
p AND p (simplificación)
p

- Aritmética Matemática: Se utilizaran las reglas de la suma, resta, multiplicación de naturales y enteros.

- Reglas de Aplicación de la Metodología:

  - Aplicación de la Metodología: Son reglas fijas de la metodología para su aplicación. Ejemplo:

"El aserto generado por la derivación de una postcondición ha de ser suficientemente fuerte para garantizar el cumplimiento de esta postcondicion."

  - Propuestas de Derivación: Este tipo de reglas son propuestas que intentan, pero no garantizan, llegar a una solución que lleve a una derivación adecuada. Ejemplo:

"Cuando en la postcondicion tenemos una conjunción de igualdad entre solo dos variables podemos asignar una variable a otra."

Es en este conjunto de reglas en los cuales se va a aplicar el autómata adaptativo. Para poder comprender el funcionamiento de cada propuesta es necesario tener en cuenta algunos conceptos relacionados a la programación metódica:

- Conjunción: Parte de un cuantificador que se encuentra relacionado a un valor booleano.
- Postcondición: Cuantificador que se encarga de explicar la relación con los resultados.
- Variable Ligada: Variable vinculada siempre a un cuantificador.
- Variable Libre y Constantes: Variable inicial o constante del programa.
- Cuantificador: Son formados por variables ligadas y un dominio.
- Precondicion: Cuantificador que se encarga de imponer las condiciones a los datos.
- La instrucción de asignación simple: Corresponde a la modificación de una determinada variable, y por tanto conlleva a una modificación del estado. Se denota con ":=".
- La instrucción de asignación múltiple : Corresponde a la modificación de una dos o más variables, y por tanto conlleva a una modificación del estado. Se denota con "<a,b>:=<c,d>" siendo asignado a igual a b y c a d respectivamente.
- La instrucción alternativa: Corresponde a la ejecución de una instrucción dependiendo de si su protección se encuentra validada.
- Función de cota: También llamada función limitadora se encarga de validar el razonamiento por inducción en los casos de programación recursiva.

A continuación se listan y explican las propuestas que se pueden aplicar en la resolución de problemas relacionados a la programación metódica. Estos no garantizan la resolución del problema pero proponen una alternativa de solución. Estas reglas también son las que se aplicarán directamente en el autómata adaptativo para la toma de decisiones de aplicación de reglas:

1. "Cuando en la postcondición tenemos una conjunción de igualdad entre solo dos variables podemos asignar una variable a otra". Si en la postcondición nos encontramos que una ecuación, dentro de un conjunto de ecuaciones, es la igualdad entre dos variables entonces son propuestas de estado la asignación de una variable a otra. Cualquiera de ellas puede ser la asignada siempre y cuando se cumplan con las reglas de aplicación de la metodología.

   Ejemplo:

   {Post: … AND x=z}

   Siendo propuestas validas z:=x y x:=z

2. "Si en la postcondición existe una única variable ligada, siendo las demás libres o constantes, habremos de efectuar una asignación sobre esta variable en función a todas las variables (variables de la precondicion y la postcondición)". Si en la postcondición dentro de cada ecuación y conjunción existe solo una variable ligada, es entonces la asignación a esta variable ligada una propuesta de estado de derivación.

   Ejemplo:
   {Pre: x+y=T}
   {Post: z+y=T}

   Siendo x una variable que no se encuentra en la postcondición, z una variable libre y T una constante entonces:

y:= E(x,y,z) es una propuesta.

3. "Si en la postcondición tenemos una conjunción de igualdad entre una variable ligada y una constante y esta constante se encuentra asignada a un variable en la precondición entonces se puede efectuar una asignación sobre esta variable con la variable de la precondición la cual tiene a la constante asignada". Si en la postcondición existe dentro de una ecuación una igualdad entre una variable ligada y una constante, es una propuesta la asignación a esa variable del valor de la constante, el cual sea igual al valor de otra variable en la precondición.

Ejemplo:

{Pre: x=X}

{Post: y=X}

Debido a que la variable y es equivalente a la constante X en la postcondición y a que existe en la precondición una variable que es equivalente a esta constante, entonces y:=x seria la propuesta generada por esta regla.

4. "Si la regla 1 no se puede aplicar en ninguna de las conjunciones se puede asignar la variable a una nueva variable ". Si por alguna razón la regla 1 o la regla de igualdad entre dos variables no se puede aplicar o no lleva a ningún resultado, se puede aplicar asignar la variable a una nueva variable creada.

Ejemplo:

{Post: x=y}

Asumiendo que no se puede aplicar la regla uno entonces se propone hacer x:= z siendo z una variable nueva.

5. "Si en la postcondición existe disyunción entre dos conjunciones se sugiere entonces por cada una establecer un conjunto de instrucciones alternativas". Si la postcondición está conformada por disyunciones de conjunciones entonces es una propuesta un conjunto de instrucciones alternativas donde para cada una se le proponga cada disyunción por separado.

Ejemplo:

Si la postcondición es:

{Post: (z = x OR z = y) AND…}

Entonces se propone,

[… →… para el aserto {z = x AND …} y

[]…→… para el aserto {z = y AND …}

6. "Si en la postcondición existen conjunciones y ninguna es de igualdad entonces se puede establecer una asignación múltiple de todas las variables ligadas". Si la postcondición está conformada por conjunciones y ninguna ecuación es de igualdad entonces se propone una asignación múltiple a cada variable ligada.

Ejemplo:

Si a y b son variables iniciales:

{Pre: a= 2*b*c + r AND r< 2*b}

{Post: a= b*c + r AND r< b}

Entonces se puede intentar hacer una asignación múltiple con ambos:

{Pre: a= 2*b*c+r AND r< 2*b}

<c,r>:=<E1,E2>

{Post: a= b*c+r AND r< b}

7. "Si al generar un nuevo aserto este contiene conjunciones que no pueden ser garantizadas por la precondición entonces se puede utilizar la condición (la conjunción que no puede ser garantizada) como protección". Es una forma de establecer las protecciones de las instrucciones alternativas, se establecen como condiciones para la aplicación de la instrucción, lo cual es a su vez la nueva postcondición. La disyunción de todas representan el nuevo aserto generado por el conjunto de instrucciones alternativas.

Ejemplo:

{Pre: a= 2*b*c+r AND r< 2*b}

[r<b → c := 2*c

[] ? → …

{Post: a= b*c+r AND r< b}

Dado que el nuevo aserto de c:= 2*c es r<b este actúa también como protección de la instrucción. Siendo a un vector, x un entero y k un numero natural.

Para seguir con la notación formal descrita en [6], la cual define al autómata adaptativo encargado de la construcción de palabras fonológicas, se va a utilizar la siguiente analogía:

- Una frase es un aserto.
- Una palabra es una ecuación, siendo la frase y el aserto un conjunto de estos.
- Una letra es una variable, constante o relación dentro de una ecuación.

X. Implementación

A continuación se va a definir la implementación de cada uno de los autómatas diseñados para este proyecto. En primer lugar, para la aplicación de las propuestas de derivación se utilizaron tres tipos de autómatas. Para todos estos autómatas el estado de entrada es el a0:

- Como se muestra en la Fig. 1 la configuración del autómata tiene 3 salidas.

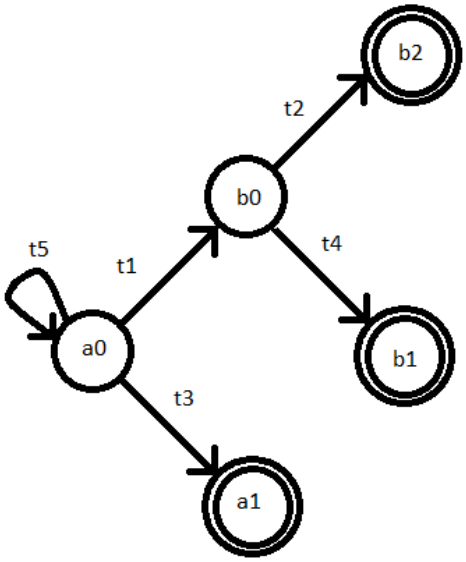


Fig. 1. Configuración del autómata del tipo 1.

Esta se aplica en la regla 1 la cual consta de los siguientes estados. El estado inicial (a0) es el encargado de determinar si existe una conjunción de igualdad entre solo dos variables; si existe se utiliza la transición t1 al estado b0, sino se toma la transición t3 al estado de salida a1. Luego en el estado b0 se verifica si ambas variables son ligadas, de serlas se va al estado final b2 mediante la transición t2. Si solo una es ligada entonces se va al estado b1 por el estado t5. Debido a que este análisis se hace por cada ecuación, la transición t5 representa la transición de análisis entre ecuaciones de un aserto.

- Como se muestra en la Fig. 2 es una configuración de análisis simple

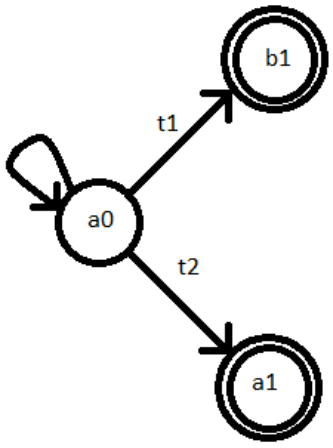


Fig. 2. Configuración del autómata del tipo 2.

Esta configuración se utiliza para aplicar las propuestas de derivación 2,4 y 6 debido a que estas comparten la misma naturaleza. El estado inicial a0 se encarga de validar que se cumpla su condición la cual involucra todo el aserto. De no cumplirlo se va por la transición t2 al estado a1, en otro caso se traslada por la transición t1 al estado b1.

- Como se muestra en la Fig. 3 es una configuración de análisis doble.

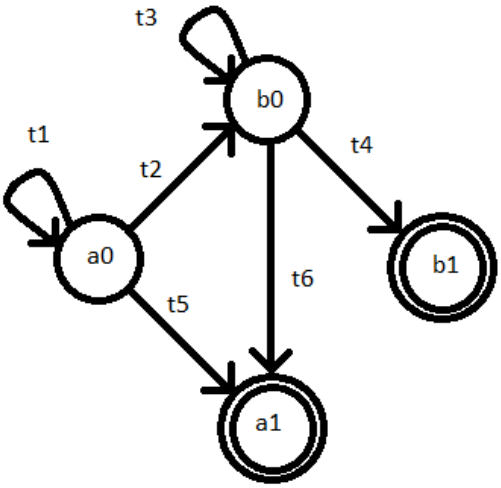
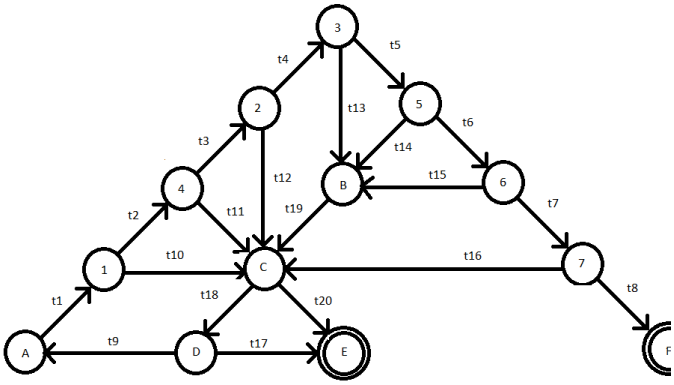


Fig. 3. Configuración del autómata del tipo 3.

Esta configuración se utiliza para aplicar las propuestas de derivación 3,5 y 7 debido a que estas comparten la misma naturaleza. El estado inicial a0 se encarga de validar que se cumpla con la existencia de una característica del conjunto de ecuaciones. Cada una de ellas sin excepción debe cumplirla, la transición t1



representa entonces el análisis de cada uno de las ecuaciones. De no cumplirlo se va por la transición t5 al estado a1, en otro caso se traslada por la transición t2 al estado b0. Este estado valida que se cumpla con una condición en el aserto inicial o precondición. La transición t3 representa e análisis de cada uno de las ecuaciones de la precondición. Si la condición se cumple entonces se procede al estado b1 por la transición t4; sino se va al estado a1 por la transición t6.

Además para el funcionamiento de derivación se utilizó un autómata. El cual se encarga de la aplicación de las reglas de derivación. La configuración de este autómata es como se muestra en la Fig. 4.:

Fig. 3. Configuración del autómata de derivación.

El estado A, representa el estado inicial en el cual nos encontramos cuando tenemos un par de asertos los cuales no son equivalentes. Para los autómatas que representan las propuestas de derivación, los estados finales del tipo a representan a la no aplicabilidad de la propuesta y los estados de tipo b a su aplicabilidad. A continuación el estado A aplica el autómata de la regla 1(representado por el estado 1) por la transición t1. Si el resultado de esta propuesta no es aplicable se procede a evaluar a la siguiente regla siguiendo un orden específico. El orden de evaluación de las reglas permite dar prioridad a determinadas propuestas.

Si no se puede aplicar ninguna regla o el par de asertos ya ha sido validado se procede al estado final F. Este estado representa la posibilidad de un resultado nulo, es decir, que excede la capacidad del compilador. Para las reglas 5,6 y 7 se tiene que hacer ejecutar el estado B para generar el conjunto de adaptaciones adecuadas para poder ir al estado C. Si el resultado es positivo para los autómatas se procede al estado C. En el estado C se convierten estas propuestas en un nuevo par de asertos. Si el nuevo par es equivalente se va al estado E dando por terminado el autómata, siendo el resultado, las instrucciones generadas por el conjunto de propuestas aplicadas para la transformación de asertos.

XI. Conclusiones

Durante el desarrollo del proyecto se encontraron diversas dificultades relacionadas a la aplicación automática de procesos que para el ser humano con entrenamiento adecuado puede aplicar con facilidad pero que son difíciles de aplicar para un computador; en la mayoría de estos casos se optó por diseñar un algoritmo capaz de aplicar estos procesos, sin embargo, la limitación de estos algoritmos reducen el alcance del proyecto.

Durante el análisis y síntesis de las reglas a aplicar por el compilador se tuvieron que descartar propuestas de derivación debido a que su aplicación requería de algoritmos especializados como la generación de programas recursivos, aplicaciones en función a una estructura abstracta de datos específica y algoritmos de transformación por inducción. En otros casos las reglas no podían ser generalizadas y eran aplicadas en función a sugerencias arbitrarias del autor. Sin embargo se incluyeron suficientes reglas como para satisfacer los objetivos del proyecto y el alcance del mismo, debido a que se incluyeron todas las reglas relacionadas a la mecánica de aplicación de propuestas de derivación. Es decir, se incluyeron todas las reglas que forman la base de la aplicación de esta metodología.

Para el análisis léxico, semántico y sintáctico se diseñó un lenguaje capaz de expresar los asertos que el compilador puede resolver. Se incluyeron variables, enteros, constantes, ecuaciones y cuantificadores. Estos últimos son los únicos capaces de representar un vector y su transformación se hace de manera automática. Era necesario incluir los vectores para enriquecer el lenguaje, pero debido a que las propuestas de derivación relacionadas a los vectores implicaban en su gran mayoría la inclusión de algoritmos np se opto por derivar de manera automática cada cuantificador sin utilizar las propuestas no implementadas. De esta manera se logró incluir la representación de vectores en el lenguaje cumpliendo con la expresividad necesaria para satisfacer los objetivos del lenguaje.

En la implementación del traductor de código intermedio se aplicaron y adaptaron los conceptos relacionados a un autómata adaptativo. El algoritmo no solo es capaz de utilizar las propuestas de derivación de programas con éxito, sino que también soporta la inclusión de nuevas reglas ya que la formación de cada submaquina es independiente del funcionamiento del algoritmo. La expresividad del lenguaje, en algunos casos, excede la capacidad de traducción del compilador sin embargo, en estos casos el compilador no resuelve dar ningún resultado a dar uno erróneo. As así que el traductor es capaz de generar instrucciones en un lenguaje de alto nivel, y garantizar la correctitud formal de los programas que pueden ser generados por el compilador.

De esta manera se logró cumplir con el objetivo del proyecto estableciendo además la base para la derivación automática de programas y la construcción de un compilador de programación automática. Los programas obtenidos por este compilador son regidos por las reglas de programación metódica y al cumplir estas son formalmente correctos lográndose cumplir con el objetivo. Este proyecto es la base de muchos proyectos que se pueden generar en pro de la programación automática.

XII. TRABAJOS FUTUROS

Existen varios trabajos futuros que se pueden generar de este proyecto:

- Debido a que el resultado del traductor intermedio es una representación de instrucciones universal es posible traducir el lenguaje a cualquier lenguaje de alto nivel. Para la traducción solo sería necesaria la adaptación adecuada de las estructuras

esenciales de todo lenguaje como son las variables, constantes, asignación de vectores, las condicionales y los bucles.

- El aumentar en el lenguaje la capacidad de incluir especificaciones de algebraicas de tipos abstractos de datos se podría incluir más reglas relacionadas a estas estructuras.

- Se puede implementar un algoritmo de deducción por inducción e incluir más reglas de programación metódica.

- Si se implementa un mecanismo de aprendizaje se podría entrenar al compilador para resolver los problemas cuya solución tienen un origen arbitrario o son tomados a partir de juicio experto.

- Durante la presentación del conjunto de estados que forman parte de la solución de cada derivación se tiene como información la regla aplicada por cada estado. Es decir que si se deseara aplicar este proyecto como herramienta de aprendizaje de la metodología, es posible adaptar el compilador para que este explique paso a paso la derivación de cada par de asertos.

REFERENCIAS

[1] C. Hoare "An Axiomatic Basis For Computer Programming, New York, Communications of the ACM", *Communications of the ACM*, vol. 12, n° 19, pp. 576-580, 1969.

[2] J.L. Balcázar "Programación Metódica", Madrid, McGraw-Hill/Interamericana de España S. A, 1993.

[3] I. Feinerer "Formal Aspects of Computing, Formal Program Verification: a Comparison of Selected Tools and Their Theoretical Foundations" , Viena 2005, vol 21 pp 293-301, 2005.

[4] Project Management Institute "Guia de los Fundamentos para la Dirección de Proyectos (Guia de PMBOK)", 4ta Edición, Pennsylvania.

[5] J.J. Neto, "Adaptive Automata for Context-Sensitive Languages", *ACM Sigplan Notices*, vol. 29, n° 9, pp. 115-124, 1994.

[6] H. Pistori, "Tecnologia adaptativa en Engenharia de computação: estado da arte e aplicações", Dissertação de doutorado, orientada por J. J. Neto, Departamento de Engenharia de Comparação e Sistemas Digitais, Escola Politecnica da Universidad de Sao Paulo, 2003.

[7] Esdger Dijkstra, "Applying Design By Contract", USA, IEEE-Computer, USA, vol 25, n° 10, pp40-51, 1992.

[8] Ravi Sethi, "Programming Languages Concepts & Constructs" , New Jersey, Pearson, 2001.

[9] Anne Kaldewaij, Programming: The Derivation of Algorithms, Inglaterra, Prentice Hall International, 1990.

[10] David Rosemblum, "IEEE Transactions on Software Engineering, "A Practical Aproach to Programming With Assertions", New Jersey, 1995, vol 21 -1 ,pp19-31, 1990.

Adaptalker: Um Framework para o Gerenciamento Adaptativo do Diálogo

D.A. Alfenas, C. E. Bogik, D. P. Shibata, R. P. da Silva, M. R. Pereira-Barretto

***Abstract*— Dialog management is the central activity of a spoken dialog system, whose responsibility is choosing the communicative actionssent to the system user. This paper presents a formal model for such activity that uses adaptive technology. This model is being used to drive the development of a spoken dialog system framework.**

***Keywords*—spoken dialog systems,dialog management, adaptive technology.**

I. Introdução

Os sistemas de diálogo falado são projetados para o uso da voz como principal canal de interação com o usuário através de linguagem natural. Na maioria desses sistemas, o gerenciador de diálogo é o componente central na arquitetura [1]. Este trabalho apresenta um modelo para o gerenciamento adaptativo do diálogo, primeiramente descrito em [1], e também uma plataforma para construção de gerenciadores, ainda em desenvolvimento, chamada de Adaptalker. Tanto o modelo quanto a plataforma são evoluções de trabalhos apresentados no VI Workshop de Tecnologia Adaptativa [2] e no VII Workshop de Tecnologia Adaptativa [3].

A adaptatividade é um termo que se refere à capacidade de um sistema de tomar a decisão de modificar seu próprio comportamento, em resposta ao seu histórico de operação e aos dados de entradasem a interferência de qualquer agente externo[4]. A utilização de tecnologia adaptativa foi motivadapela sua capacidade de transformar dispositivos livres de contexto em dispositivos sensíveis ao contexto com poucas modificações estruturais. Esta capacidade foi utilizada para aperfeiçoar uma técnica existente e comercialmente estabelecida, o gerenciamento de diálogo baseado em estados finitos, de forma a eliminar ou reduzir as deficiências de tal tecnologia, já discutidas por outros autores [5][6].

O restante deste artigo está estruturado da seguinte maneira: o tópico II apresenta uma revisão sucinta da literatura de gerenciamento de diálogo falado. No tópico III é apresentado, em alto nível, a proposta de uma arquitetura completa para sistema de diálogo falado, no qual a plataforma proposta de gerenciamento está inserida. O tópico IV detalha o modelo de adaptativo de gerenciamento, com foco na utilização da tecnologia adaptativa. O tópico V apresenta a plataforma de desenvolvimento.O último capítulo contém a conclusão sobre o trabalho.

D. A. Alfenas, Fundação para o Desenvolvimento da Engenharia (FDTE), São Paulo, Brasil, d.alfenas@fdte.org.br.

C. E. Bogik, Fundação para o Desenvolvimento da Engenharia (FDTE), São Paulo, Brasil, carlos.bogik@fdte.org.br

D. P. Shibata, Fundação para o Desenvolvimento da Engenharia (FDTE), São Paulo, Brasil, d.shibata@fdte.org.br

R. P. da Silva, Fundação para o Desenvolvimento da Engenharia (FDTE), São Paulo, Brasil, raphael.pinheiro@fdte.org.br

M. R. Pereira-Barretto, Escola Politécnica da USP (EPUSP), São Paulo, Brasil, marcos.barretto@poli.usp.br.

II. Gerenciamento de diálogo e os sistemas de diálogo falado

A. Sobre o diálogo

Os sistemas computacionais de diálogo são baseados no sistema de tomada de turnos para a conversação proposta porSacks, Schegloff e Jefferson [7]. Este sistema determina, entre várias outras regras,que (i) a pessoa que fala muda ao menos uma vez, (ii) que as pessoas falam uma de cada vez, (iii) sobreposições de fala acontecem, mas tendem a ser curtas, (iv) há pouca latência entre os turnos e (v)mecanismos de reparo existem para lidar com erros na tomada de turnos. Os turnos são compostos por unidades menores, chamadas de unidades de construção de turno. Estas unidades podem ser de vários tipos, tais como sílabas, palavras e frases.

A unidade de construção de turno mais comum em sistemas computacionais é o *segmento funcional*. Um segmento funcional pode ser entendido como o menor trecho do turno que tenha uma função comunicativa.Esta últimapode ser definida como o tipo da intenção comunicativa expressa no segmento funcional. A pergunta, a pergunta de verificação (uma especialização da função pergunta), a oferta,o cumprimento e o*feedback* são exemplos de funções comunicativas. Um segmento funcional pode ter mais de uma função comunicativa. Neste caso, diz-se que o segmento tem dois ou mais *atos dialogais*, cada um em uma diferente *dimensão comunicativa* [8]. Um ato dialogal pode ser entendido como uma operação de atualização de estado da informação em cada um dos participantes do diálogo. Ele é definido por uma função comunicativa, um ou mais *itens semânticos* e uma ou mais relações funcionais com outros atos dialogais.Atos dialogais, segmentos funcionais e dimensões comunicativas fazem parte da norma ISO24617-2 para anotação de diálogos [9]. Esta norma define também um conjunto padrão de funções comunicativas.

O seguinte trecho de diálogo descreve um segmento com dois atos dialogais na ação do sistema:

```
Usuário: A que horas sai o próximo trem
para Jundiaí?
Sistema: O próximo trem para Jundiaí
parte de São Paulo 19:35h.
```

Nele, pode-se observar que o turno do sistema tem um ato dialogal de função comunicativa *resposta* (de propósito geral, sem dimensão específica), com item semântico igual ao horário de partida do trem (19:35h) e relação funcional com o ato dialogal de função *pergunta* do turno do usuário. Ele também tem um ato de função*feedback*(específico da dimensão *feedback*), pois repete o conteúdo da pergunta (“o próximo trem para Jundiaí”).

Outro conceito importante do diálogo é o*common ground*, conjunto de informações que um participante supõe que seu interlocutor saiba e que são básicas para o bom relacionamento e funcionamento da conversa[10][11]. *Grounding* é o nome dado ao processo interativo pelo qual o common ground é construído e mantido entre dois ou mais indivíduos[12]. O feedback é o mecanismo do diálogo utilizado para sinalizar o *grounding*.

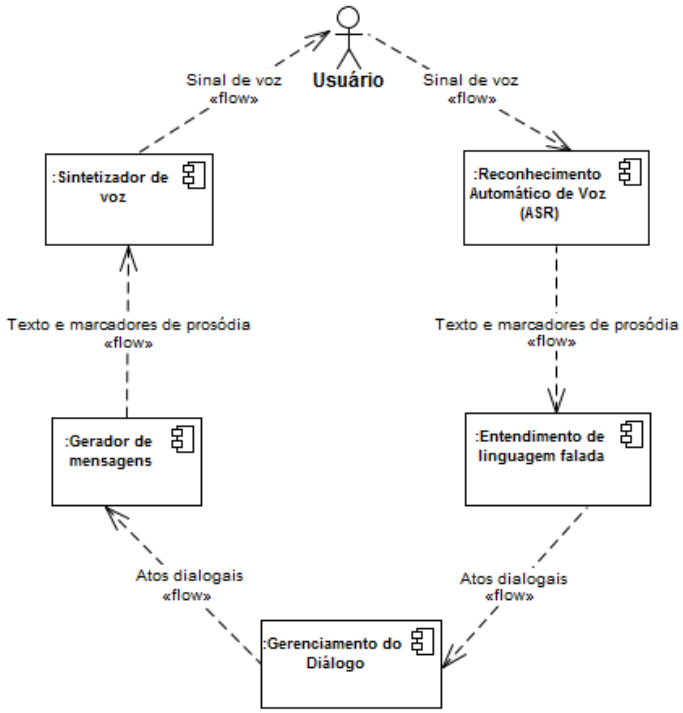


Figura 1: Arquitetura típica de um Sistema de Diálogo Falado

B. *Sobre sistemas computacionais de diálogo falado*

A Figura 1 mostra a arquitetura típica de um sistema de dialogo falado. O reconhecedor de voz é o componente responsável por transformar os sinais de voz em texto e outras anotações relacionadas à prosódia. Entretanto, a prosódia e a parte verbal podem ser tratadas separadamente, como na arquitetura proposta em [2]. O entendimento de linguagem falada é o responsável por geraros segmentos funcionais, atos dialogais e itens semânticos a partir do texto anotado. O gerenciamento de diálogo é um processo que envolve, tipicamente, duas tarefas: uma estimativa do estado corrente do diálogo e a tomada de decisão da próxima ação comunicativa do sistema. Alguns trabalhos deixam explícita a separação entre essas duas tarefas, como em [13]. A estimativa do estado corrente do sistema envolve observar a última ação comunicativa do usuário e relacioná-la com as ações anteriores do sistema e do usuário. O gerador de mensagens transforma a saída do gerenciador de diálogo, formada por atos dialogais ou outros formatos estruturados, em texto em linguagem natural e anotações de prosódia que são, finalmente, transformados em voz pelo sintetizador de voz.

Existem outros componentes, comuns a todo o ciclo de processamento, relacionados à fonte de conhecimento e de dados. Podemos citar, por exemplo, a base de dados léxica (com substantivos, adjetivos, sinônimos, etc.) e a memória semântica. Esta últimacontém as*crenças*do sistema sobre o mundo, incluindo informações sobre o usuário, como identificação e objetivos.

Sistemas *multimodais* possuem módulos específicos para cuidar de outros canais de comunicação que não a voz, tais como expressões faciais e gestos manuais. Eles possuem, tipicamente, um módulo adicional responsável pela *fusão* das informações fornecidas pelos diversos canais de entrada, de forma que eles fiquem devidamente sincronizados. Este tipo de arquitetura é discutido em [14].

Devido a taxas de erros significativas tanto no módulo de reconhecimento quanto no de entendimento de linguagem falada[15], esses módulos não geram, usualmente, saídas únicas sobre o turno do usuário: eles geram um conjunto de hipóteses. Cada hipótese possui uma representação da ação do usuário e um grau de confiança associado. Um gerenciador de diálogo se torna mais robusto ao considerar essas múltiplas hipóteses no processo de tomada de decisão [16].

O sistema de diálogo pode ser classificado quanto à sua capacidade de suportar o processamento *incremental*, isto é, sua capacidade de processar o turno do diálogo enquanto este ainda está sendo falado, ao invés de processar apenas o turno do usuário de uma única vez. A maioria dos sistemas de diálogo pesquisada não são incrementais. Exemplos de arquitetura incrementais podem ser encontrados em [17] e [18].

C. *Sobre as técnicas de gerenciamento de diálogo*

Vários textos fazem um revisão ampla das técnicas de gerenciamento de diálogo e as classificam em grupos, tais como [5], [6] e [19].A técnica em que o Adaptalker se baseia é o gerenciamento baseado em estados finitos. Elafoi e talvez ainda sejaa mais utilizada em sistemas comerciais. Neles, o usuário é levado a uma sequência de passos pré-definidos. O fluxo é especificado como um conjunto de estados ligados por transições denotando os caminhos alternativos conforme as respostas do usuário às solicitações feitas pelo sistema.A maior vantagem deste método é que o vocabulário e gramática podem ser especificados previamente, tornando mais simples o desenvolvimento, treino e teste do sistema. O maior problema é que ele dificulta a modelagem de situações em que o usuário obtém a iniciativa, isto é, situações em que o usuário faz as perguntas ou introduz um assunto. Outro problema é a impossibilidade de construir fluxos dependentes de contexto; por exemplo, não há maneira imediata de o usuárioreparar uma informação dada vários turnos antes, requerendo que ele solicite ao sistema retornar ao passo anterior por várias vezes até atingir o ponto em que a informação errada foi dada,

exceto se houver replicação a cada estado da transição que permite a correção da informação.

Outras técnicas bastante utilizadas incluemgerenciadores baseados em *frames* ou *templates*, baseados em processo de decisão de Markov ou emprocesso de decisão de Markov parcialmente observável [13], baseados em planejamento [20] com ou sem*agendas* de execução. Um exemplo de gerenciamento baseado em agenda é o Ravenclaw [21].

III. A Arquitetura do Sistema do Adaptalker

O sistema em que o framework Adaptalker é utilizadofoi projetado para que os componentes tenham acoplamento mínimo entre si. Foi utilizado o padrão *blackboard*[22] para integração entre os componentes, de forma que eles nunca precisem se conhecer.

Outra característica desta arquitetura é o suporte a um modelo incremental, de uma forma muito similar às arquiteturas propostas em [17] e [18]. Buffers são utilizados para comunicação de eventos de atualização, compromisso e revogação de hipóteses e eventos para sinalizar fim e início de ação.Esses buffers podem trafegar hipóteses de cada um dos tipos de dados suportados.

Para preservar estas características também no gerenciador de diálogo, é necessário definir os tipos de dados de entrada e saída.

A. Entradas

Existem três tipos de ações que resultam em entradas no gerenciador de diálogo: falas, gestos e ações não comunicativas.

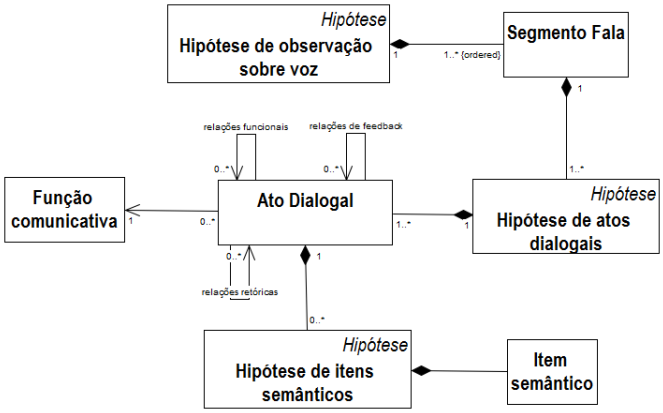


Figura 2: Modelo de entrada do gerenciador de diálogo para voz

Apenas o modelo para falas foi detalhado e implementado. Este modelo é baseado na norma ISO 24617-2 e é exibido na figura 2. Existe uma proposta para modelo de gestos feita em[1], mas cuja validade ainda não foi demonstrada. Eventos não comunicativos podem usar a mesma estrutura da figura 2, utilizando funções comunicativas distintas daquelas dos eventos comunicativos.

O conjunto de funções comunicativas suportadas deve ser o mesmo para todos os módulos que as utilizam. A função comunicativa deve especificar quais são os itens semânticos e relacionamentos opcionais e obrigatóriosassociados aos atos dialogais. O formato dos itens semânticos também deve ser o mesmo. Pode-se utilizar um modelo bastante restrito para representar os itens semânticos, como pares variáveis-valor(como em [15] ou no Segundo Desafio de Rastreamento de Estado [23]) ou utilizar uma representação mais flexível, através de alguma linguagem descritiva (como em [1]).

B. Saídas

A principal saída do gerenciador é a próxima ação a ser executada pelo sistema. Tal ação é composta de um ou mais atos dialogais. Se for utilizado um modelo nãoincremental, a próxima ação é gerada uma única vez. Se incremental, o gerenciador pode adicionar atos dialogais à saída aos poucos, gerando eventos de atualização de hipótese.

Outra saída é a lista de expectativas sobre a próxima ação do usuário, que também é composta de atos dialogais. Essa lista pode ser utilizada pelos módulos de reconhecimento de voz e entendimento de linguagem falada para refinar as hipóteses sobre o próximo turno do usuário.

IV. Gerenciamento Adaptativo de Diálogo

O modelo de gerenciamento adaptativo do Adaptalker é descrito detalhadamente em [1].O modelo é formado por duas camadas: o *filtro* e o *dispositivo adaptativo de tomada de decisão* (doravante referenciado como DATD).

O*filtro* é a camada superior, não adaptativa. A sua principal responsabilidade é decidir quais eventos devem ser descartados. Seu modelo atual é não incremental e, portanto, apenas eventos de nova ação e de fim de ação são considerados. Outra responsabilidade dele é manipular múltiplas hipóteses.Para cada hipótese com grau de confiança maior que um parâmetro c_{min}, o filtro faz uma cópia do DATD, para quem passa a hipótese a ser processada. Quando todos os DATD finalizam o processamento de suas hipóteses, é calculado um grau de confiança a posteriori c_p para cada um, multiplicando o grau de confiança na última ação do usuário pelo grau de confiança da próxima ação do sistema. A saída associada ao maior c_p é considerada e os demais DATD são descartados. Se não é possível escolherapenas uma hipótese, um procedimento de tratamento de erro é disparado.

O DATD é um dispositivo adaptativo que organiza as regras de tomada de decisão em submáquinas:

$$DATD = (M, O, \Psi_s, M_a, M_0) \quad (1)$$

Na fórmula (1), M é o conjunto de todas as submáquinas de DATD; O é o alfabeto de entrada, isto é, o conjunto de todos os atos dialogais que podem ser observados a partir de uma ação do usuário; Ψ_s é o alfabeto de saída, isto é, o conjunto de todos os atos dialogais que podem ser utilizados como saída pelo sistema; M_aé uma lista ordenada de submáquinas ativas. Cada vez que uma nova máquina é ativada, ela entra na posição inicial da lista. Quando a máquina é inativada, ela é removida da lista. Se M_a ficar vazia, não será possível a continuidade do diálogo utilizando o dispositivo; M_0 é a configuração inicial de M_a ao início de um diálogo.

Cada submáquina $\mu \in M$ é definida por uma sétupla:

$$\mu = (Q, B, T, F, X, q_0, Q_F) \quad (2)$$

em que Q é o conjunto de estados de μ;B é o conjunto de estados de *binding* - estados especiais utilizados para o vínculo de atos do usuário com as expectativas;T é o conjunto de transições de estado de μ;F é o conjunto de funções adaptativas que podem ser utilizadas em μ;X é o conjunto de variáveis que podem ser utilizadas em μ para armazenar resultados de consultas entre as transições;q_0é o estado inicial de μ, $q_0 \notin B$; Q_f é o conjunto de estados de aceitação de μ.

Cada transição pertencente a T é definida por uma óctupla

$$\gamma_{i,\pi} = (q_i, q_j, \pi, c_\mu, \alpha_a, \alpha_p, \beta, foco) \quad (3)$$

em que:

- q_i é o estado inicial da transição;
- q_j é o estado final da transição. Se $q_i \in B$, então $q_j \notin B$;
- π é a condição para que a transição seja disparada, utilizando comparadores lógicos, identificadores de variáveis $x \in X$ e literais. Se $q_i \in B$, então π também pode ser definida através de filtros de atos dialogais. A transição pode ter uma condição vazia;
- c_π é a expectativa que o gerenciador de diálogos tem de que a condição π será satisfeita. Sua definição é opcional e permitida apenas se $q_i \in B$;
- α_aé uma ação adaptativa executada antes da transição;
- α_pé uma ação adaptativa executada após a transição;
- β é a tarefa que será executada ao final da transição;
- *foco*: diz quem receberá o foco de execução das regras após a execução da tarefa β, dentre o usuário, a submáquina atual, outra submáquina ou neutro.

Uma tarefa pode ser classificada em um de quatro tipos: requisição ao usuário, informação ao usuário, chamada de submáquina e tarefa de sistema. α_ae $\alpha_p$são utilizadas para chamar funções adaptativas. Uma função adaptativa é definida em termos de parâmetros, variáveis, geradores e ações elementares, de forma similar ao mecanismo utilizado para autômatos adaptativos[24][25]e detalhado para o DATD em [1]. Podem ser utilizadas ações elementares de inserção, remoção, consulta, alteração, comparação entre duas variáveis e chamada de ação adaptativa não elementar, sendo queas três últimas podem ser definidas a partir das três primeiras [26].

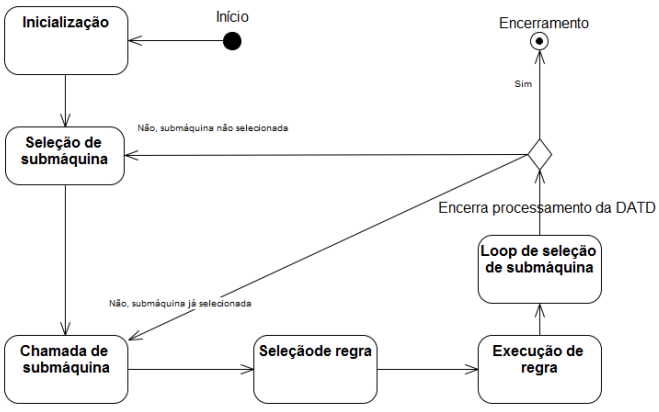


Figura 3 – Ciclo de vida de um DATD

O ciclo de vida de um DATD pode ser visto na figura 3. A inicialização é executada uma única vez, no início do diálogo.Nela, as submáquinas em M_0 são colocadas em M_a, em suas configurações iniciais. A cada turno do usuário, um DATD recebe uma lista de entrada ζ contendo atos dialogais ordenados representando uma única hipótese. Esta lista é utilizada para selecionar em qual das submáquinas ativas será a próxima execução de regra. Cada submáquina deve indicar o quão distante os atos $\varphi_i \in \zeta$estão de suas expectativas atuais. A submáquina com expectativas mais próxima é selecionada para execução.

O loop de seleção inativa submáquinas que chegaram a um estado de aceite e decide quando o dispositivo deve parar de processar e esperar o próximo turno do usuário. A inativação é importante para reduzir o espaço de procura por regras.A parada de processamento é feita em duas situações: (i) quando ζestiver vazia e não há requisições pendentes referentes ao turno anterior do sistema e (ii) quando não houver submáquinas ativas (neste caso o encerramento é definitivo). No caso da situação (i), se existirem atos dialogais de saída Φ esperando execução, o processamento é parado para processarΦ e obter a próxima ação do usuário. Se Φ está vazio, então existe um erro que causa o fim do DATD, pois nem o usuário nem o sistema estão com a inciativa – este erro acontecerá somente se houver algum erro de projeto das submáquinas.

V. O Framework do Adaptalker

Em [1], foi feita uma implementação em linguagem Java para uma aplicação de venda de pizzas por telefone, demonstrandoo funcionamento do Adaptalker, utilizando-se OWL[27] para descrição dos itens semânticos e das ontologias de memória semântica e o OWL API [28] e o *reasoner* Hermit [29] para manipulação das ontologias. Esta implementação criou seis submáquinas. Delas, destaca-se a submáquina para gerenciamento de erros, cujas regras poderiam ser reutilizadas em outras aplicações. Ela trata, por exemplo, situações em que nenhuma regra foi encontrada para algum ζ e quando não foi possível entender a ação do usuário de forma total ou parcial.

Os trabalhos atuais consistem em reorganizar ou mesmo refazer partes do código de tal forma que novas aplicações possam ser desenvolvidas facilmente utilizando o Adaptalker a partir de uma biblioteca Java, requerendo apenas aimplementação de interfaces para tratamento dos atos dialogais específicos e a produção de regras de tomada de decisão do DATD também específicos da aplicação final.

Outro trabalho importante em andamento se refere à construção dos outros módulos. O Adaptalker foi testado apenas isoladamente, com entradas produzidas manualmente por um operador. Um problema que fica mais evidente apenas com todos os módulos integrados é o tratamento dos atos dialogais e da memória semântica, pois eles não são utilizados apenas no gerenciador, mas também nos módulos de entendimento e de geração.

VI. Conclusão

O Adaptalker se mostrou adequado para o tratamento de gerenciamento de diálogo e o trabalho em andamento no framework se mostra promissor.A adaptatividade permitiu

aperfeiçoar o modelo original, como demonstrado em [1], sem remover as vantagens que um gerenciador baseado em estados finitos possui: a facilidade de desenvolvimento e de testes. É possível descrever um DATD sem ações adaptativas, da mesma forma que no dispositivo original, adicionando adaptatividade para adicionar expressividade, facilitando o projeto de regras de reparo, de iniciativa mista e de situações com dependência de contexto.

REFERÊNCIAS

[1] D. A. Alfenas, "Aplicações da tecnologia adaptativa no gerenciamento de diálogo falado em sistemas computacionais," dissertação de mestrado, Escola Politécnica da USP, 2014.

[2] D. A. Alfenas and M. R. Pereira-Barretto, "Adaptatividade em Robôs Sociáveis: uma Proposta de um Gerenciador de Diálogos," in *Memórias do WTA 2012 Sexto Workshop de Tecnologia Adaptativa*, 2012.

[3] D. A. Alfenas, M. R. Pereira-Barretto, and R. dos S. Paixão, "Sobre o uso de formalismos adaptativos no gerenciamento de diálogos em robôs sociáveis," in *MEMÓRIAS DO WTA 2013 SÉTIMO WORKSHOP DE TECNOLOGIA ADAPTATIVA*, 2013.

[4] J. José Neto, "Um Levantamento da Evolução da Adaptatividade e da Tecnologia Adaptativa," *IEEE Lat. Am. Trans.*, vol. 5, no. 7, 2007.

[5] M. F. McTear, "Spoken Dialogue Technology : Enabling the Conversational User Interface," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 90–169, 2002.

[6] T. H. Bui, "Multimodal Dialogue Management - State of the art," Centre for Telematics and Information Technology, University of Twente, Enschede, Holanda, 2006.

[7] H. Sacks, E. A. Schegloff, and G. Jefferson, "A Simplest Systematics for the Organization of Turn-Taking for Conversation," *Language (Baltim).*, vol. 50, no. 4, pp. 696–735, 1974.

[8] H. Bunt, "Dimensions in Dialogue Act Annotation," in *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, 2006, pp. 919–924.

[9] H. Bunt, J. Alexandersson, J. Carletta, J. Choe, A. C. Fang, K. Hasida, K. Lee, V. Petukhova, A. Popescu-belis, L. Romary, C. Soria, and D. Traum, "Towards an ISO standard for dialogue act annotation," in *Proceedings of LREC 2010, the Seventh International Conference on Language Resources and Evaluation*, 2010, pp. 2548–2555.

[10] M. Baker, T. Hansen, R. Joiner, and D. Traum, "The role of grounding in collaborative learning tasks," in *Collaborative learning: Cognitive and computational approaches*, Elsevier Science, 1999, pp. 31–63.

[11] Tomasello, "Human Cooperative Comunication," in *Origins of Human Communication*, Cambridge, MA: MIT Press, 2008, pp. 57–108.

[12] H. H. Clark and S. E. Brennan, "Grounding in communication," in *Perspectives on socially shared cognition*, L. B. Resnick, J. M. Levine, and S. D. Teasley, Eds. Washington DC: American Psychological Association, 1991, pp. 127–149.

[13] B. S. Young, M. Gasic, B. Thomson, and J. D. Williams, "POMDP-Based Statistical Spoken Dialog Systems : A Review," *Proc. IEEE*, vol. 101, no. 5, 2013.

[14] B. Dumas, D. Lalanne, and S. Oviatt, "Multimodal Interfaces : A Survey of Principles , Models and Frameworks," in *Human Machine Interaction*, D. Lalanne and J. Kohlas, Eds. Springer Berlin Heidelberg, 2009, pp. 3–26.

[15] B. Thomson, "Statistical methods for spoken dialogue management," doctoral thesis, University of Cambridge, 2009.

[16] J. D. Williams and S. Young, "Partially observable Markov decision processes for spoken dialog systems," *Comput. Speech Lang.*, vol. 21, no. 2, pp. 393–422, Apr. 2007.

[17] D. Schlangen and G. Skantze, "A General , Abstract Model of Incremental Dialogue Processing," in *Proceedings of the 12th Conference of the European Chapter of the ACL*, 2009, no. April, pp. 710–718.

[18] D. Traum, D. Devault, J. Lee, Z. Wang, and S. Marsella, "Incremental Dialogue Understanding and Feedback for Multiparty , Multimodal Conversation," in *Intelligent Virtual Agents*, Springer Berlin Heidelberg, 2012, pp. 275–288.

[19] V. V. Petukhova, "Multidimensional Dialogue Modelling," Ph.D thesis, Tilburg University, 2011.

[20] S. Kang, Y. Ko, and J. Seo, "Generating Effective Responses for a Plan-based Dialogue System in Human-robot Interaction," *INFORMATION-AN Int. Interdiscip. J.*, vol. 15, no. 2, pp. 949–960, 2012.

[21] D. Bohus and A. I. Rudnicky, "The RavenClaw dialog management framework: Architecture and systems," *Comput. Speech Lang.*, vol. 23, no. 3, pp. 332–361, Jul. 2009.

[22] D. Deugo, M. Weiss, and E. Kendall, "Reusable patterns for agent coordination," in *Coordination of Internet agents*, A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, Eds. Berlin: Springer, 2001, pp. 347–368.

[23] K. Georgila and M. Stone, Eds., "15th Annual Meeting of the Special Interest Group on Discourse and Dialogue," in *Proceedings of the Conference*, 2014, no. June.

[24] J. José Neto, "Contribuiçoes à metodologia de construçao de compiladores," tese de doutorado, USP, São Paulo, Brasil, 1993.

[25] J. José Neto, "Adaptive Automata for Context-Sensitive Languages," *SIGPLAN Not.*, vol. 29, no. 9, pp. 115–124, 1994.

[26] D. A. Alfenas, D. P. Shibata, J. José Neto, and M. R. Pereira-Barretto, "Sistemas de Markov Adaptativos: Formulação e Plataforma de Desenvolvimento," in *Memórias do WTA 2012 Sexto Workshop de Tecnologia Adaptativa*, 2012.

[27] C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler, and M. Smith, "OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)," *W3C Recommendation*, 2012. [Online]. Available: http://www.w3.org/TR/owl2-syntax/. [Accessed: 20-Jul-2014].

[28] M. Horridge and S. Bechhofer, "The owl api: A java api for owl ontologies," *Semant. Web*, vol. 2, pp. 11–21, 2011.

[29] I. Horrocks, B. Motik, and Z. Wang, “The HermiT OWL Reasoner,” pp. 1–6.

Daniel Assis Alfenasformou-seem Engenhariada Computação pela Escola Politécnica da Universidade de São Paulo (EPUSP)em 2004.Trabalhou, nos últimos dez anos, nas diversas áreas do desenvolvimento de sistemas, desde a definição da demanda até a implantação do sistema. Recentemente tem trabalhado como coordenador técnico no desenvolvimento de sistemas complexos que envolvem simulação, otimização e interfaces ricas. É mestrando em Engenharia de computaçãopela EPUSP, com pesquisa na aplicação da tecnologia adaptativa no gerenciamento de diálogo em linguagem naturalentre usuários e sistemas.

Carlos Eduardo Bittencourt Bogik é bacharel em Ciências da Computação pela Pontifícia Universidade Católica de São Paulo (PUC-SP) em 2003. Trabalha como programador em linguagem Java e analista de sistemas desde 2002, principalmente com sistemas Web para instituições financeiras do Brasil.

Danilo Picagli Shibata é mestre em Engenharia da Computação pela Escola Politécnica da USP (EPUSP), com título obtido em 2008 pelo estudo da aplicação de autômatos adaptativos na tradução texto-fala para textos escritos na língua portuguesa. Trabalhou com análise de segurança e confiabilidade de sistemas computacionais aplicados a sistemas críticos, desenvolvimento de software na área de segurança da informação e no desenvolvimento de simuladores de transporte de fluídos. Hoje é arquiteto de sistemas desenvolvidos em Java.

Raphael Pinheiro da Silva formou-se em Engenharia de Computação pela Escola Politécnica da Universidade de São Paulo (EPUSP) em 2004 e concluiu o curso de MBA em Inovação e Tecnologia pela Fundação Instituto de Administração (FIA) em 2013. Atua na área de desenvolvimento de software desde 2002. Atualmente é arquiteto de sistemas e líder técnico de equipe de desenvolvimento atuando principalmente com soluções na linguagem JAVA.

Marcos Ribeiro Pereira-Barretto é graduado em Engenharia Elétrica (1983), mestre em Engenharia Elétrica (1988) e doutor em Engenharia Mecânica (1993) pela Escola Politécnica da Universidade de São Paulo. Éprofessor da Escola Politécnica da USP desde 1986. Atua nas seguintes áreas de pesquisa: robôs sociáveis, computação afetiva e arquitetura de sistemas para aplicações críticas como Automação Industrial.

Determinação do escopo geográfico de textos através de uma hierarquia adaptativa de classificadores

Eduardo Marcel Maçan
Escola Politécnica da Universidade de São Paulo
Email: macan@usp.br

Edson Satoshi Gomi
Escola Politécnica da Universidade de São Paulo
Email: gomi@usp.br

***Resumo*—A ambiguidade geográfica de topônimos em textos é o principal obstáculo para a atribuição correta de coordenadas geográficas a textos que mencionam locais e a principal causa de erros em tarefas de anotação geográfica. Este artigo apresenta um novo método para determinação do escopo geográfico de um texto através de uma abordagem adaptativa baseada em uma hierarquia de classificadores de texto. Adicionalmente, este trabalho apresenta uma análise da estrutura da ambiguidade geográfica de topônimos brasileiros. A Wikipédia foi utilizada como fonte de um conjunto de documentos anotados geograficamente para o treinamento de uma hierarquia de SVMs (Support Vector Machines) para a determinação do escopo geográfico e consequente redução da ambiguidade geográfica dos textos.**

I. Introdução

De acordo com o Google [4], em 2011 cerca de 20% de todas as consultas realizadas a partir de computadores pessoais foram efetuadas por usuários em busca de informação local. Este número chega a 40% para consultas realizadas a partir de dispositivos móveis. A busca por informações locais a partir de dispositivos móveis é uma consequência natural do fato de que pessoas em movimento geralmente buscam informações sobre pontos de interesse nas suas imediações. A busca por informações locais é facilitada se os textos disponíveis na internet estiverem anotados geograficamente, isto é, se para cada texto houver um rótulo (tag) que identifique a região à qual o texto se refere. A disponibilidade de textos anotados geograficamente permitirá o surgimento de novas classes de aplicativos móveis, como sistemas automotivos embarcados que selecionem e leiam notícias relevantes para o motorista durante seu trajeto.

A anotação geográfica é o processo de associar uma região geográfica a um documento. Esta associação é feita a partir da identificação de nomes de locais, denominados topônimos, existentes no texto. No entanto, anotar geograficamente documentos da internet é um processo custoso e impraticável se realizado manualmente. Por esta razão é fundamental encontrar um modo automático para extrair informação geográfica precisa de textos.

A criação de métodos de anotação geográfica exige a resolução de diversos problemas. Um dos problemas relevantes é o da ambiguidade de topônimos. Um topônimo é ambíguo quando mais de um local possui o mesmo nome. Por exemplo, existem no Brasil 826 ruas "Getúlio Vargas", localizadas em 26 estados diferentes, segundo dados do Censo de 2010 [7]. Assim, mesmo que seja possível encontrar nomes de locais dentro de um texto, ainda é preciso resolver as eventuais ambiguidades dos topônimos encontrados, antes de associar o texto a uma determinada região.

A probabilidade de existência de topônimos ambíguos é maior numa área geográfica mais ampla. Por esta razão é importante reduzir a área a ser associada a um texto, de forma a permitir a eliminação da ambiguidade dos topônimos nele contidos. A região geográfica resultante, que contém os topônimos do texto, é chamada de foco ou escopo geográfico.

Neste artigo é apresentado um novo método adaptativo para determinação do escopo geográfico. A correta determinação do escopo geográfico do texto reduz a quantidade de alternativas de mapeamento para topônimos, pois locais fora da área estabelecida são descartados para a resolução dos topônimos encontrados no documento.

O restante dste artigo está estruturado da seguinte forma: a Seção "Anotação Geográfica de Textos" apresenta uma breve descrição das principais referências em anotação geográfica relevantes para este trabalho. A seção "Ambiguidade Geográfica" analisa a ambiguidade geográfica dos logradouros brasileiros, apresentando visualizações de sua estrutura. A seção "Dados e Experimentos" descreve como foram obtidos os dados e estruturados os experimentos. A seção "Análise e Conclusões" apresenta considerações sobre os resultados obtidos e "Conclusões" sumariza os resultados obtidos até o presente momento.

II. Anotação Geográfica de Textos

A maioria dos algoritmos para a anotação geográfica de textos utiliza gazetteers para encontrar informação geográfica relativa aos topônimos identificados. Gazetteers, ou diretórios toponímicos, são bancos de dados que armazenam atributos associados a nomes de locais, como a população de uma cidade, suas coordenadas geográficas e dados políticos e econômicos. Estes atributos são utilizados por heurísticas para decidir quais coordenadas ou áreas geográficas melhor representam os nomes de locais mencionados no texto. O Getty Thesaurus of Geographic Names Online [6] e o GeoNames [5] são dois exemplos notórios de gazetteers que podem ser consultados pela internet.

Um dos primeiros sistemas para anotação geográfica foi o GIPSY (Geographical Information Processing System, ou Sistema de Processamento de Informação Geográfica), desenvolvido por Woodruff e Plaunt [23]. O sistema GIPSY utiliza um algoritmo de três passos para a anotação geográfica. O primeiro passo identifica palavras e frases que possuem relevância geográfica, comparando-as com os nomes existentes num gazetteer. O segundo passo busca representações poligonais das áreas mencionadas. O último passo combina os polígonos em uma representação tridimensional, tal como a apresentada na Figura 1. A sobreposição de polígonos das áreas mencionadas no texto cria um relevo tridimensional, cujos picos indicam no mapa as principais regiões às quais o documento se refere.

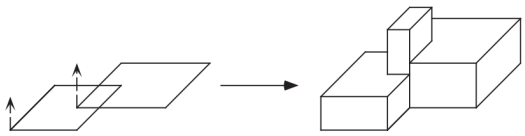


Figura 1: Sobreposição de áreas geográficas. [23]

Leidner [9] cunhou os termos "Extração de Topônimos" e "Resolução de Topônimos" para representar os dois passos principais que a maior parte dos algoritmos de anotação geográfica de textos implementa. No GIPSY, a extração de topônimos é realizada no primeiro passo e a resolução de topônimos, nas demais etapas. Leidner também descreveu 16 heurísticas e regras comuns, utilizadas por diferentes autores para eliminar a ambiguidade de topônimos. Por exemplo, uma dessas heurísticas recomenda escolher os topônimos dentro das áreas com maior população quando em dúvida sobre que local o topônimo identificado representa.

O sistema Web-a-Where [1] foi um dos primeiros a propor a anotação geográfica automática de conteúdo da Web. O primeiro passo executado pelo Web-a-Where é a busca nomes de cidades, estados e países e atribuição de um peso a cada topônimo encontrado no texto. Estes pesos variam entre 0 e 1 e são atribuídos de acordo com uma medida arbitrária de ambiguidade. Para topônimos não ambíguos, como "Chicago, IL" o valor 0.95 é atribuído. Se existem vários topônimos iguais, mas apenas um deles não ambíguo, um valor entre 0.8 e 0.9 é atribuído a todos (por exemplo, múltiplas menções a "Chicago" e apenas uma a "Chicago, IL"). Para todos os outros topônimos ambíguos encontrados é atribuído o valor 0.5 e cada topônimo é associado ao local com a maior população entre aqueles de mesmo nome. O foco geográfico é então calculado através de um sistema de pontuação que utiliza o peso atribuído a cada topônimo no primeiro passo. Ao encontrar um topônimo que representa uma cidade, representada por um foco geográfico no formato Cidade/Estado/País, soma-se para este foco geográfico o valor p^2, onde p é o peso do topônimo encontrado. Este peso é então propagado para o foco geográfico hierarquicamente superior, com um fator de atenuação $d = 0.7$. Ao encontrar uma menção ao foco geográfico Cidade/Estado/País e somar p^2 a sua pontuação, os valores p^2d e p^2d^2 são somados às pontuações de Estado/País e País, respectivamente. Os focos geográficos são finalmente ordenados de acordo com sua pontuação e as quatro maiores entre aquelas com valor maior que 0.9 são selecionadas para representar o(s) foco(s) geográfico(s) do documento. Os valores de d e p utilizados pelo Web-a-Where foram determinados de forma empírica.

O Web-a-Where foi testado com um conjunto de páginas web classificadas manualmente [24] e foi capaz de identificar corretamente a que país um texto fazia referência em 92% dos casos, dos quais apenas 38% identificavam corretamente o estado ou cidade a que o texto se referia.

Overell [11] mapeia termos da Wikipédia para termos da WordNet [10] e usa os verbetes que representam palavras associadas a locais no banco de dados léxico WordNet para determinar quais verbetes da Wikipédia se referem a locais. No passo seguinte, buscam-se pares de nomes de locais que ocorrem dentro de cada texto da Wikipédia. A ideia é que locais geograficamente próximos são frequentemente mencionados dentro no mesmo verbete. A frequência com que pares de locais ocorrem juntos é utilizada para determinar o escopo geográfico correto para os pares encontrados no documento sendo analisado.

Estas abordagens representativas das técnicas de determinação de escopo geográfico e de extração de topônimos baseiam-se em consultas a tabelas de nomes de locais para a identificação de topônimos em um texto. A extensão e correção destas tabelas influencia diretamente na qualidade dos algoritmos, como observador por Amitay [1]. Muitos dos nomes de locais em um gazetteer serão ambíguos. Ao lidar com nomes de locais em uma coleção de textos históricos Smith e Crane [16] observaram que de todos os topônimos em seus documentos, 92% podiam ser traduzidos em mais de uma coordenada geográfica.

Algumas das heurísticas de desambiguação de nomes de locais catalogadas por Leidner [9] forçam que o resultado do algoritmo se adapte a uma característica da distribuição geográfica dos documentos, sem incremento à eficiência do algoritmo em si. Por exemplo, a regra de se privilegiar locais com maior população sempre que um topônimo resultar em referências a mais de uma localidade no mapa.

Overell [11] descreve e valida experimentalmente o que batizou de "Hipótese de Steinberg", que afirma que locais geograficamente próximos ao assunto do texto e a seu autor possuem relevância muito maior do que locais distantes. Desta forma, para se identificar os topônimos de um texto e posicioná-los corretamente em um mapa é necessário estimar previamente a que região geográfica o documento se refere. Um local pouco povoado, porém geograficamente próximo ao assunto do texto é muito mais relevante ao traduzir topônimos em posições no mapa. A heurística de desambiguação por população está em direta oposição a este princípio.

Outras heurísticas comuns se utilizam de regras específicas da língua em que os textos estão escritos, como assumir que sequências de palavras escritas com letras maiúsculas possam

ser candidatas a nomes de locais, ou, a exemplo de Overell, utilizam léxicos da língua inglesa como a WordNet, não disponíveis para outras línguas, para identificar que palavras são nomes de locais.

A Wikipédia em português foi utilizada como fonte de documentos para este trabalho, mas não são assumidas premissas a respeito da língua ou estrutura sintática dos documentos.

III. Ambiguidade Geográfica

O problema da ambiguidade de nomes de locais foi investigado desde os primeiros experimentos para a extração e resolução de topônimos. [23] estiveram entre os primeiros a enumerar as diferentes maneiras com as quais locais falham em ser identificados corretamente por seus nomes. Eles encontram os seguintes fatores que contribuem para a ambiguidade de nomes de locais:

- **Nomes de locais raramente são únicos**. Por exemplo, Cambridge pode se referir a Cambridge, Massachussets ou Cambridge, Inglaterra;
- **Fronteiras políticas mudam com o tempo**, como resultado de tratados ou guerras;
- **Nomes de locais mudam**, o que faz com que seja difícil manter uma lista atualizada de nomes ao mesmo tempo em que se preserva sua utilidade para referências em documentos históricos;
- **Variação de grafia**. Nome de locais são frequentemente escritos de maneiras diversas em diferentes línguas.

Amitay et Al. [1] classificam os principais tipos de ambiguidade em geo/geo e geo/non-geo, pois locais podem ser confundidos com outros locais homônimos ou com palavras que não representem um lugar, respectivamente.

A ambiguidade do tipo geo/non-geo, ou seja, nomes que não representem locais, mas são confundidos com nomes de locais ("Mariana" e "Mariana, MG", por exemplo) depende da interpretação semântica do texto e da área geográfica sendo considerada. Por outro lado, a ambiguidade do tipo geo/geo, locais diferentes que possuem o mesmo nome ("California, Estados Unidos" e "Califórnia, Paraná", por exemplo) pode ser estimada, desde que uma lista de nomes de locais suficientemente completa esteja disponível para uma região. Para este trabalho um gazetteer com mais de 2,5 milhões de nomes de locais brasileiros foi construído e as métricas de ambiguidade absoluta de uma área e de um documento foram definidas.

- **Ambiguidade Absoluta de uma área**: representa a ambiguidade de um conjunto de topônimos em relação aos topônimos da área que os contém, por exemplo: a ambiguidade absoluta da cidade de Londrina em relação ao Estado do Paraná. Dados os conjuntos L_c e E de logradouros tal que L_c contém logradouros da cidade c e E contém logradouros de uma área que inclui a cidade c e a função F definida para pares $[logradouro, cidade]$ l_i de L_c, a ambiguidade absoluta de L_c é definida pela função $A(L_c)$ da equação 1.

$$A(L_c) = \frac{\sum_{i=1}^{n} F(l_i)}{|E|},$$
$$F(l_i) = \begin{cases} 0 & se \quad l_i \notin E - L_c, \\ 1 & se \quad l_i \in E - L_c \end{cases} \quad (1)$$

- **Ambiguidade total de um documento**: A quantidade de topônimos ambíguos identificados em um documento de texto. Sejam n topônimos T_i (i de 1 a n) identificados no texto e para cada topônimo T_i, seja A_i o número de locais diferentes com o mesmo nome de T_i identificados, a Ambiguidade total do documento A_{doc} é definida pela equação 2.

$$A_{doc} = \sum_{i=1}^{n} A_i \quad (2)$$

IV. Dados e Experimentos

A. Dados do Censo IBGE 2010

O IBGE (Instituto Brasileiro de Geografia e Estatística) disponibiliza arquivos contendo dados de todos os endereços visitados por pesquisadores de campo em áreas urbanas e rurais durante o Censo de 2010 [7]. Grande parte desses endereços possui coordenadas geográficas precisas. Neste projeto de pesquisa, nos casos em que latitude e longitude não estão disponíveis são consideradas as coordenadas da cidade da qual os endereços fazem parte.

Estes dados foram utilizados para construir um gazetteer suficientemente completo do território Brasileiro. A Figura 2 apresenta a área coberta por este gazetteer.

Figura 2: Os pontos cinza representam as cidades brasileiras cobertas pelo Gazetteer construído para este projeto de pesquisa.

B. Documentos Georreferenciados da Wikipédia

A base de artigos da Wikipédia em português foi utilizada como fonte para um conjunto de documentos georreferenciados.

A Wikipédia é composta por documentos escritos com uma linguagem própria que inclui marcação para representar coordenadas geográficas no texto. Sempre que um verbete diz respeito a um local estas marcações são traduzidas para o usuário final como links para seviços de mapas online, através do projeto [21]. Todos os documentos da Wikipédia em português que contêm marcações do tipo "geocoordenadas" [19], "satélite" [20] e "coord" [18] foram identificados através do uso de expressões regulares e tiveram suas respectivas posições geográficas extraídas.

O conteúdo do verbete e metainformação como título, URL e coordenada de cada documento identificado foi inserido em uma tabela de um banco de dados geográficos PostGIS [13], resultando em uma base georreferenciada de documentos com 22438 itens. A figura 3 exibe um detalhe da distribuição geográfica destes documentos.

Figura 3: Detalhe dos Artigos Georreferenciados da Wikipédia em Português

O IBGE disponibiliza polígonos [8] em alta resolução dos contornos de todas as cidades do país. Para cidade são selecionados os documentos da Wikipédia com coordenadas internas ao perímetro municipal através de uma query ao banco de dados geográfico. Rótulos são atribuídos a cada documento, correspondentes à cidade, micro e mesorregião, estado e região do país (Norte, Nordeste, Centro-Oeste, Sudeste ou Sul) a que pertencem.

Estas regiões possuem entre si relacionamentos do tipo "contém" e "está contida em", formando uma hierarquia geográfica.

A representação HTML dos verbetes selecionados foi obtida através de requisições web para a Wikipédia e convertidas para texto puro, totalizando 7863 documentos dentro do território brasileiro, georreferenciados e anotados com seus respectivos rótulos geográficos.

Cada um destes rótulos atribui uma classe ao documento, ao mesmo tempo em que o relaciona a uma região geográfica. Um classificador de textos capaz de atribuir corretamente um destes rótulos a um documento determinará também seu escopo geográfico.

C. Cálculo da ambiguidade absoluta de locais

A noção intuitiva de que haverá menos ambiguidade de nomes de locais quanto menor for a área considerada pode ser verificada medindo-se a ambiguidade absoluta dos logradouros de um município em áreas progressivamente menores.

A Tabela I apresenta amostras da ambiguidade absoluta de algumas cidades brasileiras. A ambiguidade absoluta mínima foi observada para a cidade de Brasília, DF com $A_{Brasilia} = 0.0145$ de ambiguidade e a máxima para a cidade de São José do Hortêncio, RS com$A_{SJH} = 0.93$. É interessante observar que Brasília possui um modelo de endereçamento único no território nacional e que na pequena São José do Hortêncio a maior parte de seus 61 endereços é apenas numerada, daí os extremos observados.

Tabela I: Ambiguidade absoluta de cidades medida dentro de diferentes áreas

City	Brazil	Region	State
S. J. Hortêncio	0.9344	0.8852	0.8033
Belo Horizonte	0.4806	0.4336	0.3125
Rio de Janeiro	0.4765	0.3959	0.2113
Campinas	0.3272	0.2930	0.2113
São Paulo	0.3249	0.2345	0.1672
Curitiba	0.2341	0.1748	0.1425
Brasília	0.0145	0.0099	-

Observe que a ambiguidade dos logradouros de uma cidade decresce quando calculada para logradouros dentro de uma área menor. A ambiguidade geo/geo calculada desta forma pode ser tomada como uma estimativa do limite inferior da ambiguidade, pois apenas correspondências perfeitas entre nomes de locais completos e sem abreviações foram levadas em consideração. "Rua Rui Barbosa" e "Avenida Rui Barbosa" não são considerados homônimos neste experimento.

D. Wikipédia: Número de exemplos por região

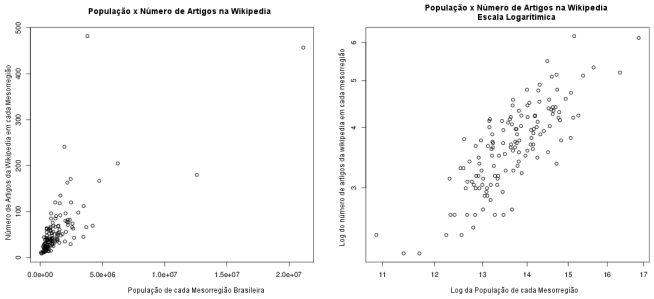


Figura 4: Número de documentos georeferenciados da Wikipédia por mesorregião brasileira.

O IBGE subdivide os estados brasileiros em 137 mesorregiões e 554 microrregiões compostas de cidades vizinhas que compartilham algumas características geopolíticas, econômicas ou sociais [12], como as regiões metropolitanas, por exemplo. A Figura 4 apresenta a relação entre número

de páginas georreferenciadas da Wikipédia dentro de cada mesorregião brasileira, em escala natural e logarítmica.

Estes dados foram obtidos pela extração de coordenadas geográficas presentes em aproximadamente 1% dos artigos da Wikipedia em português, a partir da base de verbetes da Wikipedia disponível em formato XML [22]. A figura 4 sugere uma correlação forte entre o número de artigos em uma determinada área do mapa e a população desta área.

O coeficiente de correlação de Spearmam [17] é utilizado para a análise não-paramétrica de correlação entre duas listas de valores (rankings). O coeficiente ρ, calculado para as variáveis "população" e "número de artigos na Wikipédia" para cada mesorregião é de $\rho = 0.775$, valor que indica uma correlação forte. Esta constatação confirma a noção intuitiva de que existe uma relação entre a população de uma área e o número de artigos na Wikipédia sobre esta área e permite conjecturar que esta relação também se aplique a outros tipos de documentos, como por exemplo, notícias online. Um banco de documentos anotado geograficamente a partir de documentos da web apresentará portanto uma distribuição não uniforme de documentos por toda a área considerada. Esta distribuição não uniforme de documentos deve ser levada em conta para o uso de ferramentas automáticas de classificação de textos.

E. Classificação entre duas cidades

O uso de classificadores de texto que usam a abordagem "Bag of Words"de contagems de palavras do texto é frequente em aplicações como detecção automática de propaganda não solicitada por e-mail (SPAM), como proposto por Sahami et Al. [14]. O uso de mais de uma palavra como token individual, conhecido como n-grama, foi estudado por Berrkerman [2] e por Caropreso [3], entre outros, como features para a classificação de textos, com diferentes resultados dependendo do domínio de aplicação.

Para este experimento inicial de classificação foi utilizado um classificador Naive Bayes [14] com seleção automática de atributos usando TF-IDF [15] em uma base de treinamento limitada a documentos pertencentes às cidades do Rio de Janeiro e São Paulo, extraídos da base de verbetes georreferenciados da wikipedia. Variando-se o número de documentos reservados para o treinamento, pode-se observar que a partir de poucas dezenas de artigos exemplo é possível atingir perto de 90% de acerto de classificação, conforme mostra a figura5

Este resultado permite avaliar que ao menos algumas dezenas de exemplos são necessários em cada região geográfica, para que os classificadores possam ser adequadamente treinados. Os documentos da Wikipédia não são distribuídos uniformemente. Algumas microrregiões, como as regiões metropolitanas de São Paulo e Campinas, possuem sozinhas mais documentos georreferenciados do que alguns estados inteiros do centro-oeste e norte do Brasil.

F. Hierarquia de Classificadores de Aprendizagem de Máquina

Ao invés de treinar um único classificador capaz de atribuir um escopo geográfico mais específico diretamente (um

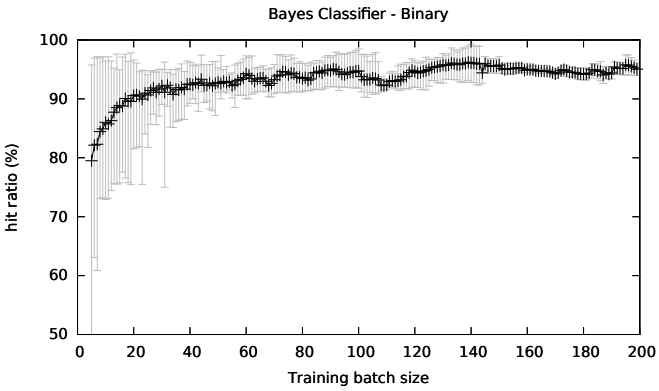


Figura 5: Sucesso de classificação Naive Bayes em verbetes georreferenciados da wikipedia para o Rio e São Paulo, com lotes de treinamento de tamanhos diversos. As barras de erro representam o melhor e pior resultado de cada lote de treinamento e validação cruzada.

único classificador com 5566 classes) foram treinados diversos classificadores para cada região, treinados com os documentos pertencentes a suas subdivisões regionais, de forma a compor uma hierarquia de classificadores, correspondente à hierarquia das regiões geográficas.

Um classificador capaz de atribuir corretamente um destes rótulos a um texto também determinará seu escopo geográfico, pois cada rótulo corresponde diretamente a uma área no mapa. A identificação correta dos locais mencionados no texto se tornará mais simples, pois muitas interpretações incorretas de nomes de locais com coordenadas fora do escopo geográfico poderão ser eliminadas.

A cada execução do experimento foram escolhidos aleatoriamente documentos para compor um conjunto de treinamento e outro de validação, na proporção de 70% e 30% respectivamente. Um classificador capaz de atribuir o documento a categorias correspondentes às 5 regiões brasileiras é treinado para a raiz da hierarquia. Para cada uma destas regiões treinamos um classificador capaz de atribuir o documento a um dos estados da região selecionada como escopo geográfico e assim por diante até o nivel das cidades. A figura 6 exibe um subconjunto dos nós da hierarquia proposta.

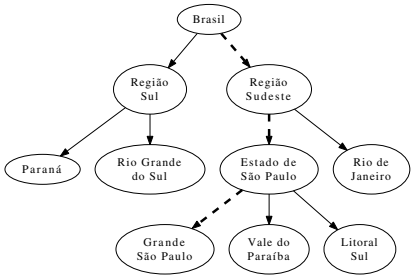
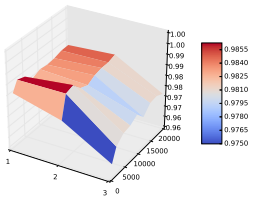


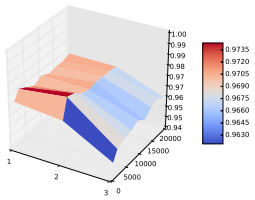
Figura 6: Hierarquia geográfica. Cada classificador corresponderá a um nó, com classes que representam seus nós filhos

Para a determinação do escopo geográfico o documento

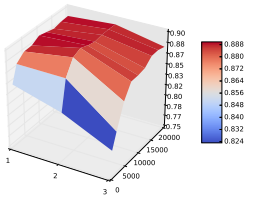
foi avaliado pelo classificador da raiz da hierarquia e foi atribuído um rótulo correspondente a uma região brasileira. Na iteração seguinte, o documento será avaliado pelo classificador correspondente àquela região que por sua vez lhe atribuiu o rótulo de um estado e assim sucessivamente, até chegarmos a atribuição de um rótulo correspondente a uma folha da árvore da hierarquia geográfica. As linhas tracejadas da figura 6 exemplificam um possível "percurso" de um documento pela hierarquia até a atribuição do escopo geográfico mais específico.



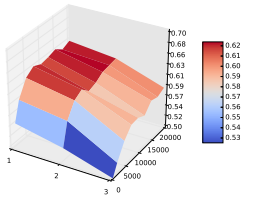
(a) Escopo geográfico: Região.



(b) Escopo geográfico: Estado.



(c) Escopo geográfico: Mesorregião



(d) Escopo geográfico: Microrregião

Figura 7: Taxas de acerto da hierarquia de SVMs em função dos tamanhos do vocabulário e dos n-gramas utilizados

Podemos observar que o desempenho cai, à medida que se busca determinar um escopo geográfico mais específico, é possível determinar que um texto pertence a um determinado estado do país com mais de 95% de correção, mas este índice cai a cerca de 65% para as microrregiões e abaixo de 20% para as cidades, conforme a figura 8

G. Hierarquia de classificadores: Abordagem Adaptativa

A distribuição geográfica não uniforme dos documentos, evidenciada pela figura 3 faz com que muitas cidades não possuam documentos exemplo em número suficiente para o treinamento do classificador de textos e este fator foi ignorado na construção da hierarquia de classificadores. Este é um fator significativo, que não pode ser ignorado.

Outro ponto a ser considerado é que o escopo geográfico de um documento não precisa ser sempre o mais específico. Um documento que tenha seu escopo geográfico determinado como "Nordeste"é correto, embora menos preciso do que o escopo geográfico "Ceará". Ainda assim, é preferível obter como resposta o escopo correto "Nordeste", do que o escopo geográfico errado "Pernambuco", se o documento se refere ao "Ceará".

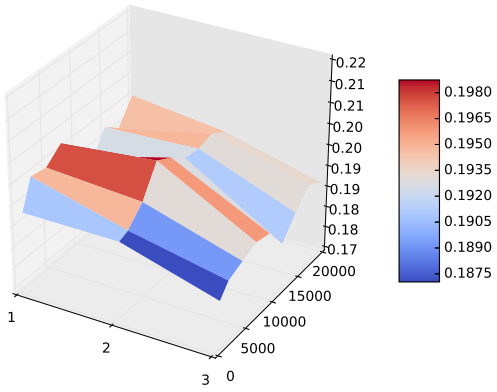


Figura 8: Desempenho para o escopo geográfico Cidade

Uma versão adaptativa da hierarquia de classificadores foi desenvolvida de forma a treinar classificadores apenas para as regiões que já possuem um número mínimo de exemplos disponíveis na base de treinamento. À medida que novos exemplos de treinamento se tornarem disponíveis, o número de classificadores e a topologia da hierarquia serão reconfiguradas, segundo os seguintes passos:

Para cada nó da hierarquia de classificadores, a partir da raiz, percorrido em largura:

- Verficar se o número de exemplos para treinamento deste nó mudou desde a última execução e há exemplos suficientes
- Obter os exemplos de treinamento para relevantes para este nó da hierarquia.
- Treinar o classificador do nó atual da hierarquia.

A inserção de novos exemplos na base de treinamento se deve dar sempre numa folha da árvore da hierarquia geográfica. Neste projeto de pesquisa o escopo mais específico é a cidade. Portanto todo novo documento a compor a base de treinamento deve ter tido uma cidade atribuída.

Porém, esta versão adaptativa da hierarquia de classificadores não atribui sempre uma cidade a um documento. Neste caso, pode-se utilizar o gazetteer para identificar os locais mencionados no texto, aproveitando-se da redução de ambiguidade proporcionada pela determinação do escopo geográfico, ou recorrer à validação de um usuário, conforme figura 9.

Alteração do número de documentos disponíveis para treinamento para um nó se propaga para todos os nós hierarquicamente superiores até a raiz, pois também farão parte de seus conjuntos de treinamento.

A figura 7 mostra a taxa de acertos da determinação do escopo geográfico em cada nível da hierarquia geográfica em função dos tamanhos dos n-gramas e do vocabulário dos classificadores SVM utilizados pela hierarquia adaptativa.

Embora note-se melhora de performance nas classificações de meso e microrregiões, os resultados para classificação

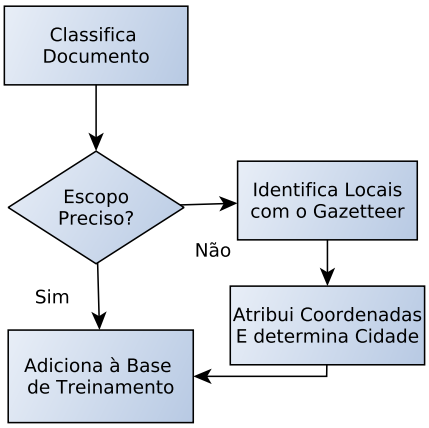


Figura 9: Inserção de novo documento na base de treinamento

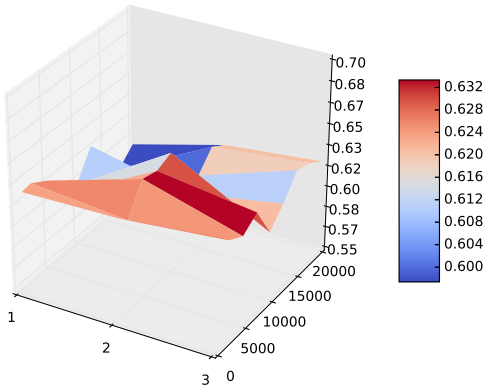
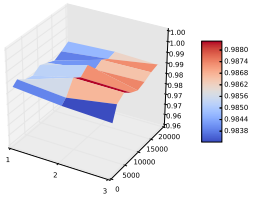
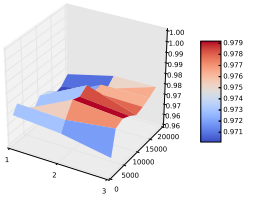


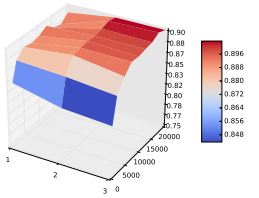
Figura 11: Desempenho para o escopo geográfico Cidade



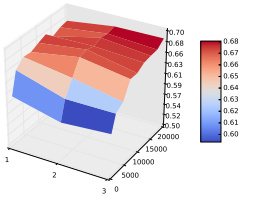
(a) Escopo geográfico: Região.



(b) Escopo geográfico: Estado.



(c) Escopo geográfico: Mesorregião



(d) Escopo geográfico: Microrregião

Figura 10: Taxas de acerto da hierarquia adaptativa de SVMs em função dos tamanhos do vocabulário e dos n-gramas utilizados

de documentos entre cidades é significativamente melhor, na ordem de 60% de classificações corretas, conforme figura 11.

Esta melhora se deve ao fato que esta nova abordagem de treinamento irá interromper o percurso do documento pela árvore de classificadores nas regiões que não possuem informação suficiente para treinamento, atribuindo um escopo geográfico menos preciso, mas que tem mais chances de estar correto.

V. Análise e Conclusões

A hipótese de Steinberg, formulada por Overell [11] diz que "todas as pessoas possuem visões de mundo similares com respeito a sua própria localidade" e que a relevância de um local para uma pessoa decresce com a distância. Ou seja, todos os indivíduos interpretam a relevância de locais da mesma maneira, tomando sua própria posição como referencial. Uma consequência importante desta hipótese é que para desambiguar corretamente os locais mencionados em um documento é preciso primeiro ter uma idéia aproximada de que região considerar para o texto. A hierarquia de classificadores proposta neste trabalho se mostrou uma ferramenta com potencial a ser explorado para esta tarefa.

A popular heurística das áreas mais populosas para desambiguação de textos assume que áreas mais populosas são provavelmente as corretas devido à correlação forte entre a população de uma área e o número de textos disponíveis pertencentes àquela área. No caso de uma alteração desta relação, esta heurística contribuiria para a classificação incorreta destes textos.

Não é prático construir um único classificador com 5565 classes para a tarefa de atribuição de escopo geográfico para todas as cidades brasileiras. Muitas cidades podem não possuir exemplos disponíveis em número adequado para treinamento. Uma abordagem hierárquica em que cada classificador pode se adaptar às peculiaridades de cada região geográfica é mais adequada.

Ao invés de depender de um único gazetteer com informações de todo o mundo em todas as línguas, com uma estrutura de manutenção complexa, é possível utilizar gazetteers menores e especializados, para atribuição de coordenadas precisas aos topônimos de sua região, após a determinação do foco geográfico, quando necessário.

A hierarquia de classificadores permite também, ao se utilizar seleção automática de atributos por tf-idf, que as características mais importantes para cada região específica, emergentes dos documentos exemplo sejam utilizadas, mesmo quando o número de atributos é limitado para evitar a explosão dimensional ocasionada pelo uso de n-gramas. Uma analogia visual pode ser feita em relação às interfaces gráficas para navegação em mapas. Quando o nível de zoom é tal que

abrange todo o território nacional, apenas alguns atributos de cada estado como rios e principais cidades são exibidos. Os atributos mudam e se tornam mais específicos, conforme o escopo é estreitado e passamos a exibir o conteúdo de um estado, ou de uma cidade.

O método apresentado exibe excelente desempenho para documentos que possuam foco geográfico único e bem definido, como notícias online. Um texto comparando Paris a São Paulo, por exemplo, não possui um único foco geográfico definido, mas pode ser dividido em partes menores submetidas individualmente à determinação do foco geográfico.

REFERÊNCIAS

[1] E. Amitay, N. Har'El, R. Sivan, and A. Soffer. Web-a-where. In *Proceedings of the 27th annual international conference on Research and development in information retrieval - SIGIR '04*, page 273, New York, New York, USA, July 2004. ACM Press.

[2] R. Bekkerman and J. Allan. Using bigrams in text categorization, 2003.

[3] M. F. Caropreso, S. Matwin, and F. Sebastiani. Text databases & document management. chapter A Learner-independent Evaluation of the Usefulness of Statistical Phrases for Automated Text Categorization, pages 78–102. IGI Global, Hershey, PA, USA, 2001.

[4] N. Fox. Official google blog: Ads are just answers. http://googleblog.blogspot.com.br/2011/10/ads-are-just-answers.html, Oct. 2011. Acessado em 10/12/2013.

[5] Geonames. Geonames geographical database. http://www.geonames.org/, Oct. 2005. Acessado em 17/06/2014.

[6] Getty. Getty thesaurus of geographic names. http://www.getty.edu/research/tools/vocabularies/tgn/, Oct. 1987. Acessado em 17/06/2014.

[7] IBGE. Cadastro Nacional de Endereços para Fins Estatísticos. ftp://ftp.ibge.gov.br/Censos/Censo_Demografico_2010/Cadastro_Nacional_de_Enderecos_Fins_Estatisticos, 2010. Accessado em 19/07/2012.

[8] IBGE. Malhas digitais Brasileiras. ftp://geoftp.ibge.gov.br/malhas_digitais/municipio_2010/, 2010. Acessado em: 19/07/2012.

[9] J. L. Leidner. *Toponym Resolution in Text: Annotation, Evaluation and Applications of Spatial Grounding of Place Names*. PhD thesis, School of Informatics, University of Edinburgh, 2007.

[10] G. A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, Nov. 1995.

[11] S. Overell. Geographic Information Retrieval: Classification, Disambiguation and Modelling. (July):1–181, 2009.

[12] J. R. Portela, editor. *Divisão Regional do Brasil em Mesorregiões e Microrregiões Geográficas*, volume 1. IBGE, 1990.

[13] P. S. C. PostGIS. PostGIS. http://postgis.net, 2012. acessado em: 19/07/2012.

[14] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk e-mail, 1998.

[15] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. In *INFORMATION PROCESSING AND MANAGEMENT*, pages 513–523, 1988.

[16] D. Smith and G. Crane. Disambiguating geographic names in a historical digital library. . . . *and Advanced Technology for Digital Libraries*, 2001.

[17] C. Spearman. The proof and measurement of association between two things. *American Journal of Psychology*, 15:88–103, 1904.

[18] Wikipedia. Predefinição: Coord. http://pt.wikipedia.org/wiki/Predefini%C3%A7%C3%A3o:Coord/doc, Oct. 2005. Acessado em 10/12/2013.

[19] Wikipedia. Predefinição: Geocoordenadas. http://pt.wikipedia.org/wiki/Predefini%C3%A7%C3%A3o:Geocoordenadas/doc, Oct. 2005. Acessado em 10/12/2013.

[20] Wikipedia. Predefinição: Satélite. http://pt.wikipedia.org/wiki/Predefini%C3%A7%C3%A3o:Sat%C3%A9lite, Oct. 2005. Acessado em 10/12/2013.

[21] Wikipedia. GeoHack. https://wiki.toolserver.org/view/GeoHack, 2012. Accessado em 19/07/2013.

[22] Wikipédia. Wikimedia database dumps. http://dumps.wikimedia.org/backup-index.html, 2014. Acessado em 10/05/2014.

[23] A. G. Woodruff and C. Plaunt. GIPSY : Automated Geographic Indexing of Text Documents. pages 1–21, 1994.

[24] Yahoo. Dmoz.org - the open directory project. http://www.dmoz.org, 1999. Acessado em: 10/01/2014.

Modelo de um sistema de conservação de alimentos baseado na IoT-A e seleção de elementos para Tabela de Decisão Adaptativa (TDA)

B. R. Kawano; R. M. Marè; B. C. C. Leite; C. E. Cugnasca[1]

***Abstract*— Supply chains of agricultural products are essential to the development of the Brazilian economy. Among the main exported products, we emphasize perishable agricultural products. Traceability of agricultural products is a requirement of consumers in developed countries and with respect to the collection, storage, transmission and provision of information related to products and processes they have undergone. Current systems do not allow full traceability of agricultural bulks, linking the final product in European supermarkets to the Brazilian farms. Therefore, this paper proposes the use of tools of Information and Communication Technology, in the Internet of Things context, for developing a traceability model for agricultural chains. As a result of this study, we propose a model of agricultural traceability products based on model reference architecture IoT-A. This model adds the SISTEMA MONITORAR®, enabling the monitoring, transport and storage of agricultural products throughout the chain by reducing the likelihood of human error and time involved with sampling. In addition, we present elements for building an adaptive decision table to control environmental conditions for food preservation. Thus, an increase in the competitiveness of the agricultural supply chain is expected, as well as meeting consumer's demands.**

***Keywords*— IoT-A; Food conservation; Adaptive Decision Table.**

I. INTRODUÇÃO

ATUALMENTE, o número de usuários da Internet vem crescendo exponencialmente devido ao surgimento de novas aplicações envolvendo *smartphones* e outros dispositivos móveis. Desde o início do século XXI, vive-se a chamada Terceira Era da Computação, ou o Terceiro Paradigma da Computação em que há uma pessoa para muitos computadores [1]. Segundo Mark Weiser em 1991, considerado o pai da Computação Ubíqua, o mundo iria vivenciar uma realidade de bilhões de dispositivos computacionais e uma das principais preocupações seria relativa à segurança e à privacidade dos dados manipulados [2].

Nesse sentido, a Internet das Coisas (*Internet of Things* - IoT) considera a conectividade de objetos do dia a dia em qualquer momento e em qualquer lugar [3]. Esse termo foi usado inicialmente por Kevin Ashton em uma apresentação feita em 1999 à Procter & Gamble, considerando a tecnologia de Identificação por Radiofrequência (RFID) na gestão da cadeia de suprimentos com conexão à Internet [4].

A ideia básica da IoT é a presença generalizada de coisas ou objetos em torno das pessoas que, por meio de um esquema único de endereçamento, são capazes de interagir uns com os outros e cooperar com seus vizinhos para alcançar objetivos comuns [5]. Segundo [6], ela abrangerá de 50 a 100 bilhões de dispositivos até 2020.

Este cenário considera não apenas as comunicações máquina-a-máquina (*Machine-to-Machine* – M2M), que poderá levar esse número potencial de objetos interconectados para 100 trilhões neste mesmo período [7]. A IoT considera a nomeação de todos os objetos que a ela encontram-se ligados, o que levará, segundo [8], a uma necessidade de intenso tráfego e transmissão de dados, como por exemplo a de sensores que coletam dados das mais diversas naturezas e aplicações.

Uma das linhas de pesquisa de aplicação da IoT é a cadeia agrícola que requer, em algumas etapas, o monitoramento das condições de armazenamento e transporte dos produtos agrícolas perecíveis. Para tanto, outras tecnologias precisam ser agregadas à tecnologia de identificação RFID, sendo uma das mais promissoras a Rede de Sensores Sem Fio (RSSF). Esse cenário favorece a melhoria da rastreabilidade de alimentos e a sua segurança alimentar [9].

Nas cadeias de suprimentos do agronegócio, a logística assume um papel relevante, devido às características particulares de cada produto e à sua perecibilidade, demandando cuidados e especialização nas etapas de transporte e armazenamento.

Visando-se agregar valor aos elos da cadeia, faz-se necessário não apenas um planejamento adequado às estratégias de cada organização, mas também a adoção da tecnologia da informação às suas etapas [10, 11].

Considerando-se a área de conservação de alimentos envolvendo uma câmara fria, este artigo se baseou no modelo de arquitetura de referência para IoT, denominado Internet of Things Architecture (IoT-A) para a concepção do modelo de conservação de alimentos. A IoT-A é apresentada por [12] e se baseia nos pré-requisitos conceituais e práticos da IoT como, por exemplo, escalabilidade, segurança, privacidade e interoperabilidade.

Neste trabalho, buscou-se inserir no contexto descrito, tópicos e elementos de Tecnologia Adaptativa [13,14]. A Tecnologia Adaptativa é relevante para se compreender o conceito de adaptatividade. Este termo é utilizado para representar uma característica de um dispositivo que tem a

[1] B. R. Kawano, Universidade de São Paulo (USP), São Paulo-SP, Brasil, bruno.kawano@usp.br

R. M. Marè, Universidade de São Paulo (USP), São Paulo-SP, Brasil, renata.mare@usp.br

B. C. C. Leite, Universidade de São Paulo (USP), São Paulo-SP, Brasil, brenda.leite@poli.usp.br

C. E. Cugnasca, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, carlos.cugnasca@poli.usp.br

capacidade de alterar sua própria estrutura interna de forma autônoma, sem que haja a interferência externa.

O estudo de [13], define um dispositivo adaptativo que detecta situações em que são necessárias ações de mudança de um determinado estado para outro estado mais estável ou desejado (pré-programado). Assim, o dispositivo tem a capacidade de reagir a mudanças necessárias para o devido funcionamento de um sistema, segundo variáveis de interesse. As condições ambientais de uma câmara fria destinada à conservação de alimentos consideradas neste trabalho são: temperatura do ar, umidade relativa, teor de gás carbônico, teor de gás oxigênio, teor de gás etileno (gás indicativo da maturação do produto) e carga térmica inicial dos alimentos.

Apresenta-se aqui a proposta de uma arquitetura de um sistema de conservação de alimentos visando à rastreabilidade desse processo em que haja o monitoramento contínuo de variáveis diversas, sendo estas as condições ambientes de uma câmara fria. Este modelo possui a capacidade de fornecer histórico de informações relevantes ao longo das cadeias de suprimentos no agronegócio por meio da *web*, somado à tecnologias adaptativas, por meio da Tabela de Decisão Adaptativa (TDA) que permitam o controle de forma eficiente de operação dos sistemas de conservação dos produtos agrícolas.

Assim, o controle das variáveis ambientais é proposto por meio de um sistema já concebido no Projeto Fapesp [15] chamado Sistema Monitorar®, a ser detalhado no item III deste artigo. Para tal, são considerados elementos adaptativos, a serem utilizados em uma TDA como forma de suporte e controle automático das definições dos níveis das variáveis de controle ambiental da câmara fria.

A estrutura do artigo está organizada da seguinte forma: no Item II são apresentados alguns conceitos e funcionalidades da IoT, bem como a arquitetura de referência IoT-A; o Item III apresenta o funcionamento do Sistema Monitorar®; o Item IV apresenta uma proposta de arquitetura do Sistema Monitorar® baseada no modelo de referência IoT-A e o Item V apresenta quais elementos são preponderantes para uma TDA, considerando-se uma câmara de conservação de alimentos.

II. INTERNET DAS COISAS E O MODELO DE REFERÊNCIA IOT-A

A IoT traz consigo alguns requisitos como escalabilidade, ou seja, o quão preparada uma rede está para suportar uma grande quantidade de dados sendo transmitidos. Outra característica é a interoperabilidade, que trata da capacidade de um sistema de agregar diferentes tecnologias e dispositivos, de forma a permitir que funcionem mesmo em um cenário heterogêneo.

Outros requisitos são preponderantes, como segurança e privacidade dos dados coletados. Considerando-se um ambiente onde sensores coletam dados privados, sendo transferidos e armazenados remotamente, estes requisitos são fundamentais para garantir um sistema de tráfego seguro e que atenda a um determinado modelo de referência para IoT.

Segundo [12] um modelo de referência de IoT permite uma abstração em alto nível para a concepção de um Modelo de Arquitetura de Referência, que permite que aspectos da arquitetura sejam considerados, como implementações e padrões já existentes.

Por exemplo, o modelo de referência IoT-A proposto e descrito em [12] destina-se a orientar a concepção de novos sistemas.

Nesta pesquisa ele será considerado na concepção do sistema de controle das variáveis ambientais da câmara fria. Os aspectos considerados neste modelo foram: a Visão Funcional, a Visão Informacional e a Visão de Desenvolvimento e Operação.

Na Visão Funcional, tem-se grupos funcionais como Comunicação, Serviços de IoT, Entidade Virtual, Gestão de Processos e Negócios em Internet das Coisas, Organização do Serviço, Segurança e Administração.

A Visão Informacional permite obter uma visão geral das estruturas de informação e de fluxos de informações dinâmicas como a forma de definir, gerenciar, armazenar e processar estas informações.

Por último, a Visão de Desenvolvimento e Operação permite aos usuários obterem, guias e manuais para diferentes *designs*. Nesta visão, são preponderantes três grupos de elementos como dispositivos, recursos e serviços. Em relação às escolhas de implementação, diversos aspectos devem ser considerados, a exemplo de: Redes de Sensores Sem Fio, RFID, *WiFi* ou outras tecnologias e Redes de Celulares. Outros aspectos devem fazer parte da modelagem do sistema como quesitos de conectividade.

Neste sentido, durante o desenvolvimento deste artigo, serão levados em consideração todos estes aspectos citados para a concepção do Modelo de Referência para Arquitetura de IoT.

III. SISTEMA MONITORAR®

O Sistema Monitorar® [15] é um software baseado em Internet, que tem por objetivo permitir o monitoramento remoto, contínuo e seguro de variáveis diversas associadas a ambientes ou processos, provendo informações confiáveis como suporte à gestão do objeto monitorado. É fruto de um projeto de Pesquisa Inovativa na Pequena Empresa – PIPE, fomentado pela Fundação de Amparo à Pesquisa do Estado de São Paulo – Fapesp [16] Financiadora de Estudos e Projetos – FINEP. O seu protótipo encontra-se em operação contínua desde 2009, em duas salas de aula no edifício de Engenharia Civil da Escola Politécnica da USP.

O cenário genérico de aplicação do Sistema Monitorar® é um ambiente (edificação, container ou meio de transporte) provido de uma rede de sensores adequados às variáveis de interesse. Por meio de um *gateway* conectado à Internet (Ethernet, GPRS), os dados são transmitidos continuamente a um servidor *web*, onde são armazenados e processados.

Na Figura 1, encontra-se uma interface customizada em que é possível a visualização de usuários autorizados, por exemplo, o comprador da carga agrícola que deseja acompanhar parâmetros de qualidade do ambiente de transporte e conservação. É possível também a consulta dos dados por meio de dispositivos conectados à Internet, acessando gráficos

diversos, *dashboards* de valores instantâneos, ou mesmo recebendo alarmes de valores críticos (SMS, e-mail).

Figura 1. Exemplo de gráfico do Sistema Monitorar® (umidade relativa do ar)

O Sistema Monitorar® atende aos seguintes requisitos:

- Segurança de dados;
- Permitir o uso por diversos clientes;
- Permitir o acesso a mais de um usuário por cliente, com a devida autenticação;
- Receber e armazenar dados distintos de clientes distintos (histórico de dados);
- Receber dados provenientes de redes de sensores com tecnologias e arquiteturas diversas;
- Receber dados de sensores, independentemente de fabricante ou tecnologia de rede;
- Oferecer interfaces customizadas por cliente;
- Permitir a geração de alarmes customizados por cliente.

Para tal, um conjunto de nove aplicativos atuam coordenadamente (Figura 2).

A troca de dados entre eles ocorre por meio de *web services* Restful [17] tecnologia que, dentre outras vantagens, proporciona pacotes de dados de tamanho reduzido, característica importante em um sistema com grande número de troca de mensagens. Descrevem-se a seguir os nove aplicativos.

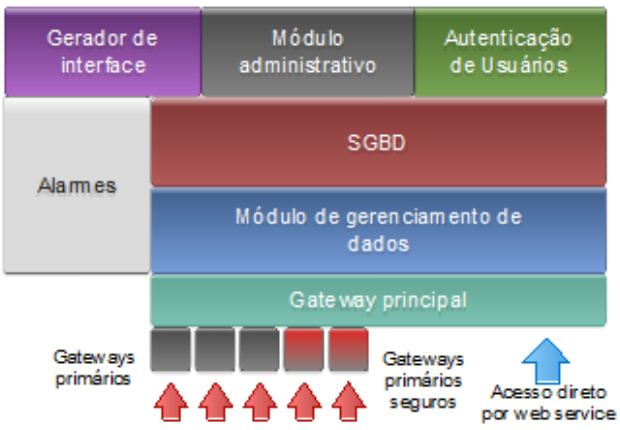


Figura 2. Sistema Monitorar®: arquitetura simplificada.
Fonte: Os autores.

A. *Gateways primários*

Para atender o requisito de receber dados de redes de sensores formadas por equipamentos diversos, é necessária a criação de soluções de software específicas já que cada equipamento conecta-se de maneira distinta, usando um protocolo distinto. Utilizam-se pequenos módulos de software que permitem a conexão a *gateways* ou equipamentos remotos distintos.

Tipicamente a codificação de um destes módulos é da ordem de 20 a 30 linhas de código. Uma vez criado, o módulo pode ser utilizado com outras redes que utilizem o mesmo equipamento. A segunda importante tarefa dos *gateways* primários é codificar os dados recebidos em formato JSON [18] e enviar o pacote de dados ao *gateway* principal.

B. *Gateways primários seguros*

Sua funcionalidade é idêntica à dos *gateways* primários. Todavia, incorporam as funções da biblioteca OpenSSL [19] para autenticação bi-lateral, tráfego criptografado e verificação de certificados digitais.

C. *Gateway principal*

Ao contrário dos *gateways* primários, o protocolo e o formato dos dados recebidos pelo *gateway* principal é único. Sua função é analisar os dados recebidos, identificar o cliente, ambiente e pontos monitorados e encaminhar todas estas informações ao módulo do Sistema Gerenciador de Banco de Dados (SGBD).

D. *Módulo de gerenciamento de dados*

Este módulo recebe os pacotes de dados do *gateway* principal. Suas funções são:

- Identificar o cliente que gerou o pacote;
- Identificar o ambiente monitorado;
- Identificar o ponto monitorado;
- Identificar a grandeza monitorada, seu valor e unidade;
- Montar e enviar um pacote de dados aos módulos SGBD e Alarmes.

As funções de identificação de clientes, ambientes e pontos monitorados são possíveis graças à passagem de chaves de identificação, parâmetros criados na configuração dos *gateways* primários e definidos na administração do Sistema Monitorar®.

E. *Módulo de alarmes*

O módulo de alarmes recebe pacotes do módulo de gerenciamento e verifica a necessidade de emissão de alarmes. Para tal, compara os valores medidos com as condições armazenadas no SGBD. Se necessária a emissão de alarmes, o módulo identifica os usuários destinatários, e realiza o envio das mensagens pelo canal especificado (e-mail e/ou SMS).

F. *Sistema Gerenciador de Banco de Dados (SGBD)*

Armazena todos os dados do Sistema Monitorar®. Comunica-se com o módulo gerenciador de dados (recebendo pacotes de dados para armazenamento), com o módulo

administrativo (recebendo dados de novos clientes, usuários, ambientes e pontos monitorados).

Também comunica-se com o módulo de autenticação de usuários (verificando a validade dos dados de *login*), com o gerador de interfaces (fornecendo os dados de telas das interfaces gráficas disponíveis) e com o módulo de alarmes (guardando as condições pré-definidas para a emissão de alarmes). O módulo utiliza o software MySQL V.5.5 [20] em um servidor dedicado na infraestrutura da Amazon Web Services [21], dado o número elevado de requisições a que está sujeito.

G. Módulo de autenticação de usuários

Solicita dados de autenticação ao usuário para determinar as condições de acesso ao Sistema Monitorar®. A autenticação se realiza por digitação de par usuário/senha. Opcionalmente pode-se exigir código gerado por dispositivo *token*. Para esta funcionalidade, utiliza-se o software Google Authenticator [22] que permite a utilização de dispositivos móveis com sistemas operacionais Android ou iOS para gerar os códigos.

H. Módulo administrativo

Permite o cadastramento de novos clientes, usuários, ambientes e pontos monitorados. Permite também a definição das grandezas monitoradas por cliente e as condições para emissão de alarmes (parametrização).

I. Gerador de interfaces

Armazena os *layouts* de tela específicos para cada cliente, definindo, para cada tipo de página disponível, os mecanismos de visualização (gráficos, medidores instantâneos, etc), seus parâmetros visuais (cores, estilos, etc) e sua disposição na tela.

Há também a possibilidade de sua interface com sistemas de automação (legados ou não) ou seja, o sistema pode fornecer informações complementares às pré-existentes, conferindo maior granularidade à atuação. Na solução aqui proposta, essas características são particularmente interessantes, visto proporcionarem maior liberdade ao cliente no aproveitamento de infraestrutura prévia bem como, na escolha para aquisição de novos equipamentos.

Outra importante característica deste sistema, a segurança de dados, é proporcionada pelos seguintes aspectos:

- Autenticação, que garante a identidade dos usuários legítimos do sistema, impedindo o acesso dos demais;
- Confidencialidade, que garante que os dados coletados e armazenados não sejam acessados por terceiros;
- Integridade, que garante que os dados não sejam modificados durante o tráfego pela rede;
- Autorização, que garante que os usuários possam acessar somente a parcela dos dados a eles disponibilizados por uma administração central;
- Não-repudiação de origem dos dados (em implantação), que implica na assinatura digital dos pacotes de dados, utilizando-se certificados digitais emitidos por autoridade certificadora pertencente à estrutura da ICP-Brasil [23].

Figura 3: Arquitetura do sistema de conservação de alimentos baseado no modelo de referência IoT-A.
Fonte: Os autores.

No que concerne às cadeias de suprimentos, o conjunto destas propriedades pode proporcionar a redução de perdas bem como, a rastreabilidade das etapas monitoradas, fornecendo histórico confiável aos usuários interessados e/ou a sistemas externos, inclusive compatíveis com a arquitetura EPCglobal® [24]. A utilização da Internet como canal de troca de dados permite o monitoramento de ambientes de difícil acesso, viabilizando a consulta dos dados por usuários em praticamente qualquer localidade e horário.

IV. PROPOSTA DE ARQUITETURA DE UM SISTEMA DE CONSERVAÇÃO DE ALIMENTOS PARA RASTREABILIDADE

O modelo de arquitetura proposto na Figura 3, corresponde às bases do modelo de referência IoT-A considerando o Sistema Monitorar®, que se situa na camada de aplicação. Um sistema de atuação (também na camada de aplicação), ligado ao Sistema Monitorar®, permitiria o ajuste automático das condições ambientais de armazenamento.

O bloco nomeado como Segurança Monitorar® corresponde a todos os itens relacionados à segurança e privacidade do sistema, nos quais elementos como Autenticação, Autorização, Integridade, Confidencialidade e a Não-repudiação de dados e informações transmitidas são pré-requisitos. O bloco Gerenciamento, integra todas as características de Configuração, Controle de Falhas, Alarmes e de Desempenho do Sistema.

Na parte inferior da Figura 3, encontra-se a camada dos dispositivos que coletam os dados de controle ambiental do objeto em questão, no caso a câmara fria. Fazem parte dessa camada, sensores de temperatura, umidade relativa do ar, gás carbônico, oxigênio e de etileno, que irão coletar os dados e transmiti-los, seguindo a camada de pilha de protocolos do sistema.

Os dispositivos Gateways, possuem o papel de intermediar ou "traduzir" os dados coletados e transmiti-los até a camada de aplicação, no caso para o software Monitorar®. Neste modelo, considera-se que há um módulo de atuação do controle das condições ambientais baseada em uma tecnologia adaptativa. Como será apresentado neste trabalho, selecionaram-se elementos que possam compor a TDA, considerados fundamentais para a aplicação em questão, juntamente com os

outros elementos, para o controle autônomo e sem interferências externas para atuação do sistema.

V. ELEMENTOS DE UMA TABELA DE DECISÃO ADAPTATIVA (TDA) PARA UMA CÂMARA DE CONSERVAÇÃO DE ALIMENTOS

Adotando-se como base os elementos envolvidos na conservação de alimentos em uma câmara fria, pode-se conceber a proposta de um modelo de tomada de decisão, baseada no conceito de TDA.

Segundo [13], dispositivos adaptativos dirigidos por regras podem dar maior flexibilidade às tabelas de decisão básicas, e a partir disso, podem ser geradas inclusões e exclusões de regras durante a operação do sistema.

Para a realização das ações do sistema, propõe-se a utilização, neste modelo, da TDA, a partir da qual é possível se estabelecer e designar regras, funções e ações desejadas para que o sistema possa atuar de forma autônoma e sem interferência externa.

Assim, é apresentado na Figura 4, um esquema que representa o modelo de uma TDA cuja descrição detalhada pode ser melhor compreendida no trabalho de [13].

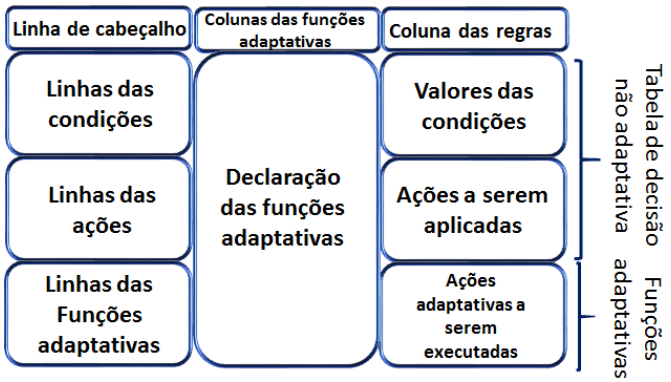


Figura 4: Elementos da tabela de decisão básica e adaptativa.
Fonte: Adaptado de [13].

Em relação aos elementos adaptativos da TDA, no campo das Linhas de Funções Adaptativas, há os elementos de funções adaptativas como os nomes das funções, parâmetros, variáveis e geradores das funções adaptativas.

Já no campo das Ações Adaptativas a Serem Executadas, são inseridos os valores dos parâmetros, geradores e variáveis necessários para a execução do sistema.

Neste trabalho, serão considerados apenas os levantamentos de algumas funções e ações adaptativas que poderão alimentar a TDA.

Independentemente do alimento considerado, alguns fatores passíveis de controle são relevantes para a conservação de alimentos em câmaras frias, tais como:

- Temperatura do ar;
- Umidade relativa do ar;
- Teor de gás carbônico;
- Teor de gás oxigênio;
- Teor de gás etileno (gás indicativo da maturação);
- Carga térmica inicial dos alimentos.

Todos estes elementos necessitam de controle, a fim de que haja uma correta preservação do alimento, considerando-se quesitos de qualidade como:

- Aparência (tamanho, forma, cor);
- Textura (firmeza, suculência);
- Flavor (acidez, adstringência);
- Nutritivo (teor de carboidratos, proteínas, lipídeos, vitaminas, minerais, água);
- Segurança (resíduos, micotoxinas, microrganismos patogênicos).

A Tabela 1 apresenta informações de possíves funções e valores de condições adaptativas, que poderiam ser adotados como padrão para controle de intervalos de temperatura ideais de conservação de diversos tipos de frutas e legumes. Além disso, o sistema poderia ser configurado para controlar os limites mínimos de temperatura dos intervalos apresentados, a partir dos quais um alerta poderia ser acionado, permitindo ao sistema reconhecer automaticamente a necessidade de manutenção da temperatura dentro dos níveis ideais.

Tabela 1: Pontos de congelamento e intervalos de temperatura de armazenamento ideais para diversos frutos, adaptado de [25].

Fruto	Pontos de Congelamento (°C)	Temperatura de Armazenamento (°C)
Abacate	-0,3	4,5 a 13,0
Banana	-0,7	13,0 a 14,0
Berinjela	-0,8	8,0 a 12,0
Caqui	-2,1	-1,0
Figo	-2,4	-0,5 a 0,0
Limão	-1,4	12,0 a 14,0
Laranja	-1,2	3,0 a 9,0
Maçã	-1,5	-1,0 a 4,0
Manga	-0,9	13,0
Pepino	-0,5	10,0 a 13,0
Pêssego	-0,9	-0,5 a 0,0

A partir dessas referências poderia se estabelecer e programar, por meio de simulações, as ações adaptativas desejadas para o sistema, bem como, os geradores necessários para as suas execuções.

Feitas essas simulações iniciais, levando-se em conta apenas a temperatura do ar, as referências para os demais parâmetros passíveis de controle seriam levantadas e incorporadas à TDA. Isto permitiria análises mais complexas e aderentes à realidade, onde seriam realizadas diversas combinações de fatores que levariam a ações distintas, visando-se à melhor qualidade dos produtos.

Aos quesitos de qualidade, também poderiam ser atribuídas relevâncias distintas, por exemplo, de acordo com o mercado alvo (existem vários tipos de mercado interno, além do mercado tipo exportação), incluindo-se também estas condições na TDA.

VI. CONCLUSÕES

Considerando-se o modelo proposto, há potencial de inovação e aplicação do sistema de conservação de alimentos

baseado na arquitetura de referência IoT-A. Esses elementos se sustentam devido à abordagem simultânea da aplicação das tecnologias adaptativas e conceitos de IoT às cadeias produtivas do agronegócio, mais particularmente à etapa de armazenamento de produtos.

Acredita-se que estratégias de operação dos sistemas de refrigeração adaptáveis ao contexto, levando em conta o tipo de produto, suas características particulares, condições do ambiente interior (armazenamento) e condições climáticas, proporcionem a redução otimizada de perdas, contribuindo para a maior competitividade da cadeia como um todo. Ressalta-se que este modelo pode ser estendido para sistemas embarcados, logicamente com as devidas adaptações, a fim de que dados de qualidade de uma carga de produtos agrícolas possam ser coletados ao longo de um trajeto em seu transporte. Além disso, a TDA demonstra elevado potencial no auxílio de manutenção de um sistema de conservação de alimentos, aplicada, por exemplo, a uma câmara fria.

Como trabalho futuro, pretende-se inserir todas as variáveis necessárias para o modelo da TDA, a fim de que seja possível uma simulação do sistema proposto. Deste modo, será possível realizar-se o controle das condições ambientais de uma câmara fria de uma forma rápida, inteligente e fazendo-se o uso racional da energia em processos de conservação de alimentos.

AGRADECIMENTOS

Os autores agradecem: ao Conselho Nacional de Desenvolvimento Científico e Tecnológico, auxílio nº 161109/2013-6, ao Instituto de Transporte e Logística (ITL) pelas bolsas de doutorado concedidas, à Fundação de Amparo à Pesquisa do Estado de São Paulo e à Financiadora de Estudos e Projetos pelo apoio ao projeto de desenvolvimento do Sistema Monitorar®.

REFERÊNCIAS

[1] Weiser, M. The computer of 21st century. Pervasive Computing, 1991.

[2] Weiser, M. "Some Computer Science Problems in Ubiquitous Computing,". Communications of the ACM, July 1993. (reprinted as "Ubiquitous Computing". Nikkei Electronics; December 6, 1993; pp. 137-143.)

[3] ITU Report. The Internet of Things. International Telecommunication Union, 2005.

[4] Ashton, K. That 'Internet of Things' Things in the real world, things matter more than ideas. Disponível em : http://www.rfidjournal.com/articles/view?4986. Junho 2009.

[5] Giusto, D.; Iera, A.; Morabito, G.; Atzori, L. The Internet of Things, Springer, 2010.

[6] Soundmaeker, H. Vision and challenges for realising the Internet of Things. EUR-OP, 2010

[7] Cross-ETP Vision Document. Disponível em <http://www.future-internet.eu/fileadmin/documents/reports/Cross-ETPs_FI_Vision_Document_v1_0.pdf>.

[8] Elkhodr, M.; Shahrestani, S.; Cheung, H. The Internet of Things: Vision & Challenges. Proceeding of the IEEE Tencon Spring 2013 Conference. IEEE, February 17, 2013.

[9] Hirai, W. G. Food Security and Sustainability. Global Social Transformation and Social Action: The role of the social workers. Social Work-Social Development, vol. 3, p.47-50, 2014.

[10] Kawano, B.R., Mores, G.V., Silva, R.F., Cugnasca, C.E. "Estratégias para Resolução dos Principais Desafios da Logística de Produtos Agrícolas Exportados pelo Brasil". *Revista de Economia e Agronegócio*, v. 10, n. 1, p. 71-88, 2012.

[11] Protil, R.M.; Souza, A.B. "Diagnóstico de Tecnologia da Informação nas Cooperativas Agropecuárias do Paraná". XII Congresso do Saber-Sociedade Brasileira de Economia e Sociologia Rural. Ribeirão Preto, SP: Anais... 2005.

[12] Joachin, W.; Walewski, S. Internet of Things Architecture IoT-A. Deliverable D1.4. Converged archictural reference model for the IoT, v.2.0, 2012.

[13] Tchemra, A. H. Tabela de decisão adaptativa na tomada de decisão multicritério. Tese apresentada na Universidade de São Paulo, 2009.

[14] José Neto, J. Um levantamento da evolução da adaptatividade e da tecnologias adaptativa. Revista IEEE, América Latina, v. 5, p. 496-505, 2007.

[15] Sistema Monitorar. Disponível em: <http://www.minotorar.net.br>. Acesso em 01 jan de 2015.

[16] Projeto PIPE Fapesp. Disponível em: <http://www.fapesp.br/58>. Acesso em 03 de jan de 2015.

[17] Pautasso, C.; Zimmermann, O.; Leymann, F. "Restful Web Services Vs. "big"' Web Services: Making the Right Architectural Decision". Proceedings of the 17th international conference on World Wide Web, v. n. p. 805-814, 2008.

[18] Bray, T. The Javascript Object Notation (Json) Data Interchange Format. IETF, Internet Engineerign Task Force. Disponível em: <https://tools.ietf.org/html/rfc7159>. Acesso em: 10/04/2014, 2014.

[19] OpenSSL Software Foundation Openssl: The Open Source Toolkit for Ssl/tls. Adamstown: Disponível em: <http://www.openssl.org>. Acesso em: 18 ago. 2010, 2010.

[20] Oracle Mysql Ab :: The World's Most Popular Open Source Database. Uppsala: Disponível em: <http://www.mysql.com>. Acesso em: 01 Ago. 2007, 2007.

[21] Amazon Web Services. Disponível em: <http://aws.amazon.com.pt>. Acesso em 03 de jan de 2015.

[22] Google Authenticator. Disponível em: <http://support.google.com/accounts/answer/1066447?hl=pt-BR>. Acesso em 03 de jan de 2015.

[23] ITI – Instituto Nacional de Tecnologia da Informação. Certificado Conceitos. Disponível em: <http://www.icpbrasil.gov.br/twiki/bin/view/Certificacao/CertificadoConceitos>. Acesso em: Jul. 2010

[24] GS1. The EPCglobal Architecture Framework Version 1.4. Brussels: 2010. Disponível em: <http://www.gs1.org/gsmp/kc/epcglobal/architecture/architecture_1_4-framework-20101215.pdf>. Acesso em: 10 Mai. 2012

[25] Walter, E. H. Bioquímica e Fisiologia do Desenvolvimento e Pós-colheita de Frutas e Hortaliças. Universidade Federal do Pampa, 2010.

Bruno Rógora Kawano é graduado em Engenharia Agronômica pela Escola Superior de Agricultura "Luiz de Queiroz" da Universidade de São Paulo (ESALQ-USP) em 2010. Realizou intercâmbio acadêmico na École Supérieure d´Agriculture d´Angers – France em 2010. Obteve o título de mestre em Planejamento de Sistemas Energéticos pela Universidade Estadual de Campinas (UNICAMP), em 2013. Atualmente é doutorando no Programa de Pós-Graduação em Engenharia Elétrica – PPGEE na Escola Politécnica da Universidade de São Paulo (EPUSP) no Laboratório de Automação Agrícola (LAA).

Renata Maria Marè é graduada em Engenharia Civil pela Escola de Engenharia Mauá (1985), mestre em Engenharia de Construção Civil e Urbana (2010) e doutoranda em Engenharia de Computação e Sistemas Digitais (desde 2013), ambos pela Escola Politécnica da USP (EPUSP). Possui MBA em Gestão Empresarial pelo Instituto Mauá de Tecnologia (2011). Suas principais áreas de pesquisas são: Sustentabilidade em Edificações, Edifícios Inteligentes, Cidades Inteligentes, Redes de Sensores para Monitoramento.

Brenda Chaves Coelho Leite possui Doutorado em Engenharia Mecânica (2003), Mestrado em Arquitetura e Urbanismo (1997) pela Universidade de São Paulo - Brasil, e graduação em Engenharia Civil pela Universidade Federal de Minas Gerais (1979). Atualmente é Professor Doutor no Departamento de Engenharia de Construção Civil da Escola Politécnica da Universidade de São Paulo. Tem experiência na área de conforto térmico e qualidade do ar em edifícios, Sistemas de HVAC, Tecnologias para distribuição de ar pelo piso, conforto térmico individualizado e painéis radiantes; teto verde; simulação de desempenho energético de edifícios; Dinâmica de Fluidos Computacional (CFD); Sistemas de Automação e controle aplicados à climatização de ambientes.

Carlos Eduardo Cugnasca é graduado em Engenharia de Eletricidade (1980), mestre em Engenharia Elétrica (1988) e doutor em Engenharia Elétrica (1993) pela Escola Politécnica da USP (EPUSP). É livre-docente (2002) pela EPUSP. Atualmente, é Professor Associado 3 da EPUSP e pesquisador do Laboratório de Automação Agrícola (LAA) do Departamento de Engenharia de Computação e Sistemas Digitais (PCS) da EPUSP. Tem experiência na área de Supervisão e Controle de Processos e Instrumentação, aplicadas a processos agrícolas e Agricultura de Precisão, atuando principalmente nos seguintes temas: instrumentação inteligente, sistemas embarcados em máquinas agrícolas, monitoração e controle de ambientes protegidos, redes de controle baseados nos padrões CAN, ISO11783 e LonWorks, Redes de Sensores Sem Fio e computação pervasiva. É editor da Revista Brasileira de Agroinformática (RBIAgro).

Semântica no Reconhecedor Gramatical Linguístico

A. T. Contier, D. Padovani e J. J. Neto

***Resumo*— Este trabalho faz uma breve revisão dos conceitos de Tecnologia Adaptativa, apresentando seu mecanismo de funcionamento e seus principais campos de aplicação, destacando o forte potencial de sua utilização no processamento de linguagens naturais. Em seguida são apresentados os conceitos de processamento de linguagem natural, ressaltando seu intricado comportamento estrutural. Por fim, é apresentada uma proposta para incorporar semântica no reconhecedor gramatical Linguístico.**

***Palavras Chave* — Autômatos Adaptativos, Processamento de Linguagem Natural, Reconhecedores Gramaticais, Gramáticas Livres de Contexto.**

I. Autômatos Adaptativos

O AUTÔMATO adaptativo é uma máquina de estados à qual são impostas sucessivas alterações resultantes da aplicação de ações adaptativas associadas às regras de transições executadas pelo autômato [1]. Dessa maneira, estados e transições podem ser eliminados ou incorporados ao autômato em decorrência de cada um dos passos executados durante a análise da entrada. De maneira geral, pode-se dizer que o autômato adaptativo é formado por um dispositivo convencional, não adaptativo, e um conjunto de mecanismos adaptativos responsáveis pela auto modificação do sistema.

O dispositivo convencional pode ser uma gramática, um autômato, ou qualquer outro dispositivo que respeite um conjunto finito de regras estáticas. Este dispositivo possui uma coleção de regras, usualmente na forma de cláusulas if-then, que testam a situação corrente em relação a uma configuração específica e levam o dispositivo à sua próxima situação. Se nenhuma regra é aplicável, uma condição de erro é reportada e a operação do dispositivo, descontinuada. Se houver uma única regra aplicável à situação corrente, a próxima situação do dispositivo é determinada pela regra em questão. Se houver mais de uma regra aderente à situação corrente do dispositivo, as diversas possíveis situações seguintes são tratadas em paralelo e o dispositivo exibirá uma operação não determinística.

Os mecanismos adaptativos são formados por três tipos de ações adaptativas elementares: consulta (inspeção do conjunto de regras que define o dispositivo), exclusão (remoção de alguma regra) e inclusão (adição de uma nova regra).

A. Contier, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, contier@gmail.com

D. Padovani, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, djalmapadovani@gmail.com

J. J. Neto, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, joao.jose@poli.usp.br

Autômatos adaptativos apresentam forte potencial de aplicação ao processamento de linguagens naturais, devido à facilidade com que permitem representar fenômenos linguísticos complexos tais como dependências de contexto. Adicionalmente, podem ser implementados como um formalismo de reconhecimento, o que permite seu uso no pré-processamento de textos para diversos usos, tais como: análise sintática, verificação de sintaxe, processamento para traduções automáticas, interpretação de texto, corretores gramaticais e base para construção de sistemas de busca semântica e de aprendizado de línguas auxiliados por computador.

Diversos trabalhos confirmam a viabilidade prática da utilização de autômatos adaptativos para processamento da linguagem natural. É o caso, por exemplo, de [2], que mostra a utilização de autômatos adaptativos na fase de análise sintática; [3] que apresenta um método de construção de um analisador morfológico e [4], que apresenta uma proposta de autômato adaptativo para reconhecimento de anáforas pronominais segundo algoritmo de Mitkov.

II. Processamento da Linguagem Natural: Revisão da Literatura

O processamento da linguagem natural requer o desenvolvimento de programas que sejam capazes de determinar e interpretar a estrutura das sentenças em muitos níveis de detalhe. As linguagens naturais exibem um intricado comportamento estrutural visto que são profusos os casos particulares a serem considerados. Uma vez que as linguagens naturais nunca são formalmente projetadas, suas regras sintáticas não são simples nem tampouco óbvias e tornam, portanto, complexo o seu processamento computacional. Muitos métodos são empregados em sistemas de processamento de linguagem natural, adotando diferentes paradigmas, tais como métodos exatos, aproximados, pré-definidos ou interativos, inteligentes ou algorítmicos [5]. Independentemente do método utilizado, o processamento da linguagem natural envolve as operações de análise léxico-morfológica, análise sintática, análise semântica e análise pragmática [6]. A análise léxico-morfológica procura atribuir uma classificação morfológica a cada palavra da sentença, a partir das informações armazenadas no léxico [7]. O léxico ou dicionário é a estrutura de dados contendo os itens lexicais e as informações correspondentes a estes itens. Entre as informações associadas aos itens lexicais, encontram-se a categoria gramatical do item, tais como substantivo, verbo e adjetivo, e os valores morfossintático-semânticos, tais como gênero, número, grau, pessoa, tempo, modo, regência verbal ou nominal. Na etapa de análise sintática, o analisador verifica se uma sequência de palavras constitui uma frase válida da língua, reconhecendo-a ou não. O analisador sintático faz uso

de um léxico e de uma gramática, que define as regras de combinação dos itens na formação das frases. Nos casos nos quais há a necessidade de interpretar o significado de um texto, a análise léxico-morfológica e a análise sintática não são suficientes, sendo necessário realizar um novo tipo de operação, denominada análise semântica [7]. Na análise semântica procura-se mapear a estrutura sintática para o domínio da aplicação, fazendo com que a estrutura ganhe um significado. O mapeamento é feito identificando as propriedades semânticas do léxico e o relacionamento semântico entre os itens que o compõe [8]. Já a análise pragmática procura reinterpretar a estrutura que representa o que foi dito para determinar o que realmente se quis dizer. Inserem-se nessa categoria as relações anafóricas, correferências, determinações, focos ou temas, dêiticos e elipses [9].

Em [10] são apresentados trabalhos de pesquisas em processamento de linguagem natural para a Língua Portuguesa tais como o desenvolvido pelo Núcleo Interinstitucional de Linguística Aplicada (NILC) no desenvolvimento de ferramentas para processamento de linguagem natural; o projeto VISL – Visual Interactive Syntax Learning, sediado na Universidade do Sul da Dinamarca, que engloba o desenvolvimento de analisadores morfossintáticos para diversas línguas, entre as quais o português; e o trabalho de resolução de anáforas desenvolvido pela Universidade de Santa Catarina. A tecnologia adaptativa também tem contribuído com trabalhos em processamento da linguagem natural. Em [11], são apresentadas algumas das pesquisas desenvolvidas pelo Laboratório de Linguagens e Tecnologia Adaptativa da Escola Politécnica da Universidade de São Paulo: um etiquetador morfológico, um estudo sobre processos de análise sintática, modelos para tratamento de não determinismos e ambiguidades, e um tradutor texto voz baseado em autômatos adaptativos.

III. RECONHECEDOR ADAPTATIVO: SUPORTE TEÓRICO LINGUÍSTICO

A Moderna Gramática Brasileira de Celso Luft [12] foi escolhida como suporte teórico linguístico do reconhecedor aqui proposto. A escolha foi feita em função da forma clara e precisa com que Luft categoriza os diversos tipos de sentenças de língua portuguesa, diferenciando-se das demais gramáticas que priorizam a descrição da língua em detrimento da análise estrutural da mesma. Luft diz que a oração é moldada por padrões frasais ou oracionais, compostos por elementos denominados sintagmas (Tabela 1).

TABELA 1.
ELEMENTOS FORMADORES DE SINTAGMAS

Sintagmas	
Substantivo	Quantitativos+Pronomes Adjetivos+ Sintagma Adjetivo1+Substantivo+ Sintagma Adjetivo2+ Sintagma Preposicional+ Oração Adjetiva
Verbal	Pré-verbais+ Verbo Auxiliar+ Verbo Principal

TABELA 1.
CONTINUAÇÃO

Sintagmas	
Adjetivo	Advérbio de Intensidade+ Adjetivo+ Sintagma Preposicional
Adverbial	Advérbio de Intensidade+ Adverbio+ Sintagma Preposicional
Preposicional	Preposição+ Sintagma Substantivo

Sintagma é qualquer constituinte imediato da oração, podendo exercer papel de sujeito, complemento (objeto direto e indireto), predicativo e adjunto adverbial. É composto por uma ou mais palavras, sendo que uma é classificada como núcleo e as demais como dependentes. As palavras dependentes podem estar localizadas à esquerda ou à direita do núcleo. Luft utiliza os seguintes nomes e abreviaturas:

1. Sintagma substantivo (SS): núcleo é um substantivo;
2. Sintagma verbal (SV): núcleo é um verbo;
3. Sintagma adjetivo (Sadj): núcleo é um adjetivo;
4. Sintagma adverbial (Sadv): núcleo é um advérbio;
5. Sintagma preposicional (SP): é formado por uma preposição (Prep) mais um SS.
6. Vlig: verbo de ligação
7. Vi: verbo intransitivo
8. Vtd: verbo transitivo direto
9. Vti: verbo transitivo indireto
10. Vtdi: verbo transitivo direto e indireto
11. Vt-pred: verbo transitivo predicativo

Um padrão oracional é determinado pelos tipos de sintagmas e pela sequência em que aparecem. Por exemplo, o padrão oracional SS Vlig SS, indica que a frase é composta por um sintagma substantivo, seguido de um verbo de ligação e de outro sintagma substantivo. Os padrões são classificados em cinco tipos (Tabela 2):

1. Padrões pessoais nominais: Neste caso, existe sujeito e o núcleo do predicado é um nome (substantivo, adjetivo, advérbio) ou um pronome (substantivo, adjetivo, advérbio). O verbo, nesses casos, é chamado de verbo de ligação (Vlig).
2. Padrões pessoais verbais: São aqueles nos quais existe o sujeito e o núcleo do predicado é um verbo. O verbo pode ser transitivo direto (Vtd), transitivo indireto (Vti), transitivo direto e indireto (Vtdi), e intransitivo (Vi). Se o verbo for transitivo direto (Vtd), o complemento será um objeto direto; se o verbo for transitivo indireto (Vti), o complemento será um objeto indireto; se o verbo for transitivo direto e indireto (Vtdi), o complemento será um objeto direto e um indireto; se o verbo for intransitivo (Vi), não há complemento.
3. Padrões Pessoais Verbo-Nominais: Neste caso, existe o sujeito e o núcleo do predicado é um verbo transitivo predicativo (Vt-pred), cujo complemento é um objeto direto e um predicativo do objeto.

4. Padrões Impessoais Nominais: Ocorrem quando não existe sujeito e o núcleo do predicado é um nome (substantivo, adjetivo, advérbio) ou um pronome (substantivo, adjetivo, advérbio).
5. Padrões Impessoais Verbais: Neste caso, não existe sujeito e o núcleo do predicado é um verbo.

TABELA 2
PADRÕES ORACIONAIS DE LUFT

Padrões Pessoais Nominais				
SS	Vlig	SS		
SS	Vlig	Sadj		
SS	Vlig	Sadv		
SS	Vlig	SP		
Padrões Pessoais Verbais				
SS	Vtd	SS		
SS	Vti	SP		
SS	Vti	Sadv		
SS	Vti	SP	SP	
SS	Vtdi	SS	SP	
SS	Vtdi	SS	Sadv	
SS	Vtdi	SS	SP	SP
SS	Vi			
Padrões Pessoais Verbo-Nominais				
SS	Vtpred	SS	SS	
SS	Vtpred	SS	Sadj	
SS	Vtpred	SS	SP	
SS	Vtpred	SS	Sadv	
SS	Vtpred	SS		
SS	Vtpred	Sadj		
SS	Vtpred	SP		
Padrões Impessoais Nominais				
	Vlig	SS		
	Vlig	Sadj		
	Vlig	Sadv		
	Vlig	SP		
Padrões Impessoais Verbais				
	Vtd	SS		
	Vti	SP		
	Vi			

Os sintagmas e os padrões oracionais de Luft foram mapeados em uma gramática para que pudessem ser processados computacionalmente, ficando da seguinte forma:

F → [Conec] [SS] SV [Conec]
Conec → F
SS → [Sadj] SS [Sadj | SP]
SS → [Quant | PrA] (Sc | Sp | PrPes)
SV→ [Neg] [Aux | PreV] (Vlig | Vtd| Vti | Vtdi| Vi)
[SS | Sadj | Sadv | SP] [SS| Sadj| Sadv| SP] [SP]
SP → Prep (SS | Sadj)
Sadj → Sadj [SP]
Sadj →[Adv] Adj
Sadv → Sadv [SP]
Sadv → [Adv] Adv
PrA → Ind | ArtDef | ArtInd| Dem| Pos

Sendo:
F – Frase
SS – Sintagma substantivo
SV – Sintagma verbal
SP – Sintagma preposicional
SN – Sintagma nominal
Sadv – Sintagma adverbial
Sadj – Sintagma adjetivo
Adv – adverbio
Adj – adjetivo
ArtDef – artigo definido
ArtInd – artigo indefinido
Aux – Partícula auxiliar (apassivadora ou pré-verbal)
Conec – Conector (conjunção ou pronome relativo)
Dem – pronome demonstrativo indefinido
Ind – pronome indefinido
Neg – partícula (negação)
PrA – pronome adjetivo
PrPes – pronome pessoal
Prep – preposição
Quant – numeral
Sc – substantivo comum
Sp – substantivo próprio
V– verbo
Vlig – verbo de ligação
Vi – verbo intransitivo
Vtd – verbo transitivo direto
Vti – verbo transitivo indireto
Vtdi – verbo transitivo direto e indireto

IV. PROPOSTA DE UM RECONHECEDOR GRAMATICAL

O Linguístico é uma proposta de reconhecedor gramatical composto de cinco módulos sequenciais que realizam cada qual um processamento especializado, enviando o resultado obtido para o módulo seguinte, tal como ocorre em uma linha de produção, até que o texto esteja completamente analisado (Fig.1).

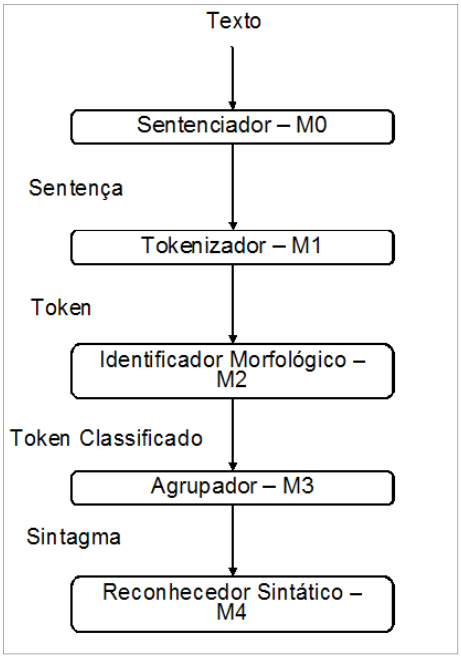


Figura 1. Estrutura do Linguístico.

O primeiro módulo, denominado Sentenciador, recebe um texto e realiza um pré-processamento, identificando os caracteres que possam indicar final de sentença, palavras abreviadas e palavras compostas, e eliminando aspas simples e duplas. Ao final, o Sentenciador divide o texto em supostas sentenças, para análise individual nas etapas seguintes.

O segundo módulo, denominado Tokenizador, recebe as sentenças identificadas na etapa anterior e as divide em *tokens*, considerando, neste processo, abreviaturas, valores monetários, horas e minutos, numerais arábicos e romanos, palavras compostas, nomes próprios, caracteres especiais e de pontuação final. Os *tokens* são armazenados em estruturas de dados (*arrays*) e enviados um a um para análise do módulo seguinte.

O terceiro módulo, chamado Identificador Morfológico, foi concebido para utilizar tecnologias adaptativas (Fig.2.1). O Identificador Morfológico é composto por um Autômato Mestre e um conjunto de submáquinas especialistas que acessam bases de dados com regras de acesso às bases de dados de classificações morfológicas. A prioridade é obter as classificações do corpus Bosque [13]; caso o termo procurado não seja encontrado, o Identificador Morfológico procura por substantivos, adjetivos e verbos, no formato finito e infinito (flexionados e não flexionados), através de uma submáquina de formação e identificação de palavras, que usa, como base, o vocabulário do TeP2.0 [14]; por fim, o Identificador Morfológico procura por termos invariáveis, ou seja, termos cuja classificação morfológica é considerada estável pelos linguistas, tais como, conjunções, preposições e pronomes, no caso, extraídos no léxico do Portal São Francisco [15].

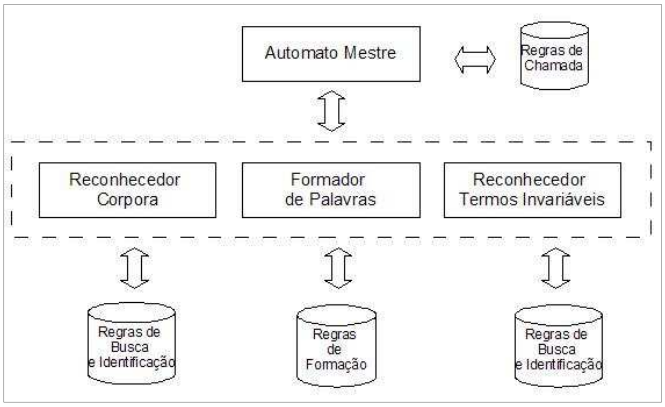


Figura 2.1. Arquitetura do Identificador Morfológico.

O Autômato Mestre (Fig.2.2) é responsável pelo sequenciamento das chamadas às submáquinas, de acordo com um conjunto de regras cadastradas em base de dados. Ele inicia o processamento recebendo o token da coleção de tokens criada pela classe Texto. Em seguida, antes de processá-lo, o autômato se modifica através de uma função adaptativa, criando uma submáquina de processamento (M1), uma transição entre o estado 1 e M1, e uma transição que aguarda o estado final da submáquina M1. O token é passado para M1 e armazenado em uma pilha. Quando M1 chega ao estado final, o autômato se modifica novamente, criando uma nova submáquina M2, o estado 2 e as transições correspondentes. Caso o estado final de M1 seja de aceitação, o processo é finalizado no estado 2, caso contrário a máquina M2 é chamada, passando o token armazenado na pilha.

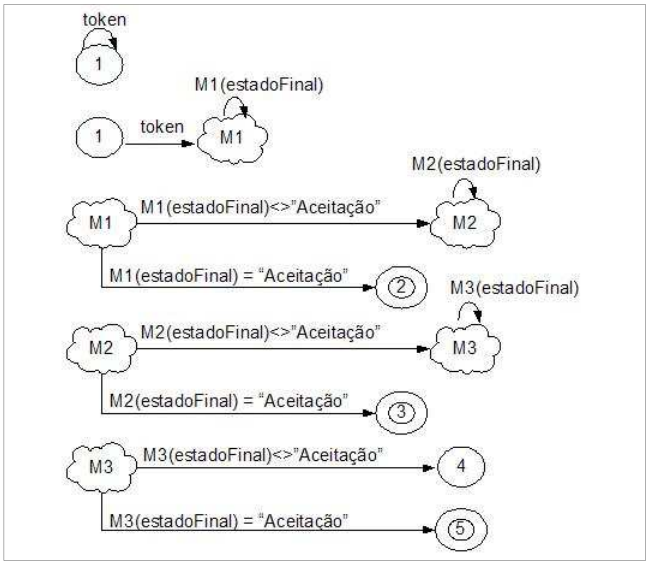


Figura 2.2. Estrutura Adaptativa do Autômato Mestre.

O processo se repete quando M2 chega ao final do processamento, com a criação da submáquina M3, do estado 3 e das transições correspondentes. Um novo ciclo se repete, e se o estado final de M3 é de aceitação, o autômato transiciona para o estado 5, de aceitação, caso contrário, ele vai para o estado 4 de não aceitação. M1, M2 e M3 representam, respectivamente, as submáquinas do Reconhecedor de Corpus, do Formador de Palavras e do Reconhecedor de Termos Invariáveis. Portanto, caso o Autômato Mestre encontre a classificação morfológica ao final de M1, ele não chama M2; caso encontre em M2, não chama M3 e, caso também não encontre em M3, ele informa aos demais módulos do Linguístico que não há classificação morfológica para o termo analisado.

As submáquinas M1, M2 e M3 também foram projetadas de acordo com a tecnologia adaptativa. A Máquina M1 usa um autômato adaptativo que se auto modifica de acordo com o tipo de token que está sendo analisado: palavras simples, palavras compostas, números, valores e símbolos. A Fig. 2.3 apresenta a estrutura adaptativa de M1.

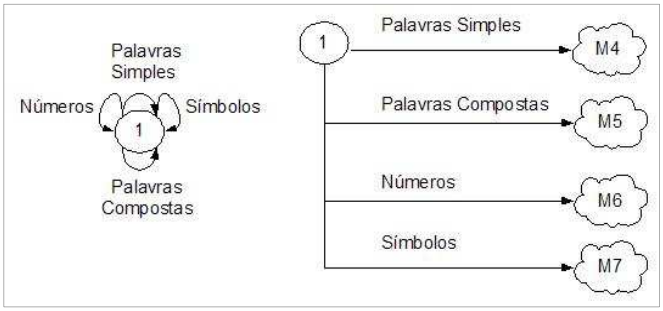


Figura 2.3. Estrutura Adaptativa do Reconhecedor de Corpus M1.

Inicialmente o autômato é composto por um único estado e por transições para ele mesmo (Fig.2.3, à esquerda). Ao identificar o tipo de token que será analisado (obtido no processo de tokenização), o autômato cria submáquinas e as transições correspondentes. As alternativas de configuração

são apresentadas na Fig.2.3, à direita. As submáquinas M4, M5, M6 e M7 reconhecem os tokens através de outro tipo de autômato que processa os tokens byte a byte (Fig. 2.4).

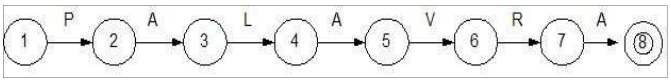


Figura 2.4. Reconhecedor de Palavras Simples.

No exemplo apresentado na Fig.2.4, o token “Palavra” é processado pela máquina M4; se o processamento terminar em um estado de aceitação, o token é reconhecido. As submáquinas M4, M5, M6 e M7 são criadas previamente por um programa que lê o Corpus e o converte em autômatos finitos determinísticos. A estrutura de identificação morfológica é composta por um par [chave, valor], no qual a chave é o estado de aceitação do elemento lexical e, o valor, sua classificação morfológica. No exemplo apresentado, a chave do item lexical “palavra” seria o estado ”8”, e, o valor, a classificação morfológica, H+n (substantivo, núcleo de sintagma nominal), proveniente do Corpus Bosque.

O Formador de Palavras, submáquina M2, também é montado previamente por um programa construtor, levando em consideração as regras de formação de palavras do Português do Brasil, descritas por Margarida Basílio em [16]. A submáquina M2 utiliza o vocabulário do TeP2.0 (formado por substantivos, adjetivos e verbos), como léxico de formas já feitas e o conjunto de regras de prefixação, sufixação e regressão verbal descrito pela autora para construir novas formas. No entanto, são necessários alguns cuidados para evitar a criação de estruturas que aceitem palavras inexistentes. A Fig. 2.5 apresenta um exemplo de autômato no qual são aplicados os prefixos “a” e “per”, e os sufixos “ecer” e “ar” (derivação parassintética) ao radical “noit” do substantivo noite, que faz parte do TeP2.0.

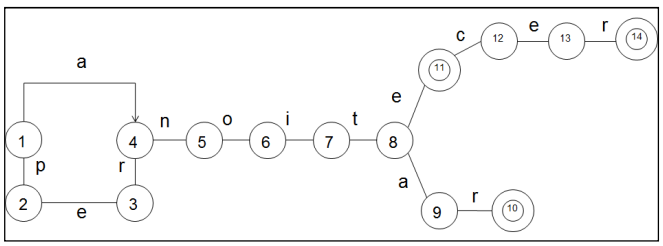


Figura 2.5. Autômato Formador de Palavras.

No exemplo apresentado, as palavras “anoitecer”, “pernoitar” e “anoitar” (sinônimo de “anoitecer”) existem no léxico do português do Brasil. Já pernoitecer é uma combinação que não existe na língua portuguesa. Margarida Basílio diz que algumas combinações não são aceitas simplesmente porque já existem outras construções consagradas pelo uso [17]. Para reduzir o risco de aceitar derivações inexistentes, o processo de construção do autômato restringe as possíveis formações, utilizando apenas as regras que a autora destaca como sendo mais prováveis. É o caso de nominalização de verbos com o uso dos sufixos –ção , –mento e – da. Em [16], Basílio cita que o sufixo –ção é responsável por 60% das formações regulares, enquanto o sufixo –mento é responsável por 20% destas formações. Já o sufixo –da é, via de regra, usado em nominalizações de verbos de movimento, tais como, entrada, saída, partida, vinda, etc.

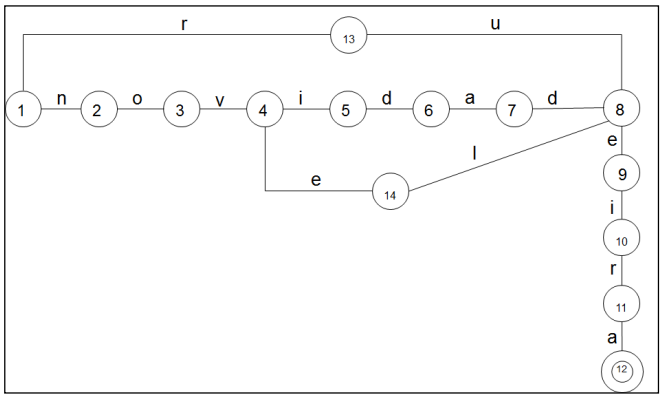


Figura 2.6. Autômato Formador de Palavras.

Outra característica do construtor do autômato é representar os prefixos e sufixos sempre pelo mesmo conjunto de estados e transições, evitando repetições que acarretariam o consumo desnecessário de recursos computacionais. A Fig. 2.6 apresenta exemplo de palavras formadas por reutilização de estados e transições na derivação das palavras rueira, novidadeira e noveleira. Os estados e transições usados para representar o sufixo “eira”, usado para designar são os mesmos nas três derivações.

A submáquina M2 também reconhece palavras flexionadas, obtidas, no caso de substantivos e adjetivos, através da aplicação de sufixos indicativos de gênero, número e grau aos radicais do vocabulário TeP2.0. A Fig. 2.7 apresenta um exemplo de autômato usado para a formação das formas flexionadas do substantivo menino. Foram adicionados ao radical “menin” as flexões “o(s)”, “a(s)”, “ão”, “ões”, “onona(s)”, “inho(s)” e“inha(s)”.

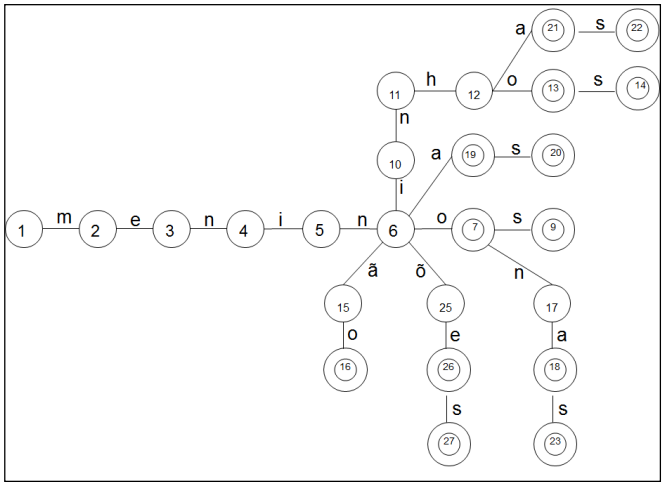


Figura 2.7. Autômato Formador de Flexões Nominais.

No caso de verbos, foi criada uma estrutura de estados e transições para representar as flexões de tempo, modo, voz e pessoa, obtendo-se, assim, as respectivas conjugações.

A Fig. 2.8 apresenta um exemplo de autômato usado para a formação das formas flexionadas do presente do indicativo do

verbo andar. Foram adicionados ao radical “and” as flexões “o”, “as”, “a”, “andamos”, “ais” e “andam”.

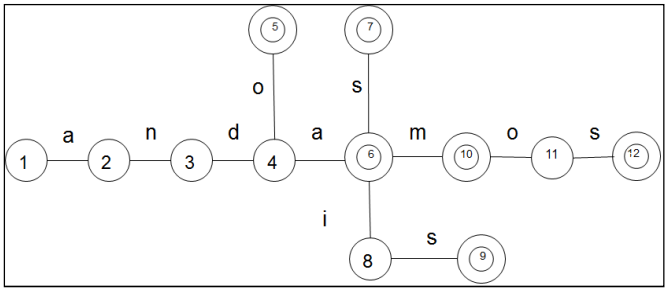


Figura 2.8. Autômato Formador de Flexões Verbais.

A estrutura de armazenamento da submáquina M2 também é composta por um par [chave, valor], no qual a chave é o estado de aceitação do elemento lexical e, o valor, sua classificação morfológica, acrescida da origem do termo, indicando que ele foi gerado pelo construtor. Por exemplo, o termo “novidadeira” seria classificado como H+n+C – substantivo, núcleo de sintagma nominal, gerado pelo construtor.

Já a submáquina M3 é um autômato que varia em função do tipo de termo (conjunções, preposições e pronomes) e utiliza uma estrutura arbórea similar a M4. A Fig. 2.9 apresenta um exemplo de autômato usado no reconhecimento de termos deste domínio.

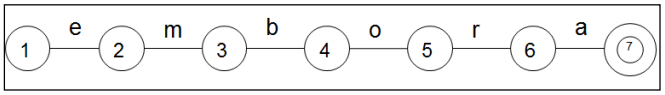


Figura 2.9. Autômato Reconhecedor de Conjunções, Preposições e Pronomes.

A estrutura de armazenamento da submáquina M3 é composta por um par [chave, valor], no qual a chave é o estado de aceitação do elemento lexical e, o valor, sua classificação morfológica, acrescida da origem do termo, indicando que ele faz parte da base de dados de apoio do Portal São Francisco. Por exemplo, o termo “embora” seria classificado como cj+Ba –conjunção da base de dados de apoio.

O Autômato Mestre também é responsável por desambiguar as classificações morfológicas e informar ao Agrupador apenas as que forem mais adequadas. Como as palavras podem ter mais do que uma classificação morfológica, é necessário utilizar uma técnica para selecionar aquela que é mais apropriada para o contexto analisado. Por exemplo, a palavra “casa” pode ser classificada como substantivo comum, feminino, singular ou como verbo flexionado na 3ª pessoa do singular no tempo presente, modo indicativo. No entanto, tendo em vista o contexto em que palavra se encontra, é possível escolher a classificação mais provável. Por exemplo, se a palavra “casa” vier precedida de um artigo definido, é mais provável que ela seja um substantivo; já se a palavra antecessora for um substantivo próprio, é mais provável que “casa” seja um verbo.

Collins [18] apresenta um modelo estatístico que leva em consideração 2 classificações anteriores à palavra analisada para fazer a desambiguação, criando distribuições nas quais a etiqueta de máxima probabilidade é o resultado da função:

P = max p(E,S), sendo:
E = etiquetas
S = palavras da sentença
p = probabilidade de E, dado S
max = máxima probabilidade

Por exemplo, para etiquetar a frase “O advogado entrevista a testemunha”, a função receberia as palavras da sentença e as sequências de etiquetas possíveis para elas (obtidas a partir de um Corpus de testes), calculando, então, a sequência de maior probabilidade. No entanto, Collins alerta que a complexidade para executar tal função inviabiliza sua utilização, pois ela cresce exponencialmente em função do número de palavras da sentença (Em uma sentença de “n” palavras e “K” possíveis classificações, existiriam $|K|^n$ possíveis etiquetas de sequencia).

Para evitar o problema da complexidade da execução, Collins propõe o uso do algoritmo Viterbi [19]. O algoritmo Viterbi é usado para encontrar a sequência mais provável de estados ocultos que resultam da observação de uma sequência de eventos. No caso da identificação das etiquetas morfológicas, os eventos são as palavras do texto analisado e os estados são as etiquetas de identificação morfológica. O algoritmo Viterbi pode ser implementado por meio de um autômato finito determinístico. No entanto, um autômato com estas características poderia ficar muito grande e, consequentemente, lento, devido ao tamanho do léxico usado para montá-lo. Uma alternativa para evitar este tipo de problema, é usar a tecnologia adaptativa. A Fig. 2.10 apresenta a estrutura do autômato que implementa o algoritmo Viterbi usado pelo desambiguador morfológico do Linguístico. O autômato recebe como parâmetro de entrada a palavra que está sendo analisada e monta dinamicamente os estados e transições de acordo com as possíveis etiquetas e as respectivas probabilidades, identificando, ao final do processamento, a etiqueta mais provável.

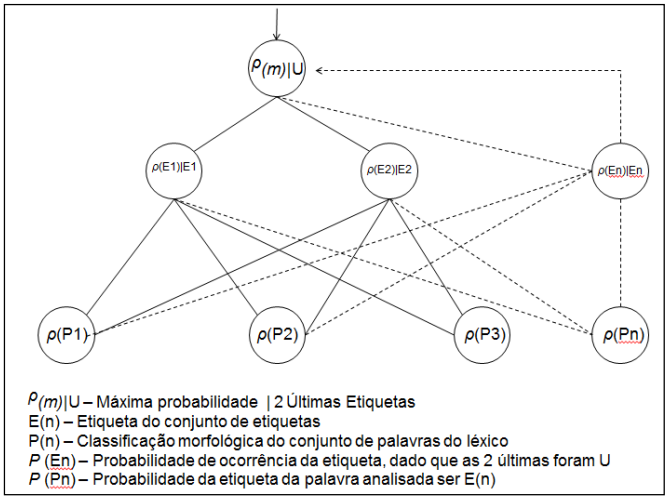


Figura 2.10. Autômato Adaptativo Viterbi.

O quarto módulo, denominado Agrupador é composto de um autômato, responsável pela montagem dos sintagmas a partir de símbolos terminais da gramática e um bigrama, responsável pela montagem dos sintagmas a partir de não-terminais (Fig.3). Inicialmente, o Agrupador recebe do Identificador as classificações morfológicas dos *tokens* e as agrupa em sintagmas de acordo com as regras de Luft. Neste processo são identificados sintagmas nominais, verbais, preposicionais, adjetivos e adverbiais Para isso, o Agrupador utiliza um

autômato adaptativo cuja configuração completa é definida da seguinte forma:
Estados = {1, 2, 3, 4, SS, SP, V, Sadj, Sadv, A},
Onde:
1,2,3 e 4 = Estados Intermediários
SS, SP, V Sadj, Sadv = Estados nos quais houve formação de sintagmas, sendo:
SS= Sintagma substantivo
SP = Sintagma preposicional
V = Verbo ou locução verbal
Sadj = Sintagma adjetivo
Sadv = Sintagma adverbial
A = Estado após o processamento de um ponto final
Tokens = {art, num, n, v, prp, pron, conj, adj, adv, rel, pFinal, sClass}, onde:
art = artigo, num = numeral
n = substantivo, v = verbo
prp = preposição, pron = pronome
conj = conjunção, adj = adjetivo
adv = advérbio, rel = pronome relativo
pFinal = ponto final, sClass = sem classificação
Estados de Aceitação = {SS, SP, V, Sadj, Sadv, A}
Estado Inicial = {1}
Função de Transição = {(Estado, *Token*)→Estado}, sendo:
{(1, art)→2, (2, art)→2, (3, art)→3
(1, num)→Sadv, (2, num)→2, (3, num)→3
(1, n)→SS, (2, n)→SS, (3, n)→SP
(1, v)→SV, (2, v)→ SV, (3, v)→SP
(1, prp)→3, (2, prp)→2, (3, prp)→3
(1, prop)→SS, (2, prop)→SS, (3, prop)→SP
(1, pron)→SS, (2, pron)→SS, (3, pron)→SP
(1, conj)→conj, (2, conj)→Ø, (3, conj)→ Ø
(1, adj)→Sadj, (2, adj)→Sadj, (3, adj)→3
(1, adv)→Sadv, (2, adv)→2, (3, adv)→3
(1, rel)→conj, (2, rel)→ Ø, (3, rel)→conj
(1, pFinal)→A, (2, pFinal)→ Ø, (3, pFinal)→ Ø}

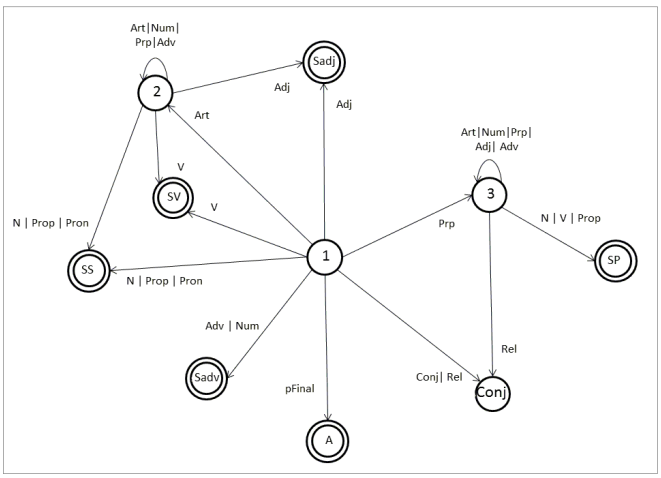


Figura 3. Configuração Completa do Autômato Construtor de Sintagmas.

Por exemplo, no caso de sintagmas substantivos teríamos:
SS → [Quant | PrA] (Sc | Sp | PrPes)
Pela regra acima, o conjunto de *tokens* "A" e "casa" formam um sintagma substantivo, da seguinte forma:
PrA = "A" (artigo definido)
Sc = "casa" (substantivo comum)
Da direita para esquerda, são realizadas as seguintes transições:
PrA Sc → SS; A Sc → SS; A casa → SS
Já o Agrupador recebe o *token* "A", identificado pelo Tokenizador como artigo definido, e se movimenta do estado 1 para o estado 2. Ao receber o *token* "casa", identificado como substantivo comum, ele se movimenta do estado 2 para o estado SS, que é um estado de aceitação. Neste momento o Agrupador armazena a cadeia "A casa" e o símbolo "SS" em uma pilha e reinicializa o autômato preparando-o para um novo reconhecimento. Em um passo seguinte, o Agrupador usa o bigrama para comparar um novo sintagma com o último sintagma formado, visando identificar elementos mais altos na hierarquia. Para isso ele usa a matriz apresentada na Tabela 3. A primeira coluna da matriz indica o último sintagma formado (US) e a primeira linha, o sintagma atual (SA). A célula resultante apresenta o novo nó na hierarquia.

TABELA 3.
MATRIZ DE AGRUPAMENTO DE SINTAGMAS

SA / US	SS	SP	V	Sadv	Sadj	Conj
SS	SS	SS	-	-	SS	-
SP	SP	-	-	-	-	-
V	-	-	V	-	-	-
Sadv	-	-	-	Sadv	Sadj	-
Sadj	SS	Sadj	-	-	Sadj	-
Conj	-	-	-	-	-	Conj

Esta técnica foi usada para tratar as regras gramaticais nas quais um sintagma é gerado a partir da combinação de outros, como é o caso da regra de formação de sintagmas substantivos: SS → [Sadj] SS [Sadj | SP]. Por esta regra, os sintagmas substantivos são formados por outros sintagmas substantivos precedidos de um sintagma adjetivo e seguidos de um sintagma adjetivo ou um sintagma preposicional. No exemplo anterior, supondo que os próximos 2 *tokens* fossem "de" e "madeira", após a passagem pelo autômato, o Agrupador formaria um sintagma SP. Considerando que na pilha ele tinha armazenado um SS, após a passagem pelo bigrama, e de acordo com a Tabela 3, o sintagma resultante seria um SS e o conteúdo que o compõe seria a combinação dos textos de cada sintagma que o originou. Caso não haja agrupamentos possíveis, o Agrupador envia o último sintagma formado para análise do Reconhecedor Sintático e movimenta o sintagma atual para a posição de último sintagma no bigrama, repetindo o processo com o próximo sintagma.

O quinto e último módulo, denominado Reconhecedor Sintático, recebe os sintagmas do módulo anterior e verifica se estão sintaticamente corretos de acordo com padrões oracionais de Luft. O Reconhecedor Sintático utiliza um autômato adaptativo que faz chamadas recursivas sempre que recebe conjunções ou pronomes relativos, armazenando, em uma estrutura de pilha, o estado e a cadeia de sintagmas reconhecidos até o momento da chamada. Caso o Reconhecedor Sintático não consiga se movimentar a partir do sintagma recebido, ele gera um erro e retorna o ponteiro para o

último sintagma reconhecido, finalizando a instância do autômato recursivo e retornando o processamento para aquela que a inicializou. Esta, por sua vez, retoma posição em que se encontrava antes da chamada e continua o processamento até o final da sentença ou até encontrar uma nova conjunção, situação na qual o processo se repete.

A configuração completa do autômato é definida da seguinte forma:

Estados = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27}

Tokens = {SS, SP, Vli, Vi, Vtd, Vti, Vtdi, Sadj, Sadv, Conj, A}

Estados de Aceitação = { 4, 5, 6, 9, 12, 13,14, 15, 17, 18, 19, 21, 22, 24, 25, 26, 27}

Estado Inicial = {1}

Função de Transição = {(Estado, *Token*)→Estado}, sendo:

{(1, SS)→2, (2, Vti)→3, (3, SP)→4, (4, SP)→4,
(3, Sadv)→5, (2, Vi)→6, (2, Vtdi)→7
(7, SS)→8, (8, SP)→9, (9, SP)→9, (8, Sadv)→10,
(2, Vlig)→11, (11, SP)→12, (11, Sadv)→13,
(11, Sadj)→ 14, (11, SS)→15, (2, Vtd)→16, (16, SS)→17,
(2, Vtpred)→18, (18, SP)→19, (18, Sadj)→20
(18, SS)→ 21, (21, SS)→22, (21, Sadj)→23, (21, Sadv)→24,
(21, SP)→25}

Pilha = {[Texto, Sintagma, Estado]}

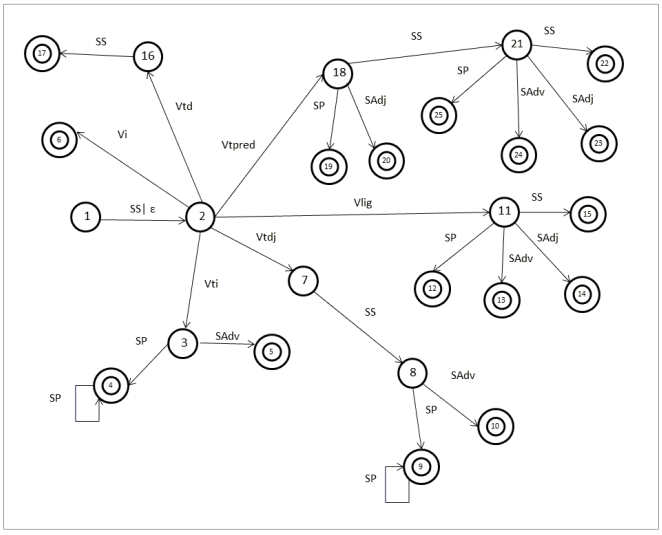


Figura 4.1. Configuração Completa do Reconhecedor Sintático.

No entanto, para que a análise sintática seja feita, não são necessárias todas as ramificações da configuração completa do autômato (Fig. 4.1). Por exemplo, quando se transita um verbo de ligação a partir do estado 2, o autômato vai para o estado 11 e todas as demais ramificações que partem deste estado para os estados 3, 7, 16 e 18, não são usadas. Com a tecnologia adaptativa, é possível criar dinamicamente os estados e transições do autômato em função dos tipos de verbos, evitando manter ramificações que não são usadas.

A Fig. 4.2 apresenta a configuração inicial do autômato adaptativo equivalente ao autômato de pilha apresentado anteriormente. No estado 1, o autômato recebe os *tokens* e transita para o estado 2 quando processa um sintagma substantivo (SS) ou quando transita em vazio. No estado 2, o autômato transita para si mesmo quando recebe qualquer tipo de verbo: Vi, Vtd, Vlig, Vtpred, Vtdi e Vti. Todas as outras ramificações são criadas por meio de funções adaptativas chamadas em função do tipo de verbo processado.

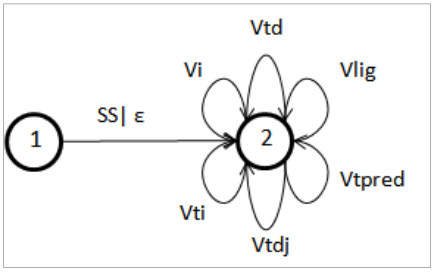


Figura 4.2. Configuração Inicial do Reconhecedor Gramatical.

Por exemplo, se o verbo é de ligação (Vlig), o autômato utiliza as funções adaptativas α (j) e β (o), definidas da seguinte forma:

α (j): { o* :
- [(j, Vlig)]
+[(j, Vlig) :⟶ o, β (o)]
}

β (o): { t*u*v*x*:
+ [(o, SP) :⟶ t]
+ [(o, Sadv) :⟶ u]
+ [(o, Sadj) :⟶ v]
+ [(o, SS) :⟶ x]
}

A função adaptativa α (j) é chamada pelo autômato antes de processar o *token*, criando o estado 11 e a produção que leva o autômato do estado 2 ao novo estado criado. Em seguida, o autômato chama a função β (o), criando os estados 12, 13, 14 e 15 e as produções que interligam o estado 11 aos novos estados. A Fig. 4.3 mostra a configuração do autômato após o processamento do verbo de ligação. Neste exemplo, o autômato criou apenas os estados 11, 12, 13, 14 e 15 e as respectivas transições, evitando alocar recursos que seriam necessários para criar o autômato completo, conforme apresentado na Fig. 4.1.

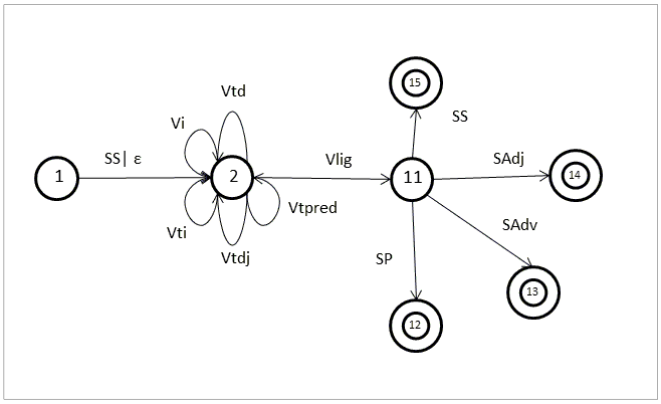


Figura 4.3. Configuração do autômato após o processamento do verbo de ligação.

V. SEMÂNTICA COMO APOIO PARA DECISÃO

A proposta de uso de critérios semânticos para decisão apoia-se no fato de que apenas recursos gramaticais são insuficientes para questão de desambiguação, tomando, por exemplo, o problema da identificação de termos referenciados por anáforas pronominais. Além disso, questões como concordância verbal e nominal e regência necessitam do contexto semântico para o correto reconhecimento da frase

apresentada. Autômatos adaptativos podem ser uma alternativa válida para a implantação de reconhecimento semântico, tendo em vista o conjunto de regras de auto modificação que fazem parte de seu formalismo. Tais regras garantem flexibilidade para implantar tanto validações de regência como questões de correferência.

VI. CONSIDERAÇÕES FINAIS

Este artigo apresentou uma revisão dos conceitos de Tecnologia Adaptativa e de Processamento da Linguagem Natural. Em seguida, foi apresentado o Linguístico, uma proposta de reconhecedor gramatical que utiliza autômatos adaptativos como tecnologia subjacente. Por fim, procurou-se apresentar cenários em que o uso de critérios semânticos pode auxiliar na resolução de ambiguidades e no descarte de caminhos de reconhecimento gramaticais inválidos.

O Linguístico está fase de desenvolvimento e testes preliminares confirmaram a tecnologia adaptativa como uma alternativa válida para o processamento da linguagem natural, proporcionando economia de recursos computacionais e flexibilidade para implantação de novas regras, o que nos motiva a aprofundar a pesquisa, provendo embasamento quantitativo para chegar a respostas mais conclusivas.

REFERÊNCIAS

[1] http://www.pcs.usp.br/~lta/

[2] C. Taniwaki. Formalismos adaptativos na análise sintática de Linguagem Natural . Dissertação de Mestrado, EPUSP, São Paulo, 2001.

[3] C. E. Menezes. Um método para a construção de analisadores morfológicos, aplicado à língua portuguesa, baseado em autômatos adaptativos. Dissertação de Mestrado, Escola Politécnica da Universidade de São Paulo, 2000.

[4] D. Padovani. Uma proposta de autômato adaptativo para reconhecimento de anáforas pronominais segundo algoritmo de Mitkov. Workshop de Tecnologias Adaptativas –WTA 2009, 2009.

[5] M. Moraes. Alguns aspectos de tratamento sintático de dependência de contexto em linguagem natural empregando tecnologia adaptativa, Tese de Doutorado, Escola Politécnica da Universidade de São Paulo, 2006.

[6] E. Rich and K. Knight. Inteligência Artificial, 2. Ed. São Paulo: Makron Books, 1993.

[7] R. Vieira and V. Lima. Linguística computacional: princípios e aplicações. IX Escola de Informática da SBC-Sul, 2001.

[8] C. Fuchs and P. Le Goffic. Les Linguistiques Contemporaines.

[9] M. G. V.Nunes et al. Introdução ao Processamento das Línguas Naturais. Notas didáticas do ICMC Nº 38, São Carlos, 88p, 1999. Paris, Hachette, 1992. 158p.

[10] T. B. Sardinha, A Língua Portuguesa no Computador. 295p. Mercado de Letras, 2005.

[11] R.L.A. Rocha. Tecnologia Adaptativa Aplicada ao Processamento Computacional de Língua Natural. Workshop de Tecnologias Adaptativas – WTA 2007, 2007.

[12] C. Luft. Moderna Gramática Brasileira. 2ª. Edição Revista e Atualizada. 265p. Editora Globo, 2002.

[13] http://www.linguateca.pt/

[14] http://www.nilc.icmc.usp.br/tep2/

[15] http://www.portalsaofrancisco.com.br/alfa/materias/index-lingua-portuguesa.php/

[16] M. Basilio. Formação e Classes de Palavras no Português do Brasil. Ed.Contexto, 2004.

[17] M. Basilio. Teoria Lexical. Ed.Atica, 1987.

[18] http://www.cs.columbia.edu/~mcollins/hmms-spring2013.pdf/

[19] G.D. J. Forney. IEEE. Proceedings of the IEEE, Volume 61 , Issue 3,1973.

Ana Teresa Contier formou-se em Letras-Português pela Universidade de São Paulo (2001) e em publicidade pela PUC-SP (2002). Em 2007 obteve o título de mestre pela Poli-USP com a dissertação: “Um modelo de extração de propriedades de textos usando pensamento narrativo e paradigmático”.

Djalma Padovani formou-se em administração de empresas pela Faculdade de Economia e Administração da Universidade de São Paulo, em 1987 e obteve o mestrado em engenharia de software pelo Instituto de Pesquisas Tecnológicas de São Paulo - IPT, em 2008. Trabalhou em diversas empresas nas áreas de desenvolvimento de software e tecnologia de informação e atualmente é responsável pelos sistemas de e-commerce e internet da Serasa Experían.

João José Neto graduado em Engenharia de Eletricidade (1971), mestrado em Engenharia Elétrica (1975) e doutorado em Engenharia Elétrica (1980), e livre-docência (1993) pela Escola Politécnica da Universidade de São Paulo. Atualmente é professor associado da Escola Politécnica da Universidade de São Paulo, e coordena o LTA - Laboratório de Linguagens e Tecnologia Adaptativa do PCS - Departamento de Engenharia de Computação e Sistemas Digitais da EPUSP. Tem experiência na área de Ciência da Computação, com ênfase nos Fundamentos da Engenharia da Computação, atuando principalmente nos seguintes temas: dispositivos adaptativos, tecnologia adaptativa, autômatos adaptativos, e em suas aplicações à Engenharia de Computação, particularmente em sistemas de tomada de decisão adaptativa, análise e processamento de linguagens naturais, construção de compiladores, robótica, ensino assistido por computador, modelagem de sistemas inteligentes, processos de aprendizagem automática e inferências baseadas em tecnologia adaptativa.

Uso de Técnicas Adaptativas no Reconhecimento Biométrico por Impressão Digital

F. B. R. do Val,P.R. Marcelino e J. J. Neto

***Abstract*— Recognition of individuals, with practicality and reliability, is an important need of our society, especially when it comes to provide access to restricted places and systems. Based on this, studies about biometric authenticationare increasing, which usually combines easiness for users and security. This article focus on proposing the application of adaptivity concepts on traditional algorithms of fingerprint authentication, the most used biometric technique, in order to improve the performance of these algorithms through the reduction of mistaken decisions.**

***Keywords*— Fingerprint Recognition, Adaptivity, Gabor filter, Image Processing.**

I. INTRODUÇÃO

IMPRESSÕES digitais possuem características únicas capazes de distinguir uma pessoa em uma população. Essas características são denominadas minúcias. As principais minúcias observadas para identificar um indivíduo são: bifurcações e terminações, como é possível observar na imagem a seguir[1].

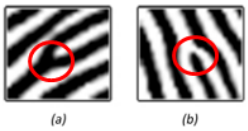


Figura 1. (a) Bifurcação, (b) Terminação.

Cristas são os relevos da pele observados na impressão digital, e sulcos são as depressões localizadas entre as cristas. A bifurcação é quando uma crista da digital se divide em duas e terminação é quando uma crista termina, iniciando-se o sulco.

A qualidade da impressão digital varia de acordo com cada pessoa, muitas vezes estando associada à função laboral da mesma. No geral, indivíduos que trabalham realizando esforços com as mãos - como pedreiros, costureiras, operadores de caixa, entre outros - apresentam digitais em que as minúcias têm qualidade inferior à de um indivíduo que não realiza grandes esforços manuais.

Além disso, a digital de cada ser humano varia com o tempo, seja por envelhecimento ou por feridas e cicatrizes. Por isso, é comum, em sistemas de autenticação biométrica por impressão digital, a pessoa cuja digital alterou-se ter dificuldades para se autenticar no sistema. Desta forma, com o passar do tempo os sistemas passam a rejeitar com maior frequência usuários cadastrados.

Também é comum o sistema detectar falsas minúcias devido a impurezas no leitor de impressão digital, devido a resíduos no dedo do indivíduos ou mesmo distorção elástica da pele ao ser pressionada contra o sensor. A detecção de falsas minúcias dificulta a correta identificação dos usuários.

II. OBJETIVOS

Este projeto objetiva melhorar a qualidade dos sistemas de autenticação por impressão digital através da aplicação de técnicas adaptativas.

As métricas comumente utilizadas para avaliar a qualidade de sistemas biométricos são:

- Taxa de falsa aceitação (*FAR – False Acceptance Rate*): Indicador para avaliar a frequência que o sistema aceita pessoas não cadastradas no mesmo. Idealmente, este indicador deve ser zero ou um valor significativamente baixo.
- Taxa de falsa rejeição (*FRR – False Rejection Rate*): Métrica que indica a frequência que usuários cadastrados no sistemas são rejeitados, isto é, o sistema não reconhece a pessoa como usuária do sistema.
- Taxa de erro equivalente (*EER – Equivalent Error Rate*): Esta métrica representa o equilíbrio entre a taxa de falsa aceitação e a taxa de falsa rejeição.

Tempo médio de processamento de digitais de entrada: Indicador em medida de tempo para avaliar se o sistema leva muito tempo para aceitar ou rejeitar um indivíduo.

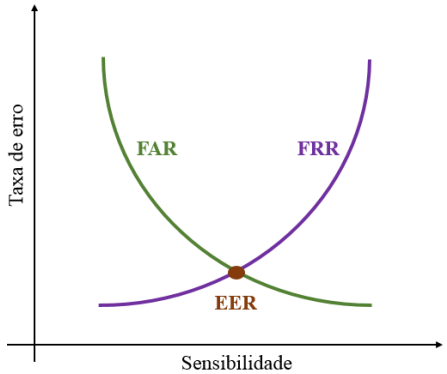


Figura 2. Curvas das principais métricas de sistemas de autenticação por impressão digital.

F. B. R. do Val, Universidade de São Paulo (USP), São Paulo, Brasil, fernanda.brval@gmail.com
P. R. Marcelino, Universidade de São Paulo (USP), São Paulo, Brasil, priscila.ribeiro.marcelino@gmail.com
J. J. Neto, Universidade de São Paulo (USP), São Paulo, Brasil, jjneto@gmail.com

O objetivo do projeto é, por meio de recursos adaptativos, melhorar as métricas de desempenho do sistema, para que o mesmo aceite mais usuários cadastrados e rejeite mais usuários não cadastrados. Busca-se que as melhorias mencionadas não comprometam o tempo de processamento.

Estão em andamento testes para mensurar a melhoria proporcionada pelo sistema desenvolvido com aplicação de técnicas adaptativas. O progresso mencionado já foi observado qualitativamente.

III. VISÃO GERAL DO SISTEMA

O sistema é composto por três etapas:

1. Pré-processamento da imagem da impressão digital;
2. Extração das minúcias;
3. Autenticação (matching).

Na primeira etapa, a imagem de entrada passa por diversos passos que têm a finalidade de melhorar a qualidade da mesma, buscando reduzir a quantidade de falsas minúcias a serem extraídas na etapa seguinte.

Na etapa de extração das minúcias, a imagem tratada anteriormente é avaliada, buscando encontrar terminações e bifurcações. O sistema não faz distinção entre terminações e bifurcações. Para cada minúcia extraída, o sistema indica as coordenadas "x" e "y", o ângulo de orientação da minúcia em relação ao eixo x e a nota de qualidade da minúcia extraída. Ao final da extração de minúcias, gera-se o template da digital, ou seja, um mapa com todas as minúcias referentes à imagem pré-processada. Este template servirá como base para a etapa seguinte.

Figura 3. Imagem original, minúcias extraídas e template gerado.

Na última etapa do sistema, autenticação, faz-se a comparação entre templates cadastrados e o template de entrada. A comparação é feita a partir dos parâmetros extraídos de cada uma das imagens das digitais, isto é, a partir da informação da minúcias, que possuem posição, ângulo de orientação e nota de qualidade.

A comparação retorna uma nota de similaridade entre os *templates* comparados. De acordo com a nota de corte pré-estabelecida, o indivíduo pode ser aceito ou rejeitado no sistema.

Na figura 4, é possível observar as etapas do sistema. Estas etapas estão exibidas de forma simplificada.

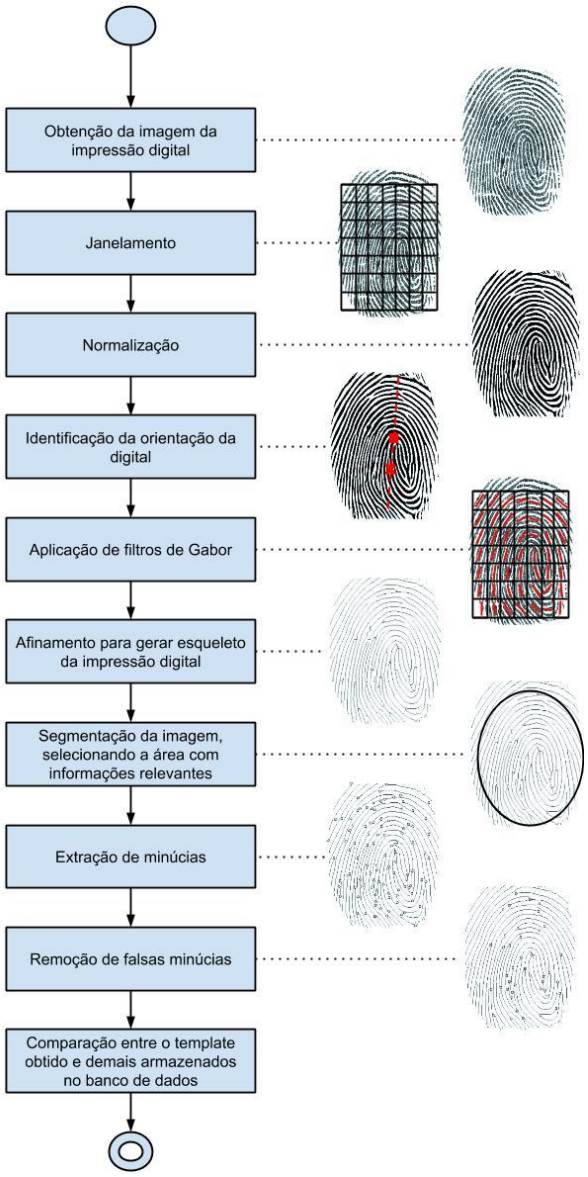


Figura 4. Etapas do sistema de autenticação por impressão digital.

Para uma compreencão global dos algoritmos tradicionais de reconhecimento biométrico, consultar as referências [1], [2] e [3].

IV. APLICAÇÕES DE ADAPTATIVIDADE NO SISTEMA

O projeto aplicou técnicas de adaptatividade em algumas das etapas do sistema, como será explicado a seguir.

IV. I.PRÉ-PROCESSAMENTO DA IMAGEM DA IMPRESSÃO DIGITAL

Na etapa de pré-processsamento da imagem da impressão digital, técnicas adaptativas são aplicadas em dois momentos: na normalização e no filtro de Gabor. Estas etapas podem ser identificadas na figura 4.

IV. I. I.NORMALIZAÇÃO

Normalização é a segunda etapa do sistema, como observa-se na figura 4. Este passo é responsável por

eliminarcomponentes espúrias de intensidade, de baixa frequência espacial, devido a sujeira ou contato irregular entre a superfície do dedo e o sensor, de forma a padronizar a intensidade da mesma.

A figura normalizada é definida por [3]:

$$N(i,j) = M0 + \sqrt{(V0(I(i,j) - M))/V}\text{, se } I(i,j) > M\text{,}$$

ou

$$N(i,j) = M0 - \sqrt{(V0(I(i,j) - M))/V}\text{, se } I(i,j) \leq M\text{.}$$

Em que:

- $I(i,j)$ é o valor do pixel (i,j) em escala de cinza;
- $N(i,j)$ é o valor do pixel (i,j) normalizado em escala de cinza;
- M é a média da intensidade de todos $I(i,j)$ da imagem ou região selecionada;
- V é a variância de todos $I(i,j)$ da imagem ou região selecionada;
- $M0$ é a média desejada;
- $V0$ é a variância desejada;

Neste projeto, a imagem é fragmentada em janelas de tamanhos iguais e, então, aplica-se a normalização para cada uma dessas janelas. Nas imagens a seguir, é possível observar a melhoria na qualidade da imagem ao aplicar a normalização por janelas em vez de na imagem inteira.

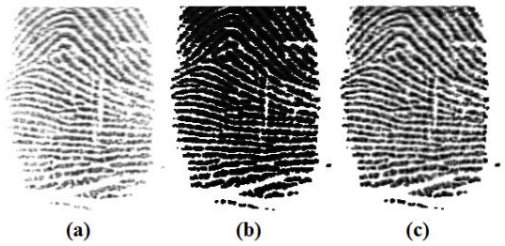


Figura 5. (a) Imagem original, (b)normalização aplicada na imagem não segmentada e (c) normalização aplicada na imagem segmentada em janelas.

Realizando a normalização por janelas, a qualidade da figura melhora significamente se comparada à qualidade da imagem normalizada sem segmentação alguma.

IV. I. II.FILTRO DE GABOR

A etapa do filtro de Gabor é dependente de duas etapas anteriores: mapas de frequência e de orientação, como é possível observar na figura 4.

A frequência de cristas indica o quão próximas ou espaçadas estão as cristas da impressão digital. Para calcular essa frequência, utilizamos transformada de Fourier (DFT: Discret Fourier Transform), passando como entrada a imagem e obtendo como resultado suas componentes senoidais. Na figura 6, é possível observar uma ilustração do funcionamento da transformada.

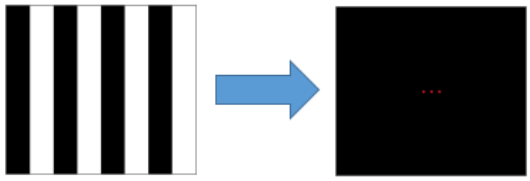


Figura 6. Frequência das cristas de uma janela, após a aplicação da transformada de Fourier.

Na figura 6, a imagem à direita é o resultado da *dft* aplicada na imagem da esquerda. O ponto central representa a componente DC e os outros dois pontos representam a variação no "sinal". Os outros dois pontos estão afastados 4 pixels em relação ao ponto central. Isso significa que na imagem de entrada, há 4 ciclos completos do sinal. Com essa informação e o tamanho da imagem de entrada, é possível calcular a periodicidade espacial.

O cálculo da frequência é realizado para cada janela da imagem de entrada. Denomina-se mapa de frequência, pois é um mapeamento da frequência de cada uma das janelas.

Outra etapa necessária para execução do filtro de Gabor, é a geração do mapa de orientação. Nesta etapa, a direção das cristas de cada janela é avaliada e uma direção que aponta no sentido do gradiente é definida. O gradiente aponta na perpendicular das cristas, que podem ser vistas como análogas a curvas de nível. Nesta etapa, faz-se uso do filtro de Sobel. A figura 7 permite observar a orientação obtida para cada uma das janelas.

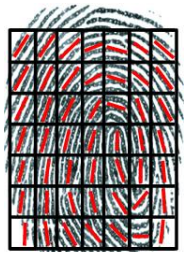


Figura 7. Mapa de orientação da imagem da digital.

Em [4], os modelos matemáticos utilizados como base para as etapas de geração dos mapas de freqüência e orientação estão explicados detalhadamente.

Após obtidos os mapas de frequência e orientação da imagem, aplica-se o filtro de Gabor para cada uma das janelas.

Filtro de Gabor é um filtro passa-banda, que de acordo com a orientação e frequência, apresenta ótimas melhorias na qualidade da imagem. Este filtro é uma combinação de uma função harmônica com uma função gaussiana [1].

Matematicamente, o filtro de Gabor pode ser descrito da seguinte maneira [1]:

$$g(x,y:\theta,f) = exp\left\{-\frac{1}{2}\left(\frac{x_\theta^2}{\sigma_x^2} + \frac{y_\theta^2}{\sigma_y^2}\right)\right\}.cos(2\pi.f.x_\theta)$$

Em que:

- f é a frequência de cada janela;
- σ_x e σ_y são os valores das variâncias do sinal nas direções x e y, respectivamente;
- $[x_\theta, y_\theta]$ são os pontos $[x, y]$ rotacionados por θ.

$$\begin{bmatrix} x_\theta \\ y_\theta \end{bmatrix} = \begin{bmatrix} sen\theta & cos\theta \\ -cos\theta & sin\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Denomina-se filtragem adaptativa, visto que cada janela será filtrada de forma diferente, com base na frequência e orientação daquela janela. O Filtro de Gabor é muito importante para eliminar ruídos da imagem que resultariam em falsas minúcias. Ele analisa o padrão da imagem e corrige regiões que apresentem desvios em relação a esse padrão. Por exemplo, para uma dada frequência média de cristas, se existirem pixels de ruído fazendo com que a frequência seja maior numa pequena parte da imagem, o ruído é eliminado. Na figura 8, há uma ilustração da aplicação do filtro de Gabor, nela nota-se a melhoria obtida com a eliminação de falsas minúcias.

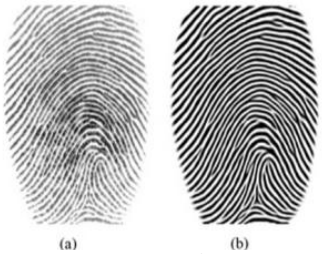


Figura 8. (a) Imagem antes da aplicação do filtro de Gabor e (b) imagem após filtragem.

IV. II.AUTENTICAÇÃO

A etapa de autenticação (*matching*) é realizada a partir da comparação entre dois *templates*.

Cada *template* equivale a um grafo em que os nós são compostos pelas minúcias extraídas de uma imagem de impressão digital. O *template* é uma relação de distância e ângulo entre diferentes minúcias de uma mesma digital. Para verificar a similaridade entre digitais, compara-se os ramos e nós de dois *templates* e quanto mais parecidos, maior é a nota de similaridade.

De acordo com a nota de similaridade resultante da comparação,o sistema decide se a pessoa é aceita ou não na aplicação.

A seguir, estão explicadosos usos de técnicas adaptativas na etapa de autenticação.

IV.II. I.ALTERAÇÃO DA NOTA DE CORTE DE ACORDO COM A FINALIDADE DO SISTEMA

Duas das principais métricas de desempenho de sistemas de autenticação por digitais são as taxas de falsa aceitação e falsa rejeição, que são recíprocas, isto é, uma aumenta enquanto a outra diminui em relação a uma variável comum, que no caso é a nota mínima de similaridade para aceitação.

A proposta de aplicação de métodos adaptativos é modificar a atuação do sistema de acordo com o tipo de aplicação. Por exemplo, aplicação do governo para controlar a entrada e saída de pessoas no país, este tipo de sistema deve ser altamente seguro, com baixíssimos riscos de confundir os indivíduos que saíram ou entraram no país. Outra aplicação seria de controle de acesso de ajudantes de obra que, no geral, têm impressão digital de baixa qualidade.

O que se deseja ilustrar com estes exemplos é: na primeira aplicação, sistema governamental, o ideal é que a taxa de falsa aceitação (*FAR*) – ocorrência de *matching* indevido – seja muito baixa, porém, ao estabelecer isso, compromete-se a taxa de falsa rejeição (*FRR*) – não ocorrência de *matching*, quando deveria ocorrer. Esta aplicação teria maiores dificuldades para identificar as pessoas, muitas vezes solicitando novas digitais de entrada.

Já o segundo exemplo, o sistema de controle de acesso de ajudantes de obra, deveria ser mais tolerante, visto que a qualidade das digitais de entrada não é alta. Para que o sistema não solicite muitas vezes que os usuários insiram a digital novamente, reduz-se a nota de corte; ou seja, a nota de similaridade entre o template da digital de entrada e o *template* de uma das digitais salvas no banco de dados não precisa ser tão alta quanto a da primeira aplicação exemplo. Isso compromete a taxa de falsa aceitação (*FAR*), que tende a apresentar valores maiores, mas reduz a taxa de falsa rejeição (*FRR*).

A solução proposta no projeto é: o administrador do sistema determina qual o tipo de aplicação deseja, se é um sistema rigoroso ou tolerante, e realiza uma etapa de treinamento do sistema antes de colocá-lo em operação. Durante o treinamento, o administrador entra com diversas imagens e informa se cada uma delas deveria ser aceita ou rejeitada. Dessa forma, a nota de corte para a similaridade entre *templates* é alterada automaticamente com base nesse feedback, de forma a reduzir erros futuros.

Essa alteração da nota de corte se dá conforme a seguinte lógica:

a) Caso o administrador opte por limitar a taxa de falsa aceitação, o sistema irá aumentar a nota de corte em uma unidade após cada aceitação errônea que for cometida, se tornando cada vez mais rigoroso, até que essa taxa se mantenha dentro de um limite pré-estabelecido.
b) Caso o administrador opte por limitar a taxa de falsa rejeição, o sistema irá reduzir a nota de corte em uma unidade após cada rejeição errônea que for cometida, se tornando cada vez mais tolerante, até que essa taxa se mantenha dentro de um limite pré-estabelecido.

Caso o administrador do sistema deseje aumentar a confiabilidade, é possível solicitar aos indivíduos que além de inserir a digital, entrem com outras informações, como data de nascimento, RG, CPF, entre outras. Essas informações complementares são campos do banco de dados, e o sistema só analisará as digitais correspondentes a indivíduos que, por exemplo, tenham uma determinada data de nascimento.

IV. II. II.ALTERAÇÃO DA NOTA DE CORTE DE ACORDO COM O USUÁRIO

Outra aplicação de métodos adaptativos é realizar a alteração da nota de corte de similaridade para cada usuário. Caso o administrador não defina o tipo de aplicação – se deve ser rigorosa ou não – o sistema controla a nota de corte de forma individualizada, ou seja, de acordo com a qualidade da digital de cada usuário, o valor da nota de corte muda.

O sistema é constantemente retroalimentado pelo administrador, que indica quando o sistema realiza uma falsa aceitação ou uma falsa rejeição.

No caso de uma falsa aceitação, o sistema aumenta a nota de corte de similaridade do usuário que foi identificado erroneamente. Para que o sistema aprenda, o administrador deve indicar quando foi tomada uma decisão incorreta.

Quando ocorre uma falsa rejeição (um usuário cadastrado no sistema tentou acessar o mesmo, mas não foi aceito), o administrador deve indicar ao sistema qual usuário que tentou entrar. Assim, o sistema aprende que não fez uma identificação correta para determinado usuário. Automaticamente, o sistema altera a nota de corte daquele indivíduo. Obviamente, há um limite para o valor mínimo da nota de corte a fim de garantir que o sistema permaneça confiável.

Alguns dos motivos para que ocorra falsa rejeição são a existência de resíduos no aparelho de detecção de impressão digital, resíduos no dedo ou até mesmo machucados.

Como já mencionado, as principais métricas de desempenho de sistemas de autenticação por digitais são as taxas de falsa aceitação e falsa rejeição, que são recíprocas. Ao adaptar a nota de corte de cada usuário, o sistema reduz ambos os tipos de falsas decisões.

IV. II. III.CADASTRO AUTOMÁTICO DE NOVAS DIGITAIS

Esta terceira aplicação de métodos adaptativos é complementar à segunda – alteração da nota de corte de acordo com o usuário. Considerando que o dedo do indivíduo sofreu alguma modificação, inicialmente o sistema é mais tolerante com aquela pessoa, ou seja, a nota de corte de similaridade sofre redução, como mencionado no item anterior. Porém, se a digital sofreu uma modificação permanente, como é o caso de envelhecimento ou cicatrizes, o sistema, após reduzir a nota de corte de similaridade para aquele indivíduo, identifica que o dedo sofreu alguma modificação permanente, visto que a nota de similaridade está constantementebaixa.

Identificada a alteração permanente na digital do usuário, o sistema – que armazenou anteriormente as digitais de tentativas de acesso sem sucesso, porém sem adicionar como *templates* do usuário para fazer a similaridade com novas digitais de entrada – escolhe as digitais de melhor qualidade que o indivíduo inseriu anteriormente e com as quais o sistema não conseguiu realizar autenticação, e então sim adiciona essas imagens no cadastro do usuário como novos *templates*.

Todos esses procedimentos são transparentes aos usuários. O administrador do sistema é responsável por retroalimentar o programa a fim de indicar quando o mesmo tomou uma decisão errônea.

V. CONSIDERAÇÕES FINAIS

Com a conclusão deste projeto, que segue em andamento, muitos outros trabalhos complementares podem agregar a esta pesquisa.

Podemos agrupar os trabalhos complementares em dois grupos principais: expansão do algoritmo de reconhecimento de padrões para novos tipos de padrões; e expansão do uso de adaptatividade no algoritmo de reconhecimento de impressão digital.

Quanto ao reconhecimento de padrões, a partir de uma imagem são extraídos parâmetros relevantes que tornam determinado objeto ou ser vivo identificável. A aplicação escolhida foi o padrão de impressões digitais, mas também é possível expandir o reconhecimento para: outros padrões biométricos; padrões da natureza; padrões de doenças em exames médicos, dentre outros.

Para citar alguns exemplos, no caso de padrões biométricos, há diversas outras características que nos torna identificáveis, por exemplo o padrão da íris do olho, formato da mão, padrão de voz, formato da face e assinatura. É possível identificar os parâmetros relevantes de cada um desses itens e, a partir da extração deles, identificar os indivíduos.

No caso de padrões da natureza, um exemplo que pode ser citado é a identificação de espécies de abelhas a partir do desenho em suas asas. Outro exemplo são plantas, que podem ser identificadas a partir das ranhuras em suas folhas.

Também é possível identificar padrões para doenças, como câncer de pele, irregularidades no cérebro, dentre outras.

Do ponto de vista de adaptatividade, podemos manter a mesma aplicação escolhida – autenticação por impressão digital – e englobar novos parâmetros ou incluir novas técnicas. Por exemplo, parâmetros internos da etapa de *matching*, relacionados ao cálculo da nota de similaridade, poderiam ser investigados e tornados adaptativos. Ou seja, o sistema definiria o ponto ótimo de operação desses parâmetros com base em treinamento, da mesma forma que faz para o parâmetro de nota de corte. Ainda mais interessante seria tornar o sistema capaz de identificar os próprios parâmetros de interesse, os quais ele deveria variar para melhorar o desempenho da autenticação, de forma que os pesquisadores não precisem pré-definir esses parâmetros.

Vale ressaltar que além de otimizar o desempenho da autenticação de digital, quanto mais amplamente foram aplicados conceitos e técnicas adaptativas, mais apto a reconhecer diversos tipos de padrão de forma automática o sistema pode se tornar. Isso reduziria ou até eliminaria a necessidade de adaptação do código por parte dos pesquisadores a cada nova aplicação. Por exemplo, o sistema poderia, a partir da análise do formato do objeto de entrada, identificar de que tipo ele é (uma impressão digital, um animal, uma planta, um órgão, etc.), para então decidir se ele deve extrair minúcias ou outros parâmetros diferentes, se ele deve aplicar um ou outro filtro de tratamento na imagem, e assim por diante.

Por fim, fica evidente que a área deste projeto pode ser bastante desenvolvida com diversas outras pesquisas e que adaptatividade pode agregar muito valor ao reconhecimento de padrões.

AGRADECIMENTOS

Os autores gostariam de agradecer a contribuição que Albert Nissimoff, diretor de tecnologia da empresa Control ID, fez ao projeto por meio de explicações e esclarecimentos técnicos relacionados aos algoritmos tradicionais de identificação por impressão digital. Sua disponibilidade e interesse no projeto, combinados a sua vasta experiência na área, foram importantes para os resultados alcançados.

Os autores também gostariam de agradecer a Rafael Camargo Leite, estudante da Escola Politécnica, pelo suporte e apoio durante todo o projeto, em especial pela sua contribuição no esclarecimento de diversas dúvidas técnicas relacionadas ao ambiente de desenvolvimento do sofware.

REFERÊNCIAS

[1] T. d. S. Castro, "Identificação de Impressões Digitais Baseada na Extração de Minúcias," Juiz de Fora, Minas Gerais, Brasil, 2008.

[2] X. Jiang, W. Y. Yau e W. Ser, "Fingerprint Image Processing for Automatic Verification," em *IEEE 2nd Intrenational Conference on Information, Communication & Sigal Processing*, Singapore, 1999.

[3] B. N. Lavanya, K. B. Raja e K. R. Venugopal, "Fingerprint Verification based on Gabor Filter Enhancement," *Intrenational Journal of Computer Science and Information Security,* vol. 6, pp. 138-144, 2009.

[4] R. Thai, "Fingerprint Image Enhancement and Minutiae Extraction," University of Western Australia, Perth, Western Australia, Australia, 2003.

[5] W. Chen e Y. Gao, "A Minutiae-based Fingerprint Matching Algorithm Using Phase Correlation," em *9th Biennial Conference of the Australian Pattern Recognition Society on Digital Image Computing Techniques and Applications*, Glenelg, Australia, 2007.

Fernanda Baumgarten Ribeiro do Val é graduanda em Engenharia Elétrica com ênfase em Computação pela Escola Politécnica da Universidade de São Paulo(EPUSP), São Paulo, SP, Brasil. Trabalhou por 2 anos na empresa júnior da EPUSP (Poli Júnior) na aréa de Tecnologia da Informação e Gestão de Projetos. Em 2012, realizou intercâmbio, estudando durante 6 meses na Pontifícia Universidade Católica do Chile, onde aprofundou seus conhecimentos em invoção. Atualmente trabalha na Procter & Gamble na área de análise de categorias. Sua pesquisa se concentra em adaptatividade aplicada ao reconhecimento biométrico.

Priscila Ribeiro Marcelino é graduanda em Engenharia Elétrica com ênfase em Computação pela Escola Politécnica da Universidade de São Paulo (EPUSP), São Paulo, SP, Brasil. Trabalhou por 4 anos na empresa júnior da EPUSP (Poli Júnior), na qual atuou com desenvolvimento de software, gestão de projetos, dentre outras atividades. Atualmente trabalha na Procter & Gamble na área de Tecnologia de Informação. Sua pesquisa se concentra em adaptatividade aplicada ao reconhecimento biométrico.

João José Neto graduado em Engenharia Elétrica (1971), mestre em Engenharia Elétrica (1975) e doutor em Engenharia Elétrica (1980), e livre docente, professor associado (1993) na EPUSP - Escola Politécnica da Universidade de São Paulo. Coordena atualmente o LTA - Laboratório de Linguagens e Técnicas Adaptativas do Departamento de Engenharia de Computação e Sistemas Digitais da EPUSP. Sua especialidade é centrada em Ciência da Computação, com ênfase nos fundamentos da engenharia de computação e em Adaptatividade. Sua atividade principal inclui dispositivos adaptativos e tecnologia adaptativa, autômatos adaptativos e suas aplicações à engenharia de computação e outras áreas, em particular aos sistemas de tomada de decisão, processamento de linguagem natural, construção de compiladores e outros programas de software básico e sistemas operacionais, robótica, educação por computador, modelagem de sistemas inteligentes, aprendizado computacional, reconhecimento de padrões, inferência e outras aplicações baseadas na adaptatividade e em dispositivos adaptativos.

Elaboração de especificações adaptativas: uma abordagem biomimética

Í. S. Vega, C. E. P. de Camargo e F. S. Marcondes

***Abstract*—Projetos de software são caracterizados pelo desenvolvimento de soluções personalizadas, únicas, mesmo em situações que se apoiam em camadas computacionais reutilizáveis. Especificações comportamentais adaptativas foram propostas com o intuito de lidar, explicitamente, com esta natureza dos projetos de software. A capacidade intuitiva do modelador é peça-chave na elaboração de tais especificações, que pode ser ampliada caso ele busque inspiração na biologia. A biomimética pode ser utilizada para desenvolver esta intuição em projetos de software. Este artigo apresenta uma técnica que aplica o Método de Transposição Semiótica (MTS) no contexto de dispositivos adaptativos visando introduzir uma sistemática semiótica no processo de elaboração de especificações adaptativas. O conceito de unidade biomimética adaptativa habituação-sensibilização (UBA-HS) para aplicações de software é um dos importantes resultados do emprego desta técnica.**

***Palavras-chave:*—Especificação adaptativa, transposição semiótica, padrões de análise.**

I. Introdução

A biomimética é a ciência que estuda os modelos e processos da natureza com a finalidade de reproduzi-los em busca de soluções para problemas humanos [1]. Os processos da natureza incluem os fenômenos de natureza biológica e estes são os objetos de estudo neste trabalho. As aplicações mais comuns da biomimética relacionam-se ao desenvolvimento de produtos tangíveis inspirados em modelos de tais fenômenos. No entanto, mecanismos biológicos de caráter mais abstrato também podem servir como fontes de inspiração. Aqui, os campos da Ciência da Computação, da Bio-robótica e da Ciência Cognitiva fornecem vários exemplos: computação natural [2], bio-robótica [3] e inteligência artificial [4]. Estas são algumas áreas dentro desses campos que podem ser relacionados com abordagens biomiméticas. De forma mais ou menos direta, tais áreas inspiram-se em processos biológicos na busca de sistemas computacionais mais eficientes.

A ideia de buscar inspiração em organismos vivos para o desenvolvimento de algoritmos apoia-se na intenção de buscar novas perspectivas e soluções para problemas de processamento computacional de informações. Owens define um sistema computacional biologicamente inspirado "como um sistema de processamento de informações cuja estrutura e função foram projetados com inspiração em algum sistema biológico" [5]. Ele também introduz a noção de algoritmos bio-inspirados: são "uma classe de sistemas biologicamente inspirados restritos às preocupações computacionais". Na visão de Reddy, algoritmos evolucionários são aqueles inspirados na "sobrevivência dos mais adaptados ou nos princípios de seleção natural" [6]. Neste sentido, algoritmos evolucionários são "métodos computacionais inspirados pelos processos e mecanismos da evolução biológica". O uso da inspiração biológica originou diversos paradigmas, tais como algoritmos genéticos [7], programação genética [8], programação evolucionária [9], etc. Neste artigo, o caminho que conecta a biologia com os algoritmos será percorrido com o Método de Transposição Semiótica, proposto por Camargo [10].

O Método de Transposição Semiótica (MTS) tem a finalidade de transpor comportamentos biológicos de seu campo original para o campo computacional. Devido à complexidade inerente aos fenômenos naturais, o campo da semiótica foi utilizado como elemento intermediário entre os dois campos em questão, pois atua identificando aqueles aspectos essenciais a serem convertidos em cálculos computacionais.

Este artigo apresenta uma técnica de uso deste método geral no desenvolvimento de tipos específicos de entidades computacionais baseados em dispositivos adaptativos. As entidades computacionais assim elaboradas são formalizadas por meio das especificações adaptativas propostas por Vega [11]. A título de ilustração, consideram-se os fenômenos biológicos relacionados à capacidade de aprendizagem de um invertebrado marinho, a *Aplysia californica*. Este caso aponta para possibilidades de uso da entidade computacional resultante em domínios específicos e também aponta para a possibilidade de ampliar o escopo do MTS para a geração de vários modelos de dispositivos adaptativos decorrentes do estudo de outros fenômenos biológicos gerando, assim, um dicionário de modelos biomiméticos com especificações adaptativas próprias, aos moldes dos *analysis patterns* propostos por Fowler [12].

As demais seções deste artigo encontram-se assim organizadas. Na Seção II apresenta-se a relação primária entre a biomimética e a adaptatividade. A Seção III mostra como empregar especificações adaptativas para expressar modelos elaborados com a utilização do MTS. A Seção IV introduz a ideia de unidade biomimética adaptativa no contexto de padrões de projeto de software. A Seção V aponta na direção de possíveis usos desta técnica, como no caso do problema KWIC [13] e da *Internet das Coisas* [14].

II. Biomimética e Adaptatividade

O desenvolvimento de sistemas computacionais biomiméticos pode servir a dois propósitos distintos e excludentes. Primeiro, como ferramenta para biólogos estudarem o comportamento de um determinado animal e, segundo, como suporte para engenheiros estudarem e avaliarem algoritmos biologicamente inspirados para aplicações em engenharia [3]. São

Ítalo S. Vega, Carlos Eduardo P. de Camargo e Francisco S. Marcondes, Departamento de Computação, PUC-SP

propósitos que partem de bases similares, uma vez que ambas buscam a transposição de fenômenos do campo biológico para o campo computacional,mas radicalmente diferentes em seus objetivos. O biólogo procura simular todos os aspectos orgânicos de um dado fenômeno com exatidão. O engenheiro, por outro lado, procura inspirar-se no comportamento biológico subjacente ao organismo em questão, representando apenas o seu aspecto abstrato essencial, suficiente para chegar a um algoritmo.

Com o objetivo de auxiliar no processo de desenvolvimento de softwares biomiméticos cujos propósitos coincidam com o segundo tipo descrito acima, ou seja, o estudo de algoritmos biológicos para potenciais aplicações em engenharia, foi desenvolvido o MTS [10]. A hipótese fundamental do MTS é a de que a Semiótica pode ser utilizada como campo intermediário na transposição entre os campos biológico e computacional. Este método heurístico procura reconhecer as semioses que operam no campo biológico transpondo-as como funções algoritmicas ao campo computacional. Cada tríade semiótica — objeto/signo/interpretante — corresponde, respectivamente, a uma interpretação de processo mecânico também de caracter triádico — entrada/processamento/saída — implementada por um entidade de natureza computacional, tal como um módulo ou uma classe, etc, por exemplo.

O argumento de Searle considera que as operações de um cérebro podem ser computacionalmente simuladas, descartando-se a produção da sua consciência e das suas propriedades emergentes [15]. Neste trabalho, a atribuição de funções aos comportamentos e processos biológicos não tem o objetivo de produzir a consciência de um *Aplysia californica*. A intenção primária é de elaborar modelos computacionais biologicamente inspirados em tal espécie de animal. Por conseguinte, o primeiro caso de aplicação do MTS teve como objetos de estudo os comportamentos de habituação e sensibilização daquele invertebrado marinho [10]. Esses comportamentos de aprendizagem são essenciais à sobrevivência do animal por dotá-lo da capacidade de distinguir informações relevantes provenientes de seu *habitat*. Segundo Kandel [16], a habituação permite que o comportamento do animal adquira foco. Quando ainda na sua infância, o animal quase sempre responde com exagero a estímulos não ameaçadores. Habituar-se a tais estímulos faz com que o animal se concentre em estímulos realmente importantes para a organização de sua percepção e, consequentemente, para a sua sobrevivência. Por outro lado, a sensibilização é a imagem espelhada da habituação, fazendo com que o animal tenha respostas comportamentais acentuadas a qualquer estímulo, mesmo os não nocivos, após ser exposto a um estímulo verdadeiramente ameaçador. É uma espécie de medo aprendido [16] que aumenta o nível de atenção do animal em contextos específicos. O processo bio-químico subjacente a esta dinâmica comportamental apresenta fatores de adaptatividade inerentes aos circuitos neurais envolvidos. Tais fatores, por sua vez, apontam para semelhanças com os dispositivos adaptativos da teoria das linguagens formais.

Neto define o dispositivo adaptativo como um formalismo que tem a capacidade de alterar dinamicamente sua topologia e seu comportamento, de forma autônoma [17]. Em um modelo biomimético, as mudanças comportamentais de um organismo devem-se tanto a estímulos externos quanto a internos. A sua reação comportamental resulta da consideração de ambos os tipos de estímulos. Nos termos operacionais de um dispositivo adaptativo, o modelo biomimético de um comportamento pode ser especificado por um conjunto de regras automodificáveis dinamicamente, conforme será apresentado na Seção IV. Para isso, será explorado o mecanismo adaptativo do dispositivo proposto por Neto, responsável por estas características de transformação dinâmica do seu comportamento, como sugerido por Tchemra [18]. O mecanismo faz uso de três categorias de ações adaptativas elementares: consulta, exclusão e inclusão. Uma função adaptativa abstrai particulares combinações destas ações elementares.

III. Especificações Adaptativas

Neto e Silva propuseram um arcabouço adaptativo para suportar linguagens de especificação no contexto de um processo incremental de análise [19]. A ênfase do arcabouço é no dispositivo adaptativo utilizado como base para o projeto de uma linguagem de especificação com capacidades operacionais. Em um trabalho complementar, Vega introduziu a noção de métodos adaptativos como parte de uma semântica operacional para os autômatos adaptativos baseada no paradigma de objetos [20]: "A cada transição do autômato finito subjacente, será vinculada uma abstração do efeito conjunto das funções adaptativas *before* e *after* sobre a configuração da topologia de estados e transições do autômato." Assim, quando da ocorrência de um tipo de evento vinculado a uma transição adaptativa, executa-se o correspondente método adaptativo. Tal execução poderá provocar uma reconfiguração na topologia do autômato subjacente, em termos similares aos quais um autômato adaptativo se altera por força da aplicação de alguma função adaptativa. Métodos adaptativos serviram como fundamentação das *especificações adaptativas*. Uma apropriada combinação dos efeitos dos métodos adaptativos deverá ser o objetivo durante a elaboração de uma especificação de requisitos comportamentais [11]: "Associada a cada transição, dever-se-á descrever quais mudanças a configuração do autômato deverá sofrer para que se expresse o comportamento desejado."

Na Fig. 1 ilustra-se a configuração do autômato adaptativo AA na forma de um diagrama de estados com métodos adaptativos. Inicialmente, o autômato encontra-se no estado S1. De acordo com o modelo do diagrama, ele permanecerá neste estado enquanto houver alguma ocorrência de um evento do tipo e1. O método adaptativo **ma** será executado na ocorrência de um evento do tipo e2. Neste modelo, a execução de **ma** provocará uma reconfiguração do autômato adaptativo, tanto nos seus estados, quanto nas suas transições, como será apresentado a seguir.

A Fig. 2 ilustra a nova configuração do autômato, decorrente da execução do método adaptativo **ma**. Observa-se que o estado S1 foi substituído pelo estado S2 nesta configuração. Além disso, manteve-se a reação comportamental em relação a eventos do tipo e1, mas não em relação a eventos do tipo e2. Do ponto de vista comportamental, o autômato permanecerá sensível aos eventos e1 no estado S2. Nenhum outro tipo de evento será tratado pelo autômato AA deste ponto em diante no tempo.

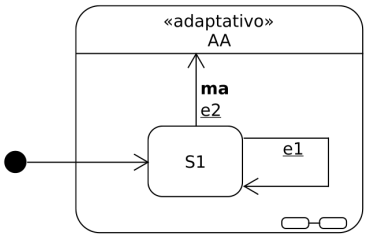


Figura 1. Configuração inicial do autômato AA

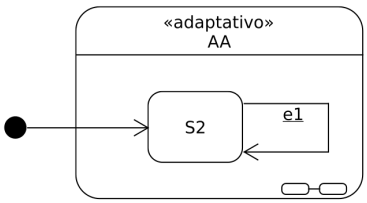


Figura 2. Configuração do AA depois do e2

Além da forma gráfica, a especificação do exemplo comportamental do autômato AA pode ser apresentada na forma de uma tabela de transições adaptativas, como mostrado por Vega [11]. A tabela ajuda na definição do modelo. No entanto, a atividade de elaboração de um modelo computacional ainda se apoia na capacidade intuitiva do especificador. O MTS pode ser utilizado para estimular esta intuição. Na próxima seção esta ideia será ilustrada em um caso de especificação adaptativa biomimética.

IV. Unidade Biomimética Adaptativa HS

A aplicação do MTS ao comportamento da *Aplysia* (Seção II) resultou num metamodelo computacional que pode ser representado por dois estados: habituado e sensibilizado. O estado Habituado, por sua vez, subdivide-se em dois subestados: Normal e Alerta. As transições entre estados decorrem de um estímulo externo combinado com um fator de ajuste interno que regula a tendência e velocidade com que as transições são realizadas. Assim, esta unidade computacional pode ser aplicada à solução de problemas algoritmicos que exijam certo tipo de capacidade de aprendizagem ou de adaptabilidade [10]. Na teoria das linguagens formais, os dispositivos adaptativos são assim considerados por seus comportamentos se adaptarem dinamicamente como resposta espontânea aos estímulos de entrada que os alimentam [11]. Por conseguinte, justifica-se a tentativa de se explorar os possíveis resultados que podem ser alcançados tomando-se o uso do metamodelo proveniente da aplicação do MTS no comportamento de aprendizagem do *Aplysia* como base para uma técnica de modelagem de software adaptativo. Este artigo propõe o uso desta unidade de modelagem como base para uma técnica de modelagem de software adaptativo, o que será descrito a seguir.

Inicialmente, elabora-se um modelo com um estado Habituado, constituído por dois subestados: «normal» N e um estado de «alerta» A. Estes dois estados são interligados por transições desencadeadas por eventos dos tipos eh e es, que modelam um comportamento computacional de alternação entre eles. Além disso, existe um método adaptativo **mh** que, ao ser executado, introduz um novo estado na configuração: «sensibilizado» S com uma autotransição associada ao evento es e outra, associada ao método adaptativo **ms**, que restaura a configuração inicial do autômato subjacente. Aquela transição adaptativa também remove os dois estados originais (normal e atenção), bem como as duas transições que os interligam. A execução do método adaptativo **ms** restaura os estados N e A, bem como as transições com os eventos en, ea e método adaptativo **mh**.

Uma unidade biomimética adaptativa habituado-sensibilizado (UBA-HS) é constituída por três categorias de estados: normal, alerta e sensibilizado, respectivamente denotados por *N*, *A* e *S* na Fig. 3. O estereótipo "*adaptativo*" indica que se trata de um estado potencialmente alterado pelo mecanismo de métodos adaptativos [11]. Na referida unidade, qualquer aplicação de um método adaptativo poderá transformar os estados de uma categoria de estados.

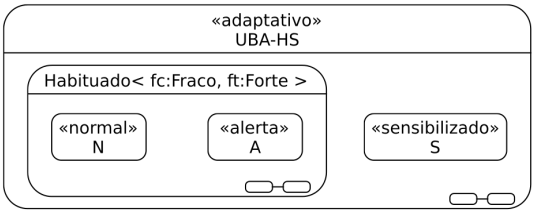


Figura 3. Modelo de estados da UBA-HS

Na configuração inicial, o dispositivo possui apenas estados da categoria «normal» e «alerta» (Fig. 4). Um estado da categoria «normal» suporta apenas estímulos externos do tipo eh, parametrizado por um sinal de fraca intensidade fc e outro de forte intensidade ft. Considera-se que o dispositivo ainda não se encontra habituado a ocorrências de estímulos do tipo eh. Assim, na ocorrência de um estímulo eh com um sinal fraco fc, o dispositivo entra em estado de alerta A. Sucessivas ocorrências de estímulos deste tipo — oriundas de um sinal fraco fc — fazem o dispositivo permanecer no estado de alerta A. Ao longo do tempo, uma condição de habituação irá se estabelecer. No *Aplysia*, tal condição está relacionada às alterações metabólicas que lhe é característico, sendo desencadeadas pela presença de uma certa quantidade de estímulos eh do sinal fc. O estabelecimento desta condição de habituação é capturado em um contador arbitrário, integrante do modelo do autômato subjacente da UBA-HS. Assim, o dispositivo se habitua e fica insensível a ocorrências adicionais de eh oriundas do sinal fc. Deste momento em diante, o dispositivo permanecerá no estado N. Duas situações podem alterar o estado do dispositivo em tais tipos de configuração. Em uma delas, mesmo diante de um sinal fc, o estado da UBA-HS se altera para A por decorrência da perda de memória em relação a fc. Em outra situação, o seu comportamento será alterado quando houver a ocorrência de um estímulo do tipo eh provocado por um sinal forte ft. Esta transição encontra-

se associada ao método adaptativo **mh** que, ao ser executado, altera a configuração do autômato adaptativo UBA-HS.

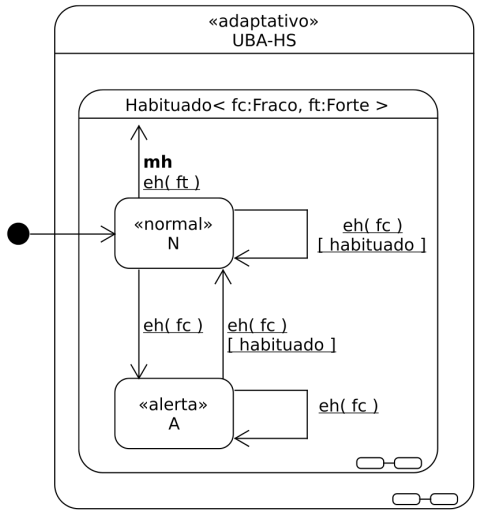


Figura 4. Configuração inicial do dispositivo adaptativo

Na ocorrência de uma transição adaptativa quando o dispositivo encontra-se em algum dos seus estados da categoria "*normal*", a configuração se altera com a incorporação de estados das outras categorias de estados biomiméticos (Fig. 5). Em tais configurações, a UBA-HS passa a exibir uma riqueza maior de comportamentos, ora guiados por estímulos externos, ora guiados por algum fator interno, indicado por *nível*. Alterações no valor deste nível podem ser modeladas por meio de diferentes estratégias probabilísticas.

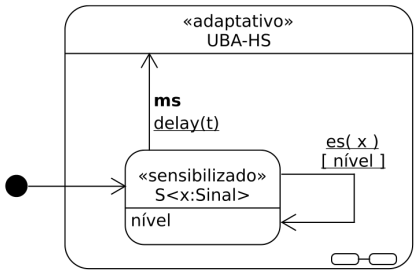


Figura 5. Efeito da função adaptativa Ah

V. Conclusão

Este artigo mostrou a utilização do MTS em conjunto com dispositivos adaptativos para a elaboração de especificações adaptativas no sentido proposto por Vega [11]. Partindo de fenômenos biológicos exibidos pelo animal *Aplysia*, o emprego das semioses do MTS produziu um modelo comportamental que combinou estados de habituação e de sensibilização. Estes são estados que modelam a reação do animal a diferentes tipos e intensidades de sinais por ele percebidos. Um dispositivo adaptativo foi utilizado para modelar os particulares fenômenos que desencadeiam as mudanças comportamentais do animal. A técnica de especificações adaptativas suportou a representação do modelo adaptativo assim elaborado. Um posterior refinamento originou um padrão de comportamento chamado de unidade biomimética adaptativa habituação-sensibilização (UBA-HS), também proposto neste artigo. Estes resultados apontam na direção de um catálogo de padrões biomiméticos adaptativos que se encontra em elaboração nesta pesquisa. Em particular, uma versão preliminar da UBA-HS foi utilizada em um ensaio de modelagem do KWIC [13] e está em estudo o seu emprego no contexto da *Internet das Coisas* [14], os quais não foram alvo de discussão detalhada neste trabalho.

Referências

[1] J. M. Benyus, *Biomimética: Inovação inspirada pela natureza*. São Paulo, SP, Brasil: Cultrix, 2003.
[2] L. N. Decastro, "Fundamentals of natural computing: an overview," *Physics of Life Reviews*, pp. 1–36, 2007.
[3] B. Webb and T. R. Consi, *Biorobotics, methods and applications*. Cambridge, MA, USA: MIT Press, 2001.
[4] J. McCarthy, M. L. Minsky, N. Rochester, and C. Shannon, "A proposal for the dartmouth summer research project on artificial intelligence," *AI Magazine*, vol. 27, pp. 12–14, 1955.
[5] N. D. L. Owens, "From biology to algorithms," Ph.D. dissertation, University of York, 2010.
[6] M. J. Reddy and D. N. Kumar, "Computational algorithms inspired by biological processes and evolution," *Current Science*, vol. 103, pp. 370–380, 2012.
[7] M. Mitchell, *An introduction to genetic algorithms*. Cambridge, MA, USA: The MIT Press, 1998.
[8] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Proceedings of the 1st international conference on genetic algorithms*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985, pp. 183–187.
[9] J. Brownlee, *Clever algorithms: Nature-inspired programming recipes*. Lulu Enterprises Incorporated, 2011.
[10] C. E. P. Camargo, "Método de transposiçãão semiótica para modelagem computacional biomimética," Master's thesis, Pontifícia Universidade Católica de São Paulo, 2014.
[11] I. S. Vega, "Especificações adaptativas e objetos, uma técnica de design de software a partir de statecharts com métodos adaptativos," in *Memórias do VI Workshop de Tecnologia Adaptativa – WTA 2012*, São Paulo, SP, Brasil, 2012, pp. 68–79.
[12] M. Fowler, *Analysis patterns reusable object models*. Addison Wesley, 1996.
[13] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, pp. 1053–1058, 1972.
[14] K. Ashton, "That 'internet of things' thing: In the real world, things matter more than ideas," *RFID Journal*, June 2009.
[15] J. R. Searle, *The Rediscovery of the Mind (Representation and Mind)*. The MIT Press, 1992, paperback.
[16] E. R. Kandel, *Em busca da memória: O nascimento de uma nova ciência da mente*. São Paulo, SP, Brasil: Companhia das Letras, 2006.
[17] J. J. Neto, "Adaptive rule-driven devices – general formulation and case study," in *International conference on implementation and application of automata*, E. Springer-Verlag, Ed., 2001, pp. 234–250.
[18] A. H. Tchemra, "Application of adaptive technology in decision making systems," *IEEE Latin America Transactions*, vol. 5, 2007.
[19] J. J. Neto and P. S. M. Silva, "An adaptive framework for the design of software specification languages," in *Proceedings of international conference on adaptive and natural computing algorithms*, 2005, pp. 21–23.
[20] I. S. Vega, "An adaptive automata operational semantic," *IEEE Latin America Transactions*, vol. 6, pp. 461–470, 2008.

Evolução de Bancos de Dados Utilizando Planejamento Automático

M.B.P. Domingues e J. R. de Almeida Jr

***Abstract*—O projeto e a implementação de bancos de dados evolutivos são importantes desafios para o desenvolvimento de sistemas de computação, tendo em vista as frequentes mudanças de requisitos das aplicações e a necessidade de fazer as atualizações em bancos de dados que atendem, simultaneamente, várias aplicações. Atualmente na literatura, existem soluções síncronas e assíncronas que utilizam refatorações para evoluir o banco de dados. Essas refatorações representam pequenas alterações estruturais, arquiteturais, de integridade e qualidade de dados que não incluem funcionalidades novas ao sistema. É muito comum a necessidade de implementação de uma grande alteração no banco de dados, a qual envolva o uso de um conjunto de refatorações programadas. Esse conjunto deve ser organizado de modo que, os requisitos de mudança dos usuários sejam atendidos e que o conjunto de restrições presentes no modelo de dados sejam satisfeitos. Atualmente técnicas de planejamento automatizado têm sido usadas para modelar problemas de planejamento de tarefas em diferentes domínios de problemas. Tais modelos possuem, de forma geral, suas ações com precondições e efeitos de suas transições e estados, restrições, recursos disponíveis e objetivos a serem atingidos O objetivo deste trabalho é utilizar técnicas de planejamento automatizado para representar o conjunto de refatorações e propor um modelo que inclua as fases para a conclusão de uma grande alteração no banco de dados.**

***Keywords*— Evolução de bancos de dados, Refatorações de banco de dados, Planejamento automático.**

I. INTRODUÇÃO

A EVOLUÇÃO de esquemas de bancos de dados tem sido reconhecida como um dos principais obstáculos que dificultam as atualizações em sistemas de informação[1]. Estudos mostram que essas dificuldades aumentaram com o uso de sistemas *Web*, nos quais a frequência de mudanças é muito grande e praticamente não há tolerância para erros ou indisponibilidade dos serviços.

Com a chegada das Metodologias Ágeis para Desenvolvimento de Software alguns conceitos das metodologias tradicionais foram substituídos por modelos iterativos e incrementais, que não precisam ter todo o modelo lógico e físico do banco de dados para o início do desenvolvimento do software[2]. Isso significa organizar o desenvolvimento do sistema em várias iterações de desenvolvimento e entregas ao cliente. Cada iteração contempla todas as fases já conhecidas nos modelos tradicionais como planejamento, especificação de requisitos, projeto, codificação e testes[3]. Essas iterações que normalmente duram em torno de duas semanas, têm como principais vantagens a detecção antecipada de erros de requisitos e de implementação e também a flexibilidade de mudar os requisitos no decorrer do projeto. Para atender a essas mudanças de requisitos é necessário que o banco de dados também tenha um desenvolvimento iterativo, ou seja, à medida que o software é desenvolvido, o banco de dados precisa ser alterado para acompanhar os novos requisitos. As alterações no esquema de banco de dados são chamadas de refatorações de bancos de dados.

Refatorar um banco de dados pode ser mais trabalhoso do que refatorar um código-fonte de uma aplicação, embora, segundo Martin Fowler[2]possa seguir as mesmas premissas. Tal afirmação baseia-se no fato de que é preciso se preocupar com todas as instâncias das aplicações que utilizam o banco de dados; realizar as alterações respeitando os requisitos dos usuários; preservar os dados existentes; não modificar a semântica do esquema atual e manter a integridade do banco de dados.

As técnicas de refatoração de código de Martin Fowler[2] foram adaptadas para bancos de dados por Ambler e Sadalage [1]e inseridas na literatura de Métodos Ágeis para Desenvolvimento de *Software*. Um refatoração de banco de dados foi definida como uma pequena mudança no esquema que melhora o projeto, não adiciona funcionalidades e não altera a semântica informacional ou comportamental do esquema.

Entretanto, nem todas as alterações se encaixam nessa definição de refatoração. É muito comum a necessidade de implementação de grandes alterações em bancos de dados. Isso implica escolher um conjunto de refatorações que represente o estado final desejado do esquema do banco de dados. Para a escolha desse conjunto de refatorações é necessário o conhecimento prévio do esquema atual do banco de dados, bem como suas limitações e objetivos. Também é necessária a escolha de um conjunto de tarefas que modifique o esquema sem negligenciar a integridade do banco de dados[4].

Neste trabalho foram utilizadas técnicas de planejamento automatizado, pararepresentar um conjunto de refatorações utilizado em uma grande alteração de banco de dados.Esta representação, foi adaptada dos trabalhos de Reiter [5],McCarth[6]e Nau et al. [7]. Ainda neste trabalho é proposto um modelo para o planejamento e análise de uma grandealteração de banco de dados.

II. REFATORAÇÃO DE BANCO DE DADOS

Segundo Ambler e Sadalage [1]uma refatoração em um banco de dados pode ser definida como uma alteração simples em seu esquema, com o objetivo de melhorar seu projeto, preservando sua semântica informacional e sua semântica

comportamental. Assim, para se realizar uma refatoração não é possível adicionar novas funcionalidades ou modificar alguma existente, não se pode alterar um dado e nem alterar o significado de um dado existente. Quando há necessidade de modificar ou adicionar algo existente é necessário aplicar o conceito de transformação, também definido por [1].

Em outras palavras, após a execução de uma refatoração ou um conjunto de refatorações no esquema de dados, o banco de dados deve responder às mesmas consultas que eram realizadas antes deste processo de refatoração.

Esta estratégia de refatoração é aplicada em ambientes complexos, nos quais existem várias aplicações acessando simultaneamente o mesmo banco de dados. Quando esse banco precisa ser alterado, praticamente todas as aplicações precisam sofrer algum tipo de adaptação, sendo que não é possível supor que todas essas aplicações serão alteradas ao mesmo momento [8]. Por esse motivo, foi proposto por [1]que se tenha um período de transição entre o início e o fim de uma refatoração, no qual o esquema antigo e o esquema novo coexistam, como mostrado na Fig. 1.

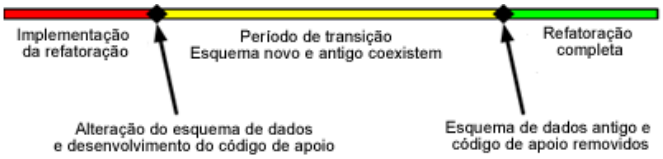


Figura 1. Ciclo de vida de uma refatoração.

Ambler e Sadalage [1]destacam alguns problemas no banco de dados que podem exigir refatorações como: tabelas ou colunas com múltiplas funções; tabelas com muitas colunas; e tabelas com colunas ou relacionamentos obsoletos.

Essas refatorações estão apresentadas na Tabela I, divididas em 4 grupos: estrutural, qualidade de dados, integridade referencial e arquitetural.

TABELA I. CATEGORIAS DE REFATORAÇÕES DE BANCO DE DADOS

Tipo	Aplicação
Estrutural	▪ Remover tabelas, colunas ou visões; ▪ Dividir ou unir tabelas ou coluna; ▪ Renomear tabelas, colunas ou visões.
Qualidade de Dados	▪ Introduzir restrição de coluna, valor padrão ou formato comum; ▪ Aplicar código ou tipo padrão; ▪ Remover restrição de coluna, valor padrão ou restrição de não nulo.
Integridade referencial	▪ Adicionar ou remover restrição de chave estrangeira; e ▪ Adicionar ou remover a exclusão em cascata.
Arquitetural	▪ Introduzir índice, tabela espelho, tabela somente para leitura; e ▪ Adicionar métodos (*Stored Procedures*).

III. PLANEJAMENTO AUTOMÁTICO

Um plano é uma sequência de ações para se atingir um objetivo. Planejamento é um processo de deliberação que escolhe e organiza ações para antecipar seus resultados esperados. Esta deliberação tem por objetivo atingir, da melhor forma possível, alguns objetivos conhecidos. Planejamento automatizado é uma área de Inteligência Artificial (IA) que estuda, computacionalmente, este processo de deliberação[7].

Este plano que segue ações apropriadas e o objetivo pode ser especificado de várias maneiras diferentes. A mais simples consiste na especificação de um estado objetivo (*goal state*) S_g, ou um conjunto de estados objetivo. Neste caso, o objetivo é alcançado por qualquer sequência de transições de estado que termina em um dos estados objetivo. Em outras palavras, o objetivo é satisfazer alguma condição sobre uma sequência de estados seguida pelo sistema. Outra alternativa consiste em indicar os objetivos como tarefas que o sistema deve executar, conforme está apresentado na Fig. 2. Estas tarefas podem ser definidas recursivamente como um conjunto de ações e outras tarefas.

Nau et al. [7] descreve esse modelo por meio da interação entre três principais componentes como mostrado na Fig. 2 e descritos a seguir.

Figura 2. Modelo Conceitual de um planejamento automático.

Planejador (*Planner*) recebe como entrada a descrição de um sistema Σ, uma situação inicial e objetivos que sintetizam o plano para o controlador executar e atingir o objetivo;

Controlador (*Controller*) recebe o estado atual (s) do sistema e providencia como saída uma ação, de acordo com o plano escolhido pelo planejador;

Sistema estado-transição Σ (*System* Σ) evolui de acordo com as ações e eventos que este recebe (um sistema estado-transição pode ser representado por um grafo direcionado onde os nós são estados e os arcos são ações ou eventos).

No modelo clássico de planejamento o sistema estado-transição pode ser definido formalmente como uma 4-tupla Σ=(S,A,E,γ), onde:

- $S = \{s_1, s_2, \ldots\}$ é o conjunto finito ou recursivo dos estados;
- $A = \{a_1, a_2, \ldots\}$ é o conjunto finito ou recursivo de ações;
- $E = \{e_1, e_2, \ldots\}$ é o conjunto finito ou recursivo dos eventos exógenos (não controlados pelo agente);
- $\gamma: S \times (A \cup E) \rightarrow 2^s$ é a função de estado-transição.

O sistema Σ da Fig. 2 pode ser representado por um grafo dirigido cujos nós são estados em *S*. Se $s' \in \gamma(s,u)$, onde *u* é o par (*a*, *e*), para todo $a \in A$ e $e \in E$, então o grafo contem um arco (chamado de estado transição) de *s* para s', rotulado com *u* como mostrado na Fig. 3.

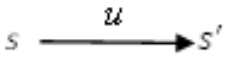


Figura 3. Estado transição de s para s'.

A ação *a* aplicada ao estado *s* leva a um outro estado $\gamma(s,a)$, assim o sistema evolui através de eventos e ações. A função de observação $\eta: S \rightarrow O$ faz o mapeamento de *S* em um conjunto discreto $O = \{o_{1,} o_{2,\dots}\}$ de possíveis observações. A entrada do controle é então a observação $o = \eta(s)$ que corresponde ao estado atual *s*, a entrada do planejador é a descrição de Σ, estado inicial $s_{0,} \in S$ e um estado objetivo e a saída do planejador produz um plano para guiar o controle.

Segundo Nau et al. [7]este modelo conceitual não pretende ser diretamente operacional. Pelo contrário, é usado como uma referência para as representações. Considera-se este modelo como um ponto inicial para avaliação de propriedades restritivas, particularmente as que seguem:

1. Σ **Finito:** O *sistema estado-transição* Σ possui um conjunto finito de estados $S = \{s_{1,} s_{2,\dots,} s_{k,}\}$para algum *k;*
2. Σ **Totalmente Observável:** O sistema Σ é totalmente observável, ou seja, conhece-se por completo o estado de Σ. Nesse caso a função η é a função identidade.
3. Σ **Determinístico:** O sistema Σ é determinístico, ou seja, cada ação ou evento tem apenas uma saída possível *u* em *A*∪*E*, |γ(*s*,*u*)| = 1;
4. Σ **Estático:** O sistema Σ é estático, isto é, o conjunto de *Eventos* exógenos E é vazio. Nenhuma mudança ocorre exceto aquelas efetuadas pelo controle
5. **Objetivos restritos:** Os planejadores lidam apenas com *objetivos restritos* que são especificados explicitamente no estado objetivo (*goal state*) s_g ou um conjunto de estados objetivo S_g. O objetivo é qualquersequência de estado-transição que termina em um dos estados objetivo. Os *objetivos estendidos*, tais como, estados a serem evitados, restrições na trajetória da solução ou funções de otimização, não são permitidos;
6. **Planos Sequenciais:** Um plano-solução de um problema de planejamento é uma sequência finita de ações ordenadas (a_1, a_2, *... a_k*);
7. **Tempo Implícito:** Ações e eventos não possuem duração. Elas são transições de estado instantâneas;
8. **Planejamento *Offline*:** O planejador não percebe mudanças do sistema Σ enquanto está planejando. Ele planeja apenas com as condições iniciais e os objetivos, independentemente da dinâmica que ocorre em Σ. O planejador não sabe o status da execução.

O planejamento clássico tem como suposições o conhecimento completo sobre um sistema determinístico, estático, estado-finito com satisfação de metas e tempo implícito. Segundo Nau et al. [7]o planejamento pode ser reduzido ao seguinte problema:

Dado$\Sigma = (S, A, \gamma)$, um estado inicial S_0, um conjunto de estados objetivo S_g encontre uma sequência de ações (a_1, a_2, *... a_k*) que corresponda a uma sequência de estados-transições ($s_{1,} s_{2,\dots,} s_{k,}$) tal que, $s_1 \in \gamma(s0, a1)$, $s_2 \in \gamma(s_1, a_2)$,..., $s_k \in \gamma(s_k–1, a_k)$ onde s_n pertença à S_g.

As restrições e suposições descritas acima pretendem tornar o planejamento apenas um processo de busca por um caminho no grafo de estados, o que é um problema bem conhecido e bem resolvido pela Ciência da Computação. De fato, se possuíssemos o grafo que representa Σ explicitamente não haveria muito planejamento a se fazer. Todavia, mesmo para um domínio de planejamento bem simples, o grafo de Σ pode ser tão grande que, representá-lo, seria inviável. Por esse motivo, é necessário representar o domínio com uma representação reduzida do sistema Σ facilitando a busca por uma solução.

Vários modelos interessantes são obtidos através do relaxamento das suposições restritivas do modelo clássico. Esse relaxamento fez com que os planejadores fossem capazes de lidar com problemas mais complexos, e esse novo modelo foi chamado de Planejamento Neoclássico [7].

O avanço das técnicas de Planejamento em Inteligência Artificial fez com que a Engenharia de Requisitos e a Engenharia do Conhecimento pudessem ser usadas como importantes ferramentas para projeto de engenharia (*Engineering Design*). Esta crescente área da IA está presente em cenários como: planejamento de trajetória e movimentação de sistemas automáticos móveis; planejamento de percepção envolvendo ações de sensoriamento para captação de informação do ambiente; planejamento de navegação que combina sensoriamento e definições de trajetórias; planejamento de manipulação relacionado com movimentação de objetos como, por exemplo, montagem de peças, organização de containers, gestão de processos de negócios e aplicações de *Business Intelligence* [7] e [9].

Uma entrada necessária para qualquer algoritmo de planejamento é uma descrição do problema para ser resolvido. Na prática, seria impossível para esta descrição do problema incluir uma enumeração explícita de todos os possíveis estados e transições de estado. A descrição do problema seria excessivamente grande e mais trabalhosa que resolver o próprio problema de planejamento, assim é mais fácilcomputá-los e descreve-los quando forem necessários[7]. Para o Planejamento Automático, o mais importante é a representação e modelagem dos domínios, ou seja, a representação do sistema Σ. Essa representação deve ser feita em alguma linguagem que possa ser compreendida por um planejador.

IV. PLANEJAMENTO AUTOMÁTICO PARA REFATORAÇÃO DE BANCOS DE DADOS

Os trabalhos de Ambler e Sadalage [1] e Domingues [10], apresentaram como alternativas o uso de pequenas mudanças no esquema do banco de dados chamadas de refatorações. Ambos os trabalhos não consideram a execução de um conjunto de refatorações em um mesmo esquema de dados. Essa situação é bastante comum em ambientes reais, já que os requisitos dos usuários, muitas vezes, envolvem diversas funcionalidades do sistema. Ambler e Sadalage [1]sugeririamo uso das refatorações bem como as tarefas de cada uma delas, mas não estudaram como seria a execução de várias refatorações para se atingir um estado final do esquema do banco de dados.

A proposta deste trabalho é representar uma grande alteração no esquema do banco de dados, por meio de um conjunto de refatorações que expressem os desejos de mudança dos usuários. As expectativas dos usuários são diferentes, dependendo do envolvimento deles com o desenvolvimento do projeto. Um usuário desenvolvedor tem um ponto de vista de requisito diferente do usuário que administra o banco de dados e do usuário que gerencia o projeto.

Esses diferentes pontos de vista podem dificultar a fase de levantamento de requisitos e implicar em mudanças equivocadas no modelo do banco de dados. Para esses casos, os diagramas de UML representam uma boa solução para resolver dúvidas sobre os requisitos. Os estados iniciais, do esquema de banco de dados, e final, após as mudanças são conhecidos por todos os usuários envolvidos. Deste modo, entre o estado inicial e final do esquema de dados deve ser implementado um conjunto de refatorações que atinjam o estado final após a grande alteração no esquema de dados, como apresentado na Fig. 4.

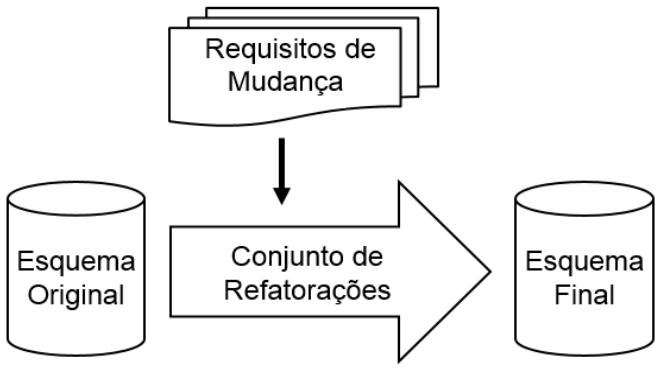


Figura 4. Modelo para evolução do esquema original para o esquema final do banco de dados.

O conjunto de refatorações deve ser escolhido utilizando o catálogo de Ambler e Sadalage [1]. Embora todos os usuários conheçam o estado final do esquema de dados, somente o administrador do banco de dados (DBA) irá se preocupar com o planejamento dessa mudança, ou seja, quais scripts serão criados e qual a ordem que eles serão executados para que se atinja o esquema final.

As refatorações de um conjunto podem ser arranjadas e combinadas de formas diferentes para que atinjam o esquema final, mas que tenham tempos diferentes para o término da grande alteração.

O conjunto de refatorações deve ser executado considerando todas as restrições impostas pelo modelo do banco de dados, ou seja, nem todo arranjo ou combinação das refatorações pode levar ao estado final esperado.

Considerando que um conjunto de refatorações seja um conjunto de ações a serem executadas no esquema inicial do banco de dados, e que a cada refatoração executada, aqui chamada de ação, o esquema de dados atinge um estado diferente, propõe-se neste trabalho representar uma grande alteração no banco de dados como sistema estado-transição definido por pelo sistema $\Sigma = (S, A, \gamma)$, onde:

- $S = \{s_{1,} s_{2,\ldots}\}$ é o conjunto finito de estados do esquema do banco de dados;
- $A = \{a_{1,} a_{2,\ldots}\}$ é o conjunto finito de ações (refatorações);

Dado $\Sigma = (S, A, \gamma)$, um estado inicial S_0, um conjunto de estados objetivo S_g deve-se encontrar uma sequência de ações (a_1, a_2, ... a_k) que corresponda a uma sequência de estados-transições ($s_{1,} s_{2,\ldots,} s_{k,}$) tal que, $s_1 \in \gamma(s0, a1)$, $s_2 \in \gamma(s_1, a_2)$,..., $s_k \in \gamma(s_k{-}1, a_k)$ onde s_n pertença à S_g. Tal como proposto por Nau et al. [7]. No sistema Σ de uma grande alteração ainda é necessário estudar o conjunto de restrições que envolvem cada ação do planejamento.

Para o Sistema estado-transição Σ de uma grande alteração pode-se relacionar as propriedades restritivas do modelo clássico de Nau et al. [7]. Assim o sistema Σ é:

- Finito, pois o conjunto de refatorações é conhecido;
- Totalmente observável;
- Determinístico, pois a cada ação (refatoração) executada existe apenas uma saída possível, ou seja, existe um único estado possível para o banco de dados após a execução da refatoração.
- Estático: Nenhuma mudança ocorre exceto aquelas executadas pelo controle;
- Objetivos restritos: O objetivo é qualquer sequência de estado-transição que leve a um estado final.
- Planos sequenciais;
- O tempo pode ser um parâmetro para execução das ações;
- Planejador off-line;

É necessário criar um plano de conhecimento do domínio, considerando o sistema estado-transição proposto, em linguagem PDDL(*Planning Domain Definition Language)*descrita porMcDermott [11].Nos estudos preliminares deste trabalho, identificou-se a necessidade de uma notação gráfica e uma notação formal, para intermediar os requisitos de uma grande alteração e a execução do seu plano de execução.

Na literatura são encontradas algumas técnicas de levantamento de requisitos que envolvem desde a criação,

representação e análise por diagramas esquemáticos de UML (*Unified Modeling Language*) e Redes de Petri.É possível combinar um método semiformal como diagramas de UML e métodos formais como representação em Redes de Petri. O diagrama de caso de uso de UML define uma visão que constitui um modelo funcional do sistema a partir da perspectiva do usuário [12]. Ambiguidades, contradições, omissões e imprecisões são frequentemente encontradas utilizando esse diagrama. Uma estrutura bem definida de casos de uso serve como base para futuras evoluções do banco de dados.

Em Domingues [13]as tarefas das refatorações foram descritas em Redes de Petri[14]. Nesse trabalho, também foi mostrado como é possível combinar as tarefas de cada uma delas, utilizando uma notação formal. Neste artigo, a pesquisa avançou com o objetivo de formalizar as refatorações utilizando o planejamento automático e propor um modelo para automatizar o plano de uma grande alteração em um banco de dados. Na Fig. 5 éapresentado um diagrama para o estudo do planejamento de uma grande alteração.

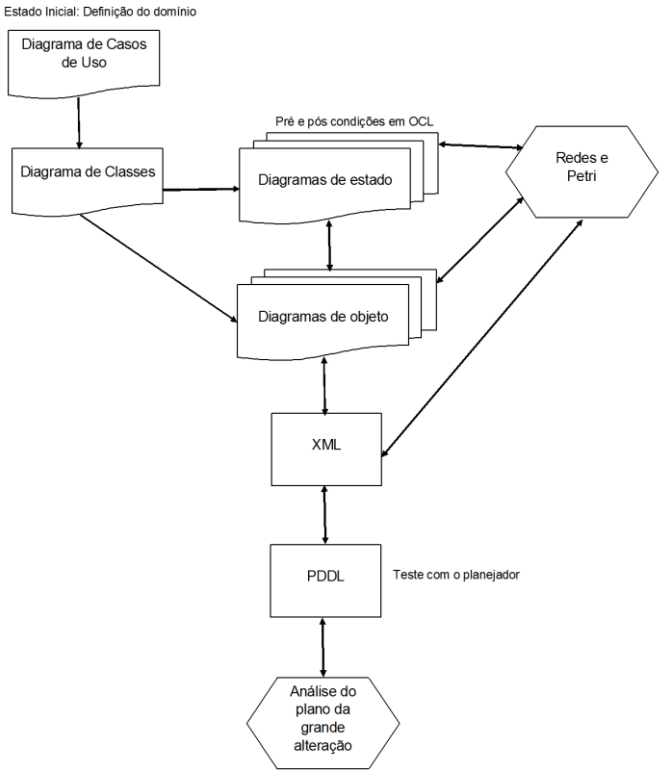


Figura 5. Modelo para o planejamento de uma grande alteração em um banco de dados

Nesse planejamento são esperados como entradas os diagramas casos de uso e de classes para representar o estado inicial do esquema de dados e os requisitos do usuário. Para traçar o plano de ações é necessária uma tradução dos diagramas UML, tais como diagramas de classes, objetos e sequencia, para linguagem PDDL que é interpretada por softwares planejadores. As pré e pós-condições são traduzidas para ações OCL (*Object Constraint Language*), a partir dos diagramas e estado de cada classe. Com o plano escrito em linguagem PDDL é possível a conversão para XML e posteriormente para Redes de Petri, na qual é possível realizar uma análise dinâmica do modelo e também das dependências do sistema.

Com a análise do plano das refatorações foi possível validar o plano de ações proposto pelos planejadores com os requisitos iniciais do usuário. Concluída essa etapa, foi possível traduzir as ações de saída do planejador para SQL (*Structured Query Language*). Desde modo, a solução proposta para planejamento de uma grande alteração pode ser resumida como segue na Fig. 6.

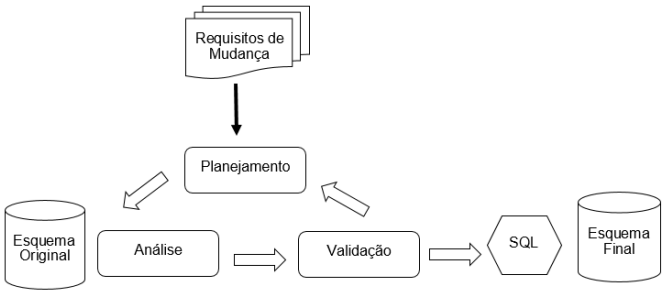


Figura 6. Modelo de transformação para SQL

Para Ambler e Sadalage [1]as alterações do modelo que não se encaixam no conceito de refatoração, por não modificarem ou adicionarem funcionalidades são chamadas de transformações. Assim a Fig. 4 é completada na Fig. 7 com o conceito de transformações para abranger qualquer alteração que esteja nos requisitos dos usuários.

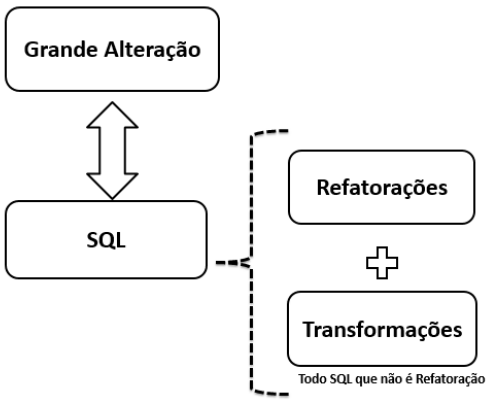


Figura 7. Modelo abrangendo as transformações

V. APLICAÇÃO DA PROPOSTA

A Fig. 8 mostra um exemplo de uma refatoração na qual será renomeada a coluna Pnome da Tabela Cliente para PrimeiroNome. Esse procedimento exige que sejam seguidos os passosda Fig. 9.

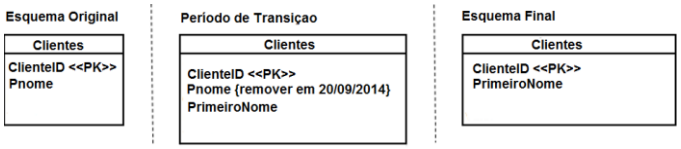


Figura 8. Exemplo de refatoração renomear coluna

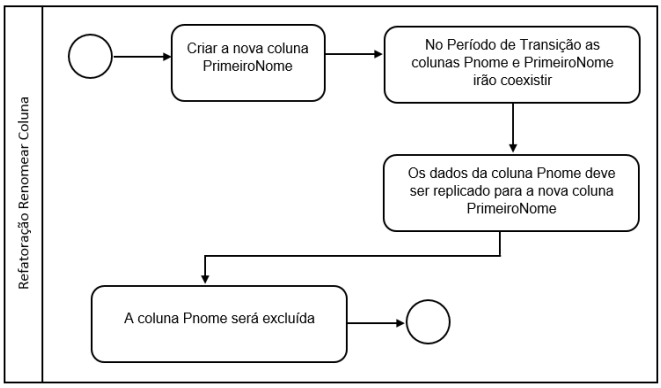


Figura 9. Passos para a refatoração renomear coluna

A refatoração Renomear Coluna está dentro do contexto de uma grande alteração (Fig. 4). O processo de uma grande alteração foi representado em um diagrama de classes conforme a Fig. 11. Neste diagrama conceitual todas as tabelas do modelo do banco de dados são representadas pela entidade "Tabela" e estão em relacionamentos com cardinalidade 1:N com colunas. Cada coluna, além dos atributos comuns como tipo de dados, possuem ainda atributos booleanos apenas para controle: excluída, quando a coluna não pertence mais à tabela; backup, quando os dados da coluna já foram replicados para a nova coluna e PeriodoTransicao, quando a coluna nova e a coluna antiga coexistem para manter a consistência do banco de dados. No diagrama de classe da Fig. 10 é representada a classe Refatoração Remover Coluna. Essa Classe possui métodos para realizar os passos descritos na Tabela I e herdam atributos e métodos da interface "Grande Alteração" como mostrado na Fig. 11.

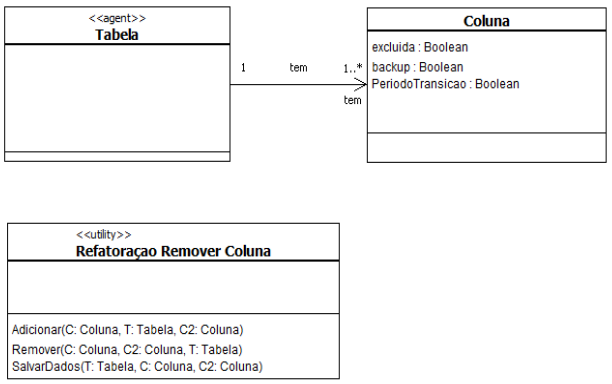


Figura 10. Exemplo de diagrama de classes conceitual para a Refatoração Remover Colunas

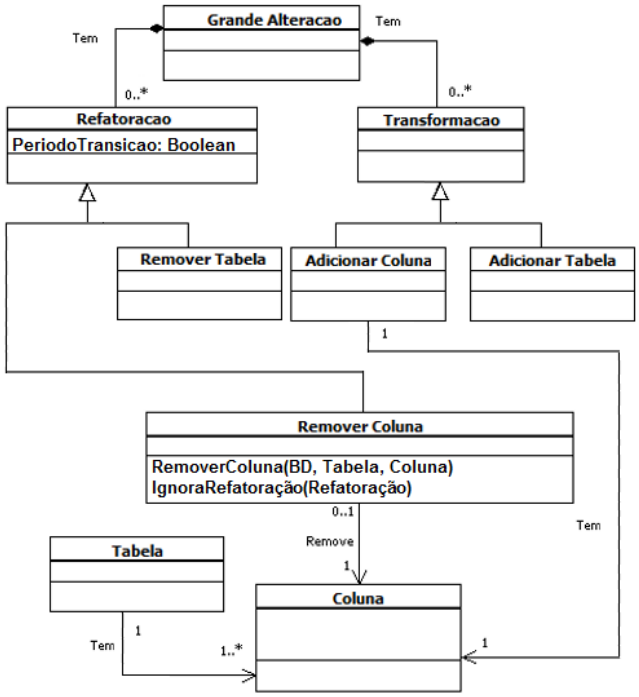


Figura 11. Diagrama de Classes: Conceitual para uma grande alteração

Na próxima fase é necesário instanciar os objetos reais do modelo do banco de dados, ou seja, instanciar a tabela "Cliente" e dois objetos da classe "Coluna": Coluna1 e Coluna2. Coluna1 representa a coluna existente Pnome que deverá ser substituída. Coluna2 representa a nova coluna PrimeiroNome. Ambas possuem os mesmos atributos, por se tratarem de instâncias da mesma classe. A Fig. 12 mostra estas instâncias e também o relacionamento já existente entre o objeto "Cliente" (tabela) e o objeto "Pnome" (coluna).

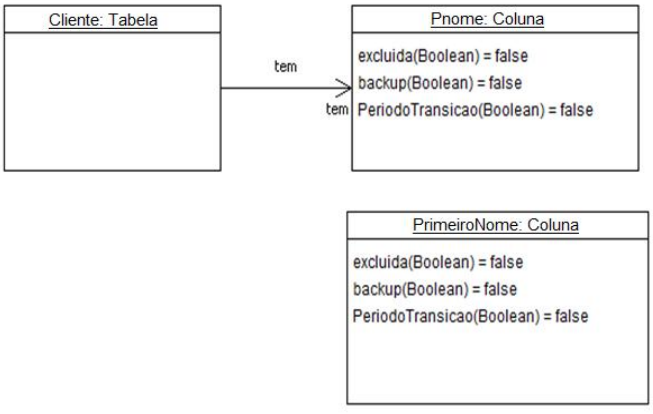


Figura 12. Instância dos objetos do para representar as tabelas do modelo do banco de dados.

Outra forma de representar os passos da Fig. 9 é utilizar o um diagrama de estados. Este diagrama além de poder ser utilizado para simular os passos, pode ser traduzido para linguagem de planejadores automáticos como a PDDL. A Fig. 13 apresenta os estados para conferência da Coluna 1 já

existente (Pnome), a entrada no período de transição onde a Coluna 1 e Coluna 2 (PrimeiroNome) coexistem no modelo e o último estado no qual a Coluna 1 é excluída do modelo do banco de dados. Neste caso os atributos do objeto Pnome do diagrama da Fig. 12 são atualizados quando o processo entra no período de transição, no qual os dados precisam ser replicados para a nova coluna (backup = true) e ao término o valor do atributo Excluída é alterado para true para representar a exclusão física da coluna Pnome do modelo do banco de dados, como é mostrado na Fig. 13.

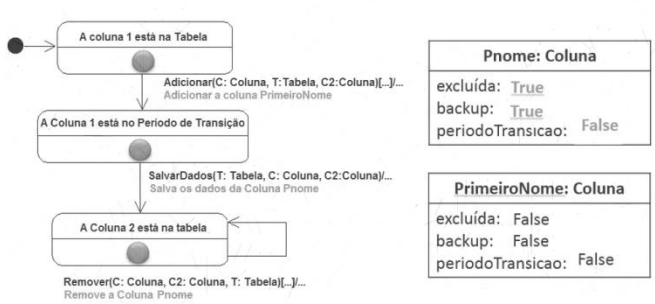


Figura 13. Diagrama de estados para remoção física da coluna Pnome do modelo do banco de dados

Após excluir a coluna Pnome é necessário refazer o relacionamento do diagrama conceitual da Fig. 10, para que o objeto "Cliente" (Tabela) se relacione agora como o objeto "PrimeiroNome" (Coluna) como é mostrado na Fig. 14.

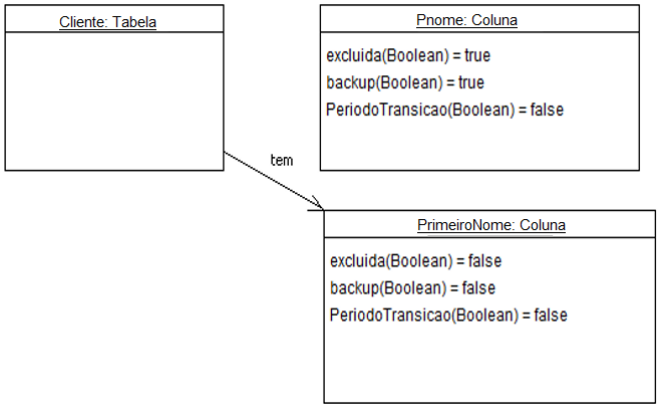
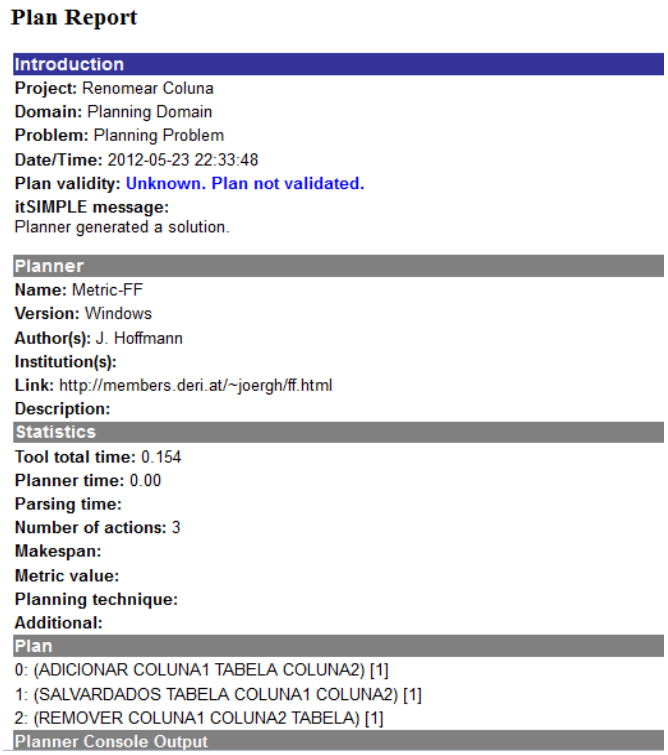


Figura 14.Instância dos objetos do para representar o relacionamento com a nova coluna criada.

O diagrama de estados da Fig. 14 foi traduzido para linguagem PDDL para ser executado por um aplicativo planejador. Como sugestão fizemos os testes com o planejador Metric-FF [16]. A seguir é apresentada a implementação em PDDL da refatoração Remover Coluna, como parte do processo de uma grande alteração, descrito na Fig. 11. A Fig. 15 mostra a execução do plano em PDDL para a refatoração Remover Coluna utilizando o planejador Metric-FF.

```
(define (domain Renomear_Coluna)
    (:requirements :typing :negative-
    preconditions)
(:types
      Tabela - object
      Coluna - object
 )
    (:predicates
      (tem ?tab - Tabela ?col - Coluna)
      (excluida ?col - Coluna)
      (backup ?col - Coluna)
      (PeriodoTransicao ?col - Coluna)
    )
    (:action Adicionar
     :parameters (?C - Coluna ?T - Tabela
     ?C2 - Coluna)
     :precondition
       (and
         (not (excluida ?C))
         (not (backup ?C))
(tem ?T ?C)
       )
     :effect
       (and
         (tem ?T ?C2)
         (not (excluida ?C2))
         (not (backup ?C2))
         (not (tem ?T ?C))
       )
 )

(:action SalvarDados
     :parameters (?T - Tabela ?C - Coluna
     ?C2 - Coluna)
:effect
       (and
         (backup ?C)
         (PeriodoTransicao ?C)
       )
 )
(:action Remover
     :parameters (?C - Coluna ?C2 -
     Coluna ?T - Tabela)
:precondition
       (and
(tem ?T ?C2)
         (backup ?C)
)
     :effect
       (and
         (excluida ?C)
         (not (tem ?T ?C))
         (not (PeriodoTransicao ?C))
)
 )
```



Plan Report

Introduction
Project: Renomear Coluna
Domain: Planning Domain
Problem: Planning Problem
Date/Time: 2012-05-23 22:33:48
Plan validity: Unknown. Plan not validated.
itSIMPLE message:
Planner generated a solution.

Planner
Name: Metric-FF
Version: Windows
Author(s): J. Hoffmann
Institution(s):
Link: http://members.deri.at/~joergh/ff.html
Description:

Statistics
Tool total time: 0.154
Planner time: 0.00
Parsing time:
Number of actions: 3
Makespan:
Metric value:
Planning technique:
Additional:

Plan
0: (ADICIONAR COLUNA1 TABELA COLUNA2) [1]
1: (SALVARDADOS TABELA COLUNA1 COLUNA2) [1]
2: (REMOVER COLUNA1 COLUNA2 TABELA) [1]

Planner Console Output

Figura 15. Execução do plano para a Refatoração Remover Coluna utilizando o planejador Metric-FF.

CONCLUSÕES

O trabalho inicial do administrador de banco de dados é coletar os requisitos de alterações dos usuários, identificar os problemas de modelagem ou que podem influenciar o desempenho de consultas. Esse conjunto de alterações no esquema de banco de dados é a entrada do processo refatoração que leva o esquema do banco de dados do modelo original até o final.Este trabalho teve como objetivo apresentar uma nova forma de realizar refatorações de banco de dados utilizando técnicas de planejamento automático. Estas técnicas incluem atividades multidiciplinares de diagramação das tabelas físicas do modelo do banco de dados em UML, com o objetivo de conhecer os passos e restrições existentes para se realizar uma refatoração no modelo do banco de dados. O presente trabalho apresentou testes em linguagem PDDL que no futuro podem ser utilizadas para o desenvolviimento de uma ferramenta de interface amigável para o usuários de banco de dados planejarem e documentarem as etapas de uma grande alteração de banco de dados.

REFERÊNCIAS

[1] S. W. Ambler and P. J. Sadalage, *Refactoring databases: evolutionary database design*. New York: Addison-Wesley Professional, 2006.

[2] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, “Manifesto for Agile Software Development,” 2014. [Online]. Available: http://www.agilemanifesto.org/. [Accessed: 07-Jan-2014].

[3] R. S. Pressman, *Software engineering: a practitioner’s approach*, 7th ed. New York: McGraw-Hill, 2010.

[4] M. B. P. Domingues, J. R. de Almeida Júnior, W. F. Costa, and A. M. Saraiva, “Refactoring database to improve queries performance,” in *In: 21th Italian Symposium on Advanced Database Systems (SEBD 2013)*, 2013.

[5] R. Reiter, “Formalizing Database Evolution in the Situation Calculus,” in *FGCS*, 1992, pp. 600–609.

[6] J. McCarthy, “Programs with Common Sense,” in *Semantic Information Processing*, 1968, pp. 403–418.

[7] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory & Practice (The Morgan Kaufmann Series in Artificial Intelligence)*. San Francisco: Morgan Kaufmann, 2004, p. 635.

[8] H. H. Domingues, F. Kon, and J. E. Ferreira, “Replicação assíncrona em modelagem evolutiva de banco de dados,” in *In: SBBD*, 2009, pp. 121–135.

[9] S. Biundo, R. Aylett, M. Beetz, D. Borrajo, A. Cesta, T. Grant, L. McCluskey, A. Milani, and G. Verfaille, *Technological Roadmap on AI Planning and Scheduling*. PLANET, 2003.

[10] H. H. Domingues, F. Kon, and J. E. Ferreira, “Asynchronous replication for evolutionary database development: a design for the experimental assessment of a novel approach,” in *In: Proceedings of the 2011th Confederated international conference on On the move to meaningful internet systems - Volume Part II*, 2011, pp. 818–825.

[11] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, “{PDDL} -- {T}he {P}lanning {D}omain {D}efinition {L}anguage -- {V}ersion 1.2,” 1998.

[12] D. Xu and X. He, “Generation of test requirements from aspectual use cases,” in *Proceedings of the 3rd workshop on Testing aspect-oriented programs*, 2007, pp. 17–22.

[13] M. B. P. Domingues, J. R. de Almeida Júnior, and J. R. Silva, “Evolution of Databases Using Petri Nets,” in *In: CBA - Congresso Brasileiro de Automática*, 2012.

[14] T. Murata, “Petri nets: Properties, analysis and applications,” *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[15] S. Ambler, *Agile modeling: effective practices for eXtreme programming and the unified process*. New York, NY: John Wiley & Sons, 2002.

[16] B. Nebel, “The FF Planning System: Fast Plan Generation Through Heuristic Search,” *J. Artif. Intell. Researc*, vol. 14, pp. 253–302, 2001.

Márcia Beatriz Pereira Domingues: é graduada em Bacharelado em Ciência da Computação pelasFaculdades Adamantinenses Integradas (FAI), Adamantina, São Paulo, Brasil, em 2002. Obteve o título de mestre em Engenharia Elétrica pela Universidade Estadual Paulista (Unesp), Ilha Solteira, São Paulo, Brasil, em 2008 e de Doutora, em Engenharia da Computação pela Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, em 2014. Atualmente é pos-doutoranda na Universidade de São Paulo (USP) e suas pesquisas se concentram na área de Engenharia de Software, com ênfase em Bancos de Dados e desenvolvimento de sistemas, sistemas de informação e sistemas de apoio a decisão.

Jorge Rady de Almeida Jr.: Possui mestrado em Engenharia Elétrica pela Escola Politécnica da Universidade de São Paulo (1990) e doutorado também em Engenharia Elétrica pela Escola Politécnica da Universidade de São Paulo (1995). É Professor Associado do Departamento de Engenharia de Computação e Sistemas Digitais (PCS) da Escola Politécnica da Universidade de São Paulo desde 2003. Suas áreas de pesquisa são o Desenvolvimento e Avaliação de Sistemas Críticos, com ênfase nos aspectos de tempo real e a área de Engenharia de Software, com ênfase em Bancos de Dados, Data Warehouse e Data Mining, além da segurança em Sistemas de Informação.

Adaptatividade em um sistema de criação de atividades de leitura para o ensino de línguas

J. L. Moreira Filho and Z. M. Zapparoli

***Resumo* — Neste trabalho, apresenta-se a descrição de um estudo sobre o uso de abordagens de análise de corpora da Linguística de Corpus, aliadas a técnicas de Processamento de Língua Natural (PLN), Aprendizagem de Máquina (AM) e Tecnologia Adaptativa (TA), para o desenvolvimento de um protótipo de sistema de processamento de língua natural capaz de gerar, automaticamente, atividades de leitura em língua inglesa a partir de um texto e corpus. A proposta do sistema concentra-se na automatização das análises linguísticas, para a criação automática de atividades de leitura. Assim, são apresentados os objetivos e premissas do estudo, a metodologia, uma pequena descrição do processo de desenvolvimento do sistema e os avanços em sua arquitetura para a implementação da adaptatividade.**

***Palavras-chave* — Linguística de Corpus (*Corpus Linguistics)*, Tecnologia Adaptativa *(Adaptative Technology)*, Processamento de Língua Natural *(Natural Language Processing)*.**

I. INTRODUÇÃO

Este trabalho apresenta os avanços obtidos no trabalho de pesquisa na criação de um sistema de processamento de língua natural para a geração de atividades de leitura em língua inglesa com corpora.

Embora o uso de materiais baseados em corpus seja positivo para o processo de ensino-aprendizagem, a preparação e elaboração desses materiais ainda não é uma realidade fora do contexto acadêmico. Mesmo com as mais variadas ferramentas computacionais para análise de corpora, o processo de preparação de unidades didáticas inteiras e até mesmo de atividades pode ser considerado problemático para a maioria dos professores.

A tarefa, que geralmente leva tempo, é realizada apenas por pesquisadores; muitas vezes, requer a análise prévia de grandes quantidades de dados por programas de computador especializados, como concordâncias, listas de frequência, listas de palavras-chave, anotação de corpus, entre outros tipos. Podemos citar, como exemplo, a pesquisa de [1], que descreveu todo o percurso do uso de dois corpora na elaboração de uma tarefa para ensino de inglês por meio de análises propiciadas por essas ferramentas.

Não é possível esperar que todo professor seja um especialista em Linguística de Corpus para que possa aproveitar os benefícios do uso de corpus e suas ferramentas computacionais de análise em sala de aula.

Devido a esses motivos, professores podem ter dificuldades na preparação de tais materiais e, em consequência, não utilizá-los com certa frequência e/ou fazer uso de materiais tradicionais não significativos para a aprendizagem dos alunos.

Assim, a partir do desenvolvimento, aplicação e análise de um sistema de criação e montagem automática de atividades online de leitura em língua inglesa com corpora, por meio do uso de técnicas de análise de PLN, de práticas de análise de corpus para o ensino de línguas, esta pesquisa tem como um de seus objetivos tentar suprir a necessidade de professores de língua estrangeira que desejam utilizar materiais baseados em corpora em suas aulas, mas que não estão familiarizados com o uso de ferramentas de processamento e exploração de corpora e/ou que não possuem muito tempo para preparar atividades.

A investigação está baseada em um estudo inicial realizado em uma pesquisa de mestrado [2], que teve como produto final um *software desktop* para preparação semiautomática de atividades de leitura em inglês. O produto final do estudo considera como entrada um texto selecionado pelo usuário e, através de etapas, como um assistente eletrônico, levá-lo até a publicação de uma unidade didática.

Nos primeiros protótipos, com base no conceito de *standard exercise*[1] (SCOTT et al., 1984, p. 1) para o ensino de leitura de *English For Especific Purposes* (*ESP*)[2], um conjunto fixo de exercícios é preparado automaticamente, incluindo atividades baseadas em concordâncias, *data-driven learning*[3], predição, léxico-gramática e questões para leitura crítica. Para tanto, o programa faz várias análises automáticas do texto selecionado por meio de fórmulas estatísticas: lista de frequência, palavras-chave, possíveis palavras cognatas, etiquetagem morfológica, possíveis padrões (n-gramas) e densidade lexical do texto.

Embora os resultados obtidos tenham demonstrado a viabilidade e o potencial de, por meio do computador, analisar textos e gerar automaticamente determinados tipos de exercícios para ensino de estratégias de leitura, há ainda a necessidade de muita pesquisa e desenvolvimento de melhorias para a consolidação de uma ferramenta que possa ser usada pelo usuário final. Há situações em que o programa gera exercícios com erros, como incluir uma palavra de conteúdo na lista de palavras gramaticais. Nesses casos, há a possibilidade de intervenção do usuário, mas, se a quantidade de erros for numerosa de forma a exigir constantemente esse tipo de intervenção do usuário/professor, o sistema pode perder sua utilidade, impossibilitando seu uso pedagógico.

Outro ponto essencial é o aumento da variedade de exercícios disponíveis e a quebra da limitação do usuário a um modelo fixo. O conceito de atividade padrão está relacionado à necessidade de treinar a compreensão leitora do aprendiz a partir de um conjunto fixo de questões que

[1] atividade padrão, tradução nossa.
[2] inglês para fins específicos.
[3] ensino movido a dados, tradução nossa.

poderiam ser utilizadas a quase qualquer texto. A invariabilidade do modelo padrão faz com que o programa tenha sua funcionalidade limitada e estática. A adição de recursos adicionais significaria toda a sua reprogramação.

Em uma nova proposta de sistema, a geração de exercícios em relação a outros itens comuns no ensino de estratégias de leitura e léxico-gramática, adequados a uma gama maior de textos é pretendida. Em vez de um modelo fixo e estático, procura-se desenvolver um ambiente propício com um conjunto de instruções, parâmetros e regras que possibilitem uma funcionalidade flexível e dinâmica, em que a criação e montagem dos exercícios sejam feitas autonomamente de acordo com o texto de entrada, sem a necessidade de reprogramação.

Desse modo, buscando estender os objetivos da pesquisa de mestrado [2], empregam-se, em diálogo, pressupostos teóricos e metodológicos, além de técnicas das áreas de Linguística de Corpus, Processamento de Língua Natural, Aprendizagem de Máquina e Adaptatividade em todos os níveis aplicáveis de um sistema de processamento de língua natural, que, a partir do fornecimento de um texto e/ou algum tipo de entrada preestabelecida, gere atividades didáticas de leitura em língua inglesa.

A princípio, considera-se relevante a exploração do trabalho em quatro níveis/camadas do sistema: i. Opções do usuário; ii. Análise linguística da entrada; iii. Análise pedagógica; iv. Montagem de exercícios. A atividade de ensino a ser gerada deverá levar em consideração as informações de todos os níveis. O produto final estará condicionado às opções do usuário (seleção de itens de ensino, tipos de exercício e itens léxico-gramaticais incluídos), análise linguística de textos (informações de frequência, palavras-chave, anotação morfossintática, entre outras), análise pedagógica da entrada (a partir da análise linguística, quais tipos de exercícios são possíveis e adequados), montagem de exercícios (como extração e organização de itens linguísticos).

Embora ainda haja muito trabalho a ser realizado, a investigação é importante para o estabelecimento de um diálogo concreto entre Linguística de Corpus, Processamento de Língua Natural, Aprendizagem de Máquina e Adaptatividade, especificamente no que diz respeito ao desenvolvimento de aplicações dinâmicas para análise de corpora e ensino de línguas.

II. OS OBJETIVOS E PREMISSAS DO ESTUDO

O objetivo geral desta pesquisa é conceber um sistema de processamento de língua natural capaz de analisar corpora e textos para gerar, automaticamente, informações pedagógicas sobre o texto, atividades de leitura e ensino de padrões (léxico gramática) em língua inglesa.

Os objetivos específicos para consecução do objetivo geral envolvem:

- estudar as abordagens e ferramentas de análise de corpora para a criação de materiais de ensino na área de Linguística de Corpus;
- estudar as abordagens, técnicas, algoritmos e recursos das áreas da computação como Processamento de Língua Natural, Aprendizado de Máquina e Adaptatividade e suas tecnologias;
- estudar a implementação de algoritmos por meio de programação em linguagem Python;
- coletar e analisar a padronização de corpora em gêneros específicos a fim de serem utilizados no desenvolvimento e avaliação de um sistema de análise de corpora e textos;
- construir módulos de análise de corpora e textos para obtenção de recursos e dados linguísticos para avaliação pedagógica de um texto e criação de exercícios;
- avaliar por meio de experimentos práticos os módulos de análises construídos, além de sua eficácia na criação de atividades;
- contrastar os benefícios de uso do sistema em relação às ferramentas de programas de análise de corpora, especificamente os concordanciadores;
- formalizar a arquitetura do sistema;
- apresentar uma análise de problemas com soluções adotadas;
- apresentar possíveis adaptações do sistema para uso de textos em língua portuguesa.

As principais premissas da pesquisa são as de que:

- as ferramentas de análise de corpora atuais, por serem desenhadas para uso geral, em diferentes propósitos, podem demandar grande esforço de adaptação de suas funcionalidades para a pesquisa e construção de materiais baseados em dados de análise de corpora;
- a preparação de materiais no ensino de línguas pode ser enriquecida com o auxílio de recursos computacionais que contribuam para a otimização dos estudos linguísticos em larga escala, tanto em relação ao tempo e esforço, quanto em qualidade de seus resultados;
- a automatização de análises e exploração de corpora para fins específicos, como a extração de dados para elaboração de materiais didáticos e atividades para o ensino de línguas, pode ser uma realidade, a partir do estudo das ferramentas computacionais disponíveis e o diálogo entre outras áreas de estudo;
- os dados de análises básicas, comuns no processamento de língua natural, podem ser extrapolados na criação de atividades de leitura e ensino de léxico-gramática;
- o uso de uma abordagem específica de análise automática de corpus pode trazer benefícios significativos para a tarefa de elaboração de materiais de ensino baseados em corpora;

um maior diálogo da Linguística de Corpus com outras áreas que também fazem uso de corpora, as quais também fazem uso de corpora, especificamente Processamento de Língua Natural, Aprendizagem de Máquina e

Adaptatividade, pode otimizar os procedimentos de pesquisa e análise de corpora;

a formalização da necessidade de análise automática de corpora pode ser objeto interessante de pesquisa para o linguista de corpus em novos caminhos de investigação.

III. METODOLOGIA

A abordagem de desenvolvimento do sistema de análise de corpora e textos em língua inglesa para a criação automática de atividades de leitura e ensino de padrões baseia-se nas funcionalidades das ferramentas existentes de análise e exploração de corpora, como ponto de partida.

A metodologia do processo de desenvolvimento consiste em:

- analisar os elementos constituintes das principais ferramentas de análise e exploração de corpora e as metodologias de uso de tais ferramentas no âmbito do ensino de línguas relacionado à Linguística de Corpus;
- coletar e compilar os corpora de estudo, desenvolvimento e teste para todo o processo de desenvolvimento do sistema proposto;
- estudar, construir e adaptar métodos, técnicas e algoritmos existentes para análise de corpora e textos, necessários para as análises pretendidas;
- anotar dados de corpora para o treinamento e teste de algoritmos;
- elaborar módulos de análise de apoio, por exemplo, módulos de itemização, etiquetagem, contagem de palavras, extração de palavras-chave, extração de n-gramas entre outros, para a análise de corpora e textos;
- aplicar os módulos de análise nos corpora de estudo, desenvolvimento e teste, a fim de obter a descrição dos corpora e possibilidades de dados e recursos a serem utilizados para a criação de atividades;
- avaliar e otimizar os módulos de análise a partir de sua aplicação nos corpora de desenvolvimento e teste;
- explorar, criar e adaptar os módulos de análise na construção de algoritmos de criação de atividades de ensino.

IV. DESENVOLVIMENTO

Para o desenvolvimento da pesquisa, utilizamos a linguagem de programação Python e a biblioteca de processamento de língua natural NLTK para a criação dos módulos de análise. A implementação da interface foi realizada as linguagens PHP, HTML, CSS e javascript.

Para o desenvolvimento da pesquisa, utilizamos dois principais tipos de corpora:corpus de referência e corpus de treinamento/estudo. Os corpora de referência são utilizados para comparação entre os corpora de treinamento e estudo, além de servirem para a extração de padrões léxico-gramaticais. Os corpora de treinamento/estudo são utilizados para o desenvolvimento das funções de análise do sistema e para a realização de testes.

Os principais corpora do estudo são:

- Corpus de língua geral do inglês;
- Corpus de anúncios de emprego em inglês;
- Corpus de artigos em inglês da revista *Scientific American*.

Além dos corpora descritos, foram utilizados os recursos e léxicos da biblioteca *NLTK* (*Natural Language Tool Kit*), tal como o corpus *Penn Treebank*.

As principais etapas de desenvolvimento do sistema foram a criação de:

- módulos de análise linguística de texto em linguagem Python;
- um módulo que compila todas as análises e cria um arquivo do texto analisado em *XML*;
- um módulo de leitura e extração de informações de um texto analisado em *XML*;
- um módulo de leitura e extração de informações de um corpus formado por textos já analisados em *XML*;
- um módulo interpretador que recebe como entrada um código *XML (templates)* e retorna, a partir de funções parametrizadas, uma atividade ou exercício;
- Criação da interface com o usuário.

Embora os módulos sejam projetados para análise de textos em língua inglesa, sua criação foi desenhada para permitir a análise de textos de outras línguas.

Para cada módulo, há uma opção de seleção de língua. O uso do módulo para uma outra língua pode ser conseguido, acrescentando-se um novo conjunto de regras, léxico ou outro tipo de dado, conforme a especificidade da análise.

Por exemplo, no módulo de extração de palavras-chave, há o diretório '*data/en*' e, dentro, um arquivo de referência '*reflist.txt*'. Para extração de palavras-chave em português, bastaria adicionar um novo diretório e arquivo de referência ('*data/br/reflist.txt*') e, no código, na instanciação da classe de análise, passar o argumento *'br*' como língua de análise.

Os principais módulos de análise linguística compreendem: segmentação, itemização, etiquetagem morfossintática, contagem de palavras, extração de palavras-chave, extração de n-gramas, identificação de palavras-cognatas, identificação de afixos, identificação de grupos nominais simples (*noun phrases*), identificação de referência pronominal, reconhecimento de entidades nomeadas e extração de relações. Tendo em vista a arquitetura projetada, é possível estender as a qualquer momento, com a adição de novos módulos.

As análises podem propiciar uma série de informações para a análise pedagógica do texto e matéria prima para atividades de leitura e ensino de lexicogramática, que podem, a princípio, serem geradas em formato de questão aberta ou formato de múltipla escolha, havendo a possibilidade de já indicar a resposta correta. Como exemplo ilustrativo, a análise automática pode identificar no corpus de anúncios de empregos as entidades '*COMPANY*' e '*POSITION*' (marcadas com códigos) na seguinte sentença:

*<**ne type='COMPANY'**>WCMC<**/ne**> are now seeking a permanent <**ne type='POSITION'**>Head of Programme<**/ne**> to work with us.*

A partir da análise, é possível criar questões sobre qual é o nome da empresa ou qual vaga. A identificação de tais entidades podem também contribuir para a análise dos movimentos ou passos do gênero. Pode-se criar atividades de localização de informações, a partir de perguntas como: em qual parte do texto podemos encontrar a vaga disponibilizada.

Outro exemplo é a análise quantitativa realizada das palavras do texto. Pode-se criar uma atividade em que deve-se identificar as palavras mais frequentes do texto e/ou quais são suas palavras-chave. A intenção é aproveitar ao máximo as análises para os fins pedagógicos de leitura e ensino de léxicogramática.

A programação dos módulos para as análises como as descritas foi realizada, em grande parte, a partir de heurísticas, métodos e algoritmos utilizados nas áreas de Linguística de Corpus, Processamento de Língua Natural e Aprendizagem de Máquina. Tendo em vista o número de análises e a complexidade do processamento de língua natural, há limitações reconhecidas que poderão minimizadas conforme a evolução do sistema, seja por meio de acréscimos de novas regras ou aumentos dos léxicos e dados de treinamento.

V. TECNOLOGIA ADAPTATIVA

Conforme [4], a Tecnologia Adaptativa está relacionada a técnicas, métodos e disciplinas que estudam as aplicações da adaptatividade, que pode ser entendida como uma propriedade que um determinado modelo tem de modificar espontaneamente seu próprio comportamento em resposta direta a uma entrada, sem auxilio externo.

Um sistema adaptativo é aquele que possui a propriedade de se automodificar a partir de determinada entrada, sem a necessidade de um agente externo.

Dentro da Tecnologia Adaptativa, há a noção de dispositivo, uma abstração formal. O dispositivo pode ser adaptativo ou não adaptativo. O dispositivo não adaptativo pode ser formado por um conjunto finito de regras estáticas que, em linguagem de programação, pode ser representado na forma de cláusulas IF-THEN. A operação do dispositivo se dá pela aplicação das regras, tendo como retorno determinados estados. Quando o dispositivo não aplica nenhuma regra, a operação é terminada, gerando um erro. As ações de dispositivos adaptativos podem ser chamadas quando ocorre algum erro (quando nenhuma regra é aplicável), ou quando a operação do dispositivo não adaptativo está em um determinado estado. Basicamente, os dispositivos adaptativos são formados por três ações adaptativas elementares [5]: i. Consulta de regras/estados; ii. Exclusão de regras; iii. Inclusão de regras. Seu uso está ligado a situações complexas em que há a necessidade de tomadas de decisões não triviais, por exemplo, na área de estudos da linguagem, resolução de ambiguidades em programas de anotação (morfológica, sintática, etc.).

Tendo em vista a complexidade do estudo pretendido, uma das áreas de interesse para a pesquisa, em relação à Tecnologia Adaptativa, é o processamento de linguagens naturais. A aplicação da Tecnologia Adaptativa ao processamento de línguas naturais é um campo de extrema importância.

VI. ARQUITETURA DO SISTEMA

O sistema recebe um texto como entrada e tem como objetivo realizar análises linguísticas automáticas que permitem sua exploração para extração de informações relevantes para a análise pedagógica do usuário e a criação automática de atividades.

Descrevemos aqui a arquitetura planejada para a implementação da adaptatividade no sistema, tendo em vista que sua interface está ainda em construção. A figura a seguir ilustra a arquitetura do sistema.

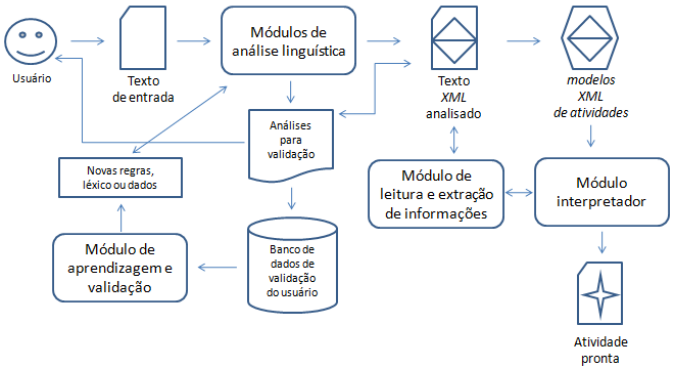


Figura 1. Arquitetura do sistema

A arquitetura apresentada pode ser descrita a partir dos seguintes passos:

1. O usuário insere o texto de entrada, o qual será o foco para a criação das atividades de leitura;
2. Os módulos de análise linguística processam o texto e geram um conjunto de itens de análise automática para validação do usuário;
3. O usuário valida e/ou corrigi os itens de análise automática;
4. Os itens validados e/ou corrigidos são armazenados em um banco de dados;
5. Um módulo de aprendizagem e validação analisa o banco de dados de validação do usuário e gera novas regras, léxicos e/ou dados de aprendizagem;
6. Os módulos de análise linguística são atualizados com os dados gerados pelo módulo de aprendizagem e validação;
7. Os módulos de análise linguística geram um arquivo *XML* com a análise do texto de entrada;
8. Modelos de atividades escritos em XML são instanciados para o texto analisado (selecionados pelo usuário ou requeridos a partir das características do texto analisado);

9. Um interpretador traduz cada instrução dos modelos de atividade e, por meio de um módulo de leitura e extração de informações, os modelos são preenchidos para gerar as atividades;
10. O sistema retorna a atividade pronta.

Embora não especificado em detalhes no esquema da figura 1., os modelos de atividades escritos em *XML* poderiam ser gerados no momento de execução conforme as características do texto analisado. Tal funcionalidade estaria relacionada a uma análise pedagógica automática, em que seriam determinados quais tipos de exercícios e itens seriam mais adequados dada a análise do texto de entrada.

Entende-se que um texto pode ser explorado de diversas formas. Por exemplo, um professor pode preferir explorar as características gramaticais do texto em relação a sua organização discursiva. Neste caso, modelos de atividades gramaticais seriam instanciados preferencialmente em relação a outros, conforme as características do texto de entrada.

Uma forma de implementação seria a criação de perfis, conforme as possibilidades de exploração pedagógica de um texto. A figura a seguir ilustra o processo de seleção de modelos de atividades:

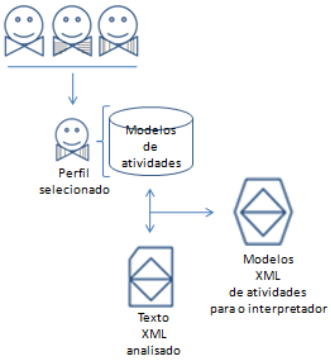


Figura 2. Arquitetura do sistema

Para cada perfil, um conjunto de modelos de atividades estaria disponível. A seleção de um determinado perfil implicaria na instanciação de determinados modelos de atividades que seriam adicionados de acordo com regras definidas, cujos argumentos seriam satisfeitos com base nas análises automáticas do texto de entrada. Após as adições dos modelos de atividades, todo o conjunto é submetido ao interpretador para gerar as atividades.

Como já citado, o sistema está em constante modificação. Os módulos de análise em sua maioria estão consolidados. Os passos seguintes incluem a construção dos módulos necessários para a validação e correção das análise automáticas, realizadas pelo usuário no momento de entrada do texto, e o gerenciamento da aprendizagem a partir do banco de dados das ações do usuário. Considera-se que tais modificações contribuam para conferir uma característica adaptativa ao sistema, o que promoverá sua constante evolução e aumento de suas funcionalidades para a criação de atividades.

VII. Considerações finais

O trabalho buscou mostrar alguns dos avanços feitos em uma pesquisa no desenvolvimento de um sistema de processamento de língua natural para a criação de atividades de leitura em língua inglesa com corpora, com vistas à utilização da adaptatividade na melhoria dos resultados obtidos em análises linguísticas para a ampliação das funcionalidades do sistema proposto.

Os resultados da investigação contribuem para fomentar o interesse em relação ao desenvolvimento de ferramentas computacionais para exploração de corpora na automatização da extração de dados linguísticos para a elaboração de materiais de ensino e currículo.

O percurso do desenvolvimento do sistema é considerado parte integrante da pesquisa, uma vez que seu cenário pode ser considerado pouco usual, dada a natureza interdisciplinar do estudo e suas exigências, tendo o pesquisador linguista de enveredar pela análise computacional por meio de programação.

O esforço pode propiciar um contato maior com as diferentes áreas e trazer benefícios para a construção do diálogo pretendido, não só nesta pesquisa, mas também nas pesquisas de linguagem via computador em geral, em que há separação entre equipes: de um lado, aqueles que fazem a programação para a realização de tarefas automaticamente; de outro, aqueles que apenas fazem uso dos instrumentos.

Referências

[1] CONDI, R. Dois corpora, uma tarefa. O percurso de coleta, análise e utilização de corpora eletrônicos na elaboração de uma tarefa para ensino de inglês como Língua Estrangeira. 2005. Dissertação (Mestrado em Linguística Aplicada e Estudos da Linguagem), LAEL, PUC-SP, São Paulo, 2005.

[2] MOREIRA FILHO, P. Desenvolvimento de um software para preparação semiautomática de atividades de leitura em inglês. 2007. Dissertação (Mestrado em Linguística Aplicada e Estudos da Linguagem), LAEL, PUC-SP, São Paulo, 2007.

[3] SCOTT, M., CARIONI, L., ZANATTA, M., BAYER, E. &QUINTANILHA, T. Using a Standard Exercise in Teaching Reading Comprehension.English Language Teaching Journal, v. 38, n. 2, pp. 114-20.

[4] DIZERÓ, W. Formalismos Adaptativos Aplicados na Modelagem de Softwares Educacionais. 2010. Tese (Doutorado em Engenharia Elétrica e Sistemas Digitais), EPUSP, São Paulo, 2010.

[5] MENEZES, C. E. D. ; JOSÉ NETO, J. . Um método híbrido para a construção de etiquetadores morfológicos, aplicado à língua portuguesa, baseado em autômatos adaptativos.. In: Conferencia Iberoamericana en Sistemas, Cibernética e Informática - CISCI 2002, 2002, Orlando. Anais da Conferencia Iberoamericana en Sistemas, Cibernética e Informática - CISCI 2002., 2002.

José Lopes Moreira Filho é Doutorando em Semiótica e Linguística Geral (USP). Possui Mestrado em Linguística Aplicada e Estudos da Linguagem pela Pontifícia Universidade Católica de São Paulo (PUCSP). Possui graduação em Letras – Português e Inglês (Bacharelado Tradução) pela Universidade de Mogi das Cruzes (UMC). Atualmente, é Professor Coordenador do Núcleo Pedagógico da Diretoria Regional de Ensino Leste 3 da SEE-SP, mantendo interesses na área de Linguística, Linguística Aplicada, Linguística Informática, Linguística de *Corpus*, Processamento de Linguagem Natural, atuando principalmente no desenvolvimento de ferramentas computacionais para exploração de *corpora*, ensino de línguas, entre outras aplicações que envolvem linguagem e tecnologia.

Zilda Maria Zapparoli é professora associada aposentada junto ao Departamento de Linguística da Faculdade de Filosofia, Letras e Ciências Humanas da Universidade de São Paulo, instituição em que obteve os títulos de Mestre, Doutor e Livre-Docente, e onde continua desenvolvendo atividades de ensino, pesquisa e orientação no Curso de Pós-Graduação em Linguística, área de Semiótica e Linguística Geral, linha de pesquisa Informática no Tratamento de *Corpora* e na Prática da Tradução. Desde 1972, atua em Linguística Informática, com tese de doutorado, tese de livre-docência, pós-doutorado na Université de Toulouse II e trabalhos publicados na área. É líder do Grupo Interdisciplinar de Pesquisas em Linguística Informática, certificado pela USP e cadastrado no Diretório de Grupos de Pesquisa no Brasil do CNPq, em 2002. Integrou comissões e colegiados na USP, destacando-se os trabalhos relativos ao processo de informatização da FFLCH-USP, enquanto membro da Comissão Central de Informática da USP e presidente da Comissão de Informática da FFLCH-USP por cerca de treze anos.

Operação Sustentável de Sistemas de Ar Condicionado Usando Tecnologia Adaptativa

R. M. Marè, M. F. Barros, M. A. Dota, C. E. Cugnasca e B. C. C. Leite

***Abstract*— It is possible to define activity scenarios for air conditioning systems associated with indoor environmental quality. It is necessary, for each scenario, set points adjustment and parameters calibration for a sustainable performance with energy efficiency. These adjustments are necessary during the implementation of the system and in every significant change in environment use. They require that tests be performed by a specialist. This work objective is to propose a technique for automatic classification of thermal comfort and even indoor air quality to allow the programming of set points and the re-calibration of system parameters, whenever there are significant changes in environment use. Adaptive technologies such as Adaptive Decision Tree associated with Induction of Decision Trees tools are used to solve the problem. It is expected that the proposed technique is able to determine, without the specialist need, the class of the indoor environmental conditions once they change.**

***Keywords*— Air Conditioning System, Decision Tree, Thermal Comfort, Indoor Air Quality, Adaptive Technology.**

I. INTRODUÇÃO

A operação sustentável das edificações é um tema de relevância crescente, especialmente nos países desenvolvidos [1]. Um dos grandes desafios é compatibilizar consumo de energia e qualidade do ambiente interior (conforto térmico, qualidade do ar, conforto lumínico), que impacta diretamente no bem estar, saúde e produtividade dos ocupantes [2] [3]. No Brasil, estima-se que quase metade de toda energia produzida no país seja consumida pela operação e manutenção de sistemas prediais de iluminação, climatização e aquecimento de água [4]. No mercado europeu, o consumo de energia em edifícios devido aos sistemas de ar condicionado vem crescendo há décadas [5] e, nos Estados Unidos, os edifícios consomem mais de 70% de toda energia gerada, devido aos sistemas de resfriamento, aquecimento e iluminação [6].

Os edifícios necessitam de sistemas que os auxiliem em sua operação e sejam capazes de acompanhar as inúmeras mudanças de uso e ocupação ao longo de sua vida útil [7]. A base para que se estabeleçam estratégias eficientes na operação dos sistemas é a obtenção contínua de informação sobre o comportamento da edificação, o que pode ser equacionado pela implantação de redes de sensores que monitorem parâmetros de interesse. Em se tratando de sistemas de ar condicionado, estratégias adaptativas ao contexto baseadas em informações relativas à ocupação, como teor de CO_2 e carga térmica nos ambientes interiores, condições climáticas externas e consumo de energia associado, podem ser de grande valia para a sua operação sustentável.

Apresenta-se neste trabalho uma proposta de uso de Árvores de Decisão Adaptativas (ADA) para a criação de um classificador das condições do ambiente interior relativas ao conforto térmico e à qualidade do ar, que auxiliará na definição de estratégias de operação sustentáveis de sistemas de ar condicionado central, minimizando-se operações manuais e intervenção humana.

Na seção II discute-se a qualidade de ambientes interiores bem como, a importância da eficiência energética em edifícios. Na seção III apresentam-se as ADA, indicando-se sua aplicação na concepção de um classificador para a qualidade do ambiente interior. Na seção IV apresentam-se os materiais e método empregados, discutindo-se um caso de aplicação das ADA no controle da qualidade de ambientes interiores, enquanto que na seção V, os principais resultados dessa aplicação são discutidos. Finalizando, na seção VI apresentam-se as conclusões, seguidas dos agradecimentos e relação de referências.

II. QUALIDADE DO AMBIENTE INTERIOR E EFICIÊNCIA ENERGÉTICA

A baixa qualidade do ambiente interior de edificações pode causar grande impacto em seus usuários, a exemplo da Síndrome do Edifício Doente e das Doenças Relacionadas à Edificação, que comprometem sobremaneira o bem estar e a produtividade de seus usuários, tanto por problemas de saúde temporários como por infecções crônicas [8] [9] [2]. Nos Estados Unidos estima-se que cerca de 10 milhões de trabalhadores são afetados por esses problemas, resultando em prejuízos da ordem de bilhões de dólares em absenteísmo, queda na produtividade, ações trabalhistas e propaganda negativa [10].

Os benefícios econômicos obtidos com o incremento da qualidade do ambiente interior decorrem, principalmente, da melhora do desempenho dos ocupantes (tanto em qualidade como em velocidade das ações), redução do absenteísmo e redução nos custos com tratamentos de saúde [11].

Em relação à eficiência energética, apenas os sistemas de climatização são responsáveis por 50% dos gastos com energia em edifícios [12]. No entanto, estes sistemas são as

R. M. Marè, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, renata.mare@usp.br

M. A. Dota, Universidade Federal do Mato Grosso (UFMT), Rondonópolis, Mato Grosso, Brasil, maraadota@gmail.com

M. F. Barros, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, barros.mfb@gmail.com

C. E. Cugnasca, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, carlos.cugnasca@poli.usp.br

B. C. C. Leite, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, brendaccleite@gmail.com

melhores ferramentas para proporcionar condições de conforto térmico, assim como renovação do ar com tratamento por meio dos sistemas filtrantes, especialmente em locais onde a ventilação natural é impraticável devido a problemas de segurança e poluição sonora, dentre outros fatores. Requerem, portanto, atenção especial desde a fase de projeto até a sua operação [13]. Edifícios energeticamente eficientes têm seu m^2 valorizado em 10%, comparativamente a edifícios convencionais e, mesmo entre edifícios verdes, aqueles que possuem maior eficiência térmica e energética ganham em valores de aluguéis e do próprio ativo [1].

III. ÁRVORES DE DECISÃO ADAPTATIVA

Sistemas de ar condicionado fazem uso de um ou mais *loops* de controle, cujas funções são previamente definidas. Alterações significativas no uso do ambiente interior ou mesmo na influência do ambiente exterior exigem também alterações nas funções de controle que em geral, são realizadas por meio de ajustes manuais em um conjunto de *setpoints* que definem a função de controle.

Este trabalho propõe o uso de tecnologia adaptativa para definir as novas funções de controle ou especificamente um novo conjunto de *setpoints* que, neste trabalho, serão qualificados como uma nova classe.

A técnica proposta neste estudo para a criação de um classificador da qualidade do ambiente interior é a Indução por Árvore de Decisão (IAD) [14]. A IAD tem um grande potencial para diversas aplicações, tais como: diagnóstico de falhas, detecção de eventos, e explicação e avaliação de fenômenos ambientais.

A Árvore de Decisão (AD) é um tipo de classificador treinado por seleção iterativa de características individuais mais relevantes em cada nó na árvore [15]. Um espaço de entrada X é dividido em subconjuntos repetidamente descendentes, começando com o próprio X. Existem vários métodos heurísticos para a construção de classificadores de AD. Eles são geralmente construídos de cima para baixo, começando no nó raiz e, sucessivamente, o conjunto vai sendo particionado de acordo com suas características. A construção envolve três etapas principais:

1. Selecionar uma regra de divisão para cada nó interno, ou seja, a determinação da funcionalidade, juntamente com um limiar, será usada para dividir o conjunto de dados em cada nó;
2. Determinar quais nós são nós terminais. Isso significa que para cada nó deve-se decidir pela continuidade ou não do processo de divisão;
3. Atribuir rótulos de classe para nós terminais, minimizando a taxa de erro estimada.

As AD possuem duas fases de trabalho: a aprendizagem e a execução. Árvores de Decisão Adaptativas (ADA) são AD que podem executar as duas tarefas na mesma fase [16]. Pode-se representar a ADA como uma Tabela de Decisão Adaptativa (TDA) dividida em duas camadas: camada de decisão convencional (formada pelas funções convencionais) e camada adaptativa (formada pelas funções adaptativas).

A fim de obter-se a classificação, utilizou-se a ferramenta Waikato Environment for Knowledge Analysis (WEKA) para a execução do aprendizado [17]. Como a ferramenta usada não possui recursos de técnicas adaptativas, optou-se por considerar todo o IAD como uma única função adaptativa como na TDA representada pela Figura 1. Desta forma, pode-se incluir o aprendizado na mesma fase que a execução.

As variáveis que disparam o aprendizado são as mesmas do processo normal, porém foram incluídas duas novas variáveis:

- Aprendizado: variável booleana que ativa manualmente o aprendizado e é disparada sempre que o usuário alterar significativamente o uso do ambiente, inclusive acrescentando ou retirando carga térmica expressiva do ambiente.
- Manutenção: variável booleana que indica que existem elementos do sistema inativos. Esta variável dispara automaticamente um novo aprendizado.

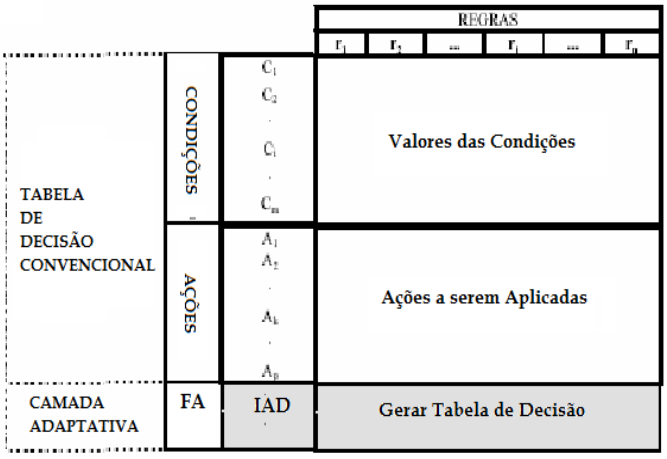


Figura 1. Elementos que compõem a TDA proposta. Fonte: Adaptação de [18].

A Figura 2 apresenta a representação gráfica da AD considerada.

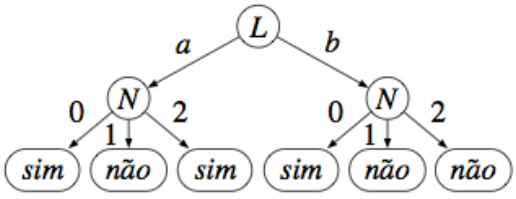


Figura 2. Exemplo de árvore de decisão. Fonte: [16].

A árvore apresentada representa a função

$$f : N \text{ X } L \rightarrow C$$

onde:

$$N = \{0,1,2\}$$
$$L = \{a,b\}\ e$$
$$C = \{sim,não\}$$

com os valores de C definidos na Tabela I.

Os vértices internos da árvore representam testes efetuados sobre variáveis que conduzem a cada possível valor a uma nova aresta, representando o resultado obtido. Parte-se da raiz da árvore submetendo-se exemplos de conjuntos de atributos, chegando-se a resultados da classificação representados nas folhas da árvore.

Tabela I. Definição da função $f: N\,X\,L \rightarrow C$.

N	L	C
0	a	SIM
0	b	SIM
1	a	NÃO
1	b	NÃO
2	a	SIM
2	b	NÃO

Em relação ao classificador para a qualidade do ambiente interior, neste trabalho são consideradas as seguintes características: temperatura do ar e teor de CO_2. Essas características tornam-se atributos considerados para a construção da árvore.

As ADA possuem ampla gama de aplicações bem como, a ferramenta WEKA, que dispõe de uma biblioteca com inúmeras implementações de algoritmos de aprendizagem. Combinados, estes recursos revelaram o seu potencial em estudos anteriores [19].

IV. MATERIAIS E MÉTODO

Como estudo de caso, foram utilizadas duas salas de aula situadas no edifício de Engenharia de Construção Civil da Escola Politécnica da Universidade de São Paulo (Figura 3), que possuem uso e geometria similares. Cada uma é atendida por um sistema central de ar condicionado distinto: a Sala 17 possui um sistema com distribuição de ar pelo piso, enquanto a Sala 23 possui um sistema misto de insuflamento de ar pelo teto, com teto radiante frio.

Figura 3. Sala 17 (à esquerda) e Sala 23 (à direita). Fonte: autores.

Ambas vêm sendo monitoradas continuamente desde 2009, por uma rede de sensores e um sistema de monitoramento baseado em Internet. Monitoram-se - tanto internamente quanto externamente às salas - temperatura do ar, umidade relativa do ar e teor de CO_2. Este sistema permite a coleta contínua de dados dos dois ambientes, enviando-os a um servidor *web* remoto onde são armazenados. Usuários autorizados podem acessá-los por meio de um *web site* dedicado, utilizando-se de um navegador padrão para a Internet.

Visto que as leituras nos sensores são efetuadas a cada 30 segundos, uma considerável massa de dados está disponível *online*. Com base nessas informações, podem-se realizar diversas análises, por exemplo: verificação da adequação das condições ambientais frente aos parâmetros vigentes em normas; comparação dos sistemas de ar condicionado a partir da eficiência na promoção de temperatura, umidade relativa do ar e teor de CO_2 conformes; identificação de padrões de variação das grandezas monitoradas em cada ambiente, ao longo do tempo, em função da ocupação e das estratégias de operação adotadas para cada sistema; análise cruzada de informações de qualidade do ambiente interior e consumo de energia para cada sistema de ar condicionado.

Por meio do sistema, também são monitorados os consumos de energia dos dois *fan coils* e do *chiller*.

O sistema de monitoramento possui um modelo em camadas, assim como, um *middleware* capaz de receber dados remotamente por meio de várias tecnologias como *web services* [20; 21], conexões por soquetes, ftp, sftp ou WebDav [22]. Ele possui ainda uma camada opcional de segurança de dados, visando permitir diferentes níveis de criptografia e até assinatura digital de pacotes de dados, ou seja, a não-repudiação de origem de dados poderá ser garantida.

Na Figura 4 apresenta-se a arquitetura simplificada do software.

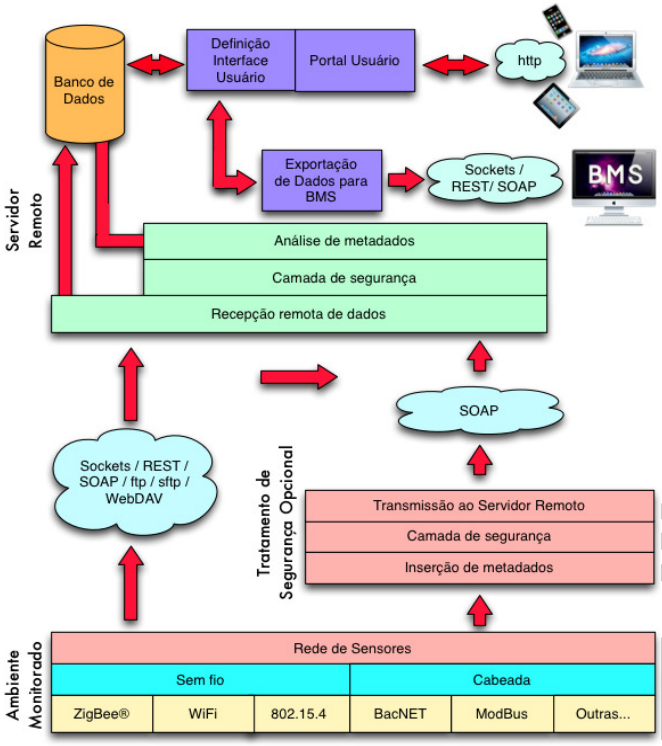


Figura 4. Arquitetura simplificada do software de monitoramento. Fonte: autores (onde BMS = *Building Management System*).

Já o sistema de controle proposto será baseado em um Controlador Lógico Programável, que receberá informações tanto dos sensores quanto do resultado da classificação do ambiente interior, agindo sobre os atuadores, acionando ventiladores, compressores, *dampers* e válvulas, dentre outros componentes do sistema de ar condicionado. Sempre que os resultados não forem compatíveis com o conforto esperado pelos ocupantes, uma nova ADA será desenvolvida, gerando novas classificações. Em função destas classes, o controle acionará o conjunto de pontos de operação (*setpoints*) mais adequados ao novo cenário, ou mesmo a estados emergenciais ou críticos (valores não-conformes das variáveis). Atualmente, um sistema supervisório proprietário fornece informações do sistema de ar condicionado, permitindo a sua monitoração e intervenções manuais (Figura 5).

Neste trabalho será dado enfoque ao sistema de ar condicionado da Sala 17 (insuflamento de ar pelo piso). A estratégia de controle utilizada desde a sua implantação foi elaborada a partir de cinco *loops* descritos a seguir [13; 23]:

Loop 1: A frequência de rotação do ventilador do *fan coil* varia para manter constante o diferencial de pressão entre o *plenum* e o ambiente (ΔP). O ΔP é obtido a partir de leituras de três transdutores de pressão (no *plenum* e ambiente) instalados no local.

Loop 2: A frequência de rotação do ventilador do retorno varia para manter constante o diferencial de temperatura (ΔT) entre o ar de retorno e o ar de insuflamento (mistura). O ΔT é obtido com base em leituras feitas por sensores de temperatura instalados no duto de retorno e no ponto de descarga do ar de insuflamento.

Loop 3: A válvula de água gelada da serpentina do *fan coil* modula para que a temperatura do ar frio seja mantida em torno de um determinado *setpoint*. A temperatura do ar frio é medida por um sensor instalado na saída do *fan coil*.

Loop 4: O *damper* de *bypass* de retorno e o *damper* de retorno para o *fan coil* modulam inversamente para que se atinja o *setpoint* da temperatura de insuflamento (mistura de ar frio e um percentual de ar de retorno). Um sensor de temperatura instalado no ponto de descarga do ar de insuflamento no *plenum* e outro no duto de retorno do ar ao *fan coil* determinam o percentual de abertura dos dois *dampers*, de modo a que se atinja uma mistura de ar a uma temperatura mais próxima possível do *setpoint*.

Loop 5: Os *dampers* de expurgo e de ar externo modulam diretamente e de acordo com os valores de entalpia do ar de retorno e do ar externo calculados com base em sensores de temperatura e umidade instalados no duto de tomada de ar externo e no de retorno.

Observa-se pela descrição dos *loops,* a forte inter-relação entre variáveis, deixando claros tanto a complexidade da estratégia em questão, quanto o grande potencial de utilização de tecnologias adaptativas.

Diante deste contexto, definiram-se como condições de contorno para este trabalho a análise do conforto térmico no ambiente e, dentre as inúmeras variáveis associadas a esta condição, adotou-se a temperatura do ar.

Os padrões de conforto térmico encontram-se definidos em normas [24; 25], que apresentam a zona de conforto definida por uma combinação ótima de parâmetros físicos (a exemplo de temperatura do ar, velocidade do ar e umidade do ar) e fatores pessoais (tipo de vestimenta e nível de atividade), com os quais, ao menos 80% dos ocupantes se sintam satisfeitos [26].

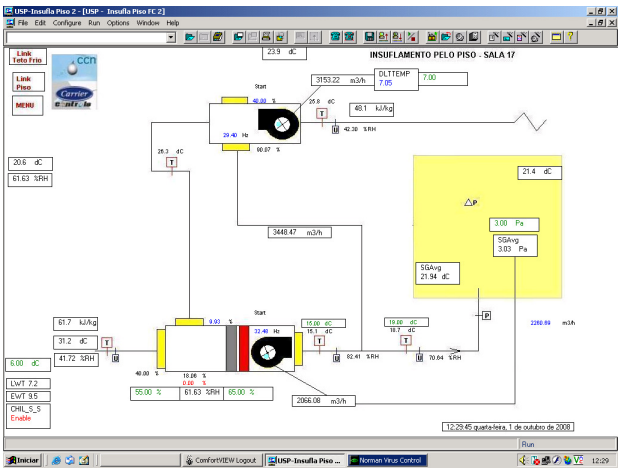


Figura 5. Tela de interface do Sistema Supervisório ComfortView. Fonte: [13].

Para a atividade predominante no ambiente em questão (sala de aula), a temperatura deve ser verificada a 0,60 m de altura (zona de conforto para pessoas sentadas). O parâmetro dinâmico (adaptativo) em função da variação da carga térmica será a temperatura do ar de insuflamento, visando-se manter constante a temperatura do ar à cota 0,60 m.

Para verificar o acerto no uso da TA, simulações foram realizadas com dados do sistema de monitoramento obtidos em [13]. Estes dados são o resultado de medições contínuas de temperatura do ar e teor de CO_2, realizadas por um conjunto de sensores fixados em cada parede lateral da sala (paredes opostas) a 2,35 m do piso, proporcionando cerca de 3000 leituras por variável, por ponto. Embora a cota das medições não seja a ideal (0,60 m), para fins de simulação estes dados mostraram-se satisfatórios. Além do mais, em condições reais de uso de um ambiente, medições contínuas a 0,60 m seriam praticamente inviáveis devido, por exemplo, a possível vandalismo junto aos sensores e distorções nas leituras causadas pela movimentação de pessoas.

Foram realizadas simulações com a ferramenta WEKA para a montagem da ADA com dados de quatro condições de uso do ambiente:

- Aula com alunos sentados utilizando computadores;
- Pós-aula (alunos em movimentação na sala);
- Sala inativa (vazia);
- Monitoria (monitor e alguns alunos).

Com a árvore montada foi possível confrontar-se a classificação gerada com os horários de uso de cada tipo de atividade da sala.

V. RESULTADOS

Visando-se avaliar a coerência da classificação, os dados foram submetidos ao algoritmo Functional Trees (implementado no WEKA), sendo utilizado o procedimento

experimental *cross-validation* (*k-fold-cross-validation*) com k = 10. A execução utilizando o *cross-validation* consiste em dividir a amostra em k vezes de teste, usualmente com k = 10. Considerando-se k = 10, a amostra é dividida em 10 partes, sendo 9 delas utilizadas para treino e uma para teste, repetindo-se o procedimento 10 vezes e calculando-se ao final a média das 10 execuções. O objetivo é criar um modelo para prever a produção de outros conjuntos de dados diferentes dos utilizados para a sua construção. O arquivo de dados de treino é utilizado para construir um modelo, enquanto o arquivo de dados de teste é utilizado para verificar se o modelo construído apresenta e mantém coerência (precisão) com diferentes conjuntos de dados. Desta forma, garante-se que o modelo será capaz de analisar dados novos e classificá-los de acordo com o conjunto de dados utilizado previamente.

A Figura 6 mostra a árvore gerada pelo algoritmo.

Um resultado interessante da execução foi o valor de Kappa. Ele é um índice análogo a um coeficiente de correlação, sendo que igual a zero, indica ausência de relação estatística entre as instâncias de uma classe e próximo a 1, uma forte relação. O Kappa do teste realizado resultou em 0,9184, muito próximo a 1, demonstrando uma forte relação estatística entre as diferentes instâncias dentro de uma mesma classe corretamente classificada.

Outros resultados relevantes foram os relativos às Instâncias Corretamente Classificadas (ICC) e às Instâncias Incorretamente Classificadas (IIC). Uma instância nesse caso é um conjunto de leituras em um determinado intervalo de tempo do monitoramento. O valor de ICC indica quantas instâncias foram classificadas de forma apropriada à sua classe, e o IIC indica quantas foram classificadas de forma incorreta. Os valores de ICC e IIC para o teste realizado foram 94,8% e 5,2% respectivamente. O ICC resultante indicou coerência da árvore gerada, representando o modelo real.

Estes resultados mostram que a classificação proposta é coerente, ou seja, demonstram uma forte relação estatística entre as instâncias dentro da classe.

No entanto, outros testes com análises mais aprofundadas mostram-se necessários para que se possam responder a questões como:

- É possível determinar-se um conjunto de classes para a qualidade do ambiente interior, de modo que cada classe corresponda a um cenário de atuação do sistema de ar condicionado?
- Genericamente, quais seriam estas classes?
- Dentro de limites pré-estabelecidos para cada variável, é possível determinar-se a classe da qualidade do ambiente interior por meio de Tabelas de Decisão?
- A TDA proposta permite a configuração automática dos *setpoints* do sistema de ar condicionado com insuflamento de ar pelo piso, mesmo quando instalado em um novo ambiente, sem a necessidade de um técnico especialista?
- O sistema reagirá automaticamente a uma mudança de uso do ambiente, "aprendendo" e se reconfigurando?
- O sistema responderá automaticamente à inclusão de novas variáveis ou sensores, "aprendendo" e se reconfigurando?

```
N2#1 <= 0.340644
|  N1#2 <= 0.743126
|  |  T-1 <= 26.8
|  |  |  N1#4 <= 0.229765
|  |  |  |  N1#5 <= 0.208509
|  |  |  |  |  N2#6 <= 0.026964
|  |  |  |  |  |  T-2 <= 24
|  |  |  |  |  |  |  CO2-1 <= 776
|  |  |  |  |  |  |  |  N0#9 <= 0.781024: FT_1:15/150 (70)
|  |  |  |  |  |  |  |  N0#9 > 0.781024: Class=0
|  |  |  |  |  |  |  CO2-1 > 776: Class=0
|  |  |  |  |  |  T-2 > 24: FT_4:15/120 (617)
|  |  |  |  |  N2#6 > 0.026964: Class=0
|  |  |  |  N1#5 > 0.208509
|  |  |  |  |  CO2-2 <= 626: Class=0
|  |  |  |  |  CO2-2 > 626: Class=1
|  |  |  N1#4 > 0.229765
|  |  |  |  N1#18 <= 0.264423: Class=2
|  |  |  |  N1#18 > 0.264423
|  |  |  |  |  N1#20 <= 0.681701
|  |  |  |  |  |  T-1 <= 22: Class=1
|  |  |  |  |  |  T-1 > 22
|  |  |  |  |  |  |  CO2-2 <= 707
|  |  |  |  |  |  |  |  T-2 <= 22.1
|  |  |  |  |  |  |  |  |  N0#25 <= 0.814639
|  |  |  |  |  |  |  |  |  |  CO2-1 <= 502: Class=0
|  |  |  |  |  |  |  |  |  |  CO2-1 > 502
|  |  |  |  |  |  |  |  |  |  |  N0#28 <= 0.420307: Class=1
|  |  |  |  |  |  |  |  |  |  |  N0#28 > 0.420307: Class=0
|  |  |  |  |  |  |  |  |  N0#25 > 0.814639: Class=0
|  |  |  |  |  |  |  |  T-2 > 22.1
|  |  |  |  |  |  |  |  |  CO2-1 <= 775
|  |  |  |  |  |  |  |  |  |  N1#33 <= 0.360909: Class=0
|  |  |  |  |  |  |  |  |  |  N1#33 > 0.360909
|  |  |  |  |  |  |  |  |  |  |  N0#35 <= 0.197423: Class=1
|  |  |  |  |  |  |  |  |  |  |  N0#35 > 0.197423
|  |  |  |  |  |  |  |  |  |  |  |  N0#37 <= 0.259154: Class=0
|  |  |  |  |  |  |  |  |  |  |  |  N0#37 > 0.259154
|  |  |  |  |  |  |  |  |  |  |  |  |  T-1 <= 23.9
|  |  |  |  |  |  |  |  |  |  |  |  |  |  N0#40 <= 0.14392: Class=1
|  |  |  |  |  |  |  |  |  |  |  |  |  |  N0#40 > 0.14392: FT_18:15/240 (34)
|  |  |  |  |  |  |  |  |  |  |  |  |  T-1 > 23.9
|  |  |  |  |  |  |  |  |  |  |  |  |  |  N0#43 <= 0.355772: FT_19:15/240 (41)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  N0#43 > 0.355772: FT_20:15/240 (45)
|  |  |  |  |  |  |  |  |  CO2-1 > 775: Class=0
|  |  |  |  |  |  |  CO2-2 > 707: Class=0
|  |  |  |  |  N1#20 > 0.681701: FT_23:15/105 (205)
|  |  T-1 > 26.8: Class=1
|  N1#2 > 0.743126: Class=1
N2#1 > 0.340644
|  N1#51 <= 0.157046
|  |  N0#52 <= 0.276217
|  |  |  N2#53 <= 0.91625
|  |  |  |  N1#54 <= 0.164136
|  |  |  |  |  T-1 <= 24.7: Class=2
|  |  |  |  |  T-1 > 24.7: FT_27:15/105 (41)
|  |  |  |  N1#54 > 0.164136: Class=2
|  |  |  N2#53 > 0.91625: Class=2
|  |  N0#52 > 0.276217
|  |  |  T-1 <= 24.7: Class=2
|  |  |  T-1 > 24.7
|  |  |  |  T-2 <= 24.4: FT_31:15/90 (95)
|  |  |  |  T-2 > 24.4: Class=2
|  N1#51 > 0.15704
|  |  N1#65 <= 0.63318: FT_33:15/60 (60)
|  |  N1#65 > 0.63318: Class=1
```

Figura 6. Árvore de Decisão obtida pelo WEKA.

VI. CONCLUSÕES

A proposta deste artigo, de criar uma metodologia para que sistemas de ar condicionado central com insuflamento de ar pelo piso possam se autoconfigurar por meio de técnicas adaptativas, mostra-se promissora e útil ao mercado.

A proposta de utilizar ferramentas IAD como única função adaptativa de uma ADA mostra-se como uma solução simples, de fácil implementação e eficaz. Porém, esta solução consome mais recursos computacionais e aumenta o tempo de execução, comparativamente a uma ADA típica.

Como trabalho futuro, pretende-se analisar as condições ambientais proporcionadas (temperatura, umidade relativa e teor de CO_2), frente ao consumo de energia correspondente, para cada sistema de ar condicionado do estudo de caso. A partir dessa análise prévia, pretende-se incluir na estratégia adaptativa de operação faixas de consumo de energia aceitáveis, tanto para os *fan coils* como para o *chiller*, concomitantes às condições ambientais internas conformes. Desta forma, espera-se chegar a estratégias de operação

efetivamente sustentáveis para os sistemas de ar condicionado em questão.

AGRADECIMENTOS

Os autores agradecem a: Conselho Nacional de Desenvolvimento Científico e Tecnológico pela bolsa de doutorado concedida; Fundação de Amparo à Pesquisa do Estado de São Paulo e Financiadora de Estudos e Projetos pelo apoio ao projeto de desenvolvimento do sistema de monitoramento mencionado.

REFERÊNCIAS

[1] Eichholtz, P.; Kok, N.; Quigley, J. M. The economics of green building. *Review of Economics and Statistics*, 2011.

[2] MitchelL, C. S. et al. Current State of the Science: Health Effects and Indoor Enviromental Quality. *Environmental Health Perspectives*, v. 115, n. 6, p. 958-964, 2007.

[3] Lan, L.; Wargocki, P.; Lian, Z. Quantitative measurement of productivity loss due to thermal discomfort. *Energy and Buildings*, v. 43, n. 5, p. 1057-1062, 2011.

[4] Eletrobrás. *Procel - Apresentação*. Rio de Janeiro: 2013. *Disponível em*: <http://www.eletrobras.com/elb/procel/main.asp?TeamID=%7BA8468F2A-5813-4D4B-953A-1F2A5DAC9B55%7D>. *Acesso em*: 05 Jan. 2013.

[5] Hitchin, R.; Pout, C.; Riviere, P. Assessing the market for air conditioning systems in European buildings. *Energy and Buildings*, v. 58, n. 0, p. 355-362, 2013.

[6] Gershenfeld, N.; Samouhos, S.; Nordman, B. Intelligent Infrastructure for Energy Efficiency. *Science*, v. 327, n. 5969, p. 1086-1088, 2010.

[7] Zhu, W.; Rui, Y.; Lingfeng, W. Multi-agent intelligent controller design for smart and sustainable buildings. *In:* Systems Conference, 2010 4th Annual IEEE, 2010, *Proceedings*... 2010. p. 277-282.

[8] Carmo, A. T.; Prado, R. T. A. Qualidade do Ar Interno. São Paulo, 1999. *Texto técnico/PCC/23*, 35pf. Escola Politécnica da USP, Departamento de Engenharia de Construção Civil.

[9] Mendell, M. J. et al. Indoor Environmental Risk Factors for Occupant Symptoms in 100 U.S. Office Buildings: Summary of Three Analyses. *EPA Base Study*. *Disponível em:* <http://eetd.lbl.gov/ied/pdf/LBNL-59659.pdf>. *Acesso em:* 30 Nov. 2006.

[10] Reynolds, S. J. et al. Indoor Environmental Quality in Six Commercial Office Buildings in the Midwest United States. *Applied Occupational and Environmental Hygiene*, v. 16, n. 11, p. 1065-1077, 2001.

[11] Fisk, W.; Seppanen, O. Providing Better Indoor Environmental Quality Brings Economic Benefits. *In:* 9th REHVA World Congress Clima: Wellbeing Indoors, 2007, Helsinki. *Proceedings* ...Helsinki: 2007.

[12] Erickson, V. L. et al. Energy efficient building environment control strategies using real-time occupancy measurements. *In:* 1st ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings - BuildSys '09, 2009,. *Proceedings* ... Berkeley, California, New York, NY, USA: ACM, 2009. p. 19–24.

[13] Marè, R. M. Estudo de Eficiência da Ventilação em Sistema de Climatização com Distribuição de Ar pelo Piso. São Paulo, 2010. *Dissertação de Mestrado*, 205 p. Escola Politécnica da USP.

[14] Witten , I.H. et al. *Data mining: Practical machine learning tools and techniques*. 3ed. San Francisco: Morgan Kaufmann. 2011. [14]

[15] Tarca, A. L. et al. Machine learning and its applications to biology. *PLoS Comput Biol*. 2007. [15]

[16] Pistori, H.; Neto, J. J. AdapTree - Proposta de um Algoritmo para Indução de Árvores de Decisão Baseado em Técnicas Adaptativas. *In:* Conferência Latino Americana de Informática - CLEI 2002. *Anais* ... 2002.

[17] The University of Waikato *WEKA*. *Disponível em*: <http://www.cs.waikato.ac.nz/ml/weka/>. *Acesso em:* 20 Dez. 2014.

[18] Tchemra, A.H. Aplicação da Tecnologia Adaptativa em Sistemas de Tomada de Decisão. *IEEE América Latina*. Vol. 5, Num. 7, ISSN: 1548-0992, p. 552-556, 2007.

[19] Pistori, H.; Souza, K. P. Tecnologia Adaptativa Aplicada à Biotecnologia: Estudos de Caso e Oportunidades. *In:* 4o. Workshop de Tecnologia Adaptativa, *Anais* ...São Paulo, p. 16 – 21, 2010.

[20] World Wide Web Consortium. SOAP Specifications. Cambridge: 2007. Site Internet da especificação do protocolo SOAP, mantido pelo W3C. *Disponível em:* <http://www.w3.org/TR/soap/>. *Acesso em*: 01 Ago. 2007.

[21] Fielding, R. T. Architectural Styles and the Design of Network-based Software Architectures. 2000. Irvine, *Doctoral thesis*.

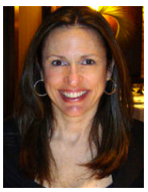
[22] Stein, G. WebDAV Resources. *Disponível em:* <http://www.webdav.org>. *Acesso em*: 01 Jul. 2012.

[23] Leite, B.C.C. Sistema de Ar Condicionado com Insuflamento pelo Piso em Ambientes de Escritórios: Avaliação do Conforto Térmico e Condições de Operação. 2003. 152 p. *Tese de Doutorado* - Escola Politécnica, Universidade de São Paulo, São Paulo, 2003.

[24] Associação Brasileira de Normas Técnicas. NBR 16401. Instalações de ar condicionado – Sistemas centrais e unitários Parte 2: Parâmetros de conforto térmico. Brasil, 2008.

[25] American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc. ANSI/ASHRAE Standard 55: Thermal Environmental Conditions for Human Occupancy. Atlanta - USA: 2010.

[26] International Standards Organization. ISO 7730: Ergonomics of the thermal environment -- Analytical determination and interpretation of thermal comfort using calculation of the PMV and PPD indices and local thermal comfort criteria. Geneva, 2005.



Renata Maria Marè é graduada em Engenharia Civil pela Escola de Engenharia Mauá (1985), mestre em Engenharia de Construção Civil e Urbana (2010) e doutoranda em Engenharia de Computação e Sistemas Digitais (desde 2013), ambos pela Escola Politécnica da USP (EPUSP). Possui MBA em Gestão Empresarial pelo Instituto Mauá de Tecnologia (2011). Suas principais áreas de pesquisas são: Sustentabilidade em Edificações, Edifícios Inteligentes, Cidades Inteligentes, Redes de Sensores para Monitoramento.

Marcelo Freire de Barros é graduado em Engenharia Elétrica pela Universidade de São Paulo USP (1984), MBA em finanças pela Fundação Instituto de Administração - FIA (2006), mestrado em engenharia elétrica pela Escola Politécnica da Universidade de São Paulo - USP (2010). Tem experiência em pesquisa e desenvolvimento na área de Engenharia Elétrica e de Computação, com ênfase em Automação Industrial, desenvolvimento de PLCs, IHMs e interfaces de comunicação.

Mara Andrea Dota é graduada em Ciência da Computação pela Universidade Estadual Paulista Júlio de Mesquita Filho (1998), mestre em Ciências da Computação e Matemática Computacional pelo Instituto de Ciências Matemática e Computação da USP (ICMC/USP - 2001) e doutora em Engenharia de Computação e Sistemas Digitais pela Escola Politécnica da USP (EPUSP – 2014). Atualmente é docente da Universidade Federal de Mato Grosso (UFMT). Tem experiência na área de Ciência da Computação, com ênfase em Linguagens de Programação, Redes de Sensores e Sistemas Distribuídos.

Carlos Eduardo Cugnasca é graduado em Engenharia de Eletricidade (1980), mestre em Engenharia Elétrica (1988) e doutor em Engenharia Elétrica (1993) pela Escola Politécnica da USP (EPUSP). É livre-docente (2002) pela EPUSP. Atualmente, é Professor Associado 3 da EPUSP e pesquisador do Laboratório de Automação Agrícola (LAA) do Departamento de Engenharia de Computação e Sistemas Digitais (PCS) da EPUSP. Tem experiência na área de Supervisão e Controle de Processos e Instrumentação, aplicadas a processos agrícolas e Agricultura de Precisão, atuando principalmente nos seguintes temas: instrumentação inteligente, sistemas embarcados em máquinas agrícolas, monitoração e controle de ambientes protegidos, redes de controle baseados nos padrões CAN, ISO11783 e LonWorks, Redes de Sensores Sem Fio e computação pervasiva. É editor da Revista Brasileira de Agroinformática (RBIAgro).

Brenda Chaves Coelho Leite é graduada em Engenharia Civil pela Universidade Federal de Minas Gerais (1979), mestre em Arquitetura e Urbanismo pela Universidade de São Paulo (1997) e doutora em Engenharia Mecânica pela Escola Politécnica da USP (2003). Atualmente é Professor Doutor no Departamento de Engenharia de Construção Civil

da Escola Politécnica da Universidade de São Paulo. Tem experiência na área de conforto térmico e qualidade do ar em edifícios, Sistemas de HVAC, Tecnologias para distribuição de ar pelo piso, conforto térmico individualizado e painéis radiantes; teto verde; simulação de desempenho energético de edifícios; Dinâmica de Fluidos Computacional (CFD); Sistemas de Automação e controle aplicados à climatização de ambientes.

Controle de acesso adaptativo através do monitoramento do ambiente

G. M. Toschi[1], C. E. Cugnasca[2]

***Abstract* — This paper presents the proposal of computer system model for the control of people access in buildings and facilities through adaptive techniques to make it sensitive to the context and take into account the history of the environment. Adaptive techniques have been incorporated into two parts of this process, at the monitoring of the environment for the calculation of the level of threat, and at the calculation of levels of access of each room of a building. In this work we propose, the union of these processes to compose a broader, more flexible and safer system.**

***Keywords* — monitoring, inbound and outbound flow of people, access control, adaptive techniques.**

I. Introdução

Os sistemas de controle de acesso (SCA) vem sendo utilizados há muito tempo, inicialmente com o controle sendo realizado por um porteiro. Trata-se de uma forma pouco eficiente, de baixa segurança e dependente da habilidade e competência de uma pessoa.

Com o intuito de melhorar esse processo, sistemas automatizados foram gradualmente introduzidos em função da tecnologia disponível em cada época. Atualmente busca-se automatizar cada vez mais o sistema de acesso, conseguindo maior eficiência e segurança.

O controle de acesso as instalações vem se tornando cada vez mais importante para a segurança. Contudo, em função das particularidades de cada local, há a necessidade de adaptações, de forma que uma característica desejável nos sistemas de controle é a flexibilidade, viabilizando a implantação de sistemas mais eficazes e seguros.

Diversos modelos de controle de acesso têm sido estudados, um exemplo são os tradicionais, que são independentes de contexto. Atualmente , novos modelos estão surgindo como o de autorização contextual, na qual variáveis de ambiente alteram o comportamento do sistema. Um outro modelo é o controle de acesso baseado em papéis (CABP) [5]. O *National Institute of Standards and Technology* (NIST) propõe um padrão para o CABP, que estabelece um modelo referencial de autorização contextual.

Em [5] descreve-se uma especialização desse modelo, que foi aplicado no controle de acesso sensível ao contexto aplicado no Instituto do Coração do Hospital das Clínicas da Faculdade de Medicina da Universidade de São Paulo (InCor).Formas de fazer esse controle de acesso se tornar cada vez mais dinâmico estão em estudo, com o objetivo de fazer com que ele se adapte às condições do ambiente para proporcionar maior segurança.

Para que o SCA se adapte ao ambiente, este deve possuir um sistema de monitoramento, que prove a capacidade de coletar e processar informações sobre o ambiente e sobre as pessoas que nele circulam.

Encontram-se disponíveis no mercado sistemas de monitoramento baseados em diversas tecnologias e sensores [1], destinados a coletar os dados do ambiente. Um exemplo é o monitoramento do centro de Londres, que é feito por câmeras espalhadas com movimento rotacional e *zoom* para seguir automaticamente o foco de atividades suspeitas durante a noite. Técnicas de inteligência artificial são utilizadas para classificar movimentações com características suspeitas, alertando os operadores, que são os responsáveis pela confirmação da identificação e ações a realizar em cada caso.

O objetivo deste trabalho é apresentar uma proposta de evolução para o modelo de SCA utilizando técnicas adaptativas para torna-lo mais sensível ao contexto através do ajuste dinâmico dos intervalos de cálculo do nível de ameaça do ambiente. A seção II discute o nível de segurança e ameaça e como o cálculo deste pode ser realizado de forma adaptativa. A seção III descreve o processo de controle de acesso predial de acordo com a patente americana em [6], enquanto a seção IV apresenta um modelo de controle de acesso adaptativo. A seção V apresenta futuras simulações utilizando a ferramenta AdapTools [9], enquanto a seção VII discute os resultados e apresenta as conclusões. Por fim, a última seção apresenta a lista de referências.

II. Nível de Ameaça/Segurança

Um sistema de monitoramento predial consegue calcular o nível de ameaça referente ao ambiente por meio das variáveis monitoradas, um processo denominado *multisensor* [1], que rastreia os usuários durante sua permanência dentro do prédio, registrando todos os seus movimentos e ações. Por exemplo, o sensor de luminosidade pode indicar o período noturno, no qual o índice de ocorrências de furtos ou roubos é mais elevado. Configurando-se convenientemente as regras de interpretação dos sinais dos sensores, pode-se calcular o nível de ameaça em cada instante.

[1] Eng. Eletricista, Mestrando, Escola Politécnica da Universidade de São Paulo.
[2] Eng. Eletricista, Mestre, Doutor e Livre-Docente, Professor Associado 3, Escola Politécnica da Universidade de São Paulo.

Esse exemplo evidencia que é possível identificar padrões de comportamentos a partir com os dados coletados e, utilizando-se as regras contextuais de autorização do processo de CABP, pode-se fazer o cálculo do nível de ameaça [5].

A. Modelo adaptativo para o cálculo de ameaça

Uma das formas de aplicação do modelo adaptativo no controle de acesso consiste em mudar o intervalo de amostragem das variáveis do ambiente, utilizadas para o cálculo do nível de ameaça. Para o controle dessa frequência é proposto um Autômato Adaptativo (AA) [10] descrito a seguir que foi adaptado de [9].

Definem-se limites para o nível de ameaça, e quando algum dos valores amostrados das variáveis monitoradas ultrapassá-los, a frequência de amostragem é alterada. Dessa forma, quando esse nível ultrapassar um limite superior, a frequência aumenta; ao retornar a um valor inferior a um dos limites, a frequência volta a diminuir.

A Figura 1 mostra um exemplo de frequência de cálculo do nível de ameaça ao longo do tempo, no qual o nível de ameaça é calculado a partir de amostragens das variáveis de um ambiente. Foram definidos somente dois limites, mas podem ser definidas varias faixas de valores, definidas por dois limites, dentro dos quais a frequência terá um valor específico. A frequência dessas amostragens e do cálculo depende do nível calculado: quando esse nível sai da região de normalidade (faixa definida por a e b). o intervalo entre uma amostra e outra é aumentado, alterando a frequência do cálculo.

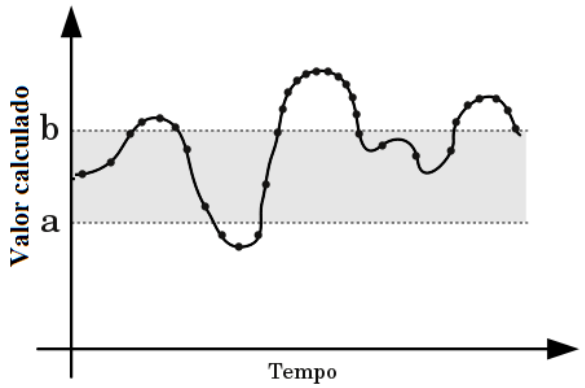


Figura 1 - Exemplo de uma sequência de cálculo do nível de ameaça ao longo do tempo. Adaptado de [9]

III. Controle de Acesso

O processo de controle de acesso consiste basicamente em identificar a pessoa que está se movimentando, validar suas permissões e responder com a liberação ou negação do seu acesso dessa pessoa ao ambiente controlado, e eventualmente acionando outras medidas de segurança para proteger o ambiente.

A Figura 2 mostra o fluxograma base da patente americana de um sistema de segurança para o controle de acesso predial [6]. Esse processo tem entradas de variáveis, por exemplo, a luminosidade do ambiente, e essas variáveis são utilizadas para controle do ambiente externo, como acender a luz quando estiver escuro. Porém a validação de acesso é independente das condições do ambiente.

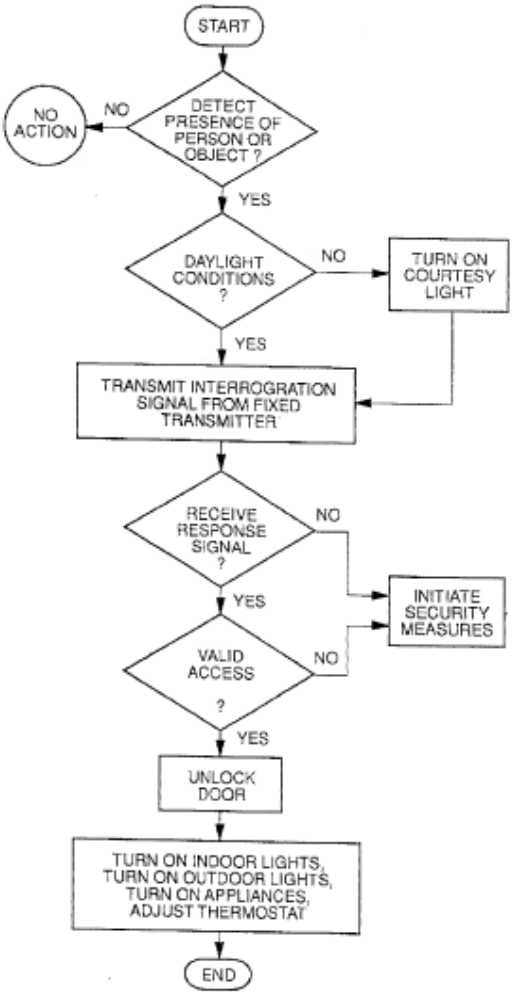


Figura 2 - Fluxograma de controle de acesso extraído de [6]

IV. Modelo de Controle de Acesso Adaptativo

O modelo de controle de acesso da Figura 2 foi estendido para um modelo de controle de acesso adaptativo, baseado no nível de ameaça calculado. Para isso a etapa de validação de acesso do usuário foi modificada para torná-la adaptativa ao nível de ameaça calculado.

A. Cálculo da autorização de acesso

O cálculo da autorização de acesso pode ser feito utilizando-se duas variáveis:

- Nível de permissão do usuário;
- Nível de permissão do ambiente.

Para um usuário ter acesso a um ambiente, seu nível de permissão deve ser maior ou igual ao do ambiente. Dessa forma o nível de permissão do ambiente torna-se o limite inferior para a permissão de acesso do usuário.

Na Figura 3 podem-se visualizar os níveis de acesso fixos, não adaptativos. Um ambiente que tenha nível de acesso 4, por exemplo, sempre exigirá um nível de permissão do usuário maior ou igual a 4 independentemento do contexto.

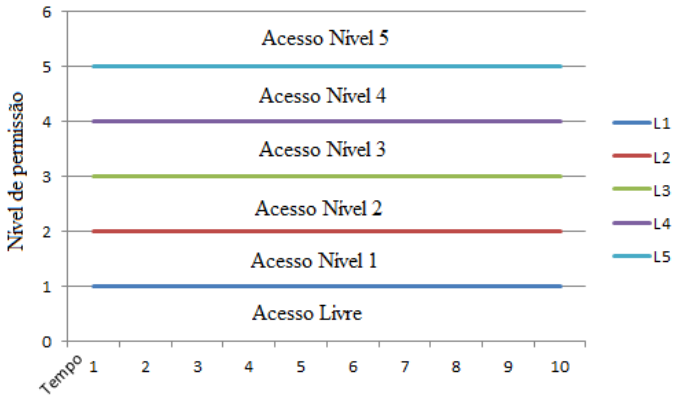


Figura 3 - Níveis de acesso estáticos

B. *Calculo da autorização de acesso adaptativo*

Para tornar o controle de acesso mais seguro, pode-se fazer o cálculo da autorização de acesso de modo adaptativo. Dependendo do nível de ameaça do ambiente, calculado pelo sistema de monitoramento, o grau de permissão dos ambientes pode aumentar, diminuir ou podem ser criados novos níveis de permissão intermediários para melhor se adequar à situação do prédio. O nível de acesso é composto por um nível base, *offset*, somado a um variável proposcional ao nível de ameaça calculado. A formula a seguir representa esse modelo:

Nível de Permissão = offset + c*(Nível de Ameaça)

Na Figura 4 podem-se observar os limites dinâmicos dos diferentes níveis de acesso. No Instante de Tempo 2 algum evento externo ocorreu, modificanto o nível de ameaça calculado, e assim, os limites para os diferentes níveis de acesso são alterados dinamicamente. Cada ambiente pertence a um nível de acesso e, portanto, somente os usuários com nível de permissão maior ou igual aos limites estabelecidos para acesso em determinado momento serão autorizado a entrar no ambiente.

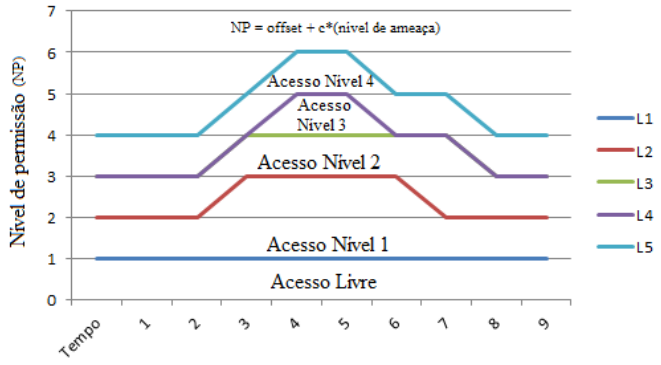


Figura 4 - Níveis de acesso adaptativos

Podem-se observar que o acesso Nível 3 somente surge em um momento específico, Instante de tempo 3, para suprir uma necessidade momentânea, sendo depois descartado, quando a situação volta ao normal.

C. *Controle de acesso adaptativo*

A Figura 6 apresentado a extensão do passo de validação de acesso, apresentado na Figura 5, que será substituido no fluxograma de controle de acesso apresentado na Figura 2. O processo de validação de acesso é baseado no cálculo da autorização de acesso adaptativo da Figura 4.

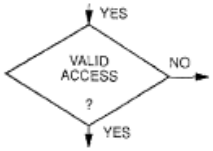


Figura 5 - fluxograma de controle de acesso original retirado de [6]

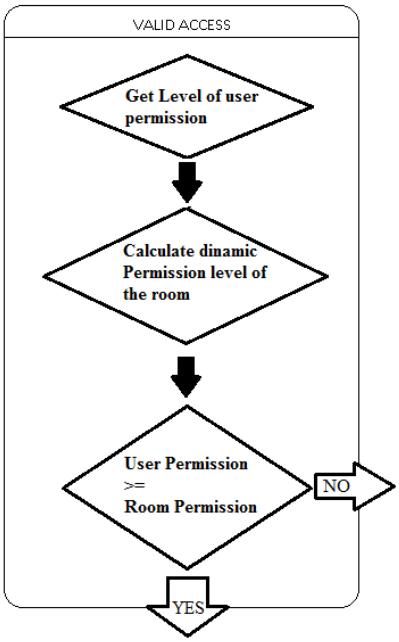


Figura 6 - fluxograma de controle de acesso adaptativo

Utilizando-se a nova forma de cálculo, o controle de acesso se torna adaptativo às condições do ambiente.

Os sensores espalhados pelo prédio se tornam entradas de dados do sistema. A entrada de dados é adaptativa em relação às condições do ambiente, variando a frequência de amostragem de acordo com o nível de ameaça calculado.

A partir deste nível calculado podem-se definir dinamicamente as faixas de permissão de cada ambiente do prédio. Dessa forma as faixas de permissão se tornam adaptativas às condições do ambiente também.

A utilização de técnicas adaptativas para o controle de acesso tem como objetivo tornar mais seguro esse processo, de forma a garantir a integridade do prédio que tem um sistema inteligente e adaptativo.

V. Simulações Futuras

A ferramenta AdapTools [8] pode ser utilizada para realização de simulações computacionais de um modelo adaptativo. O AA descrito anteriormente será construído no AdapTools para gerar a simulação. A partir dessa simulação pode-se validar a utilização desse AA para o calculo do nível de ameaça e do controle de acesso de ambientes.

Em uma primeira etapa de simulação pode-se utilizar o modelo adaptativo de cálculo de ameaça com apenas uma variável de ambiente para simplificação da simulação. Em

seguida, utiliza-se esse nível de ameaça calculado para a verificação da autorização de acesso adaptativo. Dessa forma pode-se visualizar como a alteração de uma variável do ambiente interfere em ambos os cálculos, e diretamente influenciar o acesso de um usuário ao ambiente controlado.

Em uma segunda etapa pode-se adicionar múltiplas variáveis de ambiente, que são alteradas para simular as condições do ambiente de um prédio. Assim, pode-se analisar a simulação de forma mais próxima de um sistema em ambiente real.

V. VI. Comentários e Discussões

Este artigo apresentou o estudo do processo de monitoramento adaptativo para o calculo do nível de ameaça. Em seguida apresentou o estudo do processo de controle de acesso adaptativo que utiliza os dados do processo de monitoramento para calcular os níveis de acesso dinamicamente e assim tornando-se sensível as condições externas do ambiente.

A integração desses dois processos em um único sistema proporcionou maior flexibilidade para adaptação as políticas de controle de acesso mais complexas que se moldam ao contexto, para garantir um melhor e mais efetivo controle de monitoramento do ambiente.

A abordagem utilizada neste artigo para tornar o controle de acesso adaptativo também pode ser utilizada em ambientes virtuais. Da mesma forma que o processo de controle de acesso valida o acesso de um usuário à um ambiente de modo adaptativo, como foi descrito neste artigo, o processo de controle de acesso virtual pode determinar as permissões de acesso, leitura, escrita e de execução de um usuário, ou um grupo de usuários, em um ambiente virtual. Por exemplo, em um ambiente corporativo, este processo pode determinar se um usuário poderá se conectar ou não em um servidor dependendo das variáveis externas amostradas. Usuários de um grupo de baixo nível de permissão só podem se conectar aos servidores corporativos durante o dia estando dentro da empresa, por exemplo.

Nesse ambiente virtual as variáveis podem ser físicas, lidar no ambiente externo, ou variáveis virtuais, por exemplo, o volume de usuários conectados no servidor ou se o servidor está sendo alvo de um ataque *hacker*.

Dessa forma o ambiente virtual também pode se tornar mais seguro através do controle adaptativo de acesso.

Referências

[1] L. Osadciw, P. Varshney, e K. Veeramachaneni, “Improving personal identification accuracy using multisensor fusion for building access control applications”, in Information Fusion, 2002. Proceedings of the Fifth International Conference on, 2002, vol. 2, p. 1176–1183.

[2] C. Neves, L. Duarte, N. Viana, e V. Ferreira, “Os dez maiores desafios da automação industrial: as perspectivas para o futuro”, in II Congresso de Pesquisa e Inovacao da Rede Norte Nordeste de Educa\ccao Tecnológica, Joao Pessoa, Paraıba, Brasil, 2007.

[3] E. Oliveira, V. Nogueira, e S. Brito, “Controle de fluxo de automóveis com RFId”.

[4] Prof. Dr. A. R. Hirakawa e Prof. Dr. J. S. C. Martini, “SIGINURB – Sistema de Gestão da Infraestrutura Urbana”, Escola Politécnica da Universidade de São Paulo

[5] G. Motta e S. S. Furuie, “Um modelo de autorização contextual para o controle de acesso baseado em papéis”, in II Workshop em Segurança de Sistemas Computacionais (WSeg2002), 2002, p. 137–144.

[6] D. C. Duhame e D. V. Meyvis, “Security system for controlling building access”, U.S. Patent 5,541,585jul-1996.

[7] N. T. Nguyen, S. Venkatesh, G. West, H. H. Bui, e A. Perth, “Multiple camera coordination in a surveillance system”, Acta Automatica Sinica, vol. 29, no 3, p. 408–422, 2003.

[8] L. D. JESUS, D. G. D. SANTOS, A. A. D. CASTRO, H. PISTORI. WTA 2007 - II.2 - “AdapTools 2.0: Implementation and Utilization Aspects.” IEEE Latin Am. Trans. 5, 527-532 (2007).

[9] SANTOS, I. M. ; CUGNASCA, C. E. Autômato Adaptativo para definição autônoma do intervalo de amostragem de dados em Rede de Sensores Sem Fio. In: VI Workshop de Tecnologia Adaptativa, 2012, São Paulo. Memória do VI Workshop de Tecnologia Adaptativa 2012. São Paulo: Ed. USP, 2012.

[10] J. J. Neto. “Adaptive rule-driven devices – general formulation and case study” *Revised Papers from the 6th International Conference on Implementation and Application of Automata*, *CIAA 2001*, London, UK: Springer-Verlag, 2002, pp. 234-250, ISBN 3-540-00400-9.

Determinação do nível de conforto adaptativo em transporte público baseado em lógica nebulosa dependente do tempo

Bruno Pimentel Machado, André Riyuiti Hirakawa, José Sidnei Colombo Martini

Abstract — Transportation is a key issue in metropolitan areas. Several studies focus on different aspects related to mobility in cities, since the steering assistance solutions to studies that seek the best way to assess the level of satisfaction of users on public transportation. An aspect rarely mentioned in studies is related to the level of comfort in public transportation, and how it can influence the selection of best transportation mean. By its dynamic and subjective properties, traditional solutions of selecting best route do not have the best results. This work proposes an adaptive solution based on fuzzy logic with dependence on time. Tests were conducted to evaluate the proposal and the results suggest a higher quality in the decision making process.

Keywords — Adaptive Solution, Fuzzy Logic, Time-Dependent

Resumo — Transporte é um assunto chave em áreas metropolitanas. Vários estudos focam em diferentes aspectos associados à mobilidade nas cidades, desde soluções de assistência a direção até estudos que buscam a melhor forma de avaliar o nível de satisfação de usuários em transporte público. Um aspecto pouco mencionado nos estudos está relacionado ao nível de conforto em transporte público, e como ele pode influenciar na seleção de melhor meio de transporte. Por suas propriedades dinâmicas e subjetivas, as soluções tradicionais de seleção de melhor rota não apresentam os melhores resultados. Este trabalho propõe uma solução adaptativa baseada em lógica nebulosa com dependência no tempo. Testes foram conduzidos para avaliar a proposta e os resultados sugerem uma maior qualidade no processo de tomada de decisão.

Palavras-Chave — Adaptatividade, Lógica Nebulosa, Dependência do Tempo

I. Introdução

Transporte é um assunto chave em áreas metropolitanas. Vários estudos focam em diferentes aspectos associados à mobilidade nas cidades, desde soluções de assistência a direção até estudos que buscam a melhor forma de avaliar o nível de satisfação de usuários em transporte público [1][2][3][4][6][7][11][12][13]. Um aspecto pouco mencionado nos estudos está relacionado ao nível de conforto em transporte público, e como ele pode influenciar na seleção de melhor meio de transporte. Por suas propriedades dinâmicas e subjetivas, as soluções tradicionais de seleção de melhor rota não apresentam os melhores resultados.

Este estudo apresentará uma forma de avaliar o nível de conforto no ponto de vista do usuário, e assim suportar uma decisão de melhor caminho. Um algoritmo baseado em números nebulosos com dependência do tempo será aplicado para a determinação do conforto. A dependência do tempo permite uma adaptação contínua da avaliação dos resultados e eventual mudança de rumo.

Os critérios que fazem um usuário escolher um ônibus ao invés de outro estão relacionados ao que o usuário entende por melhor opção, o que incorpora dentre outros temas o tempo de viagem, seu custo, o nível de conforto, disponibilidade e outros. A todos estes elementos, podemos associar o conceito de Satisfação do Cliente, que em diversos estudos aparece com definições distintas [1][2][3][4][6].

II. Conforto

Ainda que, Cantwell e Caulfield [1], Eboli e Mazzulla [2], Kostakis e Pandelis [3], Karlsson e Larlsson [4] e Yeh, Deng e Chang [6] concordem que conforto é uma variável importante na composição da satisfação do cliente, eles não possuem o mesmo entendimento do que o define.

Em [6], conforto foi associado ao nível de serviço oferecido pelas empresas de ônibus e a percepção dos clientes em reação à qualidade deste serviço. Neste estudo, conforto foi desmembrado em vários aspectos (qualidade do ar condicionado, disponibilidade de informação, limpeza). Em [2], o conforto também aparece como um fator importante na definição de satisfação de cliente, no entanto, a definição de seu conceito é diferente. Para Eboli e Mazzulla, conforto depende de questões como odor e barulho enquanto a limpeza aparece como um tópico independente. Karlsson e Larsson entendem conforto usando outros fatores [4].

No ponto de vista filosófico [5], o conforto pode ser definido como um fenômeno complexo que varia de acordo com a experiência individual. Mais do que isso, a percepção de conforto tem um comportamento que degrada ao longo do tempo, e os pesos de cada variável que o definem também mudam [4]. Este comportamento justifica o uso de lógica nebulosa para tratar as incertezas do cenário.

Para este estudo, conforto foi definido como uma coleção de pares de variáveis. Cada par de variáveis é baseada em um atributo e seu grau de importância [2][4][6]. A importância será baseada na percepção do usuário e terá um comportamento dinâmico ao longo do tempo [4]. Mesmo o tempo terá um aspecto subjetivo a seu respeito, sendo classificado no formato de variáveis linguísticas dentro do conceito de lógica nebulosa.

III. Lógica Nebulosa Dependente do Tempo

Lógica nebulosa foi aplicada em diferentes cenários relacionados à modelagem de tráfego. Em [17], a lógica nebulosa foi combinada com algoritmos genéticos para suportar um problema associado a quadro de horários de transporte público. As diferentes rotas de ônibus foram mapeadas de acordo com o tempo de viagem de uma estação a outra, assim como o tempo de espera previsto em cada estação. O algoritmo permitiu estabelecer as melhores janelas de tempo para que o usuário pudesse se deslocar na rede com o tempo adequado.

A lógica nebulosa também foi usada em outros contextos envolvendo dependência de tempo. Em [14] e [15], a lógica foi usada para tomar ações baseados em entradas regulares de dados dentro de uma janela de tempo. A diferença de valores entre uma entrada de dados t_1 e t_n (em que n é maior do que um) representava uma tendência, o que era usado para suportar a tomada de decisão. O robô Nomad 200 [15] foi desenvolvido sobre um conjunto de 140 regras para navegar em um circuito com vários objetos no caminho de forma adequada. O processo de regras assumia uma entrada regular de dados. Ao longo do tempo, cada variável era usada como uma memória e a noção de tendência, gerada por estas memórias, foi usada para tomar decisões de desviar ou não de obstáculos.

Em [7], uma rede nebulosa baseada no tempo foi proposta para definir a rota mais segura que deveria ser usada para chegar a um dado destino. Huang e Ding estabeleceram premissas relacionadas aos horários de saída de cada nó, representando horários de saídas de ônibus de terminais. Se o usuário está no veículo no horário previsto de saída, ele está associado à previsão de chegada daquela viagem específica, caso chegue atrasado, um novo tempo previsto de deslocamento deve ser considerado por se tratar de um ônibus diferente. Os autores comparam a solução de rede nebulosa com algoritmos clássicos (como programação dinâmica e Dijkstra) e obteve um desempenho melhor de processamento.

Em seu trabalho, Huang e Ding consideraram a restrição de tempo como o objetivo a ser cumprido para o deslocamento, porém existem situações em que outros fatores precisam ser considerados. Quando outros fatores envolvem a percepção do usuário, uma preocupação deve ser observada. A opinião de usuários não pode ser (realisticamente) coletada sem que haja uma influência negativa em seu resultado [16]. Como alternativa, ao invés de recorrentemente questionar o usuário quanto a sua percepção, uma solução de números nebulosos dependentes do tempo, associada ao mapeamento do comportamento padrão da percepção dos usuários, pode ser aplicada para se adaptar e escolher a melhor alternativa dentro da malha.

Variáveis nebulosas são comumente representadas por figures trapezoidais ou triangulares, como definido na equação (1):

$$\mu_A(x) = \begin{cases} 0, x < a \\ \frac{x-a}{b-a}, a < x \leq b \\ 1, b < x < c \\ \frac{d-x}{d-c}, c \leq x < d \\ 0, x > d \end{cases} \quad (1)$$

No cenário com dependência no tempo, $\mu_A(x)$ pode não ser o mesmo ao longo do tempo. Seu valor pode sofrer alterações, dessa forma seu valor pode ser mais bem representado pela equação (2):

$$\mu_A(x(t)) = \begin{cases} 0, \ x(t) < a(t) \\ \frac{x(t)-a(t)}{b(t)-a(t)}, \ a(t) < x(t) \leq b(t) \\ 1, \ b(t) < x(t) < c(t) \\ \frac{d(t)-x(t)}{d(t)-c(t)}, \ c(t) \leq x(t) < d(t) \\ 0, \ x(t) > d(t) \end{cases} \quad (2)$$

Visualmente, o domínio dos números nebulosos resultaria em uma visão 3D como apresentado na Fig 1.

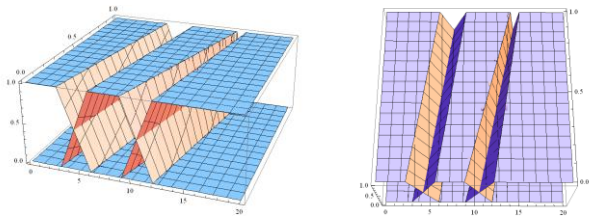


Fig. 1. Visão 3D de números nebulosos

Para determinar o grau de pertinência em um momento específico, o tempo deve ser usado para 'fatiar' o domínio no momento adequado, como representado na Fig. 2.

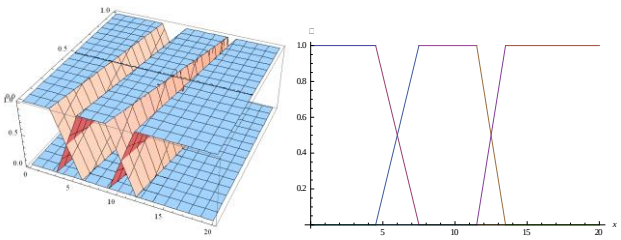


Fig. 2. Representação de corte

O processo de defuzzificação se mantém estático, mas o mapeamento do comportamento permite uma leitura dinâmica do grau de pertinência da variável.

IV. Proposta

Conforto foi definido como uma coleção de variáveis subjetivas, influenciadas pelo tempo e que é usado para suportar a determinação do índice de satisfação do cliente. Mesmo as variáveis discretas estão sujeitas à variação no tempo, como por exemplo, a disponibilidade de assentos (interpretação booleana).

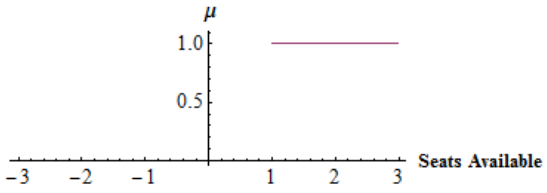


Fig. 3. Representação de Disponibilidade de Assentos

Na figura 3, a disponibilidade de assentos é apresentada em uma representação nebulosa. A partir do momento que há um assento disponível, o grau de pertinência é representado pelo valor 1. No entanto, sua importância poderia variar no tempo [4] de acordo com a função $f(importância) = Importância_{t_0} + \mathrm{Log}(tempo)$, o que resulta na curva apresentada na Figura 4.

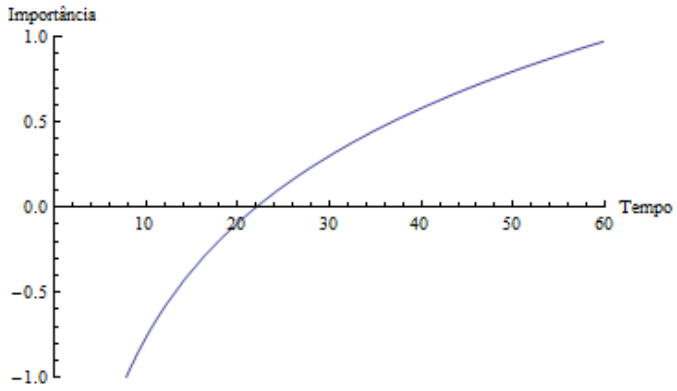


Fig. 4. Variação da importância no tempo

Se a importância for definida como um conjunto nebuloso com três variáveis linguísticas (Não Importante, Importante e Muito Importante), a Importância poderia ser graficamente representada pela Figura 5.

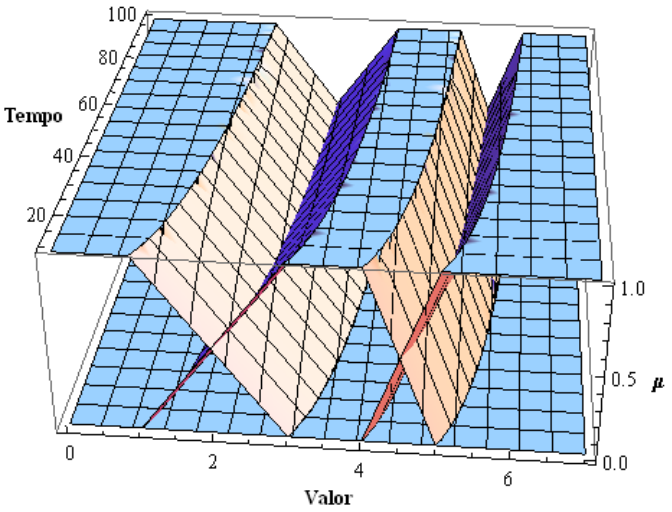


Fig. 5. Conjunto Nebuloso no tempo

A mesma lógica pode ser repetida por todas as variáveis de propriedades similares. Assim sendo, para verificar a proposta, dados foram produzidos e cenários de testes criados.

V. Prova de Conceito

O índice de conforto foi calculado baseado no índice de satisfação de cliente demonstrado por Eboli e Mazzulla em [2], em que a satisfação de cliente é formada por um conjunto de pares de elementos (variável e peso) e seu índice é dado pela média ponderada de todas as variáveis multiplicadas pelos pesos.

$$Conforto = \sum_{i=0}^{n} \frac{variável_i * peso_i}{peso_i} \qquad (2)$$

O conforto será calculado de forma semelhante como apresentado na equação (2), mas para fins de simplificação, foi definido com apenas duas propriedades de acordo com a Figura 6.

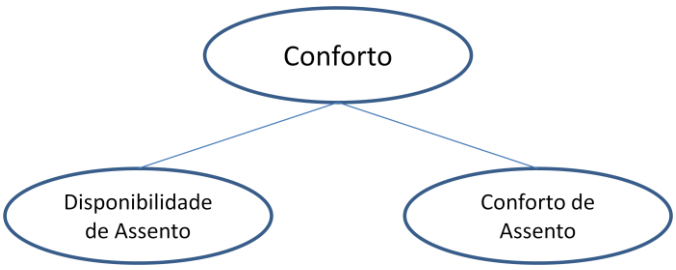


Fig. 6. Definição de Conforto

Ambas as variáveis (disponibilidade de assentos e conforto de assentos) são baseadas em valores e importâncias como apresentado por Karlsson e Larsson [4]. A importância sofre variação no tempo para as duas variáveis. Baseado ainda no trabalho dos autores, *tempo* foi definido como uma variável

nebulosa com três variáveis linguísticas: curto tempo [0, 14], tempo regular [14, 29] e longo tempo [30, ∞].

Disponibilidade de assentos foi definida como uma variável discreta {verdadeiro, falso} com um grau de importância de quatro níveis {Sem importância (1), Pouca Importância (2), Importante (3), Muito importante (4)}. Já o conforto dos assentos foi definido como {Completamente desconfortável (1), Parcialmente desconfortável (2), Parcialmente confortável (3), Completamente Confortável (4)} com a mesma escala de importância definida para a variável de disponibilidade.

Nestas condições, quarto cenários de testes foram validados de acordo com a Tabela I:

Cenário	Condição	Valor
1	Tempo	40,0
	Assento disponível	Sim
	Importância de disponibilidade	1,5
	Assento confortável	2,5
	Importância de conforto assento	0,9
2	Tempo	12,0
	Assento disponível	Sim
	Importância de disponibilidade	3,0
	Assento confortável	0,5
	Importância de conforto assento	2,3
3	Tempo	30,0
	Assento disponível	sim
	Importância de disponibilidade	3,5
	Assento confortável	2,5
	Importância de conforto assento	3,8
4	Tempo	35,0
	Assento disponível	Não
	Importância de disponibilidade	3,0
	Assento confortável	N/A
	Importância de conforto assento	N/A

Tabela I. Lista de Cenários de Teste

A variação da importância foi baseada na rota Orange Express apresentada em [4]. De acordo com o estudo, as importâncias da disponibilidade de assentos e do conforto dos assentos variam com o tempo de maneiras distintas. Viagens mais longas sugerem que a disponibilidade de assentos é mais importante do que o conforto que ele oferece. As diferenças sugerem que para a disponibilidade de assentos, a importância varia em um padrão logarítmico, mas para o conforto de assentos, é uma variação linear.

Com base nestas observações, a importância da disponibilidade foi definida pela equação (3):

$$f(t) = (\log_{22}(t) - 1) * 3 \qquad (3)$$

Já para a importância do conforto dos assentos a equação (4) utilizada foi definida por:

$$f(t) = \frac{t}{40} - 0.55 \qquad (4)$$

Vale ressaltar que mapear o comportamento não é uma atividade trivial. As funções adotadas são simplificações do comportamento destas variáveis.

Os quatro cenários propostos foram executados para duas condições:

- Lógica Nebulosa com dependência de tempo (LNDT)
- Lógica Nebulosa Tradicional sem variação no tempo (LNT)

O número nebuloso *conforto* foi calculado com base nas regras presentes na Tabela II. Foi definido um valor inicial para o índice no valor de 100. O valor inicial é usado como uma referência para permitir a comparação entre os processos, uma vez que se assume que há sempre uma degradação dos indicadores.

Regra	Descrição	Redução de Conforto
1	Assento não disponível	80
	Disponibilidade importante	
2	Assento não disponível	90
	Disponibilidade muito importante	
3	Assento disponível	10
	Disponibilidade importante	
4	Assento disponível	20
	Disponibilidade muito importante	
5	Assento desconfortável	65
	Conforto Importante	
6	Assento desconfortável	75
	Conforto muito importante	
7	Assento aceitável	30
	Conforto Importante	
8	Assento aceitável	30
	Conforto muito importante	

Tabela II. Regras de Defuzzificação

VI. Resultados

Os dados de testes propostos tinham objetivos de validar diferentes condições do cotidiano. No primeiro cenário, os dados representam uma viagem longa em que há assentos disponíveis e o conforto destes assentos é de baixa importância para o usuário. O resultado aponta níveis altos de conforto. Na Figura 8, é possível observar que com o passar do tempo, a percepção de conforto diminui. Isto se dá, pois os graus de importância das variáveis disponibilidade e conforto de assento crescem e influencia o processo de defuzzificação.

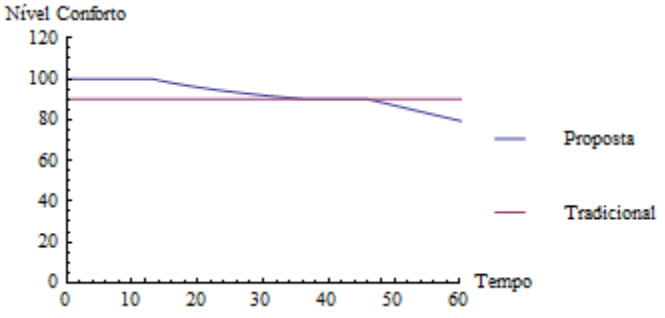


Fig. 8. Cenário de teste 1

No segundo cenário, uma viagem curta é executada em um ônibus com assentos desconfortáveis. Neste cenário o conforto dos assentos é mais importante e com o passar do tempo, a percepção de conforto se degrada rapidamente como apresentado na Figura 9.

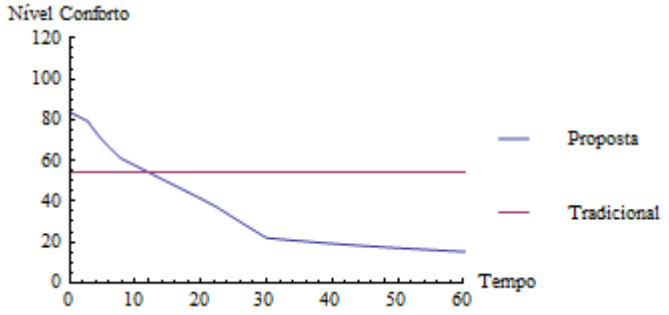


Fig. 9. Cenário de teste 2

O terceiro cenário representado pela Figura 10 reflete um cenário em que os graus de importância e a variável de conforto de assentos estão em condições médias. Os resultados são próximos, uma vez que, para as condições médias, a Tabela II de defuzzificação não tem grandes implicações.

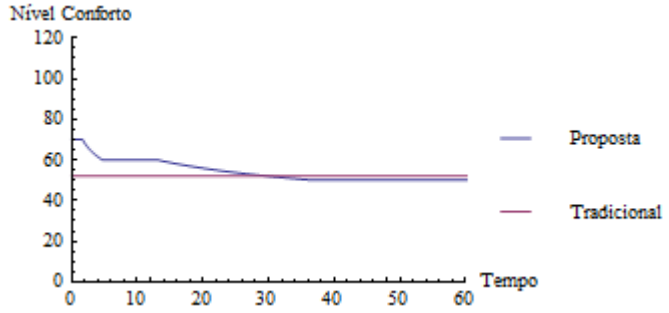


Fig. 10. Cenário de Teste 3

Por fim, o quarto cenário verificou o comportamento da avaliação de conforto em uma viagem de longa duração sem assentos disponíveis, como apresentado na Figura 11.

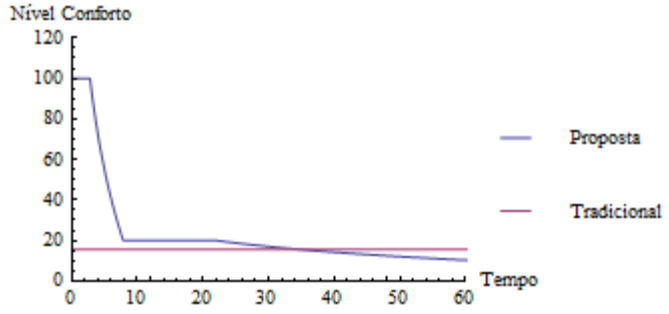


Fig. 11. Cenário de Teste 4

A forma de cálculo com dependência no tempo permite uma maior qualidade na avaliação da percepção de conforto de usuários. Os mesmos resultados poderiam ser obtidos se um conjunto extenso de regras de inferência e defuzzificação fossem aplicados. Com o mapeamento do comportamento, é possível obter resultados com um conjunto pequeno com oito regras.

VII. ADAPTATIVIDADE

A prova de conceito apresentada sugere que os resultados utilizando a proposta de Lógica Nebulosa dependente do tempo são melhores do que a alternativa em que o tempo é desconsiderado. No entanto, o resultado está diretamente relacionado à capacidade de representar adequadamente as funções que representam o conforto.

O conforto por sua vez, está sujeito a variações com base em experiências anteriores, meio em que o indivíduo está inserido, expectativas e outros fatores.

Uma representação válida pode ser observada na Figura 7, em que cada viagem é vista como um ciclo de um controle de malha fechada.

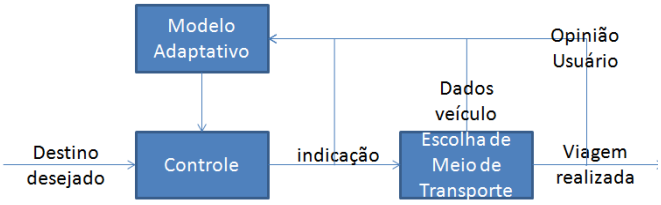


Fig. 07. Representação de viagem em malha fechada

Entende-se por esta representação que os dados do meio de transporte, assim como a opinião do usuário quanto à qualidade da viagem, podem influenciar os parâmetros que suportam a indicação do meio de transporte. Ou seja, os valores que determinam os limites dos graus de pertinência de cada propriedade que define o controle, assim como seus graus de importância, estão em constante ajuste para melhor avaliar as condições de transporte.

Ajustes em graus de pertinência de modelos nebulosos foram propostos em outros cenários, como em [19] em que os autores propõem um deslocamento das curvas que definem as variáveis linguísticas de variáveis nebulosas a partir da realimentação de saída de controles. Há ainda estudos que apontam que é possível ajustar os graus de pertinência, mesmo para os contextos em que há dependência no tempo, como em [20]. Cerrada, Aguilar, Colina e Titli apresentam uma forma de ajustar os parâmetros das definições nebulosas verificando os valores previstos e realizados na linha do tempo. O resultado é uma aproximação do modelo projetado e real, permitindo uma maior previsibilidade nos resultados.

VIII. CONCLUSÃO

Números nebulosos com dependência do tempo é uma boa opção para resolver problemas envolvendo variáveis subjetivas dinâmicas. A comparação entre o modelo tradicional e o modelo com dependência no tempo apresentou

diferenças de resultados que demonstram a capacidade do modelo com dependência de prever a situação em que o sistema se encontrará e assim poder tomar decisões mais assertivas.

A qualidade da solução está diretamente relacionada a capacidade do sistema aprender com cada evento (viagem) executada, definindo assim com maior qualidade os graus de importância de cada fator que definem o conforto.

O trabalho considerou um cenário simples envolvendo apenas dois fatores que compreendem o espectro que define conforto, mas vários estudos indicam que uma gama grande de fatores contribui com sua determinação. Estudos adicionais precisam ser executados para melhor representar o contexto das variáveis, e não só as condições instantâneas que surgem ao longo dos trajetos.

IX. Referências

[1] Cantwell M., Caulfield B., O'Mahony M.; "Examining the factors that impact public transport commuting satisfaction", Journal of Public Transportation, vol. 12, no. 2, pp. 1–21, 2009.

[2] Eboli L., Mazzulla G.; "A new customer satisfaction index for evaluating transit service quality", Journal of Public Transportation, vol. 12, no. 3, pp. 21–37, 2009.

[3] Kostakis A., I. Pandelis,; "Measuring the Customer Satisfaction in Public Transportation. An empirical study based in urban buses in the city of Larisa (Greece) The MUSA methodology", Proceedings of MIBES 2009 (Management of International Business and Economic Systems) Conference, pp. 260–275, 2009.

[4] Karlsson J., Larsson E.; "Passengers' Valuation of Quality in Public Transport with Focus on Comfort: A Study of Local and Regional Buses in the City of Gothenburg", Chalmers University of Technology 2010.

[5] Chappells H., Shove E.; "COMFORT: A review of philosophies and paradigms," University of Lancaster 2004, pp. 1–37.

[6] Yeh C.-H, Deng H., Chang Y.-H; "Fuzzy multicriteria analysis for performance evaluation of bus companies"; Eur. J. Oper. Res. vol. 126, 2000.

[7] W. Huang, L. Ding; "The Shortest Path Problem on a Fuzzy Time-Dependent Network"; IEEE Transactions on Communications, vol. 60, no. 11, pp. 3376–3385, Nov. 2012.

[8] Transport for London, <http://www.tfl.gov.uk/>, Accessed on September 04th, 2013.

[9] New York State Department of Transportation, <https://www.dot.ny.gov/divisions/policy-and-strategy/public-transportation>, Accessed on September 04th, 2013.

[10] CET - Companhia de Engenharia de Tráfego, <http://www.cetsp.com.br/>, Accessed on September 04th, 2013.

[11] Shashua, A., Gdalyahu Y., Hayun G.; "Pedestrian detection for driving assistance systems: single-frame classification and system level performance"; In Intelligent Vehicles Symposium, 2004.

[12] Cherng, S., Fang, C.; "Critical motion detection of nearby moving vehicles in a vision-based driver-assistance system"; IEEE Transactions on Intelligent Transportation Systems, vol. 10, no. 1, pp. 70–82, 2009.

[13] Jacob, R., Marathe M., Nagel, K.; "A Computational Study of Routing Algorithms for Realistic Transportation Networks", ACM Journal of Experimental Algorithms, 1998

[14] Barro, S., Bugarín, A. J., Cariñena, P., Díaz-Hermida, F., Mucientes, M.;"Fuzzy Temporal rule-based systems: new challenges"; XIV Congreso Español sobre Tecnologías y Lógica Fuzzy, pp. 17–19, 2008.

[15] Mucientes, M., Iglesias, R., Regueiro, C. V., Bugarín, A., Barro, S.; "A fuzzy temporal rule-based velocity controller for mobile robotics"; Fuzzy Sets and Systems vol. 134, pp. 83–99, 2003.

[16] The New York Times - When Businesses Can't Stop Asking, 'How Am I Doing?', <http://www.nytimes.com/2012/03/17/business/onslaught-of-surveys-is-fraying-customer-patience.html?scp=1&sq=surveys&st=cse&_r=0>, Accessed on September 04th, 2013.

[17] Tilahun, S. L., Ong, H. C.; "Bus timetabling as a fuzzy multiobjective optimization problem using preference based genetic algorithm"; Promet Traffic & Transportation, vol. 24, no. 3, 2012.

[18] Si, X. S., Hu C. H., Yang J. B.; "A new prediction model based on belief-rule-base for system's behavior prediction"; IEEE Transactions on Fuzzy Systems, 2011.

[19] Bakri, F.A., Mashor, S. M., Sharun, M. N., Saad, D.; "A Simple Adaptative Fuzzy Logic Controller base on Shifting of the Membership Function"; 2012 IEEE 8th International Colloquium on Signal Processing and its Applications

[20] Cerrada, A., Aguilar, J., Colina, E., Titli, A.; "Dynamical membership functions: an approach for adaptative fuzzy modeling"; Fuzzy Sets and Systems; 2003

Framework para Simulação e Verificação de Aplicações Adaptativas

F. M. B. Reis and A. R. Camolesi

***Abstract*— Este trabalho tem por objetivo apresentar o conceito e a implementação de um framework para o desenvolvimento de aplicações adaptativas que utilizam como base um Modelo Lógico desenvolvido com foco na representação de dispositivos adaptativos dirigidos por regras. Desta forma, facilita para que um especialista em dispositivos adaptativos possa estender um determinado dispositivo não adaptativo dirigido por regras em adaptativo. Sendo assim, ao final do projeto, busca-se obter uma ferramenta que permitirá auxiliar um especialista em seu projeto.**

***Keywords*—adaptativa, tecnologia adaptativa, framework adaptativo, simulação adaptativa**

1. INTRODUÇÃO

APLICAÇÕES complexas são caracterizadas por componentes e aspectos cuja estrutura e comportamento, comumente, podem modificar-se [1]. Tais aplicações possuem um comportamento inicial definido por um conjunto de ações que desempenham suas funções elementares e podem ter o seu comportamento modificado durante a execução para dar suporte a novas funcionalidades. Tais modificações são decorrentes dos estímulos de entrada a que são submetidos no sistema e/ou da ocorrência de suas ações internas.

Uma técnica utilizada para auxiliar os projetistas no projeto de aplicações com comportamento modificável é a tecnologia adaptativa [2]. A tecnologia adaptativa envolve um dispositivo não-adaptativo (subjacente) já existente em uma camada adaptativa que permite realizar mudanças no comportamento da aplicação definida [3]. É possível citar, por exemplo, trabalhos relacionados a reconhecedores sintáticos adaptativos [4], os *Statecharts* Adaptativos [5] - empregados na modelagem de sistemas reativos - e a modelagem de aplicações complexas com base no ISDL Adaptativo [6]. O desenvolvimento de tecnologia adaptativa aplicado a sistemas de dispositivos não adaptativos dirigidos por regras vem sendo pesquisada com totais preocupações a fim de que o usuário consiga gerenciar novos dispositivos adaptativos.

Camolesi propôs em [1] um gerador de ambientes (metambiente) que possibilita a geração automática de ambientes para o projeto de aplicações adaptativas. Tal gerador fundamenta-se nos conceitos de Tecnologia Adaptativa e permite a definição de dispositivos adaptativos dirigidos por regras [7].

No trabalho apresentado em [1] uma das etapas necessárias para o desenvolvimento do gerador de ambientes
é a construção de um *framework* para auxiliar na especificação de formas de operação para dispositivos adaptativos. Tal *framework* deve representar os elementos conceituais de operação de um dispositivo e permitir realizar a simulação e a verificação de aplicações especificadas com base em um formalismo adaptativo representado.

A. Problema

As ferramentas existentes para o projeto de aplicações são muito especificas e restritas a um ou alguns formalismos apenas e não contemplam o uso de dispositivos adaptativos. Quando um novo dispositivo adaptativo é projetado, novas ferramentas para especificação, simulação e verificação de aplicações devem ser construídas. O desenvolvimento de tais ferramentas é lento e acaba desestimulando a criação de novos dispositivos adaptativos. Este projeto teve o propósito de disponibilizar um framework para a definição da forma de operação de dispositivos adaptativos. O *framework* fornece aos especialistas em dispositivos adaptativos uma ferramenta para auxiliar na definição de novos dispositivos adaptativos de forma que ao especificarem um novo dispositivo já obtenham de forma automática ferramentas para a simulação e a verificação de aplicações produzidas com base no novo formalismo gerado.

B. Objetivos

Desenvolver e disponibilizar um *framework* para definição de forma de operação para dispositivos adaptativos. Tal *framework* possibilita depois de realizada à definição da forma de operação de um dispositivo a simulação e a verificação de aplicações especificadas com base no dispositivo adaptativo obtido.

C. Relevância

Com a concretização desse trabalho obteve-se uma ferramenta que permite a um especialista em certo dispositivo dirigido por regra realizar o mapeamento deste para uma metaestrutura que represente sua extensão adaptativa e a sua forma de operação. Além do auxílio aos especialistas em extensão de dispositivos adaptativos, a ferramenta também auxilia os projetistas na realização do trabalho de especificação e simulação de suas aplicações.

D. Estrutura do trabalho

Este trabalho encontra-se organizado da seguinte forma: inicialmente, no Capítulo 1 foi apresentada uma visão geral do trabalho, seus objetivos e as suas justificativas. Na sequência, o Capítulo 2 apresenta uma breve introdução dos conceitos de Tecnologia Adaptativa, de ferramentas para o projeto de aplicações que se utilizam deste conceito e descreve a arquitetura geral do Modelo Lógico utilizado para suportar a representação computacional de dispositivos adaptativos. Como base no Modelo Lógico foi realizado o desenvolvimento do Framework Adaptativo, descrito no

Capítulo 3. Por fim, no Capítulo 4 serão tecidas algumas conclusões e trabalhos futuros.

2. TECNOLOGIA ADAPTATIVA

O trabalho proposto fundamenta-se na Tecnologia Adaptativa, a qual permite que um dispositivo adaptativo seja capaz de se auto modificar, ou seja, seu comportamento mudar dinamicamente em tempo de execução e sem influência externa, isso quando ele detecta uma situação em que para ele não é mais possível dar continuidade a sua execução sem antes ele alterar seu comportamento.

A maior vantagem dos dispositivos adaptativos é a sua facilidade de uso, sua relativa simplicidade e o fato de ser muito rara a necessidade de existir uma descrição completamente especificada do comportamento do dispositivo para que ele possa ser utilizado. Em lugar disso, sua operação pode ser descrita de forma incremental, e seu comportamento, programado para se alterar dinamicamente em resposta aos estímulos de entrada recebidos. Adicionalmente, como o conjunto de regras que definem o seu comportamento também se altera ao longo da sua operação, pode-se dizer que o próprio dispositivo adaptativo programa de alguma forma a representação do conhecimento adquirido através de sua sequencia recebida de estímulos de entrada.

Dispositivos adaptativos podem ser empregados em uma grande variedade de situações, principalmente em segmentos complexos da solução de um problema, no qual sejam realizadas tomadas não triviais de decisão.

Para auxiliar o projeto de aplicações que use tecnologia adaptativa, faz-se necessária a utilização de um ambiente que integra um conjunto de ferramentas que dê suporte aos projetistas na realização de seu trabalho. A Fig. 1 [1] ilustra a arquitetura geral de tal ambiente, que é organizado em 3 (três) grupos de ferramentas: Ferramentas de Edição, Ferramentas de Análise e Ferramenta de Implementação.

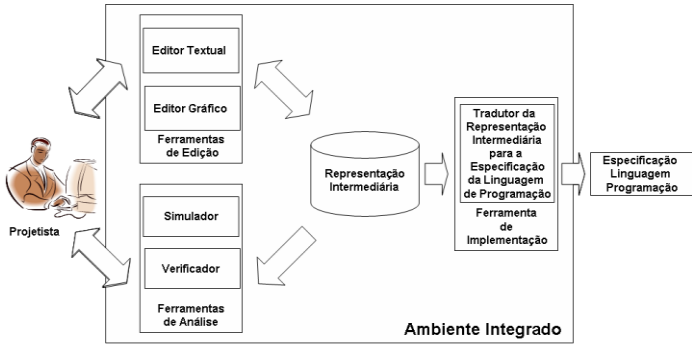


Figura 1. Ambiente para o projeto de aplicações usando Tecnologia Adaptativa.

Valendo-se das Ferramentas de Edição, um projetista de aplicações utilizando-se de um dispositivo adaptativo específico poderá realizar a especificação de sua aplicação com o auxilio de um editor de texto qualquer ou de um editor gráfico. Para possibilitar o intercâmbio das especificações produzidas, os editores devem gerar objetos no Modelo Lógico. Caso a especificação seja produzida em um editor textual, esta deverá ser compilada para transformar a codificação realizada no formato definido para o modelo lógico. Depois de realizada a especificação, o projetista de aplicações poderá utilizar as Ferramentas de Análise. Tais ferramentas utilizam a codificação da especificação (no formato do modelo lógico) como base e permitem a realização de uma análise do comportamento da aplicação adaptativa em desenvolvimento. Por fim, depois de especificada e analisada a representação de uma aplicação, o projetista pode se utilizar das Ferramentas de Produção de Representações Físicas e gerar uma representação de uma aplicação em um determinado padrão de linguagem de representação física e obter a aplicação desejada.

A. Modelo Lógico para Ambiente Adaptativo

O Modelo Lógico proposto em [1] é o núcleo central do Framework Adaptativo proposto. Neste contexto será descrito neste capítulo um resumo do modelo lógico para representação de dispositivos adaptativos e aplicações modeladas com base nestes dispositivos. O Modelo Lógico é dividido em quatro camadas:

- Camada de especificação de dispositivos;
- Camada de especificação de aplicações – não adaptativa;
- Camada de especificação de funções e ações adaptativas;
- Camada de representação de memória;

A definição do modelo em camada é fundamental para o desenvolvimento do trabalho, uma vez que permite separar a camada não adaptativa da camada adaptativa. Abaixo (Fig. 2) será mostrado o modelo lógico (diagrama de classe) completo.

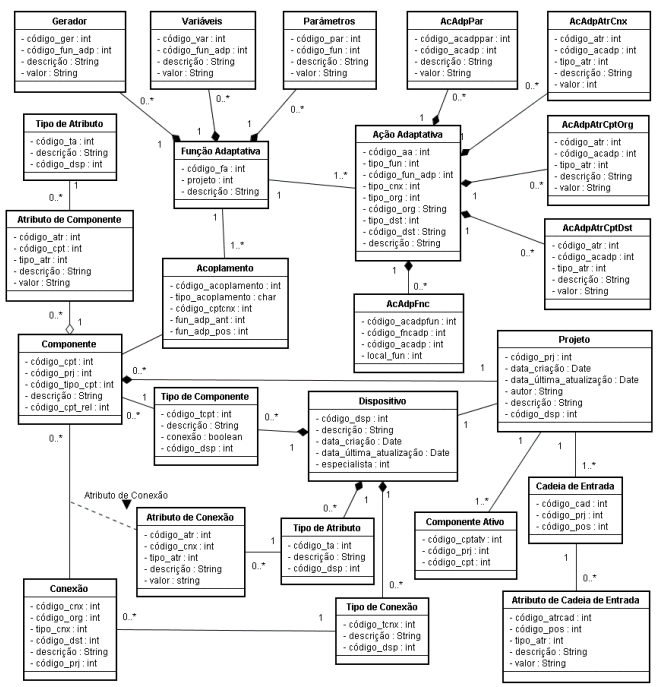


Figura 2. Modelo Lógico (Camolesi, 2007)

Conforme dito anteriormente o Modelo Lógico foi estruturado em camadas. A seguir serão apresentados os elementos que constituem cada camada e os seus respectivos

relacionamentos.

A1) Camada de Especificação de Dispositivos

Os elementos da Camada de Configuração de Dispositivos, também denominada na teoria, como Núcleo Subjacente [7], representa os elementos lógicos referentes à definição de um determinado dispositivo dirigido por regras não adaptativo. A Fig. 3 ilustra esta camada, na qual um especialista depois de estender os conceitos formais de um determinado dispositivo deve realizar um mapeamento da estrutura conceitual do referido dispositivo para a mesma. Ao instanciar objetos nesta estrutura o especialista define os elementos conceituais: tipos de componentes, tipos de conexões e tipos de atributos que definem a estrutura de dispositivo dirigido por regras não adaptativos.

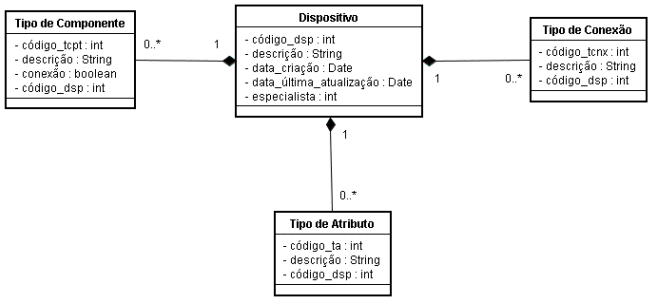


Figura 3. Camada de Configuração de Dispositivos.

A2) Camada de Especificação de Aplicações

A Camada de Especificação de Aplicações – Não adaptativa tem por objetivo representar os elementos referentes ao comportamento de uma determinada aplicação que está sendo modelada por certo projetista. Um projetista ao modelar a aplicação instancia objetos da camada de núcleo subjacente e define o comportamento da aplicação gerando novos objetos na camada de especificação de aplicações. Para cada elemento definido nesta camada são associados os seus tipos correspondentes à camada de núcleo subjacente e também é feita uma relação entre os elementos dos diversos conjuntos que representam esta camada. Esta camada, como foi dito anteriormente, representa o comportamento da aplicação e, consequentemente, a especificação não adaptativa de uma determinada aplicação. A Fig. 4 apresenta a organização estrutural da referida camada.

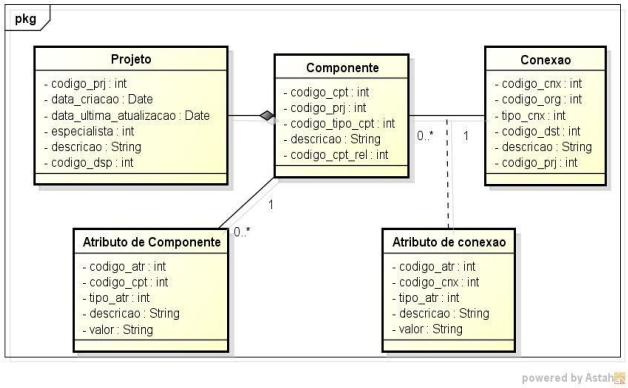


Figura 4. Camada de Especificação de Aplicações – Subjacente.

A3) Camada de Especificação de Funções e Ações Adaptativa

A Camada de Especificações de Funções e Ações Adaptativas tem por objetivo armazenar informações referentes às funções e ações adaptativas que são utilizadas em uma determinada aplicação. Os objetos desta camada são definidos com base nos objetos da camada de especificação, pois deve estar associado aos componentes e conexões de um determinado projeto. A utilização desta camada é de fundamental importância para a estrutura proposta neste trabalho.

Esta camada representa os elementos conceituais definidos em [7] para a concepção de um dispositivo adaptativo. Desta forma uma vez que um determinado projetista define um novo dispositivo na Camada de Configuração de Dispositivos (Fig. 5), estará desta forma estendendo o referido dispositivo para suportar tecnologia adaptativa.

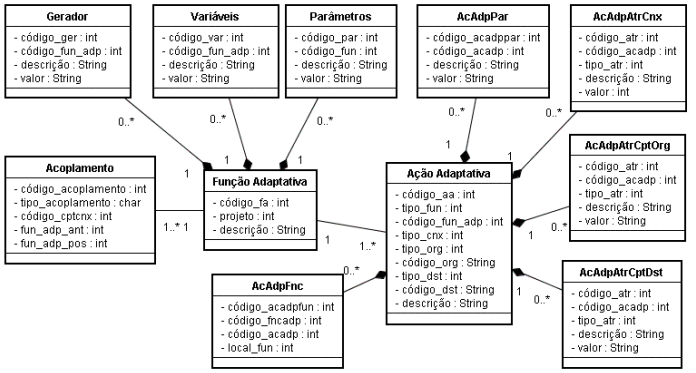


Figura 5. Camada de Especificação de Funções e Ações Adaptativa.

A4) Camada de Representação de Memória

Por fim, a Camada de Memória (Fig. 6) representa os estímulos de entrada (cadeia de entrada, estímulos externos oriundos de outros agentes, etc..) e o status (estados ou transições habilitadas a ocorrerem). Esta camada é de fundamental importância para a construção de ferramentas que darão suporte a simulações e verificações de aplicações projetadas utilizando-se de tecnologia adaptativa.

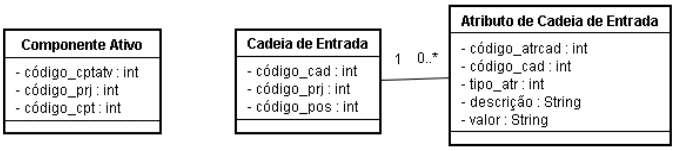


Figura 6. Camada de Representação de Memória.

3) Framework para Simulações de Tecnologias Adaptativas

Com o foco no estudo e aplicação dos conceitos de tecnologias adaptativas foi definido um estudo de caso que consiste no projeto e no desenvolvimento de um *framework* para geração de simulação de funções adaptativas. O *framework* concebido tem por objetivo de utilizar as definições no Modelo Lógico proposto. Tal framework tem por objetivo auxiliar os projetistas nas simulações de suas

aplicações. Um especialista ao iniciar uma nova simulação ganhará conhecimento suficiente para continuar com seu projeto, gerando grande base de conhecimento do funcionamento de sua aplicação em várias situações. Com isto obtém-se uma grande economia de tempo e dinheiro na concepção de suas tarefas.

A) Projeto do Framework para Simulações de Tecnologias Adaptativas

O *framework* proposto foi estruturado de forma a facilitar os estudos sobre tecnologias adaptativas, a partir do modelo lógico já existente. O *framework* gera uma simulação de um dispositivo mapeado pelo especialista com as definições básicas de seu projeto, porem ele terá de colocar as funções adaptativas com as informações necessárias para que permita simular diversos tipos de comportamento previsto. O framework foi desenvolvido com as regras de orientação a objetos, ou seja, foi estruturado em camadas.

Com base nos requisitos mínimos definidos para a concepção do framework o mesmo foi organizado em seis etapas que permitem desde a seleção do dispositivo ate a execução da simulação. As etapas são encadeadas com as informações pré-mapeadas pelo especialista. A Fig. 7 ilustra as etapas e o funcionamento do *framework* descrito.

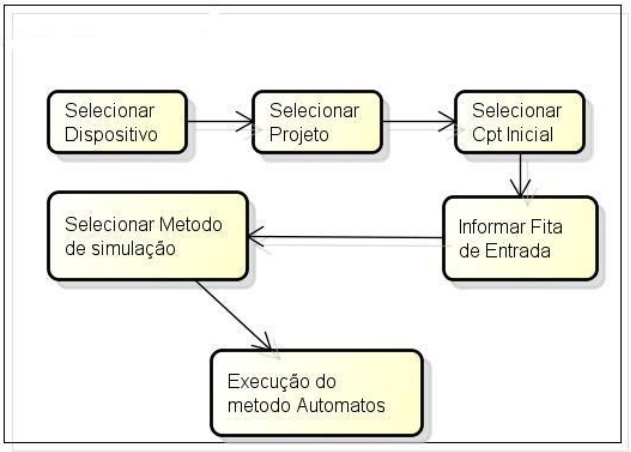


Figura 7. Etapas do Framework para Simulação Adaptativa.

Conforme apresentado na Fig. 7 o framework é definido em seis etapas, sendo:

- **Seleção de dispositivo**: etapa responsável por separar as informações desejadas para a simulação;
- **Seleção de projeto**: após a seleção do dispositivo, o framework ira mostrar os projetos relacionados ao dispositivo selecionado;
- **Seleção de componente inicial**: permite a seleção do componente inicial onde se iniciara a simulação do dispositivo;
- **Alimentação da fita de entrada**: A fita de entrada é a condição em que ocorrera a simulação do dispositivo, podendo ser informada a condição em que se quer testar o dispositivo selecionado;
- **Seleção do método de simulação**: Nesta etapa será apresentado ao especialista duas opções de execução, a primeira realiza toda a simulação sem pausas, executando a simulação do dispositivo e só mostrando o resultado final obtido. A segunda opção realiza a simulação com pausas. Neste tipo de simulação o framework mostrara cada etapa do processo e o estado da aplicação no momento da pausa.
- **Execução do método autômatos**: Neste local é onde ocorre a simulação do dispositivo selecionado com as informações da fita de entrada.

B) Implementação do framework

Na etapa inicial foram levantados os dados para a criação do framework, com base em alguns dispositivos já existentes, foi escolhido o autômato de estado finito determinístico adaptativo. A Figura 8 apresenta a interface do framework gerado para a simulação do dispositivo mencionado acima.

Figura 8. Interface para simulação de autômatos de estados finitos determinística.

No trecho de , apresentado abaixo, é responsável por carregar os dados dos dispositivos e os seus respectivos projetos. Neste local pode-se selecionar um dispositivo e os seus respectivos projetos existentes.

Código 1. Trecho de programa para seleção do dispositivo e projetos**.**

```
private void  cmbDsp_SelectedValueChanged(object sender, EventArgs e)
      {
         lbnDspId.Text = cmbDsp.SelectedValue.ToString();
         try            idDsp = int.Parse(lbnDspId.Text);
            oDsp = bllDsp.SelectCod(idDsp);
            lbnDspId.Text = oDsp.codigo_dsp.ToString();
            lbnDspDsc.Text = oDsp.descricao.ToString();
            lbnDspNome.Text = oDsp.especialista.ToString();
            dgvTpCpt.DataSource = bllDsp.SelectCod(idDsp).tipo_de_
               componente;
            dgvTpCnx.DataSource = bllDsp.SelectCod(idDsp).tipo_de_conexao;
            dgvTpAtr.DataSource = bllDsp.SelectCod(idDsp).tipo_de_atributo;
            cmbPrj.DataSource = bllPrj.SelectCodDsp(idDsp);
      }
private void cmbPrj_SelectedValueChanged(object sender, EventArgs e)
      {
         lbnPrjId.Text = cmbPrj.SelectedValue.ToString();
         try
         {
            idPrj = int.Parse(lbnPrjId.Text);
            oPrj = bllPrj.SelectCod(idPrj);
            lbnPrjId.Text = oPrj.codigo_prj.ToString();
            lbnDescPrj.Text = oPrj.descricao.ToString();
            lbnNomePrj.Text = oPrj.autor.ToString();
            dgvCnx.DataSource = bllPrj.SelectCod(idPrj).conexao;
            dgvCpt.DataSource = bllCpt.SelectCodPrj(idPrj);
            dgvCptIni.DataSource = bllPrj.SelectCod(idPrj).componente1;
            dgvFuncAdp.DataSource = bllFncAdp.SelectCodPrj(idPrj);
         }
```

Em seguida, após a seleção do dispositivo e a escolha de um mapeado anteriormente são carregados todos os componentes, atributos de componentes, conexões e atributos de conexões do respectivo projeto selecionado com as configurações do projeto escolhido pelo especialista.

Após esta etapa é apresentado na interface do framework uma janela com opções para o especialista definir qual o tipo de simulação (***Pausa*** com paradas para o especialista avaliar o estado da aplicação ou ***Contínua*** que realiza toda a simulação apresentando apenas a situação final) ele deseja (Fig. 9).

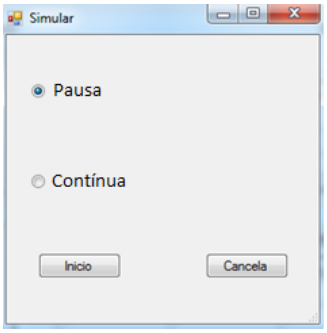


Figura 9. Interface para escolha do tipo de simulação**.**

Depois de escolhidas uma das opções (*Pausa* ou *Contínua*) e pressionado o botão "Início" ocorre a execução do trecho de código Código 2. O referido trecho é responsável por iniciar a simulação. Inicialmente são carregadas as informações escolhida pelo projetista e, na sequencia, é fechada a janela de escolha de tipo de simulação, iniciando a mesma.

Código 2. Método btnCancel_Click – Menu Simular**.**

```
private void btnCancel_Click(object sender, EventArgs e)
    {
       this.Close();
    }

    private void btnInicio_Click(object sender, EventArgs e)
    {
       if (semParada.Checked)
       {
          sele = 1;
       }
       else {
          if (comTemp.Checked)
          {
             sele = 2;
          }
          else { sele = 3; }
       }
       this.Close();
    }
```

Com esta etapa concluída, o framework reúne as informações que foram definidas pelo especialista (modelo da aplicação) para começar a simulação do comportamento da mesma. Tais informações serão armazenadas na classe autômato contendo identificação do componente inicial (idCpt), fita de entrada (fita), método de simulação (selecionado) e identificação do projeto (idPrj).

Com a primeira etapa concluída, o framework começará a simulação da aplicação modelada, desta forma ele executa as atividades de acordo com a execução definida no diagrama apresentado na Fig. 4.

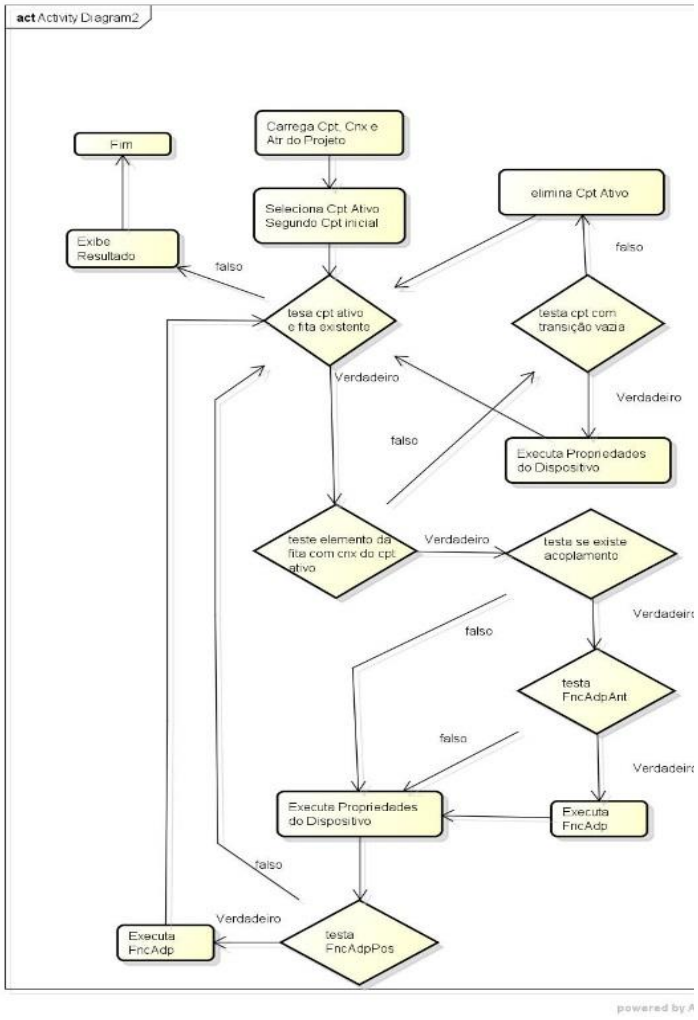


Figura 10. Diagrama de atividades do método automatos**.**

O diagrama descrito acima se refere aos trechos de código de 3 a 7, no trecho de Código 3 o método ***automatos*** carrega todos os componentes do dispositivo junto com suas conexões e atributos, além de posicionar a fita de entrada na posição inicial e selecionar o primeiro componente ativo.

Código **3**. Método automatos - Parte 1.

```
public void automatos()
    {
       par.Teste = "";
       idCptAux = idCpt;
       cont1 = cont2 = ca = 0;
       lCpt = new List<componente>();
       lCpt = cptDal.SelectCodPrj(idPrj);
       if (lCpt.Count != 0)
       {   for (int i = 0; i < lCpt.Count; i++)
             if (lCpt[i].codigo_cpt == idCptAux)
             {
                CptAtv = new componente_ativo();
                CptAtv.codigo_prj = idPrj;
                CptAtv.codigo_cpt = lCpt[i].codigo_cpt;
                oCpt = lCpt[i];

        }     }
       lCnx = new List<conexao>();
       lAtrCnx = new List<atributo_de_conexao>();
       lCnx = cnxDal.SelectCodPrj(CptAtv.codigo_prj.Value);
       for (int i = 0; i < lCnx.Count; i++)
       {  lAtrCnxAux = new List<atributo_de_conexao>();
          lAtrCnxAux = atrCnxDal.SelectCodCnx
             (lCnx[i].codigo_cnx);
          for (int j = 0; j < lAtrCnxAux.Count; j++)
             lAtrCnx.Add(lAtrCnxAux[j]);
```

O Código 4, ilustra a parte 2, do método automatos, na qual o comando "*while*" realiza uma iteração responsável por fazer a execução das ações necessários para que o dispositivo execute todas as suas tarefas. No fim do *while* após o teste é verificado para a posição da fita com as conexões do dispositivo ativo se existe uma transição para o estado corrente com o respectivo símbolo selecionado.

Código 4. Método automatos - Parte 2.

```
while ((CptAtv != null) && (fita.Length !=0))
        {  par.BusAtrCnx = lAtrCnx;
           par.LCpt = lCpt;
           par.Lcnx = lCnx;
           lCnxAux = new List<conexao>();
           for (int i = 0; i < lCnx.Count; i++)
           {   if (lCnx[i].codigo_org == CptAtv.codigo_cpt) {
               lCnxAux.Add(lCnx[i]);
            }}
           lAtrCnxAux = new List<atributo_de_conexao>();
           for (int i = 0; i < lCnxAux.Count; i++) {
              for (int j = 0; j < lAtrCnx.Count; j++) {
                 if (lCnxAux[i].codigo_cnx == lAtrCnx[j].codigo_cnx) {
                    lAtrCnxAux.Add(lAtrCnx[j]);
            } }              }
           menuSeleciona();
           bool teste = false;
           atrCnx = new atributo_de_conexao();
           if (fita.Length > ca)
           {for (int i = 0; i < lAtrCnxAux.Count; i++)
              { if (lAtrCnxAux[i].valor.Equals
              ((fita.ElementAt(ca)).ToString()))
                 {  teste = true;
                    atrCnx = lAtrCnxAux[i];
            }} }
```

Caso o teste de verificação de transição com símbolo vigente seja verdadeiro, o trecho de Código 5 é executado. Este trecho tem a função de verificar se existe algum acoplamento para a conexão ativa, caso exista um acoplamento, o código testa se existe alguma função adaptativa anterior para ser executada. Na existência da função adaptativa anterior, está é executada e a seguir o método executa os passos necessários para o dispositivo descrito anteriormente e, por fim, carrega o novo componente ativo para o próximo ciclo.

Código 5. Método automatos - Parte 3.

```
if (teste)
          {
            lAcp = new List<acoplamento>();
            lAcp = acpDal.SelectCodCpt(atrCnx.codigo_cnx.Value);
            oAcp = new acoplamento();
            if (lAcp.Count != 0)
               oAcp = lAcp[0];
            if ((oAcp != null) && (oAcp.fun_adp_ant != 0)
            && (oAcp.fun_adp_ant != null))
               funcAdp(oAcp.fun_adp_ant.Value);
            ca++;
            CptAtv = null;
            for (int i = 0; i < lCnxAux.Count; i++)
            {
               if (lCnxAux[i].codigo_cnx == atrCnx.codigo_cnx)
               {  CptAtv = new componente_ativo();
                  CptAtv.codigo_cpt = lCnxAux[i].codigo_dst;
                  CptAtv.codigo_prj = idPrj;
                  for (int j = 0; j < lCpt.Count; j++)
                  { if (lCpt[j].codigo_cpt == CptAtv.codigo_cpt)
                        oCpt = lCpt[j];
                  }
               }
            }
```

O trecho de Código 6 ilustra o teste do acoplamento para função adaptativa posterior que ocorre após a execução das tarefas elementares do dispositivo. Na existência de uma função adaptativa posterior, ela é executada. No caso da busca de conexões com símbolo corrente resultar falso, ou seja, não existem conexões com símbolo definido para o respectivo estado, é realizado o teste para verificação das transições espontâneas "ε". Caso também não exista transições vazias o método sai do *while* e finaliza a execução da simulação, retornando erro.

Código 6. Método automatos - Parte 4.

```
if ((oAcp != null) && (oAcp.fun_adp_pos != 0)
&& (oAcp.fun_adp_pos != null))
            funcAdp(oAcp.fun_adp_pos.Value);
            else {
               for (int i = 0; i < lAtrCnxAux.Count; i++)
                  if (lAtrCnxAux[i].valor.Equals("e"))
                  {  teste = true;
                     atrCnx = lAtrCnxAux[i];
                  }
               if (teste)
                  CptAtv = null;
                  for (int i = 0; i < lCnxAux.Count; i++)
                     if (lCnxAux[i].codigo_cnx == atrCnx.codigo_cnx)
                     {
                        CptAtv = new componente_ativo();
                        CptAtv.codigo_cpt = lCnxAux[i].codigo_dst;
                        CptAtv.codigo_prj = idPrj;
                        for (int j = 0; j < lCpt.Count; j++)
                          if (lCpt[j].codigo_cpt == CptAtv.codigo_cpt)
                             oCpt = lCpt[j];
               }
```

No caso do trecho de código abaixo, a função testa a fita de entrada para verificar se foi feito todas os passos para o dispositivo junto com o teste para verificar se o componente é de estado final. Caso todos os testes sejam satisfeitos a simulação "Aceita" a cadeia de entrada, caso contrário, a cadeia de entrada é "Rejeita" e ocorre o fim da simulação.

Código 7. Método automatos - Parte 5.

```
else {
                     CptAtv = null;
                  }

               }
            }
            if (ca == (fita.Length) && oCpt.codigo_tipo_cpt == 15)
            {
               par.Teste = "Aceita";
            }
            else {
               par.Teste = "Rejeita";
            }
            par.ShowDialog();

         }
```

A seguir é representado o trecho de código da função adaptativa, que pode ser chamada ate duas vezes no trecho de Código 4 (Função Adaptativa Anterior e Função Adaptativa Posterior). Incialmente o código testa a função para selecionar e separa todas as ações adaptativas relacionadas a função em execução, após a seleção, o código testa qual tipo de ação que será executada.

Código 8. Método funcAdp.

```
public void funcAdp(int cod) {
        funAdp = new funcao_adaptativa();
        lacAdp = new List<acao_adaptativa>();
        funAdp = funAdpDal.SelectCod(cod);
        lacAdp = acAdpDal.SelectCodFncAdp(funAdp.codigo_fa);
        int i = 0;
        while (i < lacAdp.Count)
        {  int caseSwitch = lacAdp[i].tipo_fun.Value;
           switch (caseSwitch)
           {
              case 1:
                 busca(lacAdp[i].codigo_aa);
                 break;
              case 2:
                 remoção();
                 break;
              case 3:
                 adicao(lacAdp[i]);
                 break;
           }
        i++;
}       }
```

Os conceitos de tecnologia adaptativa apresentados por Neto, 2001 e mapeados para o modelo lógico de Camolesi, 2007 foram utilizados para o desenvolvimento da execução de cada ação adaptativa. Sendo assim, cada ação adaptativa possui um atributo que indica o tipo de ação que vai ser executado: 1 - que representam a busca, 2 - remoção e 3 - adição.

Na ocorrência de uma ação de busca, o framework procura pela conexão compatível com a conexão da função adaptativa que foi mapeada junto com o dispositivo. Se a ação for de remoção, o framework removera a conexão selecionada na busca, e caso a ação for de adição, o framework será responsável por adicionar novas regras a aplicação que está sendo simulada. O Código 9, apresentado a seguir, ilustra a execução das ações adaptativas representadas no método busca.

Código 9. Método busca.

```
public void busca(int cod) {
        lbusAtrCnx = new List<atributo_de_conexao>();
        latraa = new List<AcAdpAtrCnx>();
        latraa = atraaDal.SelectCodAcAdp(cod);
        for(int i=0;i<latraa.Count;i++){
           for (int j = 0; j < lAtrCnx.Count; j++)
           {
              if (latraa[i].valor == lAtrCnx[j].valor)
              {
                 lbusAtrCnx.Add(lAtrCnx[j]);
              }
           }
        }
        lbusCnx = new List<conexao>();
        for (int i = 0; i < lCnx.Count; i++) {
           for (int j = 0; j < lbusAtrCnx.Count; j++) {
              if (lCnx[i].codigo_cnx == lbusAtrCnx[j].codigo_cnx) {
                 lbusCnx.Add(lCnx[i]);
              }
           }
        }
     }
```

O trecho de Código 10 ilustra a funcionalidade definida para as ações adaptativas de remoção. A execução desta ação de remoção utiliza a conexão encontrada pela ação de busca e a remove das conexões existentes referentes ao projeto selecionado. Além disso, é armazenado os códigos de identificação dos componentes que a conexão fazia parte.

Código 10. Método remoção.

```
public void remoção()
     {
        for (int i = 0; i < lCnx.Count; i++)
        {
           for (int j = 0; j < lbusCnx.Count; j++)
           {
              if (lCnx[i].codigo_cnx == lbusCnx[j].codigo_cnx)
              {
                 var1 = new variaveis();
                 var2 = new variaveis();
                 var1.descricao = lCnx[i].codigo_org.ToString();
                 var2.descricao = lCnx[i].codigo_dst.ToString();
                 lCnx.RemoveAt(i);
              }
           }
        }
        for (int i = 0; i < lAtrCnx.Count; i++)
        {
           for (int j = 0; j < lbusAtrCnx.Count; j++)
           {
              if (lAtrCnx[i].codigo_atr == lbusAtrCnx[j].codigo_atr)
              {
                 lAtrCnx.RemoveAt(i);
              }
           }
        }

     }
```

A ação de remoção só ira apagar o que foi encontrado pela ação de busca, já a função de adição tem algumas condições, por exemplo: caso exista algum argumento dela com "*" e logo após "?", a função ira criar um componente e em seguida criar a conexão necessária para que este componente seja utilizado, no Código 11, inicialmente, a ação de adição faz um teste para saber se vai utilizar o código do componente anterior para fazer a conexão do código do componente que será criado pelo código. A criação de uma nova conexão segue a seguinte ordem, componente, conexão e atributo de conexão.

Código 11. Método adicao - Parte 1.

```
public void adicao(acao_adaptativa acAdp) {
        if (acAdp.codigo_org.Contains("?") &&
acAdp.codigo_dst.Contains("*"))
        {
           cont1++;
           componente cptAux = new componente();
           cptAux.codigo_cpt = lCpt.Last().codigo_cpt + 1;
           cptAux.codigo_prj = idPrj;
           cptAux.codigo_tipo_cpt = 16;
           cptAux.descricao = "k" + cont1;
           cptAux.codigo_cpt_rel = 0;
           lCpt.Add(cptAux);
           conexao cnxAux = new conexao();
           cnxAux.codigo_cnx = lCnx.Last().codigo_cnx + 1;
           cnxAux.codigo_org = int.Parse(var1.descricao);
           cnxAux.tipo_cnx = 9;
           cnxAux.codigo_dst = cptAux.codigo_cpt;
           cnxAux.codigo_prj = idPrj;
           cnxAux.descricao = acAdp.descricao;
           lCnx.Add(cnxAux);
           atributo_de_conexao atrAux = new atributo_de_conexao();
           AcAdpAtrCnx atraaAux = new AcAdpAtrCnx();
           latraa = new List<AcAdpAtrCnx>();
           latraa = atraaDal.SelectCodAcAdp(acAdp.codigo_aa);
           for (int i = 0; i < latraa.Count; i++)
           {
              if (latraa[i].codigo_acadp == acAdp.codigo_aa)
              {
                 atraaAux = latraa[i];
              }
           }
```

O Código 12 será executado quando a função adaptativa tem que criar um componente e criar a conexão com um componente existente, a ordem da criação e a mesma do código anterior.

Código 12. Método adicao - Parte 2.

```
atrAux.codigo_atr = lAtrCnx.Last().codigo_atr + 1;
            atrAux.codigo_cnx = cnxAux.codigo_cnx;
            atrAux.tipo_atr = 9;
            atrAux.descricao = atraaAux.descricao;
            atrAux.valor = atraaAux.valor;
            lAtrCnx.Add(atrAux);
            var1.descricao = cptAux.codigo_cpt.ToString();
         }
         else {
            if (acAdp.codigo_org.Contains("*") &&
acAdp.codigo_dst.Contains("?"))
            {  cont2++;
               componente cptAux = new componente();
               cptAux.codigo_cpt = lCpt.Last().codigo_cpt + 1;
               cptAux.codigo_prj = idPrj;
               cptAux.codigo_tipo_cpt = 16;
               cptAux.descricao = "k1 " + cont2;
               cptAux.codigo_cpt_rel = 0;
               lCpt.Add(cptAux);
               conexao cnxAux = new conexao();
               cnxAux.codigo_cnx = lCnx.Last().codigo_cnx + 1;
               cnxAux.codigo_dst = int.Parse(var2.descricao);
               cnxAux.tipo_cnx = 9;
               cnxAux.codigo_org = cptAux.codigo_cpt;
               cnxAux.codigo_prj = idPrj;
               cnxAux.descricao = acAdp.descricao;
               lCnx.Add(cnxAux);
               atributo_de_conexao atrAux = new atributo_de_conexao();
               AcAdpAtrCnx atraaAux = new AcAdpAtrCnx();
               latraa = new List<AcAdpAtrCnx>();
               latraa = atraaDal.SelectCodAcAdp(acAdp.codigo_aa);
```

Caso o teste retorne vazio, será criada uma conexão com transição vazia, já explicada anteriormente.

Código 13. Metodo adicao - Parte 3.

```
for (int i = 0; i < latraa.Count; i++)
            {  if (latraa[i].codigo_acadp == acAdp.codigo_aa)
                     atraaAux = latraa[i];
               atrAux.codigo_atr = lAtrCnx.Last().codigo_atr + 1;
               atrAux.codigo_cnx = cnxAux.codigo_cnx;
               atrAux.tipo_atr = 9;
               atrAux.descricao = atraaAux.descricao;
               atrAux.valor = atraaAux.valor;
               lAtrCnx.Add(atrAux);
               var2.descricao = cptAux.codigo_cpt.ToString();
            }
            else {
               conexao cnxAux = new conexao();
               cnxAux.codigo_cnx = lCnx.Last().codigo_cnx + 1;
               cnxAux.codigo_org = int.Parse(var1.descricao);
               cnxAux.tipo_cnx = 9;
               cnxAux.codigo_dst = int.Parse(var2.descricao);
               cnxAux.codigo_prj = idPrj;
               cnxAux.descricao = acAdp.descricao;
               lCnx.Add(cnxAux);
               atributo_de_conexao atrAux = new
atributo_de_conexao();
               AcAdpAtrCnx atraaAux = new AcAdpAtrCnx();
               latraa = new List<AcAdpAtrCnx>();
               latraa = atraaDal.SelectCodAcAdp(acAdp.codigo_aa);
               for (int i = 0; i < latraa.Count; i++)
                  if (latraa[i].codigo_acadp == acAdp.codigo_aa)
                     atraaAux = latraa[i];
               atrAux.codigo_atr = lAtrCnx.Last().codigo_atr + 1;
               atrAux.codigo_cnx = cnxAux.codigo_cnx;
               atrAux.tipo_atr = 9;
               atrAux.descricao = atraaAux.descricao;
               atrAux.valor = atraaAux.valor;
               lAtrCnx.Add(atrAux);
            }} }
```

4) CONCLUSÃO

Neste trabalho foram apresentados os conceitos de Tecnologia Adaptativa, de um modelo lógico para representação para dispositivos adaptativos definidos por regras e a construção de um framework para auxiliar a construção de ferramentas para simulação de aplicações definidas com dispositivos adaptativos.

O trabalho realizado permitiu validar o modelo lógico apresentado em Camolesi, 2007. Tal modelo demonstrou-se compatível com os conceitos teóricos dos dispositivos subjacentes estudados e permitiu verificar que o modelo lógico também é consistente para a representação computacional de aplicações adaptativas modeladas com base em dispositivos adaptativos dirigidos por regras.

O framework desenvolvido foi instanciado para a representação de autômatos finitos adaptativos e demonstrou-se eficiente para a sua realização. Com base na ferramenta de simulação obtida foram realizados testes e verificou-se que tal ferramenta pode auxiliar os projetistas de aplicações no desenvolvimento de suas tarefas.

A arquitetura do framework desenvolvido permite que um especialista em programação e em um determinado dispositivo adaptativo obtenha de forma rápida uma ferramenta para simulação de aplicações.

Fundamentado na pesquisa desenvolvida, várias vertentes para trabalhos futuros podem ser identificadas. Como possíveis trabalhos futuros, pode-se citar:

- Utilização do framework desenvolvido para instanciação de outros dispositivos adaptativos, com o intuito de avaliar a consistência da implementação realizada;
- Desenvolvimento de uma ferramenta textual, acoplada ao framework para auxiliar a definição de novos dispositivos adaptativos e o projeto de aplicações com base nos dispositivos obtidos;
- Desenvolvimento de uma ferramenta gráfica para auxiliar o projeto de aplicações fundamentadas em dispositivos adaptativos;
- Melhorias no framework para definição de uma linguagem que permita a um leigo em programação definir por meio de uma ferramenta textual o comportamento de um dispositivo adaptativo, sem ter necessidade de utilizar-se de recursos de programação, ou seja, realizar a implementação de uma máquina virtual adaptativa.
- Construção de uma ferramenta que permita a tradução automática das especificações descritas no modelo lógico para uma linguagem de programação.

REFERÊNCIAS

[1] CAMOLESI, A.R. *Proposta de um gerador de ambientes para a modelagem de aplicações usando Tecnologia Adaptativa*. Tese de Doutorado, Escola Politécnica, Universidade de São Paulo, São Paulo, 2007.

[2] NETO, J. J. *Contribuições à metodologia de construção de compiladores*. Tese de Livre Docência, USP, São Paulo, 1993.

[3] PISTORI, H. *Tecnologia Adaptativa em Engenharia de Computação:* Estado da Arte e Aplicações. Tese de Doutorado, USP, São Paulo, 2003.

[4] NETO, J.J.; KOMATSU, W. Compilador de Gramáticas Descritas na Notação de Wirth Modificada. *Anais EPUSP* - Engenharia de Eletricidade - série B, vol. 1, pp. 477-517, São Paulo, 1988.

[5] ALMEIDA, J.R. STAD. *Uma ferramenta para representação e simulação de sistemas através de statecharts adaptativos*. Tese de Doutorado, Escola Politécnica, Universidade de São Paulo, São Paulo, 1995.

[6] CAMOLESI, A.R. e NETO, J.J. Modelagem Adaptativa de Aplicções Complexas. XXX *Conferência Latinoamericana de Informática* - CLEI'04. Arequipa - Peru, Setiembre 27 - Octubre 1, 2004a

[7] NETO, J. J. *Adaptive Rule-Driven Devices - General Formulation and Case Study*. Lecture Notes in Computer Science. Watson, B.W. and Wood, D. (Eds.): Implementation and Application of Automata 6th International Conference, CIAA 2001, Springer-Verlag, Vol.2494, pp. 234-250, Pretoria, South Africa, July 23-25, 2001.

Parte II

Transcrição da mesa redonda

A transcrição da mesa redonda está em fase de revisão. Em breve, esta seção será substituída pelo texto adequado.