Memórias do WTA 2014

VIII 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
2014

Ficha catalográfica

Workshop de Tecnologia Adaptativa (8: 2014: São Paulo)
Memórias do WTA 2014. – São Paulo; EPUSP, 2014. 137p.

ISBN 978-85-86686-76-4

ISBN 978-85-86686-61-0

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 oitava edição do Workshop de Tecnologia Adaptativa realizou-se em São Paulo, Brasil, nos dias 06 e 07 de Fevereiro de 2014, 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

– Almir Rogério Camolesi (Assis, SP, Brasil)

– Amaury Antônio de Castro Junior (Ponta Porã, MS, Brasil)

– André Riyuiti Hirakawa (São Paulo, SP, Brasil)

– Angela Hum Tchemra (São Paulo, SP, Brasil)

– Aparecido Freitas (São Paulo, SP, Brasil)

– Carlos Eduardo Cugnasca (São Paulo, SP, Brasil)

– Claudia Maria Del Pilar Zapata (Lima, Peru)

– Danilo de Jesus da Silva Bellini (São Paulo, SP, Brasil)

– Fabiana Soares Santana (São Paulo, SP, Brasil)

– Hemerson Pistori (Campo Grande, MS, Brasil)

– João Eduardo Kögler Junior (São Paulo, SP, Brasil)

– Jorge Rady de Almeida Junior (São Paulo, SP, Brasil)

– José Maria Novaes Santos (São Paulo, SP, Brasil)

– Júlio Luz (São Paulo, SP, Brasil)

– Márcio Lobo Netto (São Paulo, SP, Brasil)

– Marco Túlio Carvalho de Andrade (São Paulo, SP, Brasil)

– Marcos Pereira Barretto (São Paulo, SP, Brasil)

– Marcus Vinicius Midena Ramos (Juazeiro, BA, Brasil)

– Renata Luiza Stange (Guarapuava, PR, Brasil)

– Ricardo Luis de Azevedo da Rocha (São Paulo, SP, Brasil)

– Rosalia Edith Caya Carhuanina (São Paulo, SP, Brasil)

– Salvador Ramos Bernardino Silva (Manaus, AM, Brasil)

– Sidney Viana (São Paulo, SP, Brasil)

– Wagner José Dizeró (Lins, SP, Brasil)

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 2014

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 2014, o Oitavo Workshop de Tecnologia Adaptativa, realizado em São Paulo nos dias 06 e 07 de Fevereiro de 2014. A exemplo da edição de 2013, 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 2014 – Oitavo Workshop de Tecnologia Adaptativa, realizado na Escola Politécnica da Universidade de São Paulo nos dias 06 e 07 de Fevereiro de 2014.

Esta edição contou com 21 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.

Participação especial

A oitava edição do Workshop de Tecnologia Adaptativa contou com a participação especial do prof. Marcus Vinícius Midena Ramos, da Universidade Federal do Vale do São Francisco, apresentando um tutorial entitulado *Desenvolvimento e verificação mecanizada de programas, auxiliados por assistentes de prova interativos – fundamentação teórica e aplicações* nos dois dias do evento.

Esta publicação

Estas memórias espelham o conteúdo apresentado no WTA 2014. A exemplo do que foi feito no ano anterior, todo o material referente aos trabalhos apresentados no evento estará acessível no portal do WTA 2014, 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 2015.

Agradecimentos

Às instituições que apoiaram o WTA 2014, 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

Apoio operacional

- Daniel Costa Ferreira
- Edson de Souza
- Fernando Takashi
- 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 oitava edição do Workshop de Tecnologia Adaptativa.

São Paulo, 07 de Fevereiro de 2014

João José Neto
Coordenador geral

Sumário

Lista de autores ix

I Submissões **1**

1 Descoberta automática de atributos salientes para obtenção de macro-ações
Rafael Lemes Beirigo, Valdinei Freire da Silva, Anna Helena Reali Costa 2

2 Linguístico: Usando Tecnologia Adaptativa para a Construção de Um Reconhecedor Gramatical
Ana Teresa Contier, Djalma Padovani, João José Neto 8

3 Aplicação da Tecnologia Adaptativa e Reflexão Computacional por meio da programação reflexiva em Java
Diego Augusto Passareli, Almir Rogério Camolesi, Diomara Martins Reigato Barros . . . 21

4 Um Metamodelo para Programas Adaptativos Utilizando SBMM
Sergio Canovas, Carlos Cugnasca 27

5 Towards an Adaptive Internet of Things Architecture
Leonardo Barreto Campos, Carlos Cugnasca 35

6 Software adaptável ao cálculo de Satisfazibilidade Probabilística
Celso Barros, Fabio Gagliardi Cozman 40

7 Análisis de métodos para el reconocimiento de patrones en ECG
Olaf Gawron, Nahuel González, Fernando Lage 42

8 Semi-Global Alignment of Lines and Columns for Robust Table Extraction from periodic PDF Documents
Jorge Kinoshita 46

9 AWARE – um ambiente para estudo e visualização de processos adaptativos
Ian Silva Oliveira, João Eduardo Kögler Junior, Emilio Del Moral Hernandez 53

10 epsilon-Greedy Adaptativo
Alexandre dos Santos Mignon, Ricardo Luis de Azevedo da Rocha 57

11 Predição de Tempo de Compilação e Tamanho de Código em Máquinas Virtuais
Jorge Augusto Sabaliauskas, Ricardo Luis de Azevedo da Rocha 63

12 Processos Markovianos de Decisão com heurísticas, junção de abordagens backward e forward para transferência de conhecimento baseados em políticas
Rodrigo Garcia Pontes, Valdinei Freire da Silva 68

13 Classificação automática de frutas por análise de imagem – o caso da manga Tommy Atkins
Edmar Candeia Gurjão, Mário Eduardo Rangel Moreira Cavalcanti Mata, Maria Elita Martins Duarte 77

14 Adaptive Product Classification for Online Marketplace Based on Semantic Analysis
Tacio Filipe Vasconcelos de Medeiros, Fabio Gagliardi Cozman 92

15 Oportunidades de uso da Adaptatividade em um SPLN para criação de atividades de leitura
José Lopes Moreira Filho, Zilda Maria Zapparoli 95

16 Adaptive Programming in Cyan
José de Oliveira Guimarães, Paulo Roberto Massa Cereda 99

17 An Adaptive Approach for Error-Recovery in Structured Pushdown Automata
João José Neto, Hemerson Pistori, Amaury Antônio de Castro Junior, Marcelo Rafael Borth 105

18 Swarm Robotics: comportamento Adaptativo Aplicado ao problema de dispersão de pássaros em áreas agrícolas
Alexsandro Procópio da Silva, Reginaldo Inojosa da Silva Filho, Fabrício Augusto Rodrigues 111

19 Persistência em dispositivos adaptativos
Paulo Roberto Massa Cereda, João José Neto 120

20 Proposal of modification of the DJ Library for implementing Adaptive Technology
Rosalia Edith Caya Carhuanina, João José Neto 126

21 Estudo preliminar sobre a aplicação da adaptatividade em linguagens de programação imperativas
Diego Queiroz, Ricardo Luis de Azevedo da Rocha 134

II Transcrição da mesa redonda **a**

Lista de autores

B

Barros, Celso 40
Barros, Diomara Martins Reigato 21
Beirigo, Rafael Lemes 2
Borth, Marcelo Rafael 105

C

Camolesi, Almir Rogério 21
Campos, Leonardo Barreto 35
Canovas, Sergio 27
Carhuanina, Rosalia Edith Caya 126
Castro Junior, Amaury Antônio de 105
Cereda, Paulo Roberto Massa 99, 120
Contier, Ana Teresa 8
Costa, Anna Helena Reali 2
Cozman, Fabio Gagliardi 40, 92
Cugnasca, Carlos 27, 35

D

Duarte, Maria Elita Martins 77

G

Gawron, Olaf 42
González, Nahuel 42
Guimarães, José de Oliveira 99
Gurjão, Edmar Candeia 77

H

Hernandez, Emilio Del Moral 53

J

José Neto, João 8, 105, 120, 126

K

Kögler Junior, João Eduardo 53
Kinoshita, Jorge 46

L

Lage, Fernando 42

M

Mata, Mário Eduardo Rangel Moreira Cavalcanti 77
Medeiros, Tacio Filipe Vasconcelos de . . 92
Mignon, Alexandre dos Santos 57
Moreira Filho, José Lopes 95

O

Oliveira, Ian Silva 53

P

Padovani, Djalma 8
Passareli, Diego Augusto 21
Pistori, Hemerson 105
Pontes, Rodrigo Garcia 68

Q

Queiroz, Diego 134

R

Rocha, Ricardo Luis de Azevedo da 57, 63, 134
Rodrigues, Fabrício Augusto 111

S

Sabaliauskas, Jorge Augusto 63
Silva Filho, Reginaldo Inojosa da 111
Silva, Alexsandro Procópio da 111
Silva, Valdinei Freire da 2, 68

Z

Zapparoli, Zilda Maria 95

Parte I

Submissões

Descoberta automática de atributos salientes para obtenção de macro-ações

Rafael Lemes Beirigo[*], Valdinei Freire da Silva[†], Anna Helena Reali Costa[‡]

***Abstract*— A obtenção de soluções para problemas de decisão sequenciais pode ser realizada através de métodos de Aprendizado por Reforço. Entretanto devido à necessidade de interação através de tentativa e erro, esses métodos podem apresentar extrema lentidão. Para contornar esse problema são propostas técnicas de Transferência de Conhecimento que permitem explorar a estrutura hierárquica dos problemas através de abstração temporal na forma de macro-ações. Métodos de descoberta de macro-ações que utilizam eventos salientes apresentaram resultados promissores, entretanto demandam a definição de atributos salientes por parte do projetista, o que é indesejável. O presente trabalho propõe métodos de descoberta automática de atributos salientes no intuito de facilitar a descoberta automática de macro-ações.**

***Keywords*— Aprendizado por Reforço, Transferência de Conhecimento, Macro-ações, Eventos salientes, Atributos salientes.**

I. Introdução

SOLUÇÕES para problemas de decisão sequenciais sem conhecimento *a priori* podem ser obtidas com sucesso através de técnicas de Aprendizado por Reforço (em inglês, *Reinfocement Learning* (RL)) (Sutton e Barto, 1998). Entretanto, as técnicas de RL necessitam que o agente interaja com o ambiente, no intuito de descobrir as ações que, ao serem executadas, maximizam o valor esperado de recompensas recebidas. Como esse processo se dá por tentativa e erro, a aplicação das técnicas de RL na solução de problemas pode apresentar extrema lentidão.

Buscando contornar esse problema, foram propostas na literatura técnicas de Transferência de Conhecimento (Taylor e Stone, 2009; Torrey e Shavlik, 2009; Pan e Yang, 2010; Bergamo *et al.*, 2011; Da Silva e Costa, 2011; Da Silva, Selvatici e Costa, 2011; Matos, Bergamo, Da Silva e Costa, 2011; Matos, Bergamo, Da Silva, Cozman, *et al.*, 2011; Beirigo *et al.*, 2012; Da Silva, Pereira e Costa, 2012; Lazaric, 2012; Koga *et al.*, 2013), em que o agente acumula conhecimento na solução de tarefas de forma a poder utilizá-las em tarefas futuras e dessa forma alcançar um melhor desempenho.

Um elemento que possui papel fundamental no proceso de transferência é a representação do conhecimento adquirido, pois está relacionada a todas as etapas do reuso. Uma forma de representação frequentemente utilizada na literatura que explora a estrutura do domínio é a representação de *habilidades* (em inglês, *skills*) (Gullapalli, Franklin e Benbrahim, 1994; Thrun e Schwartz, 1995). A ideia por trás do conceito é a de que é possível identificar habilidades presentes em uma tarefa que são comuns a outras, permitindo ao agente aprender e solucionar cada uma dessas sub-tarefas de forma independente e posteriormente transferir de forma modular o conhecimento adquirido entre problemas que compartilhem sub-tarefas.

Uma forma usual de formalizar o conceito de habilidades em aplicações de RL tem sido usar o conceito de *macro-ações* (em inglês, *options*) (Sutton, Precup e Singh, 1999), cuja utilização possui a vantagem de permitir a aplicação do arcabouço teórico investigado na literatura de RL. Dessa forma, os algoritmos de aprendizado e provas de convergência se aplicam também quando da utilização de macro-ações. Macro-ações exploram a estrutura hierárquica dos problemas por efetuarem uma abstração temporal da solução.

Embora a abstração por macro-ações possibilite um impacto positivo no desempenho do agente (Chentanez, Barto e Singh, 2004; Konidaris, Scheidwasser e Barto, 2012), a definição das mesmas fica a cargo do projetista, o que pode implicar em uma sobrecarga em tempo de projeto.

Para solucionar esse problema, métodos foram propostos nos quais a descoberta de macro-ações se dá através da utilização de *eventos salientes* (em inglês, *salient events* (SE)) (Chentanez, Barto e Singh, 2004), que correspondem a *sub-goals* presentes na tarefa. Um evento saliente ocorre quando há uma variação *drástica* no valor de um ou mais atributos presentes na descrição fatorada de estados. Esses atributos utilizados para verificar se houve uma alteração drástica são denominados *atributos salientes* (em inglês, *salient features* (SF)) que serão utilizados para detecção dos SE.

Apesar dos resultados positivos apresentados no referido trabalho em relação à melhora de desempenho, o método exige que o projetista defina previamente os SF, o que pode representar uma indesejável sobrecarga de trabalho para o projetista, além de um conhecimento prévio do domínio. Dessa forma, seria interessante investigar métodos que descubram os SF de forma automática.

O presente trabalho propõe dois métodos de descoberta automática dos SF, que avaliam a utilização seletiva dos atributos da descrição fatorada de estados do problema e o consequente impacto no valor da aplicação de uma política.

Nas próximas seções são apresentados a conceituação

[*] R. L. Beirigo, Escola Politécnica, Universidade de São Paulo, São Paulo, SP, Brasil, rafaelbeirigo@usp.br.
[†] V. F. da Silva, EACH, Universidade de São Paulo, São Paulo, SP, Brasil, valdinei.freire@usp.br.
[‡] A. H. Reali Costa, Escola Politécnica, Universidade de São Paulo, São Paulo, SP, Brasil, anna.reali@usp.br.

teórica do trabalho e o detalhamento dos métodos envolvidos na comparação.

II. Background Teórico

A. MDP e Aprendizado por Reforço

Um problema de decisão sequencial modelado por um *Markovian Decision Process* (MDP) (Puterman, 2009) é definido pela quádrupla $\langle S, A, T, R \rangle$, na qual S compreende um conjunto de estados do ambiente, A contém as ações possíveis de serem executadas em cada estado, ambos finitos, $T: S \times A \times S \rightarrow [0,1]$ e $R: S \times A \times S \rightarrow \Re$ representam, respectivamente, a probabilidade de transição e a recompensa recebida ao, partindo de um estado $s_i \in S$ alcançar o estado $s_j \in S$ executando-se a ação $a \in A$.

Uma política $\pi: S \times A \rightarrow [0,1]$ associa a cada par (s, a) uma probabilidade de se executar a ação a estando estado s, sendo o valor de uma política π obtido através da função $V^{\pi}: S \rightarrow \Re$, onde $V^{\pi}(s)$ fornece o valor esperado de recompensas futuras ao se seguir a política π partindo do estado $s \in S$.

A solução para um MDP consiste em uma política ótima π^*, tal que $V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s), \forall s \in S$, ou seja, π^* maximiza o valor esperado de recompensas recebidas para todos os estados aí presentes.

Métodos de RL (Sutton e Barto, 1998) podem ser utilizados para solucionar MDPs. Ao utilizar uma técnica de RL, um agente interage com o ambiente, realizando uma ação para cada estado e observando a recompensa recebida. O intuito do agente é o de maximizar o valor esperado de recompensas recebidas.

Para isso, o agente necessita experimentar ações em estados ainda não visitados através de exploração, mas deve também guiar sua trajetória utilizando o conhecimento já adquirido sobre os estados visitados anteriormente.

Dessa forma, há um compromisso entre a *descoberta* de informação sobre o ambiente e a *utilização* do conhecimento adquirido até o momento, muitas vezes referido como o compromisso *exploração*/*explotação*.

Apesar de utilizados frequentemente e com sucesso para a solução de MDPs, podem apresentar lentidão, por dependerem de interação do agente com o ambiente baseada em tentativa e erro.

Esse fato promoveu a investigação de métodos de transferência de conhecimento (Taylor e Stone, 2009), cuja ideia básica é a de que o agente enfrenta problemas que possuem elementos semelhantes em sua estrutura, que podem ser aprendidos e armazenados de forma conveniente para reuso posterior.

Assim, o agente acumula conhecimento ao solucionar uma sequência de tarefas e transfere esse conhecimento para novas tarefas que, apesar de diferentes, apresentam semelhança com as já resolvidas previamente.

No entanto, para que a transferência seja possível, o conhecimento armazenado pelo agente durante seu aprendizado necessita de uma estrutura que permita esse reuso. A codificação desse conhecimento possui então papel fundamental na aplicação das técnicas de transferência de conhecimento, pois interferem diretamente tanto na possibilidade quanto na eficiência da reutilização do conhecimento adquirido.

Dessa forma, a *representação* do conhecimento desempenha papel fundamental no processo de transferência, pois está relacionado tanto ao conhecimento que está sendo adquirido quanto ao armazenado, estabelecendo a ponte entre a aquisição e o reuso.

B. Transferência de Habilidades

A utilização de *habilidades* é uma forma de representação que procura explorar a estrutura hierárquica presente em tarefas nas quais o objetivo pode ser desmembrado em objetivos intermediários, ou sub-tarefas. A intuição por trás do conceito se baseia no fato de que a solução de uma tarefa possui elementos repetidos na sua execução, ou seja, atividades que precisam ser desempenhadas mais de uma vez durante a solução do problema.

Cada uma dessas atividades pode ser vista como uma habilidade, sendo que, do ponto de vista do agente, é interessante aprender essa habilidade e armazená-la para posterior reuso. Isso porque, uma vez aprendidas, as habilidades podem ser utilizadas posteriormente com mais rapidez, pois não será necessário que o agente as aprenda novamente, o que tornará o aprendizado mais rápido e eficiente.

Por outro lado, uma habilidade pode ser enxergada como uma *sub-tarefa* de aprendizado, a qual pode ser resolvida separadamente, gerando-se soluções para cada sub-tarefa, que são soluções parciais da tarefa principal. Como há uma repetição das mesmas na tarefa a ser resolvida, é possível que habilidades sejam transferidas entre tarefas (Sutton, Precup e Singh, 1999).

Assim, a identificação e aprendizado de habilidades poderia auxiliar o agente tanto no aprendizado de uma tarefa individual, quanto no reuso de habilidades já aprendidas previamente em tarefas que necessitem delas.

Para formalizar o conceito de habilidades pode-se utilizar a estrutura de *macro-ações* (Sutton, Precup e Singh, 1999), que correspondem a políticas parciais que podem ser utilizadas pelo agente na solução das sub-tarefas. Uma vantagem significativa da utilização do formalismo de macro-ações é a possibilidade de utilização do arcabouço teórico desenvolvido na literatura de RL (Sutton, Precup e Singh, 1999).

Formalmente, uma macro-ação é definida por três elementos: uma *política* $\pi: S \times A \rightarrow [0,1]$, uma função que define a *condição de terminação da macro-ação* $\beta: S \rightarrow [0,1]$ e um *conjunto-iniciação* $\Im \subseteq S$, que define a disponibilidade da macro-ação somente para os estados $s \in \Im$.

Assim, a solução de um MDP, inicialmente definida pela obtenção de uma política ótima para o problema, pode ser desmembrada na obtenção de *políticas parciais* referentes às macro-ações aí presentes. Uma vantagem adicional é obtida pela estruturação do problema de forma *hierárquica*, onde o agente ganha a possibilidade de executar não somente ações *atômicas*, ou seja, as presentes em A, mas também as macro-ações, o que pode implicar em um aumento de eficiência no aprendizado (Sutton, Precup e Singh, 1999; Chentanez, Barto e Singh, 2004; Konidaris e Barto, 2007; Konidaris e Barreto, 2009; Konidaris, Scheidwasser e Barto, 2012).

C. *Descoberta de Habilidades*

Apesar do impacto positivo no desempenho de aprendizado do agente implicado pela utilização de habilidades (Chentanez, Barto e Singh, 2004; Konidaris, Scheidwasser e Barto, 2012), o processo de definição das mesmas pode representar uma sobrecarga ao projetista. Dessa forma, foram propostos métodos de descoberta automática de habilidades.

No trabalho presente em (Chentanez, Barto e Singh, 2004) um agente interage com o ambiente enquanto busca descobrir eventos que indiquem a necessidade da construção de uma nova *macro-ação*. Esses eventos são aí denominados eventos salientes (SE) e ocorrem quando, durante a transição de um estado a outro, há uma alteração significativa nos valores de determinados atributos da descrição fatorada de estados, sendo esses atributos denominados atributos salientes (SF), dados *a priori* pelo projestista.

O método de descoberta de macro-ações a partir de SE proposto em (Chentanez, Barto e Singh, 2004) ocorre da seguinte maneira: durante sua interação com o ambiente, a cada transição de estado s_t para s_{t+1}, o agente verifica se, ao atingir o estado s_{t+1} alguma das SF sofreu uma modificação significativa (que ultrapassa um limiar pré-definido). Em caso afirmativo, o agente cria uma macro-ação cujo estado terminal é s_{t+1}, ou seja, uma macro-ação cujo objetivo é alcançar s_{t+1}.

Além disso, analisa-se o conjunto de macro-ações definidas para verificar se alguma delas está disponível para o estado s_{t+1}, mas não para o estado s_t. Caso isso ocorra, o estado s_t é incluído na macro-ação, tendo como resultado a construção incremental das macro-ações com base nas transições que ocorrem durante a interação do agente com o ambiente. Dessa forma, as macro-ações são criadas levando em consideração variações nas SF pré-definidas para o domínio.

O trabalho apresenta resultados positivos na melhora de desempenho do aprendizado do agente, sugerindo que a abordagem seja promissora. Dessa forma, o presente trabalho propõe métodos de descoberta automática de SF, de forma a facilitar o trabalho de aprendizado e transferência de conhecimento entre tarefas de um determinado domínio.

III. Descoberta de Atributos Salientes

No trabalho de (Chentanez, Barto e Singh, 2004), para que o processo de descoberta de macro-ações seja possível, é necessário que os SFs sejam definidos previamente. Ao invés de colocar a responsabilidade dessa tarefa a cargo do projetista, neste trabalho são propostas duas maneiras de descoberta automática das mesmas.

Dada uma descrição fatorada dos estados do ambiente, tem-se que cada estado é definido por um conjunto de atributos $\mathcal{F} = \{f_1, f_2, \dots, f_n\}$. Dessa forma, o problema de Descoberta de Atributos Salientes consiste em encontrar um subconjuto $\mathcal{F}_{salient} \subset \mathcal{F}$ que contém os SFs relevantes para a classe de problemas em questão.

A hipótese analisada neste trabalho é a de que os atributos presentes em $\mathcal{F}_{salient}$ possuem uma relevância significativa na descrição do problema. Dessa forma, sua remoção da descrição dos estados do ambiente implicaria em uma alteração drástica no valor de aplicação da política ótima para o problema.

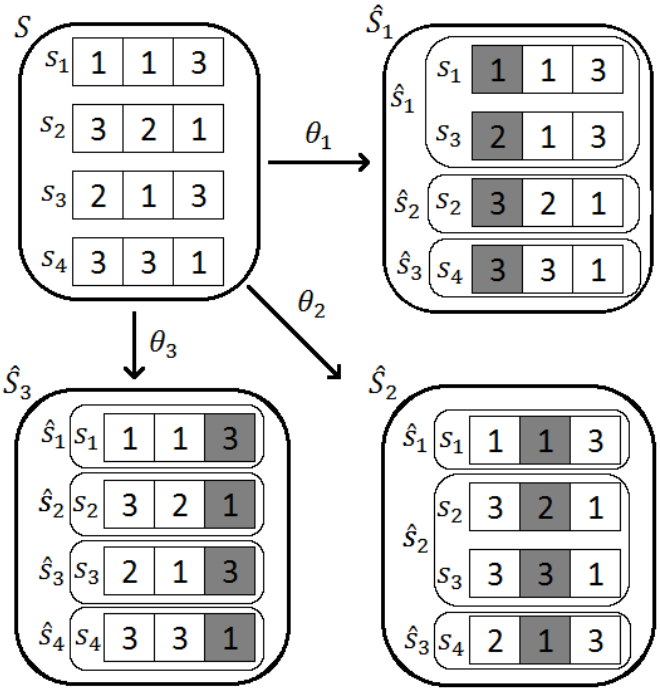
O método proposto neste trabalho para encontrar $\mathcal{F}_{salient}$ consiste em um processo iterativo no qual atributos são removidos da descrição de estados e em seguida avalia-se o impacto dessa remoção no valor da aplicação da política.

O processo de *remoção de um atributo* f_j é comum aos dois métodos propostos e se dá através da função $\theta_j: S \to \hat{S}_j$, que particiona o conjunto S, sendo que cada subconjunto gerado nesse particionamento contém estados que possuem valores idênticos para todos os atributos, desconsiderando-se o valor do atributo removido.

Portanto, dado um conjunto $\mathcal{F} = \{f_1, f_2, \dots, f_n\}$ contendo n atributos, tem-se um total n funções θ_j definidas, cada uma correspondendo à remoção do atributo $j \in \{1, 2, \dots, n\}$.

Para fins de ilustração, na Figura 1 temos um exemplo onde cada estado é definido por três atributos (representados na figura por uma quadrícula), ou seja, $\mathcal{F} = \{f_1, f_2, f_3\}$, sendo que cada atributo pode assumir um valor entre um e três. Como a cardinalidade de $\mathcal{F}$, $|\mathcal{F}| = 3$, é possível definir três funções θ_j, $j = 1, 2, 3$, cada uma responsável pela remoção de um atributo, representado por um sombreamento na quadrícula.

Figura 1 – Aplicação da função $\theta_j: S \to \hat{S}_j$, que realiza o agrupamento dos estados de S, f elementos de $\hat{S}_j$ através da remoção de atributos pela função θ_j.



Ao se aplicar θ_1, gera-se o conjunto $\hat{S}_1$, onde o atributo f_1 foi removido da representação de estados, resultando em uma nova representação onde somente são considerados os valores dos atributos f_2 e f_3. Quando isso ocorre, estados que possuem valores idênticos para f_2 e f_3 são agrupados em um mesmo estado de $\hat{S}_1$.

No exemplo da figura, a remoção do atributo f_1 resultou do agrupamento dos estados s_1 e s_3 em um novo estado $\hat{s}_1$. Isso ocorreu eles possuem os mesmos valores para os atributos restantes, f_2 e f_3. Entretanto, como os estados s_2 e s_4 possuem valores diferentes para os atributos restantes, eles

não puderam ser agrupados. Dessa forma, para cada um deles foi gerado um novo estado unitário na nova representação, sendo que o processo para as demais funções θ ocorre de maneira similar.

A remoção de um atributo implica em um impacto no valor da aplicação da política, pois a descrição de estados utilizada sofreu alteração, implicando em modificações na função-valor. Isso ocorre porque a política que está sendo avaliada pode possuir ações diferentes para cada um dos estados agrupados, e cada estado agrupado pode possuir um valor diferente.

Dessa forma, a remoção de cada um dos atributos possui um impacto no valor de aplicação da política, que pode então ser avaliado. A hipótese investigada neste trabalho é de que os atributos salientes, por sua importância na descrição fatorada de estados, são aqueles cuja remoção da descrição implicará no maior impacto de aplicação da política.

Para o processo de *avaliação*, são propostas duas abordagens, sendo que cada uma delas analisa seletivamente o impacto da remoção de cada um dos atributos, construindo incrementalmente o conjunto $\mathcal{F}_{saliente}$.

A. Avaliação através de abstração de política

A primeira abordagem proposta inicialmente gera uma política definida nos estados presentes em $\hat{S}_j$, gerados por θ_j. Inicialmente agrupam-se os estados em S de acordo com θ_j, sendo então definida uma política $\hat{\pi}_j: \hat{S}_j \times A \rightarrow [0,1]$, levando em conta uma distribuição de probabilidade proporcional ao valor de cada estado agrupado e a respectiva ação, isto é,

$$\hat{\pi}_j(\hat{s}_j, a) = \frac{\sum_{s|\theta_j(s)=\hat{s}_j} \pi(s,a) \cdot C^{\pi}(s)}{\sum_{s|\theta_j(s)=\hat{s}_j} C^{\pi}(s)}.$$

A função $C^{\pi}: S \rightarrow [0,1,2,\dots]$ fornece a quantidade de vezes que um estado $s \in S$ foi visitado durante a aplicação de uma política π. Dessa forma, o valor de $C^{\pi}(s)$, inicialmente nulo para todo $s \in S$, é incrementado de uma unidade a cada vez que o agente atinge estado s durante a aplicação de π.

O impacto $\mathcal{I}_j^{\pi}$ de remover o atributo f_j é dado pelo valor de aplicação da política $\hat{\pi}_j$, isto é, $\mathcal{I}_j^{\pi} = V^{\hat{\pi}_j}$. A definição de $V^{\hat{\pi}_j}$ decorre diretamente da aplicação da política $\hat{\pi}_j$ no problema. Assim, partindo de um estado inicial s_0, o agente segue a ação indicada pela política $\hat{\pi}_j$ para cada estado atingido.

B. Avaliação através de abstração de função-valor

A segunda abordagem proposta considera o impacto $\mathcal{I}_j^{\pi}$ da remoção do atributo f_j para representar a função valor $\hat{V}_j^{\pi}$, onde

$$\mathcal{I}_j^{\pi} = \sum_{s \in S} \left[\hat{V}_j^{\pi}\left(\theta_j(s)\right) - V^{\pi}(s) \right]^2$$

e $\hat{V}_j^{\pi}: \hat{S}_j \rightarrow R$, definida por:

$$\hat{V}_j^{\pi}(\hat{s}_j) = \frac{\sum_{s \in \hat{s}_j} V^{\pi}(s) \cdot C^{\pi}(s)}{\sum_{s \in \hat{s}_j} C^{\pi}(s)}.$$

C. Seleção dos atributos salientes

De posse dos valores de impacto de cada possível remoção de atributo, o processo de seleção dos atributos mais relevantes consiste então em escolher os atributos cuja remoção implicou no maior impacto. Os atributos assim selecionados são então incluídos no conjunto $\mathcal{F}_{saliente}$, similar ao desenvolvido em (Bogdan e Da Silva, 2013).

A limitação do processo de inclusão de atributos em $\mathcal{F}_{saliente}$ pode ser dada pelo estabelecimento prévio da quantidade máxima de atributos para o domínio em questão. Outra forma de limitação possível se daria através da definição de um valor de limiar para o impacto $\mathcal{I}_j^{\pi}$. Nesse caso, quando a variação for menor que o limiar estabelecido, o processo de inclusão para, sendo então gerado o conjunto $\mathcal{F}_{saliente}$. Ainda, pode-se combinar os dois conceitos em uma função que minimiza tanto o impacto $\mathcal{I}_j^{\pi}$, como o tamanho do conjunto $\mathcal{F}_{saliente}$ resultante.

IV. RL como Tecnologia Adaptativa

As técnicas de RL são apresentadas em um conjunto massivo de trabalhos presentes na literatura, possuindo destaque o trabalho (Sutton e Barto, 1998), onde os autores apresentam os fundamentos teóricos e algoritmos relacionados ao controle ótimo adaptativo. Dado que, na aplicação desses algoritmos, o agente busca adequar as funções internas que compreendem o aprendizado adquirido até o momento (na forma de uma função-valor, política ou macro-ação), sua adaptação ao ambiente tende a aumentar à medida em que sua interação com o mesmo lhe permite adquirir conhecimento. Sendo assim, pode-se arguir que as técnicas de Aprendizado por Reforço possuem em sua essência um núcleo de *adaptatividade* do agente em relação ao ambiente à medida em que aprende. Em um trabalho relacionado, onde há um maior destaque para o caráter adaptativo de RL (Sutton, Barto e Williams, 1992), os autores apresentam a idéia de uma forma mais específica, com destaque ao caráter adaptativo do Aprendizado por Reforço.

Essencialmente, as técnicas de Transferência de Conhecimento buscam estender a adaptação do agente a ambientes ainda desconhecidos, através de técnicas que compreendem uma vasta gama de tópicos (Taylor e Stone, 2009; Torrey e Shavlik, 2009; Pan e Yang, 2010). O trabalho apresentado em (Cao *et al.*, 2010) propõe o algoritmo *Adaptive Transfer learning algorithm based on Gaussian Processes*, em que busca-se aumentar o desempenho do agente através de uma análise de similaridade entre as tarefas envolvidas na transferência.

O presente trabalho é baseado em Transferência de Conhecimento através da adaptação de algoritmos de Aprendizado por Reforço aplicados à descoberta automática de macro-ações. Dessa forma, o foco essencial do trabalho é a busca de uma adaptação do agente ao ambiente de aprendizado através da modificação de sua estrutura interna de acordo com a interação que ocorre entre o agente aprendiz e o ambiente em que ele se encontra.

V. Discussão

Apesar da eficiência na aplicabilidade de métodos de aprendizado por reforço, há o problema de lentidão apresentado na solução de problemas mais complexos,

próximos do mundo real, o que impulsionou a investigação de métodos que buscam acelerar o processo de aprendizado.

A identificação de sub-tarefas presentes em uma tarefa a ser resolvida pode permitir que o agente aprenda a resolver cada uma das sub-tarefas de forma independente, o que consiste em um processo relativamente rápido, se comparado à solução da tarefa em si, também possibilitando o aprendizado concorrente de sub-tarefas.

Apesar de resultados positivos e promissores presentes na literatura na utilização de macro-ações para aumento da eficiência de aprendizado, o processo de definição das mesmas representa um empecilho, por ficar a cargo do projetista.

Alternativas de descoberta automática de macro-ações foram propostas na literatura para facilitar o processo de transferência de conhecimento. O presente trabalho se baseia na descoberta através de eventos salientes, que apresentou resultados positivos em relação ao aumento de eficiência de aprendizado.

Apesar desses resultados se apresentarem promissores, a utilização do método baseado em SEs necessita da definição de atributos salientes na descrição fatorada de estados, que, por sua vez, representam sobrecarga de trabalho ao projetista.

A contribuição do presente trabalho é a de propôr dois métodos de descoberta automática de atributos salientes, que por sua vez definem habilidades potenciais. A ideia comum aos dois métodos propostos neste trabalho é a de que os atributos salientes possuem importância significativa na descrição dos estados. A importância individual de cada atributo é medida pelo impacto que sua remoção tem no valor da aplicação de uma política definida no espaço de estados resultante após a remoção do atributo.

Após a avaliação individual de cada atributo, uma ordenação dos mesmos, de forma a selecionar os atributos cuja remoção causa maior impacto no valor de aplicação da política pode ser obtida.

Assim, os atributos selecionados através desse método devem possuir uma importância significativa na descrição dos estados. Esses atributos são então utilizados para compôr o conjunto de atributos salientes para o conjunto de problemas analisados.

A utilização do método proposto tem como resultado esperado um aumento de eficiência na aplicação de métodos de Aprendizado por Reforço para solução de problemas mais complexos através da facilitação da descoberta de macro-ações, cuja utilização já foi demonstrada na Transferência de Conhecimento entre problemas.

Referências

BEIRIGO, R. L. et al. Avaliacao de Politicas Abstratas na Transferencia de Conhecimento em Navegacao Robotica. **Revista de Sistemas e Computa{\c{c}}{\~a}o-RSC,** v. 2, n. 2, 2012.

BERGAMO, Y. P. et al. Accelerating reinforcement learning by reusing abstract policies. VIII Encontro Nacional de Intelig{\^e}ncia Artificial, 2011. p.596-606.

BOGDAN, K. O. M.; DA SILVA, V. F. Forward and Backward Feature Selection in Gradient-Based MDP Algorithms. In: (Ed.). **Advances in Artificial Intelligence**: Springer, 2013. p.383-394.

CAO, B. et al. Adaptive Transfer Learning. AAAI, 2010.

CHENTANEZ, N.; BARTO, A. G.; SINGH, S. P. Intrinsically motivated reinforcement learning. Advances in neural information processing systems, 2004. p.1281-1288.

DA SILVA, V. F.; COSTA, A. H. R. A geometric approach to find nondominated policies to imprecise reward MDPs. In: (Ed.). **Machine Learning and Knowledge Discovery in Databases**: Springer Berlin Heidelberg, 2011. p.439-454.

DA SILVA, V. F.; PEREIRA, F. A.; COSTA, A. H. R. Finding Memoryless Probabilistic Relational Policies for Inter-task Reuse. In: (Ed.). **Advances in Computational Intelligence**: Springer Berlin Heidelberg, 2012. p.107-116.

DA SILVA, V. F.; SELVATICI, A. H.; COSTA, A. H. R. Navigation towards a goal position: from reactive to generalised learned control. **Journal of Physics: Conference Series,** v. 285, p. 012025, 2011.

GULLAPALLI, V.; FRANKLIN, J. A.; BENBRAHIM, H. Acquiring robot skills via reinforcement learning. **Control Systems, IEEE,** v. 14, n. 1, p. 13-24, 1994.

KOGA, M. L. et al. Speeding-up reinforcement learning through abstraction and transfer learning. Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, 2013. p.119-126.

KONIDARIS, G.; BARRETO, A. S. Skill discovery in continuous reinforcement learning domains using skill chaining. Advances in Neural Information Processing Systems, 2009. p.1015-1023.

KONIDARIS, G.; BARTO, A. G. Building Portable Options: Skill Transfer in Reinforcement Learning. IJCAI, 2007. p.895-900.

KONIDARIS, G.; SCHEIDWASSER, I.; BARTO, A. Transfer in reinforcement learning via shared features. **The Journal of Machine Learning Research**, p. 1333-1371, 2012.

LAZARIC, A. Transfer in Reinforcement Learning: A Framework and a Survey. In: (Ed.). **Reinforcement Learning**: Springer, 2012. p.143-173.

MATOS, T. et al. Stochastic Abstract Policies for Knowledge Transfer in Robotic Navigation Tasks. **Advances in Artificial Intelligence**, p. 454-465, 2011.

MATOS, T. et al. Simultaneous abstract and concrete reinforcement learning. **Proc. of the 9th Symposium on Abstraction, Reformulation and Approximation**, 2011.

PAN, S. J.; YANG, Q. A survey on transfer learning. **Knowledge and Data Engineering, IEEE Transactions on,** v. 22, n. 10, p. 1345-1359, 2010.

PUTERMAN, M. L. **Markov decision processes: discrete stochastic dynamic programming**. Wiley. com, 2009.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. Cambridge Univ Press, 1998.

SUTTON, R. S.; BARTO, A. G.; WILLIAMS, R. J. Reinforcement learning is direct adaptive optimal control. **Control Systems, IEEE,** v. 12, n. 2, p. 19-22, 1992.

SUTTON, R. S.; PRECUP, D.; SINGH, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. **Artificial intelligence,** v. 112, n. 1, p. 181-211, 1999.

TAYLOR, M. E.; STONE, P. Transfer learning for reinforcement learning domains: A survey. **The Journal of Machine Learning Research,** v. 10, p. 1633-1685, 2009.

THRUN, S.; SCHWARTZ, A. Finding structure in reinforcement learning. **Advances in neural information processing systems**, p. 385-392, 1995.

TORREY, L.; SHAVLIK, J. Transfer learning. **Handbook of Research on Machine Learning Applications. IGI Global,** v. 3, p. 17-35, 2009.

Linguístico: Usando Tecnologia Adaptativa para a Construção de Um Reconhecedor Gramatical

Ana Contier, Djalma Padovani, João José 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 apresentado o Linguístico, uma proposta de reconhecedor gramatical que utiliza autômatos adaptativos como tecnologia subjacente.**

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

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). As ações adaptativas de consulta permitem inspecionar o conjunto de regras que definem o dispositivo em busca de regras que sigam um padrão fornecido. As ações elementares de exclusão permitem remover qualquer regra do conjunto de regras. As ações elementares de inclusão permitem especificar a adição de uma nova regra, de acordo com um padrão fornecido.

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.

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 morfo-sintático-semânticos, tais como gênero, número, grau, pessoa, tempo, modo, regência verbal ou nominal. Um item lexical pode ter uma ou mais

representações semânticas associadas a uma entrada. É o caso da palavra "casa", que pode aparecer das seguintes formas:

Casa: substantivo, feminino, singular, normal. Significado: moradia, habitação, sede

Casa: verbo singular, 3a pessoa, presente indicativo, 1ª conjugação. Significado: contrair matrimônio

Dada uma determinada sentença, o analisador léxico-morfológico identifica os itens lexicais que a compõem e obtém, para cada um deles, as diferentes descrições correspondentes às entradas no léxico. A ambiguidade léxico-morfológica ocorre quando uma mesma palavra apresenta diversas categorias gramaticais. Neste caso existem duas formas de análise: a tradicional e a etiquetagem. Pela abordagem tradicional, todas as classificações devem ser apresentadas pelo analisador, deixando a resolução de ambiguidade para outras etapas do processamento. Já pela etiquetagem (POS *Tagging*), o analisador procura resolver as ambiguidades sem necessariamente passar por próximas etapas de processamento. Nesta abordagem, o analisador recebe uma cadeia de itens lexicais e um conjunto específico de etiquetas como entrada e produz um conjunto de itens lexicais com a melhor etiqueta associada a cada item. Os algoritmos para etiquetagem fundamentam-se em dois modelos mais conhecidos: os baseados em regras e os estocásticos. Os algoritmos baseados em regras usam uma base de regras para identificar a categoria de um item lexical, acrescentando novas regras à base à medida que novas situações de uso do item vão sendo encontradas. Os algoritmos baseados em métodos estocásticos costumam resolver as ambiguidades através de um corpus de treino marcado corretamente, calculando a probabilidade que uma palavra terá de receber uma etiqueta em um determinado contexto.

O passo seguinte é a análise sintática. Nesta etapa, 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. A gramática adotada pode ser escrita por meio de diversos formalismos. Segundo [7] destacam-se as redes de transição, as gramáticas de constituintes imediatos (PSG ou phrase structure grammar), as gramáticas de constituintes imediatos generalizadas (GPSG) e as gramáticas de unificação funcional (PATR II e HPSG). As gramáticas de constituintes imediatos (PSG), livres de contexto, apresentam a estrutura sintática das frases em termos de seus constituintes. Por exemplo, uma frase (F) é formada pelos sintagmas nominal (SN) e verbal (SV). O sintagma nominal é um agrupamento de palavras que tem como núcleo um substantivo (Subst) e o sintagma verbal é um agrupamento de palavras que tem como núcleo um verbo. Substantivo e verbo representam classes gramaticais. O determinante (Det) compõe, junto com o substantivo, o sintagma nominal. O sintagma verbal é formado pelo verbo, seguido ou não de um sintagma nominal. O exemplo apresentado ilustra uma gramática capaz de reconhecer a frase: O menino usa o chapéu.

F → SN SV.
SN → Det Subst.
SV → Verbo SN.
Det → o
Subst → menino, chapéu
Verbo → usa

Considerando o processamento da direita para esquerda e de baixo para cima, a frase seria analisada da seguinda forma:

O menino usa o chapéu
1. chapéu = Subst: O menino usa o subst
2. O = Det : O menino usa det subst
3. Det Subst = SN:O menino usa SN
4. usa = verbo: O menino verbo SN
5. verbo SN=SV: O menino SV
6. menino = Subst: O subst SV
7. O = Det: Det subst SV
8. Det subst= SN: SN SV
9. SN SV= F: F (Aceita)

No entanto, este formalismo não consegue identificar questões de concordância de gênero e número. Por exemplo, se fossem incluídos no léxico o plural e o feminino da palavra menino, frases como: "O meninos usa o chapéu." e "O menina usa o chapéu." seriam aceitas. Por exemplo:

O meninos usa o chapéu
1. chapéu = Subst: O menino usa o subst
2. O = Det : O menino usa det subst
3. Det Subst = SN:O menino usa SN
4. usa = verbo: O menino verbo SN
5. verbo SN=SV: O menino SV
6. meninos = Subst: O subst SV
7. O = Det: Det subst SV
8. Det subst= SN: SN SV
9. SN SV= F: F (Aceita)

Para resolver este tipo de problema existem outros formalismos, tais como o PATR II:

F → SN, SV
  <SN numero> = <SV numero>
  <SN pessoa> = <SV pessoa>
SN → Det, Subst
  <Det numero> = <Subst numero>
  <Det genero > = <Subst genero>
SV → Verbo, SN
o
  <categoria> = determinante
  <genero> = masc
  <numero> = sing
menino
  <categoria> = substantivo
  <genero> = masc
  <numero> = sing
chapéu
  <categoria> = substantivo
  <genero> = masc
  <numero> = sing
usa
  <categoria> = verbo
  <tempo> = pres
  <numero> = sing
  <pessoa> = 3
  <argumento 1> = SN
  <argumento 2> = SN

Neste formalismo, a derivação leva em consideração outras propriedades do léxico, além da categoria gramatical, evitando os erros de reconhecimento apresentados anteriormente. Segundo [7], esse formalismo gramatical oferece poder gerativo e capacidade computacional, e tem sido usado com sucesso em ciência da computação, na especificação de linguagens de programação. Aplicando este formalismo ao exemplo acima, o erro de concordância seria identificado e a frase não seria aceita:

O meninos usa o chapéu
1. chapéu = Subst masc sing : O menino usa o subst
2. O = Det : O menino usa det subst
3. Det Subst = SN:O menino usa SN
4. usa = verbo pres 3a. Pessoa sing: O menino verbo SN
5. verbo SN=SVsing: O menino SVsing
6. meninos = Subst masc plural: O subst SVsing
7. O = Det: Det subst SVsing
8. Det subst= SNplur: SNplur SVsing
9. SNplur SVsing = Não aceita

Certas aplicações necessitam lidar com a interpretação das frases bem formadas, não bastando o conhecimento da estrutura, mas sendo necessário o conhecimento do significado dessas construções. Por exemplo, quando é necessário que respostas sejam dadas a sentenças ou orações expressas em língua natural, as quais, por exemplo, provoquem um movimento no braço de um robô. Ou quando é necessário extrair conhecimentos sobre um determinado tema a partir de uma base de dados textuais. 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 [8]. O mapeamento é feito identificando as propriedades semânticas do léxico e o relacionamento semântico entre os itens que o compõe. Para representar as propriedades semânticas do léxico, pode ser usado o formalismo PATR II, já apresentado anteriormente. Para a representação das relações entre itens do léxico pode ser usado o formalismo baseado em predicados: cada proposição é representada como uma relação predicativa constituída de um predicado, seus argumentos e eventuais modificadores. Um exemplo do uso de predicados é apresentado para ilustrar o processo de interpretação da sentença "O menino viu o homem de binóculo". Trata-se de uma sentença ambígua da língua portuguesa, uma vez que pode ser interpretada como se (a) O menino estivesse com o binóculo, ou (b) O homem estivesse com o binóculo. Uma gramática para a análise do exemplo acima é dada pelas seguintes regras de produção:

F → SN SV
SN → Det Subst
SN → SN SP
SV → V SN
SV → V SN SP
SP → Prep Subst

Uma possível representação semântica para as interpretações da sentença seria:

1.Sentença de interpretação (a):
agente(ação(ver), menino)
objeto(ação(ver), homem)
instrumento(ação(ver), binóculo)
2.Sentença de interpretação (b):
agente(ação(ver), menino)
objeto(ação(ver), homem)
qualificador(objeto(homem), binóculo)

Existem casos em que é necessário obter o conteúdo não literal de uma sentença, ligando as frases entre si, de modo a construir um todo coerente, e interpretar a mensagem transmitida de acordo com a situação e com as condições do enunciado [7]. Por exemplo, para uma compreensão literal da sentença: "O professor disse que duas semanas são o tempo necessário", é possível recorrer aos mecanismos de representação expostos até aqui, porém para uma compreensão aprofundada, seria necessário saber a que problema se refere o professor, já que o problema deve ter sido a própria razão da formulação dessa sentença. Nestes casos, é necessária uma nova operação denominada análise pragmática.

A análise pragmática procura reinterpretar a estrutura que representa o que foi dito para determinar o que realmente se quis dizer [2]. Dois pontos focais da pragmática são: as relações entre frases e o contexto. À medida que vão sendo enunciadas, as sentenças criam um universo de referência, que se une ao já existente. A própria vizinhança das sentenças ou dos itens lexicais também constitui um elemento importante na sua interpretação. Assim, alguns novos fenômenos passam a ser estudados, como fenômenos pragmático-textuais. Inserem-se nessa categoria as relações anafóricas, co-referência, determinação, foco ou tema, dêiticos e elipse [7]. Por exemplo, nem sempre o caráter interrogativo de uma sentença expressa exatamente o caráter de solicitação de uma resposta. A sentença "Você sabe que horas são?" pode ser interpretada como uma solicitação para que as horas sejam informadas ou como uma repreensão por um atraso ocorrido. No primeiro caso, a pergunta informa ao ouvinte que o falante deseja obter uma informação e, portanto, expressa exatamente o caráter interrogativo. Entretanto, no segundo caso, o falante utiliza o artifício interrogativo como forma de impor sua autoridade. Diferenças de interpretação desse tipo claramente implicam interpretações distintas e, portanto, problemáticas, se não for considerado o contexto de ocorrência do discurso [9]. As questões relacionadas à análise pragmática são objetos de estudos de modo a prover mecanismos de representação e de inferência adequados, e raramente aparecem em processadores de linguagem natural [7].

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.

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, se diferenciando 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 denominados frasais ou oracionais. Estes padrões são compostos por elementos denominados sintagmas. 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

A Tabela 1 apresenta os elementos formadores dos sintagmas, e a sequência em que aparecem, de acordo com Luft.

TABELA 1.
Elementos formadores de sintagmas [12]

Sintagmas	
Substantivo	Quantitativos+Pronomes Adjetivos+ Sintagma Adjetivo1+Substantivo+ Sintagma Adjetivo2+ Sintagma Preposicional+ Oração Adjetiva
Verbal	Pré-verbais+ Verbo Auxiliar+ Verbo Principal
Adjetivo	Advérbio de Intensidade+ Adjetivo+ Sintagma Preposicional
Adverbial	Advérbio de Intensidade+ Adverbio+ Sintagma Preposicional
Preposicional	Preposição+ Sintagma Substantivo

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. A Tabela 2 apresenta a relação de todos os padrões oracionais propostos por Luft.

Os padrões são classificados em 5 tipos:

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).

TABELA 2
Padrões oracionais de Luft [12]

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			

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.

TABELA 2 - CONTINUAÇÃO

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		

TABELA 2 - CONTINUAÇÃO

Padrões Impessoais Verbais				
	Vtd	SS		
	Vti	SP		
	Vi			

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.

Luft apresenta uma gramática usada para análise sintática da Língua Portuguesa no modelo moderno, em que as frases são segmentadas o mais binariamente possível: Sujeito+Predicado; Verbo+Complemento; Substantivo+Adjetivo, etc. Neste modelo, a descrição explicita somente as classes analisadas; as funções ficam implícitas. Querendo explicar estas, Luft sugere que sejam escritas à direita das classes: SS:Sj (Sujeito), V:Núc (Núcleo do Predicado), PrA:NA (Adjunto Adnominal), etc.
A gramática proposta por Luft é a seguinte:

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

PROPOSTA DE UM RECONHECEDOR GRAMATICAL

O Linguístico é uma proposta de reconhecedor gramatical composto de 5 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.

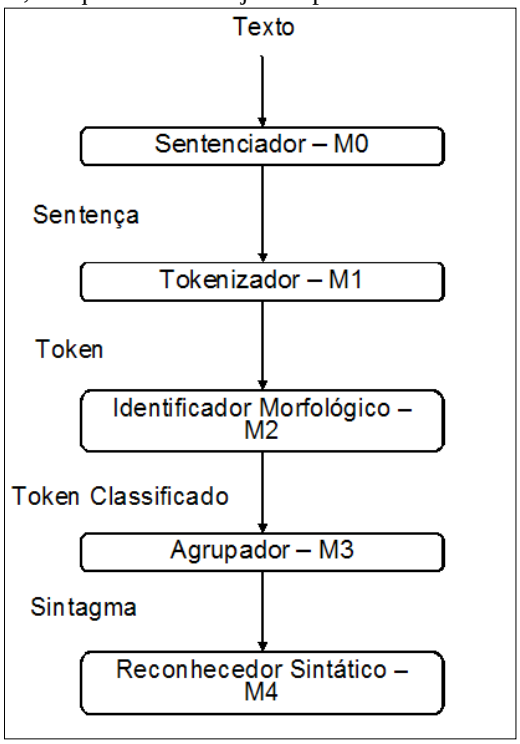


Figura 2. Estrutura do Linguístico

A Fig.2 ilustra a 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, denominado Identificador Morfológico, recebe os *tokens* da etapa anterior e os identifica morfologicamente, utilizando, como biblioteca de apoio, os

textos pré-anotados do corpus Bosque[13], os verbos, substantivos e adjetivos que fazem parte da base de dados do TeP2.0 – Thesauro Eletrônico para o Português do Brasil [14] e as conjunções, preposições e pronomes disponíveis no Portal São Francisco[15], cujas informações provém da Wikipedia [16]. O Bosque é um conjunto de frases anotadas morfossintaticamente (conhecido por *treebank),* composto por 9368 frases retiradas dos primeiros 1000 extratos do corpora CETEMPublico (Corpus de Extractos de Textos Electrónicos MCT/Público) e CETENFolha (Corpus de Extractos de Textos Electrónicos NILC/Folha de S. Paulo). A Fig. 3 apresenta um fragmento do Bosque.

```
#5156 CP1003-14 Todas as interpretações são possíveis, tudo pode
acontecer a este corpo.
(STA+cu (CJT+fcl (SUBJ+np
                    (>N+pron-det Todas_as)
                    (H+n interpretações))
                 (P+v-fin são)
                 (SC+adj possíveis))
        (,)
        (CJT+fcl (SUBJ+pron-indp tudo)
                 (P+vp
                   (AUX+v-fin pode)
                   (MV+v-inf acontecer))
                 (PIV+pp
                   (H+prp a)
                   (P<+np
                     (>N+pron-det este)
                     (H+n corpo))))
        (.))
```

Figura 3. Exemplo de frase etiquetada do corpus Bosque [11].

O TeP2.0 é um dicionário eletrônico de sinônimos e antônimos para o português do Brasil, que armazena conjuntos de formas léxicas sinônimas e antônimas. É composto por 19.888 conjuntos de sinônimos, 44.678 unidades lexicais, e 4.276 relações de antonímia [14].

O Portal São Francisco apresenta um curso online da Língua Portuguesa e, entre seus módulos, encontra-se um sumário das classes morfológicas, no qual são encontrados exemplos de palavras e locuções mais comuns de cada classe [15].

Inicialmente, o Identificador Morfológico procura pela classificação morfológica dos tokens no léxico do Bosque, caso não a encontre, então ele procura na base de dados do TeP2.0 e no léxico do Portal São Francisco.

O padrão de etiquetas usado pelo Linguístico é o mesmo do Bosque, que apresenta, além da classificação morfológica, o papel sintático que o token exerce na sentença pré-anotada. Por exemplo, a etiqueta >N+art indica que o token é um artigo que está à esquerda de um substantivo (>N). A notação usa como gramática subjacente a Gramática Constritiva proposta por Fred Karlsson [17].

O léxico do Bosque foi organizado de modo a relacionar todas as classificações de um tokens ordenadas por frequência em que ocorrem no texto pre-anotado. Por exemplo, o token "acentuada" está classificado com as seguintes etiquetas: N<+v-pcp e P+v-pcp, o que significa que, no texto pré-anotado, ele foi classificado como verbo no particípio (+v-pcp) antecedido de um substantivo (N<) e como verbo no particípio no papel de predicador (P).

No caso de ambiguidade, o Identificador Morfológico assume a classificação mais frequente como inicial e verifica se a classificação mais frequente do token seguinte é consistente com o que indica a etiqueta do token analisado. Se for, vale a classificação mais frequente, senão o Identificador analisa a próxima classificação, repetindo o algoritmo. Caso o algoritmo não retorne uma classificação única, o Identificador passa todas as classificações encontradas para os módulos seguintes, para que ambiguidade seja resolvida pelas regras gramaticais do reconhecedor.

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.4). Inicialmente, o Agrupador recebe do Identificador as classificações morfológicas dos *tokens* e as agrupa em sintagmas de acordo com a gramática proposta por 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}

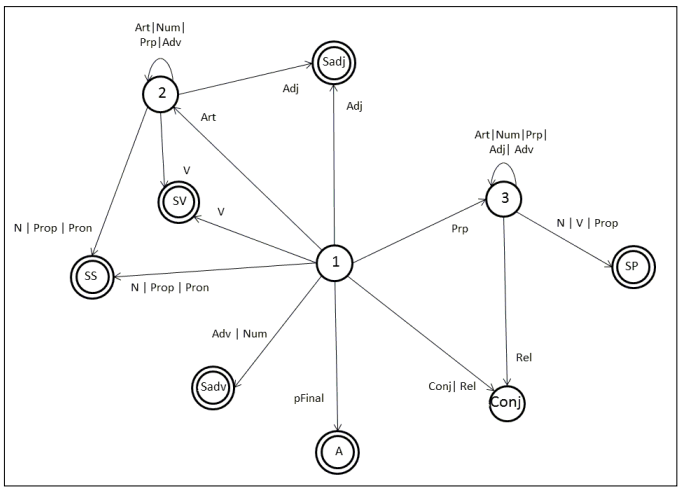


Figura 4. Configuração Completa do Autômato Construtor de Sintagmas.

Por exemplo, segundo a gramática de Luft, os sintagmas substantivos são obtidos através da seguinte regra:

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 da gramática de Luft. Para isso ele usa a matriz apresentada na Tabela 3, construída a partir da gramática de Luft. 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 da gramática.

TABELA 3
Matriz de agrupamento de sintagmas

US \ SA	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 gramaticais 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]}

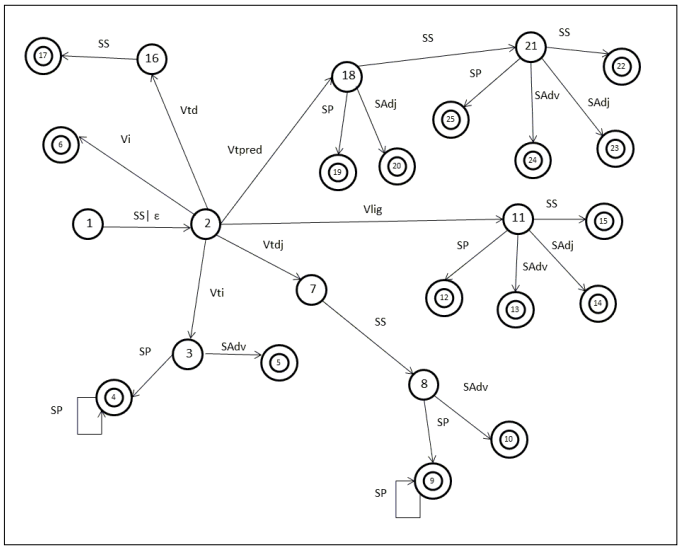


Figura 5.1. Configuração Completa do Reconhecedor Sintático.

No entanto, para que a análise sintática seja feita, não são necessárias todas as ramificações da configuração completa do autômato (Fig. 5.1). 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. 5.2 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.

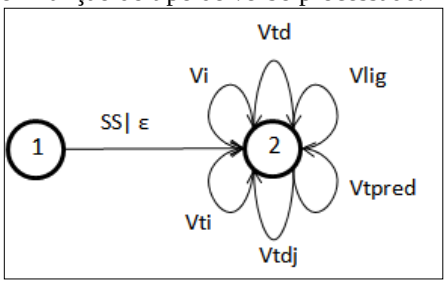


Figura 5.2. Configuração Inicial do Reconhecedor Gramatical.

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. 5.3 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. 5.1.

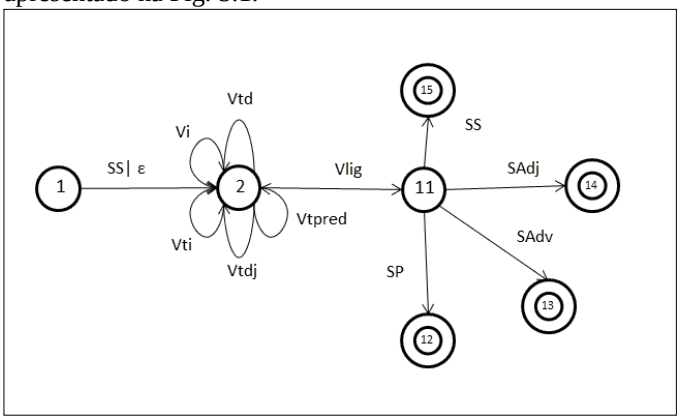


Figura 5.3. Configuração do autômato após o processamento do verbo de ligação.

Toda movimentação do autômato, assim como os sintagmas identificados em cada passagem e a classificação morfológica dos termos das sentenças, são armazenados em arquivos que podem ser acessados por um editor. Se o Linguístico não consegue reconhecer a sentença, ele registra os erros encontrados e grava uma mensagem alertando para o ocorrido.

Desenvolvimento do Linguístico – Sentenciador e Tokenizador

O Linguístico está sendo desenvolvido na linguagem de programação Python[18], escolhida devido aos recursos nativos de processamento de expressões regulares, programação dinâmica e por oferecer flexibilidade de implementação tanto no paradigma procedural quanto na orientação a objetos. O primeiro componente desenvolvido, chamado Texto, cuida da preparação do texto que vai ser analisado. Ele é formado por uma classe chamada Texto que incorpora o Sentenciador e o Tokenizador do Linguístico. A classe Texto possui 3 métodos que são acionados em sequência, sempre que encontra um novo texto para ser analisado. O método Prepara_Texto se encarrega de eliminar aspas simples e duplas, e identificar abreviaturas e palavras compostas na base de dados formada pelos léxicos Bosque e TeP2.0. O método Divide_Texto cria uma coleção de sentenças através de expressões regulares que interpretam como caracteres limitadores de cada sentença o ponto final, de interrogação e de exclamação, seguidos de um espaço em branco. Ao final, o método Tokeniza_Sentenca cria uma coleção de tokens a partir das sentenças, usando como critérios de divisão, regras para identificação de tokens alfanuméricos, valores monetários, horas e minutos, numerais arábicos e romanos, percentuais, além das abreviaturas e palavras compostas identificadas na etapa de preparação.

Desenvolvimento do Linguístico- O Identificador Morfológico

O Identificador Morfológico foi concebido para utilizar tecnologias adaptativas (Fig.6.1). O Identificador Morfológico é composto por um Autômato Mestre e um conjunto de

submáquinas especialistas. O Autômato Mestre é responsável pelo sequenciamento das chamadas às submáquinas, de acordo com um conjunto de regras cadastradas em base de dados.

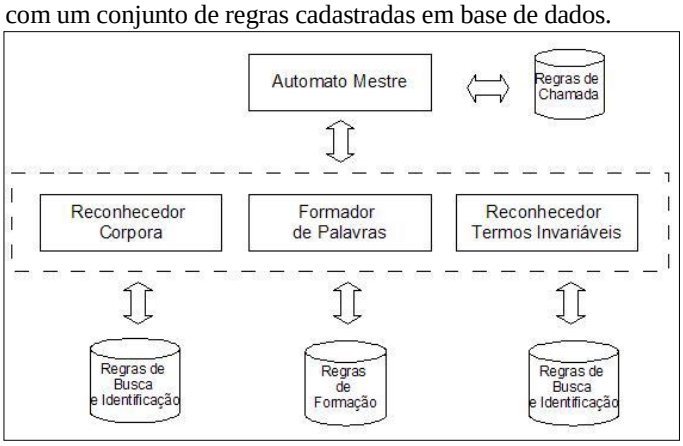
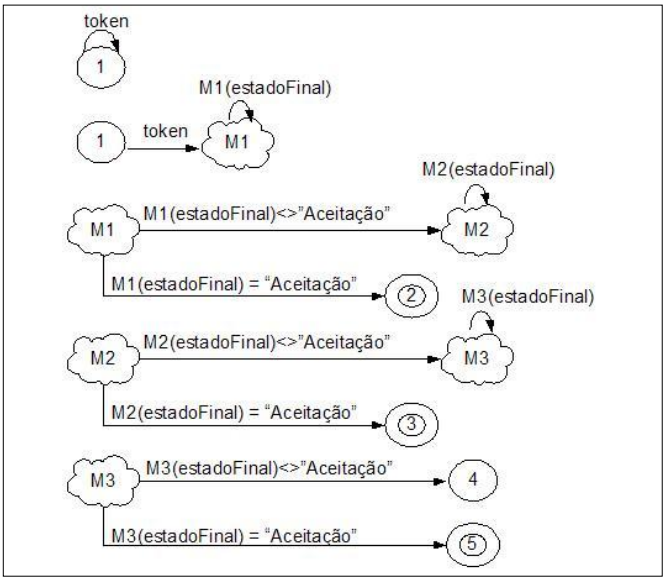


Figura 6.1. Arquitetura do Identificador Morfológico.

Inicialmente, o Autômato Mestre busca as classificações morfológicas no c (no caso, o Bosque, mas poderia ser outro, se houvesse necessidade); em seguida 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 (aqui também existe a possibilidade de usar outra base de dados, substituindo ou complementando o TeP2.0); por fim, o Autômato Mestre 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. A Fig. 6.2 apresenta a estrutura adaptativa do Autômato Mestre.

Figura 6.2. Estrutura Adaptativa do Autômato Mestre.



O Autômato Mestre inicia seu 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. 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 automodifica de acordo com o tipo de token que está sendo analisado: palavras simples, palavras compostas, números, valores e símbolos. A Fig. 6.3 apresenta a estrutura adaptativa de M1.

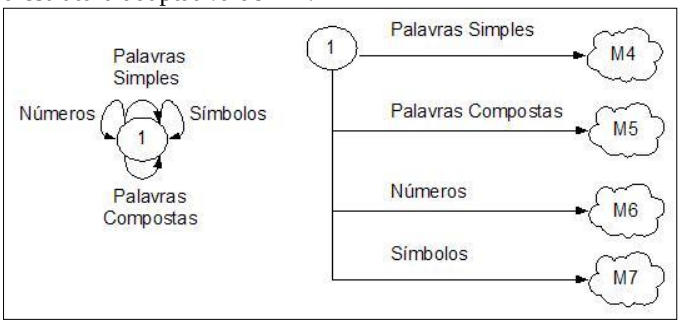


Figura 6.3. Estrutura Adaptativa do Reconhecedor de Corpus M1.

Inicialmente o autômato é composto por um único estado e por transições para ele mesmo (Fig.6.3, 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.6.3, 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. 6.4).

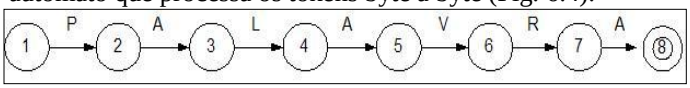


Figura 6.4. Reconhecedor de Palavras Simples.

No exemplo apresentado na Fig.6.4, 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 Basilio em [19]. Segundo a autora, o léxico é constituído por uma lista de formas já feitas e por um conjunto de padrões, os processos de formação de palavras, que determinam estruturas e funções tanto das formas já existentes quanto de formas ainda a serem construídas.

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. 6.5 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.

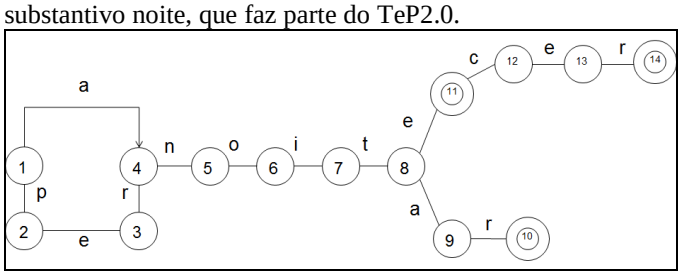


Figura 6.5. Autômato Formador de Palavras.

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 [20]. No entanto, acrescenta, existem centenas de substantivos terminados em –ção que não admitem formação adjetiva com o sufixo –oso. Por exemplo, vocação/vocacioso; intenção/intencioso; atração/atracioso. Segundo a autora, as derivações mais prováveis são aquelas em que há generalidade das funções envolvidas no processo de formação. Noções como a negação, o grau e a nominalização de verbos são bastante comuns e de grande generalidade, consequentemente, mais prováveis de serem válidas.

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 [19], 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.

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. 6.6 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.

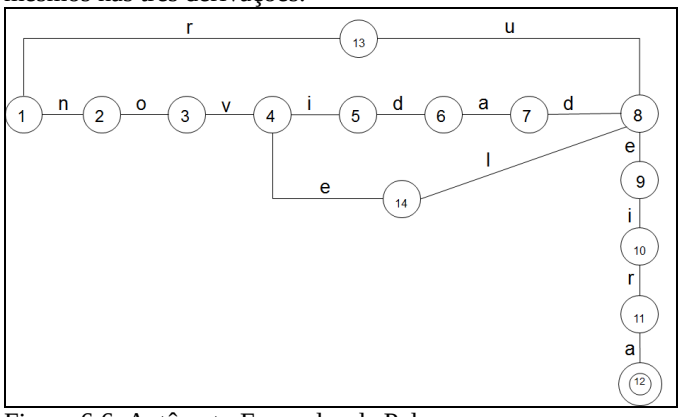


Figura 6.6. Autômato Formador de Palavras.

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. 6.7 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)".

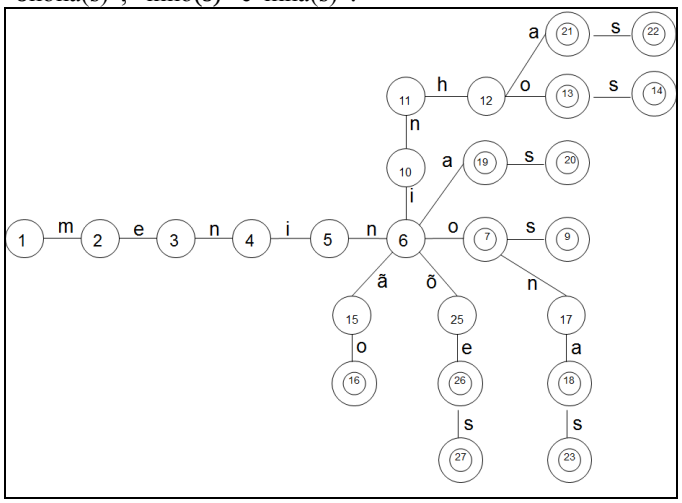


Figura 6.7. Autômato Formador de Flexões Nominais.

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. 6.8 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".

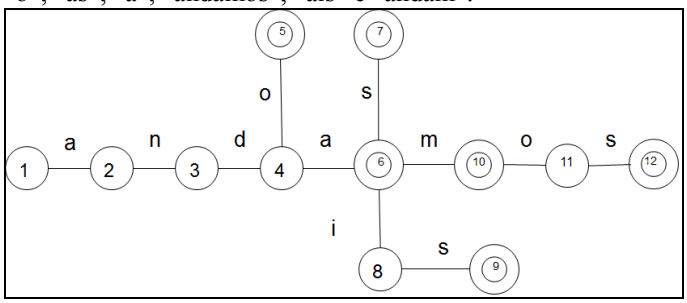


Figura 6.8. Autômato Formador de Flexões Verbais.

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. 6.9 apresenta um exemplo de autômato usado no reconhecimento de termos deste domínio.

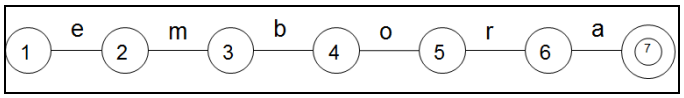


Figura 6.9. Autômato Reconhecedor de Conjunções, Preposições e Pronomes.

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 Franscisco. Por exemplo, o termo "embora" seria classificado como cj+Ba –conjunção da base de dados de apoio.

Desambiguação Morfológica – Algoritmo Viterbi

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 [21] 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 [22]. 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. 6.10 apresenta a estrutura do autômato que implementa o algoritmo Viterbi usado pelo desambiguador morfológico do Linguístico.

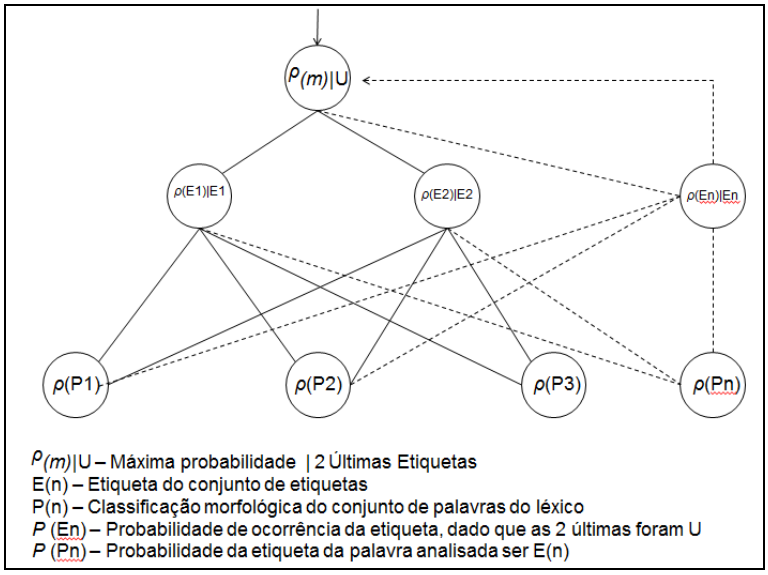


Figura 6.10. Autômato Adaptativo Viterbi.

O autômato proposto 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. Por exemplo, no caso da frase citada anteriormente "O advogado entrevista a testemunha", o autômato se comportaria da seguinte forma:

- Primeiro token: "O"

ρ (m)|U = 1 | ** (* * indica inicio de frase)
E1 = Artigo
E2 = Pronome
E3 = Preposição
ρ (E1) = 0,55: probabilidade de artigo seguido de * *
ρ (E2) = 0,25: probabilidade de pronome seguido de * *
ρ (E3) = 0,20: probabilidade preposição seguido de * *
P1 – O (artigo)
P2 – O (pronome)
P3 – O (preposição)
ρ (P1) = 0,60: probabilidade de "O" ser artigo
ρ (P2) = 0,30: probabilidade de "O" ser pronome
ρ (P3) = 0,10: probabilidade de "O" ser preposição
ρ (m) = 0,55*0,6 | 0,25*0,3| 0,2*0,1
ρ (m) = 0,55*0,6 = 0,33| U = Artigo

- Segundo token: "advogado"

ρ (m)|U = 0,33 | * Artigo
E1 = Substantivo
E2 = Adjetivo
E3 = Adverbio
ρ (E1) = 0,60: probabilidade de substantivo seguido de * e artigo
ρ (E2) = 0,20: probabilidade de adjetivo seguido de * e artigo
ρ (E3) = 0,20: probabilidade de adverbio seguido de * e artigo
P1 – Advogado (substantivo)
ρ (P1) = 1,0: probabilidade de "Advogado" ser substantivo
ρ (P2) = 0: probabilidade de "Advogado" ser adjetivo
ρ (P3) = 0: probabilidade de "Advogado" ser adverbio
ρ (m) = 0,33*0,60*1 |0,33* 0,20*0| 0,33*0,20*0
ρ (m) = 0,33*0,60*1 = 0,198| U = Artigo Substantivo

- Terceiro token: "entrevista"
ρ (m)|U = 0,198 | Artigo Substantivo
E1 = Substantivo
E2 = Verbo
E3 = Adjetivo
ρ (E1) = 0,60: probabilidade de verbo seguido de artigo e substantivo
ρ (E2) = 0,20: probabilidade de adjetivo seguido de artigo e substantivo
ρ (E3) = 0,20: probabilidade de adjetivo seguido de artigo e substantivo
P1 – entrevista (verbo)
P2 – entrevista (substantivo)
ρ (P1) = 0,70: probabilidade de "entrevista" ser verbo
ρ (P1) = 0,30: probabilidade de "entrevista" ser substantivo
ρ (P1) = 0: probabilidade de "testemunha" ser adjetivo
ρ (m) = 0,198*0,60*0,70 |0,198*0,20* 0,30|0,198*0,20* 0
ρ (m) = 0,198*0,60*0,70 = 0,0831| U = Substantivo Verbo

- Quarto token: "a"
ρ (m)|U = 0,0831 | Substantivo Verbo
E1 = Artigo
E2 = Substantivo
E3 = Adjetivo
ρ (E1) = 0,60: probabilidade de artigo seguido de substantivo e verbo
ρ (E2) = 0,20: probabilidade de substantivo seguido de substantivo e verbo
ρ (E3) = 0,20: probabilidade de adjetivo seguido de substantivo e verbo
P1 – A (artigo)
P2 – A (pronome)
P3 – A (preposição)
ρ (P1) = 0,60: probabilidade de "a" ser artigo
ρ (P2) = 0,30: probabilidade de "a" ser pronome
ρ (P3) = 0,10: probabilidade de "a" ser preposição
ρ(m) = 0,0831*0,60*0,60|0,0831*0,20* 0,30|0,0831*0,20* 0,10
ρ (m) = 0,0831*0,60*0,60= 0,0299| U = Verbo Artigo

- Quinto token: "testemunha"
ρ (m)|U = 0, 0299| Verbo Artigo
E1 = Substantivo
E2 = Verbo
E3 = Adjetivo
ρ (E1) = 0,60: probabilidade de substantivo seguido de verbo e artigo
ρ (E2) = 0,20: probabilidade de verbo seguido de verbo e artigo
ρ (E3) = 0,20: probabilidade de adjetivo seguido de verbo e artigo
P1 – testemunha (verbo)
P2 – testemunha (substantivo)
ρ (P1) = 0,70: probabilidade de "testemunha" ser substantivo
ρ (P2) = 0,30: probabilidade de "testemunha" ser verbo
ρ (P3) = 0: probabilidade de "testemunha" ser adjetivo
ρ (m) = 0, 0299*0,60*0,70 |0, 0299*0,20* 0,30|
0, 0299*0,20* 0
ρ (m) = 0,0299*0,60*0,70 = 0,0125| U = Artigo Substantivo

Portanto, ao final do processamento, a sequência de etiquetas seria: Artigo Substantivo Verbo Artigo Substantivo.

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.

O Linguístico encontra-se em fase de construção, dividida em etapas em função da estrutura do reconhecedor. A primeira versão, englobando sentenciador, tokenizador e identificador morfológico, foi finalizada.

O trabalho encontra-se na fase de projeto do Desambiguador Morfológico, que foi totalmente concebido para utilizar tecnologia adaptativa.

Referências

[1] Neto, J.J. Apresentação LTA-Laboratório de Linguagens e Técnicas Adaptativas. Disponível em: http://www.pcs.usp.br/~lta. Acesso 01/11/2009.
[2] Taniwaki, C. Formalismos adaptativos na análise sintática de Linguagem natural . Dissertação de Mestrado, EPUSP, São Paulo, 2001.
[3] Menezes, C. E. 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] Padovani, D. Uma proposta de autômato adaptativo para reconhecimento de anáforas pronominais segundo algoritmo de Mitkov. Workshop de Tecnologias Adaptativas – WTA 2009, 2009.
[5] Moraes, M. de 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] Rich, E.; Knight, K. Inteligência Artificial, 2. ed. São Paulo: Makron Books, 1993.
[7] Vieira, R.; Lima, V Linguística computacional: princípios e aplicações. IX Escola de Informática da SBC-Sul, 2001.
[8] Fuchs, C., Le Goffic, P. Les Linguistiques Contemporaines.
[9] Nunes, M. G. V. et al. Introdução ao Processamento das Línguas Naturais. Notas didáticas do ICMC Nº 38, São Carlos, 88p, 1999. Paris, Hachette, 1992. 158p.
[10] Sardinha, T. B. A Língua Portuguesa no Computador. 295p. Mercado de Letras, 2005.
[11] Rocha, R.L.A. Tecnologia Adaptativa Aplicada ao Processamento Computacional de Língua Natural. Workshop de Tecnologias Adaptativas – WTA 2007, 2007.

[12] Luft, C. Moderna Gramática Brasileira. 2ª. Edição Revista e Atualizada. 265p. Editora Globo, 2002.
[13] Linguateca : http://www.linguateca.pt/
[14]Tep2. Thesauro Eletrônico para o Português do Brasil. Disponível em: <http://www.nilc.icmc.usp.br/tep2/>
[15] Portal São Francisco. Materiais de Língua Portuguesa. Disponível em: <http://www.portalsaofrancisco.com.br/alfa/materias/index-lingua- portuguesa.php/>
[16] Wikipedia. Disponível em: <http://pt.wikipedia.org/wiki/P%C3%A1gina_principal/>
[17] Karlsson, F. Constraint Grammar as a Framework for Parsing Running Text. 13o. International Conference on Computational Linguistics, Helsinki (Vol.3, pp.168-173).
[18] Python. Python Programming Language Official Web Site. Disponível em: <http://www.python.org/>
[19] Basilio, M. Formação e Classes de Palavras no Português do Brasil. Ed.Contexto, 2004
[20] Basilio, M. Teoria Lexical. Ed.Atica, 1987
[21] Collins, M. COURSE NOTES FOR NLP. Columbia University. Disponível em: http://www.cs.columbia.edu/~mcollins/hmms-spring2013.pdf
[22] Forney, G.D., Jr. IEEE. Proceedings of the IEEE (Volume:61 , Issue: 3), 1973

Ana Teresa Contier: formada 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 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.

Aplicação da Tecnologia Adaptativa e Reflexão Computacional por meio da programação reflexiva em Java

D. A. Passareli, A. R. Camolesi, D. M. R. Barros

diepassa@bol.com.br, camolesi@femanet.com.br, diomara@femanet.com.br
Coordenadoria de Informática, Fundação Educacional do Município de Assis, Brasil

***Resumo* — Este trabalho apresenta conceitos de Tecnologia Adaptativa e Reflexão Computacional. Baseado nessas tecnologias e utilizando programação reflexiva em Java, foi desenvolvido um estudo de caso, que se trata de uma tela para cálculo da mensalidade de um plano de saúde, onde, as vantagens em nível de desenvolvedor são muito grandes, como facilidade na manutenção do código fonte e otimização do trabalho. Para ilustrar o funcionamento da aplicação, foi desenvolvido também um Autômato de Estados Finitos Adaptativo.**

***Abstract* — This paper presents concepts of Adaptive Technology and Computational Reflection. Based on these technologies and using reflective programming in Java, a case study was developed, that it is a canvas for calculating the monthly payment of a health plan, where the benefits at the level of developer are very large, as was developed in ease of maintenance source code and optimization work. To illustrate the operation of the application, was also developing a Finite Automaton for Adaptive States.**

***Palavras Chave* — Tecnologia Adaptativa, Reflexão Computacional, Programação Reflexiva, Java.**

I. INTRODUÇÃO

É comum encontrar no meio comercial, sistemas que não se modificam, em comportamento e estrutura, para solucionar um problema ou em situações inesperadas.

A Tecnologia Adaptativa deu origem à ideia de que qualquer dispositivo que possua um conjunto fixo e finito de regras para sua operação, pode variar de acordo com outro nível de regras, denominado *ações adaptativas*, que agem sobre o conjunto de regras original através das ações de inserção, remoção e consulta das mesmas, podendo assim alterar sua estrutura interna durante seu funcionamento [1].

O emprego da Tecnologia Adaptativa em sistemas comerciais, que em sua maioria apresentam estruturas fixas, é possível graças a técnicas de programação que permitem modificar suas estruturas em tempo de execução. Uma dessas técnicas é a Reflexão Computacional, que torna possível ao programador acessar as políticas de controle de execução [2].

II. DISPOSITIVOS ADAPTATIVOS

A teoria dos dispositivos baseados em regras adaptativos tem como base a ideia de que dispositivos com maior poder de expressão podem ser obtidos a partir de uma progressão de um dispositivo mais simples [1].

Um simples dispositivo guiado por regras inicia seu funcionamento com certa configuração e ao longo de seu funcionamento, de acordo com os estímulos de entrada, alterna entre as configurações baseadas nas regras existentes até que nenhuma regra possa ser aplicada ou até que os estímulos de entrada terminem [1].

Já um dispositivo adaptativo, pode alterar seu conjunto de regras em função dos estímulos de entrada, baseado em outro nível de regras. Esse segundo nível de regras, chamado de *camada adaptativa*, age sobre o nível de regras original, chamado de *camada subjacente*, transformando assim um dispositivo qualquer guiado por regras em um dispositivo adaptativo [1], como pode ser observado na Figura 1.

A camada adaptativa é exclusivamente dedicada à descrição dos fenômenos de automodificação que devem ocorrer no dispositivo adaptativo que se deseja construir [3].

Figura 1: Estrutura Geral de um dispositivo adaptativo.

Um dos dispositivos abstratos que incorporam de forma mais natural esse recurso é o autômato, entendido como qualquer dispositivo com estados cuja operação seja definida por um conjunto de transições entre esses estados [4].

Durante a execução das suas transições, o autômato pode sofrer mudanças em sua topologia, tanto de ampliação ou de redução em seu conjunto de estados e transições iniciais à medida que vai reconhecendo a cadeia de entrada [5].

Um autômato adaptativo possui como camada subjacente um autômato de pilhas estruturado e, como camada adaptativa, as ações implementadas por funções adaptativas. Essas funções determinam as modificações que devem ser realizadas na camada subjacente quando uma ação adaptativa é chamada [1].

III. REFLEXÃO COMPUTACIONAL

O fundamento de Reflexão Computacional originou-se em lógica matemática e, recentemente, mecanismos de alto nível o tornam um aliado na adição de características operacionais ou não funcionais a módulos já existentes [6].

A reflexão pode ser definida como qualquer ação executada por um sistema computacional sobre si próprio, em outras palavras, é a capacidade de um programa reconhecer detalhes internos em tempo de execução que não estavam disponíveis no momento da compilação [7].

Quando um programa reflexivo entra em execução, ele considera suas próprias condições e informações contextuais. Deste modo, um programa reflexivo tem a habilidade de "pensar" sobre o que está acontecendo e se alterar dependendo das circunstâncias [8].

O termo reflexão tem dois significados distintos: introspecção, que se refere ao ato de examinar a si próprio, e redirecionamento da luz. Na Ciência da Computação, reflexão computacional denota a capacidade de um sistema examinar sua própria estrutura e estado (relacionado à introspecção) e o poder de fazer alterações no seu comportamento através do redirecionamento [7]. A representação das informações manipuláveis de um sistema sobre si próprio é chamada de metainformação [9].

Para que o mecanismo de reflexão seja implementado de forma flexível é necessário definir uma arquitetura reflexiva [10]. A arquitetura reflexiva compõe-se de dois níveis: *metanível* e *nível base*. No metanível se encontram as estruturas de dados e as ações que são executadas sobre o sistema objeto que está presente no nível base [8].

O metanível é explorado para expressar propriedades não funcionais do sistema, de forma independente do domínio da aplicação. Essas propriedades não funcionais, ao contrário das funcionais, não apresentam funções a serem realizadas pelo software, mas comportamentos e restrições que este software deve satisfazer [6].

A Figura 2 representa o processo de reflexão em um sistema computacional que pode ser dividido em vários níveis, onde o usuário envia uma mensagem ao sistema computacional e ela é tratada pelo nível funcional, que é responsável por executar corretamente a tarefa. Assim, o nível não funcional realiza a tarefa de gerenciar o funcionamento do nível funcional [6].

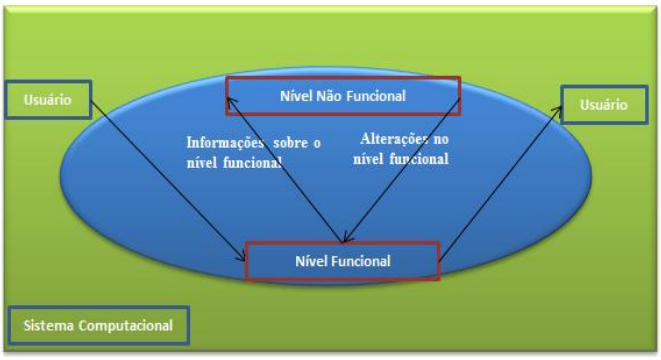


Figura 2: Visualização genérica de um sistema computacional reflexivo.

Segundo Wu (1997 apud BARTH, 2000, p. 20), o conceito básico sobre reflexão computacional está em separar as funcionalidades básicas das funcionalidades não básicas através de níveis arquiteturais, onde as funcionalidades básicas devem ser efetuadas pelos objetos da aplicação e as não básicas pelos metaobjetos. As capacidades não funcionais são adicionadas aos objetos da aplicação através de seus metaobjetos específicos e o objeto base pode ser alterado em estrutura e comportamento em tempo de execução. Metaobjetos são instâncias de uma classe pré-definida ou de uma subclasse da classe pré-definida.

Atribuir a um sistema tal capacidade significa dar-lhe flexibilidade para se adaptar dinamicamente, em estrutura e comportamento, favorecendo a reutilização (independente das classes do programa de nível base e do programa de metanível) e proporcionando adaptatividade [6].

No nível base são encontradas as classes e objetos pertencentes ao sistema objeto, e tem a funcionalidade de resolver problemas e retornar informações sobre o domínio externo, enquanto o nível reflexivo (metanível), que possui metaclasses e metaobjetos, resolve os problemas do nível base e retorna informações sobre a computação do objeto, podendo adicionar funcionalidades extras a ele [10].

A arquitetura reflexiva admite diversos metaníveis, caracterizando uma torre de reflexão, onde cada nível da torre estabelece um domínio D_i, tornando-se domínio base do domínio D_{i+1} [6]. Essa torre de reflexão pode ser analisada na Figura 3.

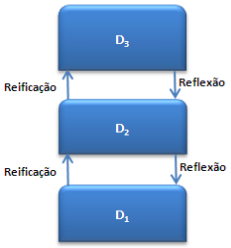


Figura 3: Torre reflexiva.

Na Figura 3, D_1 é o nível base da aplicação, D_2 é o metanível do D_1 e o nível base do D_3, e D_3 é o metanível do D_2 [6]. È possível dizer que esse é um exemplo de um adaptativo do adaptativo.

Segundo Souza (2001 apud SOUSA, 2002, p. 15), a reificação (materialização) é o ato de converter o que estava previamente implícito em algo explicitamente formulado, que é então disponibilizado para manipulação conceitual, lógica ou computacional.

É através do processo de reificação que o metanível obtém as informações estruturais internas dos objetos do nível base, tais como métodos e atributos [8].

IV. TECNOLOGIA JAVA

Java é uma plataforma e uma linguagem de programação de alto nível orientada a objetos, segura e independente de plataforma, lançada pela Sun Microsystems em 1995, mas que se originou com um projeto em 1991 sendo baseado nas

linguagens de programação C/C++.

A linguagem Java é compilada e diferentemente de muitas linguagens, seu código é primeiramente compilado para *bytecode* para depois ser executado pela JVM (*Java Virtual Machine*). Bytecode é uma espécie de código assembler para a JVM. Este código é otimizado pela JVM, que o interpreta, gerando e passando ao hardware em que esta instalada, os comandos necessários.

A maioria dos programas é escrita para trabalhar com dados, em geral para ler, escrever, manipular e exibir dados. Para alguns tipos de programas os dados a serem manipulados não são números ou textos, mas são as informações sobre programas e tipos de programa. Essas informações são conhecidas como metadados, que permitem a um assembly e os tipos dentro do assembly se autodescreverem [11].

Um programa pode olhar para os metadados enquanto ele está em execução e, isso é chamado de *reflection* (reflexão) [11]. A ênfase da programação reflexiva é a modificação ou a análise dinâmica, podendo modificar sua estrutura e praticar a autoanálise em tempo de execução.

A programação reflexiva em Java é possível graças à API (*Application Programming Interface*) *Reflection*, que fornece novos mecanismos para auxiliar no desenvolvimento de software.

A metaprogramação possui como principais vantagens a criação de aplicativos mais dinâmicos e a consequente redução do código fonte implementado. Contudo, como principais desvantagens ela apresenta a exigência de um maior nível de atenção e um domínio mais avançado de lógica de programação, além da dependência de linguagem [12].

A linguagem Java possui um pacote de reflexão, que suporta introspecção sobre classes e objetos atuais na JVM, permitindo manipular classes, interfaces, atributos e métodos em tempo de execução, a *java.lang.reflect*.

Com a programação reflexiva em Java é possível obter várias informações sobre o código fonte durante o tempo de execução, tais como a classe de um objeto, o pacote de uma classe, os atributos e métodos de uma classe e etc. É possível também criar uma instância de uma classe dinamicamente e obter e alterar os valores de atributos de uma instância [12].

Essa API também permite inspecionar e manipular metainformações, como as *annotations* [12].

Annotations, ou Anotações, são metadados que podem ser acoplados a vários elementos de codificação para posterior recuperação. Elas incorporam informações adicionais ao código, chamadas de metainformações e diminuem a necessidade de arquivos de configuração externos.

Mesmo não sendo muito utilizadas no cotidiano do desenvolvimento, as anotações são relativamente simples de serem criadas, similares às declarações de interfaces convencionais, bastando apenas adicionar o símbolo "@" precedido do nome da anotação a ser utilizada. Algumas anotações podem ser inseridas sem nenhuma informação adicional, outras, no entanto, utilizam atributos que servem para incrementar o efeito da anotação no código no momento da execução. Para usar as anotações é preciso definir a forma que elas serão lidas, podendo ser diretamente do código-fonte, arquivos de classes ou em tempo de execução (*runtime*), sendo este último o mais utilizado [13] e que será utilizado para o desenvolvimento do estudo de caso.

V. ESTUDO DE CASO

Foi proposto desenvolver um módulo de um Sistema de Planos de Saúde aplicando os conceitos das tecnologias citadas, desenvolvendo também um Autômato de Estados Finitos Adaptativo para ilustrar o funcionamento desse módulo.

O módulo realiza o cálculo da mensalidade de um plano de saúde a ser adquirido por um beneficiário em potencial, em tempo de execução, de acordo com os parâmetros relacionados ao plano de saúde escolhido.

O cálculo da mensalidade para a aquisição de um plano de saúde pode ser formado por vários parâmetros e efetuado de maneira bem simples. Para este estudo de caso foram escolhidos alguns parâmetros básicos: *tipo de contratação*, *sexo*, *data de nascimento* e *módulos*. O cálculo realizado é bem simples.

Um valor de R$ 80,00 será usado como base e será alterado de acordo com os parâmetros citados acima. Por exemplo, se o tipo de contratação for *Pessoa Jurídica*, esse valor já irá sofrer um desconto de 20%.

Para cada um dos parâmetros, têm-se as seguintes possibilidades de escolha:

- *Tipo de Contratação*: Pessoa Física e Pessoa Jurídica;
- *Sexo*: Feminino e Masculino;
- *Data de Nascimento*: o usuário informará sua data de nascimento, que será enquadrada em uma faixa etária;
- *Módulos*: existe a possibilidade de o usuário escolher de um a três módulos, sendo Consulta, Terapias, Exames;

A ilustração pode ser observada no Autômato de Estados Finitos Adaptativo a seguir:

Tomando a linguagem *L = {[Pessoa Física | Pessoa Jurídica].[Feminino | Masculino].[Faixa 1-Faixa 10].[?Consulta ?Terapias ?Exames]}* e a máquina $M = (\{q_0, q_1, q_2, \dots, q_f\}, \Sigma, q_0, \delta, \{F_1, F_2, F_3\})$, onde $\sum$ = *{Pessoa Física, Pessoa Jurídica, Feminino, Masculino, Faixa 1 ... Faixa 10, Consulta, Terapias, Exames}* e $\delta: \{\delta(q_0, tipoC.F_1()) = q_1 ; \delta(q_0, \varepsilon) = q_0\}$ e

$F_1() = \{?\ q_x, \varepsilon, q_y\ ; -q_x, \varepsilon, q_y\ ;$
$+ *l, \varepsilon, l\ ; ?\ q_w, tipoC.F_1, q_z\ ; +q_z, sexo.F_2, l\},$
$F_2() = \{?\ q_x, \varepsilon, q_y\ ; -q_x, \varepsilon, q_y\ ; +*m, \varepsilon, m\ ;$
$+q_x, faixa.F_3, m\},$
$F_3() = \{?\ q_x, \varepsilon, q_y\ ; -q_x, \varepsilon, q_y\ ;$
$+*n, btnCalcular, *o\ ; +q_x, mod, n\ ; +n, \varepsilon, n\},$
temos o seguinte autômato inicial, ilustrado na Figura 4.

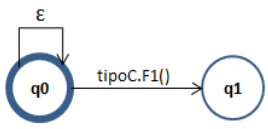


Figura 4: Autômato no estado inicial q_0.

O autômato simula o funcionamento da aplicação, onde começa com apenas dois estados, q_0, que é o estado inicial, onde o usuário ainda não fez uma interação e q_1, que é o estado para onde o sistema vai quando o usuário informa o tipo de contratação. Essa transação entre os estados q_0 e q_1 "*tipoC*" é dotada de uma função adaptativa $F_1()$, assim, ao escolher o tipo de contratação, o autômato vai sofrer uma adaptação por causa de $F_1()$, adquirindo um novo estado e permitindo a escolha de outro parâmetro pelo usuário, no caso, o sexo. Na Figura 5 está ilustrado o autômato no estado em que o tipo de contratação foi escolhido pelo usuário.

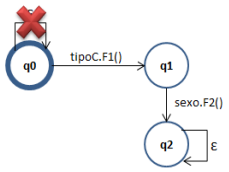


Figura 5: Autômato no estado q_1 da aplicação.

Estando no estado q_1, é possível a escolha do sexo. Essa transação entre os estados q_1 e q_2 "*sexo*" é dotada de uma função adaptativa $F_2()$, assim, ao escolher o sexo, o autômato vai sofrer uma adaptação por causa de $F_2()$, adquirindo um novo estado e permitindo a escolha de outro parâmetro pelo usuário, no caso, a data de nascimento, que será enquadrada em uma faixa etária. Na Figura 6 está ilustrado o autômato no estado em que o sexo foi escolhido pelo usuário.

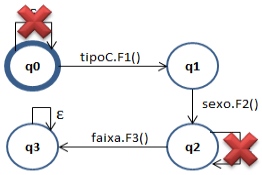


Figura 6: Autômato no estado q_2 da aplicação.

Estando no estado q_2, é possível a escolha da data de nascimento. Essa transação entre os estados q_2 e q_3 "*faixa*" é dotada de uma função adaptativa $F_3()$, assim, ao escolher a data de nascimento, o autômato vai sofrer uma adaptação por causa de $F_3()$, adquirindo um novo estado e permitindo a escolha de outro parâmetro pelo usuário, no caso, o módulo de cobertura e já permite o clique no botão Calcular após a escolha do módulo. Na Figura 7 está ilustrado o autômato no estado em que a data de nascimento foi informada pelo usuário.

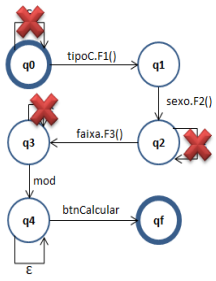


Figura 7: Autômato no estado q_3 da aplicação.

Após a escolha do módulo de cobertura, o autômato estará no estado q_4 e é possível clicar no botão Calcular para obter o valor da mensalidade.

VI. IMPLEMENTAÇÃO

Cada parâmetro escolhido/informado pelo usuário do sistema possui um valor definido nas anotações (*annotations*) e a aplicação montará uma equação em tempo de execução, que resultará no valor da mensalidade do plano de saúde escolhido. Na Figura 8 é possível verificar a definição da anotação *@RegraValor*.

```
@Retention(RetentionPolicy.RUNTIME)
public @interface RegraValor {

    String descricao();

    String valor() default "";

    float fator();

    float vInicial() default 0f;

    float vFinal() default 0f;

}
```

Figura 8: Anotação para definição de valores.

Essa anotação será usada como *metainformação* dos atributos da classe VO, que foi chamada de *CalculoVO*. É interessante verificar que essa anotação é dotada de uma anotação *@Retention(TetentionPolicy.RUNTIME)*, que significa que ela pode ser acessada através da reflexão em tempo de execução (*RUNTIME*). Se não for definida esta diretiva, a anotação não estará disponível através da reflexão.

Na Figura 9, está exemplificado a definição do atributo *dataNascimento*, com sua anotação:

```
@ListaRegras(regras = {
        @RegraValor(desc = "Dt Nascimento", vIni = 1f,  vFin = 18f, fator = 0.4f),
        @RegraValor(desc = "Dt Nascimento", vIni = 19f, vFin = 23f, fator = 1.1f),
        @RegraValor(desc = "Dt Nascimento", vIni = 24f, vFin = 28f, fator = 1.6f),
        @RegraValor(desc = "Dt Nascimento", vIni = 29f, vFin = 33f, fator = 2.0f),
        @RegraValor(desc = "Dt Nascimento", vIni = 34f, vFin = 38f, fator = 2.8f),
        @RegraValor(desc = "Dt Nascimento", vIni = 39f, vFin = 43f, fator = 3.0f),
        @RegraValor(desc = "Dt Nascimento", vIni = 44f, vFin = 48f, fator = 3.4f),
        @RegraValor(desc = "Dt Nascimento", vIni = 49f, vFin = 53f, fator = 3.9f),
        @RegraValor(desc = "Dt Nascimento", vIni = 54f, vFin = 58f, fator = 4.6f),
        @RegraValor(desc = "Dt Nascimento", vIni = 59f, vFin = 999f, fator = 5.6f)
private String dataNascimento;
```

Figura 8: Anotação no atributo dataNascimento.

Esse atributo *dataNascimento* é privado (*private*), por questões de encapsulamento, é uma *String*, e, possui *metainformações* em função da anotação *@ListaRegras*. Não é possível colocar mais de uma anotação do mesmo tipo em um atributo, por isso, na Figura 8 é possível observar que as anotações *@RegraValor* são elementos de um vetor regras, que é definido na anotação *@ListaRegras*. Assim, tem-se *metainformação* de uma *metainformação*. Isso foi necessário devido à regra de negócio que gira em torno desse atributo, que é a definição de valores diferentes para cada faixa etária e foi adotado um número de dez faixas etárias para exemplo.

Os elementos de *@RegraValor*, *descrição*, *valor inicial*, *valor final* e *fator* são usados para que o componente reflexivo realize os cálculos. Como metainformações são informações

das informações, quando o componente reflexivo recuperar o atributo *dataNascimento*, ele terá a possibilidade de recuperar também a anotação para realizar processamento.

Os demais atributos também estão declarados na classe *CalculoVO*. Na Figura 9 está a definição do atributo *tipoContratacao*:

```
@ListaRegras(regras = {
        @RegraValor(descricao = "Tipo de Contratação", valor = "F", fator = 1f),
        @RegraValor(descricao = "Tipo de Contratação", valor = "J", fator = 0.8f) })
private String tipoContratacao;
```

Figura 9: Anotação no atributo tipoContratacao.

Para esse atributo foram usados outros métodos da anotação *@RegraValor*. Não foram usados o *VIni* e o *VFin*, mas sim o *valor*. Então a parte do componente reflexivo responsável pela leitura de atributos com esse tipo de anotação, deve ser diferente da parte responsável pela leitura de um atributo como o *dataNascimento*, por exemplo.

O atributo *modulos* possui uma anotação semelhante a do atributo *dataNascimento*, mas ele é uma lista, portanto, o componente reflexivo deve ter outra parte responsável por ler uma lista.

O componente reflexivo recebeu o nome de *MensalidadeCalculator*, e pode ter uma parte vista na Figura 10:

```
import java.lang.reflect.Field;
import java.util.List;
import vo.CalculoVO;
import annotations.ContentValidator;
import annotations.ListaRegras;
import annotations.RegraValor;

public class MensalidadeCalculator {

    public static float mensalidade(Object o) throws Exception {

        float mensalidade = 80f;

        Class<?> klass = o.getClass();
```

Figura 10: Trecho da classe MensalidadeCalculator.

Como essa classe é a responsável por realizar a reflexão computacional, ela deve usar o pacote *java.lang.reflect*, e, o *.Field* serve para ela refletir os atributos. Ela recebe um parâmetro "*o*" do tipo *Object*, o que a torna adaptativa, pois, não importa se na chamada dela for passado um beneficiário, um aluno, um carro ou qualquer outra coisa, ela vai entender como um *Object* e vai realizar reflexão da mesma maneira, pois através do "*o.getClass()*" o componente realiza reflexão analisando o próprio código do programa em tempo de execução e recupera a classe do objeto.

A declaração da variável "*float mensalidade = 80f;*" vai servir como mensalidade básica, é ela que vai sofrer os cálculos ao longo do tempo de execução, na medida em que o componente reflexivo percorrer os atributos e suas anotações.

A partir do momento que o componente reflexivo recupera a classe do objeto, é possível também recuperar as anotações e os atributos definidos nessa classe, conforme Figura 11:

```
ListaRegras fieldAnnotation = field
        .getAnnotation(ListaRegras.class);

if (fieldAnnotation != null) {
    RegraValor[] regras = fieldAnnotation.regras();

    if (regras != null && regras.length > 0) {
        Object oValor = field.get(o);
```

Figura 11: Recuperação das anotações e atributos.

Esse trecho de código do componente reflexivo é responsável por recuperar a anotação *@ListaRegras*, verificar se existe essa anotação e então definir um vetor regras para obter as anotações *@RegraValor* e caso esse vetor for diferente de nulo e seu tamanho for maior que zero, o componente recupera os campos. Não importa se a classe do objeto possuir um ou mil atributos, o componente reflexivo vai ler todos automaticamente.

A partir dessa etapa, já é possível começar a trabalhar com os atributos recuperados em tempo de execução. Como dito anteriormente, é preciso criar partes diferentes no componente reflexivo para tratar os tipos de atributos que poderão ser obtidos. Uma parte da análise pode ser vista na Figura 12, onde o software verifica se o atributo é do tipo lista.

```
if (field.getType().equals(List.class)) {
    List<?> objects = (List<?>) oValor;
```

Figura 12: Verificação do tipo de atributo lido.

Após esse teste, o componente já realiza a verificação das metainformações obtidas através das anotações desse atributo, imprime no console da *IDE* de desenvolvimento Eclipse uma mensagem "*Regra encontrada:* ", que contém a descrição do atributo e seu fator para o cálculo, com fins de informação. Em seguida, ele atualiza o cálculo da mensalidade, conforme Figura 13:

```
for (RegraValor regraValor : regras) {
    if (regraValor.vInicial() != 0f
            && regraValor.vFinal() != 0f) {
        if (valorCalculo >= regraValor
                .vInicial()
                && valorCalculo <= regraValor
                        .vFinal()) {
            System.out
                    .println("Regra encontrada: "
                            + regraValor
                                    .descricao()
                            + " = "
                            + regraValor
                                    .fator());

            mensalidade += regraValor
                    .fator();
```

Figura 13: Atualização do cálculo da mensalidade.

Isso é feito para os todos os tipos de atributos citados. Esse método de programação, a programação reflexiva, permite ao software fazer uso da tecnologia adaptativa, pois, a classe *CalculoVO* pode sofrer inúmeras alterações por questões de regras de negócio e a classe *MensalidadeCalculator*, que é o componente reflexivo, não sofrerá mudança de uma linha sequer. Isso é muito importante para o programador, pois simplifica muito a manutenção do código do programa, otimizando seu trabalho.

Simulando uma alteração no software, é possível citar uma alteração nas faixas etárias do plano, por exemplo, de dez faixas para duas faixas. Para isso, basta realizar alteração apenas na anotação do atributo *dataNascimento* (diminuindo a quantidade de *@RegraValor* de dez para dois sobre o atributo), que o componente reflexivo vai se adaptar automaticamente. Isso vale também para um caso de alterações dos parâmetros, como por exemplo, a exclusão do atributo sexo, ou a criação de um atributo como *formadorOpiniao*, nenhuma linha do componente reflexivo precisará ser alterada, pois ele analisa a classe e recupera os atributos em tempo de execução, assim, a quantidade de atributos é irrelevante.

Cada campo é liberado apenas quando o anterior é informado, assim, o campo Sexo só será habilitado quando o campo Tipo de Contratação for informado e assim sucessivamente, como ilustrado no Autômato de Estados Finitos Adaptativo no Capítulo V.

Quando todos os parâmetros forem alimentados, o botão *Calcular* será liberado e será possível clicar nele para obter o valor da mensalidade que foi efetuado de acordo com as anotações definidas na classe *CalculoVO*.

VII. CONCLUSÃO

Durante o desenvolvimento deste trabalho foram apresentados conceitos e ferramentas que possibilitam implementar sistemas reflexivos, apresentando características, vantagens e desvantagens do modelo reflexivo de programação. A utilização da programação reflexiva permite a otimização do código fonte e a clareza das regras de negócio, podendo ser utilizada em diversos campos do desenvolvimento de softwares.

Foi desenvolvido um modelo de Autômato de Estados Finitos Adaptativos, que é muito utilizado em Teoria da Computação como modelo matemático que descreve máquinas automáticas, e, também foi desenvolvido um estudo de caso onde os conceitos da Tecnologia Adaptativa e Reflexão Computacional puderam ser bem aplicados. O módulo de um sistema de planos de saúde implementado ficou simples e objetivo, e, a parte teórica, tanto das tecnologias utilizadas quanto do Autômato de Estados Finitos Adaptativo, poderá servir como fonte de estudos para alunos de graduação e demais pesquisadores com o mesmo interesse.

REFERÊNCIAS

[1] PISTORI, Hemerson. Tecnologia Adaptativa em Engenharia de Computação: Estado da Arte e Aplicações. 191p. Tese (Doutorado) - Universidade de São Paulo, São Paulo, 2003.

[2] PAVAN, Willingthon. Tolerância a Falhas e Reflexão Computacional num Ambiente Distribuído. 86p. Dissertação (Mestrado) – Instituto de Informática - Universidade Federação do Rio Grande do Sul, Rio Grande do Sul, Porto Alegre, 2000.

[3] NETO, João José. Um Levantamento da Pesquisa em Técnicas Adaptativas na EPUSP. 2011. 25p. Revista de Sistemas e Computação, Salvador, v. 1, n. 1, p. 23-47, jan./jun. 2011.

[4] NETO, João José Um Levantamento da Evolução da Adaptatividade e da Tecnologia Adaptativa. Revista IEEE América Latina. Vol. 5, Num. 7, ISSN: 1548-0992, Novembro 2007. (p. 496-505).

[5] ROCHA, R. L. A. e Neto, J. J. Construção e Simulação de Modelos Baseados em Autômatos Adaptativos em Linguagem Funcional. Proceedings of ICIE 2001 - International Congress on Informatics Engineering, Buenos Aires: Computer Science Department - University of Buenos Aires, p. 509-521, 2001.

[6] BARTH, Fabrício Jailson. Utilização da Reflexão Computacional Para Implementação de Aspectos Não Funcionais Em Um Gerenciador de Arquivos Distribuídos. 2000. 90p. Monografia (Bacharelado em Ciência da Computação) - Universidade Regional de Blumenau.

[7] BRITO, Elcio Rodrigues. Desenvolvimento de Aplicações Comerciais em Java Usando Reflection. 2012. 36p. Monografia (Bacharelado em Ciência da Computação) – Fundação Educacional do Município de Assis– FEMA/Instituto Municipal de Ensino Superior de Assis - IMESA.

[8] SOUSA, Fabio Cordova de. Utilização da Reflexão Computacional Para Implementação de Um Monitor de Software Orientado a Objetos em Java. 2002. 53p. Monografia (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais – Universidade Regional de Blumenau.

[9] SENRA, Rodrigo Dias Arruda. Programação Reflexiva Sobre o Protocolo de Meta-Objetos Guaraná. 164p. Dissertação (Mestrado) – Instituto de Computação – Universidade Estadual de Campinas, Campinas, 2003.

[10] CORREA, Sand Luz. Implementação de Sistemas Tolerantes a Falhas Usando Programação Reflexiva Orientada a Objetos. 116p. Dissertação (Mestrado) – Instituto de Computação – Universidade Estadual de Campinas, Campinas, 1997.

[11] SOLIS, Daniel M. Illustrated C# 2010.Apress, 2010.

[12] FARTO, Guilherme de Cleva. Introdução à metaprogramação com Java Reflection API. Universidade Federal de São Carlos. Disponível em <http://www.slideshare.net/guilherme_farto/introduo-metaprogramao-com-java-reflection-api>. Acesso em: 31 Set. 2013.

[13] ZAGO, Adams Willians Alencr. Criando Anotações em Java. Universidade Presidente Antônio Carlos. Disponível em < http://www.devmedia.com.br/criando-anotacoes-em-java/22054>. Acesso em: 02 Out 2013.

Diego Augusto Passareli nasceu na cidade de Cândido Mota, São Paulo, Brasil, em 14 de Agosto de 1987. Possui graduação em Licenciatura Plena em Matemática pela Fundação Educacional do Município de Assis (2008) e é estudante do último ano de Bacharelado em Ciência da Computação na Fundação Educacional do Município de Assis (FEMA) em Assis, São Paulo, entre os anos de 2010 e 2013. Suas principais áreas de pesquisas são: Aplicação do Excel para modelagem de indicadores, Autômatos, Algoritmos e desenvolvimento de Aplicações Java.

Almir Rogério Camolesi possui graduação em Processamento de Dados pela Fundação Educacional do Município de Assis (1992), mestrado em Ciência da Computação pela Universidade Federal de São Carlos (2000) e doutorado em Engenharia de Computação e Sistemas Digitais pela Universidade de São Paulo (2007). Atualmente é professor titular da Fundação Educacional do Município de Assis. Tem experiência na área de Ciência da Computação, com ênfase em Tecnologias Adaptativas, atuando principalmente nos seguintes temas: Tecnologia Adaptativa, Computação Distribuída, Teoria da Computação, Modelagem Abstrata, Ensino a Distância, Algoritmos e Programação para Web.

Diomara Martins Reigato Barros possui Especialização em Sistemas de Informação pela Universidade Federal de São Carlos (1996) e graduação em Tecnologia em Processamento de Dados pela Fundação Educacional do Município de Assis - FEMA (1994). Atualmente é Analista de Sistemas e Professora da Fundação Educacional do Município de Assis (FEMA), no curso de Ciência da Computação, nas disciplinas de Teoria da Computação e Compiladores, no curso de Administração, na disciplina de Sistemas de Informação e no curso Análise e Desenvolvimento de Sistemas, na disciplina de Introdução a Computação. Possui experiência na área de Ciência da Computação, com ênfase em Sistemas de Computação atuando nos seguintes temas: Teoria da Computação, Compiladores e Tecnologia Adaptativa.

Um Metamodelo para Programas Adaptativos Utilizando SBMM

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

***Abstract* — *Model Driven Architecture* (MDA) é uma abordagem para desenvolvimento de software baseada na transformação sucessiva de modelos de alto nível até a geração do código-fonte. Utiliza tecnologias específicas mantidas pelo *Object Management Group* (OMG). Uma delas é o *Meta Object Facility* (MOF), que permite descrever metamodelos, ou seja, modelos das linguagens de modelagem de software tais como a UML. Os metamodelos são elementos chave para a descrição de transformações de modelos, base da MDA. O *Set Based Meta Modeling* (SBMM) é um formalismo que se propõe a ser uma alternativa mais simples em relação ao MOF e que, portanto, também permite descrever metamodelos. Este trabalho apresenta um metamodelo para programas adaptativos definido em SBMM. Este metamodelo descreve uma linguagem gráfica encontrada na literatura para representar programas adaptativos. Com isso, podem ser aplicadas técnicas, conceitos e ferramentas da MDA para a geração automática de código adaptativo a partir de modelos de mais alto nível.**

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

I. INTRODUÇÃO

MODEL Driven Architecture (MDA) é uma abordagem para desenvolvimento de software através da qual um software é construído através da transformação sucessiva e encadeada de modelos. Um modelo de um sistema é uma descrição ou especificação do mesmo considerando um determinado propósito. 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].

Em sua forma mais básica, a MDA propõe dois modelos: o *platform-independent model* (PIM), totalmente independente de plataforma, e o *platform-specific model* (PSM), dependente de plataforma [2]. A MDA propõe que o PSM seja gerado a partir do PIM através de uma transformação de modelos. Uma transformação é o processo de converter um modelo em outro modelo do mesmo sistema, conforme ilustrado na Figura 1.

Esta figura ilustra o esquema básico de uma transformação segundo a MDA: um PIM, junto com informações adicionais (quadro em branco), passa por um mecanismo de transformação para gerar um PSM.

As informações adicionais representadas pelo quadro em branco podem ser tanto para utilização do mecanismo de transformação (ex: mapeamentos, regras, templates, padrões, procedimentos) quanto informações extras para complementar o PIM e prover informações suficientes para gerar o PSM (ex: marcações). Isso se faz necessário, pois como o PIM não considera aspectos específicos da plataforma para a qual o sistema será gerado, mas o PSM considera, é preciso introduzir informações extras para cobrir os novos detalhes que aparecem.

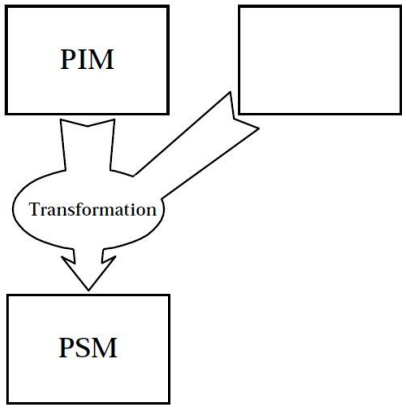


Fig. 1. Transformação de modelos segundo a MDA [1].

Uma próxima etapa de transformação a partir do PSM seria para a geração do código-fonte do sistema desejado. Este tipo de transformação é chamada de modelo-para-código (*model-to-code transformation*). Porém, se abstrairmos o conceito e considerarmos que o código-fonte também é um tipo de modelo textual, descrito em uma linguagem de programação, então esta nomenclatura tem pouca importância.

A MDA propõe diversos mecanismos para que a transformação seja realizada [1], podendo até mesmo ser manual. Porém, é claro que se deseja o máximo de automação possível no processo através da utilização de ferramentas de software, de forma que na situação ideal o PIM é suficiente para descrever o sistema por completo mas ao mesmo tempo se manter independente de plataforma. As informações adicionais necessárias para a transformação já estariam pré-definidas na forma de conjuntos de *templates*, regras e padrões disponíveis para aplicação no processo. Assim, implementações do mesmo sistema podem ser geradas automaticamente para diversas plataformas através de transformações específicas para diversos PSMs a partir do mesmo PIM.

Na situação ideal, qualquer manutenção no sistema se reduz a uma manutenção no PIM. Através da reaplicação da transformação com as informações adicionais já definidas, as implementações do sistema em cada plataforma são regeradas.

Desta forma, os arquitetos e desenvolvedores de software podem focar seus esforços em desenvolver e manter um

[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@gmail.com).

modelo conceitual abstrato da aplicação (PIM), que contém as regras de negócio e seus aspectos fundamentais. A obtenção de uma implementação para uma plataforma específica ocorreria rapidamente e de forma automatizada. Os investimentos das organizações em projetos de software seriam então preservados na medida em que um mesmo PIM, já criado, poderia dar origem automaticamente a novas implementações para outras plataformas que surgem no contexto da organização [3]. Mais interessante ainda, esta arquitetura permite gerar sistemas existentes através de seu PIM para novas plataformas que ainda estão por ser inventadas, sem necessidade de desenvolver novamente o sistema para essas plataformas.

A MDA não define plataforma, linguagem de programação ou *middleware* específicos. É um conceito abstrato dentro do qual se encaixam as ferramentas específicas. Atualmente a MDA é implementada por diversos fabricantes através de ferramentas que operam em cadeia para atingir os objetivos [4]. A MDA é mantida pela *Object Management Group* (OMG) [5].

A MDA possui como objetivos prover três benefícios principais através da separação dos modelos [1]:

- Portabilidade;
- Interoperabilidade;
- Reusabilidade.

Para a MDA funcionar na prática com transformações automáticas, é necessário especificar funções de mapeamento. Funções de mapeamento são descritas sobre metamodelos.

O *Set Based Meta Modeling* é um formalismo desenvolvido como parte desta pesquisa que se propõe a descrever metamodelos utilizando conceitos matemáticos fundamentais: conjuntos, relações e funções.

O objetivo deste trabalho é apresentar um metamodelo definido em SBMM que descreve uma linguagem gráfica de representação de programas adaptativos definida em [6]. A definição deste metamodelo é um passo para a criação de uma ferramenta CASE para programas adaptativos, ou então para a utilização de ferramentas meta-CASE. Ferramentas meta-CASE são capazes de gerar ferramentas CASE customizadas [14]. Metamodelos são entradas para essas ferramentas. Como visto anteriormente, outra importância do metamodelo é permitir especificar funções de mapeamento para transformações automáticas, inclusive geração de código-fonte.

Embora não faça parte do escopo deste trabalho abordar as funções de mapeamento, a existência do metamodelo formal viabiliza a escrita das mesmas, sendo outro passo importante para a geração automática de programas adaptativos a partir de um PIM, ou seja, para a aplicação da MDA na construção de programas adaptativos.

A seção II introduz a chamada arquitetura das quatro camadas, uma relação conceitual entre níveis ou camadas descritivas que faz parte do arcabouço teórico da MDA. A seção III apresenta a linguagem gráfica para representação de programas adaptativos definida por [6]. A seção IV descreve o *Set Based Meta Modeling*, formalismo para definição de metamodelos a ser utilizado, e aplica o SBMM para descrever o metamodelo proposto. A seção V discute os resultados, enquanto a seção VI apresenta as conclusões. Por fim, a seção VII lista as referências bibilográficas.

II. A Arquitetura das Quatro Camadas

A arquitetura das quatro camadas [15] é um modelo conceitual que contempla objetos de tempo de execução, modelos, metamodelos e um último nível conhecido informalmente como metametamodelo, conforme pode ser visto na Figura 2.

Muitas vezes fala-se sobre instâncias, classes, modelos, metamodelos, etc. de forma conjunta. Esta arquitetura permite classificar e visualizar precisamente onde cada elemento se encontra, bem como identificar suas relações horizontais e verticais, facilitando a leitura e tratamento desses elementos por máquinas.

A camada M0 contém os dados de um sistema em tempo de execução. No exemplo do sistema escolar, seriam objetos instanciados de CAluno, CCurso, etc. durante a execução do software. Registros em tabelas de um banco de dados relacional que armazenam esses objetos de forma persistente também pertencem a esta camada.

A camada M1 contém os modelos do sistema considerado: diagramas UML de classes, definições de tabelas de um banco de dados relacional, diagramas de casos de uso, códigos-fonte, etc. É o nível onde a modelagem do software ocorre e, portanto, onde trabalham os analistas e desenvolvedores. As transformações de modelos da Figura 1 ocorrem dentro deste nível.

A camada M2 contém os metamodelos, isto é, os metadados que capturam as linguagens de modelagem. Neste nível estão as definições da UML, ou seja, as especificações de seus diagramas e elementos: classes, atributos, operações, casos de uso, atores, etc. Definições sintáticas e semânticas das linguagens de programação também ocorrem nesse nível. Por isso, é aqui que atuam os desenvolvedores de linguagens. As funções de mapeamento devem ser definidas dentro deste nível, entre os metamodelos. Desta forma, como vimos, passam a ter utilidade geral pois podem ser aplicadas sobre instâncias quaisquer desses metamodelos (ou seja, os modelos de software do nível M1).

A camada M3 contém os "metametamodelos" que servem para descrever os metamodelos. Este nível contém apenas os conceitos mais simples requeridos para capturar modelos e metamodelos, sendo uma constante para suportar todas as possibilidades de modelagem das camadas acima. A OMG dedica esforços de padronização neste nível, e com isso criou o MOF [7]. O SBMM é uma alternativa ao MOF.

Uma confusão comum que se faz à primeira vista é tentar enxergar as transformações da MDA ocorrendo entre as camadas. Na verdade, as transformações da MDA ocorrem dentro da mesma camada. Em particular, as transformações de interesse para desenvolvimento de software (Figura 1) ocorrem dentro da camada M1. A relação entre as camadas é de descrição. O nível M3 possui elementos para descrever

metamodelos do nível M2, que por sua vez permitem descrever modelos do nível M1, que por sua vez permitem descrever os programas executáveis do nível M0.

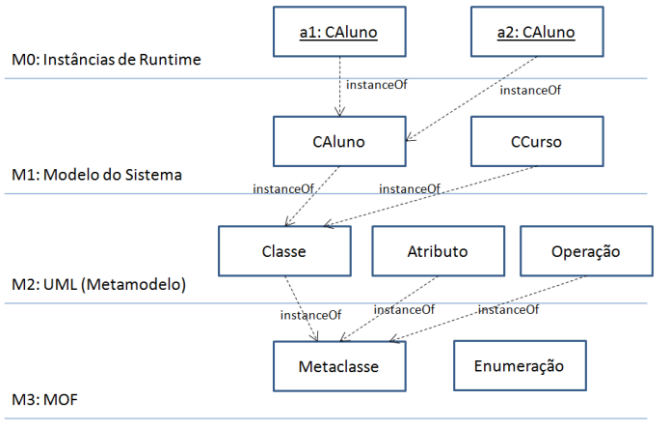


Fig. 2. A arquitetura das quatro camadas

Foi visto, e cabe ressaltar, que a utilidade prática da arquitetura das quatro camadas para a MDA está no fato de que cada camada serve para prover conceitos que permitem descrever instâncias da camada anterior. Por exemplo, os modelos da camada M1 descrevem programas cuja execução está na camada M0. Os metamodelos da camada M2 descrevem as linguagens de programação ou de modelagem que existem na camada M1.

Como as funções de mapeamento devem ser definidas sobre metamodelos, então a existência de conceitos e blocos básicos comuns na camada M3 (que são utilizados para definir diferentes metamodelos da camada M2) permitiria a definição de funções de mapeamento entre metamodelos através de técnicas comuns. Se surgirem novos metamodelos, ou seja, novas linguagens de modelagem, as mesmas técnicas ainda poderiam ser usadas para criar funções de mapeamento que atuam sobre modelos escritos nessas novas linguagens de modelagem. Daí a importância de existir uma especificação de linguagem para a camada M3 com sintaxe e semântica suficientemente simples para ser implementada e utilizada, mas ao mesmo tempo poderosa para permitir que novos metamodelos surjam conforme necessidades práticas.

III. Programas Adaptativos e Representação Gráfica

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 [11]. 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 [12]. Criação de novos estados e transições em tempo de execução são exemplos de autoreconfiguração.

Um programa adaptativo pode ser entendido como uma especificação de uma sequência automodificável de instruções, que representa um código dinamicamente alterável [6]. 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 [6]. 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 [6] gera código para o ambiente de execução desenvolvido em [13], 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 [6], 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 programa orientado a objetos.

Se for possível criar um metamodelo formal para essa representação gráfica de programas adaptativos, assim como existem metamodelos formais para os modelos UML, viabiliza-se a aplicação da MDA para a criação de código adaptativo a partir de PIMs e transformação de modelos, já que se estabelece a base para a definição de funções de mapeamento.

Desta maneira, podemos ter programas adaptativos descritos em um modelo com representação em alto nível, mesmo que parcialmente, e, futuramente, o mesmo modelo poderá ser usado para gerar código adaptativo para outras eventuais linguagens adaptativas e plataformas de execução de software adaptativo.

No restante desta seção descreveremos em linguagem natural a representação gráfica proposta por [6], para mais adiante criar um metamodelo formal.

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 3 [6] 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.

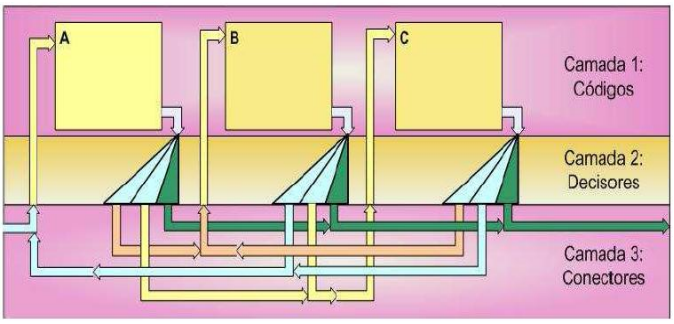


Fig. 3. Arquitetura de um programa não-adaptativo na forma de um dispositivo guiado por regras [6].

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 [6]. 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 3. 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 4 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 [6]. 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 [6].

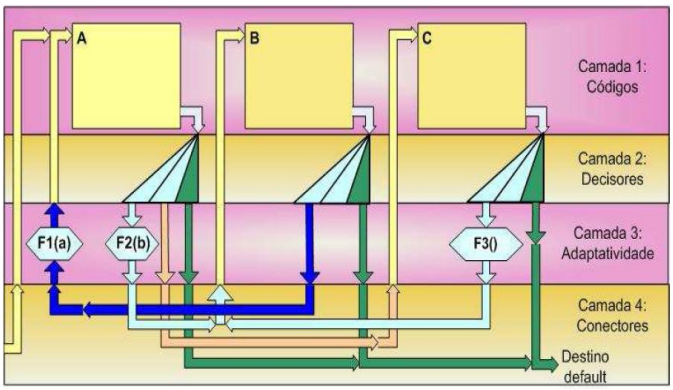


Fig. 4. Arquitetura de um programa adaptativo [6].

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

IV. Set Based Meta Modeling (SBMM)

Com a motivação de que o MOF apresenta uma definição com certo grau de complexidade [8], tais como ser descrito através dele mesmo em sua especificação *standalone* e prover capacidades a princípio desnecessárias na camada M3, aliado ao fato de quinze anos após sua publicação ainda não ser adotado em larga escala, pesquisou-se uma alternativa mais simples que deu origem ao SBMM. Trata-se de uma alternativa ao MOF para a camada M3, que permite definir metamodelos do nível M2 através da instanciação de conjuntos e relações pré-definidas bastante simples. Teve também inspiração em ideias de outros trabalhos [9][10].

Um metamodelo nomeado consiste em um nome (cadeia de símbolos) n, um conjunto C de metaclasses nomeadas, um conjunto Γ de generalizações e um conjunto E de enumerações nomeadas. Ou seja, um metamodelo é uma quádrupla MM:

$$MM = (n, C, \Gamma, E) \quad (1)$$

Onde:

- $n \in \Sigma^+$, 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 n 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 na camada M2, de forma análoga ao que representa uma classe em uma linguagem de programação orientada a objetos na camada M1.

• $\Gamma \subset C \times C$ é uma relação transitiva livre de ciclos que mapeia submetaclasses em suas supermetaclasses, representando o conceito de herança da orientação a objetos. É transitiva, pois se c_1 é subclasse de c_2 e c_2 é subclasse de c_3, então c_1 é subclasse de c_3. Não é uma relação reflexiva, pois uma metaclasse não é submetaclasse nem supermetaclasse dela mesma. Por não permitir ciclos e nem reflexão, então Γ é subconjunto próprio de $C \times C$.

• $E = \{e_1, ..., e_m\}$é o conjunto finito, eventualmente vazio, de enumerações.

Um metamodelo, e portanto suas metaclasses, existem na camada M2. As definições sobre metaclasses, generalizações e enumerações existem na camada M3. Uma instância de uma metaclasse é chamada de elemento de modelo, e esses existem no nível M1, onde trabalha o desenvolvedor de software.

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:

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

Onde:

• $w_i \in \Sigma^+$ é a cadeia de símbolos que nomeia c_i.

• $P_i = \{p_{i1}, ..., p_{in}\}$ 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:

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

Onde:

• $v_{ij} \in \Sigma^+$ 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 nossos propósitos a definição apresentada aqui é suficiente. Para tornar a notação mais semelhante a da UML e MOF, um par ordenados de multiplicidade (x,y) também será 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:

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

Onde:

• $u_i \in \Sigma^+$ 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, podemos definir uma enumeração de um tipo básico booleano como:

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

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:

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

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

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

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 teorias de conjuntos e de linguagens, para facilitar definições poderão ser utilizadas enumerações com infinitos valores permitidos, tais como:

$$e_4 = (\textit{“Integer”}, \boldsymbol{Z}) \quad (8)$$

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:

$$e_5 = (\textit{“Real”}, \boldsymbol{R}) \quad (9)$$

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.

A. Metamodelo em SBMM para Programas Adaptativos

Para facilitar o entendimento do metamodelo proposto, as sentenças que o compõem serão apresentadas ao longo das explicações.

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

$$MM_{PA} = (n, C, \Gamma, E) \quad (10)$$

Inicialmente define-se n = “AdaptiveProgramMetamodel” como o nome do metamodelo. Introduz-se então no conjunto E duas enumerações de tipos básicos e_1 e e_2, já discutidas anteriormente.

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

Essas enumerações serão referenciadas nas metaclasses.

Conforme mostrado na Figura 4, 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.

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

Um bloco básico é composto por um nome e um código-fonte na linguagem hospedeira, ambos representados como as propriedades do tipo String p_{11} e p_{12}. Neste trabalho não se entrará 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 4.

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.

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

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} são *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.

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

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 [6].

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 [6]. 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.

- 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)

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.

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.

Por fim, cria-se a metaclasse c_0, que estabelece os blocos de entrada e saída do programa adaptativo, conforme especificado em [6]. 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.

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 4 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.

Em resumo, os conjuntos principais 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}

O SBMM propõe uma notação gráfica similar a do MOF e UML, onde 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 5.

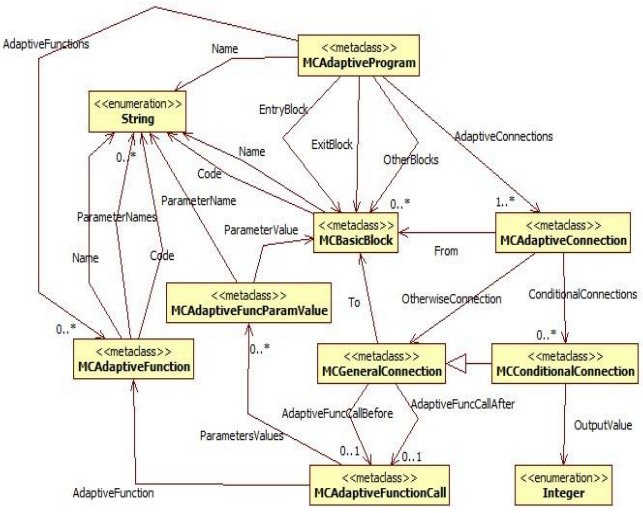


Fig. 5. Notação gráfica do metamodelo.

V. Resultados e Discussão

A seção anterior apresentou o resultado da descrição de um metamodelo para uma representação de programas adaptativos utilizando SBMM. A descrição resultante é puramente conceitual, totalmente independente de qualquer tipo de implementação. Além disso, provê um formalismo que captura os elementos da essência da definição de programas adaptativos. É extensível, ou seja, se forem utilizadas extensões na definição de programa adaptativo, o metamodelo pode ser ajustado através da substituição, remoção ou inserção de elementos que reflitam formalmente a definição modificada.

Um dos trabalhos de pesquisa atuais é o desenvolvimento da SBMM Tool (SBMMT), ferramenta capaz de prover edição de modelos genéricos de acordo com metamodelos em SBMM. O presente trabalho viabiliza a utilização futura da SBMMT para edição de modelos de programas adaptativos, podendo em um próximo passo prover geração automática de código BADAL. Também fica encaminhada a potencial geração de código em outras futuras eventuais linguagens de programação adaptativas a partir dos mesmos modelos de origem. No entanto, um PIM descrito no metamodelo proposto será capaz de gerar código adaptativo em uma ou mais linguagens 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 SBMMT ou outra ferramenta que permita o programador editar um modelo na forma da representação gráfica do programa adaptativo 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 PIM contenha informações completas sobre sua implementação sem depender de nenhuma linguagem específica.

VI. Conclusão

O presente trabalho mostra a viabilidade e caminhos para utilizar uma técnica muito poderosa, a MDA, para a criação de programas adaptativos. A MDA é considerada por alguns a grande tendência futura da engenharia de software, embora atualmente pouco explorada na prática [2]. Programas adaptativos representam uma nova abordagem para resolver problemas. O caminho apontado pelo trabalho é relevante na medida em que traz os ganhos propostos pela MDA para a área. Isso ocorreria por meio de ferramentas CASE para programas adaptativos com possibilidade de gerar código, nas quais os modelos serviriam não só como documentação, mas também como artefatos de construção dos programas adaptativos. Na situação ideal, funções de mapeamento podem ser criadas para gerar o programa para distintas plataformas ou linguagens a partir de um mesmo modelo. A possibilidade da utilização conjunta de ambas as técnicas é, portanto, de interesse não só para a engenharia de software como também para os campos onde programas adaptativos podem ser aplicados.

VII. Referências

[1] OMG, MDA Guide Version 1.0.1. 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] OMG, Model Driven Architecture Executive Overview. Disponível em: http://www.omg.org/mda/ executive_overview.htm.

[4] OMG, Model Driven Architecture FAQ. Disponível em: http://www.omg.org/ mda/faq_mda.htm.

[5] Object Management Group web site. Disponível em: http://www.omg.org/

[6] 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.

[7] OMG, Meta Object Facility (MOF) Core Specification. Disponível em: http://www.omg.org/spec/MOF/2.4.1

[8] Bézivin, J., Gerbé, O., Towards a precise definition of the OMG/MDA framework. In: Automated Software Engineering, 2001. Proceedings. 16th Annual International Conference on (pp. 273-380). IEEE.

[9] Favre, J. M., Towards a basic theory to model Model Driven Engineering. In: 3rd Workshop in Software Model Engineering, WiSME, 2004.

[10] Alanen, M., Porres, I., A Relation Between Context-Free Grammars and Meta Object Facility Metamodels, Technical report, Turku Centre for Computer Science, 2003.

[11] 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.

[12] 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.

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

[14] 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.

[15] Mellor, S., Scott, K., Uhl, A., e Weise, D., MDA Distilled: Principles of Model-Driven Architecture. Boston: Pearson Education Inc., 2004.

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. Vem desenvolvendo pesquisas sobre redes de controle, redes de sensores sem fio, eletrônica embarcada e aplicações de computação pervasiva.

Towards an Adaptive Internet of Things Architecture

L. B. Campos and C. E. Cugnasca

***Abstract*— Internet of Things (IoT) is a new communication paradigm establishing full communication between people and any object over the Internet. IoT has attracted the attention of researchers around the world and has opened new lines of research, among them, IoT Architectures. But apart from the different visions of its architects, the current proposals do not allow adaptive. Thus, this paper presents an adaptive-based approach for obtaining layered architectures of IoT. The approach is based on dynamic mapping of the requirements in an adaptive architecture. This technique shows the application of the concepts of adaptive in a developing area and helps to guide IoT Architectures.**

***Keywords*— Adaptive Architecture, Internet of Things, Requirements, Guidance Architecture.**

I. Introdução

INTERNET DAS COISAS (*Internet of Things* – IoT) é um paradigma que tem movido a sociedade em uma direção onde coisas e pessoas estarão interconectados [1]. Perspectivas futuras apontam que em 2015 haverá três vezes mais aparelhos conectados por meio da Internet do que pessoas [15]. Dessa forma, IoT tem atraído atenção de vários centros de pesquisas ao redor do mundo e gerado um pico de expectativas, conforme relatório recente emitido pela Gartner's IT [16]. Entretanto, toda convergência de esforços em busca de soluções inovadoras tem gerado o surgimento de várias abordagens e visões da IoT.

De acordo com Atzori (2011) o estado atual da IoT é a convergência de três diferentes visões [5] e que interferem diretamente em seus projetos arquiteturais, são elas: centrada nas coisas, centrada na Internet e centrada nos serviços. Na visão centrada nas coisas um objeto pode ser rastreado no espaço e no tempo ao longo da sua vida e que seja identificável unicamente [6]. Na visão centrada na Internet o principal foco é colocar o Protocolo IP sobre "qualquer coisa", ou seja, a IoT será implantada por meio de uma espécie de simplificação do IP atual para adaptá-lo a qualquer objeto e fazer esses objetos endereçáveis e acessíveis a partir de qualquer local [7][8][9]. Por fim, a visão centrada na semântica, também conhecida como *Web of Things*, preocupa-se com a representação, armazenamento, pesquisa e organização das informações geradas pelas coisas na IoT [10].

L. B. Campos, Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA), Vitória da Conquista, Bahia, Brasil, leonardobcampos@ifba.edu.br

C. E. Cugnasca, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, carlos.cugnasca@gmail.com

Dessa forma, é possível afirmar que as arquiteturas para IoT dependem diretamente dos serviços que serão prestados aos usuários e as exigências e necessidades dos interessados (*stakeholders*) nestes serviços. Ou seja, comumente, as arquiteturas para IoT refletem uma das três ou mais visões citadas e concentram-se em contemplar suas principais funcionalidades.

Nesse sentido, o presente artigo apresenta uma abordagem para concepção de arquitetura para IoT de forma adaptativa. A partir dos requisitos levantados pelos *stakeholders* é apresentada uma abordagem adaptativa para mapear uma arquitetura em camadas de IoT. O resultado obtido foi um autômato adaptativo que guia a obtenção da arquitetura. Além disso, a arquitetura adaptativa é capaz de atender as necessidades e interesses dos *stakeholders*. Ou seja, uma arquitetura capaz de modificar seu próprio comportamento como resposta aos dados de entrada, sem a interferência de agentes externos [14].

Para melhor compreensão deste artigo a seção 2 apresentará as principais arquiteturas de IoT existentes atualmente. Em seguida, o método para obtenção da arquitetura adaptativa para IoT será apresentada na seção 3. Os trabalhos relacionados a arquiteturas IoT, bem como adaptatividade em arquiteturas serão discutidos na seção 4. Por fim, na seção 5 são sumarizadas as conclusões e os trabalhos futuros.

II. Arquiteturas de Internet das Coisas

Segundo o Cluster Europeu de Pesquisa sobre Internet das Coisas (*European Research Cluster on the Internet of Things* – IERC) a IoT é uma infraestrutura de rede global dinâmica, com capacidade de autoconfiguração com base em protocolos de comunicação padrão e interoperável onde "coisas" físicas e virtuais têm identidade, atributos físicos, personalidades virtuais, usam interfaces inteligentes e são perfeitamente integradas em uma rede de informação [2].

Em diversos cenários de aplicação da IoT, as principais tecnologias que estão tornando esta revolução possível é conhecida como Identificação por Radiofrequência (*Radio Frequency IDentification* – RFID) e Redes de Sensores Sem Fio (*Wireless Sensors Network* – WSN). No geral, sistemas que usam RFID possuem, essencialmente, etiquetas (*tags*), leitores, aplicações de software, hardwares e middlewares [3]. Por outro lado, uma Rede de Sensores Sem Fio possui um grande número de sensores em atuação em uma área delimitada, sendo que a principal utilidade destes é captar informações do ambiente em que estão inseridos e comunicar-se entre si [4].

Devido a abrangência deste novo paradigma e a busca de soluções ideais sem a devida padronização inúmeras visões de Arquiteturas para IoT estão sendo apresentadas. Entre as principais visões encontradas na literatura destacam-se as seguintes:

1) Centrada nas Coisas: visão amplamente utilizada atualmente que envolve o rastreamento de objetos através de tecnologias como RFID, WSN, *Near Field Communication* (NFC).
2) Centrada nos Dados: visão focada na coleção e processamento de dados em tempo real. Bem como extrair informações úteis a partir de grandes volumes de dados (Big Data) [11].
3) Centrada na Internet: corresponde a construção de protocolos IP para permitir que objetos inteligentes conectem-se à Internet.
4) Centrada na Nuvem: pode ser vista como a união entre a visão centrada nas coisas e a visão centrada na Internet. O objetivo final é ter objetos inteligentes *Plug in Play* (PnP) que podem ser implantados em qualquer ambiente com um *backbone* interoperável que permite interagir com outros objetos inteligentes os eu redor [12].
5) Centrada nos Serviços: conhecimentos e serviços são definidos e formulados utilizando modelos de ontologias tornando as redes de grande escala mais cognitiva e viável [13].
6) Centrado na Semântica: aborda as questões de gerenciamento de dados que surgem no contexto das grandes quantidades de informação que se troca por objetos inteligentes, e os recursos que estão disponíveis através da interface web.
7) Centrada no Usuário: o usuário é colocado no centro e permitir que ele use o conhecimento e a infraestrutura para desenvolver novas aplicações.

Devido a relação entre as visões e seus conceitos, este trabalho agrupa as sete visões apresentadas em três, de acordo com [5], são elas: centrada nas coisas, na Internet e nos serviços. Acreditamos que a visão centrada na Internet pode agregar as visões centradas nos dados e na nuvem. Por outro lado a visão centrada nos serviços contempla as visões centradas na semântica e no usuário. Entretanto, reconhecemos a existência de particularidades e diferenças entre todas as visões.

III. AIoTA: Adaptive Internet of Things Architecture

A Arquitetura Adaptativa para Internet das Coisas (Adaptive Internet of Things Architecture – AIoTA) é baseada em quatro camadas, são elas: Camada Física, Camada de Interconexão, Camada de Dados e Camadas de Serviços, conforme Figura 1.

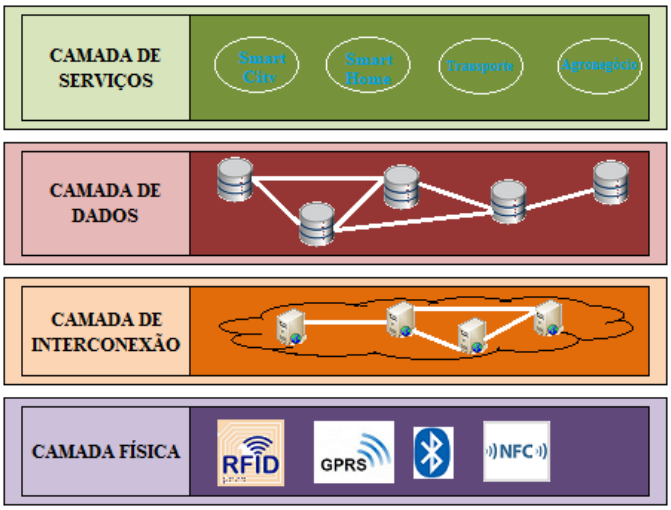


Figura 1. Arquitetura em quatro camadas da IoT

Antes de apresentar o autômato adaptativo que guiará a obtenção da arquitetura adaptativa é descrito nas subseções a seguir definições e detalhes dos módulos contidos em cada uma das camadas.

A. *Camada Física*

A camada física está focada nos objetos físicos, leitores, sensores, identificadores e atuadores. Na visão centrada nas coisas esta camada comumente recebe maior detalhamento. Os módulos próprios desta camada são:

- ID Único: diante do grande número de coisas (objetos e pessoas) que poderão estar conectado à Internet, na ordem de trilhão, e a possibilidade de rastrear/monitorar cada um deles unicamente, serviço conhecido como rastreabilidade, faz-se necessária a identificação única de cada coisa inserida na IoT;
- Segurança: módulo que assegura a troca confiável de dados entre o leitor e a etiqueta de identificação das coisas;
- Heterogeneidade: módulo responsável em viabilizar a comunicação entre diferentes padrões de comunicação criados para comunicação entre coisas e leitores;
- Smart Object: módulo correspondente às coisas que possuem sensores integrados como sensores de temperatura, umidade, acidez do solo, etc;
- Smart Tags: módulo responsável por coordenar a comunicação entre coisas, para tanto, as etiquetas devem ser inteligentes e possibilitar comunicação de coisas entre si;
- Rastreamento: módulo utilizado por coisas que possibilitem ser localizadas em qualquer lugar a qualquer momento. Módulo comumente associado a tecnologias como geoposicionamento e georeferenciamento.
- Sensor Network: módulo responsável pela integração da comunicação entre *tags* e redes de sensores sem fio.

B. Camada de Interconexão

Na camada de interconexão o foco é possibilitar a integração de Intranets das Coisas. Entende-se por Intranets das Coisas redes confinadas com serviços/aplicações próprias de IoT, que faltam apenas a integração com as demais redes com serviços/aplicações IoT. Esta camada é explorada em visões de IoT centrada na Internet e na nuvem. Veja a seguir detalhes dos módulos previsto na arquitetura adaptativa:

- Conectividade: módulo que possibilita a conexão de redes confinadas com a IoT;
- Interoperabilidade: módulo responsável pela integração de diferentes bases de dados, padrões de comunicação entre sistemas, qualidade de dados armazenados, etc;
- Escalabilidade: módulo que permite a conexão de um número crescente de coisas à rede.

C. Camada de Dados

A camada de dados é responsável pelo armazenamento e conversão de dados em informação e inteligência. Estão previsto três módulos principais, são ele:

- Rastreabilidade: esse módulo é responsável por registrar os dados de temperatura, responsável, localização, etc associado ao produto em cada etapa do processo produtivo, por exemplo. Registrar corretamente os dados nessa etapa influenciará diretamente na certificação das coisas;
- Banco de Dados: o banco de dados caracteriza-se pela elevada massa de dados a ser armazenada. Nesse sentido, este módulo prevê um sistema de data warehouse e data mining para facilitar a geração de relatórios gerenciais e agrupar os dados relevantes para diferentes consultas;
- Processamento Inteligente: módulo destinado transformar as informações armazenadas nas bases de dados em inteligência e facilitar a tomada de decisões (*Bussiness Intelligence – BI*).

D. Camada de Serviços

A camada de serviços é a mais próxima aos clientes de serviços da IoT. Esta camada é responsável em gerenciar os dados processados na camada de dados. Os serviços prestados podem ser voltados para casa inteligente, cidades inteligentes, transporte inteligente, etc. As cinco camadas presentes nesta camada são:

- Web service: módulo responsável pela disponibilização dos serviços na Web;
- Cloud computing: módulo que especifica os serviços correspondentes ao armazenamento e processamento de informações na nuvem;
- Usabilidade: requisito de qualidade contemplado neste módulo para permitir fácil acesso aos serviços disponíveis;
- Portabilidade: módulo responsável pela adaptação de conteúdos nos mais diversos aparelhos de acesso a Internet;
- Eficiência: módulo responsável pelo desempenho dos serviços;

Ainda nesta camada é possível prevê serviços como autenticação e certificação de usuários na rede. O módulo de autenticação é usado para autorizar o acesso aos dados e serviços disponível. O módulo de certificação é capaz de fornecer em tempo real mensagens certificadoras dos processos consultados.

E. Adaptatividade da Arquitetura IoT

A adapatatividade proposta neste trabalho para a arquitetura de IoT inicia-se com o levantamento dos requisitos, veja Tabela I. Em seguida, faz necessária a definição da relevância destes requisitos em quatro níveis: Alta, Média, Baixa e Não se Aplica (NA).

TABELA I
REQUISITOS DA ARQUITETURA PARA PREENCHIMENTO DA SUA RELEVÂNCIA

REQUISITO	DETALHAMENTO	RELEVÂNCIA			
		A	M	B	NA
Q_0	LER OBJETOS/COISAS E/OU AMBIENTE DE FORMA UBÍQUA	X			
Q_1	IDENTIFICAR UNICAMENTE CADA OBJETO/COISA	X			
Q_2	MONITORAMENTO DO AMBIENTE ATRAVÉS DE SENSORES		X		
Q_3	OBJETOS INTELIGENTES CAPAZES DE MONITORAR ALGUM DADO DO AMBIENTE		X		
Q_4	OBJETOS COMUNICANDO ENTRE SI SEM *GETWAY*	X			
Q_5	OBJETOS RASTREÁVEIS TODO O TEMPO	X			
Q_6	SUPORTE A DIFERENTES TECNOLOGIAS DE LEITORES		X		
Q_7	TROCA DE DADOS SEGURA ENTRE LEITORES, ETIQUETAS E SENSORES		X		
Q_8	INTERCONEXÃO DE OBJETOS/COISAS E/OU SISTEMAS DE INFORMAÇÃO ATRAVÉS DA INTERNET	X			
Q_9	TROCA DE DADOS ENTRE SISTEMAS DE INFORMAÇÃO DE DIFERENTES PARCEIROS	X			
Q_{10}	QUANTIDADE CRESCENTE DE OBJETOS MONITORADOS	X			
Q_{11}	ARMAZENAMENTO PERSISTENTE DE DADOS RELACIONADOS AOS OBJETOS/COISAS E/OU AMBIENTE	X			
Q_{12}	RECUPERAÇÃO DE TODO O HISTÓRICO DE INFORMAÇÕES		X		
Q_{13}	UTILIZAÇÃO DE DADOS ARMAZENADOS PARA TOMADA DE DECISÕES ASSOCIADAS AO OBJETO	X			
Q_{14}	SERVIÇOS PRESTADOS AOS *STAKEHOLDERS* PARTICIPANTES DO MODELO DE NEGÓCIOS	X			
Q_{15}	INTERFACES DO SISTEMA DE FÁCIL USO	X			
Q_{16}	ALTO DESEMPENHO PARA CONSULTAS E SERVIÇOS		X		
Q_{17}	SERVIÇOS ACESSÍVEIS A QUALQUER DISPOSITIVO CONECTADO A INTERNET		X		

Para determinar a arquitetura de IoT é apresentado um autômato adaptativo (AA), veja Figura 2, cujos estados representam os requisitos apresentados na Tabela 1. Caso o requisito tenha sido classificado com alta ou média relevância para a arquitetura, a transição assume o valor 1 (verdadeiro), caso contrário, caso tenha sido classificado com pequena ou não se aplica, assumirá 0 (falso). A topologia inicial deste AA é apresentado na Figura 3.

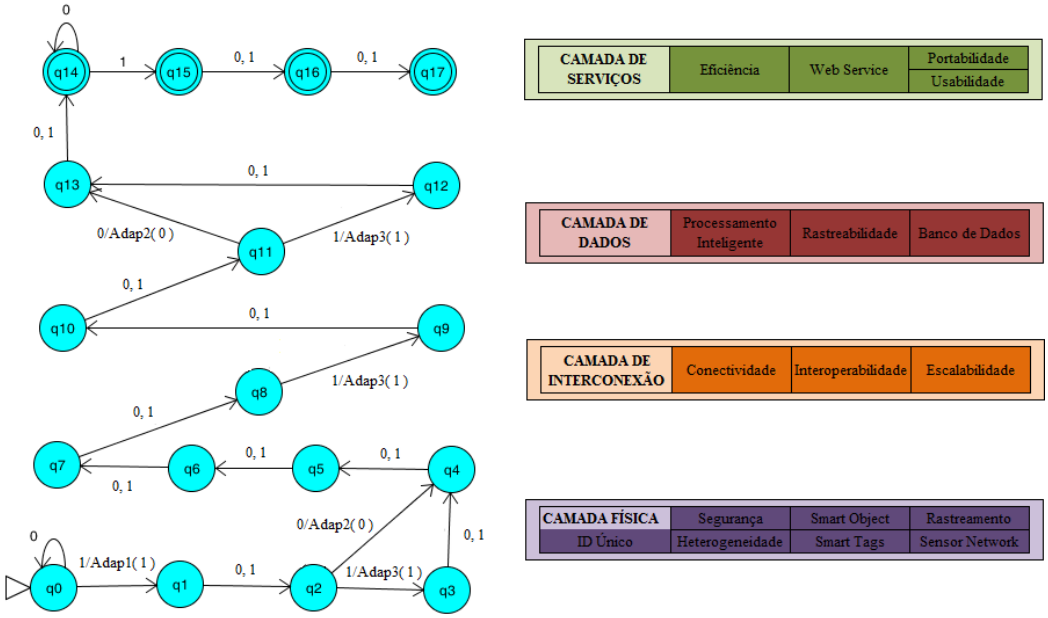


Figura 2. Mapeamento dos requisitos em uma arquitetura em camadas para IoT

Em q_0 o autômato busca garantir a leitura de objetos e/ou monitoramento de ambientes de forma ubíqua. Portanto, a primeira ação adaptativa do AA só será executada com a garantia de que o requisito q_0 será incorporado na arquitetura.

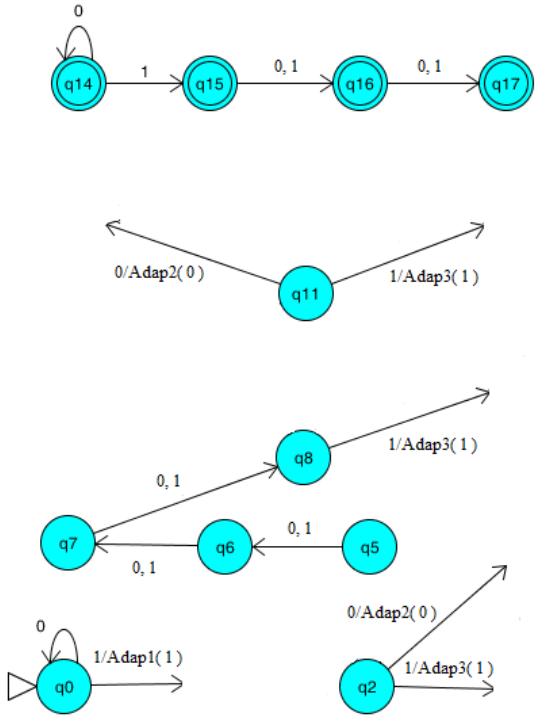


Figura 3. Topologia inicial do autômato adaptativo.

Ao todo o AA possui 3 (três) funções adaptativas, são elas Adap1, Adap2 e Adap3. A função Adap1 adapta funcionalidades da camada física. Ao ser executada é criado um novo estado q1 correspondente ao requisito Q1. A função Adap1 pode ser representada, em notação algébrica, como:

Adap1(1): (1)
f, g*{ (2)
– [(f , 1) - > q0, Adap1(1)] (3)
+ [(f , 1) - > g] (4)
+ [(g, 0) - > q_2] (5)
+ [(g, 1) - > q_2]} (6)

A semântica da função adaptativa Adap1 pode ser entendida da seguinte forma:

Linha (1): O cabeçalho especifica o nome da função (Adap1) e o valor da sua transição que ativa a função;
Linha (2): f é uma variável local, cujo valor é definido durante a execução de Adapa1, neste caso, f valerá q_0. Enquanto g é o nome de um gerador, que fornecerá a cada vez que a função A for executada, diferentes nomes para um novo estado a ser criado;
Linha (3): Elimina-se a transição que parte do estado q_0 e chega no próprio estado q_0, consumindo o símbolo 1;
Linha (4): Adiciona-se uma transição que chega ao estado q1 após consumir o símbolo 1;
Linha (5): Adiciona-se uma transição consumindo o símbolo 0, partindo do estado q_1 e tendo como destino q_2;
Linha (6): Adiciona-se uma transição consumindo o símbolo 1, partindo do estado q_1 e tendo como destino q_2;

A função Adap2 é responsável por consumir o símbolo 1 e criar um novo estado e duas transições (0 e 1) partindo deste novo estado. Por fim, a função Adap3 executa função semelhante a Adap2, entretanto, consumindo o símbolo 0.

IV. Trabalhos Relacionados

Apesar das diversas aplicações dos conceitos de adaptatividade o número de trabalhos que aplicam esta técnica em arquiteturas é pequeno. Em [17] é apresentado uma abordagem para desenvolvimento formal de especificações globais para programas dinamicamente adaptativos. São usadas máquinas de estados finitos para descrever formalmente o modelo proposto. Em [18] adaptatividade é usada para elevar capacidade de dispositivos heterogéneos comunicarem de modo a fazer o melhor da limitada disponibilidade de espectro e lidar com competição que é inevitável à medida que mais objetos conectarem ao sistema. O trabalho apresenta ainda um modelo conceitual que facilita a identificação de oportunidades para a adaptatividade, em cada camada da pilha de rede. Nesse mesmo sentido de [19] lista os fatores fundamentais que afetam a sua escalabilidade da Internet, e apresentam uma arquitetura adaptativa em direção do desenvolvimento potencial para o futuro da Internet, além disso, discute a adaptação dos seguintes aspectos: capacidade de gerenciamento e a confiabilidade.

V. Conclusões e Trabalhos Futuros

O trabalho apresentou uma abordagem adaptativa para concepção de arquitetura para IoT. Diante das inúmeras visões existentes na literatura para IoT as arquiteturas tendem a concentrar seus módulos em camadas dirigidas por essas visões privilegiando alguns serviços em detrimento de outros. Como trabalhos futuros espera-se alcançar uma arquitetura adaptativa que contemple os requisitos de qualidade especificados na norma ISO 25000.

Agradecimentos

Os autores gostariam de agradecer o apoio financeiro e de motivação fornecida pelo Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA), ao Prof. João José Neto pelas orientações que contribuíram no refinamento do artigo e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pela bolsa de pesquisa ao primeiro autor (142160/2013-0) e financiamento de projeto de pesquisa (485847/2013-2).

Referências

[1] J. Zheng, D. Simplot-Ryl, C. Bisdikian, and H. T. Mouftah, “The internet of things [Guest Editorial],” Communications Magazine, IEEE, vol. 49, pp. 30-31, 2011.

[2] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, et al., “Internet of things strategic research roadmap,”O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, et al., Internet of Things: Global Technological and Societal Trends, pp. 9-52, 2011.

[3] A. Cheung, K. Kailing, and S. Schonauer, “Theseos: a query engine for traceability across sovereign, distributed RFID databases,” in Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on, 2007, pp. 1495-1496.

[4] S. Glisic and J.-P. Makela, “Advanced wireless networks: 4G technologies,” in Spread Spectrum Techniques and Applications, 2006 IEEE Ninth International Symposium on, 2006, pp. 442-446.

[5] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” Computer Networks, vol. 54, pp. 2787-2805, 2010.

[6] B. Sterling, L. Wild, and P. Lunenfeld, Shaping things: MIT press Cambridge, MA, 2005.

[7] A. Dunkels and J. Vasseur, “IP for smart objects,” Ipso alliance white paper, vol. 1, 2008.

[8] J. Hui, D. Culler, and S. Chakrabarti, "6LoWPAN: Incorporating IEEE 802.15. 4 into the IP architecture," IPSO Alliance White Paper, vol. 3, 2009.

[9] N. Gershenfelo, R. Krikorian, and D. Cohen, “The Internet of Things-The principles that run the Internet are now creating a new kind of network of everyday devices, an Internet-O.”, Scientific American, vol. 291, pp. 46-51, 2004.

[10] I. Toma, E. Simperl, and G. Hench, “A joint roadmap for semantic technologies and the internet of things,” in Proceedings of the 3rd STI Roadmapping Workshop, 2009.

[11] C. C. Aggarwal, N. Ashish, and A. Sheth, “The Internet of Things: A Survey from the Data-Centric Perspective,” in Managing and Mining Sensor Data, ed: Springer, 2013, pp. 383-428.

[12] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” Future Generation Computer Systems, 2013.

[13] Y. Wang, P. Zeng, X. Wang, Y. Zhang, F. Kuang, and B. Zhu, “Information-centric Industrial Internet of Things: Service Model.”

[14] J. J. Neto, “Um Levantamento da Evolução da Adaptatividade e da Tecnologia Adaptativa”, Revista IEEE América Latina, vol. 5, (7), pp. 496-505, Nov. 2007.

[15] M. Presser, “Inspiring the Internet of Things,” Alexandra Institute, 2011.

[16] Gartner's 2013 Hype Cycle for Emerging Technologies < http://www.gartner.com/newsroom/id/2575515>

[17] Y. Zhao, Z. Li, H. Shen, and D. Ma, "Development of global specification for dynamically adaptive software," Computing, pp. 1-32, 2013.

[18] P. Du and G. Roussos, “Adaptive Communication Techniques for the Internet of Things,” Journal of Sensor and Actuator Networks, vol. 2, pp. 122-155, 2013.

[19] L. Chuang, J. Zixiao, and M. Kun, "Research on adaptive future Internet architecture," Chinese Journal of Computers, vol. 35, pp. 1077-1092, 2012.

Leonardo Barreto Campos nascido em Vitória da Conquista, Bahia, Brasil, graduou-se em Ciência da Computação na Universidade Estadual do Sudoeste da Bahia (2005), Mestre pela Universidade Federal de São Carlos (2007) onde iniciou sua pesquisa em Identificação por Radiofrequência (RFID) e em 2013 iniciou o doutorado na Escola Politécnica da Universidade de São Paulo (USP) no Laboratório de Automação Agrícola (LAA). Atualmente é professor adjunto do Instituto Federal de Educação, Ciência e Tecnologia da Bahia atuando nos seguintes temas: Internet das Coisas, Identificação por Radiofrequência, Redes de Sensores, Arquitetura de Software e Computação Pervasiva.

Carlos Eduardo Cugnasca é graduado em Engenharia de Eletricidade (1980), mestre em Engenharia Elétrica (1988) e doutor em Engenharia Elétrica (1993). É livre-docente (2002) pela Escola Politécnica da Universidade de São Paulo (EPUSP). Atualmente, é professor associado da Escola Politécnica da Universidade de São Paulo, e pesquisador do LAA – Laboratório de Automação Agrícola do PCS – Departamento de Engenharia de Computação e Sistemas Digitais 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).

Software Adaptável Para Cálculo de Satisfazibilidade Probabilística

C. E. de Barros and F. G. Cozman

***Abstract*— This extended abstract describes a project whose goal is to deploy adaptive techniques on a Probabilistic Satisfiability solver that uses the IBM-CPLEX package so as to exploit parallelization/concurrence in a cluster of computers. The current version of this software is being tested.**

***Keywords*— Satisfazibilidade Probabilística, Paralelização, Concorrência, Programação Inteira, Programação Linear**

I. INTRODUÇÃO

Resumidamente, o problema original da Satisfazibilidade Probabilística é uma generalização do problema da Satisfazibilidade: adicionam-se probabilidades a variáveis lógicas proposicionais [1]. O problema original pergunta sobre a existência de uma distribuição de probabilidades que satisfaz um conjunto de cláusulas, onde estas assumem valores falso ou verdadeiro, também de maneira probabilística. O problema da Satisfazibilidade Probabilística é de complexidade NP completa.

A primeira extensão desse problema é tratada no artigo [1] e a questão a ser resolvida é, então, o cálculo da faixa de variação do valor de probabilidade de uma nova afirmação lógica, dado o fato de que as informações anteriores já eram satisfazíveis.

II. TRABALHOS CORRELATOS

A Referência [2] propõe uma nova técnica para a resolução do problema original da Satisfazibilidade Probabilística e introduz o interessante estudo de transição de fase. Este mesmo estudo é posteriormente retomado pela Referência [3], onde é descrito software desenvolvido pelo Laboratório de Tomada de Decisão da Escola Politécnica.

III. QUESTÕES A SEREM TRATADAS

Primeiro, o software tal como apresentado em [3], resolve o problema original da Satisfazibilidade Probabilística, além de ter o enfoque no estudo de transição de fase, introduzido pela Referência [2]. Este software foi, então utilizado para fazer estudos mais exaustivos, com problemas envolvendo clausulas proposicionais com exatamente 3 literais.

Pretende-se estudar o benefício da utilização de computação paralela, utilizando o cluster do Departamento de Tecnologia da Informação que a Universidade de São Paulo Possui (o qual trabalha com 4 núcleos memória de 8 GB). Preliminarmente, notou-se a vantagem nativa tanto da linguagem Java, como do CPLEX, em utilizar os novos recursos de hardware.

IV. SITUAÇÃO ATUAL DO TRABALHO

Já está implementada a alteração de forma a atribuir os parâmetros de probabilidade, conforme [1], em que os principais aspectos são:

- Probabilidades associadas diretamente as cláusulas
- Numero de literais por cláusula variando uniformemente entre 1 e 3.

O programa em Java foi testado e aprovado quanto ao funcionamento para a utilização no cálculo da satisfazibilidade de *m* sentenças lógicas com probabilidades associadas a cada cláusula e número de literais variando de 1 a k.

O próximo passo é comparar seu desempenho em relação aos aspectos sugeridos por [1] e conforme o resultado, passar para a fase seguinte descrita, nos trabalhos futuros.

Para isso é necessária a realização de dois trabalhos preliminares. O primeiro é adaptar o programa atual para gerar problemas satisfazíveis, utilizando o método descrito no próprio artigo [1].

Outra importante mudança é programar o problema que é de fato tratado em [1], qual seja, a primeira extensão do problema original da Satisfazibilidade Probabilística. Ou seja, é preciso verificar a faixa de variação no valor da probabilidade de uma sentença adicional a um sistema conhecidamente satisfazível.

V. TRABALHOS FUTUROS

Tanto a linguagem Java como o pacote CPLEX nativamente já se beneficiam do poder de paralelização ou concorrência, ou combinação de ambos, oferecido pelo cluster utilizado no LCCA. No entanto é preciso fazer uma análise mais profunda de como estes e outros parâmetros da linguagem Java e do CPLEX podem melhorar o desempenho.

Normalmente, os parâmetros de configuração e entrada do CPLEX deverão ser passados de forma automatizada

pelo programa Java e isso será o próximo grande tema de estudo.

Também, em princípio, será utilizado como referência de desempenho o trabalho da referência [1], por ser considerado, até agora aquele que melhor desempenho demonstrou para a resolução do problema da Satisfazibilidade Probabilística.

Dentro dessa temática, pretende-se implantar técnicas adaptativas a serem especificadas, conforme os resultados obtidos após a conclusão das atividades preliminares descritas no andamento do trabalho.

REFERÊNCIAS

[1] P Hansen, S Perron. Merging the local and global approaches to probabilistic satisfiability.- International Journal of Approximate Reasoning, 2008 - Elsevier

[2] Marcelo Finger and Glauber De Bona. Probabilistic satisability: Logic based algorithms and phase transition. In IJCAI, pages 528{533, 2011.

[3] Cozman, Fabio Gagliardi ; di Ianni, L. F. . Probabilistic Satisfiability and Coherence Checking through Integer Programming. In: European Conference on Symbolic and Quantitative Approaches to Reasoning witih Uncertainty, 2013, Utrecht. Symbolic and Quantitative Approaches to Reasoning with Uncertainty, Lecture Notes in Computer Science, 2013. v. 7958. p. 145-156

AGRADECIMENTOS

Agradecemos o fundamental apoio das seguintes pessoas e instituições: LCCA – DVD – DTI – Reitoria – USP, Lucas Ianni, Fabio Reale, Glauber De Bona, Ettore Ligório, Francisco Ribacionka.

Celso Emidio de Barros é graduado em Engenharia Elétrica pela Universidade de São Paulo, Escola de Engenharia de São Carlos, São Carlos, SP, Especialista em Redes pelo Larc – Escola Politécnica – USP e aluno de mestrado em Engenharia Mecatrônica na Escola Politécnica.

Fábio Gagliardi Cozman é Professor Titular da Escola Politécnica da Universidade de São Paulo (Dept. Engenharia Mecatrônica) desde 2007, tendo ingressado na Escola Politécnica em 1990. Possui graduação em Engenharia Elétrica Modalidade Eletrônica pela Universidade de São Paulo (1989), mestrado em Engenharia pela Universidade de São Paulo (1991), Phd pela Carnegie Mellon University (1997), Livre-docência pela Universidade de São Paulo (2003). Atualmente é coordenador da comissão de inteligência artificial da SBC, membro do comitê editorial do Int. Journal on Approximate Reasoning e associate editor do Journal of Artificial Intelligence Research. Pesquisas focam na automação de processos de decisão sob incerteza, incluindo representação de conhecimento e aprendizado (tópicos: inteligência artificial, redes bayesianas, conjuntos de probabilidade, modelos estatísticos gráficos)

Análisis de métodos para el reconocimiento de patrones en ECG

O. Gawron, N. González, F. Lage
Universidad Tecnológica Nacional – Facultad Regional Buenos Aires - Argentina

***Abstract*— El presente trabajo tiene por objetivo analizar las herramientas existentes en el reconocimiento de patrones y evaluar su comportamiento para señales electro-cardiográficas (ECG). Dicho análisis permitirá realizar la implementación del procesamiento para un ECG isquémico.**
El reconocimiento de patrones es una técnica por la cual a través de la observación de la realidad, se identifica al objeto (problema), extrayéndose información que permita distinguir propiedades y su vinculación. Esta metodología conlleva la definición acerca de cómo se clasificarán los objetos. En ese sentido se diferencia la selección de variables, la clasificación supervisada y la clasificación no supervisada.
La detección de los eventos se realizará por medio de la búsqueda con patrones obtenidos de registros ECG típicos. Para la optimización de todos los procesos se utilizará el procesamiento paralelo.

Keywords*— *Reconocimiento de patrones, filtrado adaptativo, ECG, traza compleja.

I. Introducción

En el marco del proyecto "Procesamiento paralelo de señales electrocardiográficas aplicando traza compleja, para detección y clasificación de anomalías, por medio de patrones" (Universidad Tecnológica Nacional – Facultad Regional Buenos Aires) se ha logrado caracterizar a una señal de electrocardiograma canino a través de los atributos de la traza compleja [1].

El presente trabajo tiene por objetivo analizar las herramientas existentes en el reconocimiento de patrones y evaluar su comportamiento para señales electro-cardiográficas (ECG).
El reconocimiento de patrones es una técnica por la cual a través de la observación de la realidad, se identifica al objeto (problema), extrayéndose información que permita distinguir propiedades y su vinculación.

II. Marco Teórico

El reconocimiento de patrones involucra diferentes etapas. Se comienza con la observación de la realidad e identificación de un sistema físico, continua con la definición de un sistema de medición, la obtención y validación de los datos y finaliza con la modelización matemática a través de un modelo de reconocimiento adecuado. Existen diferentes enfoques en lo que respecta al reconocimiento de patrones, donde podemos distinguir el reconocimiento estadístico, el reconocimiento sintáctico, las redes neuronales y el reconocimiento lógico-combinatorio de patrones.

El reconocimiento de patrones conlleva la definición acerca de cómo se clasificarán los objetos. En ese sentido se destaca la selección de variables. Dentro de la selección de variables se puede trabajar sobre las características más adecuadas para la clasificación y/o el procesamiento.

III. Análisis de los métodos

Correlación cruzada

El método de reconocimiento de un patrón por medio de la correlación cruzada es tal vez el más intuitivo, consta de comparar el grado de similitud existente entre dos señales. Puede verse como una convolución gráfica entre dos señales, la cual será mayor cuanto más área compartan éstas entre sí.

Para el caso del reconocimiento de una cardiopatía, una de las señales es la obtenida del ECG y la otra es la que contiene el patrón buscado, de esta forma se compara cada uno de los datos de entrada (cada pulso de la señal de ECG) con el patrón deseado (pulso que contiene la patología buscada), en el caso en que coincidan perfectamente los pulsos analizados, la correlación será máxima (100%) y en consecuencia se establecerá que el paciente posee la patología analizada.

Debido a que una correlación de un 100% es prácticamente imposible, se establece cierto umbral de detección por encima del cual se determina que el paciente posee la patología en estudio, este umbral generalmente se toma entre el 85% y el 95%.

Esta técnica analiza directamente la señal de ECG y no las características extraídas de la misma como sí lo hacen las otras técnicas.

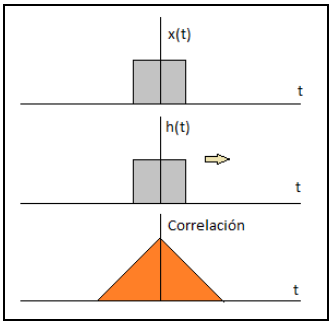


Figura 1 : Correlación de dos señales cuadradas

Como se puede ver en la *figura 1*, la correlación va a ser máxima cuando coincidan completamente las dos señales (x(t) y h(t)).

Supongamos ahora que la señal x(t) es la de ECG y la h(t) es una señal tal que contenga la patología buscada, entonces cuando coincidan completamente la función de correlación será máxima, pudiendo detectar de esta manera la presencia de la patología buscada.

Esta técnica de búsqueda no se utiliza ya que no es confiable debido a que es difícil encontrar un patrón exacto para cada patología.

Redes Neuronales – Backpropagation

La utilización de redes neuronales, es hoy en día el método más utilizado para el reconocimiento de patologías a través del análisis de señales de ECG debido a su sencilla implementación, su rápida respuesta, y demás está decir, sus buenos resultados de campo.

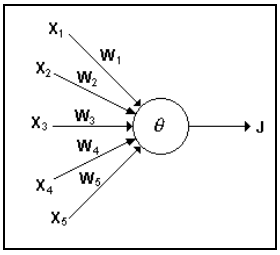


Figura 2 :Esquema básico de un red neuronal

El objetivo principal de una red neuronal es tener la capacidad de clasificar un dato de entrada, para lograr esto se deben encontrar los pesos (W) de la misma, y ello se hace realizando el entrenamiento de la red. Posteriormente se testea la red con un conjunto de señales conocidas, recién cuando se supera la etapa de testeo la red está lista para utilizarse.

Para el entrenamiento y posterior evaluación de la red se precisan dos conjuntos de datos de entrada/salida (uno para entrenar y otro para evaluar, en el caso del proyecto en curso las entradas serán los atributos obtenidos de aplicar las ecuaciones de traza compleja (amplitud en cuadratura, envolvente, fase instantánea y ancho de banda instantáneo) y las salidas serán las diferentes patologías cardíacas.

Trabajos realizados:

Yüksel Ösbay y Bekir Karlik [2] propusieron el entrenamiento de redes neuronales para cada una de 10 arritmias cardíacas propuestas en su trabajo y posteriormente unieron alguna de estas redes entre sí buscando optimizar la detección de las patologías, logrando un error del 4.3% en la detección individual y un 2.2% en la detección conjunta. Por su parte Hari Mohan Rai y Anurag Trivedi [3] propusieron en su trabajo la clasificación de las señales de ECG utilizando un una red neuronal alimentada con los atributos extraídos con la transformada Wavelet. En ambos trabajos utilizaron el banco de datos del MIT-BIH [4]. YU Sheng chen, HU Ying, YU Gui xian, JIN Xu ling, ZHANG Li nang y SHAO Tie jun, en su trabajo [5] proponen una red neuronal del tipo Back Propagation (BP) mejorada con algoritmos genéticos para la detección del la onda T de la señal de ECG, logrando resultados de detección del 98%.

Algoritmos Genéticos

Esta técnica surge como analogía de la evolución del ser humano y parte de una "población" de "individuos" (cada uno con su genotipo, o conjunto de características) que evolucionan y van modificándose por medio de diferentes agentes (selección, mutación, cruza, etc), las caractarísitcas de los individuos, llevadas a un lenguaje informático no son más que un conjunto de variables, cada una con un determinado valor. En cada generación de individuos los que se adapten mejor al ambiente, son los que tendrán mayor probabilidad de supervivencia, con lo cual generación a generación quedarán los más aptos. Una de las partes críticas en la utilización de los AG es la función de evaluación de aptitud (fitness function) la cual es la encargada de cuantificar qué tan apto es cada individuo para formar parte de la siguiente generación.

En la figura 3 se muestra el esquema básico al que responde la implementación de Algoritmos Genéticos.

Cabe destacar que el carácter probabilístico de los AG se encuentra precisamente en los agentes que modifican la población.

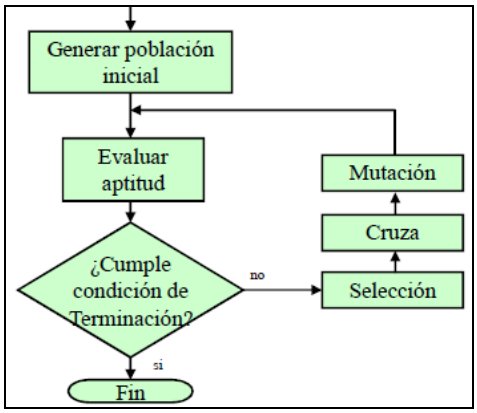


Figura 3 : Diagrama en bloques de la secuencia básica de los algortmos genéticos

Trabajos realizados:
Como se mencionó con anterioridad, generalemtne los trabajos en los que se utilizan algoritmos genéticos (AG) son aquellos en los que se necesita cierta optimización, esto se puede ver por ejemplo en el trabajo de Jalal A. Nasiri, Mahmoud Naghibzadeh, H. Sadoghi Yazdi, Bahram Naghibzadeh [7], quienes utilizan AG en conjunto con Máquinas de Soporte Vectorial (SVM), siendo la etapa de AG la encargada de mejorar el rendimiento en la generalización de los calsificadores de SVM. También se puede hacer meción al trabajo de YU Sheng chen [5] en el cual utilizan AG para optimizar la utilización de las redes neuronales implementadas.

Máquinas de soporte vectorial

Las Máquinas de soporte vectorial (SVM, siglas del inglés Support Vector Machine) son sistemas de aprendizaje que se entrenan utilizando un algoritmo basado en la teoría de la optimización. La clasificación de atributos mediante el método de SVM consta de encontrar una función capaz de separar (clasificar) datos de entrada. Mientras más mezclados estén los puntos de entrada, más compleja resultará la función a utilizar para su separación. Cuando no se logra encontrar una función que sea una combinación lineal (caso más usual) que separe los datos de entrada se realiza una transformación del espacio mediante un kernel, de esta forma una operación lineal en el espacio transformado es equivalente a una operación no lineal en el espacio sin transformar.

Trabajos realizados:
Aslı Uyar y Fikret Gürgen [8] clasifican un conjunto de arritmias a través de señales de ECG con Máquinas de Soporte Vectorial (SVM) y posteriormente presentan una comparación de la precisión de las SVM cuando se aplican diferentes kernels, la base de datos de arritmias utilizada en este trabajo es obtenida de la Unversidad de California (UCI) [9]. Narendra Kohli, Nishchal K. Verma1, and Abhishek Roy [10] comparan tres métodos SVM (uno-contra-uno, uno-contra-todos y funciones de decisión difusa) utilizados para detectar arritmias cardíacas.

Control Difuso

La lógica difusa hace mención a una método en el que los límites entre variables son borrosos, es decir la variación es gradual, debido a esto se obtiene una respuesta más suave.

El control difuso utiliza para la toma de decisiones a los sistemas basados en reglas (un ejemplo típico de una regla sería: "Si se cumple la *condición 1* y se cumple la *condición 2*, entonces realizar la *acción X*") y para la evaluación de estas reglas se utiliza fuzzy logic (lógica difusa).

Con esta técnica se logran sistemas de control robustos, sencillos, económicos y de rápida implementación. Esta técnica precisa de un experto que aporte su conocimiento para establecer las reglas y el peso de cada una de éstas. [11]

Trabajos realizados
PhanAnhPhong y KieuQuangThien proponen un sistema fuzzyTSK tipo 2, para poder distinguir entre el ritmo normal (NSR), la fibrilación ventricular (VF) y la taquicardia ventricular (VT) [12].

Learning Vector Quantization (LVQ), Self-Organizing Maps (SOM)

LVQ es la técnica la cual está basada en los Mapas Auto-organizados (SOM).

Un *mapa auto-organizado* (SOM) es un tipo de red neuronal artificial, que a diferencia de la red Backpropagation, ésta es entrenada usando un tipo de aprendizaje no supervisado y se utiliza para discretizar el espacio de entrada, al que se lo llama mapa. A diferencia de otras redes neuronales, los SOM utilizan una función de vecindad para mantener la topología del espacio de entrada.
La idea básica del SOM es que ciertas partes de la red reaccionenen de igual forma a ciertos estímulos de entrada. Algunas de sus ventajas son:

- Entrenamiento no supervisado.
- No precisa pares de entrada/salida, tan solo patrones de entrada.
- Simplemente se autoorganiza de forma autónoma para adaptarse lo mejor posible a los datos.
- Utilizados en el entrenamiento.

Trabajos realizados
M.H. Baig. A. Rasool,M.I. Bhatti uitilizan los métodos de LQV y de SOM para la clasificación de un electrocardiograma (ECG) en el reconocimiento de arritmias [13].

Filtrado adaptativo
Podemos definir un filtro digital como el proceso computacional mediante el cual una señal digital (conjunto de muestras) es transformada o alterada en su contenido frecuencial a fin de obtener ciertas características en particular. En el caso del filtrado adaptativo se busca modelizar la relación entre señales en tiempo real de forma iterativa. Este tipo de filtros se diferencia de los filtros digitales ya que sus coeficientes pueden variar en el tiempo de acuerdo a un algoritmo.
Dentro de los algoritmos adaptativos se encuentran lo de máxima pendiente, LMS (Least Mean Square), RLS (Recursive Least Square) y filtro de Kalman.

Trabajos realizados:

En función de remover la frecuencia de línea presente en la señal de ECG y resaltar las componentes de interes, Rehman y Kumar realizan una comparación de diferentes filtros adaptativos [15] implementando NLMS (normalize LMS). En la misma línea de trabajo Ju-Won Lee analiza en su trabajo [16] la influencia del ruido en la detección de los puntos Q y P. En ese sentido implementa un filtro LMS con estructura dinámica. Debido a la influencia de la frecuencia de red en la adquisición de señales biológicas, Chandrakar y Kowar [17] implementan un filtro RLS a fin de disminuir el nivel de ruido de línea y realizan una comparación de acuerdo a diferentes tipos de ruido en la señal de entrada.

IV. CONCLUSIONES

Se han analizado diferentes métodos para la clasificación de patologías mediante la observación de la señal de ECG. LVQ y SOM derivan de redes neuronales artificiales, por lo cual tanto las redes Backpropagation como LVQ y SOM tienen los mismos principios teóricos. AG se utiliza generalmente para optimizar cierta característica de otro método de clasificación.

Puede observarse en el trabajo de Václav Chudacek [14] el método de RNA Backpropagation tiene una precisión en la detección superior al 93%. Por otro lado en el trabajo presentado por YU Sheng chen [5] se obtiene una red de este tipo con una precisión del 98%. Si bien existen métodos con los que se obtiene mayor precisión, éstos son más complejos de implementar y requieren de un mayor poder de procesamiento. Se concluye entonces que por su simplicidad, rápida aplicación y los buenos resultados obtenidos en los trabajos de colegas, es que se opta por aplicar el método de RNA Bakpropagation para la siguiente etapa del proyecto, siendo factible además la realización de una comparación con otro método así como también la optimización de la red Backpropagation mediante la implementación de AG.

V. TRABAJOS A FUTURO

Como trabajos futuros, en el marco del presente proyecto se utilizará la traza compleja a fin de poder resaltar eventos en los datos provenientes de los electrocardiogramas (ECG) que puedan ser posteriormente asociados con enfermedades del corazón, es decir cardiopatías, de compleja detección con solamente un ECG.

La detección de los eventos se realizará por medio de la búsqueda con patrones obtenidos de registros ECG típicos.
A su vez, se encuentra en desarrollo un algoritmo que implementa filtrado adaptativo para mejorar la correlación de la distintas componentes de la traza compleja.
Para la optimización de todos los procesos se utilizará el procesamiento paralelo.

VI. REFERENCIAS

[1] Lage, Murana, Gawron, Cataldi. Nuevos atributos de la traza compleja. XIII Safety, Health and Environment World Congress – SHEWC'2013

[2] Yüksel Ösbay, Bekir Karlik, - A recognition of ECG arrhythmias using artificial neural networks – 2001 Proceedings of the 23rd Annual EMBS International Conference.

[3] Hari Mohan Rai, Anurag Trivedi - ECG Signal Classification using Wavelet Transform and Back Propagation Neural Network – 2012 5th International Conference on Computers and Devices for Comunications.

[4] Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng C-K, Stanley HE. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. Circulation 101(23) : e215-e220 [Circulation Electronic Pages; http://circ.ahajournals.org/cgi/content/full/101/23/e215];2000 (June 13).

[5] YU Sheng chen, HU Ying, YU Gui xian, JIN Xu ling, ZHANG Li nang y SHAO Tie jun, - ECG T Wave Detector Based on Neural Network Improved by Genetic Algorithms - 2010 Second WRI Global Congress on Intelligent Systems.

[6] Juan Carlos Gómez, Claudio Verrastro, Rodrigo Alcoberro. "Introducción a los algoritmos genéticos y susaplicaciones", Grupo de Inteligencia Artificial y Robótica. Universidad Tecnológica Nacional. Argentina

[7] Jalal A. Nasiri, Mahmoud Naghibzadeh, H. Sadoghi Yazdi, Bahram Naghibzadeh. ECG Arrhythmia Classification with Support Vector Machines and Genetic Algorithm. 2009 Third UKSim European Symposium on Computer Modeling and Simulation.

[8] Aslı Uyar, Fikret Gürgen - Arrhythmia Classification Using Serial Fusion of Support Vector Machines and Logistic Regression - IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications. 6-8 September 2007, Dortmund, Germany.

[9] University of California at Irvine (UCI) Machine Learning Repository http://archive.ics.uci.edu/ml/index.html.

[10] Narendra Kohli, Nishchal K. Verma1, and Abhishek Roy - SVM based Methods for Arrhythmia Classification in ECG - Int'l Conf. on Computer & Communication Technology –

[11] Ing. Juan Carlos Gómez. - Fuzzy Control - Grupo de Inteligencia Artificial y Robótica, Universidad Tecnológica Nacional – FRBA, Instituto Nacional de Tecnología Industrial – Electrónica.

[12] Phan Anh Phong , Kieu Quang Thien - Classification of Cardiac Arrhythmias Using Interval Type-2 TSK Fuzzy System - 2009 International Conference on Knowledge and Systems Engineering –

[13] M.H. Baig. A. Rasool,M.I. Bhatti - Classification of electrocardiogram using SOM, LVQ and Beat detection methods in localization of cardiac arrhytmias. - 2001 Proceedings of the 23rd Annual EMBS International Conference, October 25-28, Istanbul, Turkey.

[14] Václav Chudáček, Milan Petrík, George Georgoulas, Miroslav Čepek, Lenka Lhotská, Chrysostomos Stylios - Comparison of seven approaches for holter ECG clustering and classification - Proceedings of the 29th Annual International, Conference of the IEEE EMBS, Cité Internationale, Lyon, France, August 23-26, 2007.

[15] Syed Ateequr Rehman, R.Ranjith Kumar2. " Performance Comparison of Adaptive Filter Algorithms for ECG Signal Enhancement". International Journal of Advanced Research in Computer and Communication Engineering. Vol. 1, Issue 2, April 2012. ISSN 2278 – 1021

[16] Ju-Won Lee , Gun-Ki Lee. Design of an Adaptive Filter with a Dynamic Structure for ECG Signal Processing. International Journal of Control, Automation, and Systems, vol. 3, no. 1, pp. 137-142, March 2005

[17] Chinmay Chandrakar, M.K. Kowar. "Denoising ECG signals using adaptative filter algorithm". International Journal of Soft Computing and Engineering (IJSCE). ISSN: 2231-2307, Volume-2, Issue-1, March 2012

Semi-Global Alignment of Lines and Columns for Robust Table Extraction from periodic PDF Documents

Jorge Kinoshita,

***Abstract*—In a PDF document there is no meta information that describes a table structure; therefore, it is difficult to extract data from a table in PDF documents, mainly when the table is not isolated in a single column page. The extraction is harder when the table is in a multi-column page or it appears beside another table. Given a table tab1 (manually extracted from the document D1.pdf, ex: a balance sheet of 1998) and a PDF document D2.pdf, (ex: the annual report of 1999) we propose to automatically extract a table tab2 (ex: the balance sheet of 1999) inside D2.pdf through a dual (lines and columns) semi-global alignment between tab1 and D2.pdf, even when tab2 is in difficult places. This is specially useful for extracting data from financial reports because a table (e.g. a balance sheet) in a given year is very similar to the corresponding one in the next year. To our knowledge, this paper is the first to propose data extraction from tables in PDF documents through a dual semi-global alignment.**

I. Introduction

It is hard to extract data from tables in Portable Document Format (PDF) files because they have two kinds of information: text and images. All meta information such as where begins and ends a table, a column, a line, or even, a cell are lost when a PDF is generated. Tables (ex: a balance sheet) in financial reports documents, which are generally delivered in PDF, contains the same pattern from year to year. These tables may not differ very much because financial data are generally duplicated in two consecutive years: there are small differences such as some lines may be inserted or deleted, or yet, an item specification may be renamed. We took this feature as a hint to use the alignment between two consecutive financial reports in order to extract data from these tables. Many aligned tables may be merged in a single table in order to avail a company. We propose to manually mark a table tab1 (mark a rectangle, i.e., the upper left corner and the bottom right corner of a balance sheet) in the first document D1.pdf (figure 1, align tab1 with D2.pdf (Ex: a part of D2.pdf in figure 2), get tab2 (table I), align tab2 with D3.pdf, repeat it, get all the tables aligned and join all of them in a single merge-table. The main purpose of this paper is to show how to sequence and align strings from tab1 and D2.pdf by:

1) Retrieving all the strings from tab1. For the first document D1.pdf, we have to manually select a rectangle that specifies where tab1 is located in D1.pdf.
2) Retrieving all the strings from D2.pdf (see figure 2).
3) Sequencing and aligning the sequence of strings from tab1 with the sequence of strings of D2.pdf (see table I).

To our knowledge this is the first paper that uses a dual semi-global alignment in order to make a robust data extraction from PDF documents and therefore, deemed to be the best contribution of this paper. The section II contains related work. The section III contains our alignment algorithm used to extract table from a PDF file. The section IV shows test results. The section V finishes the paper enforcing its robustness.

II. Related Work

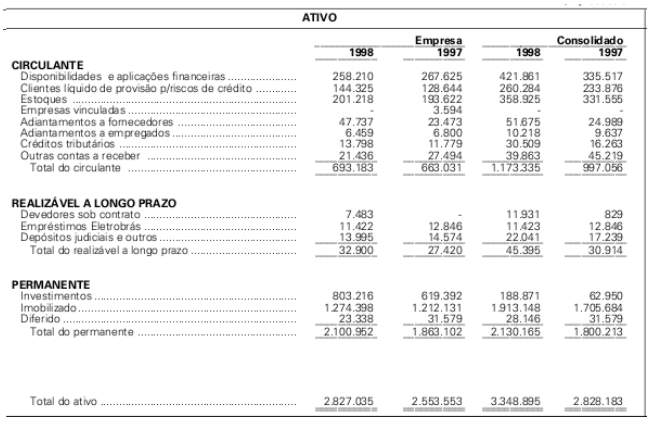
The papers were arranged in 3 areas: 1. Extraction of strings and tables from a single PDF document. 2. Extraction of tables from a series of web pages. 3. Alignment.

A. Extraction of strings and tables from a single PDF document

A PDF document is created to be printed. It contains strings and images. We found some programs that retrieve strings from PDF files in:

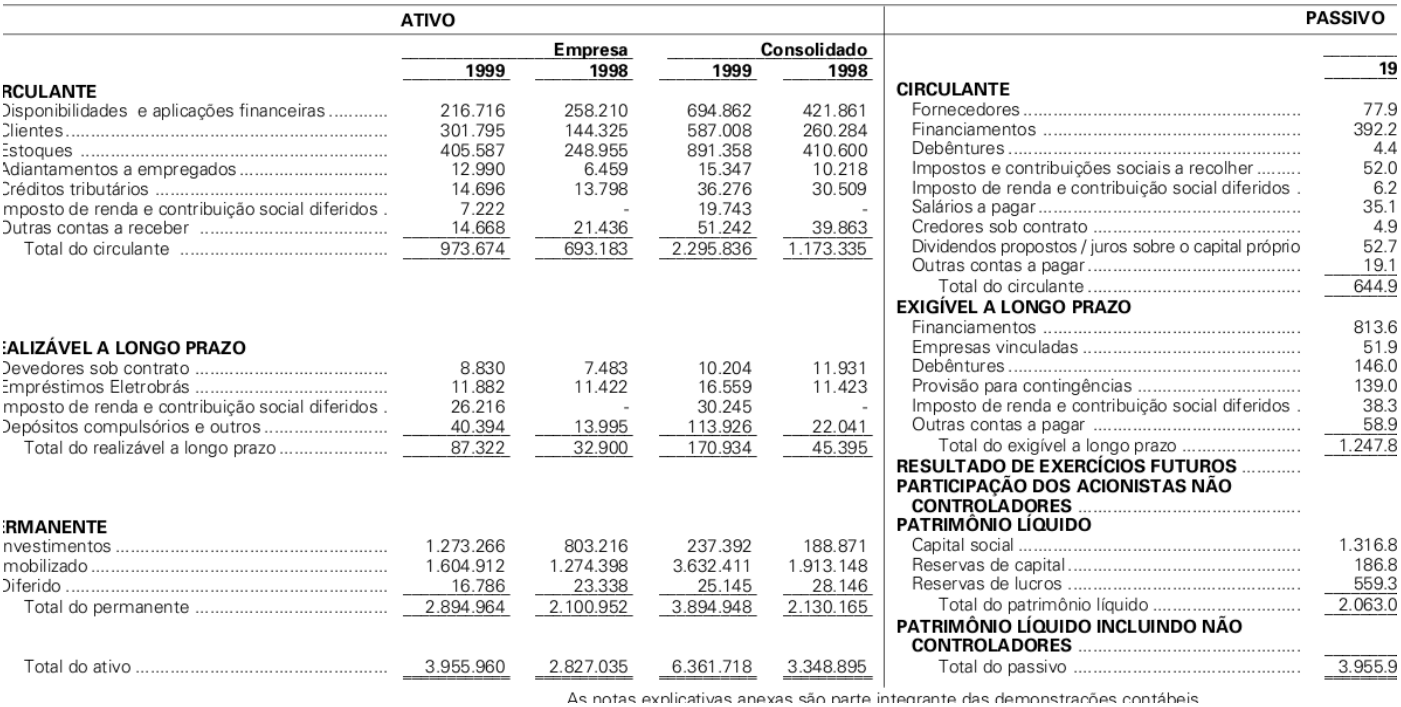
- a XML file, contains for each string, its position (page number, Cartesian coordinates, width and height). Ex: command: `pdftohtml -xml`.
- a free text with strings, spaces and line feeds that tries to keep their original geometric positions. Ex: `pdftotext - layout`.

Jorge Kinoshita is with the Department of Electrical and Computer Engineering, Polythenic School, Sao Paulo, Brazil



ATIVO	Empresa		Consolidado	
	1998	1997	1998	1997
CIRCULANTE				
Disponibilidades e aplicações financeiras	258.210	267.625	421.861	335.517
Clientes líquido de provisão p/riscos de crédito	144.325	128.644	260.284	233.876
Estoques	201.218	193.622	358.925	331.555
Empresas vinculadas	-	3.594	-	-
Adiantamentos a fornecedores	47.737	23.473	51.675	24.989
Adiantamentos a empregados	6.459	6.800	10.218	9.637
Créditos tributários	13.798	11.779	30.509	16.263
Outras contas a receber	21.436	27.494	39.863	45.219
Total do circulante	693.183	663.031	1.173.335	997.056
REALIZÁVEL A LONGO PRAZO				
Devedores sob contrato	7.483	-	11.931	829
Empréstimos Eletrobrás	11.422	12.846	11.423	12.846
Depósitos judiciais e outros	13.995	14.574	22.041	17.239
Total do realizável a longo prazo	32.900	27.420	45.395	30.914
PERMANENTE				
Investimentos	803.216	619.392	188.871	62.950
Imobilizado	1.274.398	1.212.131	1.913.148	1.705.684
Diferido	23.338	31.579	28.146	31.579
Total do permanente	2.100.952	1.863.102	2.130.165	1.800.213
Total do ativo	2.827.035	2.553.553	3.348.895	2.828.183

Fig. 1. tab1 - A piece of 1998-Gerdau-Report.pdf with the table ATIVO.



ATIVO	Empresa		Consolidado		PASSIVO	
	1999	1998	1999	1998		19
RCULANTE					**CIRCULANTE**	
Disponibilidades e aplicações financeiras	216.716	258.210	694.862	421.861	Fornecedores	77.9
Clientes	301.795	144.325	587.008	260.284	Financiamentos	392.2
Estoques	405.587	248.955	891.358	410.600	Debêntures	4.4
Adiantamentos a empregados	12.990	6.459	15.347	10.218	Impostos e contribuições sociais a recolher	52.0
Créditos tributários	14.696	13.798	36.276	30.509	Imposto de renda e contribuição social diferidos	6.2
mposto de renda e contribuição social diferidos	7.222	-	19.743	-	Salários a pagar	35.1
Outras contas a receber	14.668	21.436	51.242	39.863	Credores sob contrato	4.9
Total do circulante	973.674	693.183	2.295.836	1.173.335	Dividendos propostos / juros sobre o capital próprio	52.7
					Outras contas a pagar	19.1
					Total do circulante	644.9
					EXIGÍVEL A LONGO PRAZO	
ALIZÁVEL A LONGO PRAZO					Financiamentos	813.6
Devedores sob contrato	8.830	7.483	10.204	11.931	Empresas vinculadas	51.9
Empréstimos Eletrobrás	11.882	11.422	16.559	11.423	Debêntures	146.0
mposto de renda e contribuição social diferidos	26.216	-	30.245	-	Provisão para contingências	139.0
Depósitos compulsórios e outros	40.394	13.995	113.926	22.041	Imposto de renda e contribuição social diferidos	38.3
Total do realizável a longo prazo	87.322	32.900	170.934	45.395	Outras contas a pagar	58.9
					Total do exigível a longo prazo	1.247.8
					RESULTADO DE EXERCÍCIOS FUTUROS	
					PARTICIPAÇÃO DOS ACIONISTAS NÃO CONTROLADORES	
ERMANENTE					**PATRIMÔNIO LÍQUIDO**	
nvestimentos	1.273.266	803.216	237.392	188.871	Capital social	1.316.8
mobilizado	1.604.912	1.274.398	3.632.411	1.913.148	Reservas de capital	186.8
Diferido	16.786	23.338	25.145	28.146	Reservas de lucros	559.3
Total do permanente	2.894.964	2.100.952	3.894.948	2.130.165	Total do patrimônio líquido	2.063.0
					PATRIMÔNIO LÍQUIDO INCLUINDO NÃO CONTROLADORES	
Total do ativo	3.955.960	2.827.035	6.361.718	3.348.895	Total do passivo	3.955.9

As notas explicativas anexas são parte integrante das demonstrações contábeis

Fig. 2. D2.pdf - A piece of the 1999-Gerdau-Report.pdf with 2 tables: ATIVO and PASSIVO, to be aligned with the ATIVO table of the 1998-Gerdau-Report.pdf.

- a free text with strings that does not keep their geometric positions: Ex: `pdftotext`; `acrobat reader`.

For table extraction, the geometric information is very important. However, `pdftohtml` and `pdftotext` does not always work as expected by the user. The command `pdftotext - layout` committed errors such as deleting table cells, or even the whole table, which prevented us of using this software. The command `pdftohtml -xml` generally outputs one string to each table cell. However, in some cases, it outputs one string to two table cells (for example, it extracted the string "1998 1997" from table tab1) or two strings to one table cell. Although, `pdftohtml -xml` does not always work as expected, we considered it the best one to our purposes. In table extraction from a single PDF document, there is no information whether a string belongs to a table or to a text column; thus a large amount of heuristics may be necessary to automatically extract data from tables. Some papers ([7], [5],[3]) deal with recognizing tables from PDF documents. In general, first lines are recognized, then, columns; and finally tables are assembled. If a table were in a multi-column page or beside another table, this process can lead to many errors; for example, taking a text column as a column of the table; or yet, taking the

tables ATIVO and PASSIVO as a single one (see figure 2). We did not find any paper that deals with the table extraction from a series of PDF documents (such as financial reports).

B. Extraction of tables from a series of web pages.

Web pages, mainly an output from a cgi-script, where information is structured in a repetitive way may be aligned according to their tree structure (parsing their HTML trees and aligning them [8], [10], [9]) or no (as a flat text [1]). None of these papers mention the use of the semi-global alignment: the basic algorithm used in this paper. A closer paper is [8] that makes a partial alignment of trees when extracting data from the web. First, a web page is segmented (using also geometrical information) and then partially aligned. It does not discuss global, local and semi-global alignment as we do here. Our work, however, does not align trees because this meta information (HTML tags that describes tables) does not appear in PDF documents.

C. Alignment

Alignment is closed related to change, or to edit a string1 (a sequence of characters) into another string2 [2]. There are 4 edit operations: **replace** a character of string1 to another character in string2, **insert** a character in string2, **delete** a character of string1, **match** (copy) a character of string1 into string2. A weight is assigned to each operation, for example, match = +1; replace = -1; insert = -1; delete = -1. For example: There are many edit operations to change "CAT" into "BATOU" (example: delete C,A,T; insert B,A,T,O,U). The sum of the weight of each operation is called the similarity; in this case, -8. The alignment algorithm finds the edit operations that yields the highest similarity (ex: replace C,B; match A,T; insert O; insert U; with similarity = -1). A classical algorithm is based on dynamic programming which uses an alignment matrix (see [2]). There are 3 kinds of alignment: global, local e semi-global. An alignment algorithm receives two strings str1 and str2 as input. **Global alignment** finds the best similarity between str1 and str2. In the global alignment from [CAT] to [BATOU], the best similarity is -1. The Needleman–Wunsch algorithm resolves the global alignment using dynamic programming. **Local alignment** finds the best similarity between any substring of str1 and any substring of str2. In the local alignment from "CAT" to "BATOU", the best similarity is 2 (aligning C[AT] with B[AT]OU: match A,A; match T,T). In our case, there is no way to guarantee that any of these substrings is the whole tab1. The Smith–Waterman algorithm resolves the global alignment using dynamic programming. **Semi-global alignment** finds the best similarity between a string str1 and any substring of str2, i.e., it works well if we have a good idea of what we are looking for inside str2; that is, a balance sheet (tab2) as a substring of a financial report (D2.pdf). For example: The best similarity of the global alignment between the whole string CAT and any substring of BATOU is 1 (aligning [CAT] with [BAT]OU: replace C,B; match A,A; match T,T). Semi-global alignment is used mainly in biology, but we found few papers using it in computer engineering; the closer to our work is [6] that uses semi global alignment to aid a robot read text, because the camera can focus on a part of the text, however, it does not deal with PDF documents, neither table alignments.

III. Table Extraction using the Alignment of Lines and Columns

We decided to use an open source library called `poppler` in order to extract strings from a PDF document because it is a very mature software. The command `pdftohtml -xml` makes use of this library and generates a XML output that shows for each string its geometric position in the document page. We had to modify the library. Generally `poppler` outputs one string for each table cell. However, sometimes `poppler` makes two kinds of error: 1. join two table cells or 2. split one table cell in two strings. We changed the source code of the library `poppler` to show the second kind of error because it leads to errors that are easier to deal: it is easier to fix errors in a spreadsheet with more columns (because two strings derived from an original table cell are in two columns) than to deal with interwoven columns. From now on, we are calling "cell" to each string that comes from `pdftohtml -xml` independently of it being a real table cell. We did not use the term "string" because it has been used in a broader sense in this paper. The alignment algorithms (ex: global, local and semi-global) are made to align strings which are uni-dimensional structures. It is possible to use these algorithms depending on how strings are extracted from a PDF document. A global alignment can be done for two documents which are very alike, i.e., which have same sequence of topics and tables. The Needleman–Wunsch algorithm deals with string and characters; here we deal with document (mapped as strings) and lines or cells (mapped as characters). This algorithm works fine when the sequence of the line and cells of a whole D1.pdf aligns with the sequence of lines and cells of a whole D2.pdf, but there are many cases in which it does not happen: let us suppose D1.pdf is a single column document but D2.pdf (with almost the same

sequence of topics) is a double column document: how can we guarantee a sequence of lines and cells that aligns D1.pdf to D2.pdf? A tool such as `pdftohtml -xml` only extracts strings (taken as cells) and their geometric positions, thus we have to rely on the geometry in order to align both documents. Another problem is that there is no guarantee that the sequence of topics and tables happens for two documents. For example: It is common that tables float in a document. A table may be placed in the top or bottom of a page depending on the free space, the size of the table, the layout of the page, etc. Thus, it is difficult that two documents have the same sequence of topics and tables. A semi-global alignment algorithm is useful for searching a given table tab1 in a document D2.pdf. It has lesser restrictions than the global algorithm because the sequence of topics does not matter, but even still we may find problems: let us suppose that tab1 is side by side another table as it may happen in figure 2. In this case, it is not possible to guarantee that we may find all the cells of tab2 together, concentrated as a sub sequence of cells in D2.pdf, because the cells of the table ATIVO may be mixed to the cells of table PASSIVO as in D2.pdf (figure 2), preventing a good alignment. The solution presented in this paper consists of using their geometric positions to semi-globally align tab1 and D2.pdf even when tab2 is in a very difficult location such as besides another table. The basic idea is to find an anchor cell, i.e., a pair of cells: cell1 of tab1 and cell2 of D2.pdf where cell1 best aligns with cell2 in D2.pdf and through them build the alignment. In order to accomplish this, it is necessary: 1 - organize the cells of tab1 and D2.pdf as a sequence of lines and columns,(section III-A) 2 - make the semi global alignment between lines and columns, (section III-B) 3 - take the anchor cell III-C and 4 - based on this anchor-cell alignment and in the sequence of lines and columns of tab1 and D2.pdf, derive other cells alignments (section III-D).

A. Sequencing the cells of the PDF document in lines and columns.

There are 3 ways to sequence the cells of tab1 and D2.pdf:

1) by the order provided by `pdftohtml -xml` but this order is not reliable, sometimes the software outputs the strings in the human order reading, other times, does not.
2) by line: Taking the cells line by line, first from top to bottom and then, from left to right. The line-sequence for D2.pdf (figure 2) is: (..., L, *Disponibilidades e aplicações financeiras*, *216.716*, *258.210*, *694.862*, *421.861*, L, *Fornecedores*, *77.987*, etc.), i.e., the cells from 2 tables (ATIVO and PASSIVO) are mixed.
3) by column: Taking the cells column by column, from left to right and then, from top to bottom. The basic idea is to order the sequence of cells according to the X-coordinate, but this may not work. Columns may be right aligned (ex: column of numbers), left aligned (ex: items description) or center aligned. What X-coordinate should we take in order to sequence the cells according to the columns? The basic rule for the column alignment is: given 2 cells A and B, if there is an intersection between the X-coordinates of A and B, then, the upper cell comes first. If there is no intersection between the X-coordinates of A and B, then the left cell comes first. Example: sequencing tab1 according to the columns yields: (CIRCULANTE, "Disponibilidades e aplicações financeiras", "Clientes líquido de provisão p/riscos de crédito", ...).

We sequenced the cells of tab1 and D2.pdf and arranged as lines and columns, thus we may see tab1 and D2.pdf as a spreadsheet similar to figure 3.

B. Semi global alignment between lines and columns

The Needleman–Wunsch and Smith–Waterman ([2]) aligns two strings based on their characters. Here, we semi-globally align a sequence of cells cell1 (a line or a column) of tab1 to another sequence of cells cell2 of D2.pdf. In order to find the higher similarity between a line or a column, it is necessary to define the weights of the operations: replace/match cell1,cell2; insert cell2 and delete cell1.

We are assigning -1 to the delete and insert operations. For the replace/match operation, the weight will depend on the kind of cell1 and cell2. One table is composed of cells with letters (ex: *Fornecedores*) that generally is part of the table head and cells without letters, generally numbers. We want to align mainly the heads at tab1 and tab2, thus, we are going to assign a higher weight for the similarity of two cells with letters, but it depends on how similar cell1 and cell2 are. We define the weight of the match/replace operation, in this case, as:

-20 + (1 - distance($str1,$str2)/greater($str1,$str2))*40;

which yields a number between -20 and 20. where:

- greater($str1,$str2): length of the greater string: $str1 or $str2.
- distance($str1,$str2): edit distance between str1 and str2. The edit distance is a number that varies from zero (total match) to greater($str1,$str2).

The weight to replace/match a cell without letters (ex: *216.716*) to another without letter is 1. The weight to replace a cell with letter with a cell without a letter is -20 (therefore a delete or insert operation is performed).

Let us represent tab1 and D2.pdf as a spreadsheet in figure 3 where cells of tab1 are in columns A-G and the cells of D2.pdf are in columns H-N. Then we semi-globally align all pairs ((1,1),(2,1) ...) getting a line alignment score for each pair: line-score(1,1), etc. The same idea is done to align columns yielding: column-score(A,H),column-score(A,I), etc. In order to obtain a line-score, we assemble a semi-global alignment matrix with the cells in each line. For instance, for aligning (1,1) the matrix is created taking (A1, B1, C1, ... G1) x (H1, ... N1). The matrix is composed of pairs: (A1,H1), (A1,I1), etc. For each pair (A1,H1) we apply the weights for the operations: replace/match, delete and insert. The same procedure happens to find the column-scores.

We notice that our first idea was not to consider column-scores because it is possible to have some idea of the tab2 location using only line-scores. However this region is not very well defined. For instance, we take the line ("CIRCULANTE") in tab1 and align with the line ("CIRCULANTE", "CIRCULANTE") in D2.pdf where the first "CIRCULANTE" is from the ATIVO table and the second is from the PASSIVO table (see figure). It is not very clear which columns of D2.pdf corresponds to tab2. We thought of using some statistics and line alignments to discover where tab2 is placed; but an easier way to discover the tab2 location is to use column alignments too and through it, have an anchor cell, a cell1 in tab1 that aligns with a great confidence with a cell2 in D2.pdf.

C. Finding the anchor-cell: the best cell1 and cell2 alignment

A cell C9 belongs to a line 9 and a column C of tab1 (see figure 3). If both alignments 9-6 and C-K, aligns C9 to the same K6 cell in D2.pdf then we are going to define:

cell-score(C9,K6) = line-score(9,6) * column-score(C,K)

The anchor-cell may correspond to some special cells of tab1. In figure 2 there is a table which have the first column as item descriptions; all other cells are numbers. Due to the higher weights for aligning cells with letters (section III-B), the similarity of the first column will be higher, therefore, the anchor cell is taken from this column. Another example is a table which have an item description column and a line item description as heads. In this case, the anchor cell is very likely to be in the corner of these head line and head column. Example: The best anchor cell in the tab1-D2.pdf is 'Adiantamentos a empregados'. Through the line alignment 9,6 we obtain many other cell alignments; for instance (D9,L6), (E9,M6). This line alignment will be named the "guide line". Through the column alignment C,K we obtain many other cell alignments, for instance (C6,K3), (C7,K4). This column alignment will be named the "guide column". These other cell alignments will be used to generate the table alignment.

D. Generating the Table Alignment.

We may deduce other cells alignments such as E6-M3 in figure 3 through the guide line and guide column. Due to the 9-6 alignment, if E9 aligns with M6, then column E aligns with column M. Due to the C-K column alignment, if C6 aligns to K3, then line 6 aligns with line 3. Therefore, we deduce that E6 aligns to M3. Thus, many other alignments are extracted, even with empty cells (a cell is inserted or deleted). If there were no cell in the intersection of line 3 and column M; then cell E6 would be deleted in D2.pdf, i.e., E6 aligns to an empty cell. Example: Table I shows some alignments that were extracted through this method. This method of aligning cells avoids some inconsistencies. For example, as "Empresas vinculadas" of tab1 did not column-aligned with any cell of D2.pdf, all cells corresponding to the line "Empresas vinculadas" in tab1 are simply removed. Some cells of tab1 may not align with the proper cells of tab2 in D2.pdf because their lines or their columns do not cross the guide line or the guide column. For instance the line of "1998 1997" cell in tab1 does not cross the guide line. The cell alignments found in the former phase correspond to places in the line alignment matrix and in the column alignment matrix when using a dynamic algorithm for semi-global alignment. For instance, the column alignment matrix has a cell that corresponds to

string1	string2
CIRCULANTE	CIRCULANTE
Disponibilidades e aplicações...	Disponibilidades e aplicações
Clientes líquido de provisão ...	
	Clientes .
Estoques	Estoques
Empresas vinculadas	
Adiantamentos a fornecedores	
Adiantamentos a empregados	Adiantamentos a empregados
693.183	973.674
1.800.213	2.130.165
2.828.183	3.348.895

TABLE I. PARTIAL RESULT OF THE TAB1-D2.PDF ALIGNMENT. IN COLUMN STRING2, THERE ARE THE STRINGS OF TAB2.

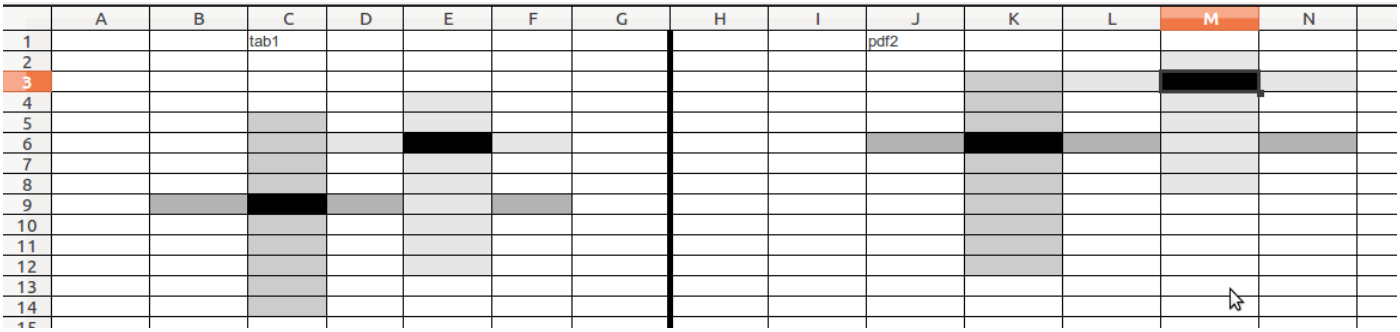


Fig. 3. Cell C9 of tab1 aligns to K6 of D2.pdf.

a reliable alignment of 258.210 in tab1 with 216.716 in D2.pdf. Based on this alignment, we can "stretch" the path, i.e., we may trace back through this alignment cell in the column semi-global alignment matrix discovering other alignments that could not be taken in the former phase. See chapter 11 of [2] for a reference about an alignment matrix.

IV. Tests

We applied the semi-global alignment to merge balance sheets from Gerdau. We expected that the algorithm would work fine in most cases. In the wrong cases, we expected that it would be easy to fix the merge-table through some macros of openoffice that merge or split lines and columns; but we had harder problems. We notice that:

- the semi-global alignment is very time and memory consuming because it assembles all matrices of lines and columns to all pages in a D2.pdf. In order to improve the algorithm, we automatically search for the page where the table is and apply the algorithm to this page only. We tested some mechanisms to automatically find the page that contains a given table. These mechanisms see pages and tables as bag of words. We concluded that it is better to use a ranker (pages that have more words of tab1) than a classifier (train a classifier to respond yes or no using pages that has the table and pages that has not the table. As table floats in the pages, the page text that contains the table is not very precise for training and testing).
- in some cases, the first or the last line of tab1 does not properly align to the correct line of D2.pdf. In our experiments, the first or the last line was not aligned to any line of D2.pdf. The consequence is that tab2 looses its beginning or its ending line becoming smaller than it should be. When aligning tab2 to a D3.pdf, the error is propagated. We tried some tricks to automatically fix this problem, but we noticed that the algorithm was becoming specialized in financial tables. Thus we decided to manually mark the beginning and ending of each table. This is the biggest problem in our algorithm because, although we always found a good anchor-cell in our tests, the automatic table extraction of many documents was not always well succeeded leading to error propagation.

V. Conclusion

To our knowledge, this is first paper to use semi global alignment to extract table (ex: a balance sheet from a certain year) from a series of PDF documents (ex: annual reports). We showed how to extract data from a balance sheet, even in very difficult cases: when a table is side by side another table or column. We accomplished it using two semi-global alignments: lines and columns. We intend to build a software to process PDF documents changing it as a sequence of lines and columns, enabling operations such as join lines, split columns, and even, building a merge-table. A critic to our work is that we tested the algorithm and we got some problems as in the section IV. To the future, we may find a way to mark the table region according to: 1. the geometry of the cells (a line or column of cells that are parallel to the discovered tab2). 2. lines and columns in the border of tab1 that were not aligned. In a former work [4], we show a method to align two strings of lengths m and n based on the global alignment algorithm but implemented using adaptative technology. It is seen that this algorithm could run in paralell and therefore, in a time proportional to m+n and not m*n. This result can easily applied to this work that uses the semi global alignment, a variant of the global alignment algorithm, however, [4] does not use semi global

References

[1] D. Guo, D. Li, M. Liu, Y. Liu, and S. Chen. The dynamic web pages information extraction algorithm based on sequence

alignment. *Computational Sciences and Optimization, International Joint Conference on*, 1:784–786, 2009.

[2] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, New York, NY, USA, 1997.

[3] T. Hassan. Object-level document analysis of PDF files. In *Proceedings of the 9th ACM symposium on Document engineering*, DocEng '09, pages 47–55, New York, NY, USA, 2009. ACM.

[4] J. Kinoshita and R. L. Rocha. ALINHAMENTO DE DUAS CADEIAS USANDO UM TRANSDUTOR ADAPTATIVO. In *Memórias do WTA'2009 : terceiro Workshop de Tecnologia Adaptativa / Laboratório de Linguagens e Técnicas Adaptativas.*, pages 78–82, 2009.

[5] E. Oro and M. Ruffolo. PDF-TREX: An Approach for Recognizing and Extracting Tables from PDF Documents. In *Proceedings of the 2009 10th International Conference on Document Analysis and Recognition*, ICDAR '09, pages 906–910, Washington, DC, USA, 2009. IEEE Computer Society.

[6] B. Suresh, C. Case, A. Coates, and A. Y. Ng. Autonomous Sign Reading for Semantic Mapping. 2011 IEEE International Conference on Robotics and Automation, May 2011.

[7] B. Yildiz, K. Kaiser, and S. Miksch. pdf2table: A Method to Extract Table Information from PDF Files. *Proceedings of the 2nd Indian International Conference on Artificial Intelligence IICAI05*, 2005.

[8] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 76–85, New York, NY, USA, 2005. ACM.

[9] S. Zheng, R. Song, J.-R. Wen, and D. Wu. Joint optimization of wrapper generation and template detection. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 894–902, New York, NY, USA, 2007. ACM.

[10] P. Zigoris, D. Eads, and Y. Zhang. Unsupervised Learning of Tree Alignment Models for Information Extraction. In *Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops*, pages 45–49, Washington, DC, USA, 2006. IEEE Computer Society.

AWARE – um ambiente para estudo e visualização de processos adaptativos

I.S. Oliveira, J.E. Kogler Jr, *Senior Member, IEEE* e E. Del Moral Hernandez, *Senior Member, IEEE*

***Abstract*— this paper presents AWARE, a software tool developed to aid research in the field of adaptive agents, consisting of a graphical user interface for description and execution of adaptive finite state machines (AFSM), and that enables the visualization of the topological changes produced by the application of the adaptive functions. It includes a simulator of a virtual agent behavior embedded in a virtual environment, under the control of the specified AFSM, which helps to understand the effects of the adaptation strategies before applying it to a real agent in a physical world. It also includes a module to handle physical robotic agents, by receiving their sensorial inputs and directing them the control actions resulting of the application of the controller rules, generated by the AFSM simulator.**

***Keywords*— adaptive devices, modeling, simulation, virtual agents, robotic agents, software tools.**

I. Introdução

ESTE trabalho apresenta uma ferramenta de software, à qual demos o nome AWARE (de *Adaptive SoftWARE*), voltada ao estudo da adaptatividade em sistemas de agentes robóticos imersos em ambientes dinâmicos, permitindo a monitoração dos comportamentos dos agentes e dos processos que os controlam. Sua principal característica consiste em permitir a visualização e documentação das mudanças estruturais decorrentes de adaptações no comportamento e controle dos agentes. O modelo de controle explorado baseia-se particularmente em um dispositivo formal adaptativo dirigido por regras. Na presente versão as regras são pré-especificadas em correspondência com as situações potencialmente causadoras de modificações. Todavia, sua formulação foi concebida de modo a poderem ser construídas através de outras regras adaptativas hierarquicamente mais abstratas e que possam ser expressas via métodos de decisão formais (traduzindo uma adaptatividade formal de ordem mais elevada) ou por decisões decorrentes de procedimentos de inferência indutiva tais como aprendizagem estatística e neurocomputação, entre outros.

Os agentes robóticos considerados são plataformas fisicamente constituídas, dotadas de hardware diretamente controlado pelo AWARE. A ferramenta aqui apresentada oferece também recursos para simular tais agentes de forma minimalista, dispondo de uma interface gráfica pela qual o comportamento estimado do agente pode ser visualizado em correspondência com a apresentação visual das modificações em sua estrutura de controle acarretadas pela ação das adaptações. Isso permite que se estude passo a passo o processo de adaptatividade antes de aplicá-lo diretamente ao agente físico, possibilitando evitar perigos físicos ou prever situações de complexidade maior que requeiram acompanhamento mais cuidadoso. A figura 1 ilustra essas características da ferramenta.

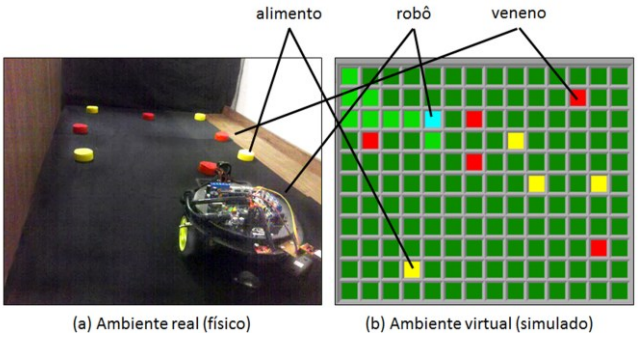


Figura 1. Agente robótico imerso no ambiente – (a) real; (b) virtual. (Note-se, todavia, que não são situações exatamente idênticas, mas são similares).

Na figura 1 apresentam-se lado a lado duas situações, uma em que um robô físico está imerso no ambiente real e, na outra um robô virtual encontra-se em um ambiente virtual similar. Ambos ambientes são dotados de dois tipos de objetos, que diferem apenas em duas propriedades: uma delas é diretamente observável (cor); a outra se manifesta pelo seu efeito sobre o agente (alimento versus veneno), que deve ser inferida. No ambiente virtual, o robô é abstraído por um quadrado azul e sua trajetória é indicada pelas posições que ocupou anteriormente, as quais estão marcadas com um tom verde mais claro que o do fundo. O AWARE proporciona através desse esquema um ambiente virtual de experimentação prévia, permitindo testar hipóteses em situações em que perturbações físicas (atrito, erros de posicionamento, incertezas sensoriais) não precisem ser levadas em consideração.

O ambiente de software que constitui o AWARE será apresentado nas próximas seções, mostrando-se sua concepção e os seus atuais componentes. Entretanto, esta ferramenta não se resume a sua atual realização, pois pode ser modificada para acomodar aplicações de maior complexidade, mantendo-se sua parte essencial que é a arquitetura de controle subjacente. Ela suporta quaisquer dispositivos adaptativos dirigidos por um conjunto finito de regras com formulação determinística. A estrutura básica do simulador do ambiente virtual também pode ser mudada sem modificar sua essência. E, quanto ao

I.S. Oliveira, Universidade de São Paulo, São Paulo, Brasil, ian.oliveira@usp.br
J.E. Kogler Jr., Universidade de São Paulo, São Paulo, Brasil, joao.kogler@ieee.org
E. Del Moral Hernandez, Universidade de São Paulo, São Paulo, Brasil, emilio_del_moral@ieee.org

agente físico, também é possível lidar com robôs ou equipamentos mais sofisticados, sem que para tanto seja necessário realizar mudanças fundamentais na arquitetura.

Discutiremos na próxima seção a concepção e a especificação do AWARE e aspectos de sua realização. Na seção seguinte apresentaremos a ferramenta tal como se encontra na versão atual e a seguir daremos alguns exemplos de testes do conceito. Finalmente concluiremos, discutindo as perspectivas de futuras versões e quanto à sua distribuição.

II. Concepção e especificação da ferramenta AWARE

A necessidade de uma ferramenta como o AWARE surgiu naturalmente como meio de se controlar e monitorar o comportamento de agentes cognitivos. Essencialmente um agente é uma abstração que se concentra nos mecanismos de produção de ações em resposta a estímulos. Agentes podem ser empregados como modelos para estudar-se o comportamento de organismos naturais (biológicos) e são úteis na produção de teorias que visem predizer as reações de criaturas vivas submetidas a determinadas situações. A validade de cada teoria decorrerá de sua capacidade de acertar as predições sobre as observações obtidas em situações experimentais. Agentes também podem ser utilizados para especificar organismos artificiais (robôs, máquinas e sistemas automáticos), que se comportem de modo análogo aos seres vivos, isto é, seguindo o princípio de associar as ações que produzem a respostas a determinadas entradas que recebem como estímulos.

Há diversas categorias de agentes, de acordo com sua capacidade de utilizar sua experiência pretérita. Há os que são puramente reativos e dotados de regras fixas que associam cada possível estímulo a uma resposta correspondente. Há também aqueles cujas regras contêm parâmetros que podem ser modificados de acordo com os estímulos, conferindo uma forma elementar de adaptação que, entretanto, não modifica a forma da regra. Há os que podem inferir regras novas a partir dos estímulos e/ou de regras anteriores, e os que usam conhecimento anterior para produzir as novas regras. Estes dois últimos tipos constituem a classe dos agentes cognitivos. Pode-se então conceber agentes cujo controle baseia-se em ingredientes de maior ou menor complexidade no mecanismo de elaboração das regras que definem seu comportamento, sendo estas dependentes ou não em relação aos estímulos e ao conhecimento anterior. Em resumo, a essência da ideia de agente está na forma de associação entre as suas ações e os estímulos que as precedem.

Há uma classe muito variada de modelos que descrevem essas associações, e entre eles encontram-se os modelos formais, que se caracterizam como dispositivos dirigidos por regras. É de grande interesse que se possam utilizar modelos formais, beneficiando-se de sua manipulação através de métodos algébricos e dedutivos, o que facilita sua construção, modificação e verificação. No caso de se trabalhar com agentes cognitivos, a dificuldade na formalização decorre de sua natureza intrinsicamente flexível, de poderem adaptar seus comportamentos em função do ambiente em que estão imersos, o que se traduziria, em termos de modelos formais, em sistemas de regras que permitam modificar-se de acordo com as circunstâncias. É nesse cenário que os dispositivos adaptativos são de grande valia.

A introdução do conceito de dispositivos formais adaptativos como modelos de sistemas capazes de se alterarem dinamicamente em resposta espontânea aos estímulos de entrada a que estão submetidos [1], [2], trouxe consigo novas possibilidades na investigação da aprendizagem por agentes.

Trabalhar com tais modelos baseados em dispositivos adaptativos constitui um grande desafio, todavia. Usualmente a especificação de um modelo cresce em complexidade na medida em que se empregam construtos elementares mais simples. O arcabouço das teorias baseadas em métodos mais formais constitui-se de elementos construtivos geralmente muito simples, como é o caso das teorias de computação: tendem a basearem-se em operações elementares. A expressão de funcionalidades nesses modelos provém da estrutura que envolve essas operações elementares. No caso dos dispositivos adaptativos, o agravante extra de complexidade resulta da adaptatividade, que se expressa através de regras que permitem modificar a estrutura que traduz cada funcionalidade, isto é, da topologia que caracteriza as relações entre os estímulos e as respostas. Lidar com esse tipo de modelo eficientemente requer ferramentas que auxiliem a apreciação dos detalhes desse sistema complexo, das mudanças topológicas que essas adaptações acarretam. Tendo-se isso em consideração, decidimos que, antes de iniciar uma linha de investigação em agentes capazes de aprender empregando modelos formais, seria importante investir na busca de uma ferramenta de análise que facilitasse lidar com os dispositivos adaptativos, que permitisse o acompanhamento de suas modificações na medida em que se adaptam, colocando-as em correspondência com as modificações no comportamento do agente.

A especificação dessa ferramenta foi orientada para a concepção de uma plataforma básica contendo apenas as características essenciais mínimas para a realização de experimentos dessa categoria, mas de maneira tal que facilmente proporcionasse sua modificação para aplicações mais complexas. Os principais requisitos considerados nessa concepção mínima compreendem: capacidade de tratar um agente robótico controlado por um dispositivo adaptativo, imerso em um ambiente dotado de pelo menos dois tipos de objetos com os quais o agente possa interagir.

A expressão ‘agente robótico’ denota um agente autônomo dotado de sensores e efetores, capaz de alterar suas relações com o ambiente em resposta a estímulos que recebe do mesmo e sob a ação de processos internos que controlam seu estado. O agente considerado na versão atual consiste de uma plataforma robótica móvel capaz de se comunicar com um computador hospedeiro. Parte da arquitetura de controle encontra-se embarcada na plataforma, sendo esta dotada de um micro controlador destinado a gerenciar o acionamento dos motores das rodas, a aquisição de dados de sensores simples tais como acelerômetros e sensores ultrassónicos de distância e

a comunicação com o computador hospedeiro. Nomeadamente, utilizou-se para esse fim a micro arquitetura Arduino [6], embora esse detalhe não seja importante, uma vez que qualquer micro arquitetura embarcada poderia ter sido utilizada, desde que fosse capaz de lidar com todos os sinais e processos locais preconizados.

Em princípio, a emulação do dispositivo adaptativo que efetivamente controla o comportamento do agente poderia ser realizada tanto localmente no processador embarcado, quanto remotamente, em um computador hospedeiro em comunicação com o agente. É essencial que seja respeitada a temporização dos processos em resposta aos estímulos, permitindo que o comportamento do agente se desenvolva sem atrasos que possam comprometer sua interação com o ambiente e seu desempenho. No caso particular da atual implementação do agente robótico, optou-se pela distribuição do controle de modo a deixar embarcado neste apenas a parte relativa à aquisição de sinais de sensores simples e o acionamento dos efetores (rodas). Os dados sensoriais são processados remotamente e convertidos nos estímulos consumidos pelo dispositivo adaptativo. Este é emulado no computador hospedeiro e produz as saídas que serão direcionadas para o controlador embarcado e que servirão para acionar os efetores. O agente robótico foi também especificado para empregar visão computacional, necessária para permitir ao agente a identificação de situações baseada em informações visuais obtidas através do processamento de video capturado com uma câmera. Esse fator reforçou a necessidade de apoiar-se em um hospedeiro remoto, haja vista que o processamento do video pode ser bastante oneroso para ser executado em uma micro arquitetura embarcada.

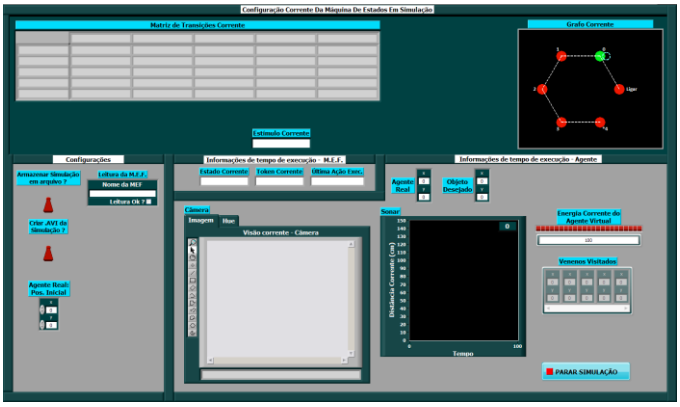
Especificado o agente robótico e a arquitetura do sistema, que consequentemente tem o controle distribuído, passou-se à concepção da ferramenta de software para a simulação dos dispositivos formais adaptativos, que exercerão o controle do agente. Primeiramente buscou-se alguma ferramenta já existente com as características desejadas. Nomeadamente, tais características são:

1. Capacidade de lidar com dados provenientes da aquisição de sinais de sensores analógicos de diferentes modalidades.
2. Capacidade de produzir saídas que serão convertidas para sinais analógicos que servirão para acionar efetores.
3. Ser compatível com o emprego de arquitetura de controle distribuída.

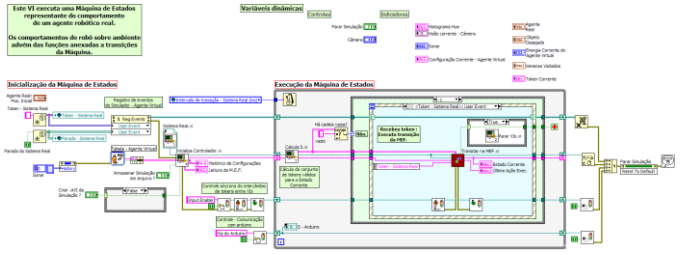
Além dessas características essenciais à configuração pretendida, a ferramenta a ser utilizada deveria oferecer flexibilidade suficiente para ser utilizável para configurações mais complexas, permitindo facilidade nas modificações e boa escalabilidade. Feita uma busca na literatura, encontrou-se apenas uma potencial ferramenta que se aproximasse do atendimento a esses requisitos. Os trabalhos de Camolesi e Neto [3] e [4], oferecem uma proposta de um metambiente, materializada por meio de uma ferramenta com interface visual, destinada a oferecer ao usuário, meios para a construção automática de ambientes completos para a definição, formalização, simulação e ensaios de dispositivos formais, adaptativos ou não, especificados pelo próprio usuário [5]. Entretanto, tais trabalhos concentraram-se na definição e formalização do metambiente e não pareceram atender a todos os requisitos propostos. Consequentemente, decidimos por desenvolver uma ferramenta própria, adequada aos fins propostos.

III. Características da Ferramenta de Software

Para se desenvolver rapidamente uma plataforma suficientemente flexível para ser aplicável a configurações experimentais bastante variadas, satisfazendo aos requisitos especificados optou-se pelo emprego do LabVIEW® , recurso disponível e adequado. O LabVIEW [7] consiste em um ambiente de desenvolvimento de programação visual e uma linguagem visual orientada pelo fluxo de dados. A linguagem do LabVIEW oferece expressividade suficiente para alcançar a flexibilidade requerida em lidar com a variedade de situações experimentais que se tem em mente e dispõe de uma biblioteca suficientemente rica de funções e construtos mais elaborados que, juntamente com os recursos de seu ambiente de desenvolvimento, permitem que se construa a ferramenta desejada em um prazo razoável. A figura 2 apresenta a interface gráfica do AWARE e o código visual em LabVIEW, o que oferece uma boa ideia de suas características.



(a)



(b)

Figura 2. Características do AWARE – (a) interface gráfica de usuário - GUI (painel frontal); (b) código do programa na linguagem visual do LabVIEW® (diagrama de blocos).

Na fig.2a pode-se apreciar os elementos essenciais do AWARE, que se encontram na parte superior do painel e consistem na matriz e no grafo que descrevem o dispositivo adaptativo representado por uma máquina de estados finitos. As linhas e colunas da matriz são indexadas, respectivamente, pelos estados de origem e de destino ligados pelas transições, e suas células contêm as regras na forma ba / cj / s / aa , denotando, respectivamente, a função adaptativa aplicada antes da ação (ba) , o *token* corrente (cj), o estado atual (s) e a função adaptativa aplicada depois da ação (aa), seguindo a notação em [2]. A figura 3 apresenta um exemplo da matriz e do grafo correspondente em duas situações entre o consumo de um token e a aplicação de uma ação adaptativa de inserção.

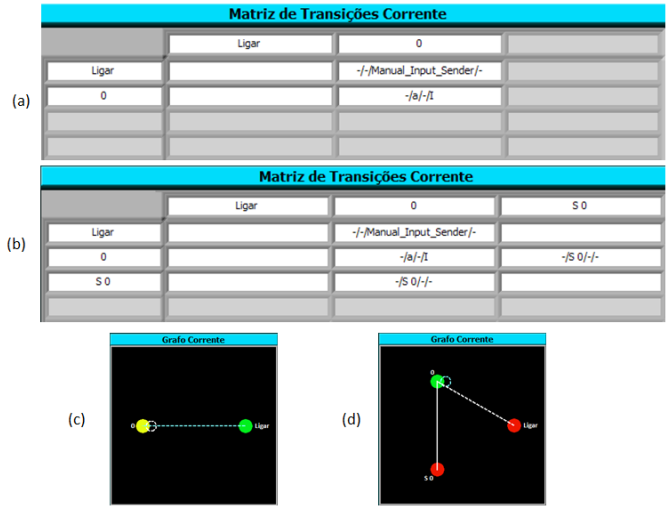


Figura 3. (a) e (c) – matriz e grafo para MEF com 2 estados: em verde o estado de origen e em amarelo o destino; (b) e (d) – MEF com 3 estados.

Nas figs. 3a e 3c apresentam-se a matriz e o grafo correspondentes a uma máquina de estados finitos (MEF) tal que, após consumir o token 'a', é aplicada uma função adaptativa de inserção, resultando na MEF indicada nas figs.3b e 3d. Nesta última, o círculo verde indica o estado atual e os vermelhos os demais estados. As linhas tracejadas indicam transições unidirecionais e as cheias transições bidirecionais. As linhas em cor ciano indicam a transição corrente.

A interface gráfica mostrada na fig.2ª apresenta também outras janelas, utilizadas para o monitoramento das diversas variáveis de interesse, cuja natureza depende do tipo de sensor ou efetor que está sendo considerado. E também apresenta uma janela, mostrada na figura 1b, que no caso de se estar trabalhando com o agente virtual, exibe a configuração do ambiente virtual. Note-se que neste caso não há sensores a se monitorizar, portanto seu espaço no painel é ocupado por essa visualização do ambiente virtual.

IV. Conclusões.

O AWARE foi desenvolvido em um prazo de aproximadamente onze meses, dentro de um programa de iniciação científica. Os autores agradecem à Associação de Ex-Alunos da Escola Politécnica da Universidade de São Paulo (AEP) pelo suporte financeiro. Diversos testes foram implementados para funções adaptativas de todos os tipos (inserção, remoção, consulta) para diversos exemplos de [2] e outros adicionais, como prova de conceito. Após estes, foram conduzidos mais testes com o ambiente virtual e alguns com o agente robótico físico, todos bem sucedidos. A continuidade do trabalho visa aplicar agora o AWARE à investigação de processos de segunda ordem, que deverão apoiar-se em inferências indutivas de natureza estatística realizadas sobre os resultados das interações entre o agente e os componentes do ambiente.

Referencias

[1] Neto, J.J. (2002). Adaptive rule-driven devices-general formulation and case study. (B. Watson & D. Wood, Eds.)Implementation and Application of Automata, 2494, 466–470.

[2] Ramos, M.V.M., Neto, J.J., Vega, I.S., Linguagens Formais, Bookman, São Paulo, 2009.

[3] Camolesi, A.R., Neto, J.J. *Modelagem Adaptativa de Aplicações Complexas*. XXX Conferencia Latinoamericana de Informática – CLEI'04. Arequipa - Peru, Setiembre 27 - Octubre 1, 2004.

[4] Camolesi, A.R. *Proposta de um Gerador de Ambientes para Modelagem de Aplicações Usando Tecnologia Adaptativa* – Tese de Doutorado, Escola Politécnica da USP, São Paulo, 2007.

[5] Neto, J.J. Um Levantamento da Evolução da Adaptatividade e das Tecnologias Adaptativas. IEEE LATIN AMERICA TRANSACTIONS, v. 5, n. 7, pp. 496-505, NOV. 2007

[6] Margolis, M. – Arduino Cookbook – O'Reilly, 2011

[7] Travis, J., Kring, J. LabVIEW for Everyone, Prntice-Hall, Upper Saddle River, NJ, 3a ed., 2007.

Ian Silva Oliveira is undergrad student of Computer Engineering at Polytechnic School of Engineering of the University of Sao Paulo. His current research interests are applications of computational intelligence techniques and computer theoretic models to adaptive aspects of intelligent agents operating in changing environments.

João Eduardo Kogler Jr. is currently researcher and lecturer at Polytechnic School of University of Sao Paulo (USP). He received his PhD in computer vision and image processing and his MSc in biomedical engineering from USP. He is graduated in electrical engineering and physics respectively at Maua Institute of Technology and USP. He was a visiting scientist at Siemens Corporate Research Lab of Princeton, NJ and of the INRIA Sophia Antipolis, France. His current research interests are in computational intelligence, adaptive computational devices, cognitive robotics and computational models of cognition and perception..

Emilio Del Moral Hernandez is currently associated professor at Polytechnic School of University of Sao Paulo (USP). He received his PhD and MSc on electrical engineering at University of Pennsylvania, PA. He is graduated and received his MSc in electronic engineering at Polytechnic School of USP. He has research collaborations with several international institutions and universities, including the CINVESTAV-Mexico Institute, the Catholic University of Peru - PUC-Lima, the Universidad Autónoma de Barcelona and the Universidad Complutense de Madrid. His current research interests are theory and applications of neural networks, bio-inspired computational systems, electronic implementation of neural models, data mining, pattern recognition, decision support systems and applications to finance engineering.

ε-Greedy Adaptativo

A. S. Mignon; R. L. A. Rocha

***Resumo*—Um dos grandes desafios em aprendizagem por reforço é o balanceamento entre *exploration* e *exploitation*. Neste trabalho apresenta-se um método para balanceamento denominado *ε-greedy* adaptativo. Este método é baseado no *ε-greedy* tradicional, que mantém o valor de ε estático. Na solução apresentada são utilizados os conceitos e as técnicas da tecnologia adaptativa para permitir que o valor do ε seja alterado ao longo da execução. Apresenta-se experimentos comparando numericamente os dois métodos para o problema *n-armed bandit*. No caso estacionário o método *ε-greedy* adaptativo apresentou melhor desempenho em relação ao método tradicional, enquanto, no caso não estacionário, o desempenho dos dois métodos foi similar.**

***Keywords*—aprendizagem por reforço, *ε-greedy*, adaptatividade.**

I. Introdução

Aprendizagem por reforço (*reinforcement learning*) é uma forma de aprendizagem de máquina (*machine learning*) não supervisionada na qual um agente aprende através de sua interação com o ambiente para atingir um determinado objetivo [1]. Ao interagir com o ambiente através de ações tomadas, o agente recebe recompensas numéricas. A meta do agente é maximizar o total de recompensas numéricas recebidas.

Um dos grandes desafios em relação à aprendizagem por reforço está no balanceamento entre *exploration* e *exploitation*[1]. *Exploitation* significa que o agente seleciona a ação que obteve maior média de recompensas. *Exploration* significa que o agente seleciona uma ação aleatoriamente, independente dos valores de recompensa obtidos anteriormente. *Exploitation* é a melhor coisa a se fazer para receber uma boa recompensa imediatamente. Entretanto, para descobrir quais ações são as melhores, é necessário executar o modo de *exploration*. Neste modo, pode-se encontrar uma ação melhor e se obter melhores resultados a longo prazo. Como não é possível executar o modo de *exploration* e de *exploitation* ao mesmo tempo, deve-se decidir por qual modo usar para realizar uma ação em um determinado instante de tempo.

Um método simples para realização do balanceamento entre *exploration* e *exploitation* é o método denominado *ε-greedy*. Este método comporta-se gulosamente (*greedily*) a maior parte do tempo, porém, de vez em quando, com uma pequena probabilidade ε, seleciona uma ação aleatoriamente entre todas as ações. Entretanto, o valor dessa probabilidade ε é estática.

A tecnologia adaptativa trata de técnicas e dispositivos que permitem a um sistema modificar seu próprio comportamento, em resposta a algum estímulo de entrada ou ao seu histórico de operações, sem a interferência de qualquer agente externo [2]. A tecnologia adaptativa permite que um sistema com regras estáticas torne-se um sistema com regras dinâmicas.

Neste trabalho apresenta-se o método *ε-greedy* adaptativo. Esse método tem como base o método *ε-greedy* tradicional, porém permite que o valor de ε seja alterado de acordo com as recompensas recebidas do ambiente. Comparou-se numericamente os dois métodos para o problema *n-armed bandit*. O *ε-greedy* adaptativo apresentou desempenho superior ao *ε-greedy* no caso estacionário. Já no caso não estacionário estudado, os dois métodos apresentaram resultados similares.

Este trabalho está dividido da seguinte forma: na seção II apresenta-se os principais conceito e elementos da aprendizagem por reforço. Na seção III descreve-se o problema *n-armed bandit* utilizado para a comparação dos métodos. Na seção IV apresenta-se o método *ε-greedy*. Na seção V descreve-se brevemente os conceito da tecnologia adaptativa. Na seção VI descreve-se o método e o algoritmo do *ε-greedy* adaptativo. Na seção VII apresenta-se os resultados dos experimentos realizados para a comparação dos métodos. Finalmente, na seção VIII apresenta-se as conclusões e trabalhos futuros.

II. Aprendizagem por Reforço

Aprendizagem por reforço é uma forma de aprendizagem de máquina com foco em aprender o que fazer. O aprendizado é realizado por um *agente* através de sua interação com um *ambiente* [1]. O agente deve possuir uma ou mais metas relativas ao ambiente. O ambiente fornece ao agente comentários (*feedback*) em relação a ações tomadas, na forma de recompensas numéricas. A meta do agente é maximizar o total de recompensas numéricas.

As recompensas servem para definir *políticas* ótimas em processos de decisão de Markov (*Markov Decision Processes - MDPs*). Uma política ótima é uma política que maximiza o total de recompensa esperada. A tarefa da aprendizagem por reforço é usar as recompensas observadas para aprender uma política ótima (ou quase ótima) para o ambiente [3].

Diferentemente de outras formas de aprendizagem de máquina, ao agente aprendiz não é dito que ações tomar, ele deve descobrir, através de um processo de tentativa e erro, quais ações produzem maior recompensa. Na maioria dos casos, as ações podem não somente afetar a recompensa imediata, mas também a próxima situação e, por conseguinte, todas as recompensas posteriores [1].

Aprendizagem por reforço, uma forma de aprendizagem não supervisionada, é diferente das formas de aprendizagem supervisionada. Na aprendizagem supervisionada o aprendizado é feito através de exemplos fornecidos por algum supervisor externo [3] [4]. É um importante tipo de aprendizado, porém ele somente não é adequado em situações de aprendizagem por interação. Em problemas que requerem interação é difícil obter-se exemplos do comportamento desejado que são

[1]Neste trabalho optou-se em manter em inglês os termos relacionados à aprendizagem por reforço.

corretos e representativos para todas as situações em que o agente tem que agir. Neste tipo de problema o agente deve ser capaz de aprender a partir de sua própria experiência.

Um dos desafios na área de aprendizagem por reforço está no dilema entre *exploration* e *exploitation*. Para obter uma maior recompensa, um agente deve preferir tomar ações que obtiveram melhor recompensa no passado. Mas para descobrir tais ações ele tem que tentar ações que ainda não foram selecionas anteriormente. O agente deve executar o modo de *exploitation*, isto é, selecionar a melhor ação conhecida até o momento, para obter imediatamente uma maior recompensa, porém ele também deve executar o modo de *exploration*, isto é, selecionar outras ações que não a melhor, para poder realizar melhores ações no futuro. Em aprendizagem por reforço são propostos diversos métodos para tentar o balanceamento entre *exploration* e *exploitation* [1].

A. Elementos da Aprendizagem por Reforço

Além do agente e do ambiente, pode-se identificar quatro sub-elementos principais de um sistema de aprendizagem por reforço: uma *política*, uma *função de recompensa*, uma *função valor*, e, opcionalmente, um *modelo* do ambiente [1].

Uma política define a forma como o agente aprendiz comporta-se em um dado momento. Ela é um mapeamento dos estados percebidos do ambiente para ações a serem tomadas quando se está naquele estado. O agente precisa aprender uma política ótima (ou próximo da ótima) de modo a realizar o seu objetivo.

Uma função de recompensa define o objetivo em um problema de aprendizagem por reforço. Ela mapeia cada estado percebido (ou o par estado-ação) do ambiente a um único número, uma recompensa. A tarefa do agente é maximizar o total de recompensas recebidas a longo prazo. A função de recompensa define quais os eventos são bons ou ruins para o agente.

Uma função valor especifica o que é bom a longo prazo. O *valor* de um estado é o montante total de recompensa que um agente pode esperar acumular em um futuro, iniciando daquele estado.

Um modelo do ambiente é algo que simula o comportamento daquele ambiente. Por exemplo, dado um estado e uma ação o modelo pode prever o próximo estado e a próxima recompensa. Ele é usado para planejamento, através do qual temos um meio de decidir o curso de uma ação considerando situações futuras antes de realmente experimentá-las.

III. O Problema *n-Armed Bandit*

Imagine que um agente depara-se constantemente com uma escolha entre *n* diferente opções, ou ações, a cada escolha recebe uma recompensa numérica do ambiente, escolhida a partir de uma distribuição de probabilidades estacionária que depende da ação selecionada. O objetivo é maximizar o total de recompensas esperadas sobre algum período de tempo, por exemplo, 1000 opções de ações ou passos de tempo. Cada ação de seleção é denominada de *jogada* (*play*).

Esta é a forma original do problema *n-armed bandit*, nomeado pela analogia a uma máquina caça-níqueis. Cada ação selecionada é como puxar uma alavanca da máquina caça-níqueis, e as recompensas são os pagamentos obtidos. Através de repetidas jogadas você tem que maximizar seus ganhos concentrando-se em puxar as melhores alavancas. Outra analogia é a de um médico escolher tratamentos experimentais para uma série de pacientes doentes. Cada jogada é a seleção de um tratamento, e cada recompensa é o bem estar do paciente.

No problema *n-armed bandit* cada ação tem uma recompensa esperada. Denomina-se essa recompensa esperada de *valor* daquela ação. Se o valor de cada ação é conhecido, então, é trivial a solução do problema. Entretanto, assume-se que não se conhece os valores das ações com certeza, tem-se apenas uma estimativa de cada valor.

Se mantivermos as estimativas dos valores de cada ação, então, em algum espaço de tempo, existe pelo menos uma ação cujo valor estimado é maior. Denomina-se essa ação com maior valor de *greedy* (gulosa). Se a ação *greedy* é selecionada, diz-se que se está *exploiting* o conhecimento corrente dos valores das ações. Se, ao invés disto, seleciona-se uma ação não *greedy*, diz-se que se está *exploring*. O modo de *exploration* permite melhorar as estimativas das ações com valores menores.

Exploitation é a melhor coisa a se fazer para maximizar a recompensa esperada em um passo, mas *exploration* pode produzir uma maior recompensa total a longo prazo. Por exemplo, suponha que os valores das ações *greedy* são conhecidos com certeza, enquanto diversas outras ações são estimadas como boas, mas com uma incerteza substancial. A incerteza é tal que pelo menos uma destas outras ações provavelmente são realmente melhores que a ação *greedy*, porém não se sabe qual é. Se existem muitos passos à frente para realizar seleções de ações, então pode ser melhor executar o modo de *exploration* para as ações não *greedy* e descobrir quais delas são melhores que a ação *greedy*. Com isso, a recompensa é baixa a curto prazo, durante o modo de *exploration*, porém pode ser alta a longo prazo porque depois de descoberta as melhores ações pode-se executar o modo de *exploitation* com elas muitas vezes. Por não ser possível executar o modo de *exploration* e o modo de *exploitation* ao mesmo tempo com uma única seleção de ações, refere-se a isso como o conflito entre *exploration* e *exploitation*.

O problema *n-armed bandit* tem sido usado para modelar o dilema do balanceamento entre *exploration* e *exploitation*. Na próxima seção, apresenta-se um método simples para tal balanceamento. Esse método, descrito em [1], é utilizado no restante deste trabalho.

IV. ε-Greedy

Nesta seção apresenta-se um método simples, denominado *ε-greedy*, para balanceamento entre *exploration* e *exploitation* que utiliza a estimativa de recompensa esperada para cada ação como forma de seleção de uma ação. A descrição deste método foi baseada em [1].

Denota-se o real valor de uma ação a como $q_*(a)$, e o valor estimado no t-ésimo passo de tempo como $Q_t(a)$. Uma forma de calcular a estimativa da ação a é através da média de recompensas recebidas quando a ação foi selecionada. Em outras palavras, se pelo t-ésimo passo de tempo a ação a foi selecionada K_a vezes antes de t, recebendo recompensas R_1, R_2,..., R_{K_a}, então seu valor estimado é:

$$Q_t(a) = \frac{R_1 + R_2 + ... + R_{K_a}}{K_a} \quad (1)$$

Se $K_a = 0$, então define-se para $Q_t(a)$ um valor padrão como, por exemplo, $Q_1(a) = 0$. Como $K_a \to \infty$, pela lei dos grandes números, então $Q_t(a)$ converge para $q_*(a)$ [1]. Esta técnica é denominada de média das amostras (*sample-average*) para a estimativa dos valores das ações porque cada estimativa é uma média simples das recompensas.

A regra mais simples de seleção de ações é selecionar a ação com maior valor estimado. Isto é, selecionar no passo t uma das ações *greedy*, A_t^*, para o qual $Q_t(A_t^*) = max_a Q_t(a)$. Este método sempre executa o modo de *exploitation* utilizando o conhecimento corrente das estimativas para maximizar a recompensa imediata. Entretanto, como visto na seção anterior, pode haver ações melhores a serem tomadas. Neste caso, é necessário que se execute o modo de *exploration* para descobrir tais ações.

Uma alternativa simples é comportar-se *greedily* (gulosamente) a maior parte do tempo, porém, de vez em quando, com uma pequena probabilidade ε, seleciona-se uma ação aleatoriamente entre todas as ações, com probabilidade igual, e independente do valor estimado para cada ação. Denomina-se esse método de seleção ação de ε-*greedy*.

Uma vantagem deste método é que, à medida que o número de jogadas aumenta, cada ação será recompensada um número infinito de vezes, garantindo que $K_a \to \infty$ para todo a, garantindo assim que todos os $Q_t(a)$ convergem para $q_*(a)$. Isto implica que a probabilidade de seleção da ação ótima converge para um valor maior que 1 - ε, isto é, para próximo da certeza.

A seguir, comparam-se numericamente os métodos *greedy* e ε-*greedy* em um conjunto de testes. Os testes foram realizados com um conjunto de 2000 tarefas geradas aleatoriamente para o problema *n-armed bandit* com $n = 10$. Para cada alavanca, os valores das ações, $q_*(a)$, $a = 1, ..., 10$, foram selecionados de acordo com uma distribuição normal (Gaussiana) com variância entre 0 e 1.

No $t-$ésimo passo de tempo com uma dada alavanca, a recompensa real R_t é o $q_*(A_t)$ para a alavanca (onde A_t é a ação selecionada) mais um termo de ruído distribuído normalmente com variância entre 0 e 1. Calculando a média sobre as alavancas, podemos traçar o desempenho e o comportamento dos métodos e verificar como eles se comportam com a experiência de 1000 passos (jogadas).

As Figuras 1 e 2 apresentam os gráficos de comparação do método *greedy* com dois métodos ε-*greedy* ($\varepsilon = 0.01$ e $\varepsilon = 0.1$), conforme descrito acima. Os dois métodos calculam as estimativas dos valores das ações usando a técnica de média das amostras.

O gráfico da Figura 1 apresenta o aumento da recompensa esperada com a experiência. O método *greedy* melhorou ligeiramente mais rápido no início em relação aos outros métodos, mas depois estabilizou-se em um nível bem abaixo. Ele conseguiu uma recompensa por passo de apenas 1, em comparação com o melhor valor obtido neste teste que foi de

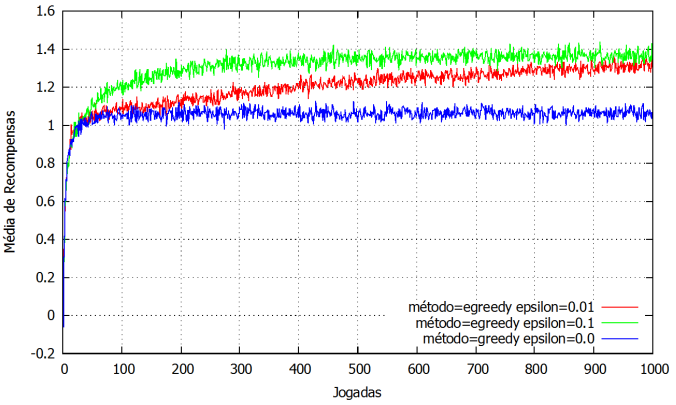


Figura 1. Desempenho médio dos métodos *greedy* e ε-*greedy* para o aumento da recompensa esperada.

cerca de 1.45. O método *greedy* executa significantemente pior a longo prazo porque fica preso a uma ação sub-ótima.

O gráfico da Figura 2 mostra que o método *greedy* encontra a ação ótima em aproximadamente um terço das tarefas. O método ε-*greedy* executa melhor porque continua a executar o modo de *exploration*, e melhora suas chances de descobrir a ação ótima. O método com $\varepsilon = 0.1$ executa mais o modo de *exploration*, e geralmente encontra a ação ótima mais cedo, mas nunca a seleciona mais de 91% das vezes. O método com $\varepsilon = 0.01$ melhora mais lentamente, mas eventualmente tem desempenho melhor que o $\varepsilon = 0.1$ a longo prazo.

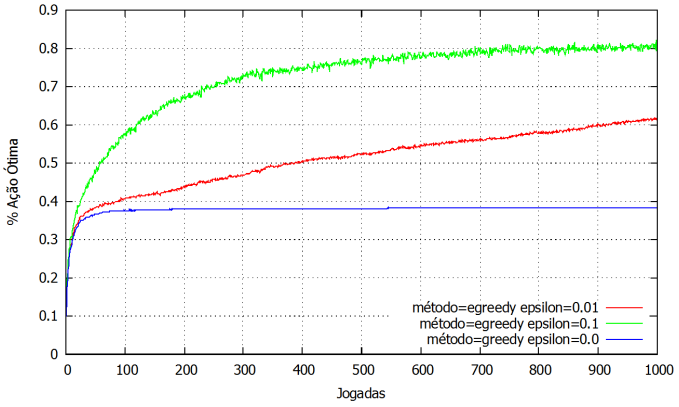


Figura 2. Desempenho médio dos métodos *greedy* e ε-*greedy* em selecionar uma ação ótima.

O método discutido utilizando a média das amostras é apropriado em um ambiente estacionário, isto é, quando o ambiente não sofre alterações. Entretanto, frequentemente encontra-se problemas de aprendizagem por reforço que são não estacionários, isto é, quando o ambiente sofre alterações ao longo do tempo. Nesses casos, é necessária uma técnica diferente da média das amostras para calcular o valor estimado de uma ação. Tais técnicas não são aqui discutidas por estarem fora do escopo deste trabalho.

V. Tecnologia Adaptativa

Adaptatividade é a capacidade que um sistema tem de modificar seu próprio comportamento, em resposta a algum

estímulo de entrada ou ao seu histórico de operações, sem a interferência de qualquer agente externo [2]. A tecnologia adaptativa trata de técnicas e dispositivos que tem característica adaptativa. Dispositivos adaptativos são descrições abstratas de problemas que tem comportamento dinâmico. Essas descrições são associadas a dispositivos não adaptativos subjacentes que representam problemas com comportamento estático.

Dispositivos não adaptativos têm seu comportamento definido por um conjunto de regras estáticas. Um dispositivo não adaptativo subjacente é melhorado adicionando-se um conjunto de ações adaptativas. Essas ações caracterizam as operações necessárias para fazer o sistema comportar-se adaptativamente [2].

Um algoritmo é dito adaptativo quando é capaz de modificar espontaneamente o seu comportamento em resposta a uma condição especial de entrada [5]. Na próxima seção, apresenta-se um algoritmo adaptativo para o método ε-*greedy*. Esse algoritmo tem por objetivo permitir que o valor do ε seja modificação ao longo de sua execução, de acordo com as recompensas obtidas do ambiente.

VI. ε-Greedy Adaptativo

Utilizando os conceitos e técnicas da tecnologia adaptativa, criou-se o método denominado ε-*greedy* adaptativo. Este método é baseado no ε-*greedy*, porém permite que o valor do ε varie ao longo da execução. A variância do valor do ε depende das recompensas retornadas pelo ambiente.

O Algoritmo 1 apresenta o algoritmo usado no método ε-*greedy* adaptativo. Esse algoritmo é usado para a seleção de uma ação a quando se está em um estado s. Ele decide se deve executar o modo de *exploration* ou de *exploitation*. Se decidir pelo modo de *exploitation* seleciona a ação com maior média de recompensa (A_t^*). Se decidir pelo modo de *exploration* seleciona uma ação aleatoriamente, conforme descrito na seção IV. Quando o algoritmo estiver no modo de *exploration* ele pode executar uma ação adaptativa, dependendo de sua configuração, que modifica o valor do ε.

As variáveis max_{ant} e k são globais, sendo utilizadas para armazenar o estado do algoritmo. Quando o algoritmo é executado para a seleção de uma ação ele deve decidir se executa o modo de *exploration* ou de *exploitation*. Ao modo de *exploration* adicionou-se uma ação adaptativa que pode alterar o valor de ε. São contadas quantas vezes o algoritmo entrou no modo de *exploration*, utilizando a variável k, após a última vez que ele alterou o valor de ε. Antes da variável k atingir um limite especificado, se houver uma grande diferença entre as variáveis max_{ant} e max_{atual}, significa que deve-se ajustar o valor de ε, pois a alteração no ambiente foi significativa. Caso o limite seja atingido, então altera-se o valor de ε independente da diferença obtida. Após a alteração do valor de ε a variável k é zerada e a variável max_{ant} recebe o valor da variável max_{atual}.

O algoritmo contém alguns valores constantes que foram definidos empiricamente como melhores após experimentos realizados entre uma faixa de valores. O valor da variável *LIMIT* é de 12, porém testaram-se valores na faixa entre 3 e 18. O fator de multiplicação 10 para o cálculo do Δ foi usado para acentuar a diferença entre os valores de max_{ant} e max_{atual}.

Algoritmo 1 ε-greedy adaptativo

```
max_ant ← 0
k ← 0
se distribuição normal < ε então
  max_atual ← Q_t(A*_t)
  k ← k + 1
  Δ ← |max_ant − max_atual| * 10
  se k < LIMIT então
    se Δ >= 0.8 então
      ε ← sigmoid(Δ)
      max_ant ← max_atual
      k ← 0
    fim se
  senão
    ε ← sigmoid(Δ)
    max_ant ← max_atual
    k ← 0
  fim se
  seleciona uma ação aleatoriamente
senão
  seleciona A*_t
fim se
```

Isso foi necessário para obter-se um valor mais adequado para ε retornado pela função sigmóide. Para definirmos o valor do fator de multiplicação testaram-se valores na faixa entre 5 e 30. Para definirmos o que seria um valor grande para a diferença Δ testou-se valores na faixa entre 0.3 e 2.0.

A definição de um novo valor para o ε é feita utilizando-se uma função sigmóide, conforme apresentada abaixo. O gráfico da função é apresentado na Figura 3. Nele pode-se verificar que o valor de ε pode variar entre 0 e 0.5. Empiricamente verificou-se que não há ganhos significativos se o valor de ε for superior a 0.5.

$$sigmoid(x) = \frac{1.0}{1.0 + \exp(-x)} - 0.5 \qquad (2)$$

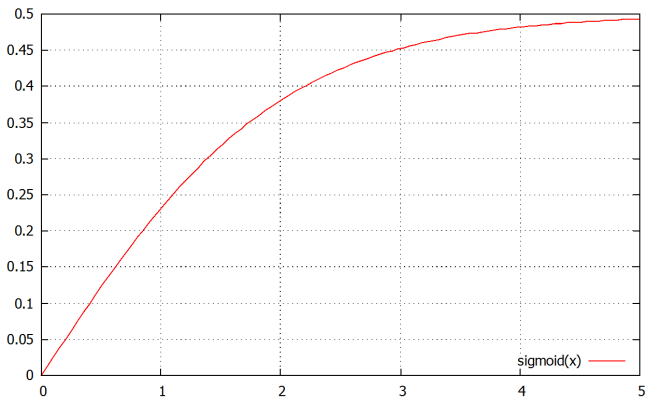


Figura 3. Gráfico da função sigmóide utilizada para definir novos valores para ε.

VII. Experimentos e Resultados

Esta seção apresenta os experimentos realizados para comparar numericamente os métodos ε-*greedy* e ε-*greedy*

adaptativo para o problema *n-armed bandit*. Implementou-se os experimentos em linguagem Java, utilizando o arcabouço *TeachingBox*[2]. Este arcabouço oferece diversas ferramentas e algoritmos de aprendizagem por reforço, inclusive o método ε-*greedy* utilizado no experimento.

Para a implementação do algoritmo ε-*greedy* adaptativo criou-se a classe *EpsilonGreedyPolicyAdapt* no ambiente do *TeachingBox*. Esta classe estende a classe *GreedyPolicy*. A execução dos experimentos utilizou os recursos disponíveis no ambiente.

Nos experimentos comparou-se dois métodos ε-*greedy* ($\varepsilon = 0.1$ e $\varepsilon = 0.5$) com o método ε-*greedy* adaptativo ($\varepsilon = 0.5$). O valor de $\varepsilon = 0.5$ na versão adaptativa é apenas um valor inicial para ε, já que ele será modificado ao longo da execução. Foram testados valores iniciais para ε variando entre 0.1 e 0.9, e os resultados foram similares. Definiu-se adotar o valor inicial de 0.5 por ser o maior valor retornado pela função sigmóide, utilizada para a modificação do ε. Os métodos calculam as estimativas dos valores das ações usando a técnica de média das amostras.

Os experimentos foram realizados com um conjunto de 2000 tarefas geradas aleatoriamente para o problema *n-armed bandit* com $n = 10$. Verificou-se o desempenho e o comportamento dos métodos com a execução de 1000 passos (jogadas). Dois tipos de experimentos são apresentados: uma para o caso estacionário e outro para o caso não estacionário.

A. Caso Estacionário

As Figuras 4 e 5 apresentam os gráficos de comparação dos métodos, conforme descritos acima.

O gráfico da Figura 4 apresenta o aumento da recompensa esperada com a experiência. O método ε-*greedy* adaptativo apresentou desempenho superior aos outros dois métodos. Pode-se verificar que demorou um pouco mais para melhorar no início em relação ao ε-*greedy* com $\varepsilon = 0.1$, porém ao final foi superior a esse método atingindo uma recompensa de cerca de 1.55, enquanto o método ε-*greedy* com $\varepsilon = 0.1$ obteve, ao final, uma recompensa de cerca de 1.45. O método ε-*greedy* com $\varepsilon = 0.5$ obteve uma recompensa em cerca de apenas 0.85.

O gráfico da Figura 5 apresenta que o método ε-*greedy* adaptativo encontra ações ótimas mais cedo e no final atinge um percentual de cerca de 85% de seleção da ação ótima. Enquanto, o método ε-*greedy* com melhor desempenho ($\varepsilon = 0.1$) atingiu, ao final, um percentual de cerca de 80% de seleção da ação ótima.

B. Caso Não Estacionário

Para a realização dos experimentos no caso não estacionário, definiu-se que o ambiente altera a sua configuração após 500 passos (jogadas) de execução. As Figuras 6 e 7 apresentam os gráficos de comparação dos métodos. Logo após a mudança de configuração do ambiente no passo 500, os dois métodos apresentam um desempenho muito baixo, uma vez que o conhecimento anterior obtido está em discordância com

[2]Disponível em: http://servicerobotik.hs-weingarten.de/en/teachingbox.php. Acesso em 25/11/2013.

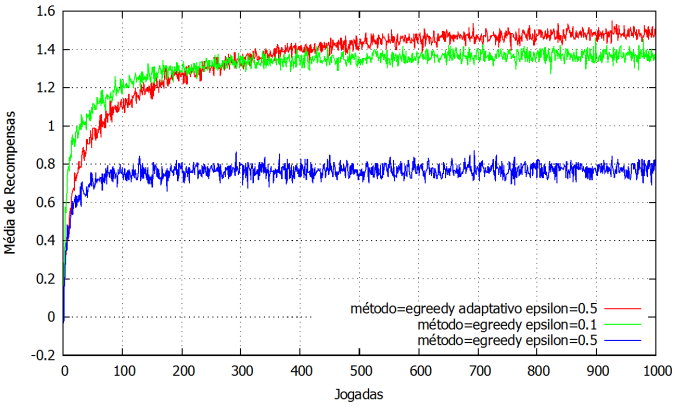


Figura 4. Desempenho médio, no caso estacionário, dos métodos ε-*greedy* e ε-*greedy* adaptativo para o aumento da recompensa esperada.

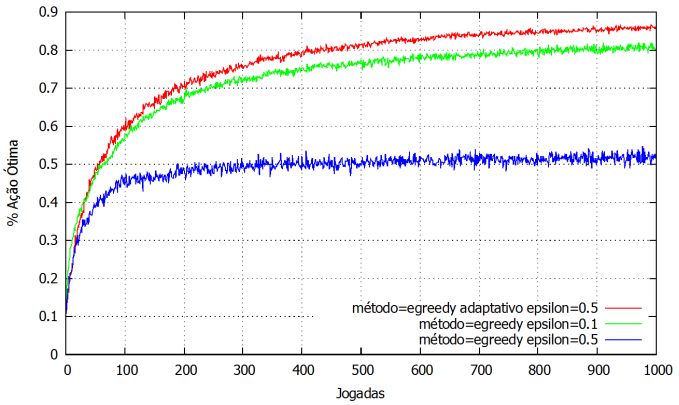


Figura 5. Desempenho médio, no caso estacionário, dos métodos ε-*greedy* e ε-*greedy* adaptativo em selecionar uma ação ótima.

a nova configuração do ambiente. Eles têm que re-aprender essa nova configuração.

O gráfico da Figura 6 apresenta o aumento da recompensa esperada com a experiência. Após a mudança de configuração do ambiente no passo 500, o método ε-*greedy* apresentou desempenho similar ao método ε-*greedy* com $\varepsilon = 0.1$, atingindo uma recompensa de cerca de 1.0. O método ε-*greedy* com $\varepsilon = 0.5$ obteve uma recompensa em cerca de apenas 0.55.

O gráfico da Figura 7 mostra que os métodos ε-*greedy* adaptativo e ε-*greedy* com $\varepsilon = 0.1$ apresentam resultados similares após a alteração da configuração do ambiente. Ambos os métodos, ao final dos 1000 passos, seleciona-se cerca de 40% das vezes a ação ótima.

VIII. Conclusão

Um dos grandes desafios da área de aprendizagem por reforço é o balanceamento entre *exploration* e *exploitation*. O ε-*greedy* é um método simples para tal balanceamento, entretanto o valor de ε é estático. Neste trabalho apresentou-se o método ε-*greedy* adaptativo. Ele possibilita que o valor de ε seja dinâmico, adaptando-se ao comportamento do ambiente.

Os experimentos realizados mostraram que no caso estacionário o método ε-*greedy* adaptativo teve melhor desempenho em relação ao método ε-*greedy*. A versão adaptativa

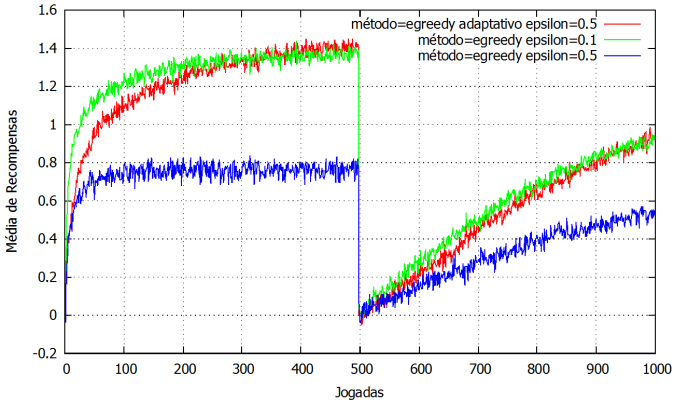


Figura 6. Desempenho médio, no caso não estacionário, dos métodos ε-*greedy* e ε-*greedy* adaptativo para o aumento da recompensa esperada.

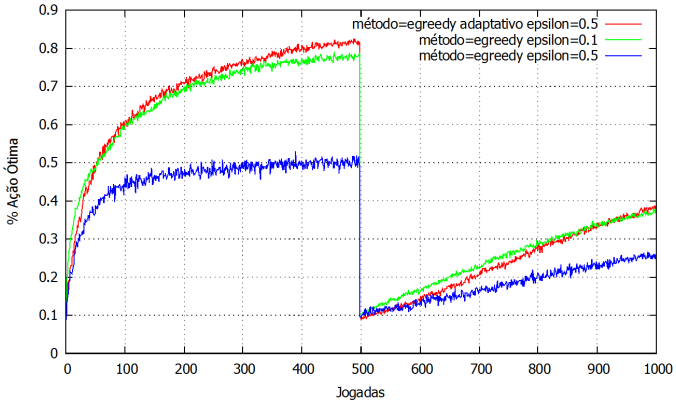


Figura 7. Desempenho médio, no caso não estacionário, dos métodos ε-*greedy* e ε-*greedy* adaptativo em selecionar uma ação ótima.

encontra mais cedo uma ação próxima da ótima e também seleciona a ação ótima uma porcentagem maior de vezes, em torno de 5% a mais. No caso não-estacionário o método ε-*greedy* adaptativo obteve resultados similares ao método ε-*greedy* após a mudança de configuração do ambiente.

Como trabalhos futuros pretende-se estudar novas políticas adaptativas de aprendizagem para melhorar o desempenho do método ε-*greedy* adaptativo para o caso não estacionário.

REFERÊNCIAS

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[2] J. J. Neto, "A small survey of the evolution of adaptivity and adaptive technology," *Revista IEEE América Latina*, vol. 5, no. 7, pp. 496–505, 2007, (in Portuguese).

[3] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Prentice Hall, 2002.

[4] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[5] J. C. Luz, "Tecnologia adaptativa aplicada à otimização de código em compiladores," Master's thesis, Escola Politécnica, Universidade de São Paulo, 2004.

Predição de Tempo de Compilação e Tamanho de Código em Máquinas Virtuais

Jorge Augusto Sabaliauskas
Escola Politécnica da Universidade de São Paulo
Email: jaugustosaba@usp.br

Ricardo Luis de Azevedo da Rocha
Escola Politécnica da Universidade de São Paulo
Email: rlarocha@usp.br

***Resumo*—O acelerado desenvolvimento de novos processadores faz com que os compiladores tenham que se adaptar continuamente. Para se adaptarem mais rápido começou-se a investigar a aplicação de técnicas de aprendizagem computacional nos compiladores. Este artigo visa introduzir a aplicação de técnicas de aprendizagem em compilação ao fazer a predição do tempo de compilação e o tamanho de código gerado para métodos na máquina virtual Maxine utilizando características extraídas do *bytecode*.**

I. Introdução

O processo manual de ajuste do compilador faz com que os benefícios de uma nova geração de processadores não sejam aproveitados imediatamente. Um processo de adaptação automático as novas arquiteturas é desejável.

O aprendizado computacional procura tentar fazer o computador aprender a executar tarefas complexas sem que um algoritmo precise ser explicitamente determinado.. Há dois modos principais de aprendizado: supervisionado e não-supervisionado. Algoritmos de aprendizado supervisionado tentam aprender um modelo através de exemplos enquanto que os algoritmos de aprendizado não-supervisionado tentam descobrir padrões nos dados geralmente por identificação de semelhanças. Combinações dos aprendizados supervisionados e não-supervisionados também podem ser empregadas.

No início dos anos 2000 ocorreram com maior frequência aplicações das técnicas de aprendizagem computacional na área de compiladores para a criação de um processo adaptativo de seleção de otimizações (por exemplo o artigo [1]). Heurísticas utilizadas anteriormente eram produzidas através de um especialista que manualmente trabalhava sobre o código gerado pelo compilador em busca de melhores oportunidades de geração de código [2].

O objetivo deste trabalho é mostrar como técnicas de aprendizagem computacional podem ser aplicadas no contexto de compiladores através de duas predições simples: tempo de compilação e tamanho de código. Para a demonstração foi escolhida a máquina virtual Java Maxine [3] e o *benchmark* DaCapo [4].

Essa predição é feita sobre o tempo de compilação e tamanho de código gerado sobre métodos de uma classe Java. Características do método são calculadas estaticamente (em tempo de compilação) e então usadas na predição. Essa predição é feita levando em conta os diferentes perfis de compilação disponíveis na Maxine.

Como algoritmo de aprendizado é utilizado o algoritmo de regressão linear, um algoritmo de aprendizagem supervisionado. Esse tipo de regressão foi escolhido por ser o mais simples e ideal para uma introdução. Nesse algoritmo a saída consistirá de uma combinação linear das características do método. Para validação é utilizado o algoritmo *k-fold validation*.

O restante desse artigo organiza-se da seguinte forma:

- Em Metodologia é apresentado como será realizado o aprendizado
- Em Experimento é descrita a preparação do experimento
- Em Resultados é mostrado os preditores obtidos
- Em Discussão os resultados são comparados com o modelo intuitivo
- Em Conclusão será abordada a importância dos resultados obtidos

II. Metodologia

Em compiladores é comum possuir quatro níveis de compilação identificados por O0, O1, O2 e O3. O perfil de compilação indicado por O0 não aplica nenhuma otimização enquanto que o perfil O3 aplica todas as otimizações disponíveis. Cada otimização requer um tempo de compilação e gera um código de determinado tamanho.

Tanto o tempo de compilação quando o tamanho do código gerado pela compilação podem ser determinados em função do trecho do programa alvo do processo de compilação.

Como tanto o tempo de compilação quanto o tamanho de código gerado são valores numéricos, o algoritmo de aprendizado utilizado é chamado de regressão. O algoritmo de regressão utilizado é o de regressão linear (maiores detalhes em [5]) descrito por (1), onde h_θ (chamado comumente de hipótese) pode ser tanto o tempo de compilação quanto o tamanho do código gerado, n é o número de características, x_i tal que $i \in \{1 \dots n\}$ são as características computadas sobre os métodos, x_0 é o valor constante 1 e θ_i tal que $i \in \{0 \dots n\}$ é o vetor de pesos.

$$h_\theta(x) = \theta^T X = \sum_{i=0}^{n} \theta_i x_i \quad (1)$$

Tabela I. CARACTERÍSTICAS CALCULADAS SOBRE OS BYTECODES DE MÉTODOS.

x_i	**Característica**	**Descrição**
1	bytecodes	Número de bytecode em um método
2	localSpace	Espaço para locais
3	synch	O método é sincronizado ?
4	exceptions	O método possui código de tratamento de instruções ?
5	leaf	O método é folha (sem chamadas para outras instruções) ?
6	final	O método é declarado final ?
7	private	O método é declarado privado ?
8	aload_astore	Fração instruções aload e astore no bytecode
9	primitive_long	Fração de instruções sobre primitivos ou longs
10	compare	Fração de bytecodes de comparação
11	jsr	Fração de instruções JSR
12	switch	Fração de instruções switch
13	putOrGet	Fração de instruções put ou get
14	invoke	Fração de instruções de invocação
15	new	fração de instruções de criação de instâncias
16	arrayLength	fração de instruções ArrayLength
17	athrow_checkCast_monitor	fração de instruções athrow, checkcast or monitor
18	multi_newarray	fração de bytecodes multiNewArray
19	simple_long_real	fração de bytecodes de conversão sobre tipos simples, long ou real

Para definir qual é a melhor reta na regressão linear é necessário definir a função de custo. A função de custo utilizada nesse artigo é a *mean squared error* (de uso mais comum) descrita por (2) onde $X^{(i)}$ e $Y^{(i)}$ representam o vetor de características $(1, x_1, \ldots, x_n)$ e o valor observado respectivamente na amostra rotulada com índice i e m é o total de amostras.

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(X^{(i)}) - Y^{(i)})^2 \tag{2}$$

A partir da equação (2) pode-se encontrar a melhor reta através do algoritmo de descida de gradiente. Logo, o vetor de pesos pode ser encontrado por (3), onde α é chamado de coeficiente de aprendizado. A escolha de α deve ser feito com cautela, pois se for um valor muito alto, o algoritmo pode não convergir. Pode-se detectar a não convergência monitorando caso função de custo (2) não diminua de uma iteração para outra.

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j \tag{3}$$

A partir do trecho do programa podem ser calculadas características que descrevem o trecho a ser compilado (nesse caso o método de uma classe). As características são obtidas a partir de uma simples passagem pelo *bytecode* coletando informações sobre instruções, como o feito em [6]. Todas as características computadas sobre métodos são descritas na Tabela I. Alguns exemplos do resultado do cálculo de características podem ser vistos na Tabela VII.

Para o processo de treinamento será utilizado o método *k-fold validation* onde as amostras de treinamento são divididas em k partições, onde k-1 partições são usadas no treinamento e 1 partição para teste. Esse processo é repetido k vezes para poder estimar o erro do algoritmo de aprendizado (nesse caso a regressão linear).

Foram retirados três diferentes conjuntos de treinamento, um para cada perfil de compilação (O1, O2 e O3). No final foram produzidas seis equações no formato de (1): três para predição de tempo de compilação e três para predição de tamanho de código gerado.

III. EXPERIMENTO

Para o experimento foi utilizada a máquina virtual Maxine em conjunto com o *benchmark* DaCapo, uma coleção de programas usualmente utilizados sobre a máquina virtual Java. O uso da Maxine envolvendo aprendizado computacional já foi feito em [7]. Através de configurações de linha de comando foi alterado o perfil de otimização. Para o processamento de *bytecodes* foi utilizada a biblioteca *javassist* [8].

A coleta do vetor de características foi feita através de um *Java Agent* que calculava as características conforme o carregamento de classes pela máquina virtual. Como o *benchmark* DaCapo também faz um gerenciamento do carregamento de classes, algumas classes não tiveram as características calculadas usando o *Java Agent*. Para contornar a perda de informações sobre métodos, foi criado um programa para varrer o arquivo do DaCapo em busca de classes e calcular suas características. Alguns métodos não tiveram características calculadas devidos a limitações da biblioteca *javassist* mas mesmo assim foi possível obter uma quantidade de amostras $m > 1500$ para cada perfil de compilação.

O resultado do tempo de compilação de um método e o tamanho do código gerado foram obtidos facilmente por uma configuração de linha de comando na Maxine. Ao ativar essa opção a Maxine passou a imprimir toda compilação que realizava. Essa impressão teve que ser feita em um arquivo pois senão pode-se misturar os dados de compilação com a saída do DaCapo, dificultando o processamento posterior.

Com os features para cada método calculado pelo *Java Agent* ou leitura do arquivo do DaCapo e o tempo de compilação e tamanho de código informados pela Maxine, bastou juntar esses dados para se produzir o conjunto de treinamento para cada perfil de compilação. Foram mantidas as assinaturas dos métodos por conveniência.

Para o treinamento foi utilizado a ferramenta *Weka* [9] que já possuía implementado o algoritmo de regressão linear. Na validação *k-fold validation* foi utilizado o valor k=10 sugerido pelo próprio Weka.

IV. RESULTADOS

Nessa seção podem ser encontrados o resultado do treinamento da regressão linear no *Weka* para o tamanho de código gerado e tempo de compilação. Além disso são fornecidas cinco previsões de saída para dados do conjunto de treinamento.

A. Otimização O1

O resultado para previsão do tamanho do código na otimização O1 fornecido pelo *Weka* pode ser visto na Figura

```
size =
     9.2062 * bytecodes +
    38.3063 *  localSpace +
   125.9147 *  synch +
  1245.0898 *  exceptions +
   155.9847 *  leaf +
   124.2994 *  aload_astore +
 -1484.4537 *  primitive_long +
-21011.2525 *  jsr +
   600.4225 *  invoke +
  2344.8719 *  new +
   479.8091 *  athrow_checkCast_monitor +
 -2287.1684 *  simple_long_real +
  -249.659
```

Figura 1. Regressão linear para a predição de tamanho de código no modo O1. O tamanho de código é dado em bytes.

Tabela II. CINCO PREDIÇÕES DE TAMANHO NA OTIMIZAÇÃO O1.

Valor Real (bytes)	Predição (bytes)	Erro (bytes)
1556	1843.419	287.419
1174	1693.233	519.233
32	20.907	-11.093
77	242.571	165.571
29	20.907	-8.093

IV-A. O comportamento para cinco previsões do tamanho de código pode ser vistos na Tabela II. O conjunto de treinamento possuía $m = 1843$ instâncias.

O resultado para previsão do tempo de compilação na otimização O1 fornecido pelo *Weka* pode ser visto na Figura 2. O comportamento para cinco previsões do tempo de compilação pode ser vistos na Tabela III.

B. Otimização O2

O resultado para previsão do tamanho do código na otimização O2 fornecido pelo *Weka* pode ser visto na Figura 3. O comportamento para cinco previsões do tamanho de código pode ser vistos na Tabela IV. O conjunto de treinamento possuía $m = 1657$ instâncias.

```
time =
   0.068  * bytecodes +
  -0.6754
```

Figura 2. regressão linear para predição de tempo de compilação no modo O1. O tempo é dado em milisegundos.

Tabela III. CINCO PREDIÇÕES DE TEMPO DE COMPILAÇÃO NA OTIMIZAÇÃO O1.

Valor Real (ms)	Predição (ms)	Erro (ms)
3.2	12.637	9.437
3.8	11.111	7.311
0.2	-0.515	-0.715
0.0	0.357	0.357
0.1	-0.515	-0.615

```
size =
    12.4861 * bytecodes +
    50.3098 *  localSpace +
  1500.5807 *  exceptions +
   362.1418 *  leaf +
   149.8093 *  private +
   296.6817 *  aload_astore +
 -2513.885  *  primitive_long +
 -3381.3528 *  compare +
-24723.3547 *  jsr +
  -279.414  *  putOrGet +
  1581.3714 *  invoke +
  3458.0678 *  new +
 -5519.8463 *  simple_long_real +
  -466.7764
```

Figura 3. regressão linear para a predição de tamanho de código no modo O2. O tamanho de código é dado em bytes.

Tabela IV. CINCO PREDIÇÕES DE TAMANHO NA OTIMIZAÇÃO O2.

Valor Real (bytes)	Predição (bytes)	Erro (bytes)
51	134.201	83.201
122	383.901	261.901
908	1260.888	352.888
171	575.66	404.66
383	779.047	396.047

O resultado para previsão do tempo de compilação na otimização O2 fornecido pelo *Weka* pode ser visto na Figura 4. O comportamento para cinco previsões do tempo de compilação pode ser vistos na Tabela V.

C. Otimização O3

O resultado para previsão do tamanho do código na otimização O3 fornecido pelo *Weka* pode ser visto na Figura 5. O comportamento para cinco previsões do tamanho de código pode ser vistos na Tabela VI. O conjunto de treinamento possuía $m = 1655$ instâncias.

```
time =
  0.0684 * bytecodes +
  3.5308 *  leaf +
  5.0397 *  private +
  5.2166 *  aload_astore +
 15.5741 *  invoke +
 -5.4619
```

Figura 4. regressão linear para predição de tempo de compilação no modo O2. O tempo é dado em milisegundos.

Tabela V. CINCO PREDIÇÕES DE TEMPO DE COMPILAÇÃO NA OTIMIZAÇÃO O2.

Valor Real (ms)	Predição (ms)	Erro (ms)
0.2	1.11	0.91
0.2	1.356	1.156
1.8	5.093	3.293
0.3	3.765	3.465
1	0.905	-0.095

```
size =
    12.7794 * bytecodes +
    64.3957 *  localSpace +
  2514.5824 *  exceptions +
   409.5668 *  leaf +
   168.9502 *  private +
   231.8959 *  aload_astore +
 -2680.7018 *  primitive_long +
 -3530.3777 *  compare +
-36919.3804 *  jsr +
  1915.7344 *  invoke +
  3883.0666 *  new +
 -5978.9789 *  simple_long_real +
  -587.0996
```

Figura 5. regressão linear para a predição de tamanho de código no modo O3. O tamanho de código é dado em *bytes*.

Tabela VI. CINCO PREDIÇÕES DE TAMANHO NA OTIMIZAÇÃO O3.

Valor Real (bytes)	Predição (bytes)	Erro (bytes)
114	259.124	145.124
29	-14.342	-43.342
1953	1051.965	-901.035
5672	5774.292	102.292
58	75.685	17.685

O resultado para previsão do tempo de compilação na otimização O3 fornecido pelo *Weka* pode ser visto na Figura 6. O comportamento para cinco previsões do tempo de compilação pode ser vistos na Tabela VII.

```
time =
   0.0815 * bytecodes +
   9.3518 *  exceptions +
   3.8292 *  leaf +
   1.4214 *  private +
   2.7379 *  aload_astore +
 -13.3196 *  primitive_long +
-140.6578 *  jsr +
  -2.2356 *  putOrGet +
  13.378  *  invoke +
 -39.7453 *  simple_long_real +
  -4.3945
```

Figura 6. regressão linear para predição de tempo de compilação no modo O3. O tempo é dado em milisegundos.

Tabela VII. CINCO PREDIÇÕES DE TEMPO DE COMPILAÇÃO NA OTIMIZAÇÃO O3.

Valor Real (ms)	Predição (ms)	Erro (ms)
0.3	0.716	0.416
0.1	-0.106	-0.206
3.6	3.471	-0.129
17.5	24.095	6.595
0.2	-0.477	-0.677

V. DISCUSSÃO

Um efeito indesejável de se utilizar a regressão linear foi o aparecimento de predições com valores negativos tanto para o tempo de compilação quanto para o tamanho de código, ambas deveriam ter sempre valores positivos.

Sendo tanto o tamanho do código gerado quanto o tempo de compilação diretamente proporcionais ao tamanho de *bytecodes* não é nenhuma surpresa o número de *bytecodes* aparecer em todas as fórmulas de predição. A predição de tempo na compilação O1, por exemplo, pode ser feita exclusivamente com o número de *bytecodes*.

Vale notar que as predições utilizam mais características conforme o aumento do nível de otimização. O aumento do nível indica que um maior número de otimizações estão sendo realizadas, algumas delas específicas mais relacionadas a uma categoria de instruções.

Mesmo no perfil de compilação O3 (todas as otimizações habilitadas), estão ausentes algumas características na expressão de predição de tamanho de código e tempo de compilação. Uma explicação pode ser o fato de que algumas instruções aparecem raramente no *bytecode*. Sabendo que a Maxine não possui tantas otimizações quanto uma máquina virtual comercial, é razoável supor que alguns *bytecodes* sejam desprezados.

VI. CONCLUSÃO

O processo de predição do tempo de compilação e tamanho de código gerado são um bom exemplo de como começar a aplicar técnicas de aprendizado computacional na área de compiladores (mais especificamente em máquinas virtuais).

O uso da regressão linear é interessante para fazer a predição devido a natureza do processo de compilação: aproximadamente tempo e tamanho proporcionais ao tamanho do código. Como próximo passo poderia ser escolhido um algoritmo de regressão que não produza valores negativos de tempo de compilação e tamanho de código.

É interessante notar que as características são calculadas sobre o código em tempo de compilação. Não daria para ser feita a previsão (ou pelo menos com uma boa precisão) de qualquer valor dependente da execução de um programa. A execução de um programa depende de seus dados de entrada, sem fixar a entrada não seria possível obter uma predição confiável.

A previsão do tempo de compilação e tamanho de código podem ter utilidade para o processo de adaptação em máquinas virtuais (a máquina virtual é um software adaptativo). Por exemplo, através de previsões do tempo de compilação nos diferentes perfis pode-se escolher o perfil cujo tempo de compilação não seja tão grande, lembrando que em máquinas virtuais o tempo de compilação concorre com o tempo de execução do programa.

Em trabalhos futuros deverão ser adicionadas medidas de desempenho para a melhoria das predições. Grupos de otimizações, determinados pelo perfil de compilação, podem ser substituídas por ativações individuais de otimizações. Diferenças observadas no desempenho devido a certas otimizações fornecerão informações úteis no treinamento de

preditores. O tempo de execução de métodos não foi usado nesse trabalho devido a complexidade da medição do tempo, o que necessitaria também de um tratamento estatístico mais rigoroso.

AGRADECIMENTOS

O primeiro autor agradece a bolsa de mestrado de número 33002010045P3 concedida pela CAPES.

REFERÊNCIAS

[1] A. Monsifrot, F. Bodin, and R. Quiniou, "A machine learning approach to automatic production of compiler heuristics," in *Proceedings of the 10th International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, ser. AIMSA '02. London, UK, UK: Springer-Verlag, 2002, pp. 41–50. [Online]. Available: http://dl.acm.org/citation.cfm?id=646053.677574

[2] R. N. Sanchez, J. N. Amaral, D. Szafron, M. Pirvu, and M. Stoodley, "Using machines to learn method-specific compilation strategies," in *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, ser. CGO '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 257–266. [Online]. Available: http://dl.acm.org/citation.cfm?id=2190025.2190072

[3] (2013, Nov.) Maxine virtual machine. [Online]. Available: https://wikis.oracle.com/display/MaxineVM/Home

[4] (2013, Nov.) Dacapo benchmark. [Online]. Available: http://www.dacapobench.org/

[5] (2013, Nov.) Stanford open class. [Online]. Available: http://stanford.io/17kztSC

[6] J. Cavazos and M. F. P. O'Boyle, "Method-specific dynamic compilation using logistic regression," *SIGPLAN Not.*, vol. 41, no. 10, pp. 229–240, Oct. 2006. [Online]. Available: http://doi.acm.org/10.1145/1167515.1167492

[7] D. Simon, J. Cavazos, C. Wimmer, and S. Kulkarni, "Automatic construction of inlining heuristics using machine learning," in *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, ser. CGO '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1109/CGO.2013.6495004

[8] (2013, Nov.) Javassist bytecode manipulation. [Online]. Available: http://www.csg.ci.i.u-tokyo.ac.jp/ chiba/javassist/

[9] (2013, Nov.) Weka 3: Data mining software in java. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/

VII. APÊNDICE

Tabela VIII. CINCO AMOSTRAS DE CLASSES JAVA ENCONTRADAS NO CONJUNTO DE TREINAMENTO. OS VALORES DE TIME E SIZE SÃO DADOS EM MILISEGUNDOS E BYTES RESPECTIVAMENTE.

method	bytecodes	localSpace	synch	exceptions	leaf	final	private	aload_astore	primitive_long	compare	jsr	switch	putOrGet	invoke	new	arrayLength	athrow_checkCast_monitor	multi_newarray	simple_long_real	time	size
Ljava/util/LinkedList; getLast()Ljava/lang/Object;	12	2	0	0	0	0	0	0.333	0.000	0.000	0.000	0.000	0.167	0.083	0.083	0.000	0.083	0.000	0.000	0.3	169
Ljava/util/LinkedList; removeLast ()Ljava/lang/Object;	13	2	0	0	0	0	0	0.385	0.000	0.000	0.000	0.000	0.077	0.154	0.077	0.000	0.077	0.000	0.000	0.4	178
Ljava/util/Calendar; getInstance ()Ljava/util/Calendar;	10	1	0	0	0	0	0	0.300	0.000	0.000	0.000	0.000	0.200	0.300	0.000	0.000	0.000	0.000	0.000	0.4	80
Ljava/util/Date; ¡init¿ (J)V	6	3	0	0	0	0	0	0.333	0.000	0.000	0.000	0.000	0.167	0.167	0.000	0.000	0.000	0.000	0.000	0.2	64
Ljava/util/concurrent/Semaphore; tryAcquire (Ljava/util/concurrent/TimeUnit;)Z	8	4	0	0	0	0	0	0.250	0.000	0.000	0.000	0.000	0.125	0.250	0.000	0.000	0.000	0.000	0.000	0.4	92

Processos Markovianos de Decisão com heurísticas, junção de abordagens backward e forward para transferência de conhecimento baseados em políticas

R. G. Pontes and V. F. Silva

***Abstract*— This article discusses studies conducted in the area of artificial intelligence for problems developed in Abstract Markov Decision Processes, involving algorithms for feature selection, based on previous studies for greedy search strategies "backward" and "forward", adopting a mixed strategy which, despite being computationally expensive, avoids local optimum and ensures best sets of attributes. Also are discussed strategies that adopt heuristics in order to reduce this computational cost, achieving success in comparison to previous algorithms, developed without blended approach and without adopting heuristics.**

***Keywords*— Markov Decision Process, selection of features, greedy search.**

I. INTRODUÇÃO

PROCESSOS Markovianos de Decisão (em inglês *Markov Decision Process, MDPs)* são formulações muito utilizadas em ambientes de decisão sequenciais e estocásticos (Puterman, 1994; Mausam & Kolobov, 2012; Kaelbling et al., 1998). As principais características que compõem os MDPs são as transições probabilísticas entre estados, a possibilidade de se observar o estado corrente do agente de acordo com a ação executada pelo mesmo e a possibilidade de se alterar este processo, em períodos nominados como épocas de decisão. A solução para um MDP consiste na definição de uma política que mapeia para cada estado uma ação; enquanto a solução ótima maximiza as recompensas recebidas dependendo de uma função valor que especifica como recompensas devem ser acumuladas (Coelho, 2009).

Na confecção de algoritmos para a resolução de MDPs, são propostos duas abordagens para a análise dos problemas: na primeira, iteração de valor, utiliza-se a avaliação no espaço de valores, onde para cada estado mapeado é atribuído um valor, atualizado de acordo com as ações executadas e recompensas acumuladas durante as épocas de execução. A execução deste algoritmo é interrompida quando não há mais alterações significativas nos valores atribuídos, ou seja, quando a variação não passa de um determinado valor atribuído pelo desenvolvedor do algoritmo (um *threshold).* Já a outra abordagem envolve avaliação no espaço de políticas, onde também há o cálculo dos valores, mas o critério de parada do algoritmo é diferente, pois envolve a análise das melhores ações a serem tomadas em cada estado para a obtenção da melhor recompensa, de acordo com as ações realizadas e valores de recompensa coletados até então, envolvendo as interações com os estados adjacentes ao mesmo. A atualização de valores é interrompida quando, em uma determinada época de execução, não é detectada troca de ação ótima para o conjunto de estados, comparando a época de execução imediatamente anterior, evitando assim uma dependência do algoritmo de um *threshold* determinado.

A transferência de conhecimento é importante no âmbito dos problemas sequenciais resolvidos com MDPs, visto que a carga computacional utilizada para a execução de modelos de aprendizagem (tanto em abordagens no espaço de valores quanto no de políticas) é elevada. Bogdan e Silva (2012) propuseram um modelo onde, utilizando abordagens *backward* e *forward* separadamente, facilitariam o cálculo de modelos de aprendizado mais genéricos, onde o conhecimento acumulado em um determinado problema fosse repassado como um modelo para outro problema, com algumas características semelhantes e que não necessitaria passar pelo treinamento completo.

Uma abordagem para possibilitar a transferência de conhecimento é a utilização de estados abstratos. A abordagem com estados abstratos se desenvolve tendo como base a existência de atributos para um determinado espaço a ser analisado. Os atributos são características que, em conjunto, podem determinar a unicidade de um estado. Por exemplo, em um problema envolvendo um conjunto de carros podemos utilizar seu identificador único (número da placa, por exemplo) ou um conjunto de atributos que o identifique (cor, modelo, ano, marca, etc). Utilizando-se deste recurso, podemos aplicar um modelo desenvolvido a partir de características genéricas para problemas semelhantes. Por exemplo, utilizando o conjunto dos veículos, podemos estender o aprendizado desenvolvido em um conjunto para outro que apresente elementos com características semelhantes, como por exemplo esquemas de rodízio para veículos com o final da placa de número semelhante.

Dependendo do problema em questão pode ser interessante escolher um subconjunto de atividades. Primeiro, esse subconjunto pode facilitar o aprendizado em uma tarefa a partir do zero, já que diminui a quantidade de observações possíveis. Segundo, a escolha de um subconjunto pode evitar

R. G. Pontes, Universidade de São Paulo (USP), São Paulo, SP, Brasil, rodrgarcia@gmail.com

V. F. Silva, Universidade de São Paulo (USP), São Paulo, SP, Brasil, valdinei.freire@usp.br

"*overfitting*", isto é, facilitar a generalização na transferência de conhecimento entre tarefas. O conjunto não pode ter um número grande de atributos ao ponto de identificar unitariamente cada elemento do conjunto original (para não haver baixa abstração dos estados) nem ser vazio o suficiente para que não agregue os conhecimentos necessários para o desenvolvimento dos problemas a serem enfrentados. Para realizar esta análise, duas abordagens foram utilizadas por Bogdan e Silva (2012), abordagens *backward* e *forward*.

Na abordagem *backward*, são considerados inicialmente todos os atributos e, recursivamente, o algoritmo identifica o atributo mais supérfluo e o descarta, até chegar a um ponto com zero atributos. Durante a execução, ele vai analisando os valores dados aos estados, executando o algoritmo MDP com o conjunto atual daquela época de execução, e vai armazenando em um vetor, que mostra ao final a eficácia de cada conjunto de atributos selecionados em relação ao conjunto original.

Já a utilização de abordagem *forward* é inversa à abordagem *backward*, pois na mesma o conjunto é iniciado sem atributos e é computado em cada época de execução o melhor atributo a ser incluído no conjunto, ordenando o vetor de atributos desde o elemento mais relevante até o menos relevante, considerando todas as execuções realizadas.

Buscamos então, através deste artigo, analisar as características e melhorias apontadas ao se desenvolver um algoritmo que une as duas abordagens *backward* e *forward* dinamicamente executadas.

II. Arcabouço teórico

A seguir, são apresentadas definições acerca dos aspectos inerentes ao desenvolvimento deste artigo, tais como definição de um MDP (Pellegrini e Wainer, 2007), abstrações e seleção de atributos (Bogdan e Silva, 2012).

A. *Formalização de um MDP*

Os MDPs são representados formalmente por uma tupla (S, A, T, R) com os seguintes componentes: S é o conjunto total de estados que o MDP pode estar em algum determinado momento de sua execução; A é um conjunto de ações que o agente pode tomar em algum determinado momento. Vale ressaltar que uma ação pode levar a uma mudança de estado ou permanência no mesmo; T(s,a,s') é uma distribuição de probabilidade sobre a chance do sistema transitar para um estado s' estando anteriormente em um estado s através da realização de uma ação a. Se for aplicado sobre uma política não-determinística, é um valor graduado entre 0 e 1 para cada possível s', se for determinística, é mapeado diretamente a um estado s'; e R(s) é a função recompensa e retorna a recompensa quando o processo está no estado "s" (pertencente a S). Essa recompensa pode ser positiva ou negativa (penalidade).

A solução de um MDP é uma política que mapeia para cada estado uma ação; enquanto a solução ótima maximiza as recompensas recebidas dependendo de uma função valor que especifica como as recompensas devem ser acumuladas. Usualmente $V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t V_t | \pi, S_0 = S]$ e π^* é ótimo se $V^{T^{\pi}}(s) > V^{T}(s)$ para todo $s \in S$ e toda política π.

B. *Transferência de conhecimento em MDPs*

Os MDPs podem ser descritos não apenas como um conjunto de estados e ações, seus estados também podem ser desmembrados em estruturas menores conhecidas como atributos. Os atributos possibilitam uma abstração do problema a ser modelado como um MDP de tal forma que possibilita a transferência de conhecimento entre problemas semelhantes (Otterlo, 2004).

Os estados podem ser abstraídos ao considerar um conjunto de atributos binários $F = \{f_1, f_2, ..., f_n\}$, que são atributos relevantes ao conjunto S de estados, tal que a tupla $f_i : s \rightarrow [0,1]$ indica a presença do atributo f_i no estado s pertencente ao conjunto S. Já σ é um estado pertencente ao conjunto S_σ, que é o conjunto de estados abstratos, ou seja, identificados pela presença de seus atributos. Cada σ é único e mapeado não através de um valor numérico ou índice, mas sim pelo conjunto $F(\sigma)$ de atributos que o compõe; e cada estado concreto s pode ter um estado σ atribuído a ele.

Utilizando estes atributos, pode-se realizar a transferência de conhecimento entre dois problemas desenvolvidos sob a forma de um MDP, pois durante a execução dos algoritmos de iteração de valores ou de políticas, ao invés de se determinar um valor para o estado atual s, é atribuído o valor ao estado abstrato σ e a todos os atributos que o compõe ($f_1, f_2 .... f_n$), formulando uma política ótima para os mesmos. Assim, em caso onde seja necessário comparar dois problemas com atributos semelhantes, caso seja aplicável, o aprendizado realizado sobre um conjunto de dados é relevante na solução do problema a ser solucionado no outro conjunto de dados.

Análogo ao caso dos estados abstratos também temos, no conjunto dos MDPs com características abstratas, ações abstratas. Estas ações $\alpha \in A_{ab}$ induzem a um conjunto de ações A_α. Como para cada estado $s \in A$ são permitidas a execução apenas de ações do conjunto A_s, também há uma restrição a quais ações abstratas pertencentes a A_{ab} possam ser executáveis pelos estados abstratos. A regra para a validade de execução de uma ação abstrata é dada pela seguinte fórmula:

$$A_\sigma = \{\alpha | \alpha \in A_{ab} \wedge \forall s \in S_\sigma \exists \alpha \in A_\alpha a \in A_s\}$$

Assim, cada ação abstrata α poderá ser executada por um estado abstrato σ apenas se for possível correlacioná-lo a pelo menos uma ação não abstrata a representada pela ação abstrata α (ou seja, uma ação $a \in A_\alpha$), em relação a cada estado s correlacionado ao estado abstrato σ (ou seja, um estado $s \in S\sigma$).

Um exemplo prático disso é a utilização do aprendizado para a resolução de problemas envolvendo navegação robótica, onde atributos comuns à tarefa são levados em conta durante o treinamento e confecção da política (proximidade a objetos, como porta e corredores; distância entre o robô e a porta, noções de localização em geral, se o mesmo está distante do objeto, se está próximo da meta, etc.). A política obtida então é aplicada sobre o outro ambiente não avaliado anteriormente, porém como o treinamento foi realizado sobre a abstração de estados através de atributos comuns, o que foi

aprendido durante o treinamento pode ser aproveitado no outro ambiente.

Para realizar a transferência de conhecimento, deve-se considerar políticas abstratas π_{ab} que mapeia estados abstratos $\sigma \in S_{ab}$em ações abstratas $\alpha \in A_{ab}$.

Como através desta abordagem existem casos onde não existam ações abstratas possíveis de serem realizadas por certos estados abstratos, é recomendado que, durante a implementação do problema, seja implementada para todos os estados abstratos uma ação abstrata válida α_R, onde em caso de sua execução seja realizada uma ação válida qualquer dente as presentes em A_s, para que assim não haja problemas durante a execução.

A partir da formulação de suas componentes, podemos declarar uma política estocástica abstrata representada por π_{ab} como sendo $\pi_{ab} : S_{ab}$ x $A_{ab} \rightarrow [0,1]$, onde após recebeu um estado abstrato σ é executada uma ação abstrata α com probabilidade $\pi_{ab}(\sigma,\alpha)$ variando entre 0 (não será executada) e 1 (será executada com 100% de certeza).

C. Formalização do algoritmo AbsProb-PI

O algoritmo AbsProb-PI é um algoritmo que implementa a iteração de políticas baseado no gradiente (Silva et al., 2012). É um algoritmo importante, pois, através do mesmo, é possível localizar políticas abstratas localmente ótimas, em relação a recompensa acumulada.

Define-se uma matriz de transição $T^{\pi_{ab}}(s,s')$ e uma função-valor $V^{\pi_{ab}}(s)$ como segue:

$$T^{\pi_{ab}}(s,s') = \sum_{\alpha \in A_{ab}} \pi_{ab}(\alpha|\sigma(s)) * \sum_{a \in A_\alpha \cap A_s} \frac{1}{|A_\alpha \cap A_s|} T(s,a,s1),$$

$$V^{\pi_{ab}} = (\mathrm{I} - \gamma T^{\pi_{ab}})^{-1} r,$$

aonde r, $T^{\pi_{ab}}$ e $V^{\pi_{ab}}$ são representações vetoriais da função recompensa $r(s)$, da matriz de transição $T^{\pi_{ab}}(s,s')$ e da função valor de estados $V^{\pi_{ab}}(s)$, respectivamente.

Análoga à matriz $T^{\pi_{ab}}$, é definida uma matriz de transição $T^{\alpha,\varepsilon}$, responsável por realizar a escolha de execução entre uma ação abstrata α com probabilidade de ocorrência 1 – ε e o conjunto das outras ações abstratas possíveis de execução com probabilidade ε, sendo ε>0. Também definimos uma função de desconto gradual $\delta(i) = (1 + \frac{i}{k})^{-1}$, onde i é a iteração de execução do algoritmo e k é uma constante. Considerando b^0 como sendo a representação vetorial da distribuição inicial de $b^0(s)$, os passos do algoritmo AbsProb-PI são descritos desta maneira, de acordo com o que foi definido por Silva et al. (2012):

1. Inicialize a política abstrata probabilística π_{ab} aleatoriamente, tal que:

$$\pi_{ab}(\sigma,\alpha) \geq \frac{\varepsilon}{|A_{ab|}} \forall \sigma \in S_{ab}, \alpha \in A_{ab} \sum_{a \in A_{ab}} \pi_{ab}(\sigma,\alpha) = 1 \forall \sigma \in S_{ab}$$

2. A cada iteração i realizar as seguintes etapas:

(a) Calcular o valor da função $V^{\pi_{ab}}$

(b) Calcular o produto de $C = \gamma b^{o\mathrm{T}}(\mathrm{I} - \gamma \mathrm{T}^{\pi_{ab}})^{-1}$

(c) Para cada $\alpha \in A_{ab}$ calcular $\Delta^{\alpha,\pi_{ab}} = (\mathrm{T}^{\alpha,\varepsilon} - \mathrm{T}^{\pi_{ab}})\mathrm{V}^{\pi_{ab}}$

(d) Para cada $\sigma \in S_{ab}$ e cada $\alpha \in A_{ab}$ calcular $W(\sigma,\alpha) = \sum_{s \epsilon S_\sigma} C(s)\Delta^{\alpha,\pi_{ab}}(s)$

(e) Para cada $\sigma \in S_{ab}$ procurar a melhor direção $\alpha_\sigma^* = \arg\max_{\alpha \in A_{ab}} W(\sigma,\alpha)$

(f) Selecionar um valor de desconto $\delta(i)$ e atualizar a política π_{ab} da seguinte maneira:

$$\pi_{ab}(\sigma,\alpha) \leftarrow \begin{cases} (1-\delta(i))\pi_{ab}(\sigma,\alpha) + \delta(i)\left(\frac{\varepsilon}{|A_{ab}|} + (1-\varepsilon)\right), \text{if } \alpha = \alpha_\sigma^* \\ (1-\delta(i))\pi_{ab}(\sigma,\alpha) + \delta(i)\frac{\varepsilon}{|A_{ab}|}, \text{if } \alpha \neq \alpha_\sigma^* \end{cases}$$

D. Abordagens para seleção de atributos

Bogdan e Silva (2012) definem quatro tipos de algoritmos diferentes para seleção de atributos no âmbito dos MDPs abstratos, baseadas em duas abordagens. Uma é conhecida como *backward*, e percorre o conjunto de atributos de um MDP desde cheio (com todos os atributos) até o conjunto vazio (sem nenhum atributo). Já o outro é conhecido como *forward* e realiza o inverso do *backward*, ou seja, percorre desde o conjunto vazio de atributos até um conjunto com todos os atributos. Também existem duas variantes destas abordagens, *backward* heurístico e *forward* heurístico, que se utilizam de heurística para reduzir substancialmente o tempo de execução dos algoritmos. Todos serão discutidos minuciosamente a seguir.

Os quatro algoritmos dependem de uma função de avaliação $V(F)$ que avalia o conjunto de atributos *F*.

Para a obtenção do valor *V* do conjunto F_{f-} (*F* menos o atributo *f*), Bogdan e Silva (2012) propõem dois métodos distintos, um denominado *full backward*, executa uma avaliação plena do valor dos atributos e outro, denominado *backward* por estimativa de peso heurística (ou apenas "*backward* com heurísticas", para simplificar), que pode até ter $[\max\{\frac{N_{passos}}{2}|F_{max} - 2\}]$ vezes menor custo do que a execução plena do método *full backward*.

O método *full backward* envolve a avaliação de um conjunto de atributos pelo valor da política ótima abstrata no conjunto de estados abstratos $S_{ab}^{F_{f-}}$. O valor determinado do conjunto de atributos seguirá a seguinte definição, envolvendo a execução do algoritmo AbsProb-PI:

$\hat{\pi}_{ab}^* \leftarrow \mathrm{AbsProb{-}PI}\left(S_{ab}^{F_{f-}}, N_{passos}\right), V(F_{f-}) = \mathrm{b}^{0\mathrm{T}}(\mathrm{I} - \gamma \mathrm{T}^{\hat{\pi}_{ab}^*})^{-1}\mathrm{r}$.

É desta abordagem que deriva o *backward* com heurísticas. A diferença consta na menor quantidade de cálculos computacionais que o mesmo executa, pois utiliza o vetor C que o algoritmo AbsProb-PI cria, vetor este que representa o desconto acumulado esperado para o valor de recompensa obtido em cada $s \in S$ do MDP seguindo a política abstrata π_{ab}. Esta heurística consiste em pegar estes valores do vetor C para criar uma estimativa do peso de cada estado baseado nas políticas anteriores. A política $\hat{\pi}_{ab}^{F_{f-}}$ é determinada pela

equação $\hat{\pi}_{ab}^{F_{f-}}(\sigma,\alpha) = \frac{\sum_{s\in S_\sigma} C(s)\hat{\pi}_{ab}^{F}(\sigma^F(s),\alpha)}{\sum_{s\in S_\sigma} C(s)}$, onde a política $\hat{\pi}_{ab}^{F_{f-}}$ é obtida através da execução do algoritmo AbsProb-PI sobre o conjunto original de atributos F e a função $\sigma^F: S \rightarrow S_{ab}^F$ mapeia cada estado original no conjunto de estados abstratos.

Após a obtenção da política $\hat{\pi}_{ab}^{F_{f-}}$, podemos definir os valores de $V(F_{f-})$ como sendo $V(F_{f-}) = \mathrm{b}^{0\mathrm{T}}(\mathrm{I} - \gamma \mathrm{T}^{\hat{\pi}_{ab}^{F_{f-}}})^{-1}\mathrm{r}$.

Como fizeram para a obtenção dos valores $V(F_{f-})$ dos atributos no algoritmo *backward*, Bogdan e Silva (2012) também buscaram a obtenção dos valores do conjunto F_{f+} (F mais o atributo f) desenvolvendo dois métodos distintos, um denominado *full forward*, que executa uma avaliação plena do valor dos atributos e outro, denominado *forward* por estimativa heurística baseada em gradientes (ou apenas "*forward* com heurísticas", para simplificar) que, assim como o algoritmo *backward* com heurísticas faz em relação ao *full backward*, pode até ter $[\max\left\{\frac{N_{passos}}{2}|F_{max} - 2\right\}]$ vezes menor custo do que a execução plena do método *full forward*.

O cálculo heurístico do método *forward* com heurísticas envolve o gradiente $W(\sigma,\alpha)$ calculado pela execução do algoritmo AbsProb-PI, sendo que este gradiente calculado é a forma representada de como (e, quantitativamente, quanto) a política corrente pode ser melhorada com a adição de um atributo. Dada a política corrente $\hat{\pi}_{ab}^{F}$ obtida na execução do algoritmo AbsProb-PI, a estimativa heurística baseada em gradiente define o valor dado a um conjunto de atributos F_{f+} da seguinte forma:

$$V(F_{f+}) = \sum_{\sigma\in S_{ab}^{F_{f+}}} \max_{\alpha\in A_{ab}} W(\sigma,\alpha)\,,$$

E. Execução gulosa dos algoritmos backward e forward

Os algoritmos *backward* e forward têm em comum a característica de serem executados de forma gulosa, considerando a iteração atual como sendo a última. Porém esta escolha gulosa acaba por minar a qualidade do conjunto em um todo, gerando uma solução subótima. Por exemplo na execução de um algoritmo *backward* onde um atributo que, de início, não fosse muito relevante ao problema, fosse retirado, poderia ganhar considerável importância a partir de outras iterações, considerando outras combinações de atributos, mas por ser uma execução gulosa do algoritmo este atributo nunca irá adentrar no conjunto de atributos posteriormente.

Para minimizar tal problema propomos uma abordagem mista na resolução de MDPs, alternando a execução de "funções *backward*" e "funções *forward*" no mesmo algoritmo, dando a chance de que certos atributos que foram descartados (ou adicionados precocemente) possam ser readmitidos (ou eliminados, respectivamente), caso haja relevância do atributo no conjunto naquela iteração corrente.

III. ABORDAGENS MISTAS PARA SELEÇÃO DE ATRIBUTOS

Nossas propostas de abordagens mistas baseiam-se nos algoritmos *backward* e *forward* já descritos anteriormente, e suas variantes com heurísticas.

Os experimentos desenvolvidos por Bogdan e Silva (2012) predispõem-se a realizar as iterações sobre os conjuntos de atributos de maneira isolada, ou seja, uma execução completa para o algoritmo *backward*, outra para o algoritmo *forward* e uma para cada variante dos dois algoritmos supracitados. Assim, pode-se realizar uma análise em separado da eficácia de cada abordagem por si só. Sob o ponto de vista da eficácia da seleção dos atributos, é uma alternativa válida, mas podem ocorrer problemas devida a natureza gulosa dos algoritmos, como já foi citado anteriormente.

A nossa proposta envolve, num mesmo algoritmo, a utilização de sequências alternadas de execução das duas modalidades de abordagem para seleção de atributos (*backward* e *forward*, com ou sem heurísticas). O mesmo pode iniciar com conjunto de atributos vazio ($F \leftarrow \emptyset$) e iterar inicialmente utilizando a abordagem *forward*, cuja abordagem nomeamos como algoritmo *Forward-Backward*; ou iterar sobre um conjunto de atributos inicialmente cheio ($F \leftarrow F_{max}$) e iterar inicialmente utilizando a abordagem *backward*, cuja abordagem nomeamos como algoritmo *Backward-Forward*. Estas mesmas proposições valem para os algoritmos baseados em heurísticas, nomeados respectivamente de algoritmo *Forward-Backward* com heurísticas e *Backward-Forward* com heurísticas, respectivamente.

A seguir, os passos de execução para a abordagem *Forward-Backward*:

1. Inicialize o conjunto de atributos $F \leftarrow \emptyset$, ou seja, com nenhum atributo contido no conjunto F.

2. Enquanto o valor do complemento de F for maior do que 1 ($|F^C > 1|$), ou seja, ainda existirem atributos a serem acrescentados ao conjunto F, fazer o seguinte:

(a) Para todos os atributos f pertencentes à F^C, simular a adição do mesmo no conjunto F, construindo o conjunto $F_{FB} = F \cup \{f\}$ e avaliar este novo conjunto, utilizando-se do algoritmo AbsProb-PI para calcular o valor $V(F_{FB})$.

(b) Encontrar o atributo f^a mais relevante do conjunto de estados abstratos, através da função $f^a = \arg\max_{f\in(F^C)} V(F_{FB})$, e adicioná-lo ao conjunto F, fazendo $F = F \cup \{f^a\}$ e eliminando f^a do conjunto F^C.

(c) Armazenar f^a em um vetor F_{add}

(d) Para todos os atributos f pertencentes à F^C, simular a remoção do mesmo no conjunto F, construindo o conjunto $F_{FB} = F_{FB} - \{f\}$ e avaliar este novo conjunto, utilizando-se do algoritmo AbsProb-PI para calcular o valor $V(F_{FB})$

(e) Encontrar o atributo f^b menos relevante do conjunto de estados abstratos, através da função $f^b = \arg\max_{f \in F} V(F_{FB})$.

(f) Caso $f^a \neq f^b$, eliminar o atributo f^b do conjunto F e do vetor F_{add} e repetir os passos (d), (e) e (f) até $f^a = f^b$

3. Como resultado, retornar a lista F_{add} de atributos adicionados, ordenados por relevância.

Segue os passos de execução para a abordagem *Backward-Forward*:

1. Inicialize o conjunto de atributos $F \leftarrow F_{max}$, ou seja, com todos os atributos contidos no conjunto F.

2. Enquanto o valor de $|F| > 1$, ou seja, ainda existirem atributos no conjunto F, fazer o seguinte:

(a) Para todos os atributos f pertencentes à F, simular a remoção do mesmo, construindo o conjunto $F_{BF} = F - \{f\}$ e avaliar este novo conjunto, utilizando-se do algoritmo AbsProb-PI para calcular o valor $V(F_{BF})$.

(b) Encontrar o atributo f^a menos relevante do conjunto de estados abstratos, através da função $f^a = \arg\max_{f \in F} V(F_{FB})$, e eliminá-lo do conjunto F, fazendo $F = F - \{f^a\}$.

(c) Armazenar f^a em um vetor F_{el}

(d) Para todos os atributos f pertencentes à F, simular a inserção do mesmo no conjunto F, construindo o conjunto $F_{BF} = F_{BF} \cap \{f\}$ e avaliar este novo conjunto, utilizando-se do algoritmo AbsProb-PI para calcular o valor $V(F_{BF})$

(e) Encontrar o atributo f^b mais relevante do conjunto de estados abstratos, através da função $f^b = \arg\max_{f \in (F^C)} V(F_{BF})$.

(f) Caso $f^a \neq f^b$, adicionar o atributo f^b ao conjunto F e ao vetor F_{el} e repetir os passos (d), (e) e (f) até $f^a = f^b$

3. Como resultado, retornar a lista F_{el} de atributos, ordenados por sua relevância.

Os mesmos passos são válidos para os algoritmos *Forward-Backward* com heurísticas e *Backward-Forward* com heurísticas, sendo que, assim como os algoritmos originais desenvolvidos por Bogdan e Silva (2012), a porção referente à ação *forward* destes algoritmos utiliza-se do vetor C (desconto acumulado esperado para o valor de recompensa obtido em cada $s \in S$ do MDP seguindo a política abstrata π_{ab}) que o algoritmo AbsProb-PI desenvolve para compor a política abstrata $\hat{\pi}_{ab}^{F_{f-}}$ e, em seguida, desenvolver os valores de $V(F_{f-})$, conforme seguem fórmulas:

$$\hat{\pi}_{ab}^{F_{f-}}(\sigma,\alpha) = \frac{\sum_{s \in S_\sigma} C(s)\hat{\pi}_{ab}^{F}(\sigma^F(s),\alpha)}{\sum_{s \in S_\sigma} C(s)}, V(F_{f-}) = \mathrm{b}^{0\mathrm{T}}(\mathrm{I} - \gamma \mathrm{T}^{\hat{\pi}_{ab}^{F_{f-}}})^{-1}\mathrm{r}$$

O mesmo é válido para a porção referente à ação *backward*, que utiliza-se do vetor de gradientes $W(\sigma,\alpha)$ e da política atual $\hat{\pi}_{ab}^{F}$ desenvolvidos pela execução do algoritmo AbsProb-PI, conforme segue:

$$V(F_{f+}) = \sum_{\sigma \in S_{ab}^{F_{f+}}} \max_{\alpha \in A_{ab}} W(\sigma,\alpha)$$

Como percebe-se pela composição dos algoritmos, a cada execução dos algoritmos *Backward-Forward* e *Forward-Backward* são executados os núcleos dos seus algoritmos de inspiração, tanto para as versões sem heurística quanto para as versões heurísticas, em relação a seus pares de origem. Portanto, a execução sempre será de, no mínimo, duas vezes a execução de uma das abordagens de origem; por exemplo, uma execução do algoritmo *Backward-Forward* será, no mínimo, duas vezes mais demorada do que a execução do algoritmo *Backward* somente. O que compensa esta abordagem mais demorada é que os resultados de otimização do conjunto de atributos permite melhores resultados do que a execução de uma abordagem não mista.

IV. Experimentos e Resultados

Para a realização de experimentos avaliativos, desenvolvemos no ambiente de desenvolvimento Matlab os algoritmos propostos por Bogdan e Silva (2012) *backward*, *forward*, *backward* com heurísticas e *forward* com heurísticas, além de nossas propostas realizadas com as abordagens mistas dos mesmos.

A. *Descrição dos ambientes utilizados*

Foi escolhido um problema simplificado de navegação robótica, onde o agente tem como ações disponíveis para a exploração "caminhar ao norte", "caminhar ao sul", caminhar a leste" e "caminhar a oeste", com 90% de chances de executar a ação e 10% de ir às outras direções, garantindo assim que, mesmo que custosamente, sempre o algoritmo tenha chances de chegar a sua meta proposta.

Foram desenvolvidos dois ambientes, "Original" e "Teste".

O ambiente original é composto por um conjunto ordenado de 273 possíveis estados, sendo apenas 155 alcançáveis por exploração. São subdivididos em estados "parede", "porta", "corredor" e "sala", conforme a imagem 1.

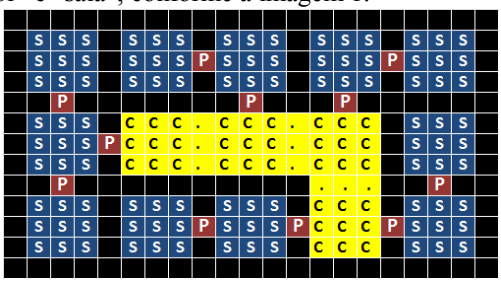


Imagem 1. Ambiente original de exploração, onde os estados marcados com "S" são pertencentes a uma sala, "C" ou "." a um corredor e "P" são portas.

Já o ambiente de testes envolve a mesma formulação do ambiente original, porém é composto de mais estados

possíveis (total de 1025, sendo 650 exploráveis), conforme a imagem 2.

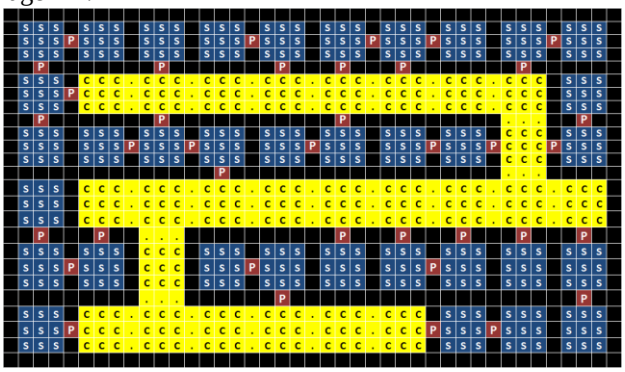


Imagem 2. Ambiente de exploração "teste", onde os estados marcados com "S" são pertencentes a uma sala, "C" ou "." a um corredor e "P" são portas.

No ambiente original, são designados 11 estados-meta, ou seja, são pontos no mapa (geralmente centros de salas) designados como uma tarefa a ser realizada, e a meta do explorador é justamente chegar até este ponto, utilizando suas ações padrão (caminhar ao norte, ao sul, a leste e a oeste), sendo que enquanto ele for explorando vai perdendo 1 ponto por iteração, forçando o algoritmo a procurar os estados-meta no menor tempo, visando a menor perda de pontos possíveis. Já no ambiente de teste o número de estados-meta aumenta para 35, e os valores de pontos perdidos por iteração e de alcance da meta mantém-se equivalentes em relação ao ambiente original.

O MDP descrito tem como atributos o mesmo conjunto descrito por Bogdan e Silva (2012) para a resolução dos algoritmos *backward* e *forward*, como segue:

TABELA 1. VARIÁVEIS REPRESENTANDO OS ATRIBUTOS (FEATURES)

Feature	Notation
see_empty_space	Es
see_door_far	Df
see_room	Sr
see_corridor	Sc
see_ambience	Sa
see_anything	An
in_room	In
near_to_goal	Ng
almost_near_to_goal	Ang

As propriedades de cada atributo são as seguintes:

- see_empty_space (Es): é um atributo que, quando ativo, representa que no estado corrente o explorador vê um espaço alcançável e vazio;

- see_door_far (Df): é um atributo que, quando ativo, representa que no estado corrente o explorador vê uma porta a mais de 3 passos de distância;

- see_room (Sr), see_corridor (Sc) e see_ambience (As) são atributos que, ao verificar proximidade dos itens citados (porta, corredor e ambiente/sala) retorna positivo;

- see_anything (An) é um atributo que retorna positivo quando see_room, see_corridor ou see_ambience retorna positivo;

- in_room (In): quando ativo, representa que o estado corrente está localizado dentro de uma sala;

- near_to_goal (Ng): quando ativo, representa que, no estado corrente, o explorador está a 5 passos (ou menos) da meta;

- almost_near_to_goal (Ang): quando ativo, representa que, no estado corrente, o explorador está entre 8 e 5 passos da meta.

Além dos atributos representados acima, para o conjunto de atributos {Es, Df, Sr, Sc, Sa, An} são considerados duas variantes, conforme nomenclatura adotada: (atributo)*_opposite_to_goal* e (atributo)*_ahead_to_go*. Assim, totalizamos 15 atributos a serem avaliados nos dois ambientes.

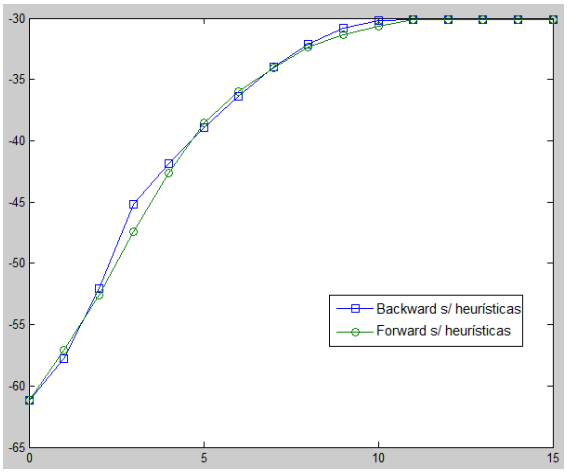
A descrição de dois ambientes, Original e Teste, foi necessária para a avaliação dos algoritmos em relação a eficácia dos conjuntos de atributos e políticas determinadas encontrados durante a etapa de aprendizado no ambiente Original para as tarefas designadas a ocorrerem no ambiente Teste; e pela proposição dos ambientes reflete a característica de utilização dos métodos de resolução de problemas com MDPs na realidade, onde em um ambiente simulado e pequeno são realizados os testes (para diminuição de custos computacionais), para posteriormente ter o aprendizado aplicado num ambiente de tamanho maior e com grau de complexidade igualmente grande.

B. Execução dos testes

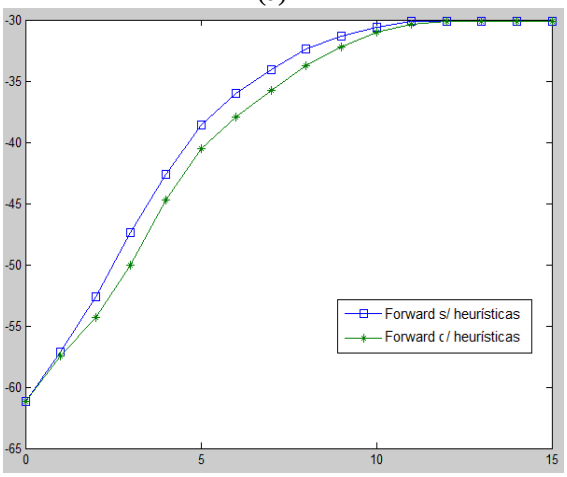
Os primeiros testes avaliaram os algoritmos baseados nos propostos por Bogdan e Silva (2012) em nossos dois ambientes propostos (Original e Teste). Para a execução do algoritmo AbsProb-PI determinamos o n_{passos} = 1000; ou seja, 1000 iterações a serem executadas e, em cada iteração, todos os estados-meta são considerados nos ambientes.

Para a realização das avaliações, foram realizados 20 testes para cada conjunto (Original e Teste), onde são selecionados metade dos estados-alvo para treinamento do conjunto e a outra metade é utilizada para avaliação das políticas e atributos.

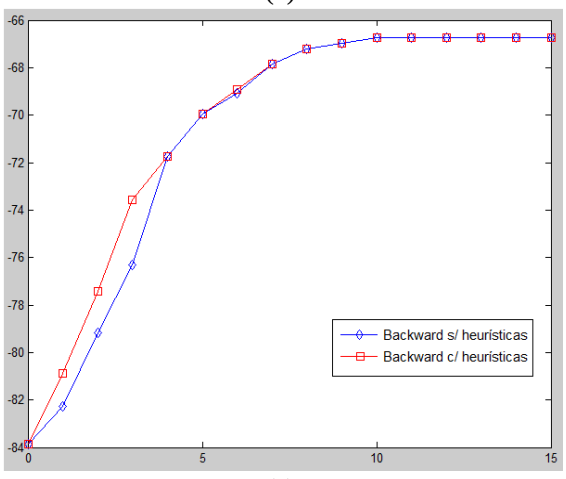
Na imagem 3, são apresentados os resultados das execuções dos algoritmos, sendo que, na escala horizontal, quanto mais próximo da extremidade direita, maior o número de atributos no conjunto de atributos considerado (F) e o valor vertical representa o valor do conjunto atual de atributos considerado (quanto maior o valor, melhor é o conjunto).



(a)



(b)



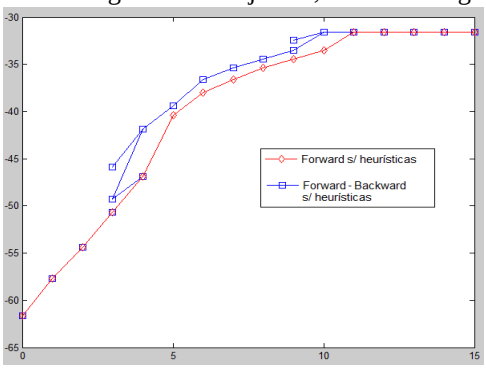
(c)

Imagem 3. Média das avaliações dos algoritmos forward com e sem heurísticas no ambiente Original (a) e comparativo de eficácia da política prescrita pelos algoritmos com e sem heurística na abordagem forward (b) e backward (c) no ambiente original, após 20 execuções.

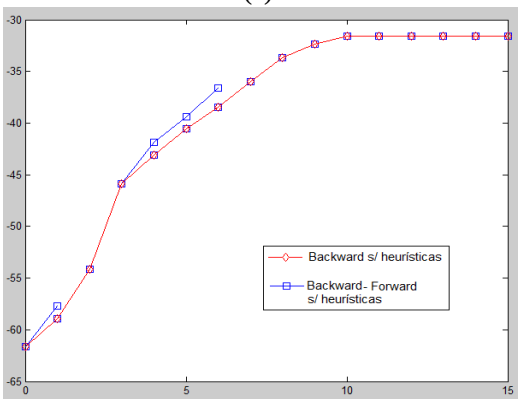
Os resultados obtidos nos gráficos da imagem 3 são semelhantes aos obtidos por Bogdan e Silva (2012) em seus experimentos, salvo as devidas adaptações referentes aos ambientes de testes e parâmetros de execução utilizados, que são ligeiramente diferentes entre si; por exemplo o valor adotado de n_{passos}, que nos testes da proposta original fora utilizado com o valor 280 e, em nossa proposta, valor 1000.

Conforme os resultados das execuções anteriores foram se adequando ao já estudado, partimos para a execução dos algoritmos próprios, tanto as versões baseadas em heurísticas quanto as que não eram baseadas neste modelo.

Através da análise dos gráficos fica visível a melhora de desempenho dos algoritmos conjuntos, conforme segue:



(a)



(b)

Imagem 4. Média das avaliações dos conjuntos de atributos com os algoritmos Forward comparado ao algoritmo Forward-Backward (a) e Backward comparado ao algoritmo Backward-Forward (b), no ambiente Original.

Conforme verificado tanto no gráfico (a) quanto no gráfico (b) da imagem 4, os algoritmos de abordagem mista realizam alternativamente a inserção e remoção de atributos, dependendo do valor do conjunto de atributos que as fórmulas de *backward* e *forward* fornecem adicionando e retirando elementos diversos do conjunto. Isto faz com que haja uma menor influência do fator limitante de eficiência aplicado pela abordagem gulosa dos dois algoritmos propostos por Bogdan e Silva (2012), que levava a diversos ótimos locais que nem sempre representavam a melhor opção para determinado conjunto de atributos selecionados, algo facilmente visível comparando, no gráfico (b) da imagem 4, o conjunto retornado pelo algoritmo *backward* sem heurísticas comparado ao algoritmo misto *Backward-Forward* no conjunto de seis atributos, onde verifica-se uma diferença de eficácia entre os dois conjuntos propostos de cerca de 1,5 ponto a mais para a abordagem mista.

TABELA 2. TEMPO DE EXECUÇÃO DOS ALGORITMOS ABORDADOS NO AMBIENTE ORIGINAL

Algoritmo	Tempo de execução (seg)
Backward sem heurísticas	1968,3 segundos
Forward sem heurísticas	1941,7 segundos
Backward com heurísticas	655,43 segundos
Forward com heurísticas	632,77 segundos
Backward-Forward sem heurísticas	5734,4 segundos
Forward-Backward sem heurísticas	5123,6 segundos
Backward-Forward com heurísticas	1509,2 segundos
Forward-Backward com heurísticas	1370,3 segundos

A utilização de abordagem mista demanda em média 2,77 vezes mais tempo de execução, comparada aos algoritmos *backward* e *forward* separadamente. Porém a utilização de heurísticas combinada a abordagens mistas consegue ser reproduzida, em média, em 73,6% do tempo que seria demandado para a execução dos algoritmos *backward* e *forward* separadamente, e em comparação a seus pares mistos sem heurística, este índice chega a promissores 26,53% de tempo de execução (ver tabela 2).

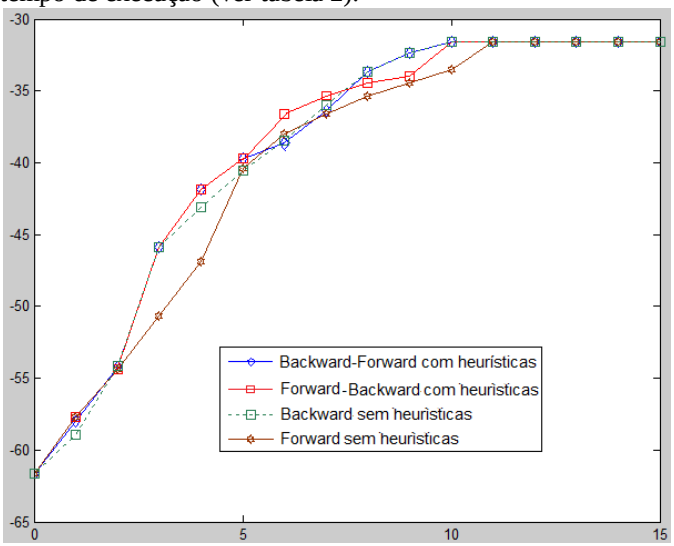


Imagem 5. Média das avaliações dos conjuntos de atributos com quatro modelos de abordagens do algoritmo de seleção de atributos.

Na imagem 5 pode-se ver a comparação entre a eficácia dos algoritmos mistos com heurísticas aos algoritmos *backward* e *forward* sem heurística, separadamente. Como podemos ver no gráfico, especialmente na faixa de 4 atributos selecionados, os algoritmos combinados, mesmo que se valendo de heurísticas para agilizar o processo de cômputo dos valores das políticas e atributos selecionados, conseguem ser melhores do que as abordagens *full* dos algoritmos *Backward* e *Forward* para certos conjuntos, e com poucos ou muitos atributos selecionados as abordagens mistas conseguem ser equivalentes às abordagens *full* supracitadas. Considerando que o tempo demandado para a execução de uma iteração dos algoritmos mistos com heurísticas seja menor do que o de *full Backward* e de *full Forward* e apresentam resultados satisfatórios, é considerada válida a utilização dos mesmos para solucionar problemas referentes a seleção de atributos em MDPs abstratos.

A imagem 6 compara a diferença entre a pior abordagem em um determinado conjunto de atributos e as outras abordagens realizadas. O gráfico têm as mesmas propriedades do gráfico da imagem 5, ou seja, na escala horizontal são representadas as quantidades de atributos naquela iteração e, na vertical, a diferença em pontos entre a pior abordagem e as outras.

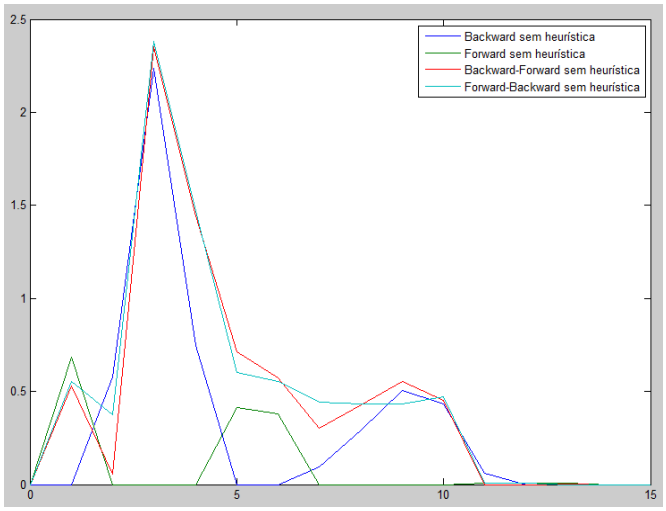
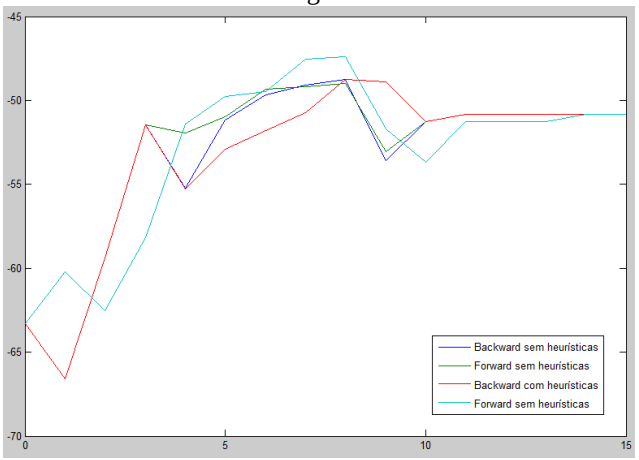


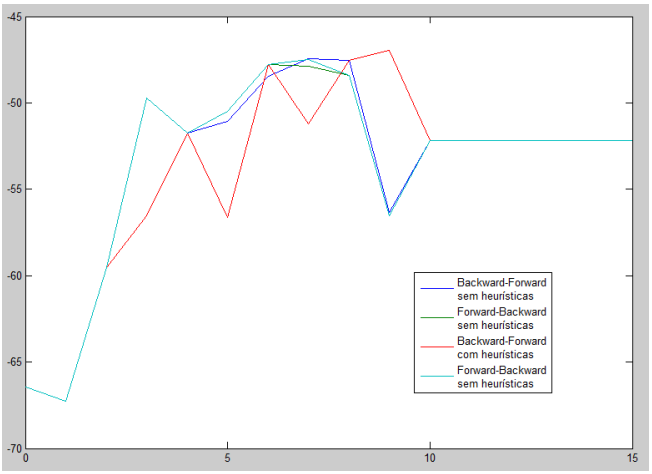
Imagem 6. Diferença computada entre a pior abordagem em um determinado conjunto de atributos e as outras abordagens, através da média de valores de políticas e atributos de 20 execuções para cada quantidade de atributos.

Com os dados apresentados, fica mais fácil notar que, na maior parte dos casos analisados, os algoritmos mistos são melhores do que os algoritmos *full*, sem heurísticas; por exemplo no conjunto entre 3 e 8 atributos ininterruptamente as abordagens mistas com heurísticas obtiveram melhor avaliação e menor custo computacional de tempo do que suas contrapartes.

No âmbito da transferência de conhecimento, foram realizados testes de migração da política computada no ambiente Original e avaliada no ambiente Teste. Para isso, foram realizadas duas execuções do algoritmo para cada valor de conjunto de atributos (de 0 a 15) e computada a média. Os resultados são exibidos na imagem 7.



(a)



(b)

Imagem 7. Valor do conjunto de atributos calculado no conjunto Original e aplicada no conjunto Teste para as oito abordagens estudadas, quatro com heurísticas e quatro sem heurísticas, sendo que em (a) são retratados os algoritmos não mistos e, em (b), as abordagens mistas.

É importante salientar que, apesar da imprecisão dos dados colhidos, devido a execução de apenas duas iterações dos algoritmos, os mesmos já apresentam comportamento semelhante ao observado por Bogdan e Silva (2012) em relação aos algoritmos propostos por eles. No caso, a partir de, aproximadamente, 2/3 da quantidade total de atributos, os algoritmos não conseguem executar mais uma política satisfatória de transferência de conhecimento entre os problemas, pois há *overfitting*. Já em relação à faixa entre 1/3 e 2/3 do total de atributos, temos uma variância entre as abordagens dos algoritmos e, com exceção de dois conjuntos de atributos (com 5 e 9 atributos, no exemplo do gráfico (b) da imagem 7), sempre obtêm uma política melhor do que a partir de 10 atributos (2/3 do total).

V. Conclusão

Apresentamos dois algoritmos que melhoram a abordagem anterior, no âmbito de seleção de atributos, ao agregar uma abordagem mista entre a técnica *backward* e *forward* propostos separadamente por Bogdan e Silva (2012) em estudos anteriores. Também realizamos experimentos para avaliar a qualidade das políticas apontadas pelos algoritmos criados e realizamos comparações de eficácia em relação aos algoritmos não mistos, obtendo uma melhor avaliação em relação aos mesmos, mesmo considerando a situação de comparação entre algoritmos mistos heurísticos e algoritmos não mistos e não heurísticos, onde obtivemos melhor qualidade da política gerada em um menor tempo de execução.

Como já foi analisada a eficácia dos conjuntos de abordagens mistas em relação aos algoritmos não mistos, é interessante realizar em futuras pesquisas a avaliação dos conjuntos de atributos obtidos no conjunto Original aplicados para treinamento e avaliação no conjunto de Testes e um maior refinamento na análise da migração da transferência de políticas entre os dois conjuntos bidirecionalmente (treinamento em Original e avaliação em Teste e vice versa).

Referências

Coelho, L. G. (2009) “Processo de Decisão Markoviano e Aprendizado por Reforço”. PCS 5019 – Probabilistic Methods in Robotics ans Vision, USP, Brasil.

Pellegrini, J., Wainer, J. (2007) “Processo de Decisão de Markov: um tutorial”. Instituto de Computação – Unicamp, Brasil.

Silva, V. F., Pereira, F., Costa, A. H. (2012) “Finding memoryless probabilistic relational policies for inter-task reuse”, Universidade de São Paulo, Brasil.

Bogdan, K., Silva, V. (2012) “Backward and Forward Feature Selection for Knowledge Transfer between MDPs”, Universidade de São Paulo, Brasil.

Otterlo, M. V. (2004) “Reinforcement Learning for Relational MDPs”. Em: Machine Learning Conference of Belgium and the Netherlands, p. 138-145 [S.1]

Mausam, A., Kolobov, A. (2012) “Planning with Markov Decision Processes: An AI Perspective”. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers.

Puterman, M. L. (1994) “Markov Decision Process – Discrete Stochastic Dynamic Programming”. John Wiley & Sons, Inc., New York/NY, EUA.

Kaelbling, L, P., Littman, M. L., Cassandra, A. R. (1998) “Planning and acting in partially observable stochastic domains”. Em: Artificial Inteligente, vol. 101, p. 99 – 134.

CLASSIFICAÇÃO AUTOMÁTICA DE FRUTAS POR ANÁLISE DE IMAGEM – O CASO DA MANGA TOMMY ATKINS

J. N. Carvalho, E. C. Gurjão, M. E. R. M. C. Mata, M. E. M. Duarte

Abstract - **Determination of fruit quality in the agricultural industry is mostly performed by evaluating the physical properties of these products. The accuracy and speed in determining aspects which are difficult to interpret is a major challenge of industrial automation. This paper presents a method for standardization, inspection and classification of fruits from image, estimating their approximate maturity. The Tommy Atkins mango is to be used as pilot study.**

Keywords - **Classification of fruits, ripeness, computer vision and image analysis.**

I. INTRODUÇÃO

AS frutas, assim como outros produtos agrícolas, estão sujeitas a danos e perdas ao longo de todas as fases de sua produção; porém, é na pós-colheita que se concentram os maiores prejuízos, que são devidos principalmente ao processamento, embalagem, manuseio e transporte inadequados, além de técnicas de conservação incipientes. Tudo isso pode vir a produzir alterações mecânicas, fisiológicas ou patológicas, degradando a qualidade do produto [1].

Dentre os processos da pós-colheita, a seleção e classificação de frutas vêm merecendo especial atenção, afinal, esse é o processo que decide se determinado produto está ou não com a qualidade exigida pelo consumidor. Nesse contexto, a avaliação precisa e rápida do estado de maturação dos produtos na indústria de alimentos representa indibitávelmente, uma das questões críticas neste setor, em virtude dos altos custos do tempo, exigidos por este processo [2]. Na tentativa de resolver esse problema, duas heurísticas básicas são adotadas: aumentar o efetivo de pessoal ou aplicar tecnologias de automação. O alto custo dessas tecnologias tem representado um obstáculo para o setor, simplesmente porque esta tecnologia não está ao alcance das pequenas e médias empresas dedicadas a esta indústria. Dessa forma, sistemas de baixo custo para análise de imagens, aplicados à classificação e seleção de frutos, podem vir a representar um diferencial na balança da competitividade desse exigente mercado. Esses sistemas são constituídos de métodos de análise não invasivos, ou seja, são muito adequados para ser utilizado na indústria alimentar. No processamento da manga, a seleção obedece a critérios definidos em padrões nacionais e internacionais que determinam as exigências para o mercado. No processo de exportação existente no Brasil, a manga não possui uma classificação específica, sendo suas características "ditadas" pelo importador; não obstantemente, a coloração característica da fruta deve ser feita selecionando frutas de cores semelhantes no processo de embalagem [19]. Os padrões definem ainda a correlação entre o estado de maturação e as cores da casca [3], [4]; alguns incluem ainda a avaliação das cores da polpa, não sendo considerados no presente trabalho, por se tratar de um procedimento destrutivo.

J. N. Carvalho, Universidade Federal da Paraíba (UFPB), Rio Tinto, Paraíba, Brasil, joelson@dce.ufpb.brBrasil

E. C. Gurjão, Universidade Federal de Campina Grande (UFCG), Campina Grande, Paraíba, Brasil, ecandeia@dee.ufcg.edu.br

M. E. R. M. C. Mata, Universidade Federal de Campina Grande (UFCG), Campina Grande, Paraíba, Brasil, mmata@deag.ufcg.edu.br

M. E. M. Duarte, Universidade Federal de Campina Grande (UFCG), Campina Grande, Paraíba, Brasil, elita@deag.ufcg.edu.br

II. OBJETIVOS

Este trabalho apresenta um método baseado em inspeção visual automática para a classificação de mangas da variedade Tommy Atkins (Mangifera indica L.) em padrões previamente definidos, através da avaliação da cor da casca. Um mecanismo de aprendizagem de máquina cria uma generalização dos modelos dos padrões de classificação a partir de exemplos apresentados, e utiliza essas generalizações para classificar novas amostras.

Um mecanismo elementar para inserção e utilização de padrões comerciais a partir de valores pré-estabelecidos também está incluído.

Técnicas de análise de imagem e computação gráfica são utilizadas para reduzir o tempo de processamento (throughput).

III. REFERENCIAL TEÓRICO

III.1 ETAPAS DO PROCESSAMENTO DE IMAGENS

O processamento de imagens envolve invariavelmente as seguintes tarefas (Fig. 1) [5]:

A *Aquisição de dados* é simplesmente a disponibilização da imagem na forma digital para a entrada do sistema.

O *pré-processamento* consiste na utilização de técnicas de computação gráfica, como transformações lineares e não lineares, que permitem ajustes de contraste, remoção de ruído, seleção de regiões de interesse,

reamostragem dos pixels em uma nova escala, extração de características de imagem para segmentação, etc.

A *segmentação* de imagens permite separar a imagem em regiões disjuntas através de critérios relevantes para a aplicação. Pode-se, por exemplo, separar um objeto de interesse do resto dos pixels da imagem separando-a em duas regiões distintas. A saída da segmentação pode ser a fronteira do objeto com seu exterior.

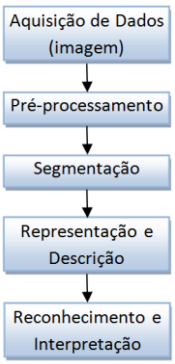


Figura 1. Etapas do processamento de imagens

A *representação* consiste das várias formas de armazenar o conjunto resultante da segmentação. Esta representação da imagem contém informações sobre a forma e a topologia dos objetos. Os processos de descrição e de representação, visam a extração de características ou propriedades que possam ser utilizadas na discriminação dos objetos em classes, características essas que se baseiam geralmente em atributos numéricos.

O *reconhecimento* associa um identificador (ou rótulo) a cada objeto segmentado, enquanto a *interpretação* associa um significado a objetos segmentados ou a um conjunto desses.

III.2 PADRÃO DE CORES CIE-RGB

Em 1931, o CIE ("Commission Internationale de l'Eclairage") adotou um sistema de representação do espaço de cores, que é constituído por três cores denominadas primárias – Vermelho, Verde e Azul. Os valores dessas cores no espectro visível são, respectivamente: 435,8 nm, 546,1 nm e 700 nm [17]. Esse padrão é denominado CIE-RGB, mais conhecido por RGB [6].

O modelo de espaço de cores RGB é provavelmente o mais usado entre os modelos de cores e é baseado na teoria tricromática da cor, de Thomas Young e Hermann Von Heimholt (1773-1829), cujo princípio é o de que diversos efeitos cromáticos são obtidos pela projeção da luz branca através dos filtros vermelho, verde e azul e pela superposição de círculos nas cores projetadas [7].

O espaço de cores RGB pode ser representado pela figura geométrica denominada de "cubo das cores", como apresentado na Fig. 2.

Assim, cada cor pertencente ao sistema RGB, pode ser identificada por uma tripla ordenada (R, G, B) de números inteiros, onde cada elemento possui valor entre 0 e 255.

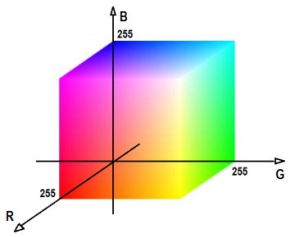


Figura 2. O cubo de cores RGB (abstração dos autores)

III.3 PADRÃO DE CORES CIE-HSL

Derivado do padrão RGB, o modelo de cores HSL também foi definido pelo CIE e consiste em decompor a cor de acordo com os seguintes critérios fisiológicos:

- Matiz ou tonalidade (H, *hue*), correspondendo à percepção da cor;
- Saturação (S, *saturation*), descrevendo a pureza da cor;
- Luminosidade (L, *lightness*), indicando a quantidade de luz da cor, ou seja o seu aspecto claro ou sombrio.

A representação gráfica do modelo HSL é formada por dois cones cuja soma dos eixos é 1,0 (normalizado), e cujas bases são coincidentes, como mostra a Fig. 3 [6].

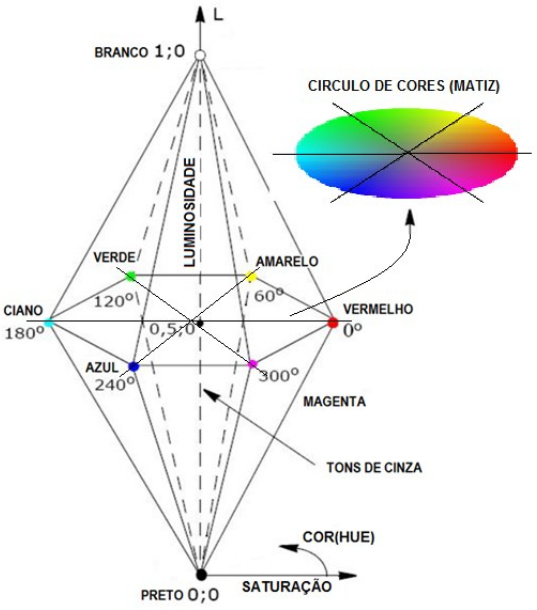


Figura 3. Abstração do cone e do círculo de cores do padrão HSL (adaptado pelos autores com base em [5]).

O matiz é determinado pelos pontos no círculo da borda das duas bases comuns aos cones,

A saturação também é normalizada, e varia de 0 a 1, conforme a distância ao eixo do cone. Quanto mais próxima da borda externa, maior a saturação da cor.

A luminosidade também varia de 0 (preto) a 1 (branco) ao longo do eixo comum aos dois cones, onde se encontra a escala de cinza.

Os matizes puros são encontrados no plano onde a luminosidade é igual a 0,5 (junção dos cones) e a saturação é igual a 1.

As cores puras primárias são defasadas em 120º entre si, enquanto as secundárias são defasadas em 60º das primárias.

O modelo HSL é um modelo de representação dito "natural", ou seja, próximo da percepção fisiológica da cor pelo olho humano, ao contrário do modelo RGB, que é fácilmente implementado em máquinas, mas de difícil utilização pelos seres humanos, visto que não se percebe naturalmente a cor como uma composição das três cores básicas, apesar de ser possível identificar com facilidade as cores básicas primárias (vermelho, verde e azul) e secundárias (amarelo, ciano e magenta) [6].

O modelo RGB pode ser convertido para o padrão HSL; dessa maneira, a tarefa de comparação de cores se dará utilizando apenas um valor (H), ao invés dos três componentes RGB. Além disso, as informações de saturação e brilho podem ser utilizadas separadamente.

III.4 DESCRITORES DE COR

Existem várias formas de representar uma imagem; as representações devem ter a capacidade de inscrever em algum meio, a informação contida na imagem, para que possa ser posteriormente recuperada e utilizada. Quando se utiliza a cor como característica para a representação de imagens, as formas contidas nas imagens são abstraídas. Apesar dessa perda de informação, torna-se possível analisar aspectos relevantes que envolvem as cores presente numa imagem, independentemente da posição física do objeto no espaço.

A cor é uma das características mais amplamente utilizadas em sistemas de análise de imagens por ser relativamente independente quanto ao tamanho, orientação e resolução da imagem e é computacionalmente menos cara quando comparada a outros descritores [8].

III.5 HISTOGRAMAS

Para representar a cor de um determinado objeto em uma imagem, diferentes espaços de cores, como o RGB e o HSL podem ser empregados. As imagens são formadas por um grande número de pontos coloridos indivisíveis (pixels) e a disposição desses pontos forma a imagem. Um *histograma de cor* é a representação da distribuição quantitativa de cores em uma imagem, e é construído a partir da contagem dos pixels de cada cor diferente em um determinado espaço de cor. Os histogramas podem ser definidos em R^2 ou R^3.

A construção de um histograma se baseia na construção de diversas regiões de cores, uma para cada cor presente na imagem, dentro de um determinado espaço de cores. Em seguida, efetua-se o somatório da ocorrência de uma cor na imagem, incrementando a cada vez, as regiões de cores. Enfim, o histograma nada mais é do que um gráfico que tem num dos seus eixos a representação numérica das cores de um sistema ou espaço, e no outro a quantidade de pontos para cada cor que uma determinada imagem contém. Dentre os descritores de cor, o Histograma é de longe o mais largamente utilizado em recuperação de imagens [9].

Os histogramas cumprem a tarefa de *representação da informação*, no sistema de processamento de imagens.

III.6 CLASSIFICAÇÃO E SISTEMAS DE APRENDIZAGEM DE MÁQUINA

A *classificação* permeia a quase totalidade das nossas atividades, principalmente as tarefas relacionadas à resolução de problemas; *classificar* é o processo de separar indivíduos, coisas e fenômenos, tomando como base algumas de suas características comuns. Os sistemas de aprendizagem utilizam mecanismos de classificação como base para sua operação. A classificação para um sistema desse tipo é o processo de atribuir a uma informação recebida (entrada do sistema) um nome que designa a classe à qual pertence. Esse processo implica em estabelecer uma *descrição* para essas classes. Como as classes se apresentam de formas diferentes dependendo do uso ou aplicação à que são submetidas, essas descrições poderão ser realizadas de várias formas.

Para que um sistema classificador possa atribuir a uma determinada entidade uma classe, se faz necessário que essa classe já tenha sido previamente definida. A definição das classes pode ser feita de várias formas [16]; uma delas é definir a classe a partir da soma ponderada de características relevantes do seu domínio. Para isso, isola-se um conjunto de características relevantes para o domínio da tarefa, constituindo um conjunto $C = \{C_1, C_2, ..., C_n\}$, onde cada elemento C_i, com $i = \{1, 2, ..., n\}$ representa o valor de um parâmetro relevante. Associa-se a cada C_i um valor p_i (discreto ou contínuo), como um peso que exprime a existência ou a importância da característica associada. Pode-se então definir uma classe como sendo uma função de avaliação sobre essas características, ou seja:

$$f(C) = \sum_{i=1}^{n} C_i p_i .$$

O número n representa a quantidade de elementos utilizados no conjunto de características.

III.7 APRENDIZAGEM INDUTIVA

A aprendizagem indutiva é caracterizada pelo uso da indução nos processos de classificação. As técnicas ou métodos de indução a serem utilizadas para a construção de classes dependem da maneira como estão descritas

essas classes.

A A*prendizagem a partir de Exemplos*, também chamada *Aquisição de Conceitos*, é um tipo de sistema de aprendizagem indutiva, que identifica caracterizações de alguns objetos (situações, processos, etc.) pré-classificados por um instrutor (humano ou não) em uma ou mais classes. A hipótese induzida pelo sistema pode ser caracterizada como uma regra de reconhecimento conceitual, de forma que se um objeto for reconhecido por esta regra, então esse objeto pertence à classe ou conceito respectivo [10], [16].

Os parâmetros utilizados na aprendizagem ou na definição de padrões cumprem a tarefa de *reconhecimento* no processamento de imagens. A aprendizagem desses parâmetros pode ainda ser realizada de maneira adaptativa; a adaptabilidade sugere a capacidade de modificação do conjunto de regras aprendidas em resposta à entrada de novos objetos com diferentes características submetidos durante o processo de aprendizagem, ou mesmo autoajustes no conjunto de parâmetros.

III.8 PADRÕES COMERCIAIS DE CORES PARA A MANGA

A utilização de padrões de cores para selecionar frutas pelas cores da casca é aplicada tanto para determinar a maturação da fruta, quanto para atender a critérios mais subjetivos, como a estética.

No caso da avaliação da maturação, os padrões são definidos em níveis, que indicam a aparência da fruta em determinado estágio de seu desenvolvimento. O estado de maturação de uma fruta não é um fenômeno linear, e mais, pode ser modificada e ter sua relação aparência x tempo modificada, com o uso de técnicas agrícolas de aceleração e retardo do amadurecimento; como a imagem da fruta também é afetada por essas técnicas, a indicação do estado (e não do tempo) de maturação a partir da imagem da fruta torna-se possível.

IV. REVISÃO BIBLIOGRÁFICA

Os padrões são geralmente estabelecidos pelo comprador, que dita os critérios de avaliação. Um desses padrões está descrito em [11]; ele estabelece faixas de cobertura cromática para a manga Tommy Atkins, divididas em cinco níveis que vão da fruta totalmente verde, passando por faixas intermediárias de maturação, até a fruta madura, como mostra a Fig. 4.

Existem várias escalas de cores para a casca para diferentes variedades de manga; alguns desses padrões são adotados para uso comercial em vários países [12], [13].

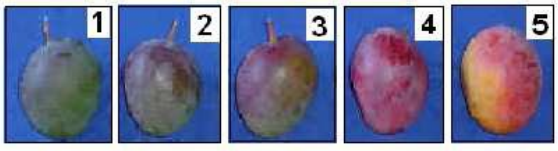


Figura 4. Conjunto de mangas associadas aos níveis do padrão Embrapa (Fonte: [11] disponibilizado p/ D. P. em www.cpatsa.embrapa.br).

Este padrão define a sua função de pertinência como é mostrado na Tab. 1:

TABELA 1
NÍVEIS DO PADRÃO EMBRAPA

NÍVEL	% COR VERDE	% COR VERMELHO	% COR AMARELO
1	100	-	-
2	75	25	-
3	50	50	-
4	25	75	-
5	-	-	25

Outro padrão para avaliação da maturidade da manga a partir da coloração da casca foi definido pela avaliação do percentual de cor vermelha na manga [14]; esse padrão denomina-se "Escala de Blush para coloração da casca (EBCC)". A escala é enumerada de 1 a 5, de maneira similar ao padrão Embrapa, onde cada número corresponde a uma faixa percentual da cor vermelha, como mostra a Tab. 2.

TABELA 2
NÍVEIS DA ESCALA EBCC

NÍVEL ESCALA BLUSH	% COR VERMELHA
1	0
2	0 - 25
3	25 - 50
4	50 - 75
5	75 - 100

A norma técnica NTC 5139 [15] é um padrão Colombiano para classificação da manga segundo seu estágio de maturação. Também define cinco níveis, sendo em sua essência, idêntico ao estabelecido pela Embrapa.

Embora os padrões acima apresentados sejam normas comerciais já estabelecidas e utilizadas, a interferência de fatores ambientais e nutricionais no desenvolvimento da casca das mangas pode não obedecer aos mesmos; isso é particularmente válido para a variedade Tommy Atkins. Por esse motivo, recomenda-se que os beneficiadores de manga adaptem sua própria escala de cores, com base na associação com outros indicadores de colheita [11], [2], [18], [19].

V. CAPTURA DE IMAGENS

Para realizar a seleção de mangas utilizando como critério a cor da casca, torna-se necessário estabelecer um padrão de referência; esse padrão permitirá ao agente selecionador, realizar uma comparação para identificar a manga segundo critérios pré-estabelecidos. Esta comparação envolverá os componentes HS e L. Embora a componente L represente apenas o aspecto do Brilho, considera-se sua inclusão relevante, pois ao amadurecer,

os frutos perdem água e sua casca se torna mais opaca, influenciando esse parâmetro.

Uma vez que seja estabelecido o padrão, existirá o risco de erros humanos na seleção, porquanto a percepção de diferenças em escala de cores não é uma tarefa trivial, principalmente sem aparelhos de mensuração adequados. Isso realça a importância dos métodos automáticos de classificação.

V.1 CAPTURA DE IMAGENS

O processo de classificação é iniciado com a captura de imagens da manga; duas imagens de cada amostra são necessárias, uma para cada lado da manga em estado de repouso, como mostra a Fig. 5.

Figura 5 – Imagens capturadas dos dois lados (A e B) de uma mesma manga.

Para capturar as imagens da manga, foi construída uma câmara escura, que proporciona um ambiente controlado, com iluminação adequada, distância da lente ao objeto invariável, posicionamento do objeto orientado pelo eixo (suporte em "V"), com uma câmera digital de média resolução (5,0 Mpixels). A Fig. 6 apresenta os aspectos externo e interno dessa câmara.

Figura. 6– Câmara para aquisição de imagens.

V.2 CONVERSÃO DO ESPAÇO RGB EM HSL

As imagens capturadas possuem um sistema de representação de 8 bits (RBG), que é padrão da câmara digital utilizada. Como se deseja imagens com a representação no padrão HSL, uma conversão deve ser feita. Inicialmente, realiza-se a normalização das cores do espaço RGB, da seguinte maneira [5]:

Efetuando o somatório dos valores das componentes do espaço RGB, obtém-se:

$$\Delta = R + G + B$$

Em seguida, a normalização transforma cada valor RGB em uma fração pertencente ao intervalo [0,1]. Cada componente de cor pode agora ser descrito da seguinte maneira:

$$r = \frac{R}{\Delta}, \qquad g = \frac{G}{\Delta}, \quad b = \frac{B}{\Delta}$$

A componente Cor (H) é dada por:

$$H = \begin{cases} \theta & se\ b \le g \\ 360 - \theta & se\ b > g \end{cases}$$

Onde:

$$\theta = cos^{-1}\left[\frac{\frac{1}{2}[(r-g)+(r-b)]}{[(r-g)^2+(r-b)(g-b)]^{1/2}}\right]$$

A componente Saturação (S) é dada por:

$$S = 1 - \frac{3}{\Delta}[\min(r+g+b)]$$

Finalmente, a componente Luminosidade (L) é dada por:

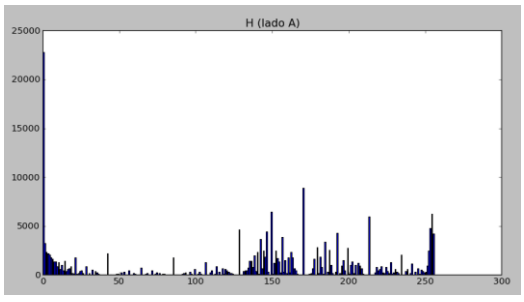
$$L = \frac{1}{3}(r+g+b)$$

Agora os valores estão convertidos para o espaço HSL.

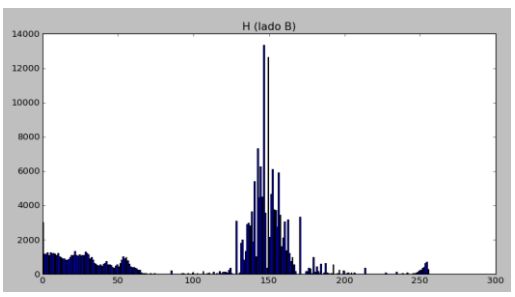
V.3 HISTOGRAMAS DAS IMAGENS

Os histogramas das imagens são obtidos individualmente, para cada componente do modelo, sendo descritos por matrizes que representam a imagem. Em cada ponto dessa matriz, encontra-se o valor HSL correspondente à cor do pixel na coordenada. Um gráfico representando os valores da componente H das imagens A e B da Fig. 5, bem como a soma dessas componentes, são mostradas na Fig. 7.

(a)



(b)



(c)

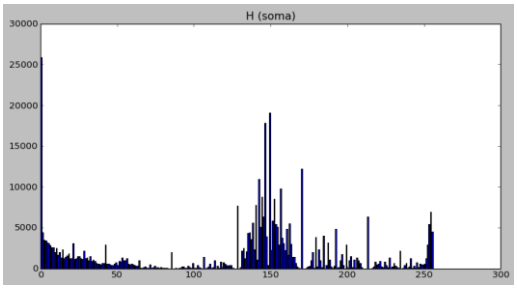


Figura 7. (a) Histograma da componente H da Fig. 5-A; (b) Histograma da componente H da Fig. 5-B; (c) Histograma soma da componente H das Figs. 7-A- e 7-B.

V.4 REDUÇÃO DO ESPECTRO DE CORES.

Com o objetivo de atenuar o custo de processamento, pode-se reduzir o espaço de varredura dos parâmetros HSL durante o cálculo dos histogramas. Isso pode ser feito porque o histograma incorpora todo o espectro de cores Fig. 8, e portanto, pode-se desprezar a faixa do espectro cuja cor (H) não seja observada em qualquer análise possível. Para o caso das mangas Tommy, as cores próximas do azul e do ciano podem ser obviamente descartadas, já que não são registradas em qualquer evento pós-colheita dessa fruta. Como a representação da imagem é um vetor, basta extrair dele os elementos cujos valores estejam próximo às referidas cores Fig. 9, resultando em um histograma reduzido, como mostra a Fig. 10. A não linearidade do eixo horizontal não prejudica a representação.

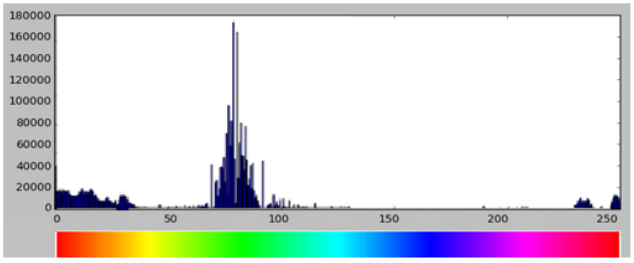


Figura 8. Histograma da imagem de um objeto, composto por todas as cores do espectro HSL.

Observando a Fig.8, pode-se verificar a irrelevância dos valores referentes às cores azul e ciano na distribuição de cores de uma manga Tommy Atkins.

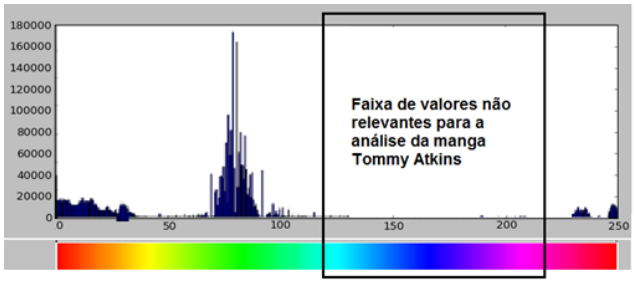


Figura 9. Faixa do espectro cujos valores não são relevantes para a análise.

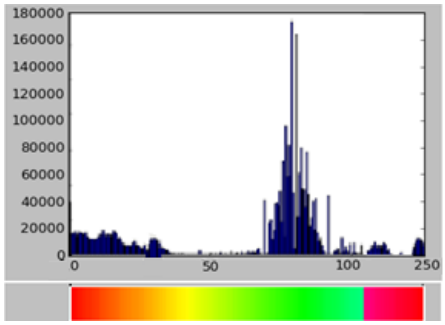


Figura 10. Histograma resultante da eliminação das cores próximas ao azul

Com isso, pode-se reduzir uma quantidade de tempo de processamento considerável, melhorando o desempenho do sistema de análise.

V.5 PERTINÊNCIA DE UMA AMOSTRA A UM PADRÃO

Para verificar o nível de uma determinada manga num determinado padrão, torna-se inicialmente necessário especificar o padrão. Isso é feito de maneira bastante objetiva, descrevendo os percentuais de cores do padrão e associando o valor do nível para cada percentual de cores. Considerando o padrão definido pela Embrapa, verifica-se que os níveis são definidos pelo percentual de cobertura de três cores: Verde, Vermelho e Amarelo. A avaliação consistirá na comparação do somatório de componentes de cada cor retirada do histograma da manga a ser classificada, a conversão desse resultado em percentuais de cobertura para cada cor e finalmente, a comparação com a tabela do padrão, com a indicação do respectivo nível.

TABELA 3
VALOR ANGULAR DAS CORES VERMELHO, AMARELO E VERDE NO ESPECTRO HSL

ÂNGULO	COR
330º	Magenta-vermelho
360º/0º	Vermelho
30º	Vermelho-amarelo
60º	Amarelo
90º	Amarelo-verde
120º	Verde
150º	Verde-Ciano

O algoritmo do programa que permite informar ao sistema qual o padrão a ser utilizado, é apresentado no Anexo A – Fluxograma I.

O algoritmo que classifica uma manga, de acordo com o padrão informado, é mostrado no Anexo A - Fluxograma IIa, que associa o intervalo da cor dentro do espaço HSL às cores definidas no padrão, e completado no Anexo B - Fluxograma IIb, que identifica em qual nível do padrão estabelecido se encontra uma fruta, cujas imagens são fornecidas.

Percebe-se que a classificação através da informação do padrão a ser obedecido, não é de todo direta, pois deve ser levado em consideração o fato de que as imagens das frutas não apresentam cores puras apenas, mas uma grande quantidade de cores diferentes. A predominância de um determinado espectro de cor é o que deve ser considerado. Assim, não se pode descartar as cores não presentes no padrão, pois algumas dessas cores representam uma composição com uma das cores lá definidas. Veja na tabela 3, que entre duas cores diferentes, existem nuances que combinam essas cores.

Padrões desse tipo podem ainda ser definidos a partir de exemplos de amostras. Esta abordagem pode ser considerada um exemplo da aplicação da Aprendizagem de Máquina, num dos seus níveis mais elementares.

V.6 CONSTRUINDO UM PADRÃO A PARTIR DE EXEMPLOS

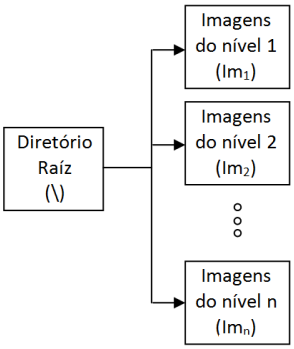


Figura 11 – Esquemas de diretórios para armazenar imagens dos frutos do padrão.

Além da possibilidade da utilização de um padrão de classificação já existente, o presente sistema permite a construção de um padrão definido pelo usuário. Para isso, o critério de classificação do padrão deve ser inicialmente estabelecido, de maneira que se possa identificar visualmente, um fruto pertencente a um dos níveis definidos no padrão. Assim, basta definir o número de níveis e submeter exemplares de frutos pertencentes a cada um dos níveis ao sistema. Um diretório será criado, com a criação de n subdiretórios, onde n é o número de níveis do padrão (Fig. 11); este algoritmo está apresentado no Anexo B – Fluxograma III.

Os algoritmos para definição do padrão, a partir de amostras de mangas é apresentado no Anexo B – Fluxograma IIb e no Anexo C – Fluxograma Va.

Os modos de operação do sistema são apresentados na Fig. 12 e na Fig. 13.

O modo manual classifica trivialmente as mangas de acordo com níveis e critérios fornecidos pelo usuário. A tarefa principal do sistema é identificar pelas imagens o nível ao qual pertence a fruta e inserir as imagens da mesma no diretório que representa esse nível.

No modo automático, o sistema constrói sua própria representação genérica dos níveis, a partir de imagens inicialmente apresentadas ao sistema (entrada de treinamento); posteriormente, o sistema será capaz de identificar os níveis de novas amostras apresentadas ao mesmo (entrada de uso).

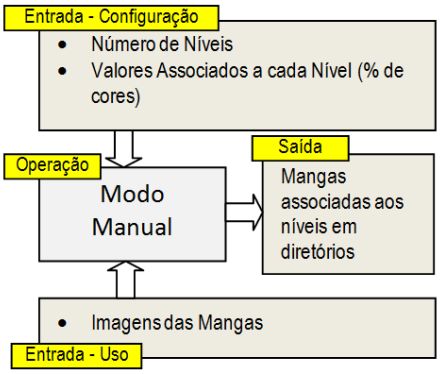


Figura 12 – Esquema funcional do sistema no modo manual.

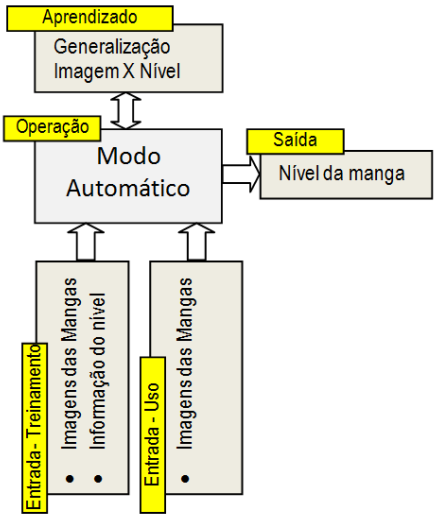


Figura 13 – Esquema funcional do sistema no modo automático.

V.7 MÉTODO DE CLASSIFICAÇÃO

Uma vez estabelecido o padrão, este pode ser utilizado para classificar outros frutos, através da comparação com as imagens do mesmo. Isso é feito da seguinte maneira: inicialmente, é extraído o histograma médio das imagens (Fig. 14), de maneira que se obtenha n histogramas médios; um para cada nível do padrão. O algoritmo para esta tarefa á apresentado no fluxograma do Anexo IV.

O histograma médio é obtido com a soma de cada componente HSL das imagens, para os 256 diferentes matizes, dividindo os resultados pelo número de imagens.

O passo seguinte é realizar a comparação da imagem capturada da fruta a classificar e a comparação do seu histograma com os histogramas médios encontrados no passo anterior.

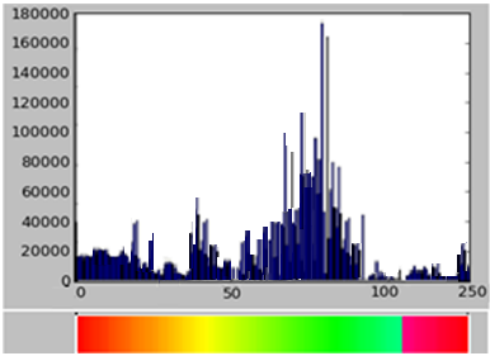


Figura 14 – Histograma médio extraído a partir de um certo número de amostras.

Inicialmente, as imagens dos lados da fruta são capturadas e seus respectivos histogramas são somados. O resultado da soma é então comparado a cada um dos histogramas médios extraídos dos diferentes níveis do padrão. Esta comparação é determinada através do cálculo da distância entre os valores de cada componente de H, S e L, e a função que minimiza a distância é definida pelo cálculo do erro médio quadrático, que deve ser encontrado para cada componente do espaço de cor, através das seguintes expressões:

a. Erro médio quadrático da componente H:

$$D_{ImH,MH} = \sqrt{\frac{1}{n}\sum_{j=0}^{255}[imH(j) - MH(j)]^2}$$

b. Erro médio quadrático da componente S:

$$D_{ImS,MS} = \sqrt{\frac{1}{n}\sum_{j=0}^{255}[imS(j) - MS(j)]^2}$$

c. Erro médio quadrático da componente L:

$$D_{ImL,ML} = \sqrt{\frac{1}{n}\sum_{j=0}^{255}[imL(j) - ML(j)]^2}$$

Uma vez obtida as distâncias pelo erro, os testes sobre as mesmas é realizado em cascata, ou seja: caso a comparação da componente H resulte numa igualdade, então a comparação se realiza sobre a componente S; de maneira similar, se a igualdade persistir, a comparação se realizará sobre a componente L.

O algoritmo do Anexo C – Fluxograma Va, mostra como a comparação é realizada.

Finalmente, deve-se encontrar a distância mínima entre a imagem e os *n* níveis do padrão. Este procedimento está descrito no algoritmo do Anexo D – Fluxograma Vb.

Pode-se inferir que, quanto maior o número de mangas apresentadas ao sistema, melhor a precisão da decisão do mesmo.

V.8 ESPECIFICIDADES DO SISTEMA

O sistema desenvolvido possui as seguintes características:

- É não destrutivo, ou seja, as frutas não são danificadas para a análise.
- Utiliza Histogramas para representar a informação.
- Emprega o paradigma de aprendizagem de máquina a partir de exemplos, criando generalizações de objetos para os níveis dos padrões.
- Emprega o erro médio quadrático como medida de distância (peso da função de aprendizagem).
- A adaptabilidade se verifica no ajuste dos histogramas médios; cada uma das características dos frutos apresentados na entrada reajusta o histograma médio para reconhecer objetos da classe (nível do padrão).
- Utiliza as três componentes do sistema de cores HSL para decisão, o que o torna mais próximo das percepções visuais humanas.
- Evita o pré-processamento, com a avaliação de cores, ignorando formas, além de utilizar um ambiente controlado para aquisição de imagens, com foco fixo, fundo homogêneo e alinhamento mecânico do objeto.
- Filtra o espectro de cores dos histogramas, otimizando o tempo de resposta.
- Permite a construção de padrões através da técnica de aprendizagem por exemplos; a aprendizagem se dá através da construção de um histograma de referência para cada nível e um peso que indica a distância entre um objeto e seu nível, empregando o método do erro quadrático médio.
- Evita a indecisão, através da utilização em cascata das dimensões HSL. Em caso de dois objetos idênticos em todas essas dimensões, aplica-se a heurística de decidir pelo nível imediatamente superior.
- Pode ser aplicado a vários outros tipos de frutas.
- Pode ser adaptado para selecionar mangas em uma esteira linear, implementando o módulo de decisão.
- O sistema foi inicialmente codificado na linguagem PYTHON, mas uma versão em C++ já está sendo feita para com o propósito de aumentar a velocidade e viabilizar dispositivos de automação industrial.

V.9 PROCEDIMENTOS DE AVALIAÇÃO E RESULTADOS

Foram adquiridas três caixas de mangas Tommy Atkins no mercado central de Campina Grande-PB; as mangas eram provenientes de Petrolina-PE, com diferentes estados de maturação. As frutas totalizaram 180 unidades, das quais 10 foram descartadas por apresentarem danos físicos relevantes. As amostras foram lavadas em água corrente e enxugadas. Logo após, com o auxílio da câmara de aquisição de imagens, foram fotografadas em posição de repouso em seus dois lados, em intervalo de 72 horas, até que as mangas atingissem o estado de maturidade; cinco sequencias foram registradas. As mangas ficavam armazenadas à sombra, em temperatura ambiente com variação diária dentro do intervalo de 20ºC a 29º C. Cada uma delas foi marcada para fins de indexação. Ao final, 1600 fotos foram obtidas (Fig. 15).

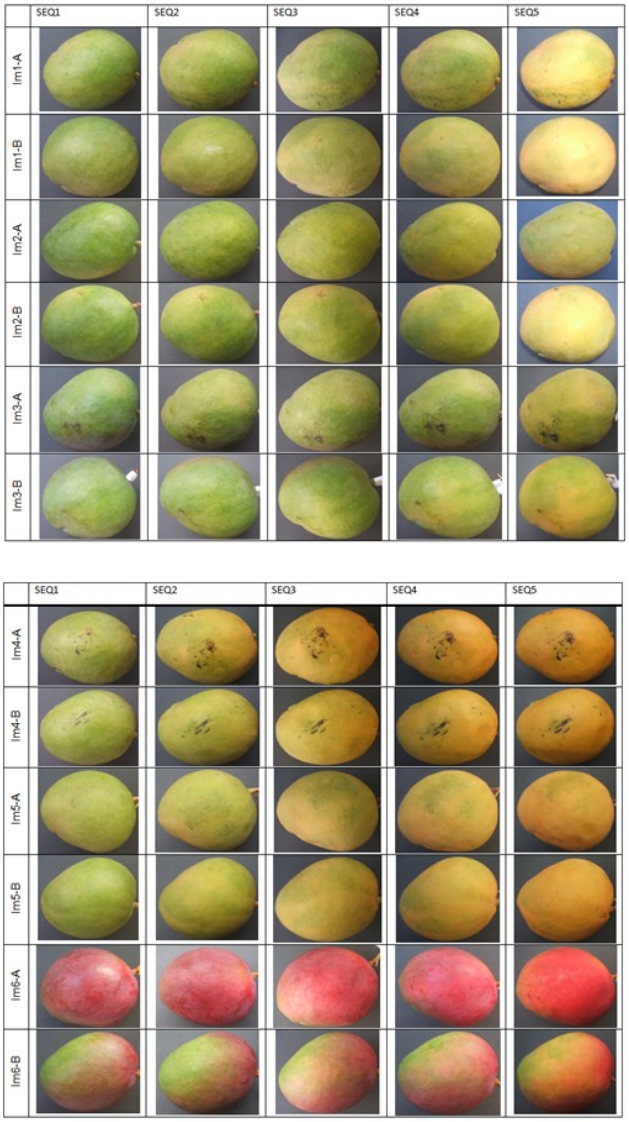


Figura 15. Algumas das sequencias típicas de imagens capturadas de mangas Tommy Atkins em diferentes estágios de maturação.

O método para criação de um padrão através da simples informação do número de níveis e dos intervalos referentes a cada nível foi testado. Submetendo 80 mangas escolhidas aleatoriamente à análise do sistema para o padrão EMBRAPA, todas foram classificadas em diferentes níveis, conforme a Tab 4. Resultados foram verificados por análise visual e diferenças (erro) entre a decisão da máquina e a humana não foi registrada para as amostras. A análise foi realizada em dois momentos: no 1º dia e no 15º dia da análise.

TABELA 4
RESULTADOS DO CLASSIFICADOR DO PADRÃO EMBRAPA DIA 1

NÍVEL	Nº DE MANGAS	ERROS
1	89	0
2	39	4
3	28	5
4	11	0
5	3	0

TABELA 5
RESULTADOS DO CLASSIFICADOR DO PADRÃO EMBRAPA DIA 15

NÍVEL	Nº DE MANGAS	ERROS
1	4	0
2	12	3
3	32	5
4	10	1
5	112	0

Observando-se a Tab.5, nota-se que no 15º dia, a maioria das mangas estavam classificadas no nível 5; isso porque a grande maioria delas estava com mais de 25% da sua área com a cor amarela - o sistema considerou valores da cor amarela com mais de 25% como maduras; essa heurística não está especificada no padrão EMBRAPA, mas foi adaptada pelo fato das frutas transmutarem para a cor amarela em função do amadurecimento. Foi observado que quatro mangas permaneceram verdes, devido ao fenômeno de encruamento. Foi observada uma pequena diferença entre a classificação do sistema (automática) e a classificação humana, com erro inferior a 1%.

Utilizando as mesmas imagens coletadas das 160 amostras do experimento anterior, foi gerado um padrão de cinco níveis utilizando o modo de aprendizagem de padrões. Ao final, foi calculado o percentual de distribuição das cores da manga, como mostra a Tab. 6.

TABELA 6
VARIAÇÃO DA COBERTURA DO MATIZ (H) EM FUNÇÃO DO TEMPO

NÍVEL	VERMELHO (%)	VERDE (%)	AMARELO (%)
1	5,02	92,01	2,97
2	4,89	76,41	18,7
3	6,16	31,86	61,98
4	4,12	9,52	86,36
5	3,98	2,36	93,66

Para verificar a consistência dos resultados, mais 60 mangas foram adquiridas do mesmo fornecedor e foram analisadas pelo sistema, no modo de uso, durante o mesmo intervalo de tempo (15 dias) e condições climáticas. Os percentuais foram registrados de acordo com a Tab. 7.

Comparando os resultados com os valores obtidos com o da Tab. 6, obteve-se a distância (erro) mostrada na Tab. 8:

TABELA 7
VARIAÇÃO DA COBERTURA DO MATIZ (H) EM FUNÇÃO DO TEMPO

NÍVEL	VERMELHO (%)	VERDE (%)	AMARELO (%)
1	7,33	90,33	2,34
2	7,33	82,25	10,42
3	7,33	28,9	63,77
4	6,25	4,66	89,09
5	4,66	3,12	92,22

TABELA 8
ERRO % (DISTÂNCIA ENTRE OS VALORES DAS TABS. 6 E 7)

NÍVEL	VERMELHO	VERDE	AMARELO
1	2%	2%	1%
2	2%	6%	8%
3	1%	3%	2%
4	2%	5%	3%
5	1%	1%	1%

O Erro médio calculado é de aproximadamente 3%. Este resultado indica que o sistema foi capaz de criar uma representação genérica para um conjunto amostra, e que essa representação foi suficiente para classificar novos elementos dentro dos níveis criados. Os erros verificados são característicos do pequeno número de amostras que constituiu o treinamento inicial do sistema. Caso o conjunto de amostras apresentados para classificação posteriormente tivesse sido muito diferente do conjunto de treinamento, o erro poderia ser maior.

VI. CONCLUSÃO

Ao final deste trabalho, foi possível deduzir de maneira conclusiva, que a construção de um sistema automático para classificar mangas pela análise de cores das cascas, tarefa reconhecidamente extenuante e repetitiva realizada por humanos, é viável não só porque o sistema substitui a atividade humana penosa, por atividades mais elevadas, como a de supervisão, mas também pelo fato da tecnologia utilizada ser implementada a baixo custo, proporcionando a melhoria do processo pós-colheita da manga Tommy Atkins e de outros frutos, que porventura venham a ser inseridos no sistema.

Verificou-se também, como já era esperado, que a performance do sistema melhora com a experiência; além disso, existe a possibilidade da implementação de um sistema de comunicação entre classificadores distintos; nesse caso, a experiência adquirida pelas máquinas poderia vir a ser compartlhada. Isso sugere a construção de uma sociedade de agentes classificadores remotamente dispostos.

A construção deste classificador provocou a demanda pela criação de sistemas complementares, como por exemplo, um estimador de propriedades físicas, definindo os próximos projetos dos autores.

BIBLIOGRAFIA

[1] DA SILVA, F.M., MORAIS, A. M. M. B. *Boas Praticas Pos-Colheita para Frutos Frescos,* ESB/UCP/ORGAL, Porto, Portugal, 2000

[2] MANICA, I.; ICUMA, I.M.; MALAVOLTA, E.; RAMOS, V.H.V.; OLVEIRA, M.E.; CUNHA, M.M.; JUNQUEIRA, N.T.V. *Tecnologia, produção, agroindústria e exportação da manga*. Ed. Cinco Continentes, Porto Alegre – RS, 2001.

[3] AWAD, M. *Fisiologia pós-colheita de frutos*. Ed. Nobel, São Paulo - SP, 1993.

[4] FILGUEIRAS, H. A. C.. *Manga – Pós-Colheita; Embrapa agroindústria Tropical (Fortaleza, CE),* Embrapa Comunicação para Transferência de Tecnologia; Brasilia, 2000.

[5] GONZALES, R. C., WOODS, R. E.: *Processamento Digital de Imagens – 3ª Edição*. Pearson Prentice Hall, São Paulo-SP. 2010.

[6] YOUNG, T. *The Bakerian Lecture: On the theory of Light and Colours*. Phil Trans. Royal Society London, 1802

[7] GOMES, J., VELHO, L.: Computação Gráfica: Imagem. IMPA/ Solgraf Publicações Ltda. Rio de Janeiro, RJ, 2002.

[8] BENDER, T.C. Classificação e Recuperação de Imagens por Cor Utilizando Técnicas de Inteligência Artificial. 2003. 126 f. Dissertação (Mestrado) – UNISINOS, Departamento de Computação, Sâo Leopoldo, 2003.

[9] TORRES, R.; FALCÃO, A.; DINSTEIN, I. *Content-Based Image Retrieval: Theory and Applications*. Revista de Informática Teórica e Aplicada, v. 13, n. 2, 2006.

[10] RICH, E. *Inteligência Artificial*. São Paulo: McGRAW-HILL, Inc., 1988.

[11] ASSIS, J. S, LIMA, M.A.C., .*Produção Integrada de Manga: Manejo Pós-Colheita e Rastreabilidade*. EMBRAPA - Circular Técnica 12/2008 – Petrolina – PE. 2008.

[12] EMEX. Norma de calidad para mango fresco de exportación. Zapopan: CIAD, Jalisco – Mexico. 1998.

[13] GTZ Deutsche Gesellschaft für Technische Zusammenarbeit. *Manual de exportacíon: frutas tropicales y hortalizas.* Eschborn, Alemanha. 1992.

[14] AMORIM, T. B. F. *Colheita e pós-colheita: manejo e conservação da manga*. Em: SÃO JOSÉ, A. R. (Org.). *O agronegócio manga*: *produção e mercado*. UESB/DFZ . Vitória da Conquista, BA, 2002.

[15] ICONTEC *Internacional Normas Y Especificaciones, Colombian Technical Standard NTC – 5139, Frutas frescas, Mangos criollos. Especificacion*es, INCONTEC, Bogotá D.C. Colombia. 2002.

[16] S. RUSSELL AND P. NORVIG, *Inteligência Artificial*. Ed. Campus/ Elsevier, Rio de Janeiro, 2004.

[17] PEDRINI, H., SCHWARTZ, W.R.. Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações. Thompson Learning Eds., São Paulo, 2008

[18] MEDINA, V. M..: Fisiologia Pós-Colheita da Manga; Embrapa

CNPMF, Cruz das armas, BA – 1995.

[19] NETTO, Á. G.[ET. AL.]: Mangas para Exportação: Procedimentos de Colheita e Pós-Colheita; Publicação técnica FRUPEX; Embrapa SPI, Brasília-DF, 1994.

Joelson Nogueira de Carvalho é graduado como Bacharel em Ciências da Computação pela Universidade Federal da Paraíba (UFPB), Campina Grande, PB, em 1989, possui mestrado em Informática obtido em 1999, também na UFPB. É professor da UFPB, lotado no Campus IV em Rio Tinto-PB. Atualmente é doutorando em Engenharia de Processos no Centro de Ciências e Tecnologia da Universidade Federal de Campina Grande (UFCG). Áreas de pesquisa: Inteligência Artificial, Análise de Imagens, Sistemas Digitais, Mecatrônica, Lógica e Teoria da Computação.

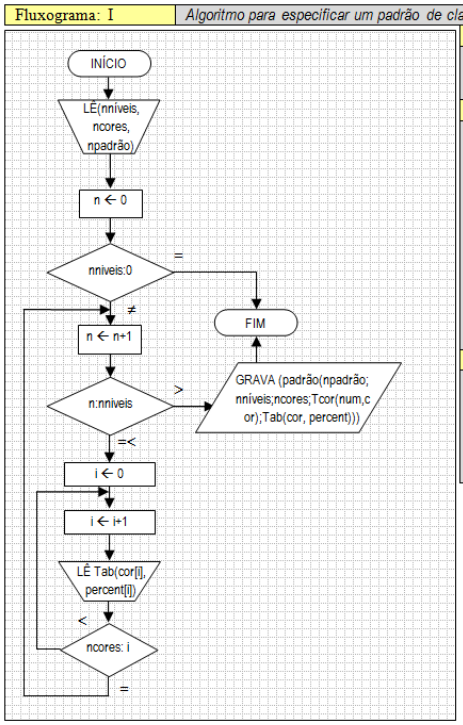
Edmar Candeia Gurjão é graduado em Engenharia Elétrica pela Universidade Federal da Paraíba (1996), mestrado em Engenharia Elétrica pela Universidade Federal da Paraíba (1999) e doutorado em Engenharia Elétrica pela Universidade Federal de Campina Grande (2003). Atualmente é professor da Universidade Federal de Campina Grande. Tem experiência na área de Engenharia Elétrica, com ênfase em Amostragem Compressiva (Compressed Sensing), Rádio Definido por Software e Aplicações da Teoria da Informação.

Mário Eduardo Rangel Moreira Cavalcanti-Mata Possui graduação em Engenharia Agronômica pela Universidade Federal de Viçosa (1975), Mestrado em Engenharia Agrícola pela Universidade Federal de Viçosa (1979) e Doutorado em Engenharia de Alimentos pela Universidade Estadual de Campinas (1997). Atualmente é Professor Associado IV da Universidade Federal de Campina Grande. Tem experiência na área de Engenharia Agrícola e Engenharia de Alimentos. Atua como membro efetivo dos programas de Pós-Graduação (Mestrado e Doutorado) em Engenharia Agrícola e Engenharia de Processos, com ênfase nos temas: Armazenamento de produtos agrícolas, Secagem de produtos agroindustriais, Crioconservação de sementes, Processamento de grãos e alimentos, Congelamento de alimentos, Armazenagem refrigerada, Automação de sistemas, Propriedades físicas dos materiais biológicos, Desenvolvimento de softwares aplicados e Processamento de imagens para sistemas operatrizes.

Maria Elita Martins Duarte Possui graduação em Engenharia Agrícola pela Universidade Federal da Paraíba (1988), mestrado em Engenharia Agrícola pela Universidade Federal da Paraíba (1991) e doutorado em Engenharia de Alimentos pela Universidade Estadual de Campinas-UNICAMP (1997). É Professora Associada IV da Universidade Federal de Campina Grande, lotada no Departamento de Engenharia Agrícola. Pesquisadora pertencente ao quadro do Programa de Doutorado em Engenharia de Processos e do Programa de Pós Graduação em Engenharia Agrícola, Co-Editora da Revista Brasileira de Produtos Agroindustriais e Membro da diretoria da Sociedade Brasileira de Corantes Naturais. Possui experiência em Armazenamento e Processamento de Produtos Agrícolas, atuando principalmente nos seguintes temas: propriedades físicas, propriedades mecânicas, óticas, elétricas, transporte, hidrodinâmica, aerodinâmica, secagem, liofilização, concentração, armazenagem, extração, crioconservação, congelamento e beneficiamento de sementes.

ANEXO A

Fluxograma: I — *Algoritmo para especificar um padrão de classificação em uma estrutura de dados*



Funções

LÊ(Leitura de dados)
Grava( ): Salva (estrutura) em disco

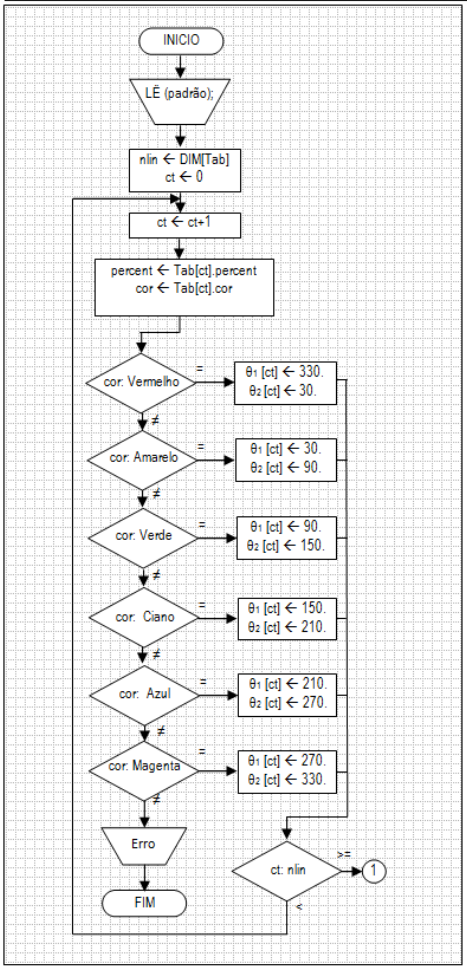
Variáveis

padrão:	Estrutura de dados
ncores:	Número de cores do padrão
nníveis :	Número de níveis do padrão
npadrão:	Nome do padrão
n, i:	Variáveis auxiliares
cor:	Componente de cor
percent:	Percentual da componente de cor associado ao nível
Tcor:	Tabela de cores com dois campos: num – número da cor e cor – nome da cor.
Tab:	Tabela do percentual de cobertura com dois campos: cor – nome da cor e percent.- % de cobertura

Funcionamento

Inicialmente, são lidos os números de níveis, de cores utilizadas e um nome para o padrão. Um contador é iniciado e um loop controlado por esse contador lê os valores das cores e respectivos percentuais para cada nível que o padrão estabelece. Quando o contador atingir um valor superior ao número de níveis do padrão, o algoritmo para.

Fluxograma: IIa — *Algoritmo para associar o intervalo do valor H (cor / matiz) ao padrão estabelecido.*



Funções

LE():	Entrada de dados
DIM[ ]:	Retorna o número de linhas de uma tabela

Variáveis

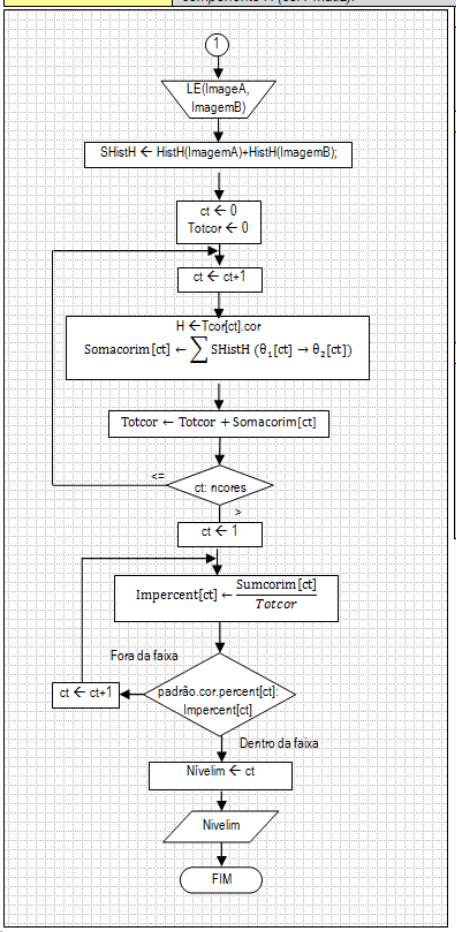
padrão:	Estrutura de dados
nníveis :	Número de níveis do padrão
npadrão:	Nome do padrão
cor:	Componente de cor
percent:	Percentual da componente de cor associado ao nível
Tcor:	Tabela de cores com dois campos: num – número da cor e cor – nome da cor.
Tab:	Tabela do percentual de cobertura com dois campos: cor – nome da cor e percent.- % de cobertura
nlin:	Número de linhas da tabela Tab
ct:	Variável auxiliar
θ_1:	Ângulo inicial da cor no padrão HSL
θ_1:	Ângulo final da cor no padrão HSL

Funcionamento

Nesta primeira etapa, o algoritmo inicia com a leitura do padrão; um loop é gerado e para cada linha da tabela de cores do padrão, onde cada cor e seu respectivo percentual são lidos e o valor dos ângulos iniciais e finais de cada cor definida no padrão é obtido.

ANEXO B

Fluxograma: IIb — *Algoritmo para identificar o Nível de uma fruta em um padrão especificado através da componente H (cor / matiz).*



Funções

LE():	Entrada de dados
HistH():	Retorna o Histograma H (hue) – cor
SHistH():	Soma dos Histogramas H (hue) das imagens das frutas.

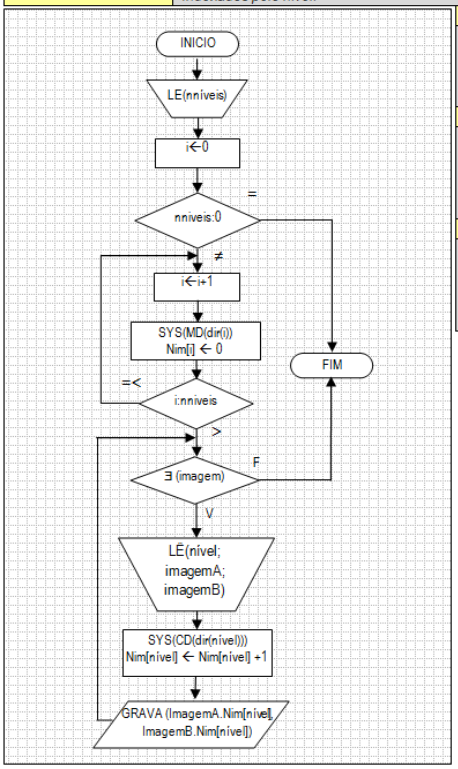
Variáveis

ImagemA:	Arquivo de imagem .PNG – Lado A da fruta
ImagemB:	Arquivo de imagem .PNG – Lado B da fruta
ct:	Variável auxiliar
Totcor	Valor total da soma das componente das cores H: cor/matiz lida do padrão
Somacorim:	Somatório de uma componente H (cor) da Imagem
θ_1:	Ângulo inicial da cor no padrão HSL
θ_1:	Ângulo final da cor no padrão HSL
Impercent:	Percentual da cor na imagem
Nivelim:	Nível da fruta (imagem) segundo o padrão
faixa:	Valor do percentual de cores do padrão

Funcionamento

No início, duas imagens - uma imagem de cada lados da fruta são lidas. Um loop é feito sobre as cores do padrão; para cada cor, realiza-se a soma das componentes da mesma nas imagens da fruta, esse valor também é somado ao total das componentes de toas as cores (Totcor). No final, calcula-se o percentual de cada cor presente na fruta (Impercent) e logo em seguida, compara-se esses percentuais às faixas de cobertura das cores do padrão, identificando o nível onde o aspecto das imagens se encaixa no padrão.

Fluxograma: III — *Algoritmo para armazenar imagens dos frutos do padrão em diferentes diretórios, indexados pelo nível.*



Funções

LE():	Entrada de dados
SYS	Ref. à funções do Sistema Operacional
MD	Cria um Diretório
dir:	Referencia um diretório

Variáveis

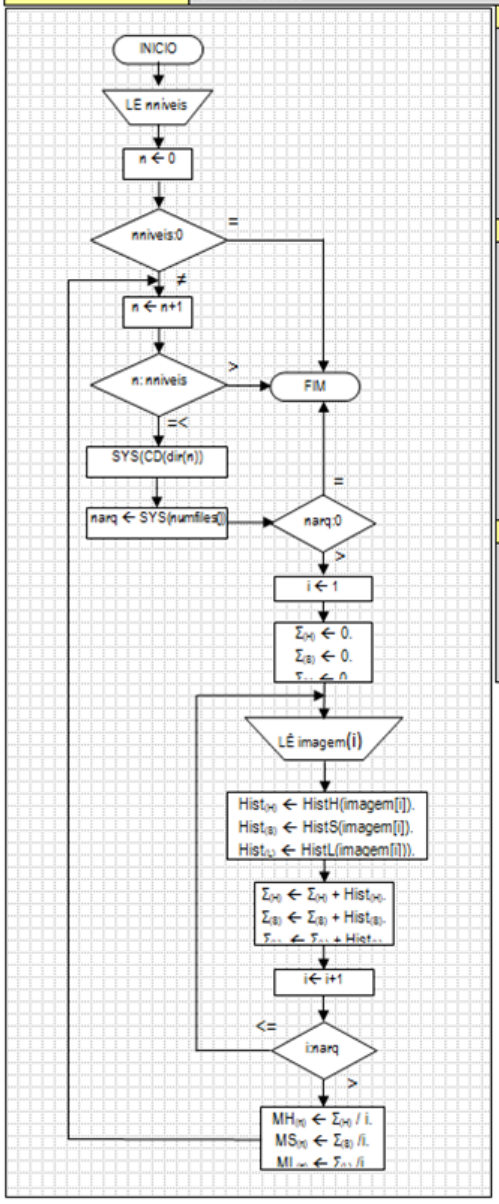
i,nível:	Variável auxiliar
Nim[]:	Número de imagens por nível
níveis:	Número de níveis do padrão
ImagemA:	Arquivo de imagem .PNG – Lado A da fruta
ImagemB:	Arquivo de imagem .PNG – Lado B da fruta

Funcionamento

No início, lê-se o número de níveis que o padrão admite; com essa informação, um loop é iniciado para a geração de um diretório para cada nível. após a geração de diretórios, um novo loop é gerado para ler as imagens dos lados das frutas e associá-las aos níveis (diretórios gerados).

ANEXO C

Fluxograma: IVa — *Cálculo dos Histogramas médios para H, S e L de n conjuntos de imagens, distribuídas em n diretórios.*



Funções

SYS	Ref. a funções do Sistema Operacional
CD	Muda o Diretório atual
dir:	Referencia um diretório
numfiles()	Retorna o número de arquivos de um diretório
HistH():	Retorna o Histograma da Cor/Matiz de uma imagem
HistS():	Retorna o Histograma da Saturação de uma imagem
HistL():	Retorna o Histograma da Luminosidade de uma imagem

Variáveis

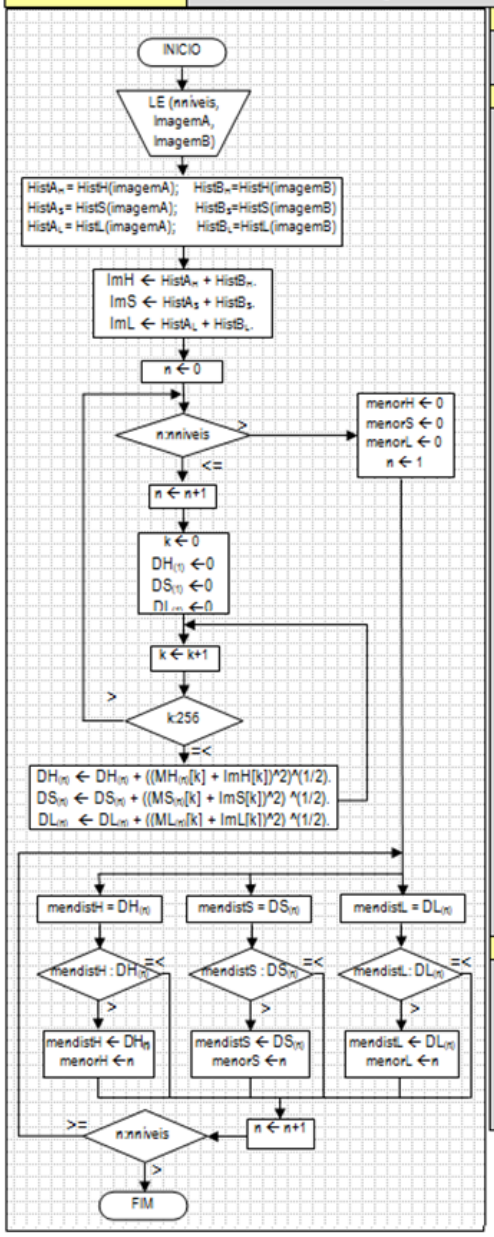
imagem:	Arquivo de imagem
narq:	Número de arquivos de imagens num diretório
$Hist_{(H)}$:	Histograma da imagem para a cor/ Matiz
$Hist_{(S)}$ =	Histograma da imagem para Saturação
$Hist_{(L)}$ =	Histograma da imagem para Luminosidade
$\Sigma_{(H)}$:	Soma dos Histogramas para cor/matiz
$\Sigma_{(S)}$:	Soma dos Histogramas para Saturação
$\Sigma_{(L)}$:	Soma dos Histogramas para Luminosidade
$MH_{(n)}$:	Valor médio dos Histogramas para Cor / Matiz
$MS_{(n)}$:	Valor médio dos Histogramas para Saturação
$ML_{(n)}$:	Valor médio dos Histogramas para Luminosidade

Funcionamento

Após ler o número de níveis do padrão, que é igual ao número de diretórios de imagens a serem lidas, um loop é iniciado para a leitura das imagens de cada um dos diretórios. Para cada diretório, suas imagens são lidas e seus histogramas HSL são calculados; a soma dos valores dos componentes HSL é realizada e ao final, as médias dos valores dessas componentes HSL são calculadas.

ANEXO D

Fluxograma: IVb — *Verifica a pertinência de uma imagem a um nível de um padrão estabelecido, considerando os níveis do padrão onde ocorrem as menores distâncias para as componentes HSL da imagem de uma fruta, pelo Erro Quadrático Médio.*



Funções

SYS	Ref. a funções do Sistema Operacional
CD	Muda para um Diretório
dir.	Referencia um diretório

Variáveis

nniveis:	Número de níveis
ImagemA:	Arquivo de imagem lado A (da fruta)
ImagemB:	Arquivo de imagem lado B (da fruta)
$HistA_H$:	Histograma do matiz do lado A.
$HistA_S$:	Histograma da Saturação - lado A.
$HistA_L$:	Histograma da Luminosidade - lado A
$HistB_H$:	Histograma do matiz do lado B
$HistB_S$:	Histograma da Saturação do lado B
$HistB_L$:	Histograma da Luminosidade do lado B
ImH:	Soma dos histogramas do matiz dos lados A e B.
ImS:	Soma dos histogramas da Saturação dos lados A e B.
ImL:	Soma dos histogramas da Luminosidade dos lados A e B.
n:	Variável auxiliar
k:	Variável auxiliar
menorH:	Nível de menor distância para Matiz
menorS:	Nível de menor distância para Saturação
menorL:	Nível de menor distância para Luminosidade
mendistH:	menor distância para o Matiz
mendistS:	menor distância para a Saturação
mendistL:	menor distância para a Luminosidade
menorH:	diretório (ou nível) onde ocorreu a menor distância para o Matiz
menorS:	diretório (ou nível) onde ocorreu a menor distância para a Saturação
menorL:	diretório (ou nível) onde ocorreu a menor distância para a Luminosidade
$DH_{(n)}$:	Distância da componente de cor (hue) entre a imagem e o nível n do modelo.
$DS_{(n)}$:	Distância da componente Saturação entre a imagem e o nível n do modelo.
$DL_{(n)}$:	Distância da componente Luminosidade entre a imagem e o nível n do modelo.
$MH_{(n)}[k]$	k-ésimo valor da componente H do histograma médio do nível n
ImH[k]	k-ésimo valor da componente H do histograma da imagem
$MS_{(n)}[k]$	k-ésimo valor da componente S do histograma médio do nível n
ImS[k]	k-ésimo valor da componente S do histograma da imagem
$ML_{(n)}[k]$	k-ésimo valor da componente L do histograma médio do nível n
ImL[k]	k-ésimo valor da componente L do histograma da imagem

Funcionamento

Lê as imagens dos dois lados da fruta e o número de níveis de um padrão; obtém a soma das componentes HSL das imagens e em seguida, executa uma varredura dentro de um loop acessando os diretórios (níveis do padrão), lendo suas respectivas imagens; para cada nível é calculada a distância entre os histogramas da imagem e os valores médios dos histogramas HSL das imagens desse nível, pelo método do Erro Quadrático Médio. No final, um loop é feito para verificar as menores diferenças entre os valores HSL das imagens e das

Adaptive Product Classification for Online Marketplace Based on Semantic Analysis

T. F. V. Medeiros and F. G. Cozman

***Abstract*— Each e-Commerce website describes and classifies their products according to what works best for themselves. But anyone willing to merge the catalog of several digital stores must face the problem of ontology matching and reclassification. In this context, this paper proposes an adaptive automatic classification system which relies on the textual description of products to find the correct category it belongs to and after human revision of possible errors, the system improves to better classify the next batch of new products. This is a work in progress, no results are shown yet.**

***Keywords*— E-Commerce, Product Classification, Machine Learning, Semantic Analysis, Adaptive Technology.**

I. INTRODUCTION

INTERNET is now part of our daily life in every aspect, social and economic, cultural and entertainment. Particularly for online retailers, dealing through the World Wide Web offers various advantages over brick and mortar stores, such as the virtually infinite shelf space. But this advantage causes a burden for the customer who has difficulty finding the wanted product among such a vast catalog.

To solve this problem three approaches are common:

- Search engines
- Catalog ontologies
- Recommender Systems

The second approach consists in building a tree of labels which applies to every product. This way, the user is able to navigate through the site narrowing the range of products at each branch chosen on the tree arrangement.

This is a great solution until two or more retailers must communicate, because their ontologies don't match perfectly even if they offer exactly the same products.[8]

Two common situations in which the merge of product catalogs is necessary are E-Marketplace, where many small vendors share the same digital roof; and Comparison Shopping sites which collect the prices of the same product in many stores and put them side by side to help the customers choose the best option.

In both of those business models, they need a unified product classification. And to make the problem harder, it is not static, new products come every day.

When the problem is not focused in one product domain, like movies or clothes or refrigerators, there is no common structured attributes except the product name and textual description, price and original seller or manufacturer.

We propose a system that extracts semantic features from the short textual description of the products and classifies each one into the appropriate category. After the automatic classification, the uncertain items are revised by humans that guarantee every product is found in this right place.

For academic purposes, the system is initially designed to use a previously labeled dataset which is broken in even batches of products, simulating the situation of successive increments in the product catalog. The system must be able to learn and be more accurate in each new batch.

In this first attempt, only the top level of the ontology is considered, leaving the problem of taking into account hierarchical to future works.

The reminder of this paper is divided in a **Brief Outline** of the system workflow, proposed **Application** and **Conclusion**.

II. BRIEF OUTLINE

The functioning of the system is as follows:

1. At first, an initial set of labeled products must be presented to train the classifier so that it shows a tolerable classification accuracy.
2. Thereafter for each new block of product that arrive and must be labeled into the fixed set of categories, this loop ensues:
   a. The current classification rules are applied.
   b. Experts validate the automatic classification, relabeling some products if necessary
   c. The adaptive layer examines the errors made and change the classification rules to avoid them.

Figure 1 illustrates this workflow.

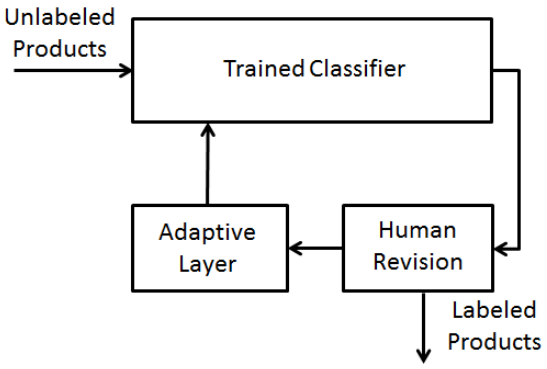


Figure 1 - System Workflow

T. F V. Medeiros, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, tacio.medeiros@usp.br
F. G. Cozman, Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, fgcozman@usp.br

The classifier box is divided in 4 steps as shown in Figure 2. Each step can be made with more than one technique, and most of them require the setting of parameters. This makes the complete classifier very hard to calibrate for best performance using the provided dataset of products.

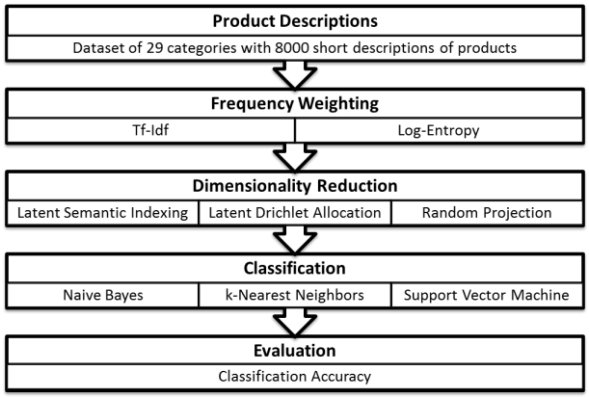


Figure 2 - Classifier steps

III. APPLICATION

The system is being implemented in Python programming language using the topic modelling library Gensim[9] for the Natural Language Processing part and using Orange[10] for the Machine Learning part. The adaptive layer will control the use of both libraries adjusting parameters.

Dataset

The original dataset is a dump of all products

A balanced subset was sampled from the original dataset for this first attempt of a classification system.

The resulting dataset is composed of 29 top level categories, each containing 8000 product descriptions. Adding up to a total of 232000 items.

The descriptions are written in Portuguese, and have been stripped of accented characters and standardized to lowercase. Those descriptions were made by different retailers and do not follow any particular pattern, often not even proper grammar.

Pre-processing

Each product description is broken in an unordered list of words. Furthermore, each pair of consecutive words is also recorded and appended to the list. So each product is now represented by its list of unigram and bigram words.

Before jumping to converting every distinct word to a dimension, some tasks are performed in order to group or disregard some words in order to reduce at firsthand the size of the dictionary.

Entity recognition

Many documents in this dataset contain a product code within the text. Each code is unique and only have meaning for the retailer, but it is recognizable as a code and not all categories have them, so all words code-like are replaced for the placeholder _CODE_ which adds some information for the semantic processing to follow. It is the only Entity Recognition performed in this version of the system.

Frequency cutoff

Words that appear only once or twice in the whole dataset or too often in different documents do not help and can be discarded. The lower and upper threshold are parameters which have to be calibrated.

Stemming

Stemming consists in removing suffixes and prefix and conjugation from words so that only the radical remains, this way a further reduction of the dictionary dimension is achieved.

Frequency Weighting

At this step, each document (product description) is stored as a vector of words. The dimension of this vector is the size of the vocabulary of this dataset.

Now, instead of a simple count of words for each document, some statistical transformations can be performed to enhance the discriminative power of those values. This process is called Frequency Weighting[7].

Among the possible techniques, the most common are Tf-Idf and Log-Entropy.

Dimensionality Reduction

For the next step, of classification, the dimension of the resulting Term-Document Matrix is still too high. So more dimension reduction must be performed.

The most common method is Latent Semantic Indexing (LSI), which is similar in nature to the even more commonly used in any Machine Learning application Singular Value Decomposition (SVD).

Other dimension reduction techniques are Latent Drichlet Allocation (LDA) and Random Projection (RP).

This step is also responsible to group words which are synonyms and distinguish polysemy.

Classification

The initial collection of texts is now in the form of a matrix where the number of rows is the number of products and the number of columns was arbitrarily chosen in the Dimensionality Reduction step.

We have chosen the following classifiers for this experiment which are readily available in the Orange framework:

- Naïve Bayes
- k-Nearest Neighbors (KNN)

- Support Vector Machine (SVM)

Evaluation

The fundamental metric for evaluating this system is the classification accuracy.

On a related work, GoldenBullet[1], it is also used the accuracy of the 10 best class candidates. Since there is a human validation after the automatic classification, the system is tolerant, at least in the beginning, to make a small mistake. If the correct class is among the first 10 predicted ones, it is considered a success, raising considerably the accuracy.

IV. Follow Up

The next steps in this project are to implement the adaptive layer and then making the classification hierarchical, descending in the product categories tree.

This learning program will be modeled as a Cellular Learning Automata as described in [5].

V. Acknowledgement

The first author was funded by the Buscapé Company.

The second author acknowledges support by the Buscapé Company and CNPq.

References

[1] Ding, Y.; Korotkiy, M.; Omelayenko, B.; Kartseva, V.; Zykov, V.; Klein, M.; Schulten, E. & Fensel, D. GoldenBullet: Automated Classification of Product Data. in *E-commerce Business Information Systems*, 2002

[2] Mattos, A.; Kampen, M.; Carriço, C.; Dias, A. & Crivellaro, A. Caseli, H.; Villavicencio, A.; Teixeira, A. & Perdigão, F. (Eds.). E-commerce Market Analysis from a Graph-Based Product Classifier. *Computational Processing of the Portuguese Language*, Springer Berlin Heidelberg, 2012, 7243, 291-297

[3] Beneventano, D. & Magnani, S. A Framework for the Classification and the Reclassification of Electronic Catalogs. *Proceedings of the 2004 ACM Symposium on Applied Computing*, ACM, 2004, 784-788

[4] Jackson, Q. Z. & Landgrebe, J. D. Design Of An Adaptive Classification Procedure For The Analysis Of High-dimensional Data With Limited Training Samples. 2001

[5] Esmaeilpour, M.; Naderifar, V. & Shukur, Z. Cellular Learning Automata Approach For Data Classification. *International Journal Of Innovative Computing Information And Control*, ICIC. INTERNATIONAL TOKAI UNIV, 9-1-1, TOROKU, KUMAMOTO, 862-8652, JAPAN, 2012, 8, 8063-8076

[6] Cereda, P. R. M. & Neto, J. J.. Mineração Adaptativa de Dados: Aplicação à Identificação de Indivíduos. *Workshop de Tecnologia Adaptativa*, 2012

[7] Solka, J. L.. Text Data Mining: Theory and Methods. *Statistics Surveys*, 2008, *2*, 94-112

[8] Ding, Y.; Fensel, D.; Klein, M.; Omelayenko, B. & Schulten, E.. The role of ontologies in ecommerce. *Handbook on ontologies, Springer*, 2004, 593-615

[9] Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA.

[10] Demšar, J.; Zupan, B.; Leban, G. & Curk, T. Boulicaut, J.-F.; Esposito, F.; Giannotti, F. & Pedreschi, D. (Eds.) Orange: From Experimental Machine Learning to Interactive Data Mining. *Knowledge Discovery in Databases PKDD 2004*, Springer, 2004, 3202, 537-539

Oportunidades de uso da Adaptatividade em um SPLN para criação de atividades de leitura

J. L. Moreira Filho and Z. M. Zapparoli

***Resumo* — Este artigo apresenta oportunidades de utilização de adaptatividade em um sistema de processamento de língua natural (SPLN) para criação automática de atividades de leitura em textos com *corpora*. O trabalho consiste na construção de um sistema capaz de ler um texto em língua inglesa e automaticamente criar exercícios em forma de questões de múltipla escolha relacionadas ao texto de entrada, em um ambiente de virtual de aprendizagem próprio desenvolvido para tal fim. O principal foco do trabalho está relacionado à identificação, categorização e extração de itens lexicais para compor os exercícios das atividades. Primeiro, apresentamos brevemente alguns estudos prévios e protótipos de *software*. Em segundo, descrevemos as principais funcionalidades do ambiente virtual em que o SPLN. Em terceiro, discutimos as oportunidades de uso da adaptatividade para o sistema proposto. Por último, apresentamos as considerações finais.**

***Palavras-chave* — Linguística de Corpus (*Corpus Linguistics)*, Tecnologia Adaptativa *(Adaptative Technology)*, Processamento de Língua Natural *(Natural Language Processing)*.**

I. INTRODUÇÃO

Este trabalho apresenta alguns avanços obtidos no trabalho de pesquisa na criação de um SPLN para criação de atividades de leitura em língua inglesa com *corpora*.

Há uma gama de pesquisas sobre a análise, o desenvolvimento e o emprego de materiais de ensino baseados em *corpora*, uma vez que privilegiam a língua em uso[1].

Embora seja positivo para o processo de ensino-aprendizagem, o uso de materiais baseados em *corpora* ainda não é uma realidade comum 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* (LC) para que possa aproveitar os benefícios do uso de *corpus* 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 Processamento de Língua Natural (PLN), de práticas de análise de *corpus* para o ensino de línguas e do conceito de Adaptatividade e Tecnologia Adaptativa [2], a pesquisa em andamento tem o objetivo de suprir a necessidade de professores de língua estrangeira que desejam utilizar materiais baseados em *corpora*, 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 [3], que teve como produto final um *software desktop* para preparação semiautomática de atividades de leitura em inglês.

Nos primeiros protótipos, com base no conceito de "atividade padrão" para o ensino de leitura de *English For Specific Purposes* (*ESP*), inglês para fins específicos, um conjunto fixo de exercícios é preparado automaticamente, incluindo atividades baseadas em concordâncias, *data-driven learning*[2], 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 que a ferramenta possa ser usada pelo usuário final.

Assim, estudam-se propostas de utilização de adaptatividade no sistema em desenvolvimento.

II. ESTUDO INICIAL E PROTÓTIPOS

1. Primeiro Protótipo – Modelo Fixo

O *software desktop* desenvolvido na pesquisa de mestrado [3], para o sistema operacional Windows, na linguagem de programação Visual Basic 6, disponível no sítio http://www.xcorpus.net/downloads/rcb.zip, permite a criação de atividades a partir de modelos estáticos e de um assistente (*wizard*) que auxilia o usuário até a preparação das atividades.

[1] Textos reais, não inventados com o objetivo de ensinar língua.

[2] Proposta que enfatiza o ensino do léxico da língua por meio de descobertas, tornando o aluno um pesquisador.

O assistente eletrônico guia o usuário através de três etapas. Na primeira etapa, o usuário escolhe que tipo de atividade de leitura será preparado. O programa fornece alguns modelos padronizados. Na segunda etapa, o usuário seleciona um texto de sua escolha para preparação da atividade. Na terceira etapa, o programa exibe informações sobre o texto escolhido e a atividade quase pronta. Após verificar as informações do texto e editar a atividade (opcional), o usuário pode salvá-la.

Os exercícios das atividades estão relacionados a concordâncias (*data-driven*), predição, léxico-gramática e questões para leitura crítica. Um modelo fixo com questões e lacunas para inserção de itens lexicais do próprio texto são utilizadas para formar os exercícios na atividade.

Para tanto, a partir do texto selecionado, o programa realiza várias análises automáticas por meio de fórmulas estatísticas: lista de frequência, palavras-chave, possíveis palavras cognatas, etiquetagem morfológico, possíveis padrões (*n-gramas*) e densidade lexical do texto. O esquema a seguir sintetiza o funcionamento do sistema.

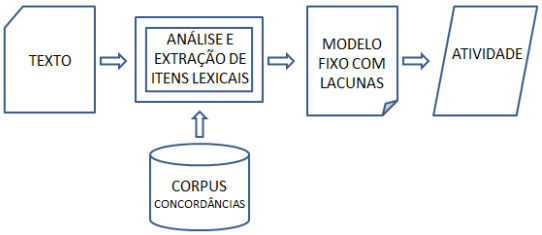


Figura 01 – Esquema de funcionamento do primeiro protótipo

2. Segundo Protótipo – Modelo Personalizável

Outra versão, tendo em vista os problemas de compatibilidade, foi escrita em C#, disponível no sítio: http://www.xcorpus.net/downloads/rcb_01-08-2010.zip, com o objetivo de manter o registro do trabalho realizado. O programa incorporou algumas mudanças em relação aos modelos utilizados para a criação das atividades e ao número de etapas.

O número de etapas do assistente foi reduzido a apenas três: Etapa 1 – escolha do modelo de atividade; Etapa 2 – escolha do texto e do *corpus* de referência, e extração de exemplos; Etapa 3 – visualização de informações do texto e atividade preparada para edição e publicação.

Os modelos podem ser criados pelo próprio usuário por meio da utilização de um conjunto finito de códigos, tal como <XC010>, que retorna dez possíveis palavras cognatas do texto, ou <XK0#>, que retorna um determinado número depor meio palavras-chave do texto (substituir # pelo número desejado). Na ajuda do programa, há uma listagem de todos os códigos disponíveis e sua descrição.

Nesse primeiro protótipo, com base no conceito de atividade padrão, introduzido em para cursos de ESP, embora os resultados obtidos tenham demonstrado a viabilidade e o potencial da solução, há ainda a necessidade de muita pesquisa e desenvolvimento para melhorias: diminuição de erros de análise, aumento da variedade de exercícios disponíveis e adequação do conjunto de exercícios ao texto de entrada.

Com base na experiência de desenvolvimento dessas duas versões, pretende-se explorar as possibilidades de montagem de atividades de forma automatizada, por meio de adaptatividade.

III. AMBIENTE VIRTUAL DE APRENDIZAGEM

O sistema de criação automática de atividade de leitura estará atrelado ao um ambiente virtual de aprendizagem, inicialmente chamado de '*Readler*', implementado em PHP (interface) e Python (processamento de língua natural). A figura abaixo ilustra a interface do ambiente *online*:

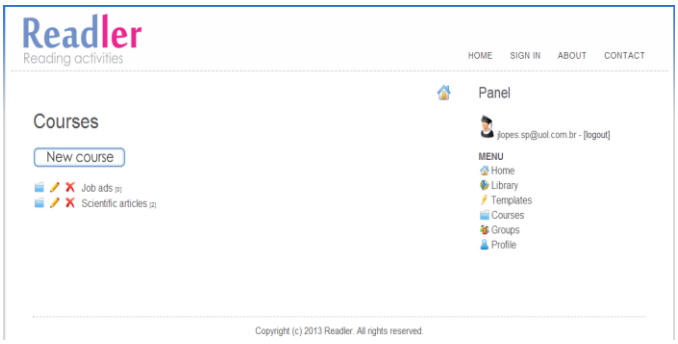


Figura 02 – Interface do ambiente virtual

O ambiente prevê a utilização de dois tipos de usuário: a. professor; b. aluno. Na interface, ao professor, são disponibilizadas ferramentas para a criação de cursos, atividades e grupos de alunos para os cursos criados. Ao aluno, são disponibilizados os cursos e suas respectivas atividades.

Para o professor, a utilização do ambiente consiste na manipulação dos seguintes principais itens do menu: *Library*, *Templates*, *Courses* e *Groups*.

A. *Library*

Seção em que é possível armazenar textos com o objetivo de servirem para a criação de atividades de leitura. É importante destacar que, antes de iniciar a criação de uma atividade, é necessário escolher um texto. O texto adicionado é analisado e armazenado em um banco de dados em formato XML para estar disponível para a criação de atividades em qualquer curso.

```
<sentence id="7">
        <token cog="y" freq="1" group="nI" id="155" key="7.0"
loc="77.9411764706" pos="NNS" pref="" suf="" syllables="4">Requirements</token>
        <token cog="n" freq="2" group="O" id="156" key="3.0"
loc="78.431372549" pos=":" pref="" suf="" syllables="0">:</token>
        <token cog="n" freq="6" group="O" id="157" key="79.0"
loc="78.9215686275" pos=":" pref="" suf="" syllables="0">-</token>
        <token cog="n" freq="1" group="nI" id="158" key="16.0"
loc="79.4117647059" pos="NNS" pref="" suf="" syllables="3">Bachelors</token>
        <token cog="n" freq="1" group="nI" id="159" key="6.0"
loc="79.9019607843" pos="NN" pref="" suf="" syllables="2">degree</token>
        <token cog="n" freq="1" group="nI" id="160" key="0.0"
loc="80.3921568627" pos="CC" pref="" suf="" syllables="1">or</token>
        <token cog="y" freq="1" group="nI" id="161" key="8.0"
loc="80.8823529412" pos="NN" pref="" suf="+ent" syllables="4">equivalent</token>
        <token cog="n" freq="11" group="O" id="162" key="1.0"
loc="81.3725490196" pos="SENT" pref="" suf="" syllables="0">.</token>
    </sentence>
```

Figura 03 – Exemplo de sentença de texto analisado no formato XML

As análises automáticas realizadas incluem: itemização, etiquetagem morfológica, contagem de frequência das palavras, extração de palavras-chave, identificação de palavras cognatas, identificação de grupos nominais, identificação de referência pronominal, identificação de afixos e marcação da posição de cada palavra no texto. Na mesma seção, há a possibilidade de o usuário visualizar as informações do texto em uma interface amigável.

B. Template

Seção em que o usuário tem a possibilidade de criar códigos em XML para a criação automática de atividades personalizadas, já que um conjunto padrão estará sempre disponível. Tal funcionalidade é considerada de nível avançado. Apresentamos abaixo um exemplo de *template* para criar uma questão de múltipla escolha com palavras cognatas.

```
<activity>
    <multiple_choice>
    <stem><label>What are the cognate words from the text?</label></stem>
    <opt_A>
    <text>a)</text>
    <cognates limit='5' delimiter=' ' sort ='text' order='asc' case='lower'></cognates>
    </opt_A>
    <opt_B><wordlist limit='5' delimiter=' ' sort ='text' order='asc' case='lower'></label></opt_B>
    <opt_C><wordlist limit='5:10' delimiter=' ' sort ='text' order='asc' case='lower'></opt_C>
    <opt_D><wordlist limit='10:15' delimiter=' ' sort ='text' order='asc' case='lower'></opt_D>
    <answer>A</answer>
    </multiple_choice>
</activity>
```

Figura 04 – Exemplo de código XML em um *template*

C. Courses

Seção para a criação de cursos. Os cursos são compostos por atividades. Cada atividade deve estar relacionada a um texto já adicionado à seção '*Library*'. Os itens que compõe as atividades são, a princípio, questões de múltipla escolha que podem ser criadas manualmente pelo usuário ou automaticamente por meio de '*templates*', já disponibilizados no ambiente ou personalizados.

D. Groups

Seção para o gerenciamento de alunos nos cursos criados. É possível organizar turmas e disponibilizar o conteúdo dos cursos, além de acompanhar o desempenho de cursistas.

Os passos básicos para a utilização do ambiente, na perspectiva do professor são: a. adição de textos à seção *Library*; b. criação de um curso na seção *Courses*; c. seleção de um curso para criação de uma atividade; d. criação de itens (manual ou automaticamente por meio de *templates*) para a atividade criada.

O ambiente *online* está em constante desenvolvimento, com adição de recursos de análises de texto e criação automática de atividades. Uma das intenções (posteriores) em relação ao sistema a ser implementado é a possibilidade de disponibilizar alguns recursos utilizados pelo professor para a interface do aluno, permitindo uma nova possibilidade de aprendizado conhecida como *self access language learning*, na qual o aluno teria ao seu dispor os recursos para escolha de textos e criação de suas próprias atividades.

Contudo, tanto em relação à utilização primária do ambiente como para uma possível estrapolação de seus objetivos, é necessário que as funções de criação automática de atividades sejam efetivas, de modo a retornar os melhores resultados possíveis sem erros ou inadequações.

IV. OPORTUNIDADES DE USO DE ADAPTATIVIDADE NO SPLN

O SPLN, desenvolvido em Python, para o ambiente virtual é composto basicamente de um módulo de análise linguística, a partir de técnicas e métodos de análises utilizados em processamento de língua natural e pesquisas em Linguística de *Corpus*, e um módulo com funções parametrizadas que faz a leitura e tradução de um texto analisado e códigos de *templates* em XML para criação de questões de múltipla escolha.

As principais análises linguísticas realizadas pelo módulo são: itemização, etiquetagem morfológica, geração de listas de frequência, extração de palavras-chave, identificação de palavras cognatas, identificação de grupos nominais e identificação de afixos.

Tendo em vista a natureza das análises e os objetivos explicitados em relação à criação de atividades, é imprescindível que erros e imprecisões sejam eliminados.

Muitas das análises e heurísticas utilizadas no módulo, principalmente as derivadas de metodologias da instrumentação de análise de *corpus*, são baseadas unicamente em estatísticas. Por exemplo, para a extração de palavras-chave, há a comparação das frequências das palavras no texto com uma lista de frequência de um banco de dados de quase 100.000.000 de palavras por meio de uma fórmula (*log likelihood*), com o retorno de uma lista de palavras ordenadas por chavicidade. Os resultados geralmente precisam ser filtrados a fim de se obter uma listagem sem ruídos.

O exemplo ilustra um problema comum com processos estocásticos, embora sua utilização seja ampla no processamento de língua natural. A obtenção de dados exatos por sua natureza não determinística torna-se insuficiente. Da mesma maneira, métodos unicamente gramaticais e simbólicos podem não bastar a determinados tipos de problema.

É o caso do método utilizado na identificação de grupos nominais por meio de uma gramática, no módulo de análise do SPLN. O método de [4] utiliza três etiquetas básicas (*inside (nI)*, *outside (O)* e *between(nB)*) e leva em consideração a etiqueta morfológica dos itens. Uma primeira etiquetagem é feita e uma gramática faz o refinamento da etiquetagem por meio de regras contextuais. Um autômato simples faz a leitura e identificação dos grupos nominais, como mostra a figura abaixo:

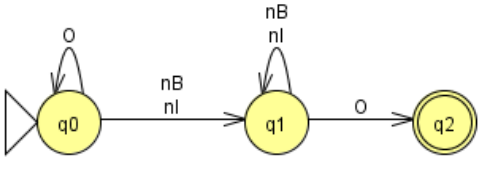


Figura 05 - Exemplo de autômato para leitura de grupos nominais

Para a obtenção de melhores resultados, a combinação de métodos gramaticais e autômatos com processos estocásticos é desejável. O uso de soluções híbridas pode ser a chave para resolução de problemas quando uma única tecnologia empregada não dá conta de toda a complexidade. No caso do processamento de língua natural, em que o problema de ambiguidades é recorrente, é justificável a busca de conciliação de métodos aparentemente antagônicos.

O elo para a conciliação desses métodos pode ser o uso da adaptatividade. A aplicação da adaptatividade pode permitir pontos de equilíbrio entre métodos simbólicos e estocásticos. Podemos citar o trabalho de [5], que estuda um método híbrido para a construção de etiquetadores morfológicos, aplicado à língua portuguesa, baseado em autômatos adaptativos.

No SPLN em desenvolvimento, um problema interessante que engloba um conjunto amplo de análises linguísticas já mencionadas e que pode ampliar as funções de criação de atividades é a localização e extração de informações do texto. Em determinados gêneros textuais, como, por exemplo, anúncios de emprego ou artigos científicos, os quais ambos formam os *corpora* utilizados no desenvolvimento do sistema, há padrões de informações que podem ser identificados a fim de evidenciar a estrutura interna do gênero/texto e, por conseguinte, como fonte para a formação de questões.

Nos textos do *corpus* de anúncios de emprego, as seguintes informações podem ser identificadas: quem/qual empresa oferece a vaga, que tipo de vaga é oferecido, quais são as exigências e qualificações necessárias para a vaga, qual o salário e os benefícios oferecidos e como proceder para se candidatar à vaga. Padrões recorrentes como '*We are seeking a…*', '*We offer a competitive salary…*' e '*Send resume to…*' podem funcionar como índices para extração dessas informações.

A princípio, uma função com expressões regulares ou uma gramática poderia ser utilizada para a extração das informações. Porém, no caso, há uma grande possibilidade de implementação de adaptatividade, tendo em vista a natureza do problema, as diferentes variáveis envolvidas, podendo ser aplicadas gramáticas e autômatos em conjunto com dados estatísticos.

Para tanto, faz-se a necessidade de uma avaliação e análise do que pode ser conseguido por meio de métodos simbólicos e estocásticos para a implementação de uma solução híbrida, com possibilidades de características de aprendizagem de máquina em caso de extensão das funcionalidades para textos de diferentes gêneros.

V. Considerações finais

O trabalho buscou mostrar alguns dos avanços feitos em uma pesquisa no desenvolvimento de um SPLN 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.

As informações apresentadas serão desdobradas na exploração do uso da adaptatividade em cada oportunidade de sua aplicação na conciliação dos diferentes métodos discutidos.

Ao final, espera-se a implementação da adaptatividade no sistema, a fim de consolidar o uso de tal tecnologia no projeto proposto, visando a uma contribuição interdisciplinar nas áreas da Linguística e Tecnologia Adaptativa.

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] DIZERÓ, W. Formalismos Adaptativos Aplicados na Modelagem de Softwares Educacionais. 2010. Tese (Doutorado em Engenharia Elétrica e Sistemas Digitais), EPUSP, São Paulo, 2010.

[3] 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.

[4] RAMSHAW, L. AND MARCUS, M. P. (1995). Text chunking using transformation-based learning. In Yarowsky, D. and Church, K., editors, Proceedings of the Third Workshop on Very Large Corpora, pages 82–94, Cambridge, Massachusetts.

[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.

Adaptive Programming in Cyan

J. O. Guimarães and P. R. M. Cereda

***Abstract*—Adaptive devices make it possible to express algorithms in a concise and readable way. However, programming languages usually do not offer explicit support for this technology. The programmer has to employ obscure language constructs and tricks to make the runtime changes demanded by adaptivity. This article shows how the Cyan object-oriented language already supports the adaptive programming style in a limited way. Since the adaptive code is made with known language constructs (rutime reflection), it is relatively easy to learn, use and understand.**

***Keywords*—Adaptive code, Cyan, Object-oriented language**

I. Introduction

Adaptive technology has emerged as an alternative and viable solution for tackling complex problems [1]. There is a quite vast record in the literature that describes methods and techniques on problem solving via adaptive devices such as robotics [2], data mining [3], natural language processing [4], compiler construction [5] [6], privacy and personalization [7], [8], access control [9], and much more. An adaptive device may spontaneously change its own behaviour over time due to some external stimuli.

This paper contributes to the adaptive technology area by showing how adaptive features are supported by the object-oriented language Cyan. The subject is not new: there are programming languages constructs such as that proposed by Freitas e Neto [10] to support adaptive programming. The features presented in this article do not allow illimited code modifications but they are implemented by the regular Cyan constructs.

This paper is organized as follows: the following section describes a subset of the Cyan language used in this paper. Section III shows two mechanisms by which Cyan supports adaptive programming: method insertion and code blocks. The last section concludes the article.

II. The Cyan language

Cyan is a new prototype-based statically-typed object-oriented language that supports single inheritance, Java-like interfaces, statically-typed closures (similar to Smalltalk blocks), an object-oriented exception system, optional dynamic typing, runtime reflection, and many other features. The language has many innovations which are described in its manual [11]. In this paper, we will present only the features necessary to understand how Cyan supports adaptive programming.

Before we proceed, let's set up an example to be used as a proof of concept for presenting the adaptive features of Cyan. The adaptive automaton M, presented in Figure 1, recognizes strings in the form $a^n b^n$, with $n \in \mathbb{N}$, $n > 0$. That is, strings with an arbitrary number of a's followed by the very same number of b's. This language is of course context-free and could be recognized by a simple pushdown automata. The adaptive automaton is overpowered to this task since it has the same computational power of a Turing machine [12]. But it suits our needs to demonstrate how Cyan supports adaptive programming.

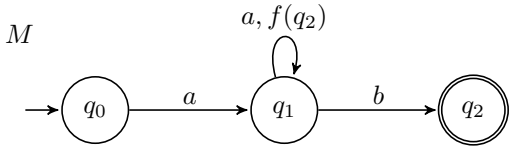


Fig. 1. An adaptive automaton that recognizes strings in the form $a^n b^n$, with $n \in \mathbb{N}$, $n > 0$.

The execution of M is very straightforward: the automaton is initially prepared to recognize the "base case"; that is, ab. If it gets a second a, an adaptive function f is triggered in order to change the topology, adding a new transition consuming b to a newly created state and updating the original transition consuming the first b to go from that state to the acceptance one, so the automaton is now prepared to recognize $aabb$. In other words, for every extra a in the input string, M will add a correspoding b to the "b recognition path". If we want our automaton to also recognize an empty string, we simply need to make q_0 an acceptance state as well.

Since now we have our example properly presented, let's go back to the language. Cyan is a prototype-based language, which means the concept of class is replaced by that of "prototype". A prototype is declared almost exactly like a class except that it is preceded by keyword "`object`". Objects are created from prototypes using methods called `new` or `new:`, which is similar to some class-based languages. Prototypes are grouped in packages which can be imported by other packages. Methods are declared with keyword "`fun`" which should be followed by the method selectors (explained later), return value type, and the method body (between `[` and `]`). An instance variable is declared by putting "`:`", the variable name and its type. An example of Cyan code is given below.

```
package main
   // this is a comment
   /* this is also a comment */

object AF_AnB
       // methods are public by default
       // the method name is "check:"
  fun check: (:in Array<Char>) -> Boolean
     self.in = in
     if ( recognize ) [
          // if the input has ended
          // after recognizition, accepts
```

jose@ufscar.br e paulo.cereda@usp.br.

```
        return index == in size;
    ]
    else [
        return false;
    ];
  ]

  private fun recognize -> Boolean [
    if (in size < 1) [ return false; ]
    else [
        // declare local variable i
        // of type Int with value 0
      :i Int = 0;
      while ( in[i] == 'a' ) [
        ++i;
      ];
        // sends message "size" to "in"
      if ( i < in size ) [
        if ( in[i] == 'b' ) [
           return true;
        ];
      ];
      return false;
    ];
  ]

    /* instance variables are
       private by default */
  :in Array<Char>
end
```

Surprisingly, the previous code implements our adaptive automaton M from Figure 1 with built-in Cyan constructs. Prototype `AF_AnB` has public method `check:` (an `Array<Char>` as parameter, `Boolean` as the return value type) and private method `recognize` (no parameters, returns a `Boolean`). There is one instance variable, `in`, which is an array of `Char`. Inside another prototype of package `main`, one can send a message directly to `AF_AnB`, which is an object that exists at runtime:

```
package main
object Test
  fun test [
     // {# starts a literal array
    AF_AnB check: {# 'a', 'b' #};
  ]
  fun fat: (:n Int) -> Int [
    if ( n <= 0 ) [ return 1; ]
    else [
        return n*(fat: n - 1);
    ];
  ]
end
```

Statement

```
AF_AnB check: {# 'a', 'b' #}
```

is the sending of message `check:` with parameter `{# 'a', 'b' #}` to object `AF_AnB`. At runtime, there is a search in object `AF_AnB` for a method named `check:` that takes a `Array<Char>` as parameter. Since Cyan is statically-typed, the compiler checks wheather prototype `AF_AnB` has a method `check:` with an `Array<Char>` parameter. We use words "object" and "prototype" to refer to "prototypes". When an object is created at runtime, as in:

```
:myTest Test = Test new;
```

the word "object" is used to refer to it. Here message "`new`" is sent to object `Test` to create a new object (it would be "`Test new()`" in Java). `Test` is the type of local variable `myTest`.

Prototype `Test` declares a method `fat:` that takes one `Int` parameter and returns an `Int`. It can be called as "`Test fat: 5`". Inside this method there is a recursive call in "`fat: n - 1`". Since there is no receiver for this message, it is a message send to "`self`", which is the receiver of the original `fat:` message. `self` is similar to `this` of Java/C++/C#.

Following Smalltalk [13], "`fat:`" is called a *selector*. There may be more than one selector for a method each one taking zero or more parameters:

```
object Hash
    fun key: String value: Int [ ... ]
    ...
end
```

This method can be called as

```
Hash key: "one" value: 1
```

We can also declare a selector without parameters:

```
fun add: key: String value: Int
```

Closures, which are unnamed literal functions, are supported by Cyan. They are called "blocks" as in Smalltalk and are declared and used as:

```
:isZero Block<Int><Boolean>;
isZero = [ |:n Int -> Boolean|
    ^ n == 0;
];
if ( isZero eval: 0 ) [
    Out println: "0 is zero";
];
```

The type of variable `isZero` is `Block<Int><Boolean>`, which means a closure that takes an `Int` parameter and returns a `Boolean`. To `isZero` is assigned a closure, which starts with `[` and ends with `]`. Between the '`|`' symbols should appear the parameter and return value (which can be omitted because it can be deduced). This closure takes a parameter called `n`. After the last '`|`' there may appear a list of statements. The return value of the closure is given after symbol '`^`'. Since closures are objects in Cyan, they have methods. In particular, statements that appear inside the closure are put in a method called `eval` or `eval:` that take the same parameters and have the same return value as the closure itself. Then "`isZero eval: 0`" calls the statement inside the closure, which is "`^ n == 0`". Since `n` receives `0`, the parameter to `eval:`, this call returns `true`.

In a regular message send the Cyan compiler makes a search for a method that matches the message send, starting in the prototype that is the static type of the receiver. If a method is not found there, the search continues in the super-prototype[1], super-super-prototype, and so on. Then in a statement

```
person name: "Anna" age: 2
```

the compiler makes a search for a method

```
name: String age: Int
```

in the static type of `person`, which is a prototype. If `person` was declared as "`:person Person`", the search starts at prototype `Person`.

A dynamic message send is a message send in which the compiler does not check wheather the static type of the receiver has a method that matches the signature of the message. It is like a regular message send in which the message selectors are preceded by "?" as in

```
anObj ?name: "Anna" ?age: 2
```

A new method may be dynamically added to a prototype using a method `addMethod: ...` defined in prototype `Any`, the super-prototype of every one. We used "`...`" to elide the many method selectors. The message send that follows add a method called `comb:` to prototype `Test`, the receiver of the message.[2]

```
Test addMethod:
     selector: "comb:"
     param: Int, Int
     returnType: Int
     body: (:self Test) [
         |:n, :k Int -> Int|
         ^(fat: n)/( (fat: k)*(fat: n-k));
     ];
```

Selector `addMethod:` does not take parameters — it is only used to make the code clearer. Selector `selector:` takes one parameter which is the selector name. Selector `param:` is followed by the types of the parameters of `"comb:"` (two `Int`'s). The parameter names should not be cited. The return type is also of type `Int`. The parameter to "`body:`" is a *context block*, a special kind of closure that represents a method that may be added to a prototype at runtime. A context block

```
(:self T)[ ... ]
```

may be added to prototype `T` or its sub-prototypes using method `addMethod:` (as in the example above). Inside this context block, the compiler checks wheather the message sends to `self` match the methods of `T`. For example, `fat: n` is a message send to `self` (since the receiver is implicit). The Cyan compiler searches for a method `fat: Int` in prototype `Test` and its super-prototypes (since the declared type of `self` in this context block is `Test`).

A context block is like a method that is not attached to any prototype but can be added to several of them dynamically. By assigning the context block to a variable, we can pass this variable as parameter to selector `body:` in several calls to `addMethod: ...` thus adding the same context block to several different prototypes.

`addMethod: ...` can add a new method to a prototype, like in the previous example, or it may replace an existing method. In the first case, we cannot call the method as in

```
numComb = Test comb: 5, 2;
```

because the compiler would not find a method `comb: Int, Int` in prototype `Test`. This method should be called using a dynamic message send:

```
numComb = Test ?comb: 5, 2;
```

The context block

```
(:self Test) [ |:n, :k Int -> Int|
    ^ (fat: n)/( (fat: k)*(fat: n - k) );
]
```

takes two `Int` parameters and returns an `Int`. After the message send `addMethod: ...` of the example is sent, method `comb:` is added to `Test`. It is as if we had copied the text of the context block to a newly created method

```
comb: Int, Int -> Int
```

of `Test`. The above context block has three message sends with selector `fat:`. These are message sends to the object to which the context block is attached. In statement "`Test ?comb: 5, 2`", message `fat: n` is sent to `self`, which is `Test`. Method `fat: Int -> Int` of `Test` is called.

There is another way of doing dynamic calls in Cyan. It is using the ‘ operator, which is better explained using an example.

```
:k Int;
:s String = "fat";
   // same as "Test fat: 5"
k = Test ‘s: 5;
s = "test";
    // same as "Test test"
Test ‘s;
```

If `str` is of type `String` and holds string `strValue`,

```
anObj ‘str
```

is a dynamic message send equivalent to

```
anObj strValue
```

The compiler considers that the return value type is `Any`, the super-prototype of every one (the equivalent of `Object` of Smalltalk and Java).

III. Development

As presented in the previous section, the reflective features of Cyan can be used for implementing adaptive code in an unlimited way. However, by lack of space in this article we will only show two ways of implementing adaptive features in the language: by *method insertion* and by simulating *code blocks*.

[1] Analogous to superclass.

[2] The method is always added to the receiver.

A. Adaptive features by method insertion

The prototype AF_AnBn, which maps our adaptive automaton M and it's given below, recognizes the language $L = \{a^n b^n \mid n \in \mathbb{N}\}$. Method check: calls method a to recognize a^n and b to recognize b^n. It only returns true if both methods return true and the input was fully consumed.

Initially method b returns true regardless of the input. This should be changed according to the number of 'a' symbols in the beginning of the input in. When method a is called, it scans all the 'a' symbols it finds and then adds to prototype AF_AnBn a new method b that will scans a number of 'b' symbols that is equal to the number of 'a's already found. The addition of a new method b is made with addMethod: ... with a context object. Note that, when the context block is created, the value of local variable n is copied to an instance variable of the context block. That means any changes to variable n inside the context block does not propagate to the local variable n declared in method a.

```
package main

object AF_AnBn
  fun check: (:in Array<Char>) -> Boolean
     self.in = in
     index = 0;
     if ( a && b ) [
          // if the input has ended,
          // accepts
         return index == in size;
     ]
     else [
         return false;
     ];
  ]

  private fun a -> Boolean [
     :n = 0;
       // recognizes n symbols 'a'
     while ( index < in size &&
             in[index] == 'a' ) [
         ++index;
         ++n;
     ];
        // replaces method 'b -> Boolean'
     AF_AnBn addMethod:
        selector: #b
        returnType: Boolean
        body: (:self AF_AnBn) [
            | -> Boolean |
            :newN = 0;
            while ( index < in size &&
                    in[index] == 'b' ) [
                ++index;
                ++newN;
            ];
            return newN == n;
        ];
     return true;
  ]

  private fun b -> Boolean [
     return true;
  ]

  public :in Array<Char>
  public :index Int

end
```

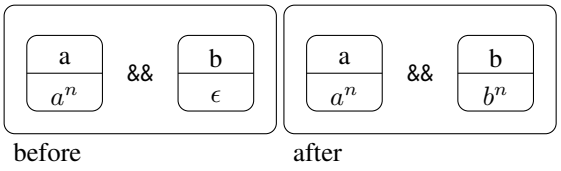


Fig. 2. Languages recognized by methods a and b.

Method check tests wheather a && b is true. Then method a and method b are called, in this order (there is no short-circuit evaluation). If method a finds n 'a' symbols in the beginning of the input, it adds to prototype Test a method b that checks if the next n symbols of the input are 'b' symbols. If this is false, the method returns false.

Figure 2 shows grafically what happens at runtime. The rectangle labelled "before" shows methods a and b and the strings they recognize in the first call to method check of AF_AnBn. Initially method b recognizes the empty string. Rectangle "after" shows these methods after method a is called. This call changes method b in such a way that it now recognizes language b^n for $n \in \mathbb{N}$. The value of n is known after all 'a' symbols are found. This value is used to create an appropriated method b, which is a context object.

Method a replaces method b with a context object that can be attached to prototype AF_AnBn and its sub-prototypes (because it is declared as (:self AF_AnBn)[...]). Then inside the context object one can send messages that match the methods of this prototype. Apparently there is no message send to self inside this context object. But in fact there are several of them because the public instance variables in and index of Test are transformed into public get and set methods that are called in statements like ++index and expressions like index < in size.

B. Code blocks in Cyan

Methods can be dynamically added to a prototype and called using a string holding the method name (as in "Test 's"). This scheme can be used to simulate code blocks in Cyan.

A code block is simply an object with a run method (or a piece of code that is put in a list. Then at runtime the run methods of the objects are called by a special method that we call a "combinator". The *combinator* may use many algorithms to define the order in which the run methods are called. It may call one by one in the list order, it may call only the method of the first list object, it may use the integer return value of one run method to define the next method to call,

and so on. There may be more than one combinator for a list, each one implementing different algorithms. For example, one combinator can recognize language $L = \{a^n b \mid n \in \mathbb{N}\}$ and other may recognize $L = \{ab \mid n \in \mathbb{N}\}$, both starting with the same object list.

The run methods may as parameter a hashtable used as shared memory. The pairs in the table would be composed by a variable name and an object. This shared table can be used for communication among the list objects.

The list can be changed dynamically to adapt the code: methods can be removed, added, and replaced. Then it makes sense to have a combinator that calls always method run of the first list object. Thi is because the first object may change from call to call.

Code blocks are implemented in Cyan using operator ‘ that calls the method whose name is in a string. The example that follows shows a prototype whose method check: returns true if its parameter belongs to the language $\{a^n b^n : n \in \mathbb{N}\}$. Method check: initializes an array funArray of methods to be called by the compinator. Instead of objects with a run method, here we put the name of the methods, as strings, in the list (which is an array). The combinator gets each of the strings of funArray and calls them in the order they appear. Initially there is only one string, "a", in funArray[0]. Method combinator puts this string in variable methodName (see first statement after while in this method). Method a is called in the message send

```
self ‘methodName
```

Method a inserts in prototype CodeBlocks a method b and in funArray a string "b". Method b is inserted using addMethod: ... as before. Statement

```
funArray add: "b"
```

inserts "b" at the end of array funArray.

```
object CodeBlocks
  fun check: (:in Array<Char>) -> Boolean
    self.in = in
    index = 0;
    funArray = {# "a" #};
    return combinator;
  ]

  fun combinator -> Boolean [
    :i = 0;
    while ( i < funArray size ) [
        :methodName String = funArray[i];
        if ( self ‘methodName == false ) [
            return false;
        ];
        ++i;
    ]
    return true;
  ]

  fun a -> Boolean [
    :n = 0;
      // recognizes n symbols 'a'
    while ( index < in size &&
            in[index] == 'a' ) [
      ++index;
      ++n;
    ];
    if ( n != 0 ) [

      // replaces method 'b -> Boolean'
      CodeBlock addMethod:
         selector: #b
         returnType: Boolean
         body: (:self CodeBlocks) [
           | -> Boolean |
           :newN = 0;
           while ( index < in size
             && in[index] == 'b' ) [
             ++index;
             ++newN;
           ];
           return newN == n;
         ];
      funArray add: "b";
    ];
    return true;
  ]

  :funArray Array<String>
  public :in Array<Char>
  public :index Int
end
```

As seen previously, code blocks implemented as a list of objects with a run method need a hashtable that is used as a shared memory. That is not necessary with the implementation of code blocks in Cyan because the instance variables of prototype CodeBlocks are used as shared memory. Let us explain that.

Methods like b are added to prototype CodeBlocks using addMethod: ... with a context object targeted to prototype CodeBlocks (this is the type of self in the context object). Therefore all added methods can call methods of CodeBlocks in message sends to self. Then all added methods share the CodeBlocks instance variables, which work like a shared memory for them.

IV. Final remarks

The Cyan constructs used in this paper are limited in their power — they are regular language constructs. However, their limitations makes them more foreseeable since many code transformations, like inserting a statement in the middle of the code, are not possible. By using regular language features we assure that a regular programmer can understand them rather easily.

By using adaptive programming, we aim at reducing the burden of writing complex code; once the transformation rules are defined, the code evolves through a well-established space

of configurations without external interference. Problems that require intricate interations may benefit from the techniques presented in this paper since they are made without adding any new constructs to the Cyan language. Therefore, they are relatively easy to understand and use.

REFERENCES

[1] J. José Neto, "A small survey of the evolution of adaptivity and adaptive technology," *Latin America Transactions, IEEE (Revista IEEE America Latina)*, vol. 5, no. 7, pp. 496–505, 2007.

[2] A. R. Hirakawa, A. M. Saraiva, and C. E. Cugnasca, "Adaptive automata applied on automation and robotics (a4r)," *Latin America Transactions, IEEE (Revista IEEE America Latina)*, vol. 5, no. 7, pp. 539–543, 2007.

[3] P. R. M. Cereda and J. José Neto, "Mineração adaptativa de dados: Aplicação à identificação de indivíduos," in *Memórias do Sexto Workshop de Tecnologia Adaptativa*, São Paulo, 2012.

[4] R. L. A. Rocha, "Adaptive technology applied to natural language processing," *Latin America Transactions, IEEE (Revista IEEE America Latina)*, vol. 5, no. 7, pp. 544–551, 2007.

[5] J. José Neto, "Contribuições à metodologia de construção de compiladores," Tese de livre-docência, Escola Politécnica da Universidade de São Paulo, São Paulo, 1993.

[6] J. C. Luz and J. José Neto, "Tecnologa adaptativa aplicada à otimização de código em compiladores," in *IX Congreso Argentino de Ciencias de la Computación*, La Plata, Argentina, 2003.

[7] P. R. M. Cereda and S. D. Zorzo, "Formalismo com autômato adaptativo em mecanismo de privacidade e personalização," in *Latin American Computing Conference*, San José, Costa Rica, 2007.

[8] P. R. M. Cereda, R. A. Gotardo, and S. D. Zorzo, "Resource recommendation using adaptive automaton," in *16th International Conference on Systems, Signals and Image Processing*, 2009.

[9] P. R. M. Cereda and S. D. Zorzo, "Access control model formalism using adaptive automaton," *Latin America Transactions, IEEE (Revista IEEE America Latina)*, vol. 6, no. 5, pp. 443–452, 2009.

[10] A. V. Freitas and J. J. Neto, "Adaptivity in programming languages," *WSEAS Transactions on Information Science & Applications*, vol. 4, no. 4, pp. 779–786, Apr. 2007.

[11] J. de Oliveira Guimarães, "The Cyan language," 2012. [Online]. Available: http://www.cyan-lang.org

[12] R. L. A. Rocha and J. José Neto, "Autômato adaptativo, limites e complexidade em comparação com máquina de Turing," in *Proceedings of the second Congress of Logic Applied to Technology*, 2001.

[13] A. Goldberg and D. Robson, *Smalltalk-80: the language and its implementation*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1983, ISBN: 0-201-11371-6.

José de Oliveira Guimarães José de Oliveira Guimarães is a UFSCar Professor since 1992. He has an undergraduate degree by USP/São Carlos, a master degree by Unicamp, and a Doctorate from USP/São Carlos. He researches in Programming Languages and Software Engineering (toos, user interface in IDE's, and design patterns). His main research area is object-oriented programming. José has designed the object-oriented languages Green and Cyan. Other interest areas are Computability and Axiomatic Set Theory.

Paulo Roberto Massa Cereda is a PhD candidate in Computer Engineering working on adaptive technology at the Languages and Adaptive Techniques Laboratory (LTA), at Escola Politécnica, University of São Paulo under the supervision of prof. João José Neto.

An Adaptive Approach for Error-Recovery in Structured Pushdown Automata

J. J. Neto, H. Pistori, A. A. Castro Jr. and M. R. Borth

***Abstract* — In this paper, we introduce a practical way to implement an efficient error-recovery scheme intended to handle incorrect input received by devices based on finite-state and structured pushdown automata. We start proposing an exact recovery scheme for single errors in deterministic finite-state devices. Then, we extend this approach to perform local recovery in the individual sub-machines of structured pushdown automata, and we complement the proposed single error-recovery scheme by considering non-deterministic devices and interactions among sub-machines. Finally, we add a panic-mode scheme to the proposed method in order to take into account even multiple-error inputs. The chosen adaptive approach used in this strategy leaves untouched the underlying original automaton until an error is detected in its input stream. At this moment, an adaptive action is taken that adequately extends the automaton in order to perform the needed recovery in response to the detected error. Afterwards, the transitions included in the extension are executed according to the contents of the input stream, and when the underlying automaton is reached anyway, the extension is discarded, restoring the automaton to its original shape.**

***Keywords* — Structured Pushdown Automata, Adaptive Automata, Error Recovery, Compiler Construction.**

I. INTRODUCTION

Language acceptors are expected to recognize symbol sequences belonging to the language they define. In practice, strings given as input to language acceptors often include misspellings and grammatical errors, and acceptors must reject them in this case. When used within a language processor, an acceptor should not simply reject the string, but bypass the error in order to proceed searching for further errors in its input.

In general, error-recovery is a complex subject, even when focusing low-complexity devices such as finite-state automata [1, 2, 3, 4, 5, 6, 9, 10, 11]. Two adjacent errors are called simple when they are sufficiently far apart that their effects in the behavior of the automaton are not superposed, otherwise they are said to be multiple. By observing practical programs, one may remark that simple errors are significantly more likely to occur than multiple ones, and that for most practical situations no complex methods are needed for handling them. Nevertheless, multiple errors do occur, and by handling them as exceptions, it is possible to concentrate efforts in recovering simple errors.

The presence of an error in the input text is detected whenever the acceptor cannot perform any valid transition in response to some input symbol. Symptoms of the presence of an error seldom occur in its neighborhood, leading the acceptor to mistakenly perform a sequence of transitions before rejecting the input string. Although being inadequate for the particular input string being handled, note that those transitions would be perfectly valid for other correct input sentences. Such unavoidable imprecision usually make automatic error-repair procedures far worse than a manual correction performed by the text's author. Therefore, instead of viewing error recovery as an approach to fix the input text errors, we will use it for re-synchronizing the acceptor with its wrong input string, and proceed searching for further errors.

An adaptive automaton is an adaptive rule-driven device whose subjacent mechanism is a structured pushdown automaton. This formalism preserves intact most part of the clean syntax of structured pushdown automata, allowing a very intuitive view for the resulting Turing-powerful formalism as an automata with a dynamically changeable set of transitions. Such modifications are expressed through a very simple language where queries, deletions and insertions of transitions are specified by adaptive functions to be executed just before or after the execution of some transition. The self-modification required to model an error-recovery behavior can be elegantly captured by adaptive automata. The efficiency of such method is also discussed and illustrated in this work, through the analysis of an implemented scheme of adaptive error-recovery.

The next section provides some notational revision for structured pushdown automata and adaptive automata. Afterwards, a classical error-recovery mechanism is presented. Section 7 presents in details our adaptive error-recovery approach, describing also some implemented examples. Last section gives some conclusions and future work proposals.

II. NOTATION

A. Structured Pushdown Automata Revisited

A structured pushdown automaton (SPA) may be viewed as a set of finite state machines with special transitions for passing the execution control back and forth among machines. Formally, a structured pushdown automaton is a tuple $M = (S, Q, \mu, \Sigma, \Gamma, Q_0, F, \delta^i, \delta^e)$ where:

- S is a finite non-empty set of sub-machines.
- Q is a finite non-empty set of states.

J. J. Neto, Polytechnic School of the University of São Paulo (EPUSP), São Paulo, Brazil, joao.jose@poli.usp.br

H. Pistori, Dom Bosco Catholic University (UCDB), Campo Grande, Mato Grosso do Sul, Brazil, pistori@ucdb.br

A. A. Castro Jr., Federal University of Mato Grosso do Sul (UFMS), Ponta Porã, Mato Grosso do Sul, Brazil, amaury.ufms@gmail.com

M. R. Borth, Federal Institute of Mato Grosso do Sul (IFMS), Ponta Porã, Mato Grosso do Sul, Brazil, marceloborth@gmail.com

- μ: $Q \rightarrow S$ is a function. The inverse of μ, μ', induces a partition on set Q, with each part corresponding to the set of states of each sub-machine. Given a sub-machine $s \in S$, $\mu'(s) \subseteq Q$ is the set of states of s.
- Σ is a finite non-empty input alphabet.
- Γ is a finite set that is called the stack alphabet.
- $Q_0 \subseteq Q$ is a set of initial states where, for each $s \in S$, $|\ \mu'(s) \cap Q_0\ | = 1$ (Each sub-machine has one, an only one, initial state).
- F is a set of final states, which also corresponds to sub-machine return transition.
- δ^i is a relation over $\mu'(s) \times \Sigma \cup \{\epsilon\} \times \mu'(s)$, where $s \in S$. Each element of this relation is called an internal transition.
- δ^e is a relation over $\mu'(s) \times S \times \mu'(s)$. Each element $(q, m, q') \in \delta^e$ is denominated an external transition or a sub-machine call, and can be interpreted as a sub-machine call from state q to the sub-machine m pushing the state q' onto a pushdown store.

Configurations in SPA are triples (q, x, z) where $q \in Q$ is the current state, $x \in \Sigma^*$ is the part of input string yet to be read and $z \in Q^*$ is the content of the pushdown store, holding a sequence of sub-machine return addresses. When $\delta^e = \emptyset$ and $|S| = 1$ the SPA specializes to an ϵ-FSA, operating as such (the third element is not used). Otherwise, the step relation, $\vdash$, which determines the machine operation, is extended to comprise the sub-machine call and return dynamics. Formally, given a SPA $M = (S, Q, \mu, \Sigma, \Gamma, Q_0, F, \delta^i, \delta^e)$, $q, q' \in Q$, $\sigma \in \Sigma \cup \{\epsilon\}$, $x \in \Sigma^*$, $y, y' \in Q^*$ we have $(q, \sigma x, y) \vdash_M (q', x, y')$ if and only if one of these three conditions apply:

1. $(q, \sigma, q') \in \delta^i$, $y = y'$ and $\mu(q) = \mu(q')$. [Internal transition]
2. $(q, s, p) \in \delta^e$, $q' \in Q_0 \cap \mu'(s)$, $y' = py$, $\mu(q) = \mu(p)$ and $\sigma = \epsilon$. [Sub-machine call]
3. $q \in F$, $y = q'y'$ and $\sigma = \epsilon$. [Sub-machine return]

B. Adaptive Automata Revisited

An adaptive automaton is an adaptive device $A_D = (S_M, A_M)$ whose subjacent mechanism is a structured pushdown automaton $SM = (S, Q, \mu, \Sigma, \Gamma, Q_0, F, \delta^i, \delta^e)$, as defined above; and the adaptive mechanism $A_M = (A_F, A_C)$, comprises a set of adaptive functions, A_F, and a function $A_C : \delta^i \cup \delta^e \rightarrow (A_F)^2$. The function A_C links each transition to a pair of adaptive functions to be executed before and after the transition. The set A_F includes the special symbol ϵ, representing the empty adaptive function. The elements of A_C are called adaptive actions, or adaptive function calls.

Each adaptive function $f \in A_F$ is a set of elementary adaptive actions, of one of three kinds: search, erase and insert elementary action. Search actions are patterns used for selecting the transitions that erase and insert actions operate on. Patterns, in this context, are transition-shaped structures whose states, input symbols and adaptive actions may be replaced by variables and generators. In this work we choose to use implicit variable declarations. Each occurrence of a variable is prefixed with a question mark. Generators designate a symbol not used elsewhere, which is intended to be used in the dynamic creation of states in the automaton. Like variables, generators, denoted with the asterisk prefix, are implicitly declared. Implicit declarations are useful for graphical representation. Since graphics are essentially non-linear, they may be read from any directions (explicit declarations would eliminate this freedom). Finally, a reserved variable, *?c*, is used for referencing the current state during automaton operation. This amendment may be used, in many cases, to replace the formerly defined [8] adaptive action parameter passing mechanism, which is not used here, so it has been omitted for simplicity.

Figure 1 represents an adaptive automaton that recognizes the classical context-dependent language $a^n b^n c^n$. The subjacent mechanism, S_M, keeps reading the symbols a and calling the adaptive function F, denoted $[.F]$ (the dot in the notation indicates that F is called after the execution of the state transition), for each symbol a read. The adaptive function just seeks for the ϵ transition (here working as a mark) and replaces it by a pair of transitions that reads the substring bc. The ϵ transition mark is kept between the transitions that read b and c, so that when the i-th a is input, the automaton will have a sequence of transitions that are able to consume the sequence $b^i c^i$.

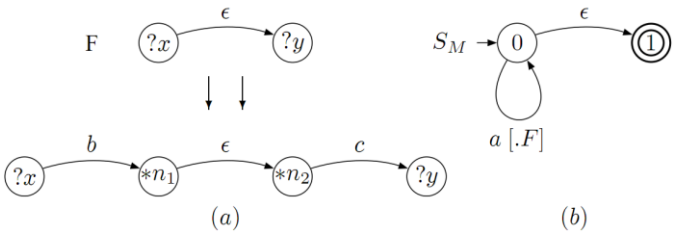


Figure 1. Adaptive Automaton that Recognizes $a^n b^n c^n$. (a) Adaptative Mechanism (b) Subjacent Mechanism.

C. Auxiliary Concepts and Definitions

Definition Let $M = (S, Q, \mu, \Sigma, Q_0, F, \delta^i, \delta^e)$ be an SPA and $q \in Q$. *First*: $Q \rightarrow 2^Q$ is a function where $q' \in$ *First(q)* if and only if $\exists \sigma \in \Sigma\ |\ (q, \sigma x) \vdash^*_M (q', x)$, $x \in \Sigma^*$ [1]. It calculates the set of states reachable from q, after reading one symbol.

Definition *SFirst* : $Q \rightarrow 2^\Sigma$ is a function where, for each $q \in Q$, $\sigma \in$ *SFirst*(q) if and only if $(q, \sigma x) \vdash^*_M (q', x)$, $x \in \Sigma^*$, $q' \in$ *First*(q).

Definition Given a state $q \in Q$ from an SPA M, let *First*$(q) = \{q_0, q_1, ..., q_n\}$. The function *Second* : $Q \rightarrow 2^Q$ is defined for each element $q \in Q$ as:

$$Second(q) = \bigcup_{0 \le i \le n} First(q_i)$$

Given some state $q \in Q$, the reserved symbol θ will denote the set $\Sigma - SFirst(q)$. Assume that each state q will carry a special transition, the error transition that is automatically activated whenever, being in q, the automaton reads a symbol in θ. The destination of such transition is a trap, non-final,

[1] $\vdash^*$ denotes the transitive closure of the step relation, $\vdash$

error state that will consume all the remainder of the input string.

III. Classic Simple Error Recovery

Given a finite state automaton, an error-recovery strategy should extend the automaton to allow it to keep consuming the input string despite the error detected. Omission, insertion and substitution of a symbol in the input string are the only sources of simple errors, which may be recovered through the reinsertion of an omitted symbol, the omission of an inserted symbol or the substitution of a wrong symbol by the correct one. The elimination of incorrect symbols may be done by adding transitions consuming the wrong symbols in the current state and leading the automaton to a specific error recovery state.

In order to complete our simple-error recovery mechanism, two further transitions are needed, replicating the normal transitions into the added extension: the first one departs from the new recovery state and the second, from the original state. A special care must be taken for preserving the structure of the original device: instead of inserting error-recovery transitions directly to the automaton, an empty-transition is added from the state in which the error is detected to an auxiliary error-recovery state. In operation, such empty-transition is activated only when no other normal transition is allowed. That is achieved by imposing greater priority to normal transitions than to empty ones, and its effect is that the recovery extension is activated only in case of errors.

The following algorithm implements the recovery scheme described:

```
Input: FSA M = (Q, Σ, q0, F, δ)
Output: M with Simple Error Recovery

For each state q ∈ Q Do
  Add two new states, e1 and e2, to Q
Add the empty transition (q, ϵ, e1) to δ //Isolate
Error States

For each c ∈ Q – {First(q) ∪ Second(q)} do
  Add the transition (e1, c, e2) to δ //Consume
Wrong Symbol
For each qs ∈ Second(q) Do
  Let b be the symbol of Σ that guarantee the
presence of qs in Second(q)
  Add the transition (e1, b, qs) to δ //Elimination
Error
  Add the transition (e2, b, qs) to δ //Substitution
Error
For each qf ∈ First(q) Do
  Let a be the symbol of Σ that guarantee the
presence of qf in First(q)
  Add the transition (e2, a, qf) to δ //Insertion
Error
If q is a final state then e2 must be made final too
```

Assuming $\Sigma = \{a, b, c\}$, Figure 2 shows the application of the error recovery mechanism to state p. Added transitions appear in dotted lines.

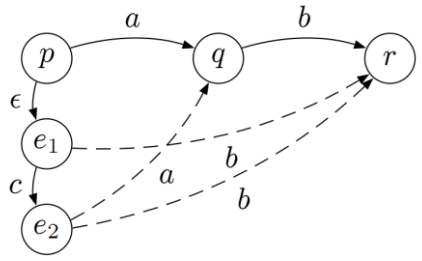


Figure 2. Simple Error Recovery

IV. panic Mode Error Recovery

Further transitions may be added to the described extension, allowing multiple errors to be handled too. A simple way to include multiple-error-recovery consists in successively eliminating symbols from the input data until some special symbol is found that allows the automaton to proceed to some corresponding synchronization state. Although being forceful, this technique produces good results in most practical cases.

By inspecting the language, a set of synchronizing symbols may be chosen such that their presence in the input text determines the start, the end or some significant point of the sentence. In order for this technique to be effective, we should choose synchronizing symbols that be likely to occur in typical input texts. In practice, it is usual to include symbols or keywords that delimit commands, expressions or groupings, as well as operators, separators and punctuation symbols. The best synchronization symbols are those that do not belong to more than one syntactic construct that are likely to occur simultaneously anywhere in the input text. From each new state created in the error-recovery extension, corresponding to the points where unsuitable symbols are discarded:

- Add a set of transitions for eliminating all non-synchronizing symbols, holding the automaton in the same state, until any synchronizing symbol is found.
- Add a set of transitions for consuming any synchronizing input symbol, moving the automaton to the state it would reach in the original automaton upon finding such symbol in correct sentences

This practice simulates replacing the discarded part of the input string by another one the automaton would expect to find instead. This technique has a wide application since the less the requirements on the use of the input string in case of multiple errors, the simpler its implementation will be. The following algorithm, which could be easily optimized if executed along the simple-error recovery, summarizes the multiple error recovery mechanism.

```
Input: FSA M with Simple Error Recovery
Output: M with Simple and Multiple Error Recovery
Let S be the set of synchronizing symbols
For each state e2 ∈ Q Do
  For each s' ∉ S Do
    Add the transition (e2, s', e2) to δ
  For each s ∈ S Do
    Let qs be the destination state of the transition
      consuming s in M
    Add the transition (e2, s, qs) to δ
```

Obviously, the symbols in *S* that are more adequate for this purpose are those that, at least in the context of recovery have unique corresponding q_s. Although the above technique has been use to complement the handling of simple errors, it may be used alone with the original automaton, especially for situations in which no rigorous recovery is needed.

The procedure described so far is enough to recover errors in a finite state automaton. However, it generates too many states and transitions, and its behavior is often non-deterministic. Additionally, error handling occurs only when there are errors in the input text, making the resulting extension remain unused in all normal cases. However, the extensions referring to each original state are mutually independent and independent of the original automaton, so it is possible to consider each of them separately, and to activate the proper one exclusively when the corresponding specific error is detected. Such independence creates an option for the designer, allowing that only the desired parts of the extension mechanism to be used. A practical option consists in pre-building all recovery extensions without physically inserting them into the original automaton, and activating them from disk only when an actual error detection occurs. Another good option is the subject of this paper, and consists of building and executing the extensions strictly at run-time, when the error is actually detected.

V. Error Recovery in SPA

Many authors address error recovery in traditional LR and LL pushdown automata [9, 4]. Being structured pushdown automata deeply based on finite-state devices, we may adapt the methods described above in order to recover errors in structured pushdown automata. The following cases have to be considered in this case:

- At internal transitions not involving final states all methods used for finite-state automata may be applied without change.
- At sub-machine call the contents of the pushdown store change and some action must take place in response.
- At sub-machine return, when a sub-machine finishes its operation, there is complementary change in the pushdown store, requiring some corresponding action.

Final states in sub-machines conceptually differ from those in finite-state automata, since the former represent the end of the syntactic construct defined by the sub-machine while the later indicate the end of the whole sentence. Therefore, while the detection of an error at the final state of a finite-state automaton initiates some error-recovery procedure, a similar situation in a sub-machine must be interpreted as a valid condition for returning to its caller sub-machine. Therefore, on the detection of an error at some final state of a sub-machine that is not the final state of the automaton, we must verify the behavior of the automaton in all states reachable after returning and consuming the next input symbol. In addition to recovering errors corresponding to internal transitions we will now consider interrelations among sub-machines in our recovery strategy. As the definition of first and second successors are based on the step relation of structured pushdown automata, a careful reading reveals that they already apply to the cases described above. However, the algorithm must be slightly modified by replacing internal transitions to sub-machine calls, when some first or second successor does not belong to the same sub-machine.

A. Multiple-error recovery in SPA

This scheme follows the one presented before for finite-state automata. In addition to the intricate methods needed for finite-state error recovery, the contents of the pushdown store affect the behavior of the automaton, therefore two cases must be considered: errors detected while the sub-machine to which the destination states of the error-recovery transitions belong, and the more complex case in which such state belongs to the calling machine. For further cases, we will adopt empirical recovery criteria.

In the first case, recovery may be locally done, since it considers only transitions internal to the sub-machine and independent of the pushdown store. In the second case, we need transitions from one sub-machine to another, conditioned to the pushdown store contents. In this case, we choose recovery transitions with destination states in the same sub-machine the top of the pushdown store refers to. Error recovery involving the current sub-machine and some other external one must provide the elimination of information previously stacked in the pushdown store, so that some reference to that sub-machine is found in the pushdown store, which is also popped out.

VI. Adaptive Error Recovery

Implementing error-recovery with adaptive automata may be achieved as follows: attach to each error transition one adaptive function, say E_1, which will perform structural transformations on the automata in order to absorb the error. This adaptive function will implement the error-recovery strategy described so far as an intrinsic part of the formalism. Some special transitions created by this adaptive function will call another adaptive function, E_2, which erases all transitions created by E_1. The adaptive solution also lowers space cost, since all transitions related to error-recovery, which should be replicated at all normal transitions in the classic solution are created and destroyed just as needed. Figure 3 illustrates adaptive functions E_1 and E_2.

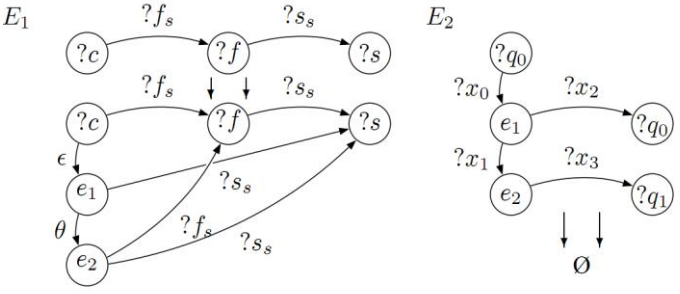


Figure 3. Adaptive Functions for Error Recovery

The algorithms presented in this paper were implemented with AdapTools 1.1[2], a software that offers a graphical environment where adaptive automata can be designed, implemented and tested. AdapTools embodies debugging and visualization tools, as well as a set of examples that may be executed by a special virtual machine included in the package. Among these examples is a compiler-compiler that produces a SPA-based syntactic acceptor, with the adaptive error-recovery described in this paper, from a Wirth notation grammar specification of the language. Figure 4 shows the AdapTools code that implements the example in Figure 2 using adaptive functions. E1 and E2 are the adaptive function shown in Figure 3. Using AdapTools we tested this automaton with different input string and could verify that it can recover from simple errors, growing in size only during the recovering process, returning to the initial size afterwards (due to the E2 adaptive function). Experiments comparing the adaptive and non-adaptive solutions were not conducted for this paper.

	Head	Orig	Inpu	Dest	Push	Outp	Adap
1	S	0	a	1	nop	nop	.E1(1,b,2)
2	S	1	b	2	nop	nop	E2.
3	S	2	c	0	fin	nop	nop
4	+E1	$\%_1$	a	9990	nop	[ERRO-Ins	nop
5	+E1	9990	$\%_2$	$\%_3$	nop	]	.E2
6	+E1	$\%_1$	eps	9990	nop	[Erro	nop
7	+E1	9990	eps	$\%_3$	nop	-Rem]	.E2
8	?E2	?x1	?x2	9990	?x3	?x4	?x5
9	?E2	9990	?y1	?y2	?y3	?y4	?y5
10	-E2	?x1	?x2	9990	?x3	?x4	?x5
11	-E2	9990	?y1	?y2	?y3	?y4	?y5

Figure 4. Code for adaptive automata with error recovery in AdapTools

VII. Conclusions and Future Work

In this paper, we presented a very practical approach to error-recovery, based on adaptive automata. This approach places the error-recovery scheme at the same formalization level of the subjacent structure, allowing error handling as a part of the machine. For instance, an adaptive automaton could be easily projected to dynamically turn the error-recovery procedures on and off, in response to adaptive actions.

The time and space complexity of our adaptive error-recovery approach is constant, both over the input string and the size of Q (states). The complexity depends on the transitions departing from a specific state, however, in practical problems, it does not lower the automata overall performance, since the size of the input alphabet is usually very small when compared to the size of the states set.

Some future experiments using adaptive error-recovery include the use of the adaptive automata self-transformation power to create a more sophisticated error-recovery mechanism. For instance, the error-recovery adaptive function may, interacting with the user or based on previous run information, detect the most likely corrections for specific errors, and change the structure permanently, so that further errors of the same kind will be automatically corrected. Another interesting research topic would be the extension of the adaptive error-recovery approach to deal with sequences of errors and the application of such extension to problems related to the edit-distance of two strings [7]. Error recovery is recently gaining attention from the computer vision community as some groups are reviving the syntactical pattern recognition approach augmented with recent advances in feature extraction techniques and more powerful machines [12, 13, 14].

[2] Freely available at https://code.google.com/p/adaptools/

References

[1] A. V. Aho and J. D. Ullman, “The Theory of Parsing, Translation and Compiling”, vol. 1 and 2. Prentice Hall, 1979.

[2] R. C. Backhouse, “Syntax of Programming Languages - Theory and Practice”. Prentice Hall, 1979.

[3] W. A. Barrett and J. D. Couch, “Compiler Construction - Theory and Practice”. SRA, 1979.

[4] K. J. Gough, “Syntax Analysis and Software Tools”. Addison-Wesley, 1988.

[5] R. Hunter, “The Design and Construction of Compilers”. John Wiley and Sons, 1981.

[6] S. Llorca and G. Pascual, “Compiladores - Teoría y Construcción”. Paraninfo, 1986.

[7] M. Mohri, “Edit-distance of weighted automata”. Conference on Implementation and Application of Automata - CIAA 2002 (July 2002), 7–29.

[8] J. J. Neto, “Adaptive automata for context-sensitive languages”. SIGPLAN NOTICES 29, 9 (September 1994), 115–124.

[9] J. P. Tremblay and P. G. Sorenson, “The Theory and Practice of Compiler Writing”. McGraw-Hill, 1985.

[10] W. M. Waite and G. Goos, “Compiler Construction”. Springer-Verlag, 1984.

[11] F. Almeida, J. Urquiza, Á. Velázquez, "Educational visualizations of syntax error recovery", 1st Annual IEEE Engineering Education Conference (EDUCON 2010), 2010, pp. 1019-1027.

[12] H. Pistori, P. Flach, A. Calway, “A new strategy for applying grammatical inference to image classification problems”, IEEE-ICIT International Conference on Industrial Technology (February 2013), 2013.

[13] B. Yao, X. Yang, L. Lin, M. Lee, and S. Zhu, “I2t: Image parsing to text description,” Proceedings of IEEE, vol. 98, no. 8, pp. 1485–1508, August 2010.

[14] R. Damaševičius, “Structural analysis of regulatory DNA sequences using grammar inference and support vector machine” Neurocomputing, vol. 73, pp. 633–638, January 2010

João José Neto holds a BS in Electrical Engineering (1971), Master in Electrical Engineering (1975), Ph.D. in Electrical Engineering (1980) and full professor (1993) from the Polytechnic School of the University of São Paulo. He is currently associate professor at the Polytechnic School of the University of São Paulo and coordinates LTA – Laboratory of Languages an Adaptive Technology of PCS – Department of Computer Engineering and Digital Systems EPUSP. Has experience in the area of Computer Science, with emphasis on Fundamentals of Computer Engineering, acting on the following topics: adaptive devices, adaptive technology, adaptive automata, and its applications to computer engineering, particularly in adaptive decision making systems, analysis and processing of natural languages, compiler construction, robotics, computer-assisted teaching, modeling of intelligent systems, machine learning processes and inferences based on adaptive technology.

Hemerson Pistori (Três Lagoas, MS, Brazil, 1970) coordinates the R&D&i computer vision group, INOVISAO, at the Dom Bosco Catholic University, UCDB, where he works since 1993. Professor Hemerson was a founder of the department of Computer Engineering at UCDB and has held the position of department head from 1998 to 2001. He also served as the chairperson of UCDB's scientific committee, as the director of research and since 2009 is the Dean of Research and Graduate Studies of this institution. He holds a permanent professor position at the Biotechnology graduate program and recently helped to implement a new graduate program on Environment Science and Agriculture Sustainability at UCDB. His BA and MA in Computer Science were obtained from the UFMS and UNICAMP universities, respectively, and the Ph.D. in Computer Engineering was held at USP. From 2011 to 2012 he stayed 6 months with the University of Bristol, UK, as a visiting researcher.

Amaury Antônio de Castro Junior is graduated in Computer Science at Federal University of Mato Grosso do Sul (1997), Masters in Computer Science at Federal University of Mato Grosso do Sul (2003) and Ph.D. at Polytechnic School of the University of Sao Paulo (2009). Currently holds the position of Regional Secretary of the SBC (Brazilian Computer Society), responsible for the state of Mato Grosso do Sul. Works since 2006 as associate professor at Federal University of Mato Grosso do Sul (UFMS) - Campus Ponta Pora (CPPP). Was the founder of Ponta Porã's Robotic Laboratory (LaRPP) and coordinates several projects of research and extension. Currently working with applied research on robotics and related areas. Has experience in Computer Science, with an emphasis Theory of Computing, acting on the following subjects: Adaptive Technologies, Adaptive Automata, Design of Programming Languages and Computer Model.

Marcelo Rafael Borth is Ph.D. candidate in Environmental Sciences and Agricultural Sustainability at Dom Bosco Catholic University - UCDB, masters in Computer Science at State University of Maringá - UEM and graduated in Information Systems at Paranaense University. Works as Professor at Federal Institute of Education, Science and Technology of Mato Grosso do Sul (IFMS - Ponta Porã). Awarded as outstanding student by SBC (Brazilian Computer Society) in 2006. Provided consultancy in software projects in Brazil and abroad. Has 3 Java certifications, OCJA, OCPJP and OCPWCD. Has experience in Computer Science, with emphasis on Pattern Recognition, Machine Learning, Semantic Web and Information Retrieval.

Swarm Robotics: comportamento Adaptativo Aplicado ao problema de dispersão de pássaros em áreas agrícolas

A. P. Silva, R. I. S. Filho and F. A. Rodrigues

***Resumo*— Swarm Robotics é o estudo dos comportamentos coletivos complexos que emergem das inúmeras interações entre dois ou mais robôs integrantes do exame. O objetivo principal desse trabalho é a formulação de um método que empregue um grupo de robôs móveis para monitoramento de uma área agrícola. Quando violado o acesso desse local por pássaros, os robôs executam procedimentos para dispersá-los da região. Para atingir essa meta são utilizados os principais conceitos e propriedades referentes a Swarm Robotic. Acredita-se que no futuro o emprego dessa proposta na agricultura poderá minimizar os danos provocados pelos pássaros à vegetação cultivada.**

***Keywords*— Swarm Robotics, Adaptatividade, Auto- organização, Agricultura.**

I. INTRODUÇÃO

O prejuízo provocado por pássaros para a produção hortícola na Austrália é estimado em quase US$ 300 milhões por ano. Nesse país existem aproximadamente 60 espécies de pássaros que são conhecidas por danificar culturas agrícola. Essas espécies possuem diferenças marcantes em estratégias de alimentação e padrões de movimentos que influenciam a natureza, a época e a gravidade dos danos que causam. [4]

O crescente desmatamento provocado pelo ser humano, em busca de moradia ou para expansão agrícola tem causado impactos nos *habitats* dos pássaros, resultando na migração de algumas espécies para regiões diferentes em busca de alimentos (brotos, frutos e sementes) e abrigo. Essa situação acarreta uma interação mais frequente dos pássaros com as áreas de cultivo. [5] [6]

Entre as espécies que causam prejuízo na agricultura podemos citar alguns exemplos:

- No Brasil, o Anu, da espécie *(Agelaius ruficapillus),* traz danos à cultura do arroz irrigado no Rio Grande do Sul. Ele arranca as plântulas na fase inicial da cultura, o que pode reduzir aproximadamente em 25 a 60 por cento a população de plantas. Lavouras de arroz pré-germinado são as preferidas dessa espécie de pássaro, principalmente as mais próximas dos bosques. Eles também atacam durante a fase de maturação dos grãos, podendo causar perdas de produtividade superiores a 1250 kg por hectare [7].

- Os pássaros da família *Scrup (Aphelocoma coerulescens)* são encontrados no oeste dos Estados Unidos, partes do México, e na região central da Flórida. Eles se alimentam de nozes, figos, uvas, grãos, ervilhas, milho, bagas, cerejas, ameixas e peras, podendo trazer sérias depredações a essas culturas; [8]

- As aves *Magpies* da família *Corvidae* são encontradas em todo o hemisfério norte e podem causar danos substanciais para culturas de amêndoas, milho, nozes, melão, uvas, pêssegos, trigo e figos. [9]

Várias são as formas utilizadas pelos agricultores para minimizar os prejuízos decorrentes da presença de pássaros nas lavouras. Tais formas incluem desde a permissão do abate, obtida junto às autoridades responsáveis, bem como o emprego de diversas técnicas para dispersá-los, entre elas: por meio do som [8][9], pelo emprego da luz [9][12][13], por meio de predadores [14], por meio de modificação do *habitat* [1][8], pelo emprego do sistema de exclusão da compensação [4], pelo emprego de repelentes químicos [8] [10], pelo uso de avicidas [8] [10] e outras soluções[7].

Entretanto, verifica-se que a maioria dessas técnicas possui algum tipo de limitação, quer seja para sua implantação quer seja para sua manutenção. Dentro desse contexto, é necessário estudar novas técnicas para minimizar os danos provocados pela ação dos pássaros.

Entre os vários fatores que possibilitaram a ênfase na pesquisa e desenvolvimento de robôs pequenos e de baixo custo pode-se citar: o aumento do poder de computação dos microcontroladores, os avanços na miniaturização de sensores e os resultados positivos observados a partir das interações de times de robôs [16] em diversos contextos: em ambientes hostis (locais contaminados ou com temperaturas extremas), na monitoração de ambientes (localização de alvos, detecção de intrusões, focos de incêndio, etc.) e em operações de busca e salvamento.

Nesse contexto, cogitou-se a aplicação de times de robôs na dispersão de pássaros. Entretanto, o comportamento altamente dinâmico da ação dos pássaros pedia uma característica importante para a solução: a adaptatividade.

O objetivo principal desse trabalho é a avaliar se o uso de um grupo de robôs móveis para o monitoramento de uma área agrícola pode ser uma nova abordagem a ser empregada para dispersão de pássaros em áreas agrícolas.

A componente adaptativa encontrou o seu modelo na área de estudo da *Swarm Robotics*, que acabou se tornando a abordagem escolhida para a criação da solução de dispersão de pássaros empregada neste trabalho.

As próximas seções deste artigo estarão organizadas como se segue. Na seção 2 são abordadas as técnicas tradicionais para dispersão de pássaros. Na seção 3 são apresentados os conceitos básicos da *Swarm Robotics*, suas características e

A. P. da Silva, Universidade Federal de Mato Grosso do Sul (UFMS), Campo Grande, Mato Grosso do Sul, Brasil, alex.procopio@hotmail.com

R. I. S. Filho, Universidade Federal de Mato Grosso do Sul (UFMS), Ponta Porã, Mato Grosso do Sul, Brasil, reginaldo.uspoli@gmail.com

F. A. Rodrigues, Universidade Federal de Mato Grosso do Sul (UFMS), Ponta Porã, Mato Grosso do Sul, Brasil, fabricio.poliusp@gmail.com

propriedades. A seção 4 apresenta o experimento realizado e a seção 5 comenta os resultados obtidos. Por fim, a seção 6 trata da conclusão.

II. Técnicas tradicionais para dispersão de pássaros

Nessa seção serão abordadas as técnicas de dispersão, ou **deslocalização**, mais usuais utilizadas pelos agricultores para reduzirem os danos provocados por espécies de pássaros.

Embora os pássaros sejam fundamentais para os ecossistemas nos quais eles estão inseridos, sua concentração em áreas agrícolas pode afetar negativamente as culturas, quer seja pela contaminação, ou pelo consumo dos grãos ou frutos.

Em 2007, John Tracey, Quentim Hart, Glen Saunders e Ron Sinclair apresentaram um estudo [4] envolvendo aproximadamente 60 espécies de pássaros existentes na Austrália no qual abordam o comportamento de pássaros. Nesse trabalho é mencionado que, quando um pássaro é exposto a um evento inesperado ou desconhecido, uma de suas principais reações é a de voar. Essa pesquisa também alega que, devido à curiosidade do pássaro, cada vez que esse evento ocorre, o pássaro busca reunir informações sobre o estímulo que o assusta, acumulando informações suficientes para saber se o estímulo representa uma ameaça real ou não. Caso tal estímulo não represente uma ameaça, o pássaro se habitua a ele, passando a ignorá-lo.

A partir desses levantamentos os autores [4] apontam que o tempo necessário para o pássaro se habituar pode variar, dependendo de um conjunto de fatores, incluindo a espécie, o habitat circundante, a regularidade e o tipo de ruído. A habituação é o fator que mais limita a eficácia das técnicas empregadas para assustar os pássaros.

Como resultado das pesquisas realizadas na literatura, foram listados oito grandes grupos de abordagens utilizadas para a manipulação de concentrações de pássaros: manipulação por meio do som, pelo emprego da luz, por meio de predadores, por meio de modificação do habitat, pelo emprego do sistema de exclusão de compensação, pelo emprego de repelentes químicos, pelo uso de avicidas e outras soluções. Cada um desses grupos será apresentado a seguir.

A. Por meio de som e luz

Essa abordagem consiste em assustar o pássaro pelo uso de dispositivos acústicos. Entre esses dispositivos, podem ser utilizados equipamentos eletrônicos que reproduzam os sons emitidos por um pássaro quando ele avista um predador ou algo que lhe ameace, ou mesmo o som de um pássaro predador. Os equipamentos eletrônicos também podem reproduzir sons não biológicos, como ruídos eletrônicos [10]. Para evitar que os pássaros se habituem ao som, os dispositivos eletrônicos não devem emiti-lo do mesmo ponto de origem, com o mesmo curto período de tempo e na mesma intensidade (volume).

O uso de dispositivos explosivos agrícolas [1][9] também pode criar sons que assustem os pássaros, entretanto, por causa dos riscos de incêndio associado a este dispositivo, medidas de prevenção devem ser adotadas. Outra solução envolve o uso de munição, as quais são disparadas por uma arma calibre 12. Embora seja mais acessível, é menos eficaz e envolve a questão da segurança em utilizá-la próxima às áreas urbanas. [11]

O emprego de luz, por outro lado, consiste em assustar os pássaros por meio do uso de dispositivos que emitem ou refletem a luz. Nesse contexto, foram constatados diversos objetos introduzidos no meio da cultura, pendurados na vegetação ou em hastes, para refletirem ondas de luz de forma aleatória pelo movimento do vento [13]. Também são utilizados equipamentos eletrônicos, que são afixados no meio da plantação, emitindo flashes de luz de alta luminosidade em intervalos de tempo. A principal limitação dessa abordagem consiste, novamente, no fato dos pássaros se habituarem muito rapidamente aos feixes luminosos.

B. Por meio de predadores

Essa técnica aborda o uso de predadores naturais, para sobrevoarem a vegetação, periodicamente, inibindo a presença de pássaros [14]. Entre as aves de rapinas empregadas, pode-se citar o falcão de coleira, o falcão peregrino e o gavião asa de telha, têm sido utilizados. Esses predadores, quando devidamente treinados e monitorados por especialistas, podem capturar os pássaros e trazê-los em suas garras, para serem presos em gaiolas e, posteriormente, soltos em áreas distantes. O problema dessa técnica consiste no alto custo do investimento, quando comparado com as outras técnicas, somado aos ferimentos ocasionados pela perseguição e captura realizada pelas aves de rapina e a dificuldade em levar os pássaros capturados para áreas distantes. [4]

Outra derivação consiste na utilização de bonecos denominados popularmente de espantalhos. Esses podem ter o formato de um ser humano ou de alguma ave de rapina [9]. Para maximar os resultados dessa abordagem, o espantalho deve ser o mais realista possível e posicionado em lugares com boa visibilidade. É interessante que ele tenha alguma mobilidade, para ajudar a evitar a habituação.

C. Por meio de modificações no habitat e pelo emprego de sistemas de exclusão da compensação

As modificações no habitat incluem atividades que podem torná-lo menos atraente para os pássaros. Por exemplo, efetuar o desbaste ou poda da vegetação pode desencorajar os pássaros a utilizá-la como poleiro [12]. Apesar de menos custosa a longo prazo essa abordagem, sua desvantagem é que não pode ser empregada em culturas nas quais não se cabe a realização de poda, como por exemplo: soja, arroz, milho, entre outras.

Uma alternativa é o sistema de exclusão de compensação, que consiste em cobrir toda a vegetação, usando redes que impeçam fisicamente os pássaros terem acesso à cultura. Essa é uma das formas mais eficazes de deslocalização, porém é geralmente usada apenas em culturas de alto valor, de modo a compensar o investimento na implantação e manutenção dos equipamentos usados para remover as redes de forma eficiente.

D. Pelo emprego de repelentes químicos e avicidas

Repelentes químicos são substâncias normalmente pulverizadas sobre as culturas para impedir a presença de pássaros. Devido ao seu sabor, cheiro e cor, faz com que as

frutas percam sua atratividade para os pássaros. O problema advém dos resíduos nos alimentos, tornando-os inadequados para o consumo humano [8]. Uma variação dessa técnica é a utilização de algumas substâncias químicas conhecidas como avicidas. Em alguns casos (e com a autorização do governo) essas substâncias são dispostas no meio da vegetação em forma de iscas e tem a função de eliminar os pássaros. Os principais dilemas dessa abordagem são exatamente seus impactos sobre as espécies que não são os alvos e os seus efeitos residuais na cadeia alimentar. [10]

E. Outras soluções

Além das soluções anteriormente citadas, verificou-se o uso de aeromodelos ou aviões controlados por operadores, utilizados para perseguirem os pássaros, combinando estímulos visuais e auditivos. [10]

Também foram encontradas orientações na literatura para evitar a implantação de lavouras próximas a banhados e bosques, uma vez que, em geral, esses lugares são *habitats* naturais de pássaros.

Apesar de tantas abordagens existentes para a dispersão de pássaros, verifica-se que, em geral, as práticas consideradas ecológicas precisam ser utilizadas de forma complementar a outras práticas, para que possam produzir melhores resultados. O objetivo de mesclar essas abordagens (luz, som, predadores, modificações no *habitat*, etc.) tem como objetivo impedir que os pássaros se acostumem com a técnica empregada de dispersão.

De uma maneira geral, percebe-se que todas as abordagens listadas se esgotam e têm sua limitação atrelada a dois fatores principais: o impacto negativo aos humanos e ao meio ambiente (como no caso da utilização de agentes químicos e dos desmatamentos) e a adaptação que os pássaros desenvolvem à sua utilização.

Assim, uma característica imprescindível em uma abordagem para dispersão de pássaros é a sua adaptatividade em relação ao comportamento dos animais. Pode-se afirmar mais: tal adaptatividade, presente na técnica de deslocalização, deve servir de resposta à adaptatividade presente no comportamento dos pássaros no processo de habituação. Em outras palavras, o problema da dispersão pede uma solução adaptativa pela sua própria natureza.

Foi pensando também na questão do impacto do meio ambiente que se procurou a abordagem oferecida pela *Swarm Robotics*. A ideia era a utilização de uma solução bio-inspirada (onde a adaptatividade é crucial) que minimizasse o impacto ao meio ambiente.

III. A origem do termo *Swarm Robotics*

A *Swarm Robotics* é a área de estudo que pesquisa como um número relativamente grande de robôs pode ser fisicamente incorporado, de forma que um comportamento coletivo possa emergir a partir das interações locais entre cada integrante e entre os integrantes e o meio ambiente [17].

O termo *Swarm* foi introduzido no contexto da Robótica por Gerardo Beni [23] em 1988. A origem desse termo vem da biologia, inspirado no comportamento de animais sociais, aparecendo na computação com diferentes nomes, entre eles: *Swarm [18], Swarming [19], Swarm Intelligence [20], Swarm Optimization [21], Swarm Engineering* e *Swarm Robotics [22]*. Apesar dessas diferentes nomenclaturas, a expressão recorrente na literatura na área de Robótica é *Swarm Robotics*, a qual se inspira na observação do comportamento de insetos sociais, entre os quais se encontram as formigas, cupins, vespas, abelhas, etc.

Em 1953, E. O. Wilson [24] descobriu que o comportamento da sociedade das formigas poderia ser explicado como um conjunto de padrões simples e fixos de ação, os quais seriam resultantes de estímulos provocados por substâncias químicas, chamadas de **feromônios**, produzidas pelas glândulas nas formigas. Os cupins também produzem essas substâncias e as utilizam para construir o cupinzal, depositando-as no solo em determinados locais, que determinam a construção, de forma descentralizada, dos pilares da cúpula da estrutura habitacional [25].

Outra grande parte da comunicação entre os insetos sociais é realizada indiretamente através de um comportamento que o entomologista Grassé, em 1959, intitulou de **estigmergia** [26]. Nesse contexto, uma mudança no ambiente proporciona uma sugestão que pode interferir no comportamento dos insetos. Em 1991, Deneubourg, junto a seu grupo de pesquisa, demonstrou que a formação de um cemitério de formigas é induzida por esse comportamento, sem nenhuma coordenação central, em função do ácido acético emanado dos corpos em decomposição das formigas mortas [27]. Eles observaram que algumas espécies de formigas, como a *Pheidole pallidula*, possuem a característica de empilharem corpos, resultando na limpeza do formigueiro. Inicialmente, as formigas formam vários grupos distintos de corpos, em várias pilhas. Nesses grupos são constantemente adicionados e removidos corpos, e com o tempo, apenas os grupos de corpos que cresceram mais rapidamente prevalecem.

A. *Swarm Robotics* e a adaptatividade

Pode-se perceber, até o momento, que a *Swarm Robotics* possui características interessantes: ela permite que seres com comportamentos individuais muito simples realizem, em grandes quantidades e juntos, atividades complexas. Mas como tal comportamento está relacionado com o conceito da adaptatividade ou mesmo da tecnologia adaptativa? Para tratar essa questão, devem-se observar os trabalhos de Mark M. Millonas e Erol Sahin, entre outros.

Millonas, em seus estudos sobre comportamentos coletivos de animais [28] também analisou algumas características que, ao longo do tempo, serviram como base para inspirar as pesquisas em *Swarm Robotics*. Uma das principais características listadas por ele é a **adaptatividade**, descrita como a capacidade de adequação às variações ambientais, de forma equilibrada, e mantendo a estabilidade do sistema. Tal capacidade de adequação é consequência de uma mudança comportamental no grupo de robôs e ocorrerá sempre que um novo investimento de energia for necessário. Retomando a inspiração biológica, tem-se, novamente, o exemplo dos cupins: quando há um evento catastrófico no seu *habitat*, os indivíduos do cupinzal iniciam a recolonização e a expansão da colônia para outras regiões. [29]

Já Sahin apresentou em suas pesquisas, entre outros assuntos, algumas propriedades funcionais que, de igual

modo, serviram de motivação para o estudo da *Swarm Robotics*; todas elas inspiradas na análise das características comportamentais de animais sociais [17]. Entre tais propriedades funcionais, encontra-se a flexibilidade e a estigmergia.

A **flexibilidade** é definida como uma consequência da **adaptatividade**, pois, através dela, o sistema tem a capacidade dinâmica de oferecer diferentes estratégias de coordenação, diante das mudanças de contexto no ambiente. Voltando ao exemplo das formigas, um formigueiro é capaz de responder imediatamente às novas condições ambientais [3]. Elas se adaptam para buscarem alimentos (uso de feromônios), para arrastarem presas para o ninho (recrutamento) e para ultrapassarem obstáculos (usando aglomerações, que servem como "pontes vivas").

Por fim, a **estigmergia** consiste em uma forma de comunicação indireta, onde um indivíduo muda o ambiente, forçando os demais a reagirem a essa mudança, através de uma interação com os estímulos gerados pelas mudanças no local, realizando, assim, uma forma de comunicação limitada mediante a adaptação ao meio. Essa propriedade reduz a sobrecarga de informações trocadas entre os robôs e, em geral, cada integrante do *swarm* se comunica apenas com os seus vizinhos mais próximos, obtendo apenas informações locais.

Portanto, agora é possível responder à pergunta: "Existe relação entre a *Swarm Robotics* e a Tecnologia Adaptativa?" A resposta é sim. Desde que a adaptatividade, em termos da Tecnologia Adaptativa, é definida como a capacidade que tem um sistema de, sem a interferência de qualquer agente externo, tomar a decisão de modificar seu próprio comportamento, em resposta ao seu histórico de operação e aos dados de entrada [34], pode-se, sem perda de generalidade, associar o termo "dados de entrada" com o conceito de estímulos e o histórico de operação pode ser encarado com a "memória" advinda dos comportamentos próprios e consequentes da estigmergia e da ação dos feromônios. Portanto, conclui-se que o *swarm* de robôs (e não os robôs individualmente) compõe um Sistema Adaptativo.

IV. Experimentos e Resultados Preliminares

Inicialmente a questão era determinar como um *grupo* de robôs poderia cobrir fisicamente a área agrícola, de modo que eles pudessem se mover próximos uns dos outros (em grupo), mas não tão perto a ponto de colidirem e nem tão longe a ponto de se dispersarem.

Como resposta para uma estratégia de controle dos robôs, verificou-se que uma possibilidade seria que eles se locomoverem formando um padrão de forma autônoma, o qual seria a estratégia de controle do ***swarm* de robôs**. Por padrão, entende-se um arranjo de objetos que determina um lugar geométrico, como por exemplo [2], topologias anel, estrela, malha, pirâmides, entre outras.

Nesse artigo defini-se ***swarm* de robôs** como um grupo de robôs com características físicas idênticas (homogêneos), com capacidade de locomoção terrestre, que realizam as mesmas simples tarefas, possuidores de uma comunicação limitada com seus vizinhos e sem conhecimento da extensão da área que irão monitorar.

A abordagem de dispor os robôs geograficamente com base em formação de padrões permite usá-los para monitorar área de diferentes tamanhos, possibilitando que os robôs se locomovam de forma coesa (agrupada). Carlo Pinciroli é autor de um método de *Swarm Robotics* baseado nesse princípio [32] [30]. Com um comportamento auto-organizado, tal método se baseia no potencial de *Lennard-Jones (LJ)*, que consiste na descrição de um campo artificial local que determina uma estratégia de controle dos robôs através da formação de arranjos locais entre robôs vizinhos.

Pensando na questão da deslocalização dos pássaros uma disposição desejada é o de cerco. Na natureza existem animais que caçam em bando cercando suas presas. Como exemplo desse comportamento, Josué Ribeiro [39] e sua equipe mencionam que as garças podem unir-se às cegonhas (*Jabiru mycteria* e *Mycteria americana*) e caçar ativamente em bandos cercando cardumes de peixes. Eles também descrevem que os indivíduos de *Phalacrocorax brasilianus* (biguá) pescam em bandos de até duzentos indivíduos, cercando as presas para obter maior sucesso na captura. O *gavião-de-asa-telha (Parabuteo unicinctus*) caça em bandos de até seis indivíduos, o que permite cercar e capturar presas maiores, como lebres e coelhos [40].

Essa estratégia de cerco é um padrão onde o pássaro tende a ocupar o centro de uma área delimitada pelos integrantes do *swarm*, mantendo, assim, a distância entre os robôs em torno de um valor desejado. Tal abordagem, denominada de *flocking* auto-organizado é um comportamento coletivo amplamente verificado na natureza. Neste trabalho, foi definido como aplicar uma força derivada de um potencial *Lennard-Jones* para movimentar os robôs integrantes de um *swarm*, obtendo-se, dessa forma, movimentos ordenados que permitam o deslocamento coeso de todos os integrantes na realização de uma tarefa comum.

As interações entre robôs vizinhos, baseada no potencial de *Lennard-Jones*, seguem as regras descritas abaixo:

(a) Se dois objetos estão demasiadamente próximos, irão estar sujeitos a uma força de repulsão, afastando-os um do outro.

(b) Se estiverem muito longe, eles estarão sujeitos a uma força de atração, empurrando-os para perto um do outro.

(c) Se eles estão na distância desejada, nenhuma força atuará sobre eles.

Nesse contexto, o *flocking* auto-organizado é um *swarm* composto por uma nuvem de robôs. Cada robô verifica a sua distância em relação aos seus vizinhos, através das leituras dos seus sensores, e todas essas distâncias determinam o valor do campo potencial virtual. Esse valor é derivado em uma força e empregado na atuação das rodas de cada robô (frente, trás, virar). Desta forma, a nuvem de robôs tende a ir para a energia mínima de configuração. Essa força permite que os robôs se movam em grupo para uma posição na qual a distância entre os robôs seja igual.

A equação que determina o cálculo do potencial de *Lennard-Jones* [38] é descrita abaixo:

$$V(\rho) = \varepsilon\left(\left(\frac{\delta}{\rho}\right)^{12} - 2\left(\frac{\delta}{\rho}\right)^{6}\right) \qquad (1)$$

Onde:

- ρ consiste na distância entre os robôs vizinhos.
- ε consiste na profundidade do potencial, que consiste no coeficiente que permite aumentar a força de repulsão em função da intensidade da atração exercida.
- δ consiste na média da distância entre o diâmetro de dois objetos, na qual a força é mínima.

Do potencial, deriva-se o cálculo da força, descrito abaixo:

$$F(\rho) = -\nabla V(\rho) = \frac{12}{\rho}\varepsilon\left(\left(\frac{\delta}{\rho}\right)^{12} - \left(\frac{\delta}{\rho}\right)^{6}\right) \quad (2)$$

Após a escolha da estratégia de controle dos robôs no espaço geográfico, o próximo passo era testá-la para verificar a sua viabilidade como solução. Para isso, optou-se pela realização de uma série de experimentos de simulação, como será abordado a seguir.

A. SIMULAÇÃO

A simulação representou a linha metodológica escolhida por uma série de motivos:

1. Um simulador torna possível testar ideias em um ambiente seguro, evitando possíveis danos aos robôs reais, que em geral, possuem um custo significativo; [31]
2. Uma experiência envolvendo um grande número de robôs pode ser muito cara, inviabilizando o projeto. Simulações têm um custo bem menor;
3. As simulações podem ser repetidas várias vezes, com todas as variações de parâmetros possíveis;
4. Permite a aplicação dos algoritmos de controle do *swarm* em um ambiente 3D.

As buscas se concentraram em simuladores gratuitos e *open-source*. A escolha recaiu no simulador ARGoS (*Autonomous Robot Go Swarming*) [32] que foi utilizado como ferramenta para as experiências de simulação. Esse simulador foi concebido e desenvolvido por Carlos Pincirol, do *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA).*

O ARGos é um simulador 3D *open source*, específico para simulações voltadas para a abordagem de *Swarm Robotics*. Ele funciona em Linux e Mac OSX e foi desenvolvido durante o projeto *Swarmanoid*, o qual tem por objetivo o estudo de ferramentas e estratégias de controle de *swarm* de robôs. O ARGoS suporta os robôs *swarmanoid* e *e-puck* [31] e o código de controle do robô é escrito na linguagem de programação orientada a Objeto C++, usando configurações através de arquivos XML.

O simulador foi instalado em um computador contendo o sistema operacional Ubuntu 64 bits, versão 12.04, processador Intel Core i7 Q740@1,73GHz e 8 Gb de memória. Ele apresentou um bom funcionamento e um bom desempenho, permitindo verificar a flexibilidade e a eficiência do simulador. A **eficiência** foi obtida por meio de uma arquitetura *multithread*, visando à otimização do uso da CPU. Nesse contexto, a eficiência refere-se a capacidade de executar experimentos envolvendo vários robôs no menor tempo possível. Isso permite que a simulação contenha um número elevado de robôs sem perder o desempenho. Uma característica marcante nesse simulador é que, no ambiente 3D simulado, o espaço da simulação pode ser dividido em subespaços, e cada um deles pode ser administrado por diferentes tipos de gerenciadores de leis físicas. [31]

Já a **flexibilidade** foi obtida por meio de uma arquitetura orientada a *plug-ins* (através dos quais é possível obter modelos de robôs, sensores, atuadores, visualizações e outros). Os usuários podem escolher quais *plug-ins* utilizar para cada aspecto da simulação e atribuir recursos apenas aonde interessa [33]. Essa característica permite que o usuário obtenha apenas os recursos necessários para a simulação.

B. DINÂMICA DA SIMULAÇÃO

O objetivo do experimento foi simular um *swarm* de robôs monitorando uma área e expulsando o pássaro invasor do local, segundo um modelo de controle por meio de formação de padrões. Nesse contexto, foi utilizado o método do potencial de *Lennard-Jones* para ser a estratégia de controle empregando o algoritmo *Flocking*.

Cada integrante do *swarm* de robôs é representado na simulação como um robô do tipo *foot-bot* [35]. Esse é um robô móvel com sensores e atuadores. O sensor de luz permite que o robô calcule sua distância em relação à outra fonte de luz presente no espaço [37]. Através do dispositivo de comunicação *range-and-bearing* [39] cada robô pode se comunicar com seus vizinhos. Os robôs possuem duas rodas com movimentos de direção e velocidades independentes.

A região agrícola que será monitorada na simulação é representada por uma arena quadrada, com área de 400 m^2.

Inicialmente os 14 robôs são dispostos em uma área quadrada (ponto inicial) de 1 m^2, dentro da arena, nas coordenadas $(x=4; y=4)$ a $(x=5; y=5)$, conforme imagem da Figura 1.

Cada robô possui um dispositivo de identificação, mais especificamente um *led*, de forma que todos os robôs que pertencem ao *swarm* possuem um *led* acesso na cor vermelha.

Na arena também são dispostos outros elementos que irão compor o cenário, entre eles vários cilindros estáticos que possuem massa de 0,5 kg e raio de 10 cm, que se encontram distribuídos em fileiras representando a vegetação.

Também é inserida uma fonte luminosa na cor amarela que representa o pássaro. Durante a execução da simulação a trajetória do pássaro será do ponto inicial $(x=3; y=17)$ até o ponto final $(x=17; y=6)$, conforme imagem da Figura1.

Ao iniciar a execução da simulação, os robôs são lançados aleatoriamente nas posições pertencentes da região do ponto inicial, representado na cor verde na imagem da Figura 1. Em seguida cada robô realiza a leitura de seus sensores para verificar se localizam a fonte de luz amarela, a sua distância em relação à fonte e a sua distância em relação a cada um dos seus vizinhos que possuem um *led* aceso na cor vermelha. Para isso, é estipulada uma determinada distância para limitar o número de vizinhos mais próximos que ele considerará para o cálculo. Com base nessas distâncias é iniciado o cálculo da força de interação de *Lennard-Jones* para todos os robôs. Esse cálculo permite a coesão na realização da tarefa. Dessa forma cada integrante do enxame de robôs que antes estavam muito

próximos um dos outros passam a se repelir e a se atrair se distanciando uns dos outros.

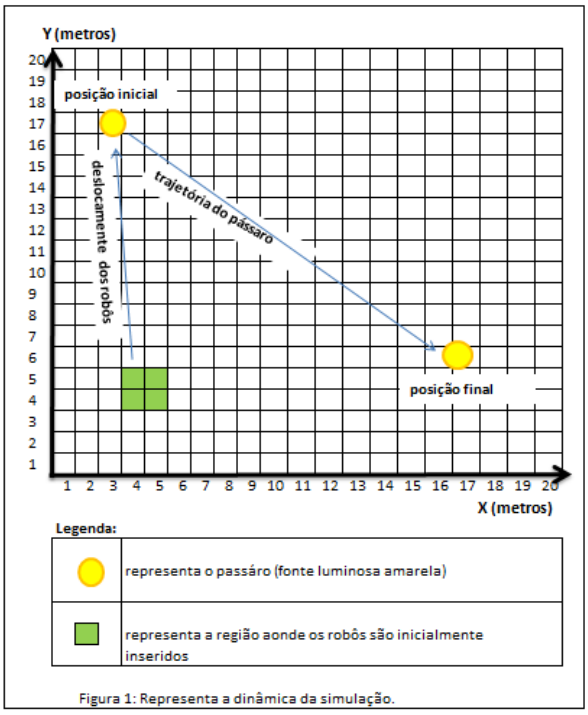


Figura 1: Representa a dinâmica da simulação.

Em seguida o *swarm* de robôs se desloca agrupado, em direção ao pássaro. A cada instante de tempo o pássaro vai se movimentando do seu ponto inicial até o seu ponto final e os robôs permanecem cercando-o. Durante toda a trajetória do pássaro ele permanece na altura constante de 1 metro.

Observa-se nessa simulação que o pássaro inicia o voo do ponto inicial até o ponto final. Abstraiu-se de que nessa trajetória o pássaro continuamente pousa na vegetação e os robôs procuram cercá-lo, no próximo instante o pássaro volta a voar pousando na próxima vegetação e os robôs sucessivamente tentam cercá-lo até retirá-lo da região agrícola.

O algoritmo *Flocking* utilizado para execução da simulação é descrito a seguir.

```
1 Algoritmo
2
3 (x1,y1)=VectorToLight(); //cálculo do ângulo da fonte de luz
4 (x2,y2)=FlockingVector(); //cálculo da forca resultante dos vizinhos
5 angulo = CalculeAngulo (x1+x2, y1+y2); //ângulo que o agrupamento se
deslocara
6 (vel1, vel2)= CalculaVelocidade(angulo); //cálculo da forca da velocidade
7 robot.wheels.set_velocity (vel1, vel2); //aplicação da velocidade nas rodas
8
9 fim algoritmo
```

As principais funções do algoritmo *Flocking* são descritas a seguir:

- ***VectorToLight()*:** tem por objetivo realizar o cálculo do ângulo do objeto, segundo as leituras dos sensores do robô.
- ***FlockingVector():*** realiza o cálculo da força resultante dos vizinhos.
- ***CalculeAngulo():*** calcula o ângulo que o agrupamento seguirá.
- ***CalculaVelocidade():*** recebe a direção do vetor que deve seguir e a transforma em uma sequência de atuação nas rodas, aplicando a velocidade adequada.

Durante a execução do algoritmo *Flocking*, cada robô integrante do *swarm* se locomove de tal forma que não fique próximo demais, a ponto de colidir com seu vizinho, nem tão longe a ponto de se dispersar e não contribuir para a formação padrão. Para viabilizar essa simulação, foram alterados os códigos fontes do algoritmo *Flocking* do simulador ARGoS, ao se criar o método *MyLoopFunctions* para ser invocado em intervalos de tempo previamente definidos, de modo a permitir a simulação da movimentação do pássaro.

É interessante destacar que cada robô possui 24 sensores de luz dispostos em forma circular. Cada sensor emite uma leitura que varia entre os valores 0 e 1. A cada execução da função *VectorToLight()* é realizada uma leitura dos valores do sensores de cada robô do *swarm* e convertidos esses valores em coordenadas cartesianas através da multiplicação dos valores do ângulo aonde o sensor está posicionado pelo valor da leitura do sensor. Ao resultado desse produto de cada robô, é calculado os cossenos e senos, obtendo-se as coordenadas X e Y, para que os robôs possam se deslocar de forma agrupada e coesa.

A velocidade de cada robô é calculada com base nos ângulos que representam a posição onde se encontra o pássaro e a posição que representa a coesão para a locomoção do *swarm*.

V. RESULTADOS

A Figura 2 ilustra o início da execução da simulação. Ela apresenta a arena na qual estão dispostos 14 robôs. Verifica-se que os robôs possuem um *led* acesso na cor vermelha indicando que pertencem ao mesmo *swarm* de robôs.

Também é possível verificar a vegetação representada pelos cilindros enfileirados na cor cinza. Percebe que inicialmente os robôs se encontram muito próximos um do outro sem nenhum padrão específico. Tal situação está ilustrada na imagem da Figura 2.

Dentro da arena e no meio dos cilindros (vegetação), é inserida uma esfera amarela que representa a fonte luminosa, a qual se desloca acima da superfície a uma altura de 1 metro. Essa esfera possui mobilidade e é identificada pelos sensores de luz dos robôs como uma fonte luminosa amarela. Tal esfera, em nossa simulação, representa o pássaro voando no meio da plantação.

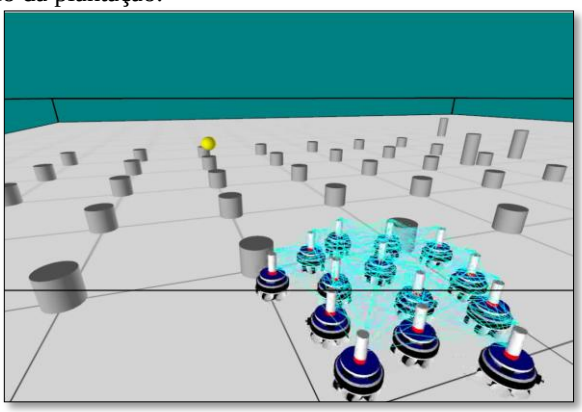


Figura 2: Início da simulação

Na Figura 3 percebe-se que cada robô realiza o cálculo da

força de interação de *Lennard-Jones* e os robôs que antes estavam muito próximos uns dos outros passam a se repelir e a se atrair gerando um padrão de formação.

A Figura 4 representa o momento em que o *swarm* detecta o pássaro e começa a se movimentar em sua direção, de forma coesa, para cercá-lo. Não importa se apenas um robô detecta o pássaro, ou mais de um consiga detectá-lo, pois, uma vez localizado pelo *swarm*, o pássaro passa a ser perseguido por ele. O *swarm* reage adaptativamente às mudanças de trajetória do pássaro mantendo a perseguição ao mesmo e evitando os cilindros em meio à plantação.

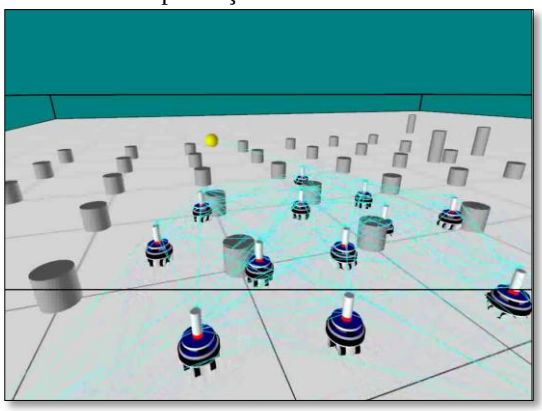


Figura 3: Enxame de robôs monitorando a área.

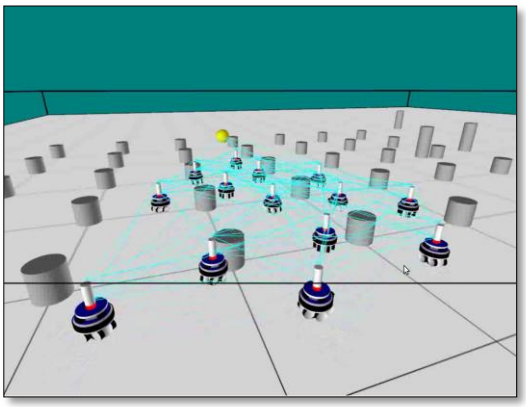


Figura 4: Robôs perseguindo o pássaro em movimento

As Figuras 5 e 6 ilustram os momentos finais da simulação, no instante em que os robôs geram um padrão de cerco em volta do pássaro na plantação. Percebe-se que tal padrão adquire a forma de um polígono, onde o pássaro ocupa o centro, exatamente como o desejado.

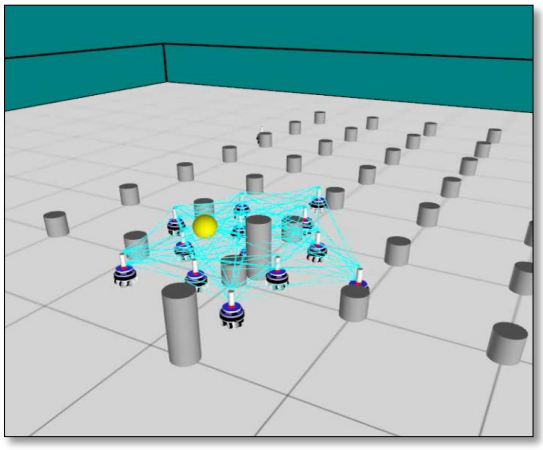


Figura 5: Robôs removendo o pássaro da vegetação.

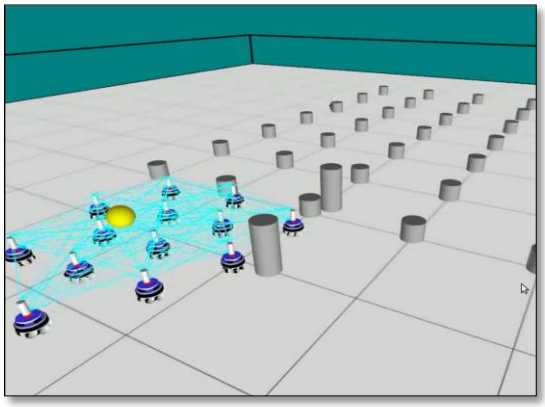


Figura 6: Finalização do processo de remoção do pássaro.

VI. CONCLUSÃO

Cabe recordar as indagações sobre as quais se construiu a pesquisa apresentada neste trabalho. A primeira questionava a possibilidade de obter um método ecológico e alternativo para a deslocalização dos pássaros em plantações. As simulações demonstraram que isso é possível. A nova abordagem não utiliza agentes químicos (repelentes ou avicidas) e não é necessária a realização de poda na plantação.

A próxima questão levantava o problema da habituação dos pássaros às técnicas tradicionais. Na ocasião, observou-se que uma nova técnica só poderia anular a adaptatividade dos pássaros se fosse, ela mesma, adaptativa.

Assim, chegou-se à abordagem da *Swarm Robotics* e, mais especificamente, à implementação da solução baseada em um padrão de cerco mediante o uso da técnica que lança mão da utilização do potencial de *Lennard-Jones*. Com base nos resultados obtidos por meio da simulação realizada, verificou-se que a aplicação do método do potencial de *Lennard-Jones* é uma boa estratégia para o controle de um *swarm* de robôs para o objetivo proposto. Ele permite que o grupo de robôs se mova em forma coesa, tal qual um grupo de insetos sociais faria, mantendo o alinhamento, a atração e a repulsão entre os integrantes do grupo.

Essa estratégia de controle possibilita o monitoramento da área por todos os integrantes do enxame simultaneamente, cercando o pássaro mesmo enquanto ele voa de um local para o outro, até dispersá-lo da região agrícola.

Como parte da sequência a ser dada à pesquisa e a esse trabalho, pode-se investigar uma série de outros caminhos interessantes. Um deles consiste em realizar a simulação com uma quantidade maior de pássaros e estudar as implicações ocasionadas por essa mudança.

Outro caminho consiste no uso de robôs reais e a análise do comportamento do *swarm* em um ambiente agrícola também real. Outro ponto para futuros trabalhos diz respeito ao uso de quadrotores (veículos aéreos impulsionados por quatro hélices) robóticos, pois ter robôs que voam tal quais os pássaros pode abrir todo um novo leque de possibilidades.

Existem estudos interessantes que também envolvem os aspectos teóricos. Uma análise que será realizada consiste na aplicação dos Autômatos Adaptativos na programação dos robôs que integram o *swarm* e estudar a composição do comportamento dos Autômatos Adaptativos com a abordagem da *Swarm Robotics*. Com um sistema hibrido que combinasse as duas abordagens, existe a possibilidade de um ganho significativo de desempenho para a solução apresentada neste trabalho.

REFERÊNCIAS

[1] BOOTH, T. W. Bird dispersal techniques. United States Department of Agriculture, 1994.

[2] MEDINA, A. J. R.; PULIDO, G. T.; RAMÍREZ-TORRES, J. G. A comparative study of neighborhood topologies for particle swarm optimizers. In: . c2009. p. 152-159.

[3] TRIANNI, V. Evolutionary swarm robotics: evolving self-organising behaviours in groups of autonomous robots. Springer, 2008. v. 108.

[4] TRACEY, J.; MARY, B.; HART, Q.; SAUNDERS, G.; SINCLAIR, R. Managing bird damage to fruit and other horticultural crops. Bureau of Rural Sciences, Canberra, 2007.

[5] NETO, J. R. S.; GOMES, D. M. Predação de milho por *arara-azul-de-lear anodorhynchus leari* (bonaparte, 1856) (aves: Psittacidae) em sua área de ocorrência no sertão da Bahia, 2007.

[6] JACINTO, J. C.; TOTI, T. P.; GUARITA, R. L.; MELO, C. Dano em um cultivo de sorgo (sorghum bicolor) causado por aves. In: . c2007.

[7] GOMES, A. D. S.; MAGALHÃES JÚNIOR, A. Arroz irrigado no sul do brasil. Embrapa Clima Temperado; Brasília: Embrapa Informação Tecnológica, 2004.

[8] CLARK, J. P.; HYGNSTROM, S. E. Horned larks. The Handbook Prevention and Control of Wildlife Damage, p. 64, 1994.

[9] HALL, T. C. Magpies. The Handbook: Prevention and Control of Wildlife Damage, 1994.

[10] DOLBEER, R. A. Blackbirds. Denver Wildlife Research Center, 1994.

[11] GODIN, A. J. Os pássaros nos aeroportos. O Manual: Prevenção e Controle da Vida Selvagem Danos, p. 56, 1994.

[12] JOHNSON, R. J. American crows. Denver Wildlife Research Center, 1994.

[13] MEDEIROS, A. R. M. Figueira do plantio ao processamento caseiro. EMBRAPA-Circular Técnica, número 35, 2002.

[14] OF AGRICULTURE OF GOVERNMENT OF AUSTRALIA, D. Best practice guidelines for bird scaring in orchards noise and threatened species. Government of Australia, 2009.

[15] ARAUJO, A.; PORTUGAL, D.; COUCEIRO, M.; ROCHA, R. Integrating arduino-based educational mobile robots in ROS. Proceedings of IEEE 13th International Conference on Autonomous Robot Systems and Competitions, p. 8-13, 2013.

[16] DOITSIDIS, L.; WEISS, S.; RENZAGLIA, A.; ACHTELIK, M. W.; KOSMATOPOULOS, E.; SIEGWART, R.; SCARAMUZZA, D. Optimal surveillance coverage for teams of micro aerial vehicles in GPS-denied environments using onboard vision. Journal Autonomous Robot, v. 33, p. 173-188, 2012.

[17] SAHIN, E. Swarm robotics - from sources of inspiration to domains of application. Technical report, Middle East Technical University, 2005.

[18] BENI, G. From swarm intelligence to swarm robotics. Workshop on Swarm Robotics. 8th International Conference on Simulation of Adaptive Behavior (SAB), 2004.

[19] TOPAZ, C. M.; BERTOZZI, A. L. Swarming patterns in a two-dimensional kinematic model for biological groups. SIAM Journal on Applied Mathematics, v. 65, n. 1, p. 152-174, 2004.

[20] BENI, G.; WANG, J. Swarm intelligence in cellular robotic systems. Robots and Biological Systems: Towards a New Bionics?, p. 703-712, 1993.

[21] COLORNI, A.; DORIGO, M.; MANIEZZO, V. Distributed optimization by ant colonies. In: c1991. p. 134-142.

[22] DORIGO, M.; GAMBARDELLA, L. M. Ant colony system: A cooperative learning approach to the traveling salesman problem. Evolutionary Computation, v. 1, n. 1, p. 53-66, 1997.

[23] BENI, G. The concept of cellular robotic systems. Proc. 3rd IEEE International Symposium on Intelligent Control, p. 57-62, 1988.

[24] HÖLLDOBLER, B.; WILSON, E. O. The ants. Harvard University Press, 1990.

[25] KENNEDY, J. Swarm intelligence. Springer, 2006.

[26] GRASSÉ, P.-P. La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement destermites constructeurs. Insectes sociaux, v. 6, n. 1, p. 41-80, 1959.

[27] DENEUBOURG, J.-L.; GOSS, S.; FRANKS, N.; SENDOVA-FRANKS, A.; DETRAIN, C.; CHRÉTIEN, L. The dynamics of collective sorting robot-like ants and ant-like robots. In: . c1991. p. 356-363.

[28] MILLONAS, M. M. Swarm, phase transitions, and collective intelligence. C.G. Langton, 1994.

[29] FILHO, J. C. M. Dissertação: Modelos computacionais para o processo de forrageamento e facilitação social em cupins. Universidade Federal de Viçosa, 2007.

[30] PINCIROLI, C.; BIRATTARI, M.; TUCI, E.; DORIGO, M.; DEL REY ZAPATERO, M.; VINKO, T.; IZZO, D. Self-organizing and scalable shape formation for a swarm of pico satellites. In: . c2008. p. 57-61.

[31] PINCIROLI, C.; TRIANNI, V.; OGRADY, R.; PINI, G.; BRUTSCHY, A.; BRAMBILLA, M.; MATHEWS, N.; FERRANTE, E.; CARO, G.; DUCATELLE, F.; STIRLING, T.; GUTIERREZ, A.; GAMBARDELLA, L. M.; DORIGO, M. ARGoS a modular, parallel, multi-engine simulator for multi-robotsystems. Swarm Intelligence, Berlin, Germany, v. 6, n. 4, p. 271-295, 2012.

[32] PINCIROLI, C.; TRIANNI, V.; OGRADY, R.; PINI, G.; BRUTSCHY, A.; BRAMBILLA, M.; MATHEWS, N.; FERRANTE, E.; CARO, G.; DUCATELLE, F.; STIRLING, T.; GUTIERREZ, A.; GAMBARDELLA, L. M.; DORIGO, M. Argos a modular, multi-engine simulator for heterogeneous swarm robotics. Los Alamitos, CA, p. 5027-5034, September 2011.

[33] BHAUMIK, A. Open source robotics: Multi-robot simulators. Linux For You, v. 10, n. 2, p. 48-50, 2012.

[34] NETO, J. J.; Um levantamento da evolução da adaptatividade e da tecnologia adaptativa. Revista IEEE América Latina, 2007, v. 5, n. 7.

[35] BONANI, M., LONGCHAMP, V., MAGNENAT, S., Rétornaz, P., BURNIER, D., ROULET, G., VAUssard, F., BLEULER, H., and MONDADA, F. (2010). The MarXbot, a Miniature Mobile Robot Opening new Perspectives for the Collective-robotic Research. In 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010), p. 4187-4193, Piscataway, NJ. IEEE Press.

[36] ROBERTS, J., STIRLING, T., ZUEREY, J., and FLOREANO, D. (2009). 2.5d infrared range and bearing system for collective robotics. In Papanikolopoulos, N., Sugano, S., Chiaverini, S., and Meng, M., editors, IEEE/RSJ International Conference on Intelligent Robots and Systems, Piscataway, NJ. IEEE Press.

[37] STRANIERI, A; TRIANNI, V.; FERRANTE, E.; DORIGO, M.; PINCIROLI, C. Self-Organized Flocking with a Heterogeneous Mobile Robot Swarm. Advances in Artificial Life, ECAL (2011): 789-796, MIT Press, Cambridge, MA.

[38] KITTEL, C. Introduction to Solid State Physics. Editora John Wiley. Inc. New York, 1976.

[39] NUNES, J. R. S.; OKI, YUMI; CARMIGNOTTO, A. P.; TELLO, P. G. Distribuição de frequência de habitats por aves aquáticas piscívoras do Lago Camaleão, Ilha da Marchantaria, Amazonas, Curso de Campo Ecologia da Floresta Amazônica, Brasil, 2002.

[40] Menq, W. Aves de rapina e suas técnicas de caça.Aves de Rapina BR - Águias, gaviões, falcões e corujas do Brasil. Brasil, 2013.

Alexsandro Procópio da Silva é graduado em Análise de Sistemas (2001) e Especialista em Engenharia de Web Sites (2004), ambos pela Universidade Federal de Mato Grosso do Sul (UFMS), Campo Grande, MS, Brasil. Mestrando em Computação Aplicada também pela UFMS desde 2013. Atualmente é Perito Criminal da Polícia Civil de MS (2009), lotado no Instituto de Criminalística de MS e suas pesquisas se concentram na área da Computação Forense e Robótica.

Reginaldo Inojosa da Silva Filho, possui graduação em Engenharia de Computação (em 2002), mestrado em Sistemas Eletrônicos (2006) e doutorado em Engenharia de Computação (2012), ambos pela Escola Politécnica da Universidade de São Paulo. Atualmente é Professor Adjunto da Universidade Federal de Mato Grosso do Sul, no Campus de Ponta Porã. Tem experiência na área de Ciência da Computação, com ênfase em Teoria da Computação. Atuando principalmente nos seguintes temas: Adaptatividade, Sistemas Complexos, Teoria Algoritmica da Informação e Teoria dos Autômatos.

Fabrício Augusto Rodrigues é Doutor em Ciências, com ênfase em Engenharia de Computação e Sistemas Digitais, pelo Programa de Pós-Graduação em Engenharia Elétrica da Escola Politécnica da Universidade de São Paulo (2012), Mestre em Informática pela Universidade Federal de Campina Grande (2002) e Bacharel em Sistemas de Informação pela Universidade Potiguar (1999). Atua na área da Ciência da Computação, cujas pesquisas estão relacionadas principalmente com as subáreas de Inteligência Artificial e Informática para Biodiversidade. Atualmente, é professor efetivo da Universidade Federal de Mato Grosso do Sul, no Campus de Ponta Porã. Dentre os seus interesses de pesquisa, destacam-se: Redes Neurais Artificiais, Algoritmos de Modelagem de Nicho Ecológico, Algoritmos Paralelos, Análise de Desempenho de Algoritmos, Tecnologia Adaptativa, Aprendizagem de Máquina, Atenção Visual e Visão Computacional.

Persistência em dispositivos adaptativos

P. R. M. Cereda e J. J. Neto

***Resumo*—Este artigo apresenta uma proposta inicial para a adição de um sistema de persistência em dispositivos adaptativos, tornando-os persistentes ao longo da própria execução ou de execuções subsequentes. Espera-se que tal funcionalidade possa contribuir para a extensão dos dispositivos adaptativos existentes, proporcionando-lhes a capacidade de preservar seus estados e de recuperá-los posteriormente.**

***Palavras-chave:*—Adaptatividade, persistência, dispositivos adaptativos**

I. Introdução

Persistência é o nome dado para o intervalo de disponibilidade do conteúdo de um determinado objeto [1], [2]. Tal característica de disponibilidade é primordial para a busca de soluções computacionais para os mais diversos problemas; informações coletadas e a própria evolução do modelo computacional ao longo do tempo necessitam de uma área de armazenamento para preservação do estado, sua atualização e eventual remoção [3].

A utilização da tecnologia adaptativa como alternativa para a resolução de problemas complexos tem apresentado resultados significativos [4]. Em [5], por exemplo, Cereda e José Neto introduzem uma proposta inicial do conceito de mineração adaptativa de dados como possível solução computacional aplicável aos problemas oriundos de grandes volumes de dados; entretanto, tais conceitos requerem, implicitamente, que existam subsídios para preservação dos modelos. A persistência é uma funcionalidade que pode contribuir para a extensão dos dispositivos adaptativos, conferindo-lhes a capacidade de preservar estados internos e recuperá-los conforme a necessidade.

Este artigo apresenta uma proposta inicial para a adição de persistência em dispositivos adaptativos, permitindo que estes tenham a capacidade de armazenar seu estado e recuperá-lo posteriormente. É importante mencionar que, no escopo deste artigo, a persistência é tratada sob um aspecto formal, sem, entretanto, entrar no mérito das técnicas de implementação. A área de armazenamento, neste caso, não é contemplada; admite-se que ela existe e que é transparente para o usuário.

Este artigo está organizado da seguinte forma: a Seção II introduz alguns conceitos iniciais para a formalização do conceito de persistência. A Seção III discute a persistência em dispositivos adaptativos propriamente dita, apresentando exemplos e discussões sobre a proposta. As considerações finais são apresentadas na Seção IV.

II. Conceitos iniciais

É necessário introduzir alguns conceitos iniciais para tratar da persistência em dispositivos adaptativos. Esta seção apresenta definições importantes para a compreensão, tratando de cada elemento componente do modelo proposto.

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

Definição 1 (Objeto)**.** Um *objeto* é qualquer elemento de armazenamento capaz de conter dados (*valores*) em um sistema. □

Um objeto é uma estrutura abstrata que contém dados. Esses dados, por sua vez, também podem ser outros objetos. Um objeto pode ter vários atributos, cada um deles armazenando um determinado dado. A estrutura

```
obj_pessoa {
   nome: Pedro
   idade: 30
}
```

é um exemplo de objeto contendo dois atributos, `nome` e `idade`, cada um deles contendo um valor associado. Cada sistema s_j possui um conjunto O_{s_j} contendo todos os objetos $o_1, \ldots, o_k$ componentes, $O_{s_j} = \{o_1, \ldots, o_k\}$, com $k \in \mathbb{N}$.

Definição 2 (Evento)**.** *Evento* é qualquer *ocorrência* que possa ser registrada ou identificada em um sistema. Por exemplo, são eventos a criação de um objeto, sua destruição, ou a alteração do seu conteúdo. □

De acordo com a Definição 2, um evento é sempre observável. Caso existam ocorrências que não interessam ao contexto do sistema, elas não serão registradas ou observadas e, portanto, não serão consideradas eventos. Seja E o conjunto de todos os eventos genéricos possíveis; existe uma relação de observação $\alpha\colon S \times E \mapsto \{0, 1\}$, no qual S é o conjunto de sistemas, retornando 0 caso o evento e_i não seja observável no sistema s_j, ou 1 caso contrário. Portanto, e_i é um evento de s_j se, e somente se, $\alpha(s_j, e_i) = 1$, e o conjunto de todos os eventos de s_j é definido como $E_{s_j} = \{e_i \mid \alpha(s_j, e_i) = 1\}$, $E_{s_j} \subseteq E$.

Definição 3 (Instante)**.** *Instante* é um número n arbitrário, $n \in \mathbb{N}$, ao qual se referem *eventos* ocorridos no sistema. □

A definição de instante permite que os eventos sejam ordenados de acordo com sua ocorrência.

Definição 4 (Caracterização de um evento)**.** Eventos caracterizam-se por sua *identificação*, pelo seu *tipo* e pelo *instante de ocorrência*. □

Cada evento ocorre em um instante, isto é, existe um número natural associado ao evento que determina uma relação de ordem. Um evento é caracterizado por um identificador unívoco (que só admite uma interpretação), pelo tipo de evento (por exemplo, uma alteração de um atributo de um objeto) e pelo instante de ocorrência. Assim, um evento $e_i \in E_{s_j}$ é uma tupla $e_i = (a, b, c)$, com a sendo um identificador, b o

tipo do evento, e $c \in \mathbb{N}$ o instante de ocorrência. Considere $\pi_n(x)$ como a projeção do n-ésimo elemento da sequência x. É importante observar que

$$\bigcap_{e_i \in E_{s_j}} \{\pi_1(e_i)\} = \emptyset$$

ou seja, os identificadores de todos os eventos do sistema s_j são unívocos – não podem existir dois ou mais eventos compartilhando do mesmo identificador.

Definição 5 (Eventos simultâneos)**.** Dois ou mais eventos que tenham o mesmo instante de ocorrência são chamados *eventos simultâneos*. □

Em um sistema s_j com seu conjunto de eventos E_{s_j}, se

$$\left| \bigcap_{e_i \in E_{s_j}} \{\pi_3(e_i)\} \right| > 0$$

existem pelo menos dois eventos que apresentam o mesmo instante de ocorrência. Portanto, o sistema apresenta eventos simultâneos.

Definição 6 (Intervalo)**.** *Intervalo* é um conjunto ordenado que compreende todos os instantes que ocorrem em um sistema, desde um instante *inicial* (*limite inferior*) até um instante *final* (*limite superior*). □

O intervalo do sistema s_j é denotado por $I_{s_j} = \{c_1, \ldots, c_l\}$, com $l \in \mathbb{N}$, sendo c_1 o limite inferior e c_l o limite superior. Existe uma relação de ordem que determine a posição de cada elemento.

Definição 7 (Caracterização de um objeto no instante)**.** O objeto se caracteriza em cada instante por sua *identificação* e pelo seu *conteúdo*. □

Um objeto $o_i \in O_{s_j}$ é representado no instante c como uma tupla $o_{i,c} = \{a, d\}$, com a sendo um identificador e d o conteúdo de o_i no instante c. Acerca da identificação de um objeto em um instante arbitrário,

$$\bigcap_{o_i \in O_{s_j}} \bigcap_{c \in I_{s_j}} \{\pi_1(o_{i,c})\} = \emptyset$$

ou seja, os identificadores de todos os objetos para cada instante no sistema s_j são unívocos – não podem existir objetos em instantes distintos compartilhando do mesmo identificador.

Definição 8 (Escopo de um objeto)**.** *Escopo* de um objeto é a parcela do programa que o contém na qual o conteúdo do objeto é de alguma forma visível (*acessível, ao menos, para consulta*). □

Seja um objeto $o_i \in O_{s_j}$, seu escopo Γ_{o_i} é definido como $\Gamma_{o_i} \colon I_{s_j} \mapsto \{0, 1\}$, com I_{s_j} sendo o intervalo do sistema s_j, retornando 0 caso o objeto não esteja disponível em um determinado instante, ou 1 caso contrário.

Definição 9 (Disponibilidade de conteúdo de um objeto)**.** O *conteúdo* de um objeto permanece *disponível* para uso desde o instante de seu armazenamento no objeto até o do armazenamento de um novo conteúdo (devido a *efeitos colaterais* ocorridos durante a operação do dispositivo), ou então até que o objeto *deixe de existir*. □

Um objeto $o_i \in O_{s_j}$ terá seu conteúdo disponível enquanto $\exists\ o_{i,c} \mid \pi_2(o_{i,c}) \neq$ vazio, tal que c é o limite superior para o_i, e $\Gamma_{o_i}(c) = 1$.

Definição 10 (Conjunto de todos os instantes)**.** O conjunto de todos os instantes do sistema é o conjunto dos números correspondentes a todas as ocasiões de ocorrências de eventos. □

Definição 11 (Atribuição de instantes)**.** Embora arbitrários, os instantes são *sempre atribuídos em ordem crescente*, exceto quando ocorrem eventos simultâneos. □

Definição 12 (Indicador de precedência das ocorrências dos eventos)**.** Não se trata do conceito de tempo, nem contínuo nem discreto, apenas de um *indicador de precedência* das ocorrências dos eventos no sistema. □

Definição 13 (Intervalo de usabilidade)**.** *Intervalo de usabilidade* de um objeto é o intervalo no qual existe algum escopo que inclua o objeto em questão, logo é o conjunto de instantes em que é possível consultar seu conteúdo. □

O intervalo de usabilidade do objeto o_i pode ser definido como $U_{o_i} = \{c \in I_{s_j} \mid \Gamma_{o_i}(c) = 1\}$, ou seja, é o conjunto formado por cada instante no intervalo em que o escopo do objeto i esteja disponível.

Definição 14 (Usabilidade do conteúdo dos objetos)**.** Durante a execução de um programa, a *usabilidade do conteúdo* dos objetos nele existentes corresponde à *união dos intervalos de atividade dos escopos* aos quais os objetos pertencem. □

A usabilidade de conteúdo V pode ser definida como

$$V = \bigcup_{o_i \in O_{s_j}} U_{o_i}$$

ou seja, é a disponibilização de todos os escopos de todos os objetos no sistema s_j.

Definição 15 (Objeto de conteúdo persistente)**.** *Objeto de conteúdo persistente* é aquele cujo conteúdo possa ser memorizado em qualquer instante desejado, para que possa posteriormente ser recuperado para uso a critério do programa que controla o uso do objeto. □

Definição 16 (Sistema de persistência)**.** Um *sistema de persistência* é um espaço de armazenamento que pode memorizar os dados contidos em objetos de conteúdo persistente, disponibilizando-o para que possam ser resgatados em qualquer instante pelo programa que controla o uso do objeto. □

Definição 17 (Conteúdo de um objeto persistente em um intervalo)**.** O conteúdo de um objeto é *persistente em um intervalo* se nesse intervalo for possível acessá-lo, ou seja, se tal conteúdo fizer parte de algum escopo ativo naquela ocasião. □

Em outras palavras, considerando um objeto $o_i \in O_{s_j}$, ele será persistente em um intervalo se seu conteúdo estiver ativo.

Definição 18 (Conteúdo ativo no intervalo de usabilidade)**.** Se uma parte do intervalo de usabilidade do conteúdo de um objeto *não pertencer ao intervalo de execução do programa* ao que ele pertence, diz-se que o conteúdo está ativo nesse intervalo. □

Sejam U_{o_i} o intervalo de usabilidade do objeto $o_i \in O_{s_j}$ e P um intervalo de execução do programa; se $|U_{o_i} \cap P| > 0$, o conteúdo de o_i está ativo no intervalo.

Definição 19 (Usabilidade de conteúdo para objetos não-persistentes)**.** Para programas que não tenham objetos com conteúdos persistentes, nenhum objeto apresentará usabilidade de conteúdo a não ser durante o intervalo de execução do programa. □

Analogamente, sejam U_{o_i} o intervalo de usabilidade do objeto $o_i \in O_{s_j}$ e P um intervalo de execução do programa; se $U_{o_i} \cup P \subseteq P$, o conteúdo de o_i só estará disponível durante o intervalo de execução do programa.

Duas operações básicas de persistência são necessárias para complementar as definições apresentadas até então: *salvar*, responsável por persistir o valor de um objeto $o_i \in O_{s_j}$ em um determinado instante c no sistema de persistência, e *restaurar*, que recupera um valor específico de um objeto $o_i \in O_{s_j}$, anteriormente persistido. As operações serão detalhadas a seguir.

Definição 20 (Operação de salvamento)**.** Seja $P_\downarrow$ uma operação básica de persistência, definida como $P_\downarrow \colon O_{s_j} \times I_{s_j} \mapsto \mathbb{N}$; $P_\downarrow$ toma um objeto $o_i \in O_{s_j}$ e um instante $c \in I_{s_j}$, ambos pertencentes ao sistema s_j, persistindo o valor de o_i no instante c no sistema de persistência, e retornando um valor inteiro que representa um identificador unívoco. □

É importante observar que a operação de salvamento gera um evento $e_k = (u, v, w)$, onde $u = P_\downarrow(o_i, c)$ (identificador), $v \equiv P_\downarrow(o_i, c)$ (tipo de evento, neste caso, salvamento de o_i no instante c), e $w \in I_{s_j}$ (instante do evento, pode ser diferente de c).

O salvamento de o_i no instante c é armazenado pelo sistema de persistência em uma estrutura de lista ligada, disponível para cada objeto no sistema (Figura 1). Cada elemento da lista de o_i contém uma operação de persistência sobre esse objeto, consistindo no objeto, no instante e no identificador unívoco gerado. Os elementos da lista estão encadeados através de referências de endereço, no qual cada elemento aponta para aquele inserido na sequência (no exemplo da Figura 1, as ligações são representadas por setas), ou nulo se este é o último elemento (na Figura 1, representado por •). O identificador gerado a partir da operação de salvamento será utilizado como rótulo de seu elemento correspondente.

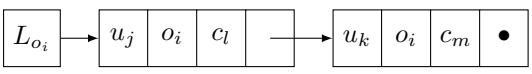


Figura 1. Estrutura de lista ligada para o objeto o_i. No exemplo, foram realizadas duas operações de salvamento, gerando os identificadores u_j e u_k.

A cada operação de salvamento do objeto o_i, os elementos anteriores na lista não são removidos, criando-se, portanto, um histórico de persistência para o_i, desde a primeira operação até a última. Todos os objetos são contemplados no sistema de persistência, e $\forall\ o_i \in O_{s_j}, \exists\ L_{o_i}$, ou seja, existe sempre uma lista associada a cada objeto.

Definição 21 (Operação de restauração)**.** Seja $P_\uparrow$ uma operação básica de persistência, definida como $P_\uparrow \colon O_{s_j} \times I_{s_j} \mapsto O_{s_j} \cup \{\epsilon\}$; $P_\uparrow$ toma um objeto $o_i \in O_{s_j}$ e um instante $c \in I_{s_j}$, ambos pertencentes ao sistema s_j, realiza a busca de o_i no instante c no sistema de persistência, e retornando o objeto o_i persistido no instante c, ou ϵ caso não exista ocorrência de persistência com tais índices. □

É importante observar que a operação de restauração gera um evento $e_k = (u, v, w)$, onde u é um identificador unívoco, $v \equiv P_\uparrow(o_i, c)$ (tipo de evento, neste caso, restauração de o_i no instante c), e $w \in I_{s_j}$ (instante do evento, pode ser diferente de c).

Além disso, a cada operação de restauração, o objeto o_i é novamente persistido, de modo que uma cópia de o_i no novo instante $c' \in I_{s_j}$ é colocada na cauda da lista L_{o_i}. Em outras palavras, $P_\uparrow(o_i, c) = o_i \wedge P_\downarrow(o_i, c')$.

É possível também definir $P_\uparrow$ como $P_\uparrow \colon \mathbb{N} \mapsto O_{s_j} \cup \{\epsilon\}$, tomando um número natural representando o identificador unívoco previamente gerado pela operação de salvamento; a nova operação realiza a busca de o_i cujo indicador é um número n na lista L_{o_i} do sistema de persistência, e retorna um objeto o_i do elemento com rótulo n, ou ϵ caso não exista ocorrência de persistência com tal índice.

Do mesmo modo, a cada operação de restauração, o objeto o_i é novamente persistido, de modo que uma cópia de o_i no novo instante $c' \in I_{s_j}$ é colocada na cauda da lista L_{o_i}.

As operações básicas do sistema de persistência são atômicas sobre um mesmo objeto o_i, ou seja, se dois ou mais eventos simultâneos sobre o_i têm como tipo de evento $P_\downarrow$ ou $P_\uparrow$, apenas *uma operação* será realizada, descartando as demais. Caso todas as operações sejam de restauração, apenas uma delas (não é possível predizer qual delas será) será efetivamente considerada; caso exista ao menos uma operação de salvamento, ela sempre terá prioridade sobre a operação de restauração mas, do mesmo modo, apenas uma delas será considerada.

III. Persistência em dispositivos adaptativos

Após a definição dos conceitos iniciais, já é possível tratar da persistência em dispositivos propriamente dita. O valor a ser persistido, neste caso, é a configuração corrente do dispositivo e o estímulo de entrada. Uma possível interpretação do sistema de persistência é através do conceito de camada, no qual o dispositivo está contido em uma camada de persistência, podendo fazer uso dela ou não (Figura 2). Mesmo no caso do dispositivo não utilizar persistência, o objeto continua ativo (Definição 19).

Uma vez que a característica de persistência é agregada a um determinado dispositivo, conferindo a característica de ser persistente, é necessário definir a interface entre eles. Para que a persistência ocorra de modo transparente, espera-se que o dispositivo, na aplicação de alguma regra, execute uma ação

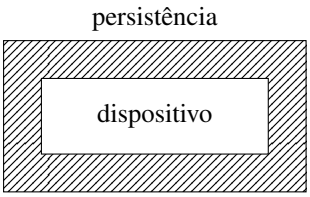


Figura 2. Interpretação de persistência como uma camada envolvendo o dispositivo. A camada em questão representa o sistema de persistência propriamente dito (Definição 16).

semântica que indique à camada de persistência qual operação realizar (Definição 22).

Definição 22 (Ação semântica de persistência)**.** Uma *ação semântica de persistência* é uma chamada a operação básica presente no sistema de persistência, de acordo com a definição explícita de um determinado dispositivo d que a dispara. □

Como exemplo, considere um autômato finito determinístico $M = (K, \Sigma, \delta, s, F)$, onde K é um conjunto finito de estados, Σ é um alfabeto, δ é uma função de transição, $\delta\colon K \times \Sigma \mapsto K$, $s \in K$ é o estado inicial e $F \subseteq K$ é o conjunto de estados finais [6]. A Tabela I contém o mapeamento δ e apresenta uma ação semântica da operação de salvamento $P_{\downarrow}$ que indica ao sistema de persistência para salvar a configuração do dispositivo e a cadeia w sendo consumida, no instante corrente. Admita a função $\rho(M, w)$ que codifica a configuração corrente de M e a cadeia w consumida até então em um valor para o objeto correspondente ao autômato.

Tabela I
EXEMPLO DE AÇÃO SEMÂNTICA ASSOCIADA AO MAPEAMENTO.

$q \in K$	$\sigma \in \Sigma$	$\delta(q, \sigma)$	Ação semântica
q_0	a	q_0	–
q_0	b	q_1	$P_{\downarrow}(\rho(M, w), 1)$
q_1	a	q_1	–
q_1	b	q_0	$P_{\downarrow}(\rho(M, w), 2)$

Na Tabela I, é possível verificar que o sistema de persistência é acionado para salvar a configuração de M e a cadeia w, consumida até então, a cada ocorrência do símbolo $b \in \Sigma$, em um determinado instante. A ação semântica $P_{\downarrow}$ (correspondente à operação de salvamento do sistema de persistência) não faz parte da definição formal do dispositivo, mas é associada às suas regras.

O exemplo apresentado apenas faz a persistência da configuração corrente de M e a cadeia de entrada consumida até então, mas não realiza a operação de restauração. Observe que o sistema de persistência possui uma lista L_M que armazenará o conteúdo persistido. É possível utilizar essa lista como forma de *backtracking*, caso o dispositivo necessite restaurar uma configuração anterior. Como novo exemplo, considere o autômato finito da Figura 3.

A linguagem reconhecida pelo autômato M da Figura 3 é $L(M) = \{w \in \{a, b\}^* \mid w = aa(ba)^*\}$. A primeira transição, $(q_0, a) \rightarrow q_1$, consome a de w e realiza a ação semântica chamando a operação de salvamento; a segunda transição, $(q_1, a) \rightarrow q_2$ consome o segundo a e chega ao estado final q_2; existe um laço em q_2 que consome b e

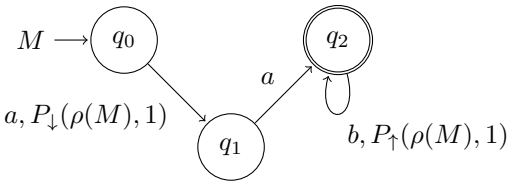


Figura 3. Autômato finito M utilizando a camada de persistência.

restaura o conteúdo persistido no instante 1; observe que, deliberadamente, a cadeia w, consumida até então, não foi persistida. Ao realizar a operação de restauração, o autômato M é restaurado para a configuração do instante 1, isto é, M retorna ao estado q_1 esperando, então, mais um a para alcançar o estado final. A Tabela II ilustra o consumo da cadeia $aababa$ por M.

Tabela II
CONSUMO DA CADEIA $aababa$ POR M.

Estado corrente	Símbolo corrente	Estado de destino	Ação semântica	Restante da cadeia
q_0	a	q_1	$P_{\downarrow}(\rho(M), 1)$	$ababa$
q_1	a	q_2	–	$baba$
q_2	b	q_2	$P_{\uparrow}(\rho(M), 1)$	aba
q_1	a	q_2	–	ba
q_2	b	q_2	$P_{\uparrow}(\rho(M), 1)$	a
q_1	a	q_2	–	–

Observe que o exemplo da Figura 3 não apresenta benefícios em relação ao mapeamento tradicional, sem a utilização de ações semânticas de persistência – a remoção do laço em q_2 e a adição de uma transição consumindo b de $q_2 \rightarrow q_1$ seriam suficientes. Entretanto, é interessante notar a possibilidade de restaurar configurações e estímulos de entrada conforme a necessidade; no caso da utilização de dispositivos adaptativos, pode partir-se de configurações já existentes, oriundas de execuções anteriores, além da possibilidade de obter um *snapshot* da configuração corrente.

Considere agora o exemplo do autômato adaptativo M', introduzido em [7], com a adição do sistema de persistência. A linguagem aceita por M' foi adicionada de marcadores de início e fim para facilitar as operações de salvamento e restauração; assim, $L(M) = \{w \in \{a, b, c, \$\} \mid w = \$\$\$^* a^n b^n c^n \$\}$, ou seja, uma cadeia com um número n de a's, seguido pelo mesmo número de b's, seguido pelo mesmo número de c's. O autômato adaptativo M' é apresentado na Figura 4. A função adaptativa f é apresentada na Figura 5.

De modo simplificado, o autômato M' foi remodelado para reconhecer cadeias na forma $a^n b^n c^n$, com n, no mínimo, igual à sua maior ocorrência. Isto é, se M' reconheceu $aaabbbccc$, com $n = 3$, ele não será mais capaz de reconhecer cadeias com $n < 3$, apenas com $n \geq 3$.

Como exemplo, considere a submissão da cadeia $w = \$\$aabbcc\$$ para reconhecimento pelo autômato adaptativo M'. Observe que, de acordo com a Figura 4, deliberadamente, a cadeia w, consumida até então, não é persistida; apenas a configuração do dispositivo é utilizada. A Tabela III ilustra o consumo da cadeia $\$\$aabbcc\$$ por M'.

O autômato M' resultante após o reconhecimento de $\$\$aabbcc\$$ foi salvo após o consumo do marcador de final.

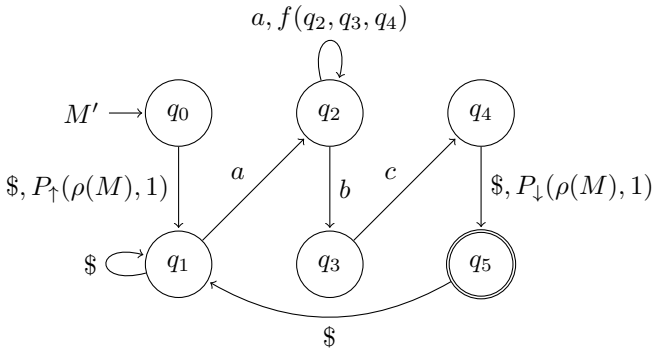


Figura 4. Autômato adaptativo M'.

Função adaptativa

```
f(p1 , p2, p3) = {
  ?x, ?y, ?z, g1*, g2*, g3* :
  ?[(?x, a) -> (p1)]
  -[(?x, a) -> (p1)]
  ?[(?y, b) -> (p2)]
  -[(?y, b) -> (p2)]
  ?[(?z, c) -> (p3)]
  -[(?z, c) -> (p3)]
  -[(q2, a) -> (q2), f(p1, p2, p3)]
  +[(?x, a) -> (g1*)]
  +[(g1*, a) -> (p1)]
  +[(?y, b) -> (g2*)]
  +[(g2*, b) -> (p2)]
  +[(?z, c) -> (g3*)]
  +[(g3*, c) -> (p3)]
  +[(q2, a) -> (q2), f(g1 , g2, g3)]
}
```

Figura 5. Função adaptativa f do autômato M'.

É importante mencionar que a restauração de M' na transição de $q_0 \rightarrow q_1$ falha na primeira vez em que M' é utilizado, pois a lista L_M está vazia, e $P_\uparrow(\rho(M), 1) = \epsilon$; entretanto, o erro na restauração não impede o que o reconhecimento de w prossiga. É possível tratar o erro adicionando ações semânticas complementares para a avaliação das operações básicas, além de estender o próprio sistema de persistência.

Apesar da sequência de consumo apresentada na Tabela III ilustrar a criação de novos estados e alteração e inclusão de transições para reconhecer b's e c's adicionais, existe uma ação adaptativa adicional no corpo da função adaptativa f (Figura 5): como o autômato adaptativo M' está sendo remodelado para oferecer a possibilidade de reuso, é necessário adicionar um estado adicional e rearranjar transições para o reconhecimento de a's adicionais, tal qual acontece com b's e c's. Observe que a adição de persistência em um dispositivo pode requerer uma análise mais profunda sobre a manifestação do fenômeno da adaptatividade.

Tabela III
CONSUMO DA CADEIA $\$\$aabbcc\$$ POR M'.

Estado corrente	Símbolo corrente	Estado de destino	Ação semântica	Restante da cadeia
q_0	$\$$	q_1	$P_\uparrow(\rho(M), 1)$	$\$aabbcc\$$
q_1	$\$$	q_1	–	$aabbcc\$$
q_1	a	q_2	–	$abbcc\$$
q_2	a	$q_2, f(\ldots)$	–	$bbcc\$$
q_2	b	q_7	–	$bcc\$$
q_7	b	q_3	–	$cc\$$
q_3	c	q_8	–	$c\$$
q_8	c	q_4	–	$\$$
q_4	$\$$	q_5	$P_\downarrow(\rho(M), 1)$	–

Ao submeter uma nova cadeia w' ao autômato M', ele não a reconhecerá caso esta tenha $n < 2$, dado que a operação de salvamento foi executada quando M' estava preparado para reconhecer cadeias com $n = 2$. A função adaptativa f não foi alterada e está disponível, portanto M' ainda é capaz de reconhecer cadeias com $n \geq 2$. Considere agora a submissão de uma nova cadeia $w' = \$\$aaabbbccc\$$ para reconhecimento pelo autômato adaptativo M'. A Tabela IV ilustra o consumo de w'.

Tabela IV
CONSUMO DA CADEIA $\$\$aaabbbccc\$$ POR M'.

Estado corrente	Símbolo corrente	Estado de destino	Ação semântica	Restante da cadeia
q_0	$\$$	q_1	$P_\uparrow(\rho(M), 1)$	$\$aaabbbccc\$$
q_5	$\$$	q_1	–	$aaabbbccc\$$
q_1	a	q_6	–	$aabbbccc\$$
q_6	a	q_2	–	$abbbccc\$$
q_2	a	$q_2, f(\ldots)$	–	$bbbccc\$$
q_2	b	q_{10}	–	$bbccc\$$
q_{10}	b	q_7	–	$bccc\$$
q_7	b	q_3	–	$ccc\$$
q_3	c	q_{11}	–	$cc\$$
q_{11}	c	q_8	–	$c\$$
q_8	c	q_4	–	$\$$
q_4	$\$$	q_5	$P_\downarrow(\rho(M), 1)$	–

Ao consumir o primeiro símbolo, o marcador de início, M' é restaurado para o instante 1, e a execução passa a assumir q_5 como estado corrente, de acordo com a Tabela IV; desse ponto em diante, o autômato segue seu fluxo normal de execução até o consumo do marcador final, sendo persistido mais uma vez. Observe que, após o reconhecimento de w', M' passa a reconhecer $a^n b^n c^n$ tão somente com $n \geq 3$.

É interessante notar que, ao executar a ação semântica $P_\downarrow(\rho(M), 1)$, o sistema de persistência sobrescreve o valor antigo ao invés de criar um novo elemento na lista L_M; dessa forma, M' não tem histórico de persistência, apenas a última referência. É possível alterar M' de tal modo que as referências em instantes anteriores sejam mantidas, ora através de ações adaptativas alterando ações semânticas no mapeamento, ou por meio de extensões no sistema de persistência, criando, por exemplo, uma operação de salvamento estendida $P'_\downarrow$ que salva o objeto sempre no instante corrente, e uma operação de restauração estendida $P'_\uparrow$ que retorna o objeto com o identificador de maior valor (portanto, a referência mais recente).

IV. CONSIDERAÇÕES FINAIS

Este artigo apresentou uma proposta para a adição de um sistema de persistência em dispositivos adaptativos, tornando-os persistentes ao longo da própria execução ou de execuções subsequentes.

A característica de persistência para dispositivos adaptativos permite salvar e restaurar configurações através de ações semânticas; a interpretação do sistema de persistência através do conceito de camada confere uma independência de modelo ao dispositivo, ou seja, não é necessário alterar explicitamente a definição do dispositivo para torná-lo persistente, bastando apenas envolvê-lo na camada de persistência e utilizar ações semânticas, que não fazem parte da definição formal do dispositivo, mas podem ser associadas às suas regras, para realizar a interface entre ambos.

REFERÊNCIAS

[1] M. P. Atkinson, P. J. Bailey, K. J. Chisholm, P. W. Cockshott, and R. Morrison, "An approach to persistent programming," in *Readings in object-oriented database systems*, S. B. Zdonik and D. Maier, Eds. Morgan Kaufmann Publishers Inc., 1990, pp. 141–146.

[2] M. Atkinson and R. Morrison, "Orthogonally persistent object systems," *The VLDB Journal*, vol. 4, no. 3, pp. 319–402, 1995.

[3] H. Kaplan, "Persistent data structures," in *Handbook on data structures and applications*, D. Mehta and S. Sahni, Eds., 1995, pp. 241–246.

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

[5] P. R. M. Cereda and J. José Neto, "Mineração adaptativa de dados: Aplicação à identificação de indivíduos," in *Memórias do Sexto Workshop de Tecnologia Adaptativa*, São Paulo, 2012.

[6] H. R. Lewis and C. H. Papadimitriou, *Elements of the theory of computation*. Prentice Hall, 1997.

[7] J. José Neto, "Contribuições à metodologia de construção de compiladores," Tese de livre-docência, Escola Politécnica, Universidade de São Paulo, 1993.

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.

Proposal of modification of the DJ Library for implementing Adaptive Technology

R. Caya, and J. J. Neto

***Abstract*—Implementation of adaptive programs have being an open concern in both scientific and commercial communities. The former one worrying about a clear and formal identification of the elements and behavior of adaptive mechanisms. The second one caring about the practical approach by which such behavior can be included at conventional applications. This paper presents a strategy for implementing Adaptive Technology in Java language using a standard library. Specifically, approaches about adaptive behavior in the Demeter Project and LTA are analyze to found correlated conceptual points. The objective is applying modifications over a tool, the DJ library, to achieve a best suit with the formalisms developed at LTA, without losing expressiveness at the implementation level. A particular case is analyzed by working with adaptive automata theory. From this case, key points are identified for future modifications through conversions in the representation of the underlying model and the traversal strategy used by the DJ library.**

***Keywords*— Adaptive Programming, Aspect-Oriented Programming, Adaptive Configuration, Adaptive Automata, Java.**

I. INTRODUÇÃO

ATUALMENTE, a adaptatividade é um dos recursos mais procurados dentro das aplicações de software. Independentemente do tamanho, a complexidade, ou do tipo de tecnologia utilizada, a capacidade de desenvolver peças de software que possuam a habilidade de reagir às mudanças dentro do contexto de execução, e sejam capazes de modificar o seu próprio comportamento para se adequar a novas situações é o principal requerimento das tecnologias de vanguarda.

Particularmente, espera-se que a adaptatividade permita que o software que a contém possa sobreviver as mudanças ao longo da sua vida útil por meio da aplicação de ajustamentos, de modo que os sistemas ganhem estabilidade e tolerância. Adicionalmente, o valor da adaptatividade se vê acrescentado ao ser entendida como uma base sobre a qual é possível desenvolver e aplicar teorias contemporâneas como *context-awareness*, *machine learning* [1], e *evolving computing*, as quais estão sendo foco de grande quantidade de pesquisas.

Ao longo dos anos, na tentativa para desenvolver peças de software com caraterísticas adaptativas têm seguido diferentes abordagens, tanto em paradigmas de programação quanto em metodologias, e até mesmo a terminologia utilizada para identificar um programa com comportamento adaptativo apresenta diversidade, aparecendo termos muito próximos como: auto-modificável e reconfigurável. No entanto, um contrato tácito, permite estabelecer uma definição geral de um programa adaptativo como aquele que é capaz de modificar o seu comportamento de acordo com as mudanças que acontecem dentro do seu ambiente de execução em determinado ponto (estado) de sua configuração [1]. Assim, o conceito de adaptatividade foi implementada dentro de programas que tem seguido diversos paradigmas de programação, de modo que existem programas adaptativos seguindo programação funcional [2], programação orientada a objeto [1], programação orientada a componentes [3], e mais recentemente programação orientada a aspectos [4].

Basicamente, esta diversificação é possível porque a adaptatividade é uma capacidade que pode ser abstraída e expressa por meio de teorias e métodos formais que expõem claramente os elementos e mecanismos que permitem o seu comportamento, dando-lhe um caráter autônomo. Não obstante, ao implementar esses dispositivos com tecnologia adaptativa é preciso levar a especificação teórica para o plano das linguagens de programação, as quais convencionalmente não fornecem suporte para adaptatividade [1]. É durante este processo que a adaptatividade fica embutida no meio do código convencional, camuflada, e perde tanto identidade quanto legibilidade.

Para superar esse problema é preciso ter um modo de implementação prático, que permita manter a clareza e expressividade que a adaptatividade tem no nível dos métodos formais, e a traduz para os vários paradigmas e linguagens de programação ao pôr em evidencia os elementos que a a compõem. Desta forma, é possível estabelecer um fluxo contínuo e facilmente reconhecível da correspondência dos elementos e métodos de um dispositivo adaptativo partindo da especificação formal até uma implementação concreta, tornando-se um elemento auto-descriptivo e diferençável dentro do código comum. Estas características podem ser aproveitadas não só no ambiente acadêmico, com finalidade didática, mas também para facilitar tarefas do desenvolvimento de software no ambiente comercial, tais como: documentação, manutenção, e até para expor as funcionalidades do sistema.

Neste trabalho se propõe o estabelecimento de uma correlação entre a especificação formal de um dispositivo adaptativo e a sua implementação concreta para uma linguagem de programação, mantendo o máximo de rastreabilidade entre os elementos que compõem o dispositivo em ambas as dimensões, e especialmente, facilitando a identificação da natureza particular do comportamento adaptativo par ao programador.

As próximas seções deste artigo estarão organizadas da seguinte maneira: na seção 2 serão abordados os conceitos correspondentes ao paradigma de Programação orientada a aspectos, sua filosofia e vantagens. Na seção 3 será apresentado o Projeto Demeter, que apresenta o conceito de Programação

Adaptativa e os mecanismos para desenvolver programas segundo aquele conceito. A seção 4, descreverá a biblioteca DJ que fornecerá a base para o trabalho deste projeto. A seção 5 fornecerão a correlação entre os conceitos de adaptatividade entre o Projeto Demeter e os trabalhos no LTA, e a seção 6 irá descrever a estratégia de mapeamento entro as descrições formais e as implementações dos dispositivos adaptativos. A seção 7 fornece uma revisão breve dos trabalhos relacionados para a tradução de notações formais e implementações nos casos de adaptatividade. Por fim, na seção 8 estarão as considerações finais.

II. PARADIGMA DE PROGRAMAÇÃO ORIENTADA A ASPECTOS

Programação Orientada a Aspectos (*Aspect-oriented programming* ou *AOP*) é um paradigma de metaprogramação que permite separar e organizar o código de um programa de acordo com a importância que ele tem dentro da aplicação (*separation of concerns*) [5]. Assim, contrário à Programação Orientada a objetos, na qual a é baseado na construção de entidades únicas que encapsulam semelhanças de estrutura e comportamento, AOP permite encapsular e modularizar aqueles métodos espalhados transversalmente na estrutura de classes destinados para atender funcionalidades de interesse global (*crosscutting concerns*). Esta nova abstração é possível de realizar graças à incorporação de um novo conceito denominado aspecto. De acordo com isso, diferentes módulos do programa são especificados durante o projeto de construção de software, cada um deles tomando conta de só um aspecto, e logo esses aspectos são entrelaçados no código principal para produzir um programa que aborde todos os requerimentos. Aqueles aspectos são entrelaçados por AOP por meio de "pontos de execução", ou *join points*, os quais são pontos dentro do programa de conformidade com as especificações do programador e que estão em capacidade de executar um comportamento adicional. Aqueles pontos podem ser entrada ou saída de métodos, criação ou eliminação de objetos [6].

As principais vantagens que a AOP traz consigo são que:

(i) um aspecto pode alterar o comportamento de um código estático (parte do programa não orientada a aspectos);
(ii) fornecem uma simplificação ao programa original;
(iii) fornecem facilidade para manutenção para o sistema pois tornasse tolerável para mudanças nas especificações por causa da evolução do software.

Apesar de que a ideia de entrelaçar aspectos é muito padronizada, o método pode variar substancialmente e se tornar muito dinâmico por meio do uso de reflexão, ou ficar completamente estático utilizando geração de código.

III. O PROJETO DEMETER

O Projeto Demeter [7] é um projeto de pesquisa desenvolvido na *Northeastern University*, cujo objetivo principal é melhorar a produtividade dos desenvolvedores de software em uma ordem de magnitude [8]. Para atingir isso são desenvolvidos métodos e ferramentas suportadas pelas teorias de Programação Adaptativa(*Adaptive Programming* ou AP) e Programação Orientada a Aspectos (*Aspect-Oriented Programming* ou AOP), as quais principalmente procuram manter um baixo acoplamento entre os objetos e as operações contidas em um programa, de modo que seja possível aplicar mudanças em qualquer aqueles aspectos sem causar impactos sérios dentro do outro [7]. As ferramentas desenvolvidas pelo equipe do projeto estão orientadas para incrementar as capacidades de tecnologias comerciais amplamente utilizadas, tais como as linguagens de programação C++, Java, C#, Tcl/Tk, Perl 5. No entanto não se limita a essas tecnologias e também trabalha em coordenação com UML e os padrões de projeto de software, de modo que os resultados podem ser más abrangentes [8]. Além disso, de acordo com o site oficial, eles estão trabalhando atualmente no desenvolvimento de ferramentas para XML e algumas linguagens de programação para criação de componentes de software.

As diferentes ferramentas disponibilizadas pelo Projeto Demeter encontrasse hierarquizadas na Fig. 1.

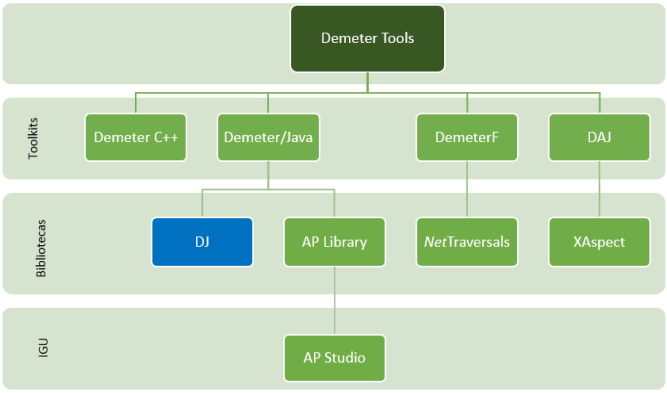


Figura 1. Ferramentas desenvolvidas pelo Projeto Demeter

A. Programação Adaptativa

De acordo com [9], a Programação Adaptativa é um caso particular do paradigma de AOP, no qual alguns dos blocos que compõem um programa são expressados em termos de grafos, e outros blocos podem consultar os primeiros por meio de conjuntos de estratégias de trajetórias. Assim, por meio de especificações breves é possível descrever o comportamento de programas nos níveis de projeto e implementação de software. A principal diferencia entre AOP e AP é que AOP tem como principal objetivo a separação de interesses que compõem o sistema, enquanto que AP ocupasse de aportar "timidez", entendido como desconhecimento parcial entre as implementações dos interesses e as estruturas onde são executados.

O objetivo principal da AP é separar as diferentes preocupações envolvidas na resolução de um problema minimizando as dependências entre as partes que o compõem, de modo que mudanças futuras em alguma das partes (sejam

arbitrarias ou produto da evolução do programa) tenha um impacto mínimo nas outras partes.

A vantagem que espera-se obter com esta abordagem é a construção de programas minimamente entrelaçados, mais flexíveis, compreensíveis, e mais curtos, sem que isso signifique perda de eficiência em tempo de execução. Em geral a AP procura elevar a Programação Orientada a Objeto para um nível maior de abstração permitindo ao usuário se focar nas classes essenciais do programa e nos aspectos invariantes das relações entre elas. Como consequência, AP utiliza protocolos que permitem especificar precondições, post condições, e invariantes para as diferentes classes participantes, e colaboradoras de uma operação lógica.

Assim, as características do AP realçam claramente a tolerância a mudanças e a incorporação de novo comportamento sobre uma base estática (invariável) como motivações importantes para o seu desenvolvimento. O importante para notar é que ambos conceitos representam pilares fundamentais dentro da teoria de Tecnologias Adaptativas especificadas e utilizadas no LTA. Esta proximidade é detalhada na seção V do presente documento.

B. Estratégias de travessia

A ideia central detrás das estratégias de travessia utilizadas em AP é implementar programas independentes da estrutura de classes especifica (*structure-shy*) de modo que a mesma estratégia pode ser utilizada com diferentes estruturas de classes [8]. Uma estratégia de travessia pode ser entendida como uma especificação parcial de um grafo, na qual são assinalados alguns nós invariantes (origem, destino, e alguns membros intermédios) e aristas estratégicas.

De modo geral, uma estratégia de travessia permite percorrer um grafo de objetos $\mathcal{O}$ (uma representação de um fluxo de dados não trivial) procurando todos os objetos de tipo c' dado um determinado objeto o de classe c, sem ter maior conhecimento sobre todas conexões da cadeia, só aquele fornecido pelo correspondente grafo de classes $\mathcal{C}$. Uma especificação formal e detalhada dos algoritmos que implementam a estratégia de travessia pode-se encontrar em [10] [11].

Assim, quando acontecem mudanças (chamadas como evoluções do programa) em partes não críticas a estratégia de travessia permanece invariante e válida, o que significa que a mesma codificação de comportamento é aplicável, e não é preciso realizar nenhuma alteração manual para adapta-a a aquela mudança, como máximo só é necessário atualizar a estrutura sobre a qual ira ser aplicada.

C. Padrão do Visitante Adaptativo

O *Visitor Pattern* é um padrão comportamental de projeto de construção de software desenvolvido em [12] para definir a comunicação entre as classes e os objetos que o compõem. A ideia principal é separar o comportamento de uma estrutura complexa de dados da funcionalidade que será executada sobre aqueles dados que ela possui. O benefício principal da implementação deste padrão é que pode ser incorporada nova funcionalidade para cada elemento da estrutura de dados sem conhecer antecipadamente qual é a relação com o resto do modelo. Essencialmente, o *Visitor Pattern* consiste em 2 partes: um método `visit()` implementado no visitante que é chamado por todos os elementos da estrutura de classes, e os métodos `accept()` em cada uma das classes que recebe um objeto visitante e acessa dinamicamente ao método `visit()` correspondente.

O padrão Visitante Adaptativo implementado nas ferramentas de Demeter é uma forma modificada do *Visitor Pattern* [13] a qual trabalha com ajuda das técnicas de reflexão para executar funcionalidades no tempo de execução. Uma das principais diferencias é que no Visitante Adaptativo só um conjunto mínimo de métodos deve ser definido, mais exatamente, só aqueles que correspondem com pontos desejados dentro da estratégia de travessia, e não todos como ocorre na versão original. Outra diferencia, é que no Visitante Adaptativo não é preciso implementar nenhum método `accept` e também não existe referência sobre o comportamento de travessia dentro dos métodos do visitante [9]. Além disso, o visitante adaptativo permite estabelecer comportamentos em quatro momentos da travessia:

- `start()`: invocado no início da travessia antes de começar o percurso do grafo.
- `before(ClassName c)`: é invocado quando um objeto `c` da classe `ClassName` é atingido. O método é executado antes de atravessar os atributos do objeto `c`.
- `after(ClassName c)`: é invocado quando um objeto c da classe `ClassName` é atingido. O método é executado depois de atravessar os atributos do objeto `c`.
- `finish()`: invocado no final da travessia, isto é, depois que todos os nós da estrutura foram percorridos

Assim, de acordo com [14] o padrão de implementação de um visitante permite construir uma unidade de comportamento independente das mudanças na estrutura de objetos e também da estratégia de travessia, de modo que pode ser reutilizado com outras configurações.

IV. BIBLIOTECA DJ

É uma biblioteca para Java, o nome vem do Demeter for Java, desenvolvida pelo equipe do Projeto Demeter com a finalidade de fornecer suporte, em código Java-nativo, para a implementação e experimentação das ideias da AOP [9]. A biblioteca DJ encontra-se disponível em [15], e em contraste com DemeterJ, para trabalhar com ela só e preciso realizar a incorporação padrão para uma biblioteca externa o qual simplifica o trabalho do programador.

Ao mesmo tempo, DJ procura cumprir a Lei, Demeter [16] [9], uma regra de estilo de programação para garantir o baixo acoplamento entre os aspectos estruturais e comportamentais de um programa, mas superando as desvantagens do espalhamento

de código por ela gerado quando ao seguir outros paradigmas de programação. A ideia principal em DJ é permitir a interpretação, em tempo de execução, de "métodos adaptativos" [13], entendidos como unidades de comportamento independentes do conhecimento do modelo de objetos do programa, mas que podem mudar a execução global do mesmo.

Aqueles métodos, junto com o comportamento que encapsulam, são expressados através dois mecanismos [13] [14]:

(1) Uma representação da estrutura de objetos do programa, chamada **grafo de classes**, a qual é gerada pela Classe *ClassGraph* de DJ utilizando reflexão em tempo de execução com ajuda do *ClassLoader* na JVM. A classe permite indicar o pacote de classes do qual deverá ser gerado o grafo, assim como também adicionar explicitamente outras classes ou pacotes ao grafo gerado, para os casos nos quais o código não se encontra disponível localmente ou as classes não foram carregadas ainda na JVM [9].

(2) Uma **estratégia de travessia**, a qual descreve como alcançar os participantes envolvidos na computação de um determinado comportamento em um alto nível, se referindo só para um número reduzido das classes no modelo de objetos do programa. Particularmente, só é preciso assinalar: uma raiz, uma classe alvo, e alguns pontos intermediários e restrições para garantir que a travessia ira seguir só aquelas trajetórias desejadas.

(3) Um **visitante adaptativo**, o agente que de fato vai percorrer o modelo de objetos de acordo com a estratégia, e ira aplicar um conjunto de ações (métodos *before* e *after*) quando um participante de determinado tipo seja alcançado.

Exemplo básico do uso de DJ em Java

```
// Importação das classes ...

public class Main {
  public static void main(String[] args) {

  // Criação de objetos e inicializações ...

    ClassGraph cg = new ClassGraph("transit");
    Strategy sg = new Strategy("from transit.BusRoute through transit.BusStop to
                                transit.Person");

    TraversalGraph tg = new TraversalGraph(sg, cg);
    tg.traverse(route, new Visitor() {
       public void start() { System.out.println("Scanning the state of a bus route");}
       public void finish() { System.out.println("end of the route scanning"); }

       public void before(BusStop b) { System.out.println("On "+ b.getName()); }
       public void after(BusStop b)  { System.out.println("going to next bus stop"); }

       public void after(Person p)  { System.out.println(p.getName()); }
    });

  // resto da implementação do método main
  }
}
```

Figura 2. Exemplo de programa utilizando a biblioteca DJ

A Fig.2 mostra um exemplo de um programa utilizando DJ para o caso de um sistema de trânsito com rotas e pontos de ônibus. O único requerimento é a impressão de todas as pessoas que aguardam em aquela rota. No exemplo, o objeto `cg` do tipo *ClassGraph* contém o grafo das classes conteúdas no pacote indicado como parâmetro [9], neste caso "`transit`", e o objeto `sg` implementa a estratégia de travessia de acordo com a cadeia de texto indicada como parâmetro. Neste caso é preciso mencionar o pacote no qual estão conteúdas as classes pois a referência para elas está sendo efeituada desde o método `main` que pertence ao pacote por defeito.

Com ambos objetos contendo a informação necessária é criado o grafo de travessia, `tg`, que contém os objetos instanciados no programa. O método `traverse` de `tg` expõe o mecanismo de percurso dentro de `tg`, iniciando no nó indicado como raiz (`route`), um objeto do tipo Visitor vai ativar os métodos adaptativos correspondentes. No exemplo, são implementados métodos `before` e `after` para o caso em que o visitante chegar para uma instancia da classe `BusStop`, e um método `after` para o caso em que uma instancia da classe `Person` seja encontrada dentro do percurso do grafo.

A codificação integral do exemplo pode ser obtida e testada de livremente desde https://github.com/rosedth/WTA2014.

V. RELACIONAMENTO ENTRE ABORDAGENS: PROJETO DEMETER E LTA

Apesar de ter sido desenvolvidos em lugares distantes, com objetivos e motivações diferentes, existem acordos do nível conceitual entre a proposta de Programação Adaptativa feita pelo Projeto Demeter e os trabalhos com dispositivos adaptativos realizados no Laboratório de Tecnologias Adaptativas (LTA) da Escola Politécnica da Universidade de São Paulo. Nesta seção irão ser apresentadas ambas abordagens de adaptatividade, primeiro de maneira individual, e em seguida, irão se expor as suas semelhanças.

A. Adaptatividade segundo Demeter

O Projeto Demeter incorpora o conceito de adaptatividade desde dois abordagens diferentes: o abordagem de transcendência dos programas às mudanças estruturais, e o abordagem de incorporação de comportamentos em tempo de execução. A continuação iremos detalhar ambos os abordagens.

Primeiro, segundo [17] para o Projeto Demeter a adaptatividade em um programa é entendida como a capacidade de aquele programa para se adaptar automaticamente as mudanças que acontecem dentro do seu contexto, particularmente mudanças estruturais, de maneira que o impacto gerado por elas seja o mínimo possível dentro do sistema integral. Para atingir esse objetivo um programa adaptativo é expressado por meio de fragmentos de código que encapsulam os conceitos e as funcionalidades requeridas para desenvolver uma determinada funcionalidade, mas com conhecimento mínimo sobre a estrutura na qual iram ser aplicadas. Assim esses fragmentos apresentam baixo acoplamento entre eles mas cooperam entre si para formar o

sistema integral [14]. De acordo com isso, é possível aplicar o mesmo comportamento para diferentes estruturas de classes, sempre e quando se cumpra com manter invariáveis um conjunto de especificações sucintas.

Segundo, o Projeto Demeter comparte com AOP as características atribuídas para o conceito de aspecto, mais especificamente para Demeter tanto a estratégia de travessia como o visitante adaptativo corresponde a aspectos em AOP. Devido a isso também admitem, por definição, a capacidade que um aspecto tem para alterar o comportamento de um programa por meio da aplicação de um comportamento adicional (*advice* em AOP, e para Demeter os métodos implementados por o Visitante Adaptativo) em um determinado ponto de execução (*join point* em AOP, e para Demeter um objeto dentro da travessia para o qual existe implementado um comportamento adicional).

B. Tecnologias Adaptativas no LTA

O Laboratório de Tecnologias Adaptativas (LTA) da Escola Politécnica da Universidade de São Paulo tem realizado pesquisas relacionadas com tecnologias adaptativas desde os anos 90. Como resultado diversos formalismos, técnicas, métodos, ferramentas e aplicações foram desenvolvidas com base na incorporação da adaptatividade em diversas áreas.

Dentro do contexto dos trabalhos desenvolvidos no LTA, a adaptatividade é entendida como a capacidade que tem um sistema de responder para estímulos externos determinando e realizando auto reconfigurações dinâmicas de tipo estrutural [18] e comportamental [19], levando em conta para efeituar aquelas mudanças a informação acumulada durante experiências anteriores [20]. Assim, o termo Tecnologia Adaptativa refere-se a aplicação da adaptatividade e fins práticos e concretos.

Um dispositivo adaptativo é entende-se como um dispositivo cuja operação é guiada por regras e que é capaz por seus próprios médios de realizar mudanças nesse conjunto regras. Na construção formal de aqueles dispositivos é possível identificar duas componentes participantes [19]: um **dispositivo subjacente** (usualmente não adaptativo), e um **mecanismo adaptativo**, responsável pela incorporação da adaptatividade. Desse modo, no momento em que uma dependência de contexto é detectada pelo dispositivo, uma ação adaptativa de automodificação é efeituada, a qual a sua vez pode mudar o conjunto de regras definidas para a execução do dispositivo e o estado corrente em que se encontra [21].

C. Relacionamento entre ambos abordagens

De acordo com o exposto, os métodos es técnicas desenvolvidos dentro do Projeto Demeter para suportar a Programação Adaptativa fornecem uma plataforma que é possível de aproveitar para a implementação de tecnologia adaptativa. Particularmente, ambas as abordagens:

- Separam conceitualmente os mecanismos adaptativos a serem aplicados da estrutura que que representa e controla o funcionamento principal do dispositivo.
- Identificam o recorrido de uma a representação abstrata do funcionamento do programa como dispositivo subjacente sobre o qual iram acontecer as mudanças adaptativas.
- Identificam as duas oportunidades de aplicação de mecanismos adaptativos: uma imediatamente antes e outra imediatamente depois de "atravessar" um elemento da estrutura subjacente.
- Admitem que a execução das ações adaptativas pode alterar o comportamento geral do dispositivo.

Essas aproximações conceituais permite identificar as ferramentas desenvolvidas no Projeto Demeter como suporte para a implementação de dispositivos adaptativos partindo da especificação formal e expondo claramente os elementos que os compõem. Assim, a expressividade outorgada na especificação formal de aqueles dispositivos pode ser identificada dentro da sua codificação, tornando-a descritiva.

VI. ESTRATÉGIA DE MAPEAMENTO ENTRE ABORDAGENS

Esta seção apresenta um exemplo que permite identificar mais claramente a compatibilidade entre os abordagens seguidos no Projeto Demeter e os Dispositivos Adaptativos do LTA. Neste caso se trabalha com a teoria de Autómatos Adaptativos (AA) descrita em [22] por apresentar uma versão simplificada e expressiva.

Um AA está definido por um dispositivo subjacente que corresponde com um autômato de pilha estruturado, e um mecanismo adaptativo. O autômato de pilha estruturado é identificado por a 7-tupla: $M = (Q, \Sigma, \Gamma, \delta, q_0, \mathrm{Z}, F)$ onde:

- Q: é um conjunto finito de estados
- Σ: é o alfabeto de entrada
- Γ: é o alfabeto da pilha
- δ: são o conjunto de regras de transição: internas, externas e de chamada e retorno de sub-máquina.
- q_0: é o estado inicial do autômato
- Z: é o símbolo inicial na pilha
- F: é o conjunto de estados finais do autômato

Entanto o mecanismo adaptativo é expressado como a aderência de um conjunto opcional de (zero, uma ou até duas) ações adaptativas correspondentes com a regra de transição a ser aplicada. Existem dois momentos nos quais aquelas ações podem ser executadas: antes ou depois da aplicação da regra selecionada.

Uma ação adaptativa descreve as modificações a serem aplicadas no autômato adaptativo no momento em que forem chamadas. As modificações geradas por uma ação adaptativa podem ser de três tipos de notadas da seguinte maneira:

- $?$: ação de tipo inspeção, a qual procura as regras de transição que cumprem um determinado padrão.
- $-$: ação de tipo remoção, a qual remove uma regra do conjunto de transições se ela cumpre com determinado padrão.
- $+$: ação de tipo inserção, a qual incorpora uma nova regra no conjunto de transições de acordo com determinado padrão.

Assim, em cada estado da execução do autômato são executadas tanto as regras de transição convencionais como as ações adaptativas, até atingir um estado final.

Partindo dessa formalização é possível descrever o relacionamento que permite aproveitar as técnicas do Projeto Demeter para a implementação de um autômato adaptativo.

Uma primeira proposta para representar o dispositivo subjacente é aproveitar o paradigma de programação orientada a objetos para representar os elementos do autômato. Assim seria possível implementar uma hierarquia de classes como a que se mostra na Fig.3.

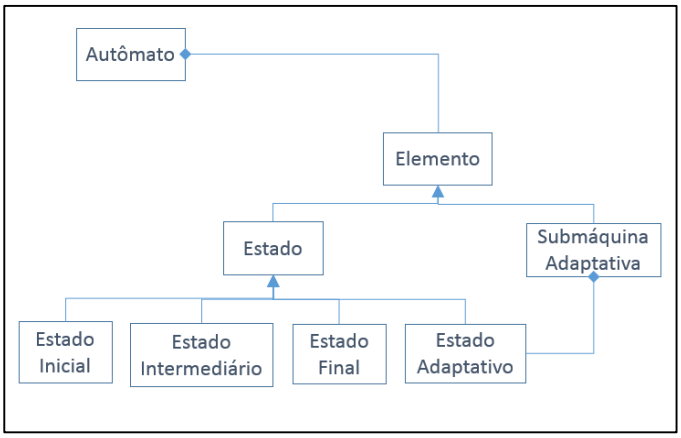


Figura 3. Proposta da hierarquia de classes para a representaçaõ de um AA

De modo que Q está representado pela lista de elementos que formam parte do autômato, e Σ corresponde à entrada, entendida como fluxo de dados, com a qual irá trabalhar o autômato. O estados identificados por q_0 e F estão representados por meio de herança na hierarquia de objetos apresentada, e formam parte da lista de elementos do autômato.

Os elementos que se referem ao trabalho com pilha, Γ e Z, vão se encontrar encapsulados dentro do mecanismo de travessia de DJ, o qual indica o caminho a percorrer. Assim, também é possível efeituar as chamadas a sub-máquina de maneira evidente e natural graças à possibilidade de inclusão de uma sub-máquina como um elemento dentro da lista de elementos do autômato.

O conjunto de transições está representado pela estratégia de travessia, o *ClassGraph* e o *ObjectGraph*. Com esses elementos DJ estabelece as regras de transição como a possibilidade de acesso entre um objeto e outro segundo a hierarquia de objetos embebida no *ClassGraph*. Atualmente, DJ só permite uma trajetória dirigida, na qual percorre cada nó do grafo uma única vez. Dentro das modificações desejadas estaria a implementação de trajetórias que permitam atravessar novamente por alguns estados.

O mecanismo adaptativo é expressado por meio da implementação do padrão do Visitante Adaptativo em DJ, no qual as ações adaptativas podem ser adicionadas para serem executadas nos momentos em que determinados estados forem atingidos. Os momentos nos quais é possível incrementar um comportamento que potencialmente realizara uma mudança no comportamento do programa estão expressos dentro de DJ como os métodos adaptativos `before` e `after` correspondentes a um objeto determinado. Dentro dos métodos adaptativos é possível realizar inspeção (examinar os valores de alguma variável ou atributo dos objetos), remoção e inserção de elementos.

Neste ponto, é importante identificar que existe inclusive um consenso de terminologia entre ambos os abordagens respeito ao nome dos mecanismos que permitem a adaptatividade.

VII. MODIFICAÇÕES PROPOSTAS

A maior parte das modificações a serem realizadas na biblioteca DJ correspondem à representação do dispositivo subjacente e à implementação da estratégia de travessia, assim como possíveis efeitos colaterais das mudanças nela aplicadas.

Inicialmente, e preciso modificar o modelo de representação de uma árvore de classes para um modelo de grafo que permita refletir mais livremente o comportamento do dispositivo.

Também é necessário fornecer regras de transição mais enriquecidas dentro da estratégia de travessia que permitam realizar a execução do autômato de maneira mais natural que aquela equivalente ao recorrido de grafos dirigidos. Atualmente, a estratégia de travessia só estabelece como regras de transição se um elemento é alcançável ou não a partir de outro com base na hierarquia de classes (ClassGraph).

Particularmente, é necessário incorporar dentro da estratégia de travessia a possibilidade de retornar para determinados estados dentro da lista de elementos do autômato para os casos nos quais existem laços entre estados.

As modificações dentro da estratégia de travessia deveriam ser transparentes para o programador já que DJ fornece a possibilidade de inserir a especificação de uma estratégia como uma cadeia de caracteres ou a partir de um arquivo de texto.

VIII. TRABALHOS RELACIONADOS

Com o surgimento das teorias sobre comportamento adaptativo, surgiu também a necessidade de elementos computacionais que permitam fazer uso da potencialidade por ela fornecida. Particularmente, a existência de uma linguagem de programação que suporte com facilidade a implementação de mecanismos adaptativos por meio da inclusão de primitivas adequadas é identificada como uma grande necessidade.

Tanto [2] quanto [1] são exemplos de esforços realizados pela comunidade cientifica para fornecer suporte para a incorporação de adaptatividade dentro dos programas convencionais. No entanto, ambos as abordagens propõem a criação de novas linguagens de programação que implementam as primitivas necessárias para adaptatividade.

Uma outra necessidade é aquela de fornecer uma transição clara entre a fundamentação teórica de um dispositivo e a implementação do mesmo. O objetivo neste caso é do nível didático e para facilitar o entendimento do comportamento de

aquele dispositivo. A biblioteca JFLAP [23](Formal Languages and Automata Theory Package for Java) procura realizer dita aproximação provendo uma ferramenta para visualização e teste de especificações formais.

Neste trabalho utiliza-se as técnicas desenvoltas dentro de Projeto Demeter, particularmente na biblioteca DJ, que são métodos que permitem o desenvolvimento de programação adaptativa e servem para resolver diversos problemas computacionais em diversos linguajes de programação e paradigmas. O intuito é adaptá-las para permitir uma representação mais natural dos elementos conteúdos nas especificações formais dos dispositivos adaptativos. Desse modo, a principal diferencia com os trabalhos anteriores é que não se propõe a modificação da linguagem ou a criação de novas linguagens para a implementação de adaptatividade, si não que se procura aproveitar as facilidades que prove DJ, como biblioteca nativa, para apresentar aos programadores uma alternativa que de forma padrão permita-lhes experimentar programas com características adaptativas.

IX. CONCLUSÃO

Diante do exposto, conclui-se que, é possível realizar um mapeamento entre as especificações formais dos dispositivos adaptativos para uma implementação assistida pelos conceitos desenvolvidos dentro do Projeto Demeter. Implementação que seja capaz de conservar e refletir a expressividade, identidade e particularidade dos elementos que definem o comportamento adaptativo. Embora, algumas modificações precisam ser feitas para atingir uma compatibilidade total, a correlação encontrada entre os abordagens seguidos tanto por o equipe do Projeto Demeter como pôr o equipe do LTA, permitem estabelecer uma plataforma sobre a qual trabalhar com miras a atingir o objetivo de facilitar a inclusão de tecnologias adaptativas pelos programadores nos sistemas.

A viabilidade do projeto de pesquisa é suportada devido a que o Projeto Demeter é um projeto de software livre que até hoje conta com uma comunidade ativa que trabalha continuamente sobre os objetivos para ele estabelecidos. Do mesmo modo, as formulações formais dos dispositivos de Tecnologia Adaptativa permitem pôr à disposição do programador uma ferramenta poderosa a qual faria possível uma mudança na metodologia de programação: programação baseada em formulações teóricas, uma conexão entre teoria e prática desejada por longo tempo.

O potencial de lograr uma implementação com clareza dos mecanismos adaptativos aporta não só como finalidade didática si não que também permite aproveitar os benefícios de um uso ciente das vantagens que eles fornecem. Ao respeito, trabalhar sobre a base de uma das ferramentas desenvolvidas pelo Projeto Demeter permite pensar na possibilidade de propagar o trabalho para outras ferramentas que suportam diferentes linguagens e paradigmas de programação.

REFERÊNCIAS

[1] C. Simpkins, S. Bhat, C. J. Isbell e M. Mateas, "Towards Adaptive Programming: Integration Reinforcement Learning into a Programming Language," em *23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, New York, 2008.

[2] U. A. Acar, G. E. Blelloch e R. Harper, "Adaptive Functional Programming," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 28, nº 6, pp. 990-1034, 2006.

[3] H. Cervantes e R. S. Hall, "A Framework for Constructing Adaptive Component-Based Applications: Concepts and Experience," em *Lecture Notes in Computer Science. Component-Based Software Engineering*, vol. 3054, I. Crnkovic, J. A. Stafford, H. W. Schmidt e K. Wallnau, Eds., Springer Berlin Heidelberg, 2004, pp. 130-137.

[4] W. Vanderperren, D. Suvée, B. Verheecke, M. A. Cibrán e V. Jonckers, "Adaptive programming in JAsCo," em *4th international conference on Aspect-oriented software development*, New York, USA, 2005.

[5] B. Donkervoet e G. Agha, "Reflecting on aspect-oriented programming, metaprogramming, and adaptive distributed monitoring," em *5th international conference on Formal methods for components and objects*, Amsterdam, The Netherlands, 2007.

[6] P. Cointe, H. Albin-Amiot e S. Denier, "From (Meta) Objects to Aspects: A Java and AspectJ Point of View," em *Formal Methods for Components and Objects. Lecture Notes in Computer Science*, vol. 3657, F. S. Boer, M. M. Bonsangue, S. Graf e W. Roever, Eds., Berlin, Springer Berlin Heidelberg, 2005, pp. 70-94.

[7] Northeastern University, "Demeter: Aspect-Oriented Software Development," College of Computer and Information Science, 2013. [Online]. Available: http://www.ccs.neu.edu/home/lieber/what-is-demeter.html. [Acesso em 28 Outubro 2013].

[8] K. Lieberherr, "Demeter and aspect-oriented programming," em *STJA Conference*, Erfurt, Germany, 1997.

[9] K. Lieberherr e D. H. Lorenz, "Coupling Aspect-Oriented and Adaptive Programming," em *Aspect-Oriented Software Development*, R. E. Filman, T. Elrad, S. Clarke e M. Akşit, Eds., Boston, Addison-Wesley, 2005, pp. 145-164.

[10] K. J. Lieberherr, B. Patt-Shamir e D. Orleans, "Traversals of object structures: Specification and Efficient Implementation," *ACM Trans. Program. Lang. Syst.*, vol. 26, nº 2, pp. 370-412, 2004.

[11] K. Lieberherr e M. Wand, "A Simple Semantics for Traversals in Object Graphs," Boston, Massachusetts, USA, 2001.

[12] E. Gamma, R. Helm, R. Johnson e J. Vlissides, Design P atterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[13] K. Lieberherr, D. Orleans e J. Ovlinger, "Aspect-oriented programming with adaptive methods," *Commun. ACM*, vol. 44, nº 10, pp. 39-41, Outubro 2001.

[14] D. Orleans e K. Lieberherr, "DJ: Dynamic Adaptive Programming in Java," em *Metalevel Architectures and Separation of Crosscutting Concerns. Lecture Notes in Computer Science*, vol. 2192, A. Yonezawa e S. Matsuoka, Eds., Berlin, Springer Berlin Heidelberg, 2001, pp. 73-80.

[15] Demeter Research Group, "DJ: Dynamic Structure-Shy Traversals and Visitors in Pure Java," College of Computer Science, Northeastern University, [Online]. Available: http://www.ccs.neu.edu/research/demeter/DJ/.

[16] K. J. Lieberherr e I. Holland, "Assuring Good Style for Object-Oriented Programs," *IEEE Software*, pp. 38-48, Setembro 1989.

[17] K. J. Lieberherr, Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns, Boston: PWS Publishing Company, 1996.

[18] M. A. A. Sousa, A. R. Hirakawa e J. J. Neto, "Adaptive Automata for Mapping Unknown Environments by Mobile Robots," *Lecture Notes in Artificial Intelligence. Advances in Artificial Intelligence - IBERAMIA 2004*, pp. 562-571.

[19] J. J. Neto, "Um Levantamento da Evolução da Adaptatividade e da Tecnologia Adaptativa," em *Revista IEEE América Latina*, 7 ed., vol. 5, IEEE, 2007, pp. 496-505.

[20] A. H. Tchemra, "Aplicação da Tecnologia Adaptativa em Sistemas de Tomada de Decisão," *Revista IEEE América Latina*, vol. 5, nº 7, pp. 552-556, Novembro 2007.

[21] J. J. Neto e A. V. Freitas, "Using Adaptive Automata in a Multi-Paradigm Programming Environment," em *International Conference on Applied Simulation and Modelling, ASM2001 - IASTED*, Marbella, Espanha, 2001.

[22] J. J. Neto e C. Bravo, "Adaptive Automata - a reduced complexity proposal," em *7th International Conference of Implementation and Application of Automata (CIAA 2002)*, Tours, France, 2002.

[23] S. H. Rodger, E. Wiebe, K. M. Lee, C. Morgan, K. Omar e J. Su, "Increasing engagement in automata theory with JFLAP," em *40th ACM technical symposium on Computer science education, SIGCSE '09*, Chattanooga, TN, USA, 2009.

Rosalia Edith Caya Carhuanina nasceu na cidade de Lima, Perú, em 27 de fevereiro de 1986. Recebeu o título de Bachiller em Ciencias e Ingeniería con Mención em Ingeniería Informática da Pontificia Universidad Católica del Perú (PUCP) em Lima, Perú. Entre os anos de 2007 e 2011, participou do Grupo de Pesquisa em Inteligência Artificial do departamento de Engenharia Informática desta Universidade. Atualmente é aluna de Mestrado em Engenharia Elétrica, na área de Engenharia de Computação, da Universidade São Paulo (USP), desenvolvendo sua pesquisa no Laboratório de Tecnologias Adaptativas. Suas principais áreas de pesquisas são: Inteligência Artificial, Processamento de Linguagem Natural e Tecnologia Adaptativa.

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 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.

Estudo preliminar sobre a aplicação da adaptatividade em linguagens de programação imperativas

D. Queiroz and R. L. A. Rocha

***Abstract*— The later decades emerged a subtle interest in the production of self-modifying code in order to achieve a higher level of protection over the source code. Besides hiding the internals of a program, the adaptivity can be seen as a new resource to programmers to develop software that can rearrange its instructions or even perform new tasks. Based on this assumption, several works propose extensions to current programming languages to facilitate the writing of adaptive code. This work represents a preliminary survey of proposals of imperative programming languages suitable for adaptive programming, presenting a comparison between two forms of implementation.**

***Keywords*— Programming languages. Adaptive programming. Imperative programming.**

I. Introdução

Sob uma perspectiva generalista, o atributo "*adaptativo*" é dado a tudo àquilo que possua a capacidade de se adaptar, isto é, de se ajustar para atender novas necessidades ou desempenhar novas funções. Tal adaptação geralmente é realizada com base em regras pré-definidas que atuam como resposta a novas informações obtidas por algum elemento sensorial do objeto adaptativo.

No contexto apresentado por esse trabalho, entende-se por *programação adaptativa* a prática de construir programas com capacidade adaptativa, isto é, programas com mecanismos que alterem seu comportamento inicial ou *programas adaptativos*. Tal alteração de comportamento pode ser realizada através da modificação do (i) fluxo de execução do programa; ou (ii) das instruções definidas em sua listagem original. Essa definição é particularmente relevante devido às divergências pelas quais a literatura categoriza trabalhos que aderem ao modelo de adaptatividade proposto por José Neto [1].

Nessa interpretação, uma linguagem é considerada adequada ao desenvolvimento de programação adaptativa se ela possui recursos que corroboram com a elaboração de mecanismos que alteram o comportamento do programa.

Considerando que a introdução da adaptatividade não aumenta, em sua teoria, a classe de expressividade do programa (dado a sua equivalência com máquinas de Turing [2]), é possível que um programa adaptativo seja transformado tanto em código adaptativo, que efetivamente produz as alterações programadas na linguagem para obter o resultado desejado; como em código não-adaptativo, que sem a necessidade de alterar seu comportamento realiza o mesmo processamento sobre um objeto e, consequentemente, produz o mesmo resultado que o código adaptativo.

Dessa forma, esse trabalho tem como objetivo analisar duas formas de introdução de mecanismos adaptativos em linguagens de programação imperativas, de forma a promover a programação adaptativa.

II. Programação adaptativa

Alguns estudos mostram que há necessidade que as linguagens de programação possuam meios de controlar, não apenas o resultado do processamento de um programa, mas também a sua estrutura e o seu fluxo de execução. Com isso, alguns trabalhos surgiram propondo linguagens de programação de alto nível que tentam satisfazer essa necessidade.

Segundo Anckaert et al. [3], desde a década de 80 a programação adaptativa foi utilizada para ocultar características internas de programas, citando jogos virtuais onde instruções eram produzidas na memória após a inicialização do programa para evitar que engenharia reversa indesejada comprometesse seus mecanismos de proteção contra cópia não autorizada.

Além desse tipo de uso, inúmeras outras tarefas podem ser realizadas através de programação adaptativa: um programa pode, por exemplo, persistir dados entre suas execuções alterando em disco o código de inicialização de suas variáveis internas.

Dada essa difusão, convém avaliar o quão expressivas e poderosas são as linguagens de programação que se apresentam adequadas a essa necessidade. Nas próximas seções apresentaremos algumas propostas de linguagens de programação e a forma como elas se introduzem à programação adaptativa.

III. Programação adaptativa baseada em blocos

Uma das dificuldades que é inerente à criação de programação adaptativa está na necessidade de garantir a coerência daquilo que será efetivamente executado pelo programa (do ponto de vista semântico). Muitas vezes a alteração da sequência de execução de um código não é factível, pois existe dependência da ordem de execução do código.

Para lidar com esse problema, uma proposta de adaptatividade é a de descrever blocos de código que executem operações autocontidas, exercendo a adaptatividade sobre o fluxo de execução desses blocos.

D. Queiroz, Escola Politécnica da Universidade de São Paulo (USP), São Paulo, SP, Brasil, diego.queiroz@usp.br

R. L. A. Rocha, Escola Politécnica da Universidade de São Paulo (USP), São Paulo, SP, Brasil, rlarocha@usp.br

Essa forma de organização dos programas remete a estrutura dos autômatos adaptativos, onde cada bloco de código representa um estado e as condições que determinam o fluxo de execução entre esses blocos são as transições do autômato. Assim como o autômato, ações adaptativas podem estar ou não associadas a essas transições, de modo a permitir a alteração do fluxo original de execução, criação de novos blocos ou a modificação/substituição dos blocos existentes.

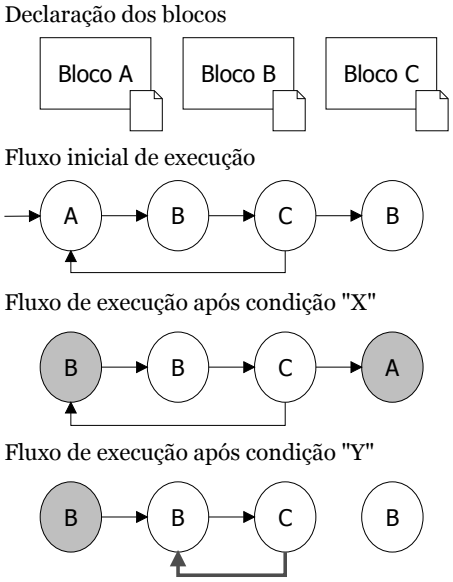


Figura 1. Organização de um programa adaptativo baseado em blocos.

Nessas linguagens, pontos do programa são marcados (através de rótulos ou numeração das instruções) de forma que seja possível indicar a próxima instrução a ser executada, mesmo que essa não esteja no fluxo usual de execução do código.

Tal alteração no fluxo de execução de um programa é facilmente obtida em linguagens de programação cuja construção esteja próxima da linguagem da máquina, como o Assembly, por exemplo, onde o fluxo de execução é determinado por saltos condicionais e o processamento dos dados é realizado por intermédio de "variáveis globais", isto é, os registradores.

No entanto, as linguagens de programação populares, como o C/C++ e Java, restringem essa forma de programação com suas construções estruturadas livres de desvios arbitrários e variáveis dependentes de escopo (locais). Em suma, não é coerente, do ponto de vista dessas linguagens, desviar o fluxo de execução de uma sub-rotina para outra sub-rotina sem questionar os efeitos dessa transição (o que ocorre com os dados?).

Para contornar essa limitação, Sabaliauskas e Rocha [4] propuseram uma extensão da linguagem de programação Oberon que aplica o conceito de programação adaptativa em blocos através do uso de sub-rotinas adaptativas. Na definição apresentada, a sub-rotina adaptativa é formada por blocos de código nomeados, chamados de *fragmentos*. Dessa forma, cada sub-rotina adaptativa possui seu escopo de variáveis, de modo que todos os fragmentos possam compartilhá-lo (as variáveis são declaradas fora dos fragmentos). A Figura 2 apresenta uma implementação da função fatorial definida nessa linguagem.

```
ADAPTIVE fatorial(n: INTEGER): INTEGER;
VAR
  result: INTEGER;

FRAGMENT init;
  result := 1
END init;

FRAGMENT calc;
  result := result * n;
  n := n - 1
END calc;

ACTIONS
  ACTION diferenteZero;
    AFTER calc
      RETURN result CASE n = 0,
      GOTO calc OTHERWISE
  END diferenteZero;

CONNECTIONS
  AFTER init
    RETURN result CASE n = 0,
    PERFORM diferenteZero OTHERWISE

END fatorial;
```

Figura 2. Exemplo de função fatorial desenvolvida na linguagem proposta por Sabaliauskas e Rocha [4].

```
ADAPTIVE MAIN [ NAME=nome, ENTRY=ini, EXIT=fim ]
IS

  CODE init  : <
     ...
  >;
  CODE calc  : <
     ...
  >;
  CODE fim   : < >;

  CONNECTION FROM ini (
    CASE 0 : TO fim
    OTHERWISE TO calc
      PERFORM adapt(ini) BEFORE
  );

  ADAPTIVE FUNCTION adapt(x)[ GENERATORS g ]: {

    REMOVE [ CONNECTION FROM x TO fim ];

    INSERT [ CODE g <
                ...
             > ];

    INSERT [ CONNECTION FROM x (
               CASE 0 : TO calc
               OTHERWISE TO ini
             ) ];
  }

END MAIN
```

Figura 3. Estrutura e sintaxe de um programa escrito na linguagem proposta por Silva e José Neto [5][6].

Após a definição dos blocos, são declaradas regras que podem ser aplicadas durante a execução da sub-rotina com o intuito de redefinir o fluxo de execução dos blocos declarados. Por fim, são estabelecidas as conexões iniciais entre os blocos declarados. As expressões condicionais que definem as regras de transição dessa linguagem (tanto das conexões iniciais quanto das ações adaptativas) também podem fazer uso das variáveis dentro do escopo da sub-rotina ou de seus parâmetros.

Embora não exista uma implementação dessa linguagem para utilização, ela nos permite avaliar uma possível forma de estruturar o código para programação adaptativa.

Silva e José Neto [5][6] propuseram uma abordagem semelhante, com a ideia de uma linguagem de alto nível que pudesse acoplar qualquer outra linguagem imperativa. Nesse trabalho é proposta uma estrutura independente de linguagem que define blocos nomeados de código. Esses blocos carregariam, a princípio, código escrito em uma linguagem qualquer, chamada de *linguagem hospedeira*. Assim, de forma semelhante ao trabalho de Sabaliauskas e Rocha, são definidas as regras de transição entre os blocos (conexões) e as ações adaptativas associadas a essas transições. A Figura 3 apresenta a sintaxe e a estrutura do código dessa linguagem de programação.

O grande diferencial nessa definição é que, além da capacidade de alterar a forma de transição entre os blocos previamente definidos, ela permite a criação de novos blocos de código através de construções dinâmicas associadas às ações adaptativas, chamadas de *geradores*. Assim, é possível conectar um bloco existente a código recém-gerado através da referência ao seu gerador.

IV. Programação adaptativa baseada em recomputação

Outra forma de lidar com as alterações no programa é determinar previamente quais elementos do código estão sujeitos a modificações, de forma que ações sejam tomadas com base nessas modificações.

Para exemplificar essa ideia, Hammer et al. propuseram uma nova linguagem de programação baseada na linguagem C, chamada CEAL [7]. Nessa proposta é introduzido o conceito de *referência modificável*, que nada mais é do que uma posição de memória cujo conteúdo pode ser lido e alterado, semelhante a um ponteiro. No entanto, ao invés de serem diretamente acessadas, a construção da linguagem exige que o conteúdo armazenado em uma referência modificável seja acessado por intermédio das funções read(modref_t) e write(modref_t, void).

Assim, sempre que uma referência modificável possui seu valor atualizado, todos os cálculos e processamentos realizados com base no valor anterior daquela referência são automaticamente recomputados para refletir suas alterações.

Para tanto, as sub-rotinas que fazem uso dessas referências precisam ser declaradas com a palavra reservada "*ceal*" (Figura 4), indicando que seu conteúdo deve ser processado novamente caso alguma das referências modificáveis utilizadas pela rotina tenham sido alteradas desde sua última execução.

A Figura 5 apresenta uma sequência de instruções que exemplifica o funcionamento da linguagem CEAL: nas duas primeiras linhas, a referência modificável "x" é inicializada. Na linha 3, a sub-rotina "adapt" é invocada passando a referência modificável "x" como parâmetro. Na linha 4, a referência "x" é alterada e, por fim, na linha 5, a função propagate() é invocada, causando a recomputação da sub-rotina "adapt".

Como é possível observar, a chamada das sub-rotinas "ceal" não é feito diretamente, mas por intermédio da função run_core. De forma semelhante, a recomputação das sub-rotinas não é realizada automaticamente sempre que as referências modificáveis são alteradas, mas apenas após a chamada da função propagate().

Essa abordagem, apesar de diferir funcionalmente dos autômatos adaptativos ou da ideia de que o código adaptativo deve obrigatoriamente alterar sua estrutura de execução, traz a ideia de que o código precisa se ajustar automaticamente de acordo com a introdução de novas informações, redefinindo as estruturas de dados previamente produzidas.

```
ceal adapt(modref_t *param)  {

  // processa "param"

}
```

Figura 4. Definição de uma sub-rotina na linguagem CEAL.

```
1  modref_t x = modref();
2  write( x, obtemEntrada() );
3  run_core( adapt, x );
4  modify( x, obtemEntrada() );
5  propagate();
```

Figura 5. Exemplo de chamada de sub-rotina e propagação das modificações na linguagem CEAL.

V. Análise dos modelos de programação adaptativa

O modelo de programação adaptativa baseado em blocos satisfaz diretamente a ideia de programas auto modificáveis, por permitir diretamente que a estrutura do código seja remanejada pelo programador. Essa característica torna o modelo ideal quando há a necessidade prévia de determinar como o código produzido deve ser organizado (com o intuito de esconder algum recurso interno do programa, por exemplo).

Por outro lado, a liberdade dada ao programador permite que o programa resultante seja inconsistente (do ponto de vista semântico), devido a alterações não controladas que tenham sido produzidas no código original. Essa característica exige uma maior atenção do programador ou que limitadores de funcionalidade sejam impostos pela linguagem de programação com o propósito de evitar que inconsistências sejam acidentalmente produzidas.

Já o modelo de programação adaptativa baseado em recomputação tem a característica de fornecer um mecanismo de automação ao programa.

Para ilustrar a funcionalidade do recurso, pode-se considerar a atualização dos componentes visuais de uma aplicação gráfica qualquer, cuja invocação é frequentemente necessária para refletir as alterações realizadas pelo programa ao usuário. No modelo de programação não adaptativo, essa chamada geralmente é realizada por meio de eventos ou de consultas periódicas sobre o estado das variáveis.

Contudo, através da modelo baseado em recomputação seria possível, por exemplo, determinar que as rotinas de atualização fossem invocadas automaticamente sempre que os dados relacionados ao componente fossem modificados, sem a necessidade de criação de eventos explícitos para essa finalidade, tornando o modelo de programação reativo e auto ajustável. O

Quadro **1** apresenta uma síntese sobre os modelos de programação adaptativa apresentados nesse trabalho.

Quadro 1. Comparação entre os modelos de programação adaptativa baseado em blocos e baseado em recomputação.

	Blocos	**Recomputação**
Permite a geração de novos programas?	Sim	Não
Permite a alteração do fluxo de execução?	Sim	Não
Permite a persistência dos cálculos realizados?	Sim	Não
Resposta automática a modificações?	Não	Sim
Permite a geração de código inconsistente?	Sim	Não

VI. CONCLUSÃO

Esse trabalho representa um estudo preliminar sobre o projeto de linguagens de programação destinadas à programação adaptativa.

Em resumo, o modelo de programação adaptativa baseado em blocos fornece ao programador uma maior liberdade de estabelecer a forma como o programa deve interagir consigo mesmo para obter o processamento desejado, assim como também fornece uma maior clareza sobre o código que deve ser produzido caso o processo de compilação procure traduzir as alterações no código em alterações reais do programa durante a sua execução.

Por outro lado, o modelo de programação adaptativa baseado em recomputação, apesar de não fornecer recursos para alterar o código ou o fluxo de execução original do programa, fornece uma nova camada de abstração sobre o programa, permitindo que estruturas de dados previamente computadas respondam automaticamente a alterações de seus parâmetros sem a intervenção direta do programa.

Por fim, apesar dessa análise comparativa, é preciso salientar que os modelos de adaptatividade apresentados nesse trabalho não são auto excludentes, isto é, é possível que uma linguagem incorpore os dois modelos de programação simultaneamente, embora os autores não tenham conhecimento de nenhum trabalho já realizado que proponha tal fusão.

Em trabalhos futuros pretende-se adicionar a esse estudo diferentes tipos de abordagem a essa forma de programação, não se limitando apenas ao paradigma imperativo.

AGRADECIMENTOS

O presente trabalho foi realizado com apoio da CAPES, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil.

REFERÊNCIAS

[1] J. José Neto, "Contribuições à metodologia de construção de compiladores," Universidade de São Paulo, São Paulo, 1993.

[2] R. L. de A. da Rocha and J. José Neto, "Autômato Adaptativo, limites e complexidade em comparação com a Máquina de Turing," in Proceedings of the second Congress of Logic Applied to Technology (LAPTEC'2000), 2001, vol. 1, pp. 33–48.

[3] B. Anckaert, M. Madou, and K. De Bosschere, "A model for self-modifying code," Inf. Hiding, vol. 4437, no. 1, pp. 232–248, 2007.

[4] J. A. Sabaliauskas and R. L. de A. da Rocha, "Project and Implementation for a Programming Language Suitable to Express Adaptive Algorithms," IEEE Lat. Am. Trans., vol. 9, no. 6, pp. 969–973, Oct. 2011.

[5] S. R. B. da Silva and J. José Neto, "Proposal of a High-Level Language for Writing Self Modifying Programs," IEEE Lat. Am. Trans., vol. 9, no. 2, pp. 192–198, Apr. 2011.

[6] S. R. B. da Silva, "Software adaptativo: método de projeto, representação gráfica e implementação de linguagem de programação," Universidade de São Paulo, 2011.

[7] M. Hammer, U. A. Acar, and Y. Chen, "CEAL: a C-based language for self-adjusting computation," ACM Sigplan Not., pp. 25–37, 2009.

Diego Queiroz é graduado em Ciência da Computação pela Universidade Nove de Julho e Mestre em Ciências pela Escola Politécnica da Universidade de São Paulo. Atualmente atua no Laboratório de Linguagens e Técnicas Adaptativas como aluno de doutorado.

Ricardo L. A. Rocha é natural do Rio de Janeiro-RJ e nasceu em 29/05/1960. Graduou-se em Engenharia Elétrica modalidade Eletrônica na PUC-RJ, em 1982. É Mestre e Doutor em Engenharia de Computação pela Escola Politécnica da USP (1995 e 2000, respectivamente). Suas áreas de atuação incluem Tecnologias Adaptativas, Fundamentos de Computação e Modelos Computacionais. Dr. Rocha é membro da ACM, IEEE e SBC.

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.